Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Projektarbeit

Data Mesh & ArangoDB

Master of Science
- Master Cyber Security & Privacy -

Fachbereich Informatik
Referent: Prof. Dr. Harm Knolle

eingereicht von:
Frederick Behringer, Simon Hüsch
Matr.-Nr. 90XXXXX 90XXXXX

Sankt Augustin, den 4. Februar 2024

Zusammenfassung

In einer Zeit, in der Unternehmen größer werden und Microservices ein Trend sind, hat Zhamak Dehghani [Deh22] im Jahr 2019 das Konzept Data-Mesh vorgestellt. Bei Data-Mesh ist jede Domäne eines Unternehmens für die eigenen Daten verantwortlich [Deh22]. Damit wird Autonomie gefördert und Entwicklungsprozesse werden vereinfacht. Beachtet werden muss aber auch, dass die größere Verantwortung bei den Abteilungen und ein erhöhter Aufwand für Einheitliche Richtlinien zur Herausforderung werden können [VHM+22].

Eine weitere Herausforderung sind die technischen Aspekte, um große Datenmengen zu verwalten. Diese fallen vor allem in den Bereichen Social-Media und Big-Data an. Um hohe Mengen an Relationen effizient verarbeiten zu können, wie es bei Social-Media-Anwendungen der Fall ist, sind Graphdatenbanken der passende Ansatz [RWE15]. Im Fall von Big-Data treten nicht zwangsläufig viele Relationen auf. Um die immensen Datenmengen von Data-Lakes dennoch bewältigen zu können, sind Document-Stores eine geeignete Lösung [CGK15]. Multi-Model-Datenbanken bieten die Möglichkeit, beide Vorteile zu kombinieren. Ein Vorreiter, der besonders gut als Graphdatenbank agiert, ist ArangoDB.

In dieser Arbeit werden die Konzepte Data-Mesh und ArangoDB einzeln und in Kombination untersucht und auf ihre Fähigkeiten getestet. Dazu werden zunächst in Kapitel 1 die Grundlagen der beiden Themen zusammengefasst. Im Kapitel 2 wird dann ein tabellarischer Überblick des Datenbanksystems ArangoDB gegeben. Danach wird im Kapitel 3 detailliert auf das Anwendungsumfeld, die technologischen Aspekte und eine konkrete Datenbankentwicklung eingegangen. In dem Unterkapitel Datenbankentwicklung wird anhand eines Beispieldatensatzes die Modellierung nach dem Data-Mesh Ansatz dargestellt. Das Kapitel 4 behandelt anschließend eine Leistungsbewertung und ein Data-Mesh-Experiment. Die Leistungsbewertung soll Aufschluss über die Fähigkeiten des Datenbanksystems ArangoDB geben und das Data-Mesh-Experiment Konzept von Data-Mesh genauer untersuchen. Abschließend werden die Ergebnisse in einem Fazit zusammengefasst.

Inhaltsverzeichnis

Zusammenfassung **II**

1 Technologische Grundlagen **1**
1.1 Data-Mesh 1

2 Ausgewählte NoSQL-Systeme at a Glance **4**
2.1 ArangoDB 4

3 Ausgewählte NoSQL-Systeme Unboxed **5**
3.1 ArangoDB 5
3.1.1 Anwendungsumfeld 5
3.1.2 Technologische Aspekte 6
3.1.3 Datenbankentwicklung 10

4 Ausgewählte NoSQL-Systeme in Action **15**
4.1 ArangoDB & Data-Mesh 15
4.1.1 Leistungsbewertung 15
4.1.2 Data-Mesh-Experiment 22
4.1.3 Fazit 27

5 Fazit **29**
5.1 ArangoDB & Data-Mesh 29
5.1.1 Zusammenfassung 29
5.1.2 Data-Mesh 29
5.1.3 ArangoDB 30
5.1.4 „Leasons Learned“ 30

A Anhang **31**
A.1 Abbildungen 31
A.2 Vorverarbeitungsskripte 37
A.3 Data-Contracts 44
A.4 Weitere Skripte 63
A.5 Leistungstestauswertungen 72
A.6 Verfügbarkeiten 75

Abbildungsverzeichnis **78**

Tabellenverzeichnis **79**

Listings **80**

Abkürzungsverzeichnis **80**

Literaturverzeichnis **82**

1 Technologische Grundlagen

1.1 Data-Mesh

Der Grundgedanke hinter dem *Data-Mesh* Konzept ist die Verteilung eines monolithischen Datenökosystems auf mehrere kleinere. Dies lässt sich gut an einem Beispiel verstehen: Das Datenökosystem verhält sich ähnlich wie ein Garten. In einem herkömmlichen monolithischen Ansatz wird ein großer zentraler Garten gepflegt. Mit Data-Mesh ist es, als ob man mehrere spezialisierte Gärten hat, die unterschiedliche Pflanzen und Ökosysteme beherbergen. Jeder Garten ist autonom, hat seine eigene Pflege, Ressourcen und wird von spezialisierten Gärtnern betreut. Der Garten wird also auf mehrere Domänen aufgeteilt. Das *Mesh* (Netz) entsteht dann, wenn man nun nicht direkt die einzelnen Pflanzen aus dem Garten verkaufen möchte, sondern das Ziel ein verfeinertes Produkt wie eine Gewürzmischung ist. Dazu benötigt man nun womöglich Ressourcen aus mehreren Gärten und muss diese zusammenführen. Dies lässt sich direkt auf das Datenökosystem übertragen, wo man nun aus mehreren spezialisierten Quellen aggregieren muss.

Der Data-Mesh Ansatz wurde 2019 von Zhamak Dehghani [Deh22] vorgeschlagen und wird von mehreren Tools wie Databricks [Dat23] oder Data-Mesh Manager [CVH23] vereinfacht. Ein Data-Mesh baut dabei auf vier Prinzipien auf. Diese sollen dafür sorgen, dass das System skalierbar ist und die Daten dennoch von hoher Qualität sind sowie eine ausreichende Integrität aufweisen, damit die Daten anschließend für Analysen und andere Zwecke genutzt werden können. Die folgenden vier Prinzipien wurden von Dehghani vorgestellt:
Das Domänenbesitzprinzip (*Domain-Ownership-Principal*) schreibt den Domänenteams die Verantwortung für ihre Daten zu. Dabei werden die Daten sinnvoll nach Themengebieten aufgeteilt und den Domänenteams zugeordnet. Diese Teams sind dann Experten für ihr Themengebiet, wodurch sich Prozesse vereinfachen und bessere Ergebnisse erzielen lassen. Im Bezug auf das Gartenbeispiel könnte der Garten in mehrere Ökosysteme unterteilt werden. Die Gärtner würden sich dann nur noch um ein spezielles Ökosystem kümmern müssen, wodurch womöglich weniger Pflanzen eingehen, da die Gärtner sich nicht mehr in die Quere kommen und Pflanzen gegebenenfalls zu viel gießen. Ein anderes konkretes Beispiel dafür ist die Firma Netflix, die zwischen Daten in Bezug mit Filmen, Übersetzungen oder Videospielen unterscheidet [Cun20].
Als zweites soll davon ausgegangen werden, dass die Daten außerhalb der Domäne verwendet werden. Demnach sollen die Daten als Produkt (*Data-as-a-Product-*

Principal) angesehen werden. Es werden also hohe qualitative Anforderungen an die Daten und die Analyse Ergebnisse gestellt, wobei auch die Sicherheit der Daten nicht vernachlässigt werden soll. Es wird zum Teil gefordert, dass diese nicht anders als ein Application Programming Interface (API) behandelt werden [CVH22].
Damit die Domänenteams problemlos Datenprodukte erstellen und instand halten können, ist es sinnvoll, dass den Teams Software zur Verfügung steht, mit der sie einfach arbeiten können. Nach dem Selbstbedienungsdateninfrastrukturplattformprinzip (*Self-Service-Data-Platform-Principal*) wird diese Software in Form von Tools und Systemen von einem domänenunabhängigen Datenplattformteam zur Verfügung gestellt. Der Name des Prinzips hebt zudem eine wichtige Eigenschaft für die Datenplattform hervor. Alle Angebote sollen durch Selbstbedienung erreichbar sein. Demnach also keinen bürokratischen Aufwand voraussetzen, sondern auf Nachfrage der Domänenteams direkt erreichbar sein. Ein dafür sehr beliebtes Werkzeug ist beispielsweise Databricks [Dat23].
Das föderierte Führungsprinzip (*Federated-Governance-Principal*) soll zuletzt erreichen, dass alle Datenprodukte als ganzes funktionieren. Dies soll beispielsweise durch Standardisierung erreicht werden. Dabei ist es ein besonders wichtiges Ziel, die Einhaltung von Organisations- und Branchenvorschriften durchzusetzen.

Abbildung 1.1 zeigt ein Architekturdiagramm für den zuvor vorgestellten Ansatz. Im oberen Teil der Abbildung werden die Aufgaben des Führungsteams gezeigt, welche sich ausschließlich mit dem zuletzt vorgestellten föderierten Führungsprinzip befassen. Der untere Block zeigt hingegen die Aufgaben des Datenplattformteams, welches sich um die Selbstbedienungsdatenstrukturplattform kümmert. Im gelben Kreis in der Mitte befinden sich die Aufgabenfelder der Domänenteams. Diese befassen sich ausschließlich mit ihrer Domäne und dem daraus resultierenden Datenprodukt. Die Abbildung zeigt zudem die Schnittstelle, an der sich weitere Domänen anbinden können. Dazu wird hier ein Data-Contract vorgeschlagen, welcher beispielsweise durch eine API implementiert werden kann. In dieser Abbildung wird zudem auf der rechten Seite ein Vorreiterteam vorgeschlagen, welches sich mit neuen Technologien und Beratung befasst.

Der Data-Mesh Ansatz, wie er gerade vorgestellt wurde, bietet einige Vorteile, welche ihn für größere Unternehmen attraktiv machen. In der heutigen Zeit gibt es viele Unternehmen, die weltweit agieren und riesige Datensätze ansammeln. Die Daten werden dazu gerne in *Data-Lakes* gespeichert. Diese Datensätze werden schnell unübersichtlich, da die Daten in einem Data Lake oftmals unstrukturiert im Rohformat gespeichert werden. Um diesen Datensumpf (*Data-Swamp*) zu organisieren, eignet sich der Data-Mesh Ansatz, welcher die Data-Lakes den einzelnen Domänenteams zuordnet und gegebenenfalls große in kleinere unterteilt. Der Ansatz bietet eine geeignete Struktur, um diesen Datenmengen ein Herr zu werden. Netflix hat daher auch für ihr Datenökosystem auf Data-Mesh gesetzt [Cun20]. Der Hauptgrund war dabei jedoch nicht einen großen Datensumpf zu verteilen, sondern viele kleine Systeme zu strukturieren und organisieren. Insbesondere die Selbstbedienungsdateninfrastrukturplattform war in diesem Fall ein Gewinn.

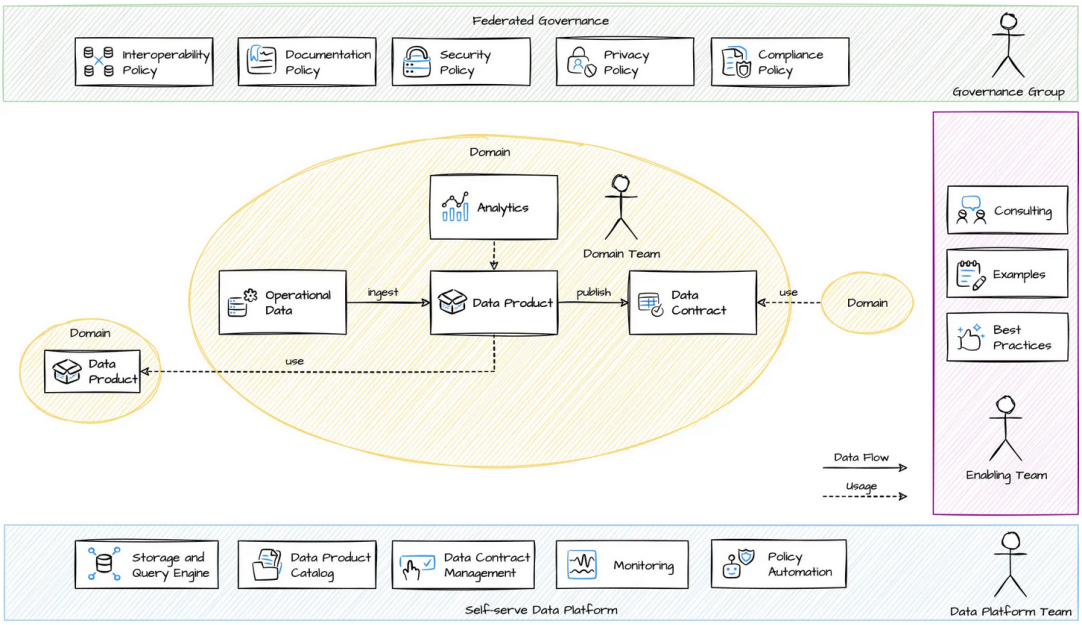


Abbildung 1.1: Architektur eines Data-Mesh [CVH22]

Neben der besseren Struktur erhöht sich durch einen Data-Mesh-Ansatz auch die Produktivität von Datenprodukten und Analyseergebnissen [CVH22]. Die treibende Kraft dahinter ist die Spezialisierung der Domänenteams auf einzelne Bereiche. Daraus folgt, dass es mehr Experten für die einzelnen Bereiche gibt und sich die Entwickler nicht gegenseitig behindern. Eine klare Aufgabenteilung ist also ein Vorteil. Ebenso wird die Produktivität erhöht, da es eine einheitliche Selbstbedienungsdateninfrastrukturplattform gibt, bei der die implementierten Technologien einfach miteinander zusammen arbeiten und klassische Anbindungsfälle bereits behandelt wurden. Durch ein Vorreiterteam können diese beiden Vorteile weiter verstärkt werden [CVH22].

Dennoch ist ein Data-Mesh Ansatz nicht für kleine Unternehmen und Projekte geeignet. Eine Spezialisierung auf die einzelnen Bereiche benötigt eine gewisse Anzahl an Mitarbeitern, um Vorteile zu erzielen. Zudem ist der Aufbau einer umfassenden Selbstbedienungsdateninfrastrukturplattform umfangreich, was vermutlich von kleineren Unternehmen nicht umgesetzt werden kann. Hier könnte jedoch auf bestehende Lösungen zurückgegriffen werden.

2 Ausgewählte NoSQL-Systeme at a Glance

2.1 ArangoDB

Datenbankmodelle	Document-Store, GraphDB, Key-Value-Store, Search-Engine
Daten Schema	Schema frei
Typisierung	verfügbar
Transaktionskonzept	ACID
Konsistenzmodell	Eventual Consistency, Immediate Consistency, OneShard
Anfragemöglichkeiten	Foxx Framework, Graph API (Gremlin), ArangoDB Query Language (AQL), Graph Query Language (GQL), Hypertext Transfer Protocol (HTTP) API, Java & SpringData, Javascript Object Notation (JSON) style queries, VelocyPack/VelocyStream
Programmier Schnittstellen	C#, C++, Clojure, Elixir, Go, Java, JavaScript (Node.js), PHP, Python, R, Rust
Trigger	-
Fremdschlüssel	verfügbar
Indizes	verfügbar
Partitionierungs Methode	Sharding
Replikations Methode	Quell-Replikation mit konfigurierbarem Replikationsmechanismus
Serverseitige Skripte	Javascript
Entwickler	ArangoDB Inc.
Veröffentlicht	2012
Aktuelle Version	`3.11.4`, 25.05.2023
Lizenz	Apache V2 (Open Source), Kommerziel
Server	Docker, Kubernetes, Windows, macOS, Debian, Ubuntu, Redhat Linux
Implementiert in	C++

Tabelle 2.1: Schnellübersicht über das DBS ArangoDB [Ara23g, Ig24]

3 Ausgewählte NoSQL-Systeme Unboxed

3.1 ArangoDB

Das Datenbanksystem (DBS) ArangoDB bietet nativ ein Multi-Model-Datenmodell zur Persistenz und Abfrage von Daten. Durch das Datenmodell, welches sich stark auf Graphen fokussiert, gehört das DBS zur Not only SQL (NoSQL)-Familie. Als Datenbanksprache wird AQL verwendet, über die auf verschiedene Weisen mit den Daten gearbeitet werden kann. In den folgenden Abschnitten wird ArangoDB kurz vorgestellt. Dafür wird zunächst der Einsatzbereich von ArangoDB beschrieben, woraufhin auf einige technologische Aspekte eingegangen wird. Auf der Basis dieses Wissens wurde anhand von Beispiel Daten eine Datenbank (DB) erstellt und erste Abfragen formuliert.

3.1.1 Anwendungsumfeld

Eine Besonderheit von ArangoDB ist die Kombination der drei führenden NoSQL-Datenmodelle innerhalb eines Kerns und einer Programmiersprache [BKJ23]. Damit verfügt ArangoDB über einen Vorteil gegenüber konkurierenden DBSen, da innerhalb einer Abfrage über mehrere Modelle mit den Daten gearbeitet werden kann (siehe Abschnitt 3.1.2). Daraus ergeben sich Vorteile bei der Entwicklung. Zunächst genügt es, dass die Entwickler nur eine Datenbanksprache beherrschen müssen, um die Daten aus mehreren Modellen zu verwenden. Durch die Möglichkeit, in einer Abfrage auf verschiedene Modelle zuzugreifen, erhöht sich auch die Performanz der Abfragen, da nicht erst in einer Anwendung die Daten aggregiert werden müssen. Ein weiterer daraus resultierender Vorteil ist die erhöhte Flexibilität. Gegebenenfalls verändert sich das optimale Datenmodell während der Entwicklung, da neue Daten oder Anforderungen entstehen. In diesem Fall können neue Anfragen formuliert werden, welche die bestehenden nicht behindern [Ara23a]. Dies ist nicht zuletzt auch gerne der Fall beim Einsatz von ArangoDB als Data Lake. Im Falle von vielen unstrukturierten Daten treten während der Analyse oft neue Sichten auf die Daten auf, welche zunächst nicht absehbar waren. Bei einem festen Datenmodell kann dies zu erhöhtem Entwicklungsaufwand führen, da das Datenmodell nicht mehr optimal ist [BKJ23].
ArangoDB wirbt des Weiteren auch mit einer starken Konsistenz. Im Einzelbetrieb

werden Atomicity (Atomizität), Consistency (Konsistenz), Isolation (Isolation) und Durability (Dauerhaftigkeit) (ACID)-Transaktionen und im Cluster-Modus atomare Operationen unterstützt [Ara23a]. Dadurch eignet sich ArangoDB auch für Anwendungen, bei denen die Korrektheit der Daten eine kritische Rolle spielen.
Dennoch empfiehlt sich ArangoDB nicht für alle Anwendungsfälle der einzelnen Datenmodelle, aus denen es zusammengesetzt ist. Insbesondere relevante Einsatzbereiche von Key-Value-Stores werden nicht optimal unterstützt. Der Hersteller von ArangoDB selbst empfiehlt den Einsatz von ArangoDB nicht als *Hyper-Scale-Key-Value-Store* und ebenfalls nicht als Binary large object (Blob)-Store [Ara23g].

3.1.2 Technologische Aspekte

Im folgenden Abschnitt wird auf die technologischen Aspekte von ArangoDB eingegangen. Dabei wird zunächst das Datenmodell beschrieben und die physische Struktur dessen erläutert. Im Anschluss wird auf die Schnittstellen, AQL und andere Zugriffsmöglichkeiten eingegangen. Zuletzt werden Möglichkeiten zur Skalierbarkeit aufgezeigt.

Datenmodelle

ArangoDB ist eine natürliche Multi-Model DB. Dabei vereint sie die Eigenschaften einer GraphDB, eines Document-Stores und eines Key-Value-Stores. Eine Stärke von ArangoDB ist die Option, diese drei Datenmodelle beliebig zu kombinieren. So lassen sich in einer Abfrage die Daten als GraphDB und zu gleich als Dokumente behandeln. Die Strategie, mit der die Daten abgefragt werden, hängt nur von der Abfrage ab. Die persistenten Daten lassen sich also zu jedem Zeitpunkt auf unterschiedliche Weisen abfragen, sodass zeitgleich mehrere unterschiedliche Sichten auf die selben Daten möglich sind [Ara23g]. Dennoch ist anzumerken, dass ArangoDB sich von anderen Graphdatenbanken im Datenmodell unterscheidet. Andere bekannte Graphdatenbanken setzen auf das *Property-Graph-Model*, welches von ArangoDB nicht unterstützt wird. Das gewählte Modell ist ähnlich, unterscheidet sich dennoch signifikant in der Eigenschaft, das Knoten nur in einer Sammlung liegen können. Das Property-Graph-Model schreibt vor, dass Knoten eine beliebige Anzahl an Bezeichnern besitzen, was Vorteile in der Modellierung und Abfrage der Daten bringt [RWE15]. Die unterschiedlichen Sichtmöglichkeiten auf das Multi-Model-Datenmodell von ArangoDB bauen auf einer gemeinsamen Struktur auf. Die einzelnen Datensätze werden im JSON-Format in Sammlungen (Collections) gespeichert. Jeder Datensatz besitzt dabei ein `_key`-Attribut, welches innerhalb der Sammlung eindeutig ist und sich nach einem Einfügen nicht mehr verändern lässt. Diese Information reicht bereits, um ArangoDB als Document-Store zu verwenden. Die Key-Value-Sicht setzt auf dieser Struktur auf. Neben dem `_key` gibt es ein virtuelles Attribut `_id`, welches sich aus dem Sammlungsnamen und dem `_key` zusammensetzt. Es ist somit global eindeutig wie bei einem Key-Value-Store, nur dass

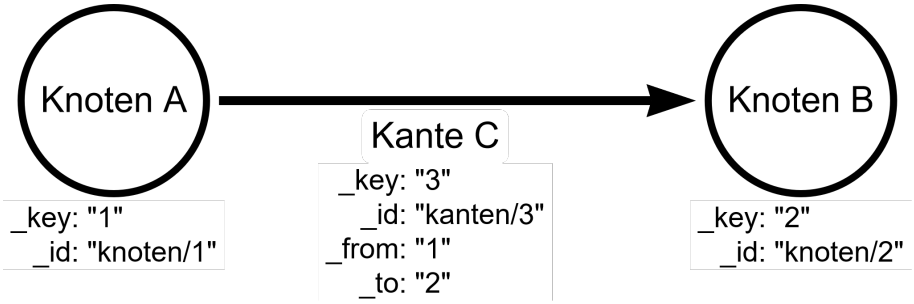


Abbildung 3.1: Graph Struktur innerhalb von ArangoDB

in diesem Fall der Schlüssel (Key) `_id` heißt. Die Graphensichtweise baut ebenfalls auf der Dokumentenstruktur auf. Die Knoten (Vertices) und Kanten (Edges) sind ebenfalls Dokumente, also Datensätze im JSON-Format, welche in Sammlungen gespeichert werden und immer eine `_id` besitzen. Die Dokumente von Kanten besitzen zusätzlich die Attribute `_from` und `_to`. Diese Felder referenzieren jeweils die `_id` eines anderen Dokuments, welches als Knoten interpretiert wird. Somit bilden sie eine gerichtete Kante. Für die Fremdschlüsselattribute bietet ArangoDB spezialisierte Indexe an, um die Performanz zu erhöhen. [Ara23g] In Abbildung 3.1 wird eine einfacherer Graph dargestellt. Der Knoten A ist mit dem Knoten B durch eine gerichtete Kante C verbunden. Dies wird mit den zuvor genannten Attributen beschrieben.

