Neo4j

Introduction to Neo4j

- Graph database system supporting both transactional and analytical processing of graph-based data
- Part of the NoSQL database family, but specialized for connected data
- Schema optional design (can impose schema if needed)
- Supports various indexing methods for performance
- ACID compliant for data integrity and reliability
- Supports distributed computing environments
- Similar graph databases include Microsoft CosmoDB and Amazon Neptune

Neo4j Query Language and Extensions

Cypher Query Language

- Developed in 2011 as a dedicated graph query language
- Goal: SQL-equivalent language specifically for graph databases
- Visual pattern-matching syntax for relationships: `(nodes)-[:CONNECTS_TO]->(otherNodes)`

Extensions and Plugins

- **APOC Plugin**: Awesome Procedures on Cypher
  - Add-on library with hundreds of procedures and functions
  - Extends Cypher capabilities
- **Graph Data Science Plugin**:
  - Efficient implementations of graph algorithms
  - Centrality, pathfinding, community detection algorithms

Neo4j Deployment with Docker

Docker Compose Setup

- Manages multi-container applications via declarative YAML files
- Consistent environment across systems
- Single command management (start, stop, scale)
- Eliminates "works on my machine" problems

Docker Compose Configuration

```
services:
  neo4j:
    container_name: neo4j
    image: neo4j:latest
    ports:
      - 7474:7474
      - 7687:7687
    environment:
      - NEO4J_AUTH=neo4j/${NEO4J_PASSWORD}
      - NEO4J_apoc_export_file_enabled=true
      - NEO4J_apoc_import_file_enabled=true
      - NEO4J_apoc_import_file_use__neo4j__config=true
      - NEO4J_PLUGINS=["apoc", "graph-data-science"]
    volumes:
      - ./neo4j_db/data:/data
      - ./neo4j_db/logs:/logs
      - ./neo4j_db/import:/var/lib/neo4j/import
      - ./neo4j_db/plugins:/plugins
```

Environment Variables

- `.env` files store environment variables securely
- Separates configuration from implementation
- Can have different files for environments (local, dev, prod)
- Example: `NEO4J_PASSWORD=abc123!!!` in `.env` file
- Security best practice: Never put secrets directly in docker-compose.yaml

Essential Docker Commands

- `docker --version`: Check Docker installation
- `docker compose up`: Start containers
- `docker compose up -d`: Start in detached mode
- `docker compose down`: Stop and remove containers
- `docker compose start/stop`: Start/stop without removing
- `docker compose build`: Build/rebuild services
- `docker compose build --no-cache`: Force rebuild from scratch

Neo4j Browser Interface

- Access via `localhost:7474` after deployment

- Components include:
    - Cypher editor for queries
    - Result frames with multiple view options
    - Database connection manager
    - Sidebar with favorites, guides, and settings
    - Visualization controls and property display

Working with Data in Neo4j

Creating Nodes

```
CREATE (:User {name: "Alice", birthPlace: "Paris"})
CREATE (:User {name: "Bob", birthPlace: "London"})
CREATE (:User {name: "Carol", birthPlace: "London"})
CREATE (:User {name: "Dave", birthPlace: "London"})
CREATE (:User {name: "Eve", birthPlace: "Rome"})
```

Creating Relationships

```
MATCH (alice:User {name:"Alice"})
MATCH (bob:User {name: "Bob"})
CREATE (alice)-[:KNOWS {since: "2022-12-01"}]->(bob)
```

Note: Relationships in Neo4j are directed

Querying Data

```
MATCH (usr:User {birthPlace: "London"})
RETURN usr.name, usr.birthPlace
```

Importing Data from CSV

Basic Import Structure:

```
LOAD CSV
[WITH HEADERS]
FROM 'file:///file_in_import_folder.csv'
AS line
[FIELDTERMINATOR ',']
// operations with 'line'
```

Import Example (creating movie nodes):

```
LOAD CSV WITH HEADERS
FROM 'file:///netflix_titles.csv' AS line
CREATE(:Movie {
  id: line.show_id,
  title: line.title,
  releaseYear: line.release_year
  }
)
```

Working with Lists and Relationships

Handling lists in imported data:

```
LOAD CSV WITH HEADERS
FROM 'file:///netflix_titles.csv' AS line
WITH split(line.director, ",") as directors_list
UNWIND directors_list AS director_name
MERGE (:Person {name: trim(director_name)})
```

Creating relationships between nodes:

```
LOAD CSV WITH HEADERS
FROM 'file:///netflix_titles.csv' AS line
MATCH (m:Movie {id: line.show_id})
WITH m, split(line.director, ",") as directors_list
UNWIND directors_list AS director_name
MATCH (p:Person {name: director_name})
CREATE (p)-[:DIRECTED]->(m)
```

Verifying Data

```
MATCH (m:Movie {title: "Ray"})<-[:DIRECTED]-(p:Person)
RETURN m, p
```

Working with External Datasets

Setup process:

1. Clone repository: github.com/PacktPublishing/Graph-Data-Science-with-Neo4j
2. Locate and unzip netflix.zip from Chapter02/data
3. Copy netflix_titles.csv to the import folder (neo4j_db/neo4j_db/import)

4. Use LOAD CSV commands to import data into Neo4j

Key Neo4j Concepts

- **Nodes**: Represent entities with labels and properties
- **Relationships**: Connect nodes with types and optional properties
- **Properties**: Key-value pairs stored on nodes and relationships
- **Labels**: Group nodes into sets
- **Cypher**: Declarative query language specific to graph operations
- **Indexes**: Improve query performance on frequently searched properties
- **Constraints**: Enforce data integrity rules

Advanced Neo4j Features

- Pattern matching for complex queries
- Support for graph algorithms
- Built-in visualization capabilities
- Support for user-defined procedures
- Transaction management
- Query optimization tools
- Advanced data import/export capabilities