Elasticsearch

Document-Based Databases

Presented by: Roland Fornvald, Jonas Hinterdorfer, Zsombor Matyas, David Naderer, Simon Wögerbauer

Ablauf

- Was ist Elasticsearch & Document-Based Databases?
- Architektur und Funktionsweise
- Besonderheiten
- Datenmodellierung
- Daten-Persistierung & Mehrbenutzerbetrieb & Transaktionen
- Vergleich zu relationalen Datenbanken

Was sind Document-Based Databases?

Definition: Klasse von NoSQL-Datenbanken, die Daten als selbstbeschreibende, hierarchische Dokumente (typischerweise JSON) speichern.

Unterschiede zu relationalen DBs:

- Keine feste Tabellenstruktur
- Verschachtelte, komplexe Datenstrukturen natürlich möglich
- Redundanz durch Denormalisierung
- Schema ist optional oder flexibel

Document-Based DBs

DB	Fokus	Use Case
MongoDB	Dokumenten-Management	CRUD, Business Apps
Firebase/Firestore	Cloud, Real-time Sync	Mobile Apps, Startups
Elasticsearch	Suche & Analytics	Logs, Full-Text Search

Elasticsearch

Elasticsearch ist eine spezialisierte Document-Based Database für Suche und Analytics:

- **Distributed Architecture:** Daten über mehrere Knoten verteilt
- **Inverted Index:** Grundlage für schnelle Textsuche
- **RESTful API:** Alle Operationen über HTTP
- **Real-time:** Daten werden sofort indexiert und durchsuchbar

Elasticsearch

Kernstärken:

- **Optimiert für Suche, nicht für Transaktionen**
- Ranking-Algorithmen für Relevanz
- Aggregations-Framework für Analytics
- Time-Series Datenanalyse (Metrics, Logs)

ELK-Stack Ökosystem:

- **E**lasticsearch: Storage & Search
- **L**ogstash: Pipeline für Log-Verarbeitung
- **K**ibana: Visualisierung & Exploration

Cluster

- Sammlung von Elasticsearch-Instanzen (Nodes)
- Koordiniert über Master-Node
- Daten konsistent über alle Knoten verteilt

Shards

- Verteilung eines Index auf mehrere Partitionen
- **Primary Shards:** Original-Kopie
- **Replica Shards:** Redundante Kopien für Ausfallsicherheit
- Horizontal skalierbar: mehr Knoten = mehr Shards möglich

Replication:

- Asynchrone Replikation zwischen Primary und Replicas
- Erhöht Verfügbarkeit und Leseleistung
- Trade-off: Eventually Consistent statt Strong Consistent

Inverted Index

Normaler Index (B-Tree):

```
Dokument → Wörter
```

Inverted Index (Elasticsearch):

```
Wort → Dokumente
"elasticsearch" → [Doc1, Doc5, Doc23, ...]

"datenbank" → [Doc2, Doc4, Doc5, ...]
```

Text Processing Pipeline

1. Tokenization
2. Lowercasing
3. Stop Words
4. Lemmatization:

Analyzer-Typen:

- Standard Analyzer: Default für die meisten Fälle
- Language Analyzers: Sprach-spezifisch (German, English)
- Custom Analyzers: Benutzerdefiniert konfigurierbar

Datenmodellierung - Index

Index:

- Logische Sammlung von Dokumenten (≈ Tabelle)
- Ein Index ≈ Ein Datentyp oder Business-Entity
- Separate Indizes für verschiedene Datentypen üblich

Datenmodellierung - Mapping

- Schema-Definition: Feldtypen, Analyzer, Indexing-Verhalten
- **Explizit:** Vor Dateneinfügen definieren
- **Implizit:** Elasticsearch errät Feldtypen (nicht empfohlen)

Beziehungen

Problem: Elasticsearch hat keine nativen Joins

Mögliche Lösungen: Denormalisierung, Nested Objects oder Parent-Child

Denormalisierung

```
{
  "title": "Artikel",
  "author": {
    "name": "Max",
    "email": "max@example.de"
  }
}
```

- Schnell, einfach
- Redundanz bei vielen Beziehungen

Nested Objects

```
{
  "title": "Artikel",
  "comments": [
    { "author": "Max", "text": "Gut" }
  ]
}
```

- Für 1:N Beziehungen
- Wird als separates Dokument indexiert

Parent-Child

- Für große N:M Beziehungen
- Eltern- und Kind-Dokumente getrennt gespeichert
- Queries über has_child / has_parent
- Eltern- und Kind-Dokumente müssen auf derselben Shard liegen
- Nur in speziellen Fällen empfohlen

Daten-Persistierung

Schreibprozess:

1. Dokument in Memory Buffer (In-Memory Index)
2. Refresh (Standard: alle 1 Sekunde) → Searchable
3. Flush → Transaktionslog geleert, Segment persistent
4. Merge → Segments optimieren

Durability - Konsistenz

- **Transaktionslog:** Jede Operation geloggt, wiederherstellung möglich
- **Replikation:** Primary Shard → Replica Shards
- Updates erst bestätigt, wenn Primary + mind. 1 Replica wird geupdated

Konsistenz:

- Write-Konsistenz: `quorum` oder `all` Replicas
- Read-Konsistenz: Reads von veraltete Replicas möglich

Mehrbenutzerbetrieb

- Jedes Dokument hat Versionsnummer (`_seq_no` , `_primary_term`)
- Bei Update: Client sendet erwartete Version
- Conflict → 409 Status, Client muss Merge lösen
- Kein LOCK/UNLOCK Mechanismus

Transaktionen

Elasticsearch Transaktionen:

- Seit 7.0: Begrenzte Multi-Document Transaktionen
- Atomicity: nur innerhalb eines Shards möglich
- Rollbacks sind nicht möglich
- Isolation: Multi-Dokument-Transaktionen sehen teilweise Updates
- Durability: Ja, nach Flush

CAP-Theorem

Eigenschaft	Wert	Grund
Partition Tolerance	Ja	Distribuiert, Fehlerbehandlung
Availability	Ja	Replica Shards ermöglichen Reads
Consistency	Eventual	Asynchrone Replikation

Elasticsearch vs. Relationale Datenbanken

Datenmodell:

- SQL: Starre normalisierte Schemas, Tabellen
- Elasticsearch: Flexible denormalisierte Dokumente

Suchanfragen:

- SQL: WHERE mit Indexes, Full-Text komplex
- Elasticsearch: Native Volltext-Suche, Relevance Ranking

Joins:

- SQL: Standard, optimiert
- Elasticsearch: Nativ unmöglich

Transaktionen:

- SQL: ACID auf mehrere Tabellen
- Elasticsearch: Begrenzt, nur Dokument-Ebene

Skalierung:

- SQL: Vertical (Scale-Up) bevorzugt
- Elasticsearch: Horizontal (Scale-Out) optimal

Vielen Dank

Fragen?