Physische Struktur

Für die Persistenz der Daten setzt ArangoDB die C++ Bibliothek VelocyPack [Ara24] ein. Diese ermöglicht eine kompakte und schnelle Serialisierung von unstrukturierten JSON-Datensätzen. Die Bibliothek ist daher bestens für ArangoDB geeignet. Jedoch schränkt sie die zur Verfügung stehenden Datentypen stark ein. Zur Verfügung stehen lediglich Zahlenwerte, Zeichenketten und Booleans als primäre Datentypen. Daneben ist die Strukturierung mit Objekten und Feldern möglich. In Tabelle 3.1 werden zwei einfache Beispieldatensätze und die dazugehörige Serialisierung mittels VelocyPack dargestellt und erklärt. Aus dem Beispiel wird die Kompaktheit von VelocyPack gut deutlich. Einfach strukturierte Datensätze werden sehr kompakt serialisiert. Laut den Autoren von VelocyPack sind die mit VelocyPack erstellten Dateien bis zu 50 % kleiner als die von der Konkurrenz [Ara24].

Die Bibliothek VelocyPack ist ein Open-Source-Projekt, welches von ArangoDB gestartet und verwaltet wird. Neben den oben genannten eigenschaften spielt zudem die Plattformunabhängigkeit eine zentrale Rolle bei der Entwicklung. VelocyPack ist fast gänzlich plattformunabhängig, wodurch es sich leicht auf unterschiedlichen System einsätzen lässt (siehe Abschnitt 3.1.2).

Tabelle 3.1: Beispiel zur Serialsierung mit VelocyPack

JSON	Hexadezimal	Bedeutung
[1, 2, 3]	02	Array ohne Index mit fester Länge
	05	Länge des Arrays beträgt 6 Byte
	31	Nummer 1
	32	Nummer 2
	33	Nummer 3
{"a": 1, "b": 16}	14	Kompaktes Objekt ohne Index
	0a	Länge in speziellem Format
	41	Unicode Zeichen Präfix
	61	Unicode a
	31	Nummer 1
	41	Unicode Zeichen Präfix
	62	Unicode b
	28	2 Byte Integer ohne Vorzeichen
	10	Nummer 16
	02	Anzahl der Einträge, insgesamt 2

Anfragesprache AQL

Die ArangoDB Query Language (AQL) ist eine proprietäre Sprache von ArangoDB und baut auf der Structured Query Language (SQL) auf. Im Gegensatz zur SQL ist AQL aber eine reine Datenmanipulationssprache. Es ist also ausschließlich möglich, Daten zu lesen, schreiben, aktualisieren oder zu löschen. Datendefinitionsoperationen wie das Erstellen oder Löschen von Sammlungen (Collections) oder Datenbanken, ist nicht möglich. Die beiden größten Unterschiede von AQL zu SQL sind die Schlüsselwörter `FOR` und `RETURN`. Das Schlüsselwort `RETURN` funktioniert im Grunde als Projektionsoperator und ist eine Art Ersatz für das aus SQL bekannte Schlüsselwort `SELECT`. Das Schlüsselwort `FOR` hingegen ermöglicht es, Operationen auf einer List von Dokumenten, z. B. einer Sammlung, auszuführen. Über das Schachteln mehrerer `FOR` Schlüsselworte lassen sich auch kartesische Produkte der Daten bilden und somit Sammlungen vereinen. Die Selektion von Datensätzen erfolgt für gewöhnlich mit dem `FILTER` Schlüsselwort, was äquivalent zu dem `WHERE` Schlüsselwort der SQL ist.
AQL verwendet keine spezifischen Leseoperationen, sondern ermöglicht es in jeder Anfrage bestimmte Werte, u. a. in der Anfrage verwendete Daten aus der Datenbank zurückzugeben. Listing 3.1 zeigt eine AQL-Anfrage, die in die genannten Schlüsselwörter nutzt, um die Namen aller Benutzenden zurückzugeben, die aktive Anfänger sind. Listing 3.2 zeigt die gleiche Anfrage in SQL. (vgl. [Ara23b]) Der Aufbau der AQL-Anfrage ähnelt dabei einer Schleife aus anderen bekannten Programmiersprachen wie Python oder JavaScript.

Listing 3.1: Einfache AQL-Anfrage – aus [Ara23b]

```
1 FOR u IN users
2   FILTER u.type == "newbie" && u.active == true
3   RETURN u.name
```

Listing 3.2: Einfache SQL-Anfrage

```
1 SELECT name
2 FROM users
3 WHERE type = "newbie" AND active = true;
```

Weitere Schnittstellen

Es gibt verschiedene Schnittstellen, über die mit einer DB kommuniziert werden kann. Das Web-Interface bietet eine grafische Oberfläche. Zahlreiche Treiber ermöglichen den Zugang über Programmiersprachen. Mithilfe von Integrationen ist es möglich, ArangoDB mit Drittanbieterwerkzeuge zu nutzen. Über die Kommandozeile kann mit arangosh mit einer Datenbank kommuniziert werden. Und das HTTP-Interface bietet eine generalisierte Möglichkeit zur maschinenlesbaren Kommunikation. Im folgenden werden die genannten Möglichkeiten genauer beschrieben.

Web-Interface Startet man einen Datenbankserver, wie in Sektion 3.1.3 angegeben, ist auf dem Port 8529 ein Web Interface verfügbar, über das alle Datenbanken verwaltet werden können. Hier können unter anderem AQL-Anfragen gestellt werden oder Daten direkt über die grafische Schnittstelle manipuliert werden. [Ara23j]

Treiber Von ArangoDB werden offizielle Treiber für die Programmiersprachen Java, Go, C#, JavaScript und Python angeboten. Mithilfe dieser Treiber können Datenbanken direkt in der jeweiligen Programmiersprache verwaltet und deren Inhalt verändert werden. Die Treiber bieten außerdem die Möglichkeit AQL-Anfragen zu senden. [Ara23h]

Integrationen ArangoDB bietet Integrationen für Spring Data, Apache Spark und Apache Kafka. Mithilfe dieser Integrationen können verschiedene Datenbanken in ein System integriert werden, das die jeweiligen Kommunikationsdetails abstrahiert, sodass auf alle Dankbanken über einem einheitlichen Weg zugegriffen werden kann. [Ara23f] Durch die Integrationen kann ArangoDB in vielen Programmen zur Realisierung eines Data-Meshs genutzt werden [Dat23].

arangosh Das Kommandozeilenwerkzeug `arangosh` bietet eine *JavaScript-Shell*, die sowohl zum Ausführen einzelner Kommandos als auch als interaktive Kommandozeile genutzt werden kann. Über diese Kommandozeile kann die gesamte JavaScript-API zur Verwaltung und Nutzung von Datenbanken genutzt werden. [Ara23c]

HTTP-Interface Hierbei handelt es sich um eine Representational State Transfer (REST)-API, die die Basis für alle vorher genannten Schnittstellen darstellt. Das HTTP-Interface kann zur direkten Datenbankverwaltung, zum Absetzen von AQL-Anfragen oder mit der JavaScript-API verwendet werden. Es wird empfohlen, das HTTP-Interface nur zu nutzen, wenn man Treiber schreiben oder überarbeiten möchte, einen Treiber nutzt, der die benötigte Funktion nicht unterstützt oder sehr große Datenmengen verarbeiten muss und zusätzliche Rechenleistung einsparen möchte. [Ara23e]

Skalierbarkeit

ArangoDB kann sowohl horizontal als auch vertikal skaliert werden. Bei der horizontalen Skalierung werden mehrere Server als Cluster betrieben und können so parallel mehrere Anfragen verarbeiten. Neben der besseren Performanz sorgt dieser Ansatz für automatische Duplizierung der Daten und sichert die Verfügbarkeit beim Ausfall einzelner Server des Clusters. Bei der vertikalen Skalierung werden auf einem einzelnen Server die Ressourcen ausgebaut. Die vertikale Skalierung hat den Nachteil, dass sie nicht die Ausfallsicherheit und die Duplizierung erhöht. Außerdem steigen die Kosten überproportional zum Nutzen. Es ist auch möglich, beide Ansätze zu kombinieren. (vgl. [Ara23i])
In Kapitel 4 werden die Performance Eigenschaften dieser beiden Skalierungsmethoden noch genauer untersucht.

3.1.3 Datenbankentwicklung

Im Rahmen dieser Projektarbeit wurde ArangoDB auf drei Rechnerknoten der Hochschule Bonn-Rhein-Sieg installiert, um anschließend Experimente auf diesen Instanzen durchzuführen. Dieser Abschnitt befasst sich mit der Systeminstallation, der Datenmodellierung nach Data-Mesh-Prinzip, dem Import der Beispieldaten und ersten AdHoc-Anfragen. Auf die Experimente wird im Kapitel 4 eingegangen.

Systeminstallation

Das DBS ArangoDB ist für alle gängigen Betriebssysteme verfügbar. Für die Linux-Distributionen Debian, Ubuntu und RedHat, sowie für Windows gibt es Package-

Installationen. Unter Linux werden diese auch von den Package-Managern apt und yum unterstützt. Für die oben genannten Systeme und macOS stehen außerdem Archive zum Herunterladen bereit, die ausführbare Dateien enthalten. Für Systeme mit Linux und macOS stehen alle Versionen sowohl für ARM als auch für x86-64 bereit. Darüber hinaus stehen für die oben genannten Systeme auch Docker- und Kubernetes-Images zur Verfügung. [Ara23d]

In den Archiven liegt eine README-Datei bei, in der erklärt wird, wie ein Datenbankserver gestartet werden kann. Um einen Server im Standalone-Modus zu starten, kann `arangodb` von der Kommandozeile aus mit der Option `--starter.mode single` aufgerufen werden. Um ArangoDB im Multi-Node-Modus zu starten, wird die Option `--starter.mode cluster` verwendet. Weitere Knoten können dann dazu geschaltet werden, indem auf einer weiteren Maschine `arangodb` mit der Option `--starter.join <IP-Adresse eines bereits teilnehmenden Knotens>` ausgeführt wird. Einem Cluster können so beliebig viele Knoten hinzugefügt werden.

Für diese Arbeit wurde ein Cluster bestehend aus drei Knoten und einem Hauptserver zur Verfügung gestellt. Alle Knoten nutzen Ubuntu 22.04.1 LTS Server 64-Bit als Betriebssystem. Für die Installation wurde sich für den Weg der manuellen Archiv-Installation entschieden. Um die Knoten zu starten, wurde ein Bash-Skript entwickelt, welches sich in Listing A.9 finden lässt.

Datenmodellierung nach Data-Mesh-Prinzip

Das Data-Mesh-Konzept schreibt bei der Datenmodellierung eine Trennung nach einzelnen Domänen vor. Wie die Daten innerhalb der Domäne physisch gespeichert werden, ist den Domänenteams überlassen. Lediglich die Aufteilung der Domänen findet in globaler Absprache statt. Dadurch wird den Domänenteams eine hohe lokale Autonomie gewährt. Die gegebenen Beispieldaten wurden in diesem Fall ebenfalls in zwei Domänen unterteilt. Es wurde dabei in die Domänen Bestellungen (Orders) und Lieferanten (Supplier) unterteilt. Die Unterteilung wird in Abbildung 3.2 dargestellt. Wichtig anzumerken ist hier, dass dies nur getan wurde, um das Data-Mesh Prinzip im Rahmen dieser Arbeit zu erforschen. In einem echten Produktivsystem wäre eine Unterteilung aufgrund der nicht komplexen Daten wenig sinnvoll. Abbildung A.1 zeigt ein logisches Schema der importierten Daten.

Import der Beispieldaten

Es gibt mehrere Wege, Daten in eine Datenbank zu importieren. Für diese Arbeit wurde der Weg über das Web-Interface genutzt. Dazu wurden zuerst über die „_systems“-Datenbank im „Databases“-Reiter die notwendigen Datenbanken erstellt. In den jeweiligen Datenbanken wurden dann im zweiten Schritt über den „Collections“-Reiter die benötigten Sammlungen angelegt. In die Sammlungen kann dann im „Content“-Reiter eine JSON-Datei mit zu importierenden Daten hochgela-

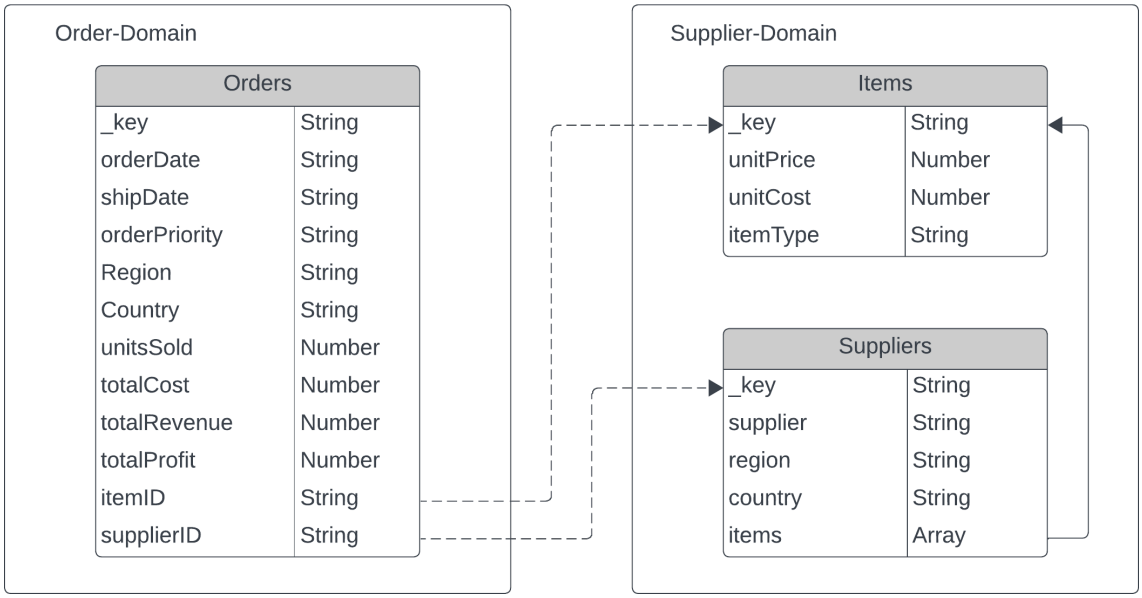


Abbildung 3.2: Physisches Schema der Beispieldaten

den werden. Um die Comma Seperated Values (CSV)-Dateien in ein JSON-Format zu bringen, wurde eine Reihe von Python-Skripten verwendet, welche im Anhang unter Kapitel A.2 zu finden sind.
Von der hochgeladenen CSV-Datei ausgehend lässt sich dann auf unterschiedliche Weisen weiter arbeiten. Ein Ansatz ist es, alle Daten im Rohformat hochzuladen und anschließend den Schemata entsprechend aufzuteilen und die Typen anzupassen. Dies ließ sich mittels AQL gut umsetzen. Die Beispieldaten wurden dabei mit dem Befehl aus Listing 3.3 verarbeitet. Ein anderer Ansatz ist, die Rohdaten vorher zu präparieren und erst dann als bereits optimierten Datensatz zu importieren. Im Listing A.5 ist aufgezeigt, wie die Daten für die Supplier-Domäne aufbereitet werden, sodass sie als JSON-Datei importiert werden können.

AdHoc-Anfragen

Die Listings 3.4 bis 3.7 beinhalten vier beispielhafte AQL-Anfragen, die die Create, Read, Update, Delete (CRUD)-Operationen auf der Items-Sammlung ausführen:

Listing 3.4: Beispiel für eine Create AQL-Anfrage auf der Items-Sammlung, mit der ein neues Item der Sammlung hinzugefügt wird. Als Antwort erhält man ein leeres Array.

```
1 INSERT {
2     _key: "MYEMEVEG0051ES",
3     unitCost: 123.45,
4     unitPrice: 607.89,
5     itemType: "Vegetables"
6 } INTO Items
```

Listing 3.3: Verarbeitung der Beispieldaten mittels AQL

```
FOR d IN allMillionData
    INSERT {
        _key: TO_STRING(d.`Order␣ID`),
        itemID: d.ItemID,
        supplierID: SUBSTRING(SHA256(CONCAT_SEPARATOR('_', d
            .Supplier, d.SuplierRegion, d.SuplierCountry)),
            0, 16),
        country: d.Country,
        region: d.Region,
        shipDate: d.`Ship␣Date`,
        orderDate: d.`Order␣Date`,
        orderPriority: d.`Order␣Priority`,
        salesChannel: d.`Sales␣Channel`,
        unitsSold: TO_NUMBER(d.`Units␣Sold`),
        totalRevenue: TO_NUMBER(SUBSTITUTE(d.`Total␣Revenue`
            , ',', '.')),
        totalCost: TO_NUMBER(SUBSTITUTE(d.`Total␣Cost`, ',',
            '.')),
        totalProfit: TO_NUMBER(SUBSTITUTE(d.`Total␣Profit`,
            ',', '.'))
    } INTO orders
```

Listing 3.5: Beispiel für eine Read AQL-Anfrage auf der Items-Sammlung, die nach Items vom Typen „Vegetables“ filtert, nach Kosten und Preis sortiert und die Felder mit Kosten, Preis und ID als Array zurückgibt. Auch das Item, das oben hinzugefügt wurde, ist dabei.

```
FOR i IN Items
    FILTER i.itemType == "Vegetables"
    SORT i.unitCost DESC, i.unitPrice
    RETURN KEEP(i, unitCost, unitPrice, _key)
```

Listing 3.6: Beispiel für eine Update AQL-Anfrage auf der Items-Sammlung, die das oben hinzugefügt Item dahin gehende verändert, dass der Preis auf das Doppelte der Kosten gesetzt wird. Die Anfrage gibt das veränderte Dokument zurück.

```
LET doc = FIRST(
    FOR i IN Items
        FILTER i._key == "MYEMEVEG0051ES"
        RETURN i
    )
UPDATE doc WITH {
    unitPrice: doc.unitCost * 2
} IN Items
RETURN NEW
```

Listing 3.7: Beispiel für eine Delete AQL-Anfrage auf der Items-Sammlung, die das oben eingefügte Item wieder löscht. Das gelöschte Objekt wird von in der Antwort zurückgegeben.

```
1 FOR i IN Items
2     FILTER i._key == "MYEMEVEG0051ES"
3     REMOVE i IN Items
4     RETURN OLD
```

4 Ausgewählte NoSQL-Systeme in Action

4.1 ArangoDB & Data-Mesh

In diesem Kapitel werden zwei Experimente durchgeführt. Zuerst wird eine Leistungsbewertung von ArangoDB durchgeführt, bei der Zeitmessungen auf den CRUD-Operationen durchgeführt werden. Anschließend wird in der Sektion 4.1.2 eine exemplarische Umsetzung des Data-Mesh-Konzepts dargestellt. Dazu werden mit verschiedenen Techniken Schnittstellen zur Order-Domäne und zur Supplier-Domäne entwickelt. Beide Schnittstellen werden dann in der Sektion "Skizzierung der Anwendung"verwendet, um eine höhere Analyseanwendung zu erstellen.
Alle erstellten Quellcodes und erzeugten Ergebnisse wurden auf GitHub[1] zur Verfügung gestellt.

4.1.1 Leistungsbewertung

In diesem Abschnitt geht es um einen Leistungstest von ArangoDB, bei dem das Verhalten unter zunehmender Hintergrundlast getestet wird. Ebenfalls erprobt werden die Effekte von den von ArangoDB angebotenen Skalierungsmethoden. Dazu wird zunächst der Versuchsaufbau dargelegt. Anschließend werden die Ergebnisse vorgestellt und eingeordnet.

Versuchsaufbau

Für die Leistungsbewertung von ArangoDB wurden alle CRUD-Operationen für Datensätze der Größen 1, 1 000 und 1 000 000 gemessen. Bei allen Operationen wurden Massenoperationen verwendet; die Daten wurden also nicht einzeln, sondern pro Operation zusammen gesendet. Die Daten wurden in einer eigenen Sammlung ohne weitere Strukturierung gespeichert. In die Sammlung wurden zu Beginn zusätzlich die fünffache Anzahl an Datensätzen geschrieben und alle Operationen griffen auf zufällige Objekte aus allen verfügbaren Datensätzen[2] zu, sodass vermieden wurde,

[1] https://github.com/some-random-int/schemaless-databases
[2] Eine Millionen verfügbare Datensätze, ein Datensatz hat 18 Attribute und ist ca. 500 Byte groß, keine Schachtelung, Beispiel in A.12

dass alle Daten physisch nebeneinander lagen. Ziel der Leistungsbewertung war es zu testen, wie sich ArangoDB unter zunehmender Last verhält. Dazu wurden alle Tests einmal ohne Hintergrundlast und je einmal mit 20, 50 und 100 Threads, die für Hintergrundlast sorgen, durchgeführt. Die Threads führen jeweils randomisierte CRUD-Operationen mit 50 bis 200 Datensätzen auf einer separaten Sammlung durch. Im Standalone-Modus sorgen 20 Threads auf den zur Verfügung gestellten Servern[3] für etwa 50 % Auslastung aller CPU-Kerne. Bei 50 Threads ist sichergestellt, dass alle CPU-Kerne dauerhaft ausgelastet sind. 100 Threads sorgen im Standalone-Modus dafür, dass der Rechner dauerhaft überlastet ist. Um zusätzlich die horizontale Skalierungsmöglichkeit von ArangoDB und deren Nutzen zu testen, wurden alle Tests jeweils einmal im Standalone-Modus und im Cluster-Modus mit zwei und drei Replikationen, sowie mit zwei und drei Shards, durchgeführt. Bei eingeschalteter Replikation werden alle Daten auf mehreren Knoten geschrieben. Beim Sharding werden die Daten anhand ihrer IDs auf einem der Knoten persistiert, sodass jeder Knoten nur einen Bruchteil der Daten speichern muss. Um grobe Abweichungen durch äußere Umstände zu vermeiden, wurden jeweils zehn Tests durchgeführt. Die Sammlung für die Hintergrundbelastung wurde jeweils nach einer Testreihe von den genannten zehn Tests zurückgesetzt. Die Sammlung für den eigentlichen Test wurde jeweils nach jedem CRUD-Durchlauf, also zehnmal häufiger wie die Sammlung für die Hintergrundbelastung, zurückgesetzt. Abgesehen von den genannten Einstellungen wurden alle Standardeinstellungen der Datenbank und der Sammlungen beibehalten. Alle benötigten Datensätze wurden auf Basis der zur Verfügung gestellten Beispieldaten generiert. Dazu wurden zuerst die Datensätze selbst verwendet und alle weiteren Datensätze wurden aus diesen mithilfe eines Skripts A.10 erzeugt. Damit ergibt sich eine Gesamtzahl von 2 400 Testläufen (4 *Operationsarten* × 3 *grössen von Datensätzen* × 4 *grössen von Hintergrundllast* × 5 *Replikationszahlen* × 10 *Testläufe je Versuchsaufbau*). Um ausschließlich ArangoDB zu bewerten, nutzt das Testskript direkt das HTTP-Interface und wurde auf dem Master-Node des Test-Clusters ausgeführt. Das Testskript A.11 ist im Anhang zu finden.

Auswertung

Tabelle A.1 zeigt die Mediane der Tests. Alle Ergebnisse und ergänzende Grafiken sind auf GitHub zu finden. Um zu verstehen, wie sich ArangoDB unter zunehmender Last verhält, wird der Fokus zuerst auf die Parameter gelegt, die zu unterschiedlichen Leistungen geführt haben. Abbildung 4.1 zeigt alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe eins. Auf der x-Achse sind die Kombinationen von Skalierungsart und Hintergrundbelastung aufgetragen. Dabei steht „R“ für Replikationen, „S“ für Shards und „H“ für Hintergrundthreads. Beispielsweise „2S 50H“ steht also für zwei Shards mit 50 Hintergrundthreads. Auf der y-Achse sind die gemessenen Laufzeiten aufgetragen und auf der z-Achse sind die zehn Test-

[3] Je Server: 8 CPU-Kerne, bis 4,4 GHz, mehr in A.13; 4 × 8 GB DDR3-RAM mit 1 333 MT/s und 40 GB Swap; 256 GB SSD, ca. 9,4 GB/s Lese- und ca. 310 MB/s Schreibegeschwindigkeit. Die Server sind über Kabel mit Transferraten von 1 GBit/s verbunden.

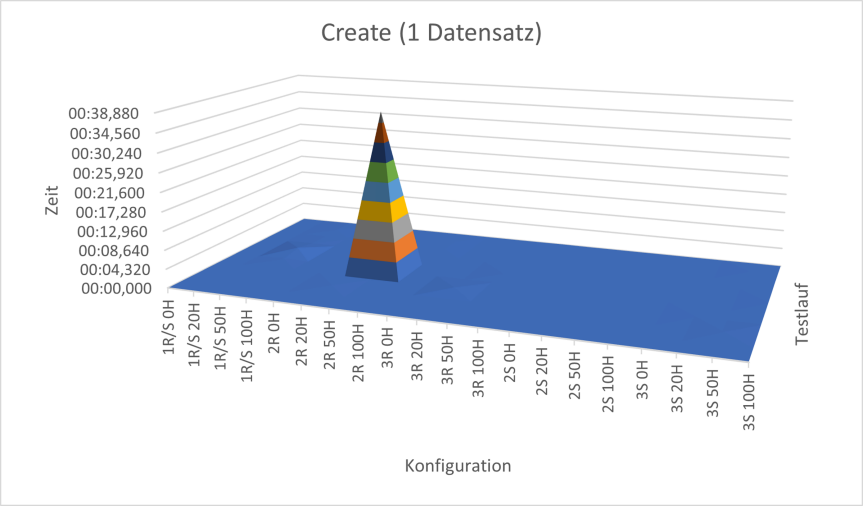


Abbildung 4.1: Alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe eins

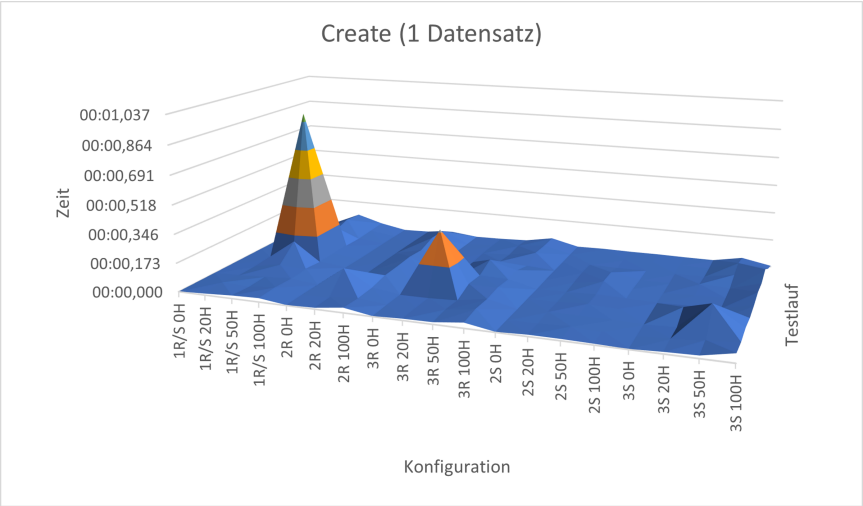


Abbildung 4.2: Alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe eins ohne größten Ausreißer

läufe dargestellt. In diesem Diagramm ist deutlich erkennbar, dass es bei kleinen Datensätzen zu großen Abweichungen zwischen einzelnen Testläufen kommen kann. Auch wenn die Konfiguration mit der sehr großen Abweichung nicht beachtet wird, wie es in Abbildung 4.2 zu sehen ist, sind weitere starke Abweichungen zwischen den Testläufen zu erkennen. In Abbildung 4.3 ist das entsprechende Diagramm für Datensätze mit 1 000 Dokumenten zu sehen. Hier zeigt sich bereits etwas mehr Stabilität zwischen den einzelnen Testläufen. Dies drückt sich in Wellenbewegungen auf der x-Achse aus. Das Diagramm für Datensätze mit einer Millionen Dokumenten ist in Abbildung 4.4 zu sehen. Hier formen sich bereits eindeutige Wellenbewegungen – die relativen Unterschiede zwischen den Testläufen sind also kleiner. Es lässt sich festhalten, dass es Abweichungen zwischen den Testläufen gibt, die mit zunehmender Datensatzgöße zueinander kleiner werden.

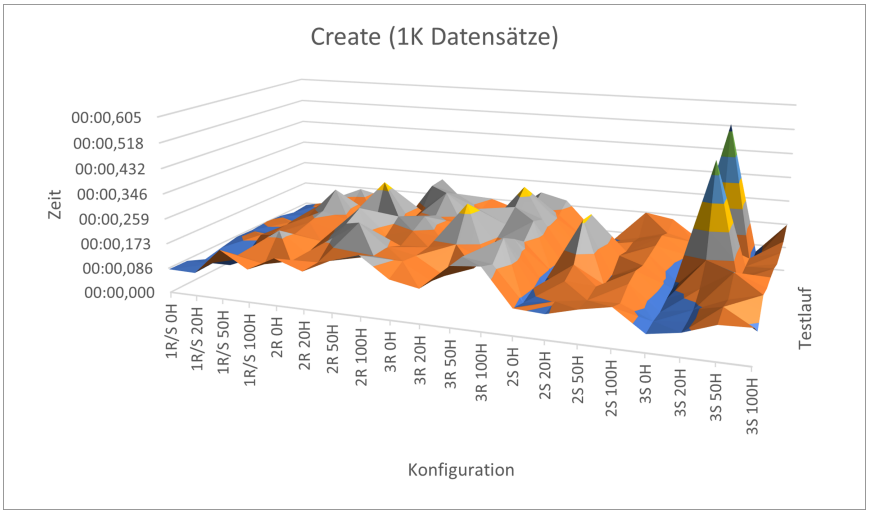


Abbildung 4.3: Alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe Eintausend

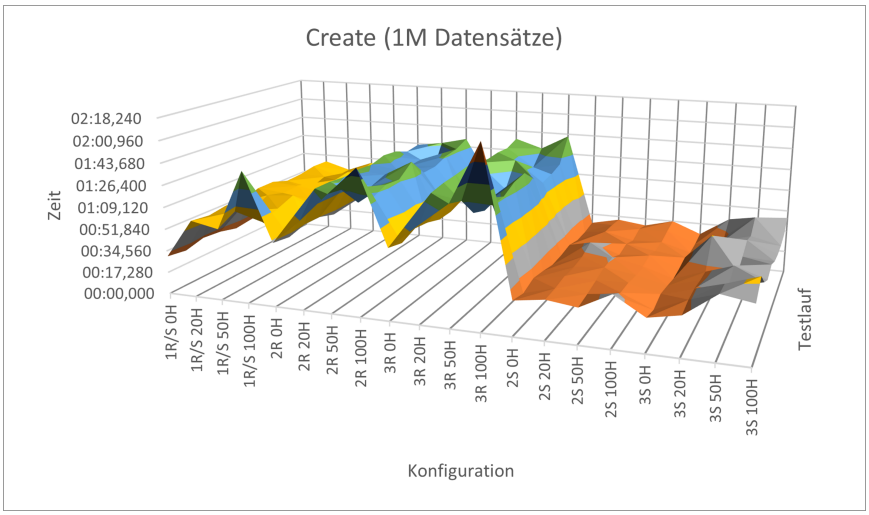


Abbildung 4.4: Alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe einer Millionen

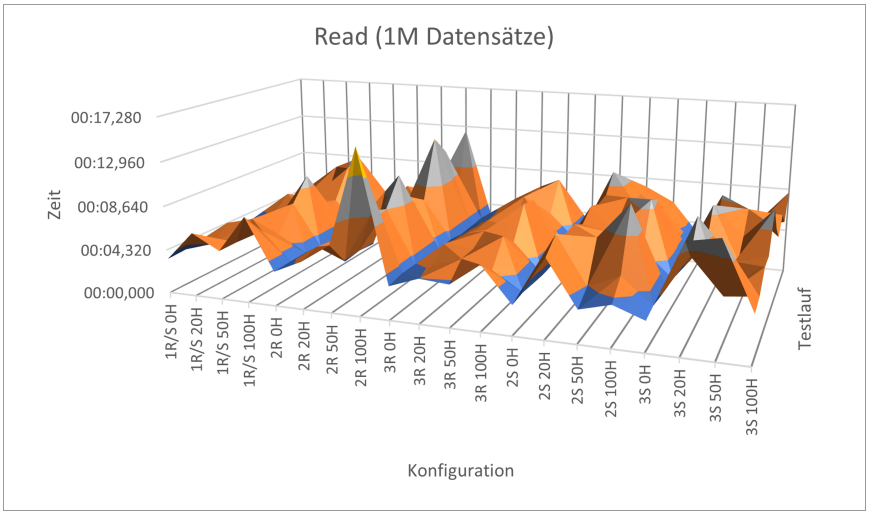
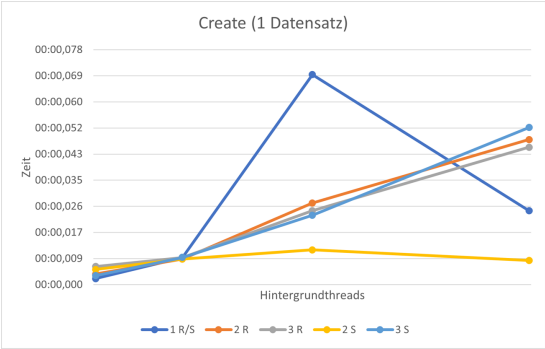
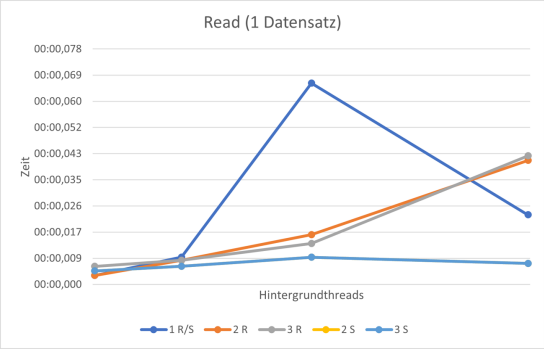


Abbildung 4.5: Alle Laufzeiten für die Read-Operation mit einem Datensatz der Größe einer Millionen

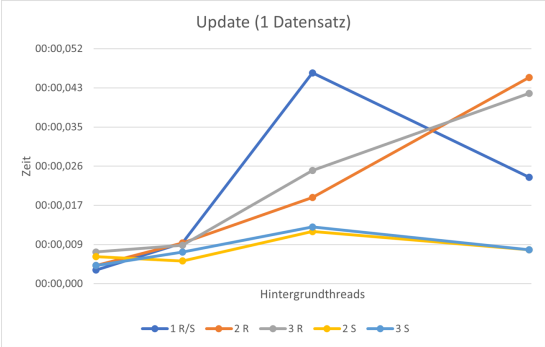
In Abbildung 4.6 sind die Medianlaufzeiten der CRUD-Operationen bei steigender Last zu sehen. Auf der x-Achse ist die Anzahl der Hintergrundthreads dargestellt. Die vier Punkte in den Graphen zeigen 0, 20, 50 und 100 Hintergrundthreads an. Auf der y-Achse ist die Medianlaufzeit der Tests zu sehen. Die Medianlaufzeiten gleichen größere Abweichungen zwischen den Testläufen aus. Die fünf Linien zeigen die äquivalent zur oben genannten Darstellung Messungen bei den unterschiedlichen Skalierungsarten. In den Diagrammen ist zu erkennen, dass die Abweichungen zwischen den verschiedenen CRUD-Operationen klein sind. Die Art der Operation hat also keine deutlich Auswirkung auf die Leistungsfähigkeit von ArangoDB.
Betrachtet man in Abbildung 4.7 die Abweichungen zwischen den Datensatzgrößen, fällt zuerst wieder auf, dass bei zunehmender Datensatzgröße die Verläufe stabiler werden. Weiter ist auffällig, dass bei unterschiedlichen Datensatzgrößen es an verschiedenen Stellen Ausreißer gibt, wie etwa die auffällig lange Medianzeit bei einer Datensatzgröße von 1 000 und zwei Shards mit 100 Hintergrundthreads oder Hochpunkte bei 20 Hintergrundthreads und eienr Datensatzgröße von einer Millionen bei allen Skalierungsarten. Zusammenhänge, die diese Ausreißer begründen, sind nicht erkennbar, obwohl sie bei allen Operationen zu sehen sind.
Um den Fokus nun auf das Verhalten von ArangoDB unter zunehmender Last zu setzen, werden Lese- und Schreiboperationen separat betrachtet. Dabei werden nach den bisherigen Erkenntnissen die Medianlaufzeiten bei Datensatzgrößen von einer Millionen Dokumenten betrachtet, um möglichst reproduzierbare Zahlen zu verwenden. Wie in Abbildung 4.8a zu sehen, ist beim Lesen bei allen Skalierungsarten eine leichte Erhöhung der Laufzeiten bei einsetzender Hintergrundlast zu erkennen. Bei weiter steigender Last bleiben die Laufzeiten aber stabil. Die Unterschiede zwischen den Hintergrundlasten und Skalierungsarten sind zu vernachlässigen, weil beim Lesen insgesamt relativ große Schwankungen zwischen den Testläufen zu beobachten sind, die bei allen Konfigurationen ähnlich groß sind (siehe Abbildung 4.8a). Beim Schreiben ist, wie in Abbildung 4.8b zu sehen, ebenfalls bei allen Skalierungsarten eine leichte Laufzeiterhöhung bei einsetzender Hintergrundlast zu beobachten.



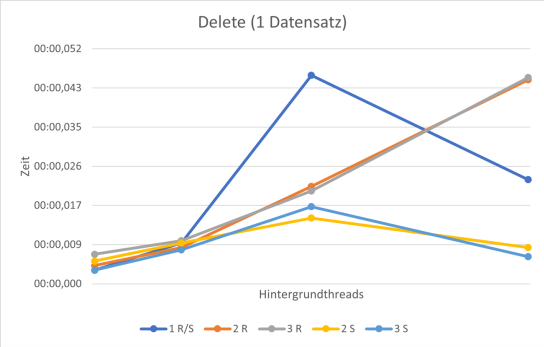
(a) Create



(b) Read



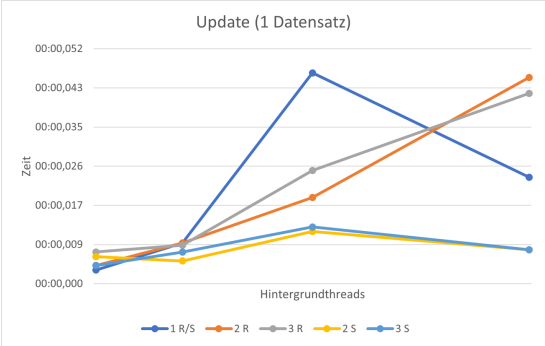
(c) Update



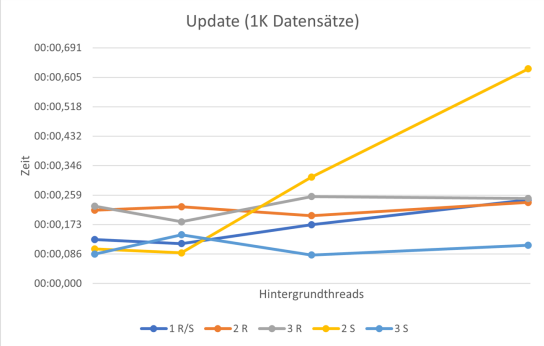
(d) Delete

Abbildung 4.6: Medianlaufzeiten der CRUD-Operationen für Datensätze der Größe eins

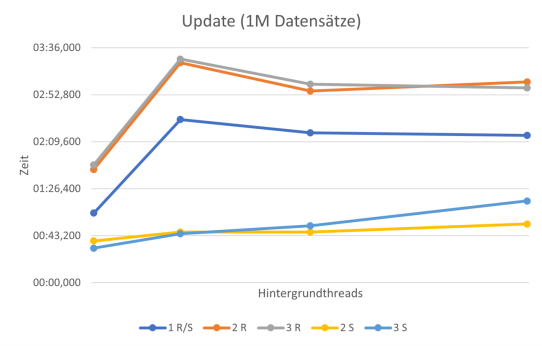
Es ist jedoch auffällig, dass bei Replikationen die Laufzeiten insgesamt höher und beim Sharding insgesamt niederiger sind als im Standalone-Modus. Das lässt sich dadurch erklären, dass bei Replikationen der Schreibprozess auf mehreren Servern durchgeführt werden muss. Währenddessen teilen sich beim Sharding die Server die Aufgaben auf, sodass jeder Server weniger schreiben muss. Im Vergleich zum Diagramm mit den Leseoperationen sind die Linienverläufe bei den Schreiboperationen stabiler. Vergleicht man Abbildung 4.4 mit Abbildung 4.5, sieht man auch, dass beim Schreiben auch weniger Abweichungen zwischen den Testläufen sind, was die stabileren Linien hier erklärt. Auch lässt sich in Abbildung 4.4 sehr gut die Kante zwischen drei Replikationen und zwei Shards erkennen. Bei allen Operationen ist auffällig, dass unter zunehmender Last das System etwas schneller wird. Auch dass zwei Shards schneller sind als drei deutet daraufhin, dass hohe Lasten das System effizienter machen, da drei Shards die Hintergrundlast so aufteilen, dass nicht alle Server voll ausgelastet sind.



(a) Größe eins

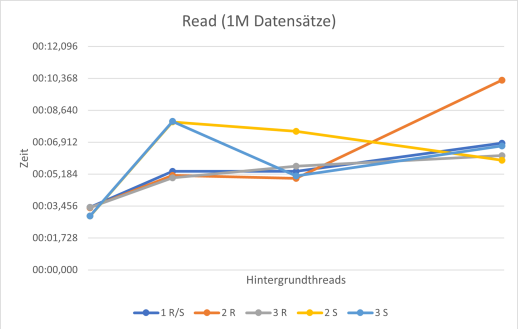


(b) Größe Eintausend

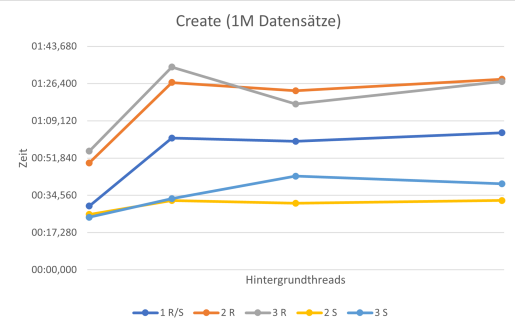


(c) Größe eine Millionen

Abbildung 4.7: Medianlaufzeiten von Create-Operationen für verschiedene Datensatzgrößen



(a) Read-Operationen



(b) Create-Operationen

Abbildung 4.8: Medianlaufzeiten aller Skalierungsarten mit Datensätzen der Größe eine Millionen

4.1.2 Data-Mesh-Experiment

Neben der Leistungsbewertung von ArangoDB wurde mit ArangoDB auch das Data-Mesh-Konzept erprobt. Dazu wurden die Beispieldaten zunächst in zwei Domänen zerteilt, wie es vom Data-Mesh-Konzept gefordert ist. Die Datenmodellierung dessen wurde bereits in Abschnitt 3.1.3 beschrieben. Dieser Abschnitt befasst sich nun mit der weiteren Umsetzung des Data-Mesh-Konzepts. Als Ziel soll eine kleine Webanwendung entstehen, welche einfache Analysen aus der Sicht der Anbieter (Supplier) ermöglicht. Diese wird zunächst skizziert und in den darauf folgenden Abschnitten realisiert. Bei der Realisierung wird zunächst auf die Implementierung in den beiden einzelnen Domänen eingegangen. Daraufhin werden die beiden Domänen verbunden. Zu guter Letzt wird ein einfaches Frontend entwickelt, in dem die Daten zugänglich dargestellt werden. Die Quellcodes der Domänen und des Frotends sind unter https://github.com/some-random-int/schemaless-databases zu finden.

Skizzierung der Anwendung

Aus Zeitgründen wird die Anwendung sehr einfach gehalten und auf einige Aspekte, wie Sicherheitsmaßnahmen, Nutzerfreundlichkeit und Barrierefreiheit nicht umfangreich eingegangen. Die Anwendung soll ein Werkzeug sein, das den Anbietern eine Möglichkeit bietet, die Bestellungen ihrer Produkte einzusehen und einige Statistiken abzurufen. Um die Anwendung zu charakterisieren, wurden zunächst fünf Userstories (USs) erstellt:

US-1 Als Anbieter (Supplier) möchte ich mich mit meiner SupplierID anmelden können, um dann alle über mich gespeicherten Daten (Land, Region, angebotene Artikel) ansehen und bearbeiten zu können.

US-2 Wenn ich als Supplier angemeldet bin, möchte ich bei mir eingegange Bestellungen einsehen können, um immer einen Überblick zu haben.

US-3 Wenn ich als Supplier angemeldet bin, möchte ich die Möglichkeit haben neue Artikel anzubieten und bestehende zu löschen.

US-4 Wenn ich als Supplier angemeldet bin, möchte ich folgende Statistiken abrufen können: Boxplot, Säulendiagramme und Zeitverläufe für einzelne Zielländer/-regionen und / oder Items mit ...

- Lieferzeit
- Umsatz
- Gewinn
- Verkaufskanal
- Einheiten pro Order

US-5 Wenn ich als Supplier angemeldet bin, möchte ich mich wieder abmelden können, um zum Anmeldebildschirm zurückzukommen.

Auf der Grundlage dieser USs wurde die Oberfläche der Anwendung unter der verwendung von Balsamiq [BS24] skizziert. Die Abbildungen A.2 und A.3 zeigen dies.

Neben einer simplen Login-Seite besteht die Anwendung aus einer Seite, welche die Daten der Anbieter zusammenfasst. Auf dieser Seite kann ein Anbieter auf der linken Seite seine Daten und Produkte einsehen (**US-1**). Über eine Tabelle, die alle Artikel auflistet, soll es möglich sein die Artikel anzupassen oder neue zu erstellen (**US-3**). Daneben ist es möglich, im rechten Teil der Anwendung diverse Diagramme zu konfigurieren und so die Bestellungen auszuwerten (**US-4**). In der Menüleiste im oberen Teil befindet sich ein Knopf, damit sich der Anbieter wieder abmelden kann (**US-5**).

Schnittstellen Auswahl

Die Programmiersprachen Javascript und Python sind 2023 die meist genutzten Programmiersprachen gewesen [Ove23]. Für diese Arbeit werden daher die offiziellen ArangoDB-Treiber für Node.js und Python genutzt. Für die Schnittstellen der Order-Domäne wird Node.js genutzt, für die der Supplier-Domäne Python. In Abbildung 4.9 wird die Struktur der Anwendung dargestellt. Die beiden Domänen greifen jeweils auf eine eigene DBen zu, was hier in unterschiedlichen Programmiersprachen umgesetzt wurde. Die folgenden Abschnitte gehen nun näher auf die Implementierung ein.

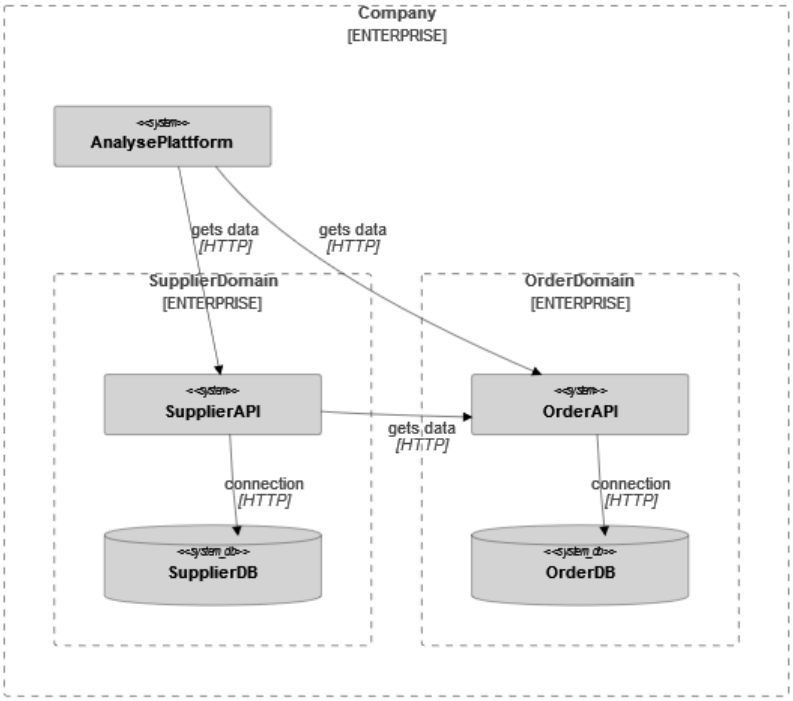


Abbildung 4.9: Komponentendiagramm der Anwendung

Order-Domäne

Die Order-Domäne verwaltet die Bestellungen innerhalb einer ArangoDB-Instanz. Diese beinhaltet die operationalen Daten, welche von dieser Domäne nun untersucht werden und anschließend als Datenprodukt den anderen Domänen zur Verfügung gestellt werden. Dafür wird ein Datenvertrag (Data-Contract) angeboten, welcher hier in Form einer API-Spezifikation nach dem OpenAPI Standard umgesetzt wurde. Dieser befindet sich in Listing A.7. Aus der API-Spezifikation wird von dem implementierten Backend-Server eine Dokumentation generiert, die anschließend aus dem Web heraus erreichbar ist. Sie ermöglicht auch die einfache Anbindung an die Endpunkte, indem sich aus dieser der Quellcode für diverse Clients generieren lässt. Doch zunächst wird auf die Analysen innerhalb der Domäne eingegangen. Die Order-Domäne selbst enthält zunächst keine Beziehungen der Daten. Jedoch lassen sich die Bestellungen nach mehreren Attributen leicht aggregieren. Dabei sind interessant das Datum der Bestellung, die Priorität, die Region und das Land sowie der verkaufte Artikel und der Anbieter. Nach diesen Punkten lassen sich die meisten restlichen Attribute aggregieren wie die Verkäufe, Gewinne oder durchschnittliche Versandzeit. Die Anwendung soll auf diese Daten Zugriff haben, weshalb eine dynamische Implementierung, bei der sich unterschiedliche Werte kombinieren lassen, sinnvoll ist. Neben den Möglichkeiten, auf die zuvor beschriebenen Analysen einzugehen, werden nach außen hin Endpunkte angeboten, um neue Bestellungen zu erstellen und zu bearbeiten sowie einzelne Bestellungen zu erhalten. Ein Endpunkt zum Löschen von Bestellungen wird zunächst nicht angeboten, da er für das Anwendungsszenario nicht relevant ist.
Der Server wurde als Node.js Backend implementiert. Es wurden die Bibliotheken arangojs, der JavaScript Client für ArangoDB, express.js, für die API, und swagger-ui-express, für die visuelle Aufbereitung des Data-Contracts, eingesetzt. Auf eine Authentifikation wurde im Rahmen dieser Arbeit zunächst verzichtet. Die vier Endpunkte lassen sich über HTTP erreichen. Besonders interessant ist die Implementierung der dynamischen Query, um die Bestellungen zu analysieren. Hier wird ein Array erstellt, welches die einzelnen Zeilen der Query beinhaltet. Diese werden am Ende verkettet und an die Datenbank gesendet.

Supplier-Domäne

Äquivalent zur Order-Domäne verwaltet die Supplier-Domäne die Anbieter (Suppliers) und Artikel (Items) in einer ArangoDB DB-Instanz. Das Datenprodukt wird anderen Domänen ebenfalls als Datenvertrags in Form einer API-Spezifikation angeboten. Die Spezifikation nach OpenAPI Standard ist in Listing A.8 zu finden.
Die DB der Domäne ist in zwei Sammlungen aufgeteilt: Die `Items`-Sammung enthält alle Artikel. Die `Suppliers`-Sammlung enthält alle Anbieter und jeder Anbieter hat eine Attribut `itemList`, das eine Liste mit den `_key`s aller angebotenen Artikel enthält.
Für die Artikel werden Endpunkte angeboten, um einen Artikel zu erstellen, zu lesen, zu aktualisieren und zu löschen. Für die Anbieter werden Endpunkte zum

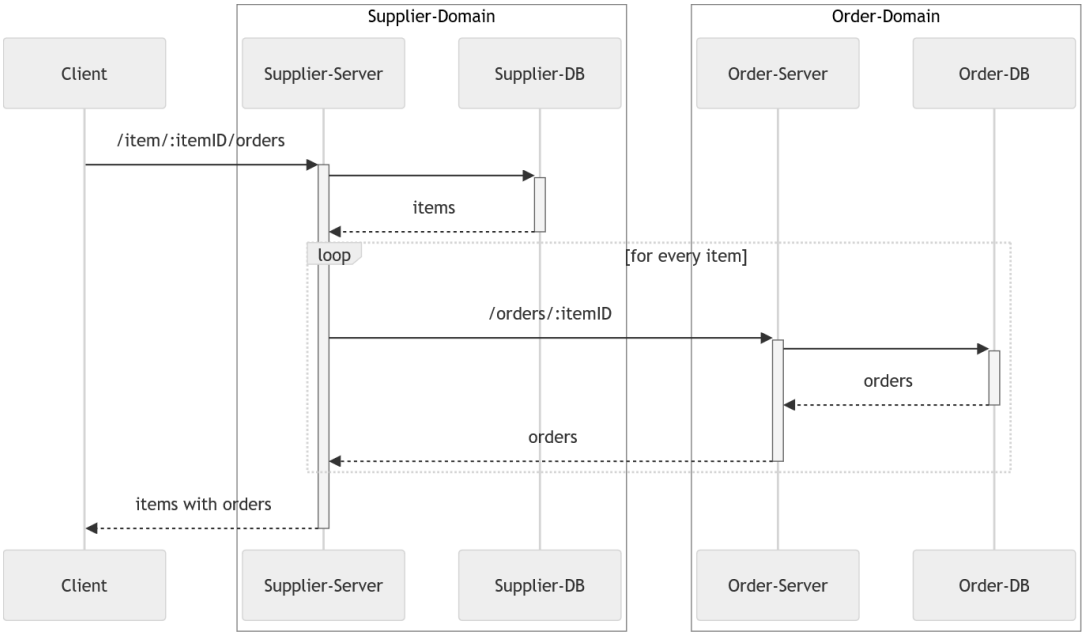


Abbildung 4.10: Sequenzdiagramm der Kommunikation bei einem Aufruf der beide Domänen einschließt

Erstellen, Lesen und Aktualisieren angeboten. Zur Aktualisieren stehen insgesamt drei Endpunkte zur Verfügung: Einer zum aktualisieren der eigentlichen Anbieterdaten und zwei um Artikel zur Liste der angebotenen Artikel hinzuzufügen bzw. zu entfernen.
Das Backend wurde in Python mit Flask und Flask-Restful-Swagger implementiert. Auf eine Authentifikation wurde auch hier verzichtet. Die Endpunkte lassen sich genau wie in der Order-Domäne über HTTP erreichen und können auf der Dokumentationsseite direkt getestet werden.

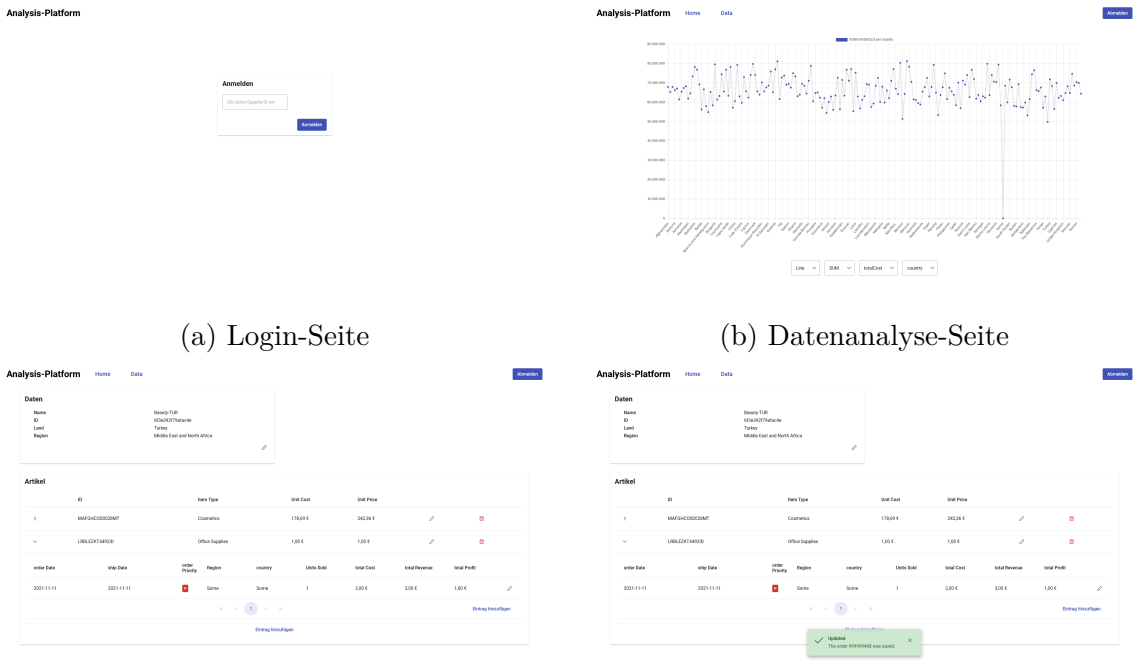
Kopplung der beiden Domänen

Die Daten der beiden Domänen sind über die zwei beschriebenen Endpunkte erreichbar. Für den Fall, dass nun die Daten aus beiden Domänen kombiniert werden müssen, muss dies in den meisten Fällen extern geschehen. Beispielsweise im Frontend. Es gibt jedoch auch den Fall, dass eine Domäne bei einer ihrer Analysen die Daten einer anderen Domäne benötigt. In diesem Fall muss ebenfalls auf den Endpunkt der anderen Domäne zugegriffen werden. Dies erhöht die Komplexität der Anfragen. Folgendes Beispiel soll dies veranschaulichen:
Die Anwendung soll alle Bestellungen eines Anbieters zeigen. Diese sollen unter den Artikeln gegliedert sein. Damit nicht pro Artikel des Anbieters vom Frontend eine Anfrage an den Server der Order-Domäne gesendet werden muss, sollen die Bestellungen direkt mit den Artikeln zusammen von der Supplier-Domäne aus versendet werden. Dafür müssen die Server miteinander kommunizieren. Die Kommunikation ist in Abbildung 4.10 als Sequenzdiagramm dargestellt. In diesem werden ebenfalls die Zugriffe auf die Datenbanken betrachtet. Aus dem Diagramm werden die

Zugriffsreihenfolge und die Anzahl der Aufrufe deutlich. Bei einer großen Anzahl von Artikeln steigt die Anwortzeit schnell an. Eine `JOIN`-Operation innerhalb einer Datenbank oder direkte Aufrufe der zwei Datenbanken innerhalb eines Backends würden den Kommunikationsaufwand stark reduzieren. Dennoch ist es für die Implementierung des Data-Mesh Konzepts notwendig, diesen Performance-Verlust hinzunehmen, um die Vorteile des Konzepts zu erhalten.
Neben dem erhöhten Kommunikationsaufwand stellte sich auch heraus, dass trotz der hohen Autonomie der einzelnen Domänen Absprachen zwischen den Domänen notwendig sind. Da die Zugriffe auf die Daten der Domänen mithilfe von APIs umgesetzt wurden, wird die Flexibilität eingeschränkt. So war es zunächst nicht vorgesehen, die Bestellungen über die `itemID` zu erhalten. Für die Supplier-Domäne war dies jedoch eine wichtige Eigenschaft, welche schlussendlich auch für das Anwendungsszenario relevant war. Das Problem liegt eindeutig in der Kapselung der Datenbank in einer API. Neben zu wenig Schnittstellen könnte sich auch ein Überschuss ergeben. Einige Schnittstellen werden zwar implementiert und angeboten, jedoch nicht zwangsläufig verwendet. Der Entwicklungsaufwand ist insgesamt unklar. Abhilfe könnten direkte Zugriffe auf die Datenbank schaffen. Diese muss dann jedoch auch die Möglichkeit einer Authentifizierung bieten. Auch wäre der Einsatz von Produkten wie Databricks [Dat23] vorstellbar, um den Zugriff auf die Datenbank einzuschränken und nach außen hin einen Data-Contract anzubieten.

Umsetzung des Frontends

Das Frontend der Webanwendung wurde mit dem Framework React umgesetzt. Daneben wurden die Bibliotheken Chart.js und PrimeReact eingesetzt, um den Entwicklungsaufwand zu reduzieren. Chart.js bietet eine umfangreiche Implementierung diverser Diagramme zur Datenvisualisierung, womit die Graphen für die Analysen realisiert wurden (Abbildung A.3). Die Bibliothek PrimeReact bietet hingegen eine Vielzahl an Komponenten, mit denen schnell eine moderne Anwendung umgesetzt werden konnte. Mit beiden Bibliotheken lassen sich die unter Skizzierung der Anwendung beschriebenen Anforderungen umsetzen.
Für die Entwicklung musste sich an die beiden APIs der Domänen angebunden werden. Dank zwei OpenAPI-Spezifikationen (Data-Contracts) konnten TypeScript Clients generiert werden, was ebenfalls Zeit einsparte. Die Clients wurden nach der Generierung in den Quellcode der Anwendung eingebunden und konnten dann über Funktionsaufrufe angesprochen werden. Darüber ließen sich die Daten von den Domänen erhalten. Die Daten wurden im Frontend nicht weiterverarbeitet, sondern nur graphisch dargestellt. Abbildung 4.11 zeigt einige Ausschnitte der Anwendung. Anzumerken ist hier, dass aufgrund der großen Testdatenmengen eine Aufteilung der Darstellung aus A.3 auf zwei separate Seiten sinnvoll war. Die Seite unter dem Pfad „Home“ stellt die Daten der Supplier über konfigurierbare Graphen dar (Abbildung 4.11b). Unter dem Pfad „Data“ ist eine tabellarische Ansicht aller Daten zu finden (Abbildung 4.11c). Über die geschachtelten Tabellen lassen sich zudem alle Daten manipulieren oder neue Datensätze hinzufügen.
Während der Implementierung des Frontends sind mehrfach Inkonsistenzen bei den



(a) Login-Seite

(b) Datenanalyse-Seite

(c) Datenansichts-Seite

(d) Datenansichts-Seite nach Update

Abbildung 4.11: Ausschnitte der Anwendung[4]

vorherigen Entwicklungsschritten aufgefallen. Diese traten vor allem bei den Identifikationswerten auf. Es kam zu einigen unterschiedlichen Schreibwerten sowie zu unterschiedlich vergebenen Werten in den beiden Domänen. Solche Inkonsistenzen traten auf, da im Kontext dieses Projekts die Governance-Group ausgelassen wurde (siehe 1). Diese ist jedoch wichtig, um genau diese Probleme frühzeitig zu vermeiden. Dennoch wirkten sich nicht nur der Einsatz von Data-Contracts positiv auf das Projekt aus. Auch das einfache Aufsetzen einer Storage- / Query-Engine (hier ArangoDB) half bei der Implementierung der beiden Domänen.

4.1.3 Fazit

Abschließend werden die Ergebnisse der vorangegangenen Abschnitte zusammengefasst. Demnach zuerst die Resultate der Leistungsbewertung und darauf folgend die des Data-Mesh-Experiments.

Leistungsbewertung

Die Experimente mit ArangoDB haben gezeigt, dass das System selbst bei extrem hoher Last gleichbleibende Laufzeiten für alle CRUD-Operationen hat. Dabei ist das System für große Datenmengen optimiert: Bei voller Auslastung ist das System etwas

[4] Alle Ausschnitte der Anwendung sind im Großformat erneut im Anhang unter A.4-A.7 zu finden.

schneller, als wenn es nicht ganz ausgelastet ist. Außerdem führen große Datensätze für einzelne Anfragen zu deutlich stabileren Laufzeiten als kleine. Warum das System unter voller Auslastung schneller als bei halber Auslastung ist, konnte im Rahmen dieser Arbeit nicht festgestellt werden. Es liegt jedoch die Vermutung nah, dass, wenn das System dauerhaft arbeitet, es effizienter cachen kann und es so eine Art „Warmlaufeffekt“ gibt – das System legt sich im übertragenen Sinne schlafen, wenn es nicht ausgelastet ist und braucht dann länger, um erst wieder aufzustehen.

Data-Mesh

Die Umsetzung des Data-Mesh-Konzepts war in dem vorgestellten Beispiel insgesamt erfolgreich. Vorteilhaft waren die Data-Contracts sowie das einfache Aufsetzen einer Datenbank, welches vom Data-Platform-Team ermöglicht wurde. Ebenso hat die Zerlegung in die Domänen dafür gesorgt, dass in mehreren Programmiersprachen entwickelt werden konnte, wodurch die Stärken der Probanden besser ausgenutzt werden konnten. Dennoch zeigte sich, dass das Konzept für größere Anwendungsfelder besser geeignet ist. Die Anwendung auf ein so kleines Beispielszenario konnte nicht alle Konzepte des Data-Mesh umsetzen, wodurch einige Stärken nicht genutzt werden konnten. Auch merkte man, dass einige Konzepte wichtig sind für eine gelungene Umsetzung. Darunter beispielsweise die Interoperability-Policy, welche für gewöhnlich von einer separaten Governance-Group bereitgestellt wird.

5 Fazit

5.1 ArangoDB & Data-Mesh

5.1.1 Zusammenfassung

In der Arbeit wurde zu Beginn der Data-Mesh-Ansatz zusammengefasst und die einzelnen Prinzipien, auf denen er aufbaut, erläutert. Diese vier Prinzipien sind das Domänenbesitzprinzip, Datenproduktprinzip, Selbstbedienungsdateninfrastrukturplattformprinzip und das föderierte Führungsprinzip. Dabei ließen sich die Prinzipien gut am Beispiel eines Gartens verstehen. Daraufhin wurde das DBS ArangoDB kurz charakterisiert (Abschnitt 2.1). Auf die Charakteristiken des DBSs wurde dann im Anschluss ausführlicher eingegangen. Dabei wurden unter anderem das Anwendungsumfeld sowie einige technologische Aspekte (Abschnitt 3.1.2) beleuchtet. Im diesem Zuge wurde im Abschnitt 3.1.3 an einem Beispiel beschrieben, wie die Entwicklung einer DB unter der Verwendung von ArangoDB aussehen kann. Dabei wurden auch Aspekte des Data-Mesh-Ansatzes in Betracht gezogen. Auf der Grundlage der entwickelten DB wurden im Kapitel 4 dann eine Leistungsbewertung durchgeführt und eine Anwendung nach dem Data-Mesh-Ansatz konstruiert. Es ergaben sich aus den Experimenten einige Erkenntnisse, die in Abschnitt 4.1.3 zusammengetragen wurden.
Die Vor- und Nachteile von ArangoDB und dem Data-Mesh-Ansatz, die in den Kapiteln zuvor erkannt wurden, wurden final in den folgenden zwei Abschnitten gesammelt.

5.1.2 Data-Mesh

Die einzelnen Prinzipien, auf denen der Data-Mesh-Ansatz aufbaut, haben allesamt ihre Stärken. Besonders nennenswert ist hier die grobgranulare Strukturierung, welche durch eine Aufteilung in mehrere spezialisierte Domänen erzielt wird. Mit der Aufteilung gehen weitere Vorteile wie verbesserte Datenprodukte, schnellere Entwicklungszyklen und spezialisierte Experten der Domänen einher. Daneben liegen auch einige Stärken von dem Ansatz in der zugrundeliegenden Infrastruktur, welche ebenfalls die Flexibilität erhöht und Redundanzen verhindert. (Abschnitt 1.1)
Neben den Stärken des Ansatzes hat sich im Data-Mesh-Experiment gezeigt, dass der Ansatz in kleinen Projekten fehl am Platz ist. Der Overhead, der durch die not-

wendige Infrastruktur entsteht und durch eine globale Administration sind zu hoch. Ebenfalls ist es schwer, sich auf seine Domäne zu spezialisieren, wenn man mehrere Rollen auf einmal erfüllen muss. Abhilfe könnte hier ein Plattform wie Databricks schaffen, welche jedoch auch nicht ganz günstig ist. (Abschnitt 1.1, 4.1.2)

5.1.3 ArangoDB

ArangoDB sticht besonders durch das flexible Datenmodell hervor. Auf die Daten lassen sich wie bei einem Document-Store, einem Key-Value-Store oder einer GraphDB zugreifen. Der Zugriffs Weg entscheidet sich durch die formulierte Abfrage, wobei auch mehrere Ansichten kombiniert werden können. Alle Sichten werden in der eigenen Abfragesprache AQL zusammengeführt, die an SQL anlehnt ist. Damit ermöglicht ArangoDB eine flexible und einfache Nutzung (Abschnitt 3.1.1, 3.1.2). Es eignet sich daher für den Einsatz als Data-Lake oder in der Webentwicklung (Abschnitt 3.1.1).
Des Weiteren hat die Leistungsbewertung gezeigt, dass ArangoDB sich sogar unter extremer Last performant verhält. Die Tests haben dabei gezeigt, dass auch das Erstellen von 1 000 000 Datensätzen sich trotz extremer Last bei ungefähr einer Minute befindet (Abbildung 4.4); das Lesen weiterhin nur im Sekundenbereich. Durch die Option des Shardings im Cluster-Betrieb lässt sich die bereits gute Performance weiter erhöhen. (Abschnitt 4.1.1)
ArangoDB wies vor allem zwei Nachteile auf. Zum einen fiel auf, dass sie nicht direkt auf Azure oder AWS deploybar ist. Dies geht nur über Umwege oder auf dem hauseigenen Cloud-Service. Der zweite Nachteil ist der Umfang von ArangoDB. An einigen Stellen ist AranogDB nicht so weit ausgebaut wie fest etablierte relationale DBSe. Beispielsweise die Auswahl der möglichen Typen ist stark eingeschränkt (Abschnitt 3.1.2).

5.1.4 „Leasons Learned“

Wie sich in dieser Arbeit mehrfach gezeigt hat ist der Ansatz von Data-Mesh für große Unternehmen gedacht und nicht für kleine Projekte, wie das hier betrachtete. Auch entstand unnötiger Kommunikationsaufwand durch den Einsatz von Data-Mesh (Abschnitt 4.1.2). Dennoch ließen sich die Vorteile für große Projekte erkennen. Wie unter Kopplung der beiden Domänen bemerkt wurde, spielt auch die Governance-Group eine wichtige Rolle, um einheitliche Schnittstellen zu implementieren.
Daneben stellte sich heraus, dass ArangoDB ein äußerst fähiges DBS ist, mit dem sich leicht arbeiten lässt. ArangoDB hat während der Datenmodellierung nach Data-Mesh-Prinzip keinerlei Probleme verursacht, was insgesamt sehr positiv aufgefallen ist. Ebenso haben die Leistungsbewertung gezeigt, dass ArangoDB auch unter hoher Last performant bleibt. Interessant wäre ein Vergleich mit anderen NoSQL-DBSen, um die Daten präziser einordnen zu können.

A Anhang

A.1 Abbildungen

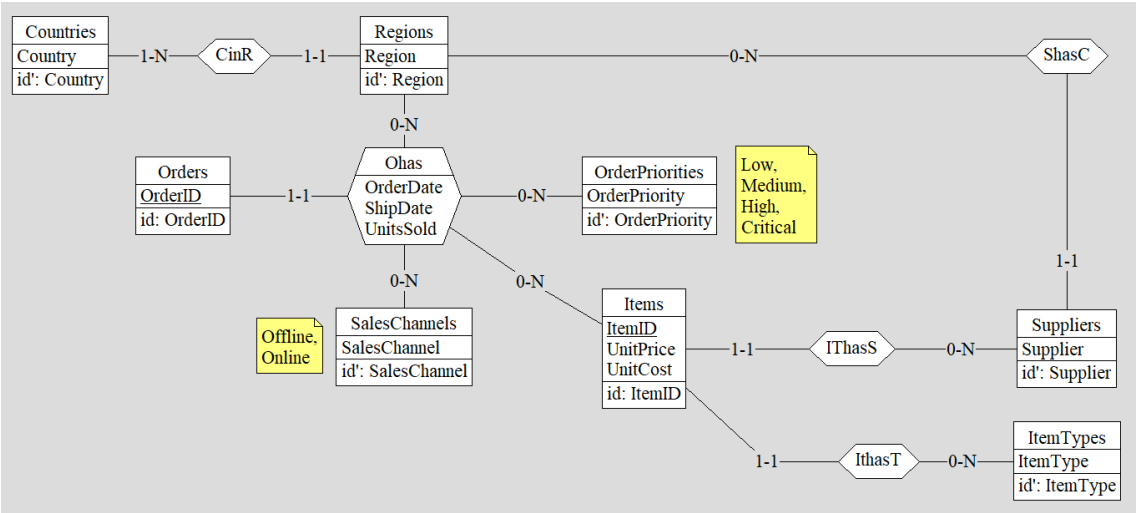


Abbildung A.1: Logisches Schema der importierten Daten

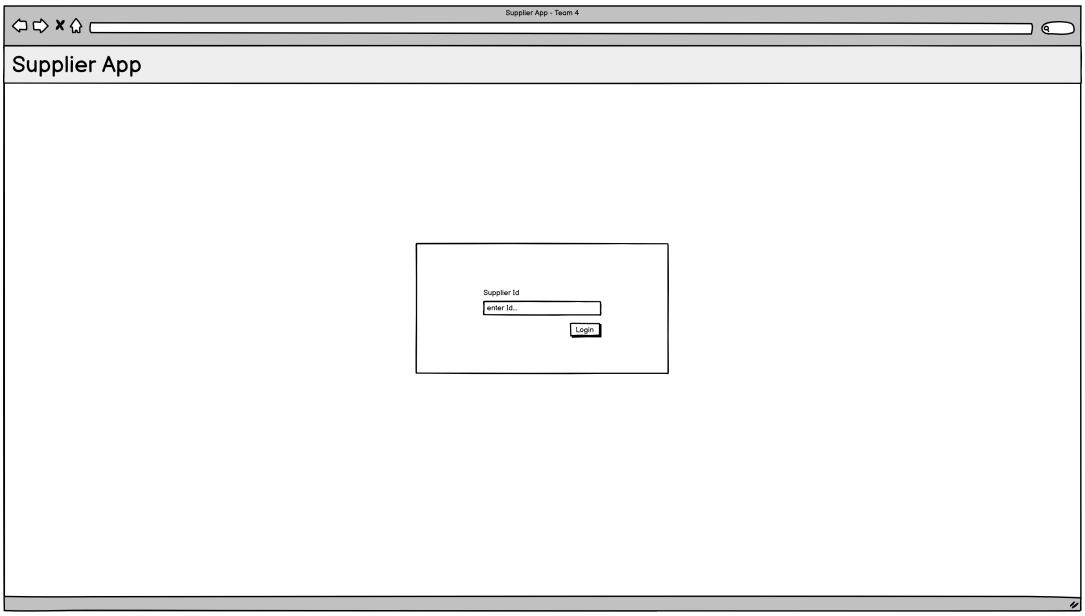


Abbildung A.2: Mockup des Logins

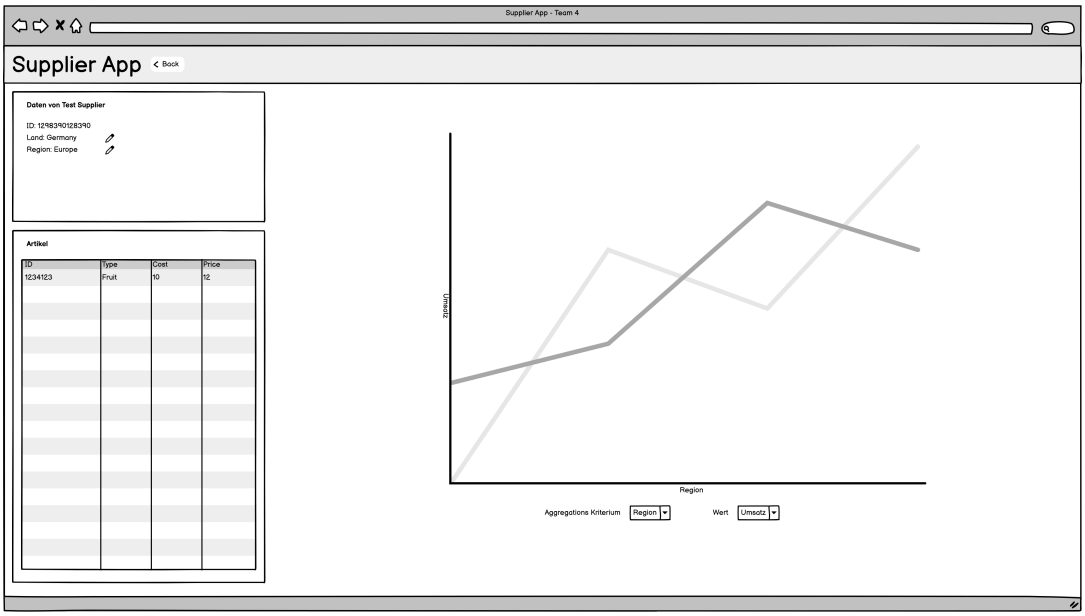


Abbildung A.3: Mockup der Anwendung

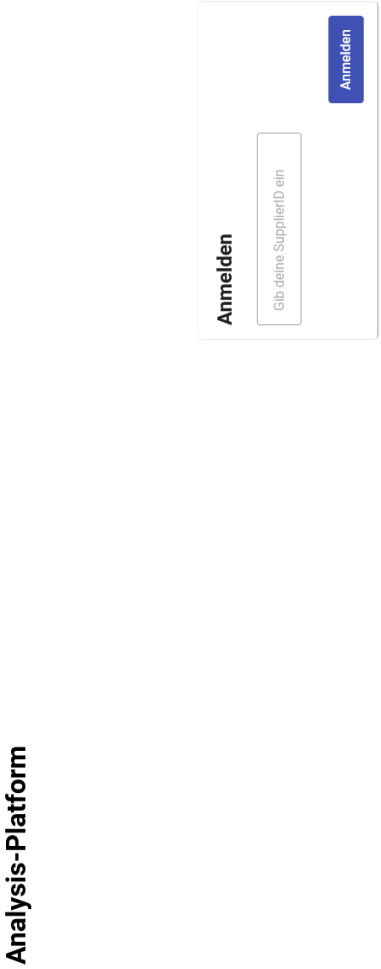


Abbildung A.4: Login-Seite

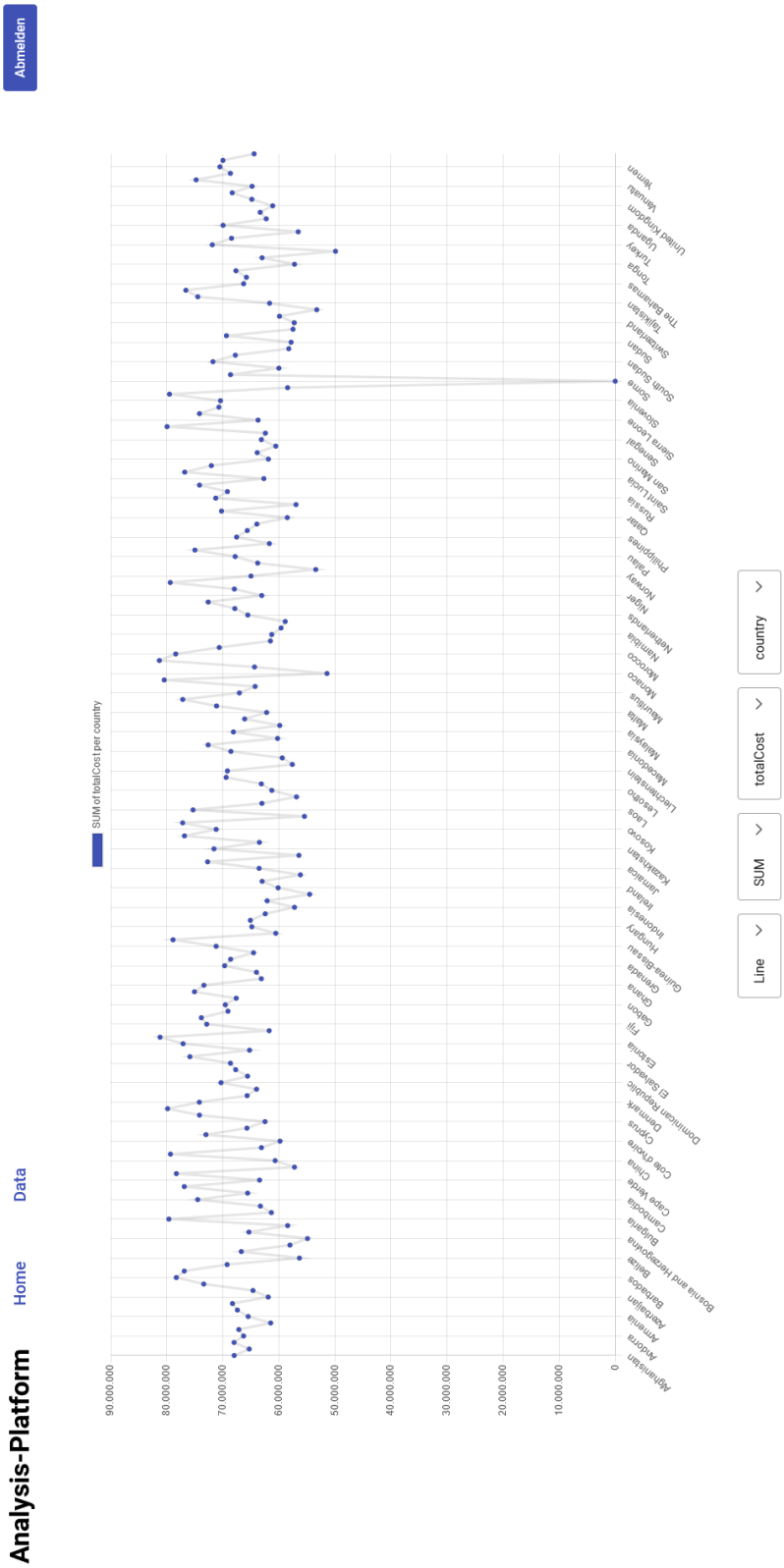


Abbildung A.5: Datenanalyse-Seite

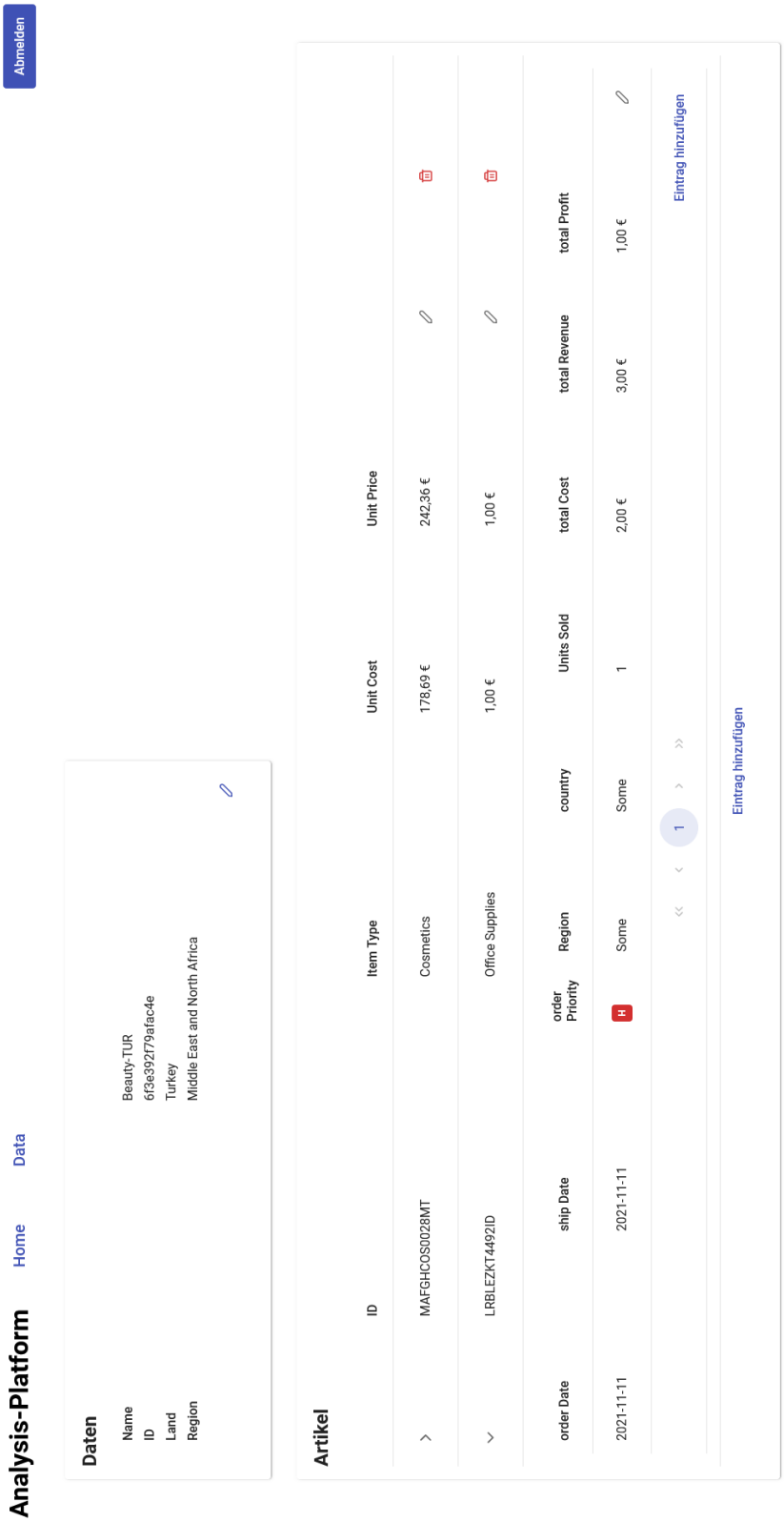


Abbildung A.6: Datenansichts-Seite

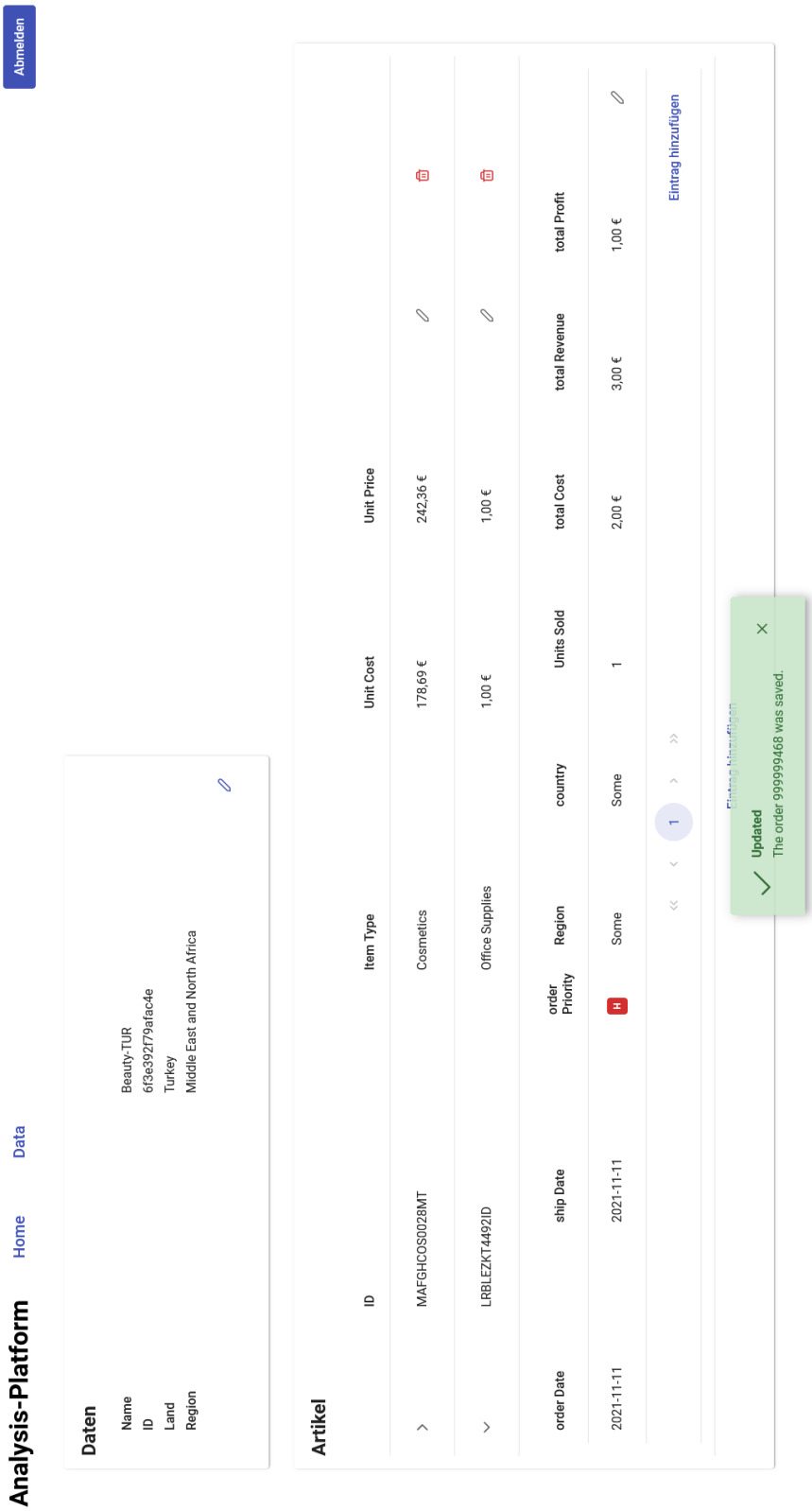


Abbildung A.7: Datenansichts-Seite nach Update

A.2 Vorverarbeitungsskripte

Alle hier aufgelisteten Skripte sind auch auf GitHub unter: https://github.com/some-random-int/schemaless-databases zu finden.

Listing A.1: subCsv.py – Ein Skript, das aus einer CSV-Datei eine andere CSV-Datei erstellt, die ein Subset der Spalten der Originaldatei enthält

```
1  import csv
2  import argparse
3
4  def get_column_input(column):
5      while True:
6          include = input(f"Should column '{column}' be included in the new file? (Y/n) ").lower()
7          if include in {'y', 'n', ''}:
8              return include
9          print("Invalid input. Please enter 'y', 'n', or nothing.")
10
11 def create_subset(input_file, output_file, columns=None):
12     # Read the CSV file
13     with open(input_file, 'r') as in_file:
14         reader = csv.DictReader(in_file, delimiter=';')
15
16         # If no columns are specified, ask about each column
17         if not columns:
18             columns = []
19             for column in reader.fieldnames:
20                 include = get_column_input(column)
21                 if include == 'y' or include == '':
22                     columns.append(column)
23
24         # Write the subset to the new file
25         with open(output_file, 'w', newline='') as outfile:
26             writer = csv.DictWriter(outfile, fieldnames=columns, delimiter=';')
27             writer.writeheader()
28
29             for row in reader:
30                 writer.writerow({column: row[column] for column in columns})
31
32 if __name__ == "__main__":
33     # Use argparse for command-line arguments
34     parser = argparse.ArgumentParser(description="Create a subset of a CSV file with specific columns.")
```

```
    parser.add_argument("input_file", help="Path␣to␣the␣input␣CSV␣file")
    parser.add_argument("output_file", help="Path␣to␣the␣output␣CSV␣file")
    parser.add_argument("-c", "--columns", nargs="+", help="Names␣of␣columns␣to␣be␣included␣in␣the␣output")

    args = parser.parse_args()

    # If no columns are specified, pass None
    columns = args.columns if args.columns else None

    create_subset(args.input_file, args.output_file, columns)
```

Listing A.2: prepare_items.py – Ein Skript, das Duplikate aus einer CSV-Datei entfernt

```
import argparse
import pandas as pd


def remove_duplicates(input_file):
    # Load CSV file
    df = pd.read_csv(input_file, sep=';')

    # Remove duplicates
    df.drop_duplicates(inplace=True)

    # Write results back to the same CSV file, overwriting the original file
    df.to_csv(input_file, index=False, sep=';')


def main():
    # Set up command-line arguments
    parser = argparse.ArgumentParser(
        description='Remove␣duplicate␣rows␣from␣a␣CSV␣file.')
    parser.add_argument('input_file', help='Path␣to␣the␣input␣CSV␣file')

    # Parse command-line arguments
    args = parser.parse_args()

    # Remove duplicates and overwrite the original file
    remove_duplicates(args.input_file)
```

```


if __name__ == '__main__':
    main()
```

Listing A.3: prepare_suppliers.py – Ein Skript, das SupplierIDs ergänzt, Rechtschreibfehler der Spaltennamen korregiert und Duplikate entfernt

```
import pandas as pd
import hashlib
import argparse

def generate_key(row):
    hash_value = hashlib.sha256(f"{row['Supplier']}_{row['SupplierRegion']}_{row['SupplierCountry']}".encode('utf-8')).hexdigest()
    key = hash_value[:16]
    return key

def main():
    # Parse command line arguments
    parser = argparse.ArgumentParser(description='Process CSV file and generate SupplierID column.')
    parser.add_argument('file_path', help='Path to the CSV file')
    args = parser.parse_args()

    # Load the CSV file
    df = pd.read_csv(args.file_path, delimiter=';')

    # Rename columns
    df = df.rename(columns={'SuplierRegion': 'SupplierRegion', 'SuplierCountry': 'SupplierCountry'})

    # Generate the _key column
    df['SupplierID'] = df.apply(generate_key, axis=1)

    # Drop duplicate rows based on all columns
    df.drop_duplicates(inplace=True)

    # Save the result to a new CSV file (same path)
    df.to_csv(args.file_path, index=False, sep=';')
    print("Modified file saved")

if __name__ == "__main__":
    main()
```

Listing A.4: create_item_list.py – Ein Skript, das in einer supplier.json-Datei auf Basis einer item2supplier.csv-Datei für jedes Objekt eine „ItemList“ ergänzt, die alle „ItemID“s enthält, die mit dem jeweiligen Supplier assoziiert werden.

```
import argparse
import json
import csv

def load_json(file_path):
    with open(file_path, 'r') as json_file:
        return json.load(json_file)

def load_csv(file_path):
    with open(file_path, 'r') as csv_file:
        reader = csv.DictReader(csv_file, delimiter=';')
        return list(reader)

def update_json_with_items(json_data, csv_data):
    for entry in json_data:
        supplier_name = entry.get('Supplier')
        associated_items = {row['ItemID'] for row in
            csv_data if row['Supplier'] == supplier_name}
        entry['ItemList'] = list(associated_items)
    return json_data

def save_json(updated_json_data, output_file):
    with open(output_file, 'w') as json_output_file:
        json.dump(updated_json_data, json_output_file,
            indent=2)

def main():
    parser = argparse.ArgumentParser(description='Update␣
        JSON␣objects␣with␣associated␣unique␣ItemIDs␣from␣a␣
        CSV␣file.')
    parser.add_argument('suppliers_json', help='Path␣to␣the␣
        suppliers␣JSON␣file')
    parser.add_argument('item2supplier_csv', help='Path␣to␣
        the␣item2supplier␣CSV␣file')
    parser.add_argument('output_json', help='Path␣to␣the␣
        output␣JSON␣file')
    args = parser.parse_args()

    suppliers_data = load_json(args.suppliers_json)
    item2supplier_data = load_csv(args.item2supplier_csv)

```

```
    updated_suppliers_data = update_json_with_items(suppliers_data, item2supplier_data)

    save_json(updated_suppliers_data, args.output_json)
    print(f'Updated JSON saved to {args.output_json}')

if __name__ == '__main__':
    main()
```

Listing A.5: create-db-json.sh – Ein Skript, dass die anderen Skripte dieser Sektion verwendet, um die Daten für die Suppliers-Domäne aufzubereiten.

```
pushd .
cd ~/team23w4/supplier/data
python3 ~/team23w4/Country-Sales-Data-GeneratorPlus03/subCsv.py ~/team23w4/Country-Sales-Data-GeneratorPlus03/Country-Sales-Data-GeneratorPlus03_1Mio.csv items.csv -c "Item Type" "Unit Price" "Unit Cost" "ItemID"
python3 ~/team23w4/Country-Sales-Data-GeneratorPlus03/subCsv.py ~/team23w4/Country-Sales-Data-GeneratorPlus03/Country-Sales-Data-GeneratorPlus03_1Mio.csv suppliers.csv -c "Supplier" "SuplierRegion" "SuplierCountry"
python3 ~/team23w4/Country-Sales-Data-GeneratorPlus03/subCsv.py ~/team23w4/Country-Sales-Data-GeneratorPlus03/Country-Sales-Data-GeneratorPlus03_1Mio.csv item2supplier.csv -c "Supplier" "ItemID"
python3 ../prepare_items.py items.csv
python3 ../prepare_suppliers.py suppliers.csv
python3 ~/team23w4/Country-Sales-Data-GeneratorPlus03/csv2json.py items.csv -o items.json
python3 ~/team23w4/Country-Sales-Data-GeneratorPlus03/csv2json.py suppliers.csv -o suppliers.json
python3 ../create_item_list.py suppliers.json item2supplier.csv suppliers.json
popd
```

Listing A.6: csv2json.py – Ein Skript, das eine CSV-Datei in eine JSON-Datei umwandelt und Zahleneintrage in den Typen „number“ umwandelt

```
import csv
import json
import argparse


def convert_csv_numeric(csv_dict_reader: csv.DictReader) -> csv.DictReader:
    """
```

```
    Converts numeric strings in a CSV DictReader to their corresponding float values.

    Parameters:
        csv_dict_reader (csv.DictReader): The CSV DictReader containing the data.

    Returns:
        csv.DictReader: The CSV DictReader with numeric string values converted to floats.
    """
    for row in csv_dict_reader:
        for key, value in row.items():
            # Check if the value is a numeric string
            if isinstance(value, str) and value.replace(",", "").replace(".", "").isdigit():
                # Replace , with . if present and convert the value to float
                if ',' in value:
                    row[key] = float(value.replace(",", ".")
                else:
                    row[key] = float(value)
    return csv_dict_reader


def csv_to_json(csv_path: str, output_file: str | None = None):
    """
    Convert a CSV file to a JSON file.

    Args:
        csv_file (str): The path or URL to the CSV file.
        output_file (str, optional): The path to the output JSON file. If not provided, the result will be printed.

    Returns:
        dict: The JSON data.

    Raises:
        FileNotFoundError: If the CSV file is not found.
    """
    try:
        # Read CSV file with semicolon as the delimiter
        with open(csv_path, 'r') as csv_file:
```

```
45                csv_reader = csv.DictReader(csv_file, delimiter=
                     ';')
46                data = list(csv_reader)
47
48            # Cast numerical values
49            typed_data = convert_csv_numeric(data)
50
51            # Convert to JSON
52            json_data = json.dumps(typed_data, indent=2)
53
54            # Output to file or print to console
55            if output_file:
56                with open(output_file, 'w') as json_output:
57                    json_output.write(json_data)
58            else:
59                print(json_data)
60
61            return data
62        except FileNotFoundError:
63            raise FileNotFoundError(f"CSV␣file␣'{csv_path}'␣not␣
                 found.")
64
65
66    def main():
67        # Argument parser setup
68        parser = argparse.ArgumentParser(description='Convert␣
             CSV␣to␣JSON.')
69        parser.add_argument('csv_file', help='Path␣or␣URL␣to␣the
             ␣CSV␣file')
70        parser.add_argument('-o', '--output', help='Path␣to␣the␣
             output␣JSON␣file')
71
72        # Parse arguments
73        args = parser.parse_args()
74
75        # Convert CSV to JSON
76        result = csv_to_json(args.csv_file, args.output)
77
78        return result
79
80
81    if __name__ == "__main__":
82        main()
```

A.3 Data-Contracts

Listing A.7: order-openapi.json – Data Contract für die Order-Domäne.

```
1  {
2    "openapi": "3.0.3",
3    "info": {
4      "title": "order-domain",
5      "description": "This is a API-Documentation for the
         order-domain. It serves as a data-contract.",
6      "contact": {
7        "email": "frederick.behringer@smail.inf.h-brs.de"
8      },
9      "license": {
10       "name": "Apache 2.0",
11       "url": "http://www.apache.org/licenses/LICENSE-2.0.
           html"
12     },
13     "version": "1.0.0"
14   },
15   "servers": [
16     {
17       "url": "http://localhost:5001"
18     }
19   ],
20   "tags": [
21     {
22       "name": "order",
23       "description": "Access orders"
24     }
25   ],
26   "paths": {
27     "/api/order": {
28       "post": {
29         "tags": [
30           "order"
31         ],
32         "summary": "Create a new order",
33         "description": "Create a new order",
34         "operationId": "createOrder",
35         "requestBody": {
36           "content": {
37             "application/json": {
38               "schema": {
39                 "$ref": "#/components/schemas/Order"
40               }
41             }
```

```
42            },
43            "required": true
44          },
45          "responses": {
46            "201": {
47              "description": "Created"
48            }
49          }
50        }
51      },
52      "/api/order/{id}": {
53        "post": {
54          "tags": [
55            "order"
56          ],
57          "summary": "Update an existing order",
58          "description": "Update an existing order",
59          "operationId": "updateOrder",
60          "parameters": [
61            {
62              "name": "id",
63              "in": "path",
64              "description": "ID of order",
65              "required": true,
66              "schema": {
67                "type": "string"
68              }
69            }
70          ],
71          "requestBody": {
72            "content": {
73              "application/json": {
74                "schema": {
75                  "$ref": "#/components/schemas/Order"
76                }
77              }
78            },
79            "required": true
80          },
81          "responses": {
82            "200": {
83              "description": "Successfull operation"
84            },
85            "400": {
86              "description": "Invalid Id"
87            }
```

```
88          }
89        },
90        "get": {
91          "tags": [
92            "order"
93          ],
94          "summary": "Get an existing order",
95          "description": "Get an existing order",
96          "operationId": "getOrder",
97          "parameters": [
98            {
99              "name": "id",
100             "in": "path",
101             "description": "ID of order",
102             "required": true,
103             "schema": {
104               "type": "string"
105             }
106           }
107         ],
108         "responses": {
109           "200": {
110             "description": "Successfull operation",
111             "content": {
112               "application/json": {
113                 "schema": {
114                   "$ref": "#/components/schemas/Order"
115                 }
116               }
117             }
118           },
119           "400": {
120             "description": "Invalid Id"
121           }
122         }
123       }
124     },
125     "/api/orders": {
126       "get": {
127         "tags": [
128           "order"
129         ],
130         "summary": "analyses the orders",
131         "description": "analyses the orders",
132         "operationId": "analyseOrders",
133         "parameters": [
```

```
134             {
135               "in": "query",
136               "name": "aggregator",
137               "schema": {
138                 "type": "string",
139                 "enum": [
140                   "country",
141                   "itemID",
142                   "supplierID",
143                   "salesChannel",
144                   "orderPriority"
145                 ]
146               },
147               "description": "After what should be grouped"
148             },
149             {
150               "in": "query",
151               "name": "aggregated",
152               "schema": {
153                 "type": "string",
154                 "enum": [
155                   "unitsSold",
156                   "totalRevenue",
157                   "totalCost",
158                   "totalProfit",
159                   "orderDate",
160                   "shipDate",
161                   "processDays"
162                 ]
163               },
164               "description": "Which parameter is subject of
                    the aggregation operator"
165             },
166             {
167               "in": "query",
168               "name": "operator",
169               "schema": {
170                 "type": "string",
171                 "enum": [
172                   "COUNT",
173                   "MIN",
174                   "MAX",
175                   "AVG",
176                   "SUM"
177                 ]
178               }
```

```
179             }
180           ],
181           "responses": {
182             "200": {
183               "description": "Successfull operation"
184             },
185             "400": {
186               "description": "Invalid parameters"
187             }
188           }
189         }
190       }
191     },
192     "components": {
193       "schemas": {
194         "Order": {
195           "type": "object",
196           "properties": {
197             "_key": {
198               "type": "string",
199               "example": "123890213"
200             },
201             "orderDate": {
202               "type": "string",
203               "format": "date-time"
204             },
205             "shipDate": {
206               "type": "string",
207               "format": "date-time"
208             },
209             "orderPriority": {
210               "type": "string",
211               "description": "Order priority",
212               "example": "L",
213               "enum": [
214                 "H",
215                 "M",
216                 "L"
217               ]
218             },
219             "region": {
220               "type": "string"
221             },
222             "country": {
223               "type": "string"
224             },
```

```
225             "unitsSold": {
226               "type": "integer",
227               "format": "int64",
228               "example": 10
229             },
230             "totalCost": {
231               "type": "integer",
232               "format": "int64",
233               "example": 10
234             },
235             "totalRevenue": {
236               "type": "integer",
237               "format": "int64",
238               "example": 10
239             },
240             "totalProfit": {
241               "type": "integer",
242               "format": "int64",
243               "example": 10
244             },
245             "itemId": {
246               "type": "string"
247             },
248             "supplierId": {
249               "type": "string"
250             }
251           }
252         }
253       }
254     }
255 }
```

Listing A.8: supplier-openapi.json – Data Contract für die Supplier-Domäne.

```
1  {
2    "openapi": "3.0.2",
3    "info": {
4      "description": "",
5      "termsOfService": "",
6      "title": "Supplier API",
7      "contact": {},
8      "license": {},
9      "version": "1.0.0"
10   },
11   "components": {
12     "schemas": {
13       "ItemModel": {
```

```
14          "properties": {
15            "_key": {
16              "maxLength": 14,
17              "minLength": 14,
18              "readOnly": true,
19              "required": true,
20              "type": "string"
21            },
22            "itemType": {
23              "enum": [
24                "Office Supplies",
25                "Vegetables",
26                "Baby Food",
27                "Cosmetics",
28                "Personal Care",
29                "Meat",
30                "Snacks",
31                "Household",
32                "Beverages",
33                "Fruits",
34                "Clothes",
35                "Cereal"
36              ],
37              "required": true,
38              "type": "string"
39            },
40            "unitCost": {
41              "minimum": 0,
42              "required": true,
43              "type": "number"
44            },
45            "unitPrice": {
46              "minimum": 0,
47              "required": true,
48              "type": "number"
49            }
50          },
51          "type": "object"
52        },
53        "SupplierModel": {
54          "properties": {
55            "name": {
56              "type": "string"
57            },
58            "_key": {
59              "maxLength": 16,
```

```
60                "minLength": 16,
61                "readOnly": true,
62                "required": true,
63                "type": "string"
64              },
65              "country": {
66                "type": "string",
67                "required": true
68              },
69              "itemList": {
70                "items": {
71                  "type": "string"
72                },
73                "type": "array",
74                "required": true
75              },
76              "region": {
77                "type": "string",
78                "required": true
79              }
80            },
81            "type": "object"
82          }
83        }
84      },
85      "paths": {
86        "/item": {
87          "post": {
88            "requestBody": {
89              "content": {
90                "application/json": {
91                  "schema": {
92                    "$ref": "#/components/schemas/ItemModel"
93                  }
94                }
95              },
96              "description": "Request body",
97              "required": false
98            },
99            "tags": [
100             "Item"
101           ],
102           "parameters": [],
103           "responses": {
104             "400": {
105               "content": {
```

```
106                 "application/json": {
107                   "schema": {
108                     "$ref": "#/components/schemas/ItemModel"
109                   },
110                   "example": {
111                     "_key": "string",
112                     "itemType": "string",
113                     "unitCost": "number",
114                     "unitPrice": "number"
115                   }
116                 }
117               },
118               "description": "Got an error while insert"
119             },
120             "200": {
121               "content": {
122                 "application/json": {
123                   "schema": {
124                     "$ref": "#/components/schemas/ItemModel"
125                   },
126                   "example": {
127                     "_key": "string",
128                     "itemType": "string",
129                     "unitCost": "number",
130                     "unitPrice": "number"
131                   }
132                 }
133               },
134               "description": "Add a new Item"
135             }
136           },
137           "summary": "Add the given Item to the collection"
138         }
139       },
140       "/item/{ItemID}": {
141         "get": {
142           "tags": [
143             "Item"
144           ],
145           "parameters": [
146             {
147               "description": "ItemID",
148               "in": "path",
149               "name": "ItemID",
150               "required": "true"
151             }
```

```
152         ],
153         "responses": {
154           "404": {
155             "content": {
156               "application/json": {}
157             },
158             "description": "No Item with the given id exists
                 "
159           },
160           "200": {
161             "content": {
162               "application/json": {
163                 "schema": {
164                   "$ref": "#/components/schemas/ItemModel"
165                 },
166                 "example": {
167                   "_key": "string",
168                   "itemType": "string",
169                   "unitCost": "number",
170                   "unitPrice": "number"
171                 }
172               }
173             },
174             "description": "Get the Item with the given ID"
175           }
176         },
177         "summary": "Get an Item by its ID"
178       },
179       "delete": {
180         "tags": [
181           "Item"
182         ],
183         "parameters": [
184           {
185             "description": "ItemID",
186             "in": "path",
187             "name": "ItemID",
188             "required": "true"
189           }
190         ],
191         "responses": {
192           "404": {
193             "content": {
194               "application/json": {}
195             },
```

```
196             "description": "No Item with the given id exists
                  "
197           },
198           "204": {
199             "content": {
200               "application/json": {
201                 "schema": {
202                   "$ref": "#/components/schemas/ItemModel"
203                 },
204                 "example": {
205                   "_key": "string",
206                   "itemType": "string",
207                   "unitCost": "number",
208                   "unitPrice": "number"
209                 }
210               }
211             },
212             "description": "Delete the Item with the given
                  ID"
213           }
214         },
215         "summary": "Delete an Item by its ID"
216       },
217       "put": {
218         "requestBody": {
219           "content": {
220             "application/json": {
221               "schema": {
222                 "$ref": "#/components/schemas/ItemModel"
223               }
224             }
225           },
226           "description": "Request body",
227           "required": false
228         },
229         "tags": [
230           "Item"
231         ],
232         "parameters": [
233           {
234             "description": "ItemID",
235             "in": "path",
236             "name": "ItemID",
237             "required": "true"
238           }
239         ],
```

```
240         "responses": {
241           "404": {
242             "content": {
243               "application/json": {
244                 "schema": {
245                   "$ref": "#/components/schemas/ItemModel"
246                 },
247                 "example": {
248                   "_key": "string",
249                   "itemType": "string",
250                   "unitCost": "number",
251                   "unitPrice": "number"
252                 }
253               }
254             },
255             "description": "No Item with the given id exists
                  "
256           },
257           "204": {
258             "content": {
259               "application/json": {
260                 "schema": {
261                   "$ref": "#/components/schemas/ItemModel"
262                 },
263                 "example": {
264                   "_key": "string",
265                   "itemType": "string",
266                   "unitCost": "number",
267                   "unitPrice": "number"
268                 }
269               }
270             },
271             "description": "No changes to do"
272           },
273           "200": {
274             "content": {
275               "application/json": {
276                 "schema": {
277                   "$ref": "#/components/schemas/ItemModel"
278                 },
279                 "example": {
280                   "_key": "string",
281                   "itemType": "string",
282                   "unitCost": "number",
283                   "unitPrice": "number"
284                 }
```

```
285                 }
286               },
287               "description": "Update the Item with the given
                    ID. Return the new item"
288             }
289           },
290           "summary": "Update an Item by its ID"
291         }
292       },
293       "/item/{ItemID}/orders": {
294         "get": {
295           "tags": [
296             "Item"
297           ],
298           "parameters": [
299             {
300               "description": "ItemID",
301               "in": "path",
302               "name": "ItemID",
303               "required": "true"
304             }
305           ],
306           "responses": {
307             "404": {
308               "content": {
309                 "application/json": {}
310               },
311               "description": "Item with the given id does not
                    exist"
312             },
313             "200": {
314               "content": {
315                 "application/json": {
316                   "schema": {
317                     "$ref": "#/components/schemas/ItemModel"
318                   },
319                   "example": [
320                     {
321                       "_key": "string",
322                       "itemType": "string",
323                       "unitCost": "number",
324                       "unitPrice": "number"
325                     }
326                   ]
327                 }
328               },
```

```
329                "description": "Get orders of item"
330              }
331            },
332            "summary": "Get all orders of a given item"
333          }
334        },
335        "/supplier": {
336          "post": {
337            "requestBody": {
338              "content": {
339                "application/json": {
340                  "schema": {
341                    "$ref": "#/components/schemas/SupplierModel"
342                  }
343                }
344              },
345              "description": "Request body",
346              "required": false
347            },
348            "tags": [
349              "Supplier"
350            ],
351            "parameters": [],
352            "responses": {
353              "400": {
354                "content": {
355                  "application/json": {
356                    "schema": {
357                      "$ref": "#/components/schemas/
                           SupplierModel"
358                    },
359                    "example": {
360                      "name": "string",
361                      "_key": "string",
362                      "country": "string",
363                      "itemList": [
364                        "string"
365                      ],
366                      "region": "string"
367                    }
368                  }
369                },
370                "description": "Got an error while insert"
371              },
372              "200": {
373                "content": {
```

```
374                 "application/json": {
375                   "schema": {
376                     "$ref": "#/components/schemas/
                          SupplierModel"
377                   },
378                   "example": {
379                     "name": "string",
380                     "_key": "string",
381                     "country": "string",
382                     "itemList": [
383                       "string"
384                     ],
385                     "region": "string"
386                   }
387                 }
388               },
389               "description": "Add a new Supplier"
390             }
391           },
392           "summary": "Add the given Supplier to the collection
                "
393         }
394       },
395       "/supplier/{SupplierID}": {
396         "get": {
397           "tags": [
398             "Supplier"
399           ],
400           "parameters": [
401             {
402               "description": "SupplierID",
403               "in": "path",
404               "name": "SupplierID",
405               "required": "true"
406             }
407           ],
408           "responses": {
409             "404": {
410               "content": {
411                 "application/json": {}
412               },
413               "description": "No Supplier with the given id
                    exists"
414             },
415             "200": {
416               "content": {
```

```
417                 "application/json": {
418                   "schema": {
419                     "$ref": "#/components/schemas/
                          SupplierModel"
420                   },
421                   "example": {
422                     "name": "string",
423                     "_key": "string",
424                     "country": "string",
425                     "itemList": [
426                       "string"
427                     ],
428                     "region": "string"
429                   }
430                 }
431               },
432               "description": "Get the Supplier with the given
                    ID"
433             }
434           },
435           "summary": "Get a Supplier by its ID"
436         },
437         "put": {
438           "tags": [
439             "Supplier"
440           ],
441           "parameters": [
442             {
443               "description": "SupplierID",
444               "in": "path",
445               "name": "SupplierID",
446               "required": "true"
447             }
448           ],
449           "responses": {
450             "404": {
451               "content": {
452                 "application/json": {}
453               },
454               "description": "No Supplier with the given id
                    exists"
455             },
456             "204": {
457               "content": {
458                 "application/json": {}
459               },
```

```
460             "description": "No changes to do"
461           },
462           "200": {
463             "content": {
464               "application/json": {
465                 "schema": {
466                   "$ref": "#/components/schemas/
                        SupplierModel"
467                 },
468                 "example": {
469                   "name": "string",
470                   "_key": "string",
471                   "country": "string",
472                   "itemList": [
473                     "string"
474                   ],
475                   "region": "string"
476                 }
477               }
478             },
479             "description": "Update the Supplier with the
                  given ID. Return the new Supplier"
480           }
481         },
482         "summary": "Update an Supplier by its ID"
483       }
484     },
485     "/supplier/{SupplierID}/supplies/{ItemID}": {
486       "delete": {
487         "tags": [
488           "Supplier"
489         ],
490         "parameters": [
491           {
492             "description": "SupplierID",
493             "in": "path",
494             "name": "SupplierID",
495             "required": "true"
496           },
497           {
498             "description": "ItemID",
499             "in": "path",
500             "name": "ItemID",
501             "required": "true"
502           }
503         ],
```

```
504         "responses": {
505           "404": {
506             "content": {
507               "application/json": {}
508             },
509             "description": "Supplier or Item does not exist"
510           },
511           "204": {
512             "content": {
513               "application/json": {}
514             },
515             "description": "Supplier already does not supply
                  the item"
516           },
517           "200": {
518             "content": {
519               "application/json": {
520                 "schema": {
521                   "$ref": "#/components/schemas/
                        SupplierModel"
522                 },
523                 "example": {
524                   "name": "string",
525                   "_key": "string",
526                   "country": "string",
527                   "itemList": [
528                     "string"
529                   ],
530                   "region": "string"
531                 }
532               }
533             },
534             "description": "Remove an item from the
                  suppliers item list"
535           }
536         },
537         "summary": "Remove the given ItemID from the item
              list of the given Supplier"
538       },
539       "put": {
540         "tags": [
541           "Supplier"
542         ],
543         "parameters": [
544           {
545             "description": "SupplierID",
```

```
546                 "in": "path",
547                 "name": "SupplierID",
548                 "required": "true"
549               },
550               {
551                 "description": "ItemID",
552                 "in": "path",
553                 "name": "ItemID",
554                 "required": "true"
555               }
556             ],
557             "responses": {
558               "404": {
559                 "content": {
560                   "application/json": {}
561                 },
562                 "description": "Supplier or Item does not exist"
563               },
564               "204": {
565                 "content": {
566                   "application/json": {}
567                 },
568                 "description": "Supplier already supplies the
                      item"
569               },
570               "200": {
571                 "content": {
572                   "application/json": {
573                     "schema": {
574                       "$ref": "#/components/schemas/
                            SupplierModel"
575                     },
576                     "example": {
577                       "name": "string",
578                       "_key": "string",
579                       "country": "string",
580                       "itemList": [
581                         "string"
582                       ],
583                       "region": "string"
584                     }
585                   }
586                 },
587                 "description": "Add an item to a suppliers item
                      list"
588               }
```

```
589         },
590         "summary": "Add the given ItemID to the item list of
                the given Supplier"
591       }
592     }
593   }
594 }
```

A.4 Weitere Skripte

Listing A.9: run-arango.sh – Startet die ArangoDB-Instanzen auf allen drei Knoten.

```
1  #!/bin/bash
2
3  nodes=("c017-node1" "c017-node2" "c017-node3")
4
5  function stop_all_nodes {
6      for ((i=${#nodes[@]}-1; i>=0; i--))
7      do
8          n="${nodes[i]}"
9          echo Stop DB on $n
10         ssh "$n" 'pkill␣arango'
11     done
12 }
13
14 trap stop_all_nodes EXIT
15
16 eval "$(ssh-agent␣-s)"
17 ssh-add ~/.ssh/team23w4
18
19 echo Start DB on ${nodes[0]}
20 ssh "${nodes[0]}" '~/Downloads/arangodb3-linux-3.11.4_x86_64
      /bin/arangodb␣--starter.mode␣cluster␣--starter.data-dir␣
      ~/arango-data/␣' &
21
22 for ((i=1; i<${#nodes[@]}; i++))
23 do
24     n="${nodes[i]}"
25     echo Start and join DB on $n
26
27     ssh "$n" '~/Downloads/arangodb3-linux-3.11.4_x86_64/bin/
          arangodb␣--starter.data-dir␣~/arango-data/␣--starter.
          join␣10.20.110.66␣' &
28 done
29
30 wait
```

Listing A.10: create_data.py – Erzeugt aus den vorhandenen Datensätze neue Varianten und speichert diese ab.

```
1  from datetime import datetime, timedelta
2  import json
3  import math
4  import random
5  import concurrent
6  import concurrent.futures
7
8
9  def generate_documents(orig: dict[list], count_per_thread:
       int):
10     new_docs = orig.copy()
11
12     while len(new_docs) < count_per_thread:
13         new_docs += orig.copy()
14     new_docs = new_docs[:count_per_thread]
15
16     for j, _ in enumerate(new_docs):
17         new_docs[j]['Sales␣Channel'] = 'Online' if random.
               random(
18         ) < 0.5 else 'Offline'
19         new_docs[j]['Order␣Priority'] = random.choice(['L',
               'M', 'H', 'C'])
20         start_date = datetime(2010, 1, 1)
21         end_date = datetime(2023, 12, 31)
22         end_date2 = datetime(2024, 1, 6)
23         order_date = start_date + \
24             timedelta(days=random.randint(0, (end_date -
                   start_date).days))
25         ship_date = order_date + \
26             timedelta(days=random.randint(
27                 0, (end_date2 - order_date).days))
28         new_docs[j]['Order␣Date'] = order_date.strftime('%Y
               -%m-%d')
29         new_docs[j]['Ship␣Date'] = ship_date.strftime('%Y-%m
               -%d')
30         new_docs[j]['Order␣ID'] = str(random.randint
               (100000000, 999999999))
31     return new_docs
32
33
34 def generate_documents_parallel(orig, count_per_thread,
       num_threads):
35     with concurrent.futures.ThreadPoolExecutor(max_workers=
           num_threads) as executor:
```

```
36          futures = [executor.submit(
37              generate_documents, orig, count_per_thread) for
                _ in range(num_threads)]
38          concurrent.futures.wait(futures)
39          results = [future.result() for future in futures]
40
41      merged_results = [item for sublist in results for item
            in sublist]
42      seen_oids = set()
43      filtered_results: list[dict] = []
44      for x in merged_results:
45          oid = x['Order␣ID']
46          if oid not in seen_oids:
47              seen_oids.add(oid)
48          filtered_results.append(x)
49      return filtered_results
50
51
52  if __name__ == '__main__':
53      docs: list[dict] = []
54      with open('../Country-Sales-Data-GeneratorPlus03/Country
            -Sales-Data-GeneratorPlus03_1Mio.json', 'r', encoding
            ='UTF-8') as f:
55          d = json.load(f)
56          docs = d.copy()
57      count_per_thread = 12500000
58      num_threads = 8
59
60      merged_results = generate_documents_parallel(
61          docs, count_per_thread, num_threads)
62
63      for i in range(math.ceil(count_per_thread*num_threads
            /5000000)):
64          with open(f'./big_data/out_{i}.json', 'w', encoding=
                'UTF-8') as file:
65              file.write(json.dumps(merged_results[i*5000000:(
                    i+1)*5000000]))
```

Listing A.11: leistungstests.py – Führt zehn Durchgänge eines Leistungstests mit einer ggb. Anzahl an Datensätzen und Hintergrundthreads durch

```
1  from datetime import datetime, timedelta
2  import json
3  import random
4  import sys
5  import threading
6  from typing import Literal
```

```
7  import requests
8
9  COLLECTION_NAME = 'BenchmarkTesting'
10 BACKGROUND_COLLECTION = 'Background'
11
12 COUNT = int(sys.argv[1])
13 BACKGROUND_THREADS = int(sys.argv[2])
14 SETS_PER_BACKGROUND_TASK = 100000
15 TIMES = 10
16
17
18 class DB:
19     @staticmethod
20     def get(path: str, collection: str = COLLECTION_NAME) ->
           requests.Response:
21         return requests.get(f'http://10.20.110.67:8529/_db/
               TestDB/_api/{path.replace("@", collection)}',
               timeout=600)
22
23     @staticmethod
24     def put(path: str, body: dict | list | None = None,
           collection: str = COLLECTION_NAME) -> requests.
           Response:
25         return requests.put(f'http://10.20.110.66:8529/_db/
               TestDB/_api/{path.replace("@", collection)}',
               json=body, timeout=600)
26
27     @staticmethod
28     def post(path: str, body: dict | list | None = None,
           collection: str = COLLECTION_NAME) -> requests.
           Response:
29         return requests.post(f'http://10.20.110.68:8529/_db/
               TestDB/_api/{path.replace("@", collection)}',
               json=body, timeout=600)
30
31     @staticmethod
32     def patch(path: str, body: dict | list | None = None,
           collection: str = COLLECTION_NAME) -> requests.
           Response:
33         return requests.patch(f'http://10.20.110.66:8529/_db
               /TestDB/_api/{path.replace("@", collection)}',
               json=body, timeout=600)
34
35     @staticmethod
```

```
36     def delete(path: str, body: dict | list | None = None,
           collection: str = COLLECTION_NAME) -> requests.
           Response:
37         return requests.delete(f'http://10.20.110.67:8529/
               _db/TestDB/_api/{path.replace("@", collection)}',
                json=body, timeout=600)
38
39
40 def truncate(collection: str = COLLECTION_NAME) -> None:
41     DB.put('collection/@/truncate', collection=collection)
42
43
44 def create(documents: list[dict], collection: str =
       COLLECTION_NAME) -> None:
45     DB.post(
46         'document/@#multiple?returnNew=false&returnOld=false
               &silent=true', body=documents, collection=
               collection)
47
48
49 def read(ids: list[str], collection: str = COLLECTION_NAME)
       -> None:
50     DB.put('document/@#get', body=ids, collection=collection
           )
51
52
53 def update(documents: list[dict], collection: str =
       COLLECTION_NAME) -> None:
54     DB.patch('document/@', body=documents, collection=
           collection)
55
56
57 def delete(ids: list[str], collection: str = COLLECTION_NAME
       ) -> None:
58     DB.delete('document/@', body=ids, collection=collection)
59
60
61 def get_keys(count: int, collection: str = COLLECTION_NAME)
       -> list[str]:
62     aql_query = f"""
63     FOR doc IN {collection}
64         RETURN doc._key
65     """
66
67     payload = {
68         'query': aql_query,
```

```
        'batchSize': count
    }

    return DB.post('/cursor', body=payload).json()['result']


def get_documents(count: int) -> list[dict]:
    if count > 10000000:
        ret: list[dict] = []
        j = 0
        while len(ret) < count and j < 20:
            with open(f'/home/team23w4/team23w4/benchmarks/big_data/out_{j}.json', 'r', encoding='UTF-8') as f:
                ret += json.load(f)
            ret = ret[:count]
            j += 1
        if len(ret) < count:
            print(f'Asked for {count} documents, but we have only {len(ret)}')
        return ret
    d: dict = None
    orig: dict = None
    with open('/home/team23w4/team23w4/Country-Sales-Data-GeneratorPlus03/Country-Sales-Data-GeneratorPlus03_1Mio.json', 'r', encoding='UTF-8') as f:
        d = json.load(f)
        orig = d.copy()
    while count > len(d):
        new_docs = orig.copy()
        for j, _ in enumerate(new_docs):
            new_docs[j]['Sales Channel'] = 'Online' if random.random(
            ) < 0.5 else 'Offline'
            new_docs[j]['Order Priority'] = random.choice(['L', 'M', 'H', 'C'])
            start_date = datetime(2010, 1, 1)
            end_date = datetime(2023, 12, 31)
            end_date2 = datetime(2024, 1, 6)
            order_date = start_date + \
                timedelta(days=random.randint(0, (end_date - start_date).days))
            ship_date = order_date + \
                timedelta(days=random.randint(
                    0, (end_date2 - order_date).days))
```

```
106                 new_docs[j]['Order␣Date'] = order_date.strftime('%Y-%m-%d')
107                 new_docs[j]['Ship␣Date'] = ship_date.strftime('%Y-%m-%d')
108                 new_docs[j]['Order␣ID'] = str(random.randint(100000000, 999999999))
109             seen_oids = set()
110             filtered_nd = []
111             for x in new_docs:
112                 oid = x['Order␣ID']
113                 if oid not in seen_oids:
114                     seen_oids.add(oid)
115                 filtered_nd.append(x)
116             d += filtered_nd
117         return d[:count]
118
119
120 def created_update_docs(documents: list[dict], ids: list[str]) -> list[dict]:
121         assert len(documents) == len(ids)
122         _documents = documents.copy()
123         random.shuffle(_documents)
124         ret = []
125         for doc, key in zip(_documents, ids):
126             new_doc = doc.copy()
127             new_doc['_key'] = key
128             ret.append(new_doc)
129         return ret
130
131
132 def background(b_docs: list[dict], exit_event: threading.Event):
133         while not exit_event.is_set():
134             action = random.randint(0, 3)
135             count = random.randint(500, 2000)
136             try:
137                 if action == 0:
138                     create([random.choice(b_docs)
139                             for _ in range(count)], BACKGROUND_COLLECTION)
140                 else:
141                     b_keys = get_keys(1000000, BACKGROUND_COLLECTION)
142                     if action == 1:
143                         read([random.choice(b_keys)
```

```
144                                 for _ in range(count)],
                                       BACKGROUND_COLLECTION)
145                     elif action == 2:
146                         u_docs = created_update_docs(
147                             [random.choice(b_docs) for _ in
                                   range(count)],
148                             [random.choice(b_keys) for _ in
                                   range(count)]
149                         )
150                         update(u_docs, BACKGROUND_COLLECTION)
151                     else:
152                         delete(b_keys, BACKGROUND_COLLECTION)
153             except IndexError:
154                 pass
155
156
157 if __name__ == '__main__':
158     truncate(BACKGROUND_COLLECTION)
159     docs = get_documents(COUNT*6)
160     thread_list = []
161     exit_signal = threading.Event()
162     if BACKGROUND_THREADS > 0:
163         background_docs = get_documents(
164             BACKGROUND_THREADS*SETS_PER_BACKGROUND_TASK)
165         init_b_docs = [random.choice(background_docs)
166                        for _ in range(int(BACKGROUND_THREADS
                              *SETS_PER_BACKGROUND_TASK/2))]
167         for t in range(BACKGROUND_THREADS):
168             create(init_b_docs[SETS_PER_BACKGROUND_TASK *
169                    t:SETS_PER_BACKGROUND_TASK*(t+1)],
                          BACKGROUND_COLLECTION)
170         for i in range(BACKGROUND_THREADS):
171             start_index = i * SETS_PER_BACKGROUND_TASK
172             end_index = (i + 1) * SETS_PER_BACKGROUND_TASK
173             thread = threading.Thread(target=background,
                   args=(
174                 background_docs[start_index:end_index],
                       exit_signal))
175             thread.daemon = True
176             thread.start()
177             thread_list.append(thread)
178
179     for t in range(TIMES):
180         random.shuffle(docs)
181         truncate()
182         for u in range(5):
```

```
183                 create(docs[COUNT*u:COUNT*(u+1)])
184             times: dict[
185                 Literal['create', 'read', 'update', 'delete'],
186                 dict[
187                     Literal['start', 'end'],
188                     datetime
189                 ]
190             ] = {
191                 'create': {'start': datetime.now(), 'end':
                        datetime.now()},
192                 'read': {'start': datetime.now(), 'end':
                        datetime.now()},
193                 'update': {'start': datetime.now(), 'end':
                        datetime.now()},
194                 'delete': {'start': datetime.now(), 'end':
                        datetime.now()}
195             }
196
197             times['create']['start'] = datetime.now()
198             create(docs[COUNT*5:])
199             times['create']['end'] = datetime.now()
200
201             keys = get_keys(COUNT*6)
202             update_docs = created_update_docs(
203                 [random.choice(docs) for _ in range(COUNT)], [
                        random.choice(keys) for _ in range(COUNT)])
204
205             times['read']['start'] = datetime.now()
206             read([random.choice(keys) for _ in range(COUNT)])
207             times['read']['end'] = datetime.now()
208
209             times['update']['start'] = datetime.now()
210             update(update_docs)
211             times['update']['end'] = datetime.now()
212
213             times['delete']['start'] = datetime.now()
214             delete([random.choice(keys) for _ in range(COUNT)])
215             times['delete']['end'] = datetime.now()
216
217             for o in ['create', 'read', 'update', 'delete']:
218                 print(f'{o}: {times[o]["end"] - times[o]["start
                        "]}')
219             print()
220
221         if BACKGROUND_THREADS > 0:
222             exit_signal.set()
```

```
223         for thread in thread_list:
224             thread.join()
```

A.5 Leistungstestauswertungen

Tabelle A.1: Ergebnisse der Tests zur Leistungsbewertung.
Zeiten in Minuten:Sekunden.

Rep.	Threads	Sätze	Creat	Read	Update	Delete
1	0	1	00:00,002	00:00,003	00:00,003	00:00,003
		1 0000	00:00,081	00:00,015	00:00,129	00:00,030
		1 000 000	00:29,631	00:03,392	01:03,984	00:25,096
	20	1	00:00,009	00:00,009	00:00,009	00:00,009
		1 0000	00:00,078	00:00,016	00:00,117	00:00,045
		1 000 000	01:01,104	00:05,331	02:30,007	01:19,077
	50	1	00:00,070	00:00,067	00:00,047	00:00,046
		1 0000	00:00,141	00:00,023	00:00,173	00:00,066
		1 000 000	00:59,622	00:05,329	02:17,685	01:12,519
	100	1	00:00,025	00:00,023	00:00,024	00:00,023
		1 0000	00:00,159	00:00,041	00:00,246	00:00,100
		1 000 000	01:03,575	00:06,861	02:15,342	01:06,057
2	0	1	00:00,004	00:00,003	00:00,004	00:00,004
		1 0000	00:00,133	00:00,015	00:00,215	00:00,066
		1 000 000	00:49,597	00:03,364	01:44,162	00:44,977
	20	1	00:00,009	00:00,008	00:00,009	00:00,008
		1 0000	00:00,129	00:00,016	00:00,226	00:00,063
		1 000 000	01:26,907	00:05,109	03:22,294	01:45,628
	50	1	00:00,027	00:00,017	00:00,019	00:00,022
		1 0000	00:00,168	00:00,015	00:00,199	00:00,078
		1 000 000	01:23,107	00:04,956	02:56,135	01:24,582
	100	1	00:00,048	00:00,041	00:00,046	00:00,045
		1 0000	00:00,175	00:00,029	00:00,238	00:00,077
		1 000 000	01:28,430	00:10,268	03:04,472	01:25,855
3	0	1	00:00,006	00:00,006	00:00,007	00:00,007
		1 0000	00:00,155	00:00,016	00:00,227	00:00,092
		1 000 000	00:55,080	00:03,385	01:48,284	00:47,211
	20	1	00:00,009	00:00,008	00:00,009	00:00,010
		1 0000	00:00,131	00:00,019	00:00,181	00:00,060
		1 000 000	01:34,077	00:04,984	03:25,546	01:42,547
	50	1	00:00,025	00:00,014	00:00,025	00:00,021
		1 0000	00:00,200	00:00,022	00:00,255	00:00,087
		1 000 000	01:16,909	00:05,606	03:02,481	01:26,679
	100	1	00:00,046	00:00,043	00:00,042	00:00,046
		1 0000	00:00,196	00:00,027	00:00,250	00:00,094
		1 000 000	01:27,393	00:06,191	02:59,167	01:19,214

Tabelle A.2: Ergebnisse der Tests zur Leistungsbewertung.
Zeiten in Minuten:Sekunden.

Shrds.	Threads	Sätze	Creat	Read	Update	Delete
1	0	1	00:00,002	00:00,003	00:00,003	00:00,003
		1 0000	00:00,081	00:00,015	00:00,129	00:00,030
		1 000 000	00:29,631	00:03,392	01:03,984	00:25,096
	20	1	00:00,009	00:00,009	00:00,009	00:00,009
		1 0000	00:00,078	00:00,016	00:00,117	00:00,045
		1 000 000	01:01,104	00:05,331	02:30,007	01:19,077
	50	1	00:00,070	00:00,067	00:00,047	00:00,046
		1 0000	00:00,141	00:00,023	00:00,173	00:00,066
		1 000 000	00:59,622	00:05,329	02:17,685	01:12,519
	100	1	00:00,025	00:00,023	00:00,024	00:00,023
		1 0000	00:00,159	00:00,041	00:00,246	00:00,100
		1 000 000	01:03,575	00:06,861	02:15,342	01:06,057
2	0	1	00:00,005	00:00,005	00:00,006	00:00,005
		1 0000	00:00,082	00:00,015	00:00,102	00:00,032
		1 000 000	00:25,713	00:02,915	00:38,393	00:13,627
	20	1	00:00,009	00:00,006	00:00,005	00:00,009
		1 0000	00:00,107	00:00,017	00:00,090	00:00,042
		1 000 000	00:32,152	00:08,013	00:46,642	00:19,540
	50	1	00:00,012	00:00,009	00:00,012	00:00,015
		1 0000	00:00,110	00:00,069	00:00,312	00:00,102
		1 000 000	00:30,898	00:07,500	00:46,624	00:19,899
	100	1	00:00,008	00:00,007	00:00,008	00:00,008
		1 0000	00:00,136	00:00,158	00:00,630	00:00,211
		1 000 000	00:32,176	00:05,940	00:54,032	00:18,311
3	0	1	00:00,003	00:00,004	00:00,004	00:00,003
		1 0000	00:00,062	00:00,014	00:00,086	00:00,026
		1 000 000	00:24,430	00:02,923	00:31,799	00:09,335
	20	1	00:00,009	00:00,007	00:00,007	00:00,008
		1 0000	00:00,151	00:00,016	00:00,143	00:00,033
		1 000 000	00:33,040	00:08,038	00:45,127	00:19,667
	50	1	00:00,023	00:00,015	00:00,013	00:00,017
		1 0000	00:00,106	00:00,019	00:00,084	00:00,028
		1 000 000	00:43,438	00:05,080	00:52,395	00:23,482
	100	1	00:00,052	00:00,016	00:00,008	00:00,006
		1 0000	00:00,134	00:00,033	00:00,113	00:00,052
		1 000 000	00:39,881	00:06,715	01:15,173	00:24,991

A.6 Verfügbarkeiten

Listing A.12: example.json – Ein Beispieldatensatz, der zur Verfügung gestellten Daten

```
1  {
2      "Region": "Europe",
3      "Country": "San␣Marino",
4      "Item␣Type": "Clothes",
5      "Sales␣Channel": "Online",
6      "Order␣Priority": "M",
7      "Order␣Date": "2014-09-12",
8      "Order␣ID": "247729957",
9      "Ship␣Date": "2014-10-12",
10     "Units␣Sold": "7654",
11     "Unit␣Price": "192,47",
12     "Unit␣Cost": "176,61",
13     "Total␣Revenue": "1473165,38",
14     "Total␣Cost": "1351772,94",
15     "Total␣Profit": "121392,44",
16     "ItemID": "EHUNGCL00069ES",
17     "Supplier": "H&M-ESP",
18     "SuplierRegion": "Europe",
19     "SuplierCountry": "Spain"
20 }
```

Listing A.13: lscpu eines Servers

```
1  $ lscpu
2  Architecture:              x86_64
3    CPU op-mode(s):          32-bit, 64-bit
4    Address sizes:           39 bits physical, 48 bits virtual
5    Byte Order:              Little Endian
6  CPU(s):                    8
7    On-line CPU(s) list:     0-7
8  Vendor ID:                 GenuineIntel
9    Model name:              Intel(R) Core(TM) i7-4790K CPU @
       4.00GHz
10     CPU family:            6
11     Model:                 60
12     Thread(s) per core:    2
13     Core(s) per socket:    4
14     Socket(s):             1
15     Stepping:              3
16     CPU max MHz:           4400,0000
17     CPU min MHz:           800,0000
18     BogoMIPS:              7981.46
```

```
19      Flags:                fpu vme de pse tsc msr pae mce cx8
           apic sep mtrr pge mca cmov pat pse36 clflush dts acpi
            mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
           rdtscp lm constant_tsc arch_perfmon pebs bts rep_good
            nopl xtopology nonstop_tsc cpuid aperfmperf pni
           pclmulqdq d
20                            tes64 monitor ds_cpl vmx est tm2
                                 ssse3 sdbg fma cx16 xtpr pdcm
                                 pcid sse4_1 sse4_2 x2apic movbe
                                 popcnt tsc_deadline_timer aes
                                 xsave avx f16c rdrand lahf_lm
                                 abm cpuid_fault invpcid_single
                                 pti ssbd ibrs ibpb stibp
                                 tpr_shadow vnmi flexpriority ept
                                  vpid ept
21                            _ad fsgsbase tsc_adjust bmi1 avx2
                                 smep bmi2 erms invpcid xsaveopt
                                 dtherm ida arat pln pts md_clear
                                  flush_l1d
22 Virtualization features:
23   Virtualization:          VT-x
24 Caches (sum of all):
25   L1d:                     128 KiB (4 instances)
26   L1i:                     128 KiB (4 instances)
27   L2:                      1 MiB (4 instances)
28   L3:                      8 MiB (1 instance)
29 NUMA:
30   NUMA node(s):            1
31   NUMA node0 CPU(s):       0-7
32 Vulnerabilities:
33   Gather data sampling:    Not affected
34   Itlb multihit:           KVM: Mitigation: VMX disabled
35   L1tf:                    Mitigation; PTE Inversion; VMX
        conditional cache flushes, SMT vulnerable
36   Mds:                     Mitigation; Clear CPU buffers; SMT
        vulnerable
37   Meltdown:                Mitigation; PTI
38   Mmio stale data:         Unknown: No mitigations
39   Retbleed:                Not affected
40   Spec rstack overflow:    Not affected
41   Spec store bypass:       Mitigation; Speculative Store
        Bypass disabled via prctl and seccomp
42   Spectre v1:              Mitigation; usercopy/swapgs
        barriers and __user pointer sanitization
```

```
Spectre v2:             Mitigation; Retpolines, IBPB
    conditional, IBRS_FW, STIBP conditional, RSB filling,
    PBRSB-eIBRS Not affected
Srbds:                  Mitigation; Microcode
Tsx async abort:        Not affected
```

Abbildungsverzeichnis

1.1 Architektur eines Data-Mesh [CVH22] 3

3.1 Graph Struktur innerhalb von ArangoDB 7
3.2 Physisches Schema der Beispieldaten 12

4.1 Alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe eins 17
4.2 Alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe eins ohne größten Ausreißer 17
4.3 Alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe Eintausend 18
4.4 Alle Laufzeiten für die Create-Operation mit einem Datensatz der Größe einer Millionen 18
4.5 Alle Laufzeiten für die Read-Operation mit einem Datensatz der Größe einer Millionen 19
4.6 Medianlaufzeiten der CRUD-Operationen für Datensätze der Größe eins 20
4.7 Medianlaufzeiten von Create-Operationen für verschiedene Datensatzgrößen 21
4.8 Medianlaufzeiten aller Skalierungsarten mit Datensätzen der Größe eine Millionen 21
4.9 Komponentendiagramm der Anwendung 23
4.10 Sequenzdiagramm der Kommunikation bei einem Aufruf der beide Domänen einschließt 25
4.11 Ausschnitte der Anwendung im Kleinformat 27

A.1 Logisches Schema der importierten Daten 31
A.2 Mockup des Logins 32
A.3 Mockup der Anwendung 32
A.4 Login-Seite 33
A.5 Datenanalyse-Seite 34
A.6 Datenansichts-Seite 35
A.7 Datenansichts-Seite nach Update 36

Tabellenverzeichnis

2.1 Schnellübersicht über das DBS ArangoDB [Ara23g, Ig24] 4

3.1 Beispiel zur Serialsierung mit VelocyPack 8

A.1 Ergebnisse der Tests zur Leistungsbewertung 73
A.2 Ergebnisse der Tests zur Leistungsbewertung 74

Listings

3.1 Einfache AQL-Anfrage 9
3.2 Einfache SQL-Anfrage 9
3.4 Create AQL-Anfrage 12
3.3 modify Data 13
3.5 Read AQL-Anfrage 13
3.6 Update AQL-Anfrage 13
3.7 Delete AQL-Anfrage 14

A.1 subCsv.py 37
A.2 prepare_items.py 38
A.3 preapre_suppliers.py 39
A.4 create_item_list.py 40
A.5 create-db-json.sh 41
A.6 csv2json.py 41
A.7 order-openapi.json 44
A.8 supplier-openapi.json 49
A.9 run-arango.sh 63
A.10 create_data.py 64
A.11 leistungstests.py 65
A.12 example.json 75
A.13 lscpu 75

Abkürzungsverzeichnis

ACID	Atomicity (Atomizität), Consistency (Konsistenz), Isolation (Isolation) und Durability (Dauerhaftigkeit)
API	Application Programming Interface
AQL	ArangoDB Query Language
CRUD	Create, Read, Update, Delete
CSV	Comma Seperated Values
GQL	Graph Query Language
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
SQL	Structured Query Language
DB	Datenbank
DBS	Datenbanksystem
SQL	Structured Query Language
NoSQL	Not only SQL
Blob	Binary large object
REST	Representational State Transfer
US	Userstory

Literaturverzeichnis

[Ara23a] ArangoDB: *Advantaes of native multi-model in ArangoDB - ArangoDB*. https://arangodb.com/native-multi-model-database-advantages/. Version: 2023

[Ara23b] ArangoDB: *AQL Documentation*. https://docs.arangodb.com/3.11/aql/. Version: 2023

[Ara23c] ArangoDB: *arangosh*. https://docs.arangodb.com/3.11/components/tools/arangodb-shell/. Version: 2023

[Ara23d] ArangoDB: *Download & Install the Latest Release*. https://arangodb.com/download-major/. Version: 2023

[Ara23e] ArangoDB: *HTTP API Documentation*. https://docs.arangodb.com/3.11/develop/http-api/. Version: 2023

[Ara23f] ArangoDB: *Integrations*. https://docs.arangodb.com/3.11/develop/integrations/. Version: 2023

[Ara23g] ArangoDB: *Multi Model - ArangoDB*. https://arangodb.com/multi-model/. Version: 2023

[Ara23h] ArangoDB: *Official ArangoDB drivers*. https://docs.arangodb.com/3.11/develop/drivers/. Version: 2023

[Ara23i] ArangoDB: *The scalability of ArangoDB and its data models*. https://docs.arangodb.com/3.11/deploy/architecture/scalability/. Version: 2023

[Ara23j] ArangoDB: *Web Interface*. https://docs.arangodb.com/3.11/components/web-interface/. Version: 2023

[Ara24] ArangoDB: *VelocyPack*. https://github.com/arangodb/velocypack/blob/main/Performance.md. Version: 2024

[BKJ23] Belgundi, Rajat ; Kulkarni, Yash ; Jagdale, Balaso: Analysis of Native Multi-model Database Using ArangoDB. In: Shakya, Subarna (Hrsg.) ; Balas, Valentina E. (Hrsg.) ; Haoxiang, Wang (Hrsg.): *Proceedings of Third International Conference on Sustainable Expert Systems*. Singapore : Springer Nature Singapore, 2023. – ISBN 978–981–19–7874–6, S. 923–935

[BS24] Balsamiq Studios, LLC: *Balsamiq*. https://balsamiq.com/. Version: 2024

[CGK15] Chickerur, Satyadhyan ; Goudar, Anoop ; Kinnerkar, Ankita: Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications. In: *2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA)*, 2015, S. 41–47

[Cun20] Cunningham, Justin: *Netflix Data Mesh: Composable Data Processing - Justin Cunningham*. https://www.youtube.com/watch?v=TO_IiN06jJ4. Version: April 2020

[CVH22] Christ, Jochen ; Visengeriyeva, Larysa ; Harrier, Simon: *Data Mesh Architectrue: Data Mesh from an engineering perspective*. https://www.datamesh-architecture.com/, 2022. – Accessed: 23.10.2023

[CVH23] Christ, Jochen ; Visengeriyeva, Larysa ; Harrier, Simon: *Das Daten- und KI-Unternehmen – Databricks*. https://www.datamesh-manager.com/?ref=dma-nav. Version: 2023

[Dat23] Databricks: *Das Daten- und KI-Unternehmen – Databricks*. https://www.databricks.com/de. Version: 2023

[Deh22] Dehghani, Zhamak: *Data Mesh*. O'Reilly Media, Inc., 2022. – 384 S. https://learning.oreilly.com/library/view/data-mesh/9781492092384/. – ISBN 9781492092384

[Ig24] IT gmbh solid: *ArangoDB Systemeigenschaften*. https://db-engines.com/de/system/ArangoDB. Version: 2024

[Ove23] Overflow, Stack: *Most used programming languages among developers worldwide as of 2023*. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. Version: 2023

[RWE15] Robinson, Ian ; Webber, Jim ; Eifrem, Emil: *Graph Databases, 2nd Edition*. O'Reilly Media, Inc., 2015 https://www.oreilly.com/library/view/graph-databases-2nd/9781491930885/. – ISBN 9781491930892

[VHM+22] Vestues, Kathrine ; Hanssen, Geir K. ; Mikalsen, Marius ; Buan, Thor A. ; Conboy, Kieran: Agile Data Management in NAV: A Case Study. In: Stray, Viktoria (Hrsg.) ; Stol, Klaas-Jan (Hrsg.) ; Paasivaara, Maria (Hrsg.) ; Kruchten, Philippe (Hrsg.): *Agile Processes in Software Engineering and Extreme Programming*. Cham : Springer International Publishing, 2022. – ISBN 978–3–031–08169–9, S. 220–235