Neo4j

Slide Visual Description (cover image)

- The slide features a simple title layout with a white background.
- Text reads: “Neo4j” in large font.
- The instructor’s name and course code (“DS 4300”) appear near the top or center.
- No additional graphics, just text.

Overview / Context

This slide introduces the day’s topic: working with the Neo4j graph database. It shows the course number (DS 4300), the instructor (Mark Fontenot, PhD), and the specific project or lesson (Market Forecast Project, Day 11).

Slide 2: Neo4j

Key Points

- **A Graph Database System** that supports both transactional and analytical processing.
- **Graph-based data model** (nodes, relationships, properties).
- **ACID transactions** (so constraints can be imposed).
    - *ACID* stands for Atomicity, Consistency, Isolation, Durability.
- Possibly **the best-known graph DB** in industry.
- Other similar technologies: **ArangoDB**, **Amazon Neptune**, etc.

Explanation / Expansion

Neo4j is a popular choice for applications where relationships between data points are as critical as the data itself. Because it is ACID-compliant, Neo4j can be used in production environments requiring strict consistency. Its graph-based nature makes it ideal for queries that are difficult to model in a traditional relational database (e.g., social networks, recommendation engines, knowledge graphs).

Slide Visual Description

- White background with bullet points summarizing Neo4j’s key features.
- Title “Neo4j” in bold.
- Bulleted list of features.

Slide 3: Neo4j – Query Language and Plugins

Key Points

- **Cypher**: Neo4j's query language, created in 2011.
- **APOC**: "Awesome Procedures On Cypher."
  - A library of user-defined procedures and functions.
  - Example usage: CALL apoc.help('csv')
  - Allows advanced transformations and extended functionality.
- APOC is **optional** (plugin you can install).

Explanation / Expansion

- **Cypher** is the SQL-like language for Neo4j but is more intuitive for graph operations.
- **APOC** extends Cypher by providing a wide range of helper procedures. For instance, you can easily import/export data, manipulate data structures, or generate random data for testing.

Slide Visual Description

- Title "Neo4j – Query Language and Plugins" at the top.
- Bullet points describing Cypher and APOC.
- Possibly a screenshot snippet or reference to APOC commands (like CALL apoc.help('csv')).

Slide 4: Neo4j in Docker Compose

Key Point

This slide likely serves as a transition to the idea of running Neo4j inside Docker containers. Running databases in containers is common for portability and easy environment setup.

Explanation / Expansion

- You can run Neo4j in Docker as a single container.
- To manage multiple services (e.g., a web app + Neo4j), Docker Compose is typically used.
- Docker Compose can define services, networks, and volumes in one YAML file.

Slide Visual Description

- Title: “Neo4j in Docker Compose.”
- The slide may show a brief list or a single statement that Docker Compose will be used to spin up a Neo4j container.

———

Slide 5: Docker Compose

Key Points

- **Supports multi-container management**: You can define multiple services in one file.
- **File is declarative**: Typically docker-compose.yaml or docker-compose.yml.
- **Cross-platform**: Works on Windows, macOS, Linux.
- **Commands**: docker-compose up -d, docker-compose down, etc.

Explanation / Expansion

- Docker Compose allows you to describe an application’s services (e.g., a database, a web front-end, a backend API) in a single YAML file.
- By running docker-compose up -d, all containers defined in the file are created and started.
- Docker Compose is particularly useful in development environments to replicate a production setup on a local machine.

Slide Visual Description

- A bullet list explaining what Docker Compose is and why it’s used.
- Possibly an example snippet showing the structure of a docker-compose.yaml.

———

Slide 6: docker-compose.yaml

Key Points / Example Snippet
A typical docker-compose.yaml might include:

```
version: '3.8'
services:
  neo4j:
    image: neo4j:latest
    container_name: neo4j
    env_file:
      - .env
```

```
    ports:
      - "7474:7474"
      - "7687:7687"
    volumes:
      - neo4j_data:/data
    environment:
      - NEO4J_AUTH=neo4j/your_password
volumes:
  neo4j_data:
```

Explanation / Expansion

- **version**: The Compose file format version.
- **services**: Each service is a container. Here, only one named neo4j.
- **env_file**: References an external .env file that holds environment variables.
- **ports**: Maps container ports to host ports (7474 for Neo4j's web interface, 7687 for the Bolt protocol).
- **volumes**: Persists data outside the container's ephemeral filesystem.

Slide Visual Description

- A screenshot or text snippet of the YAML configuration.
- Possibly highlights the env_file line or the port mapping lines.

———

Slide 7: .env Files

Key Points

- .env files store a collection of environment variables.
- Good way to keep environment-specific variables separate (e.g., passwords, hostnames, etc.).
- Example snippet:

```
NEO4J_AUTH=neo4j/somepassword
```

Explanation / Expansion

- .env files allow you to avoid hardcoding sensitive or environment-specific details into the docker-compose.yaml.
- The Compose file references these variables with syntax like ${VARIABLE_NAME} or via env_file:.

Slide Visual Description

- The slide shows a small snippet of .env content:
  - Possibly NEO4J_AUTH=neo4j=abc123!!! as an example.

Slide 8: Docker Compose Commands

Key Points

- To test if you have Docker CLI properly installed, run:

```
docker --version
docker-compose --version
```

- Common Docker Compose commands:
  - docker-compose up -d
  - docker-compose down
  - docker-compose build --no-cache

Explanation / Expansion

- docker-compose up -d: Builds (if needed) and starts containers in the background (detached mode).
- docker-compose down: Stops and removes containers and networks defined in the Compose file.
- docker-compose build --no-cache: Rebuilds images without using any cached layers.

Slide Visual Description

- Bullet list of commands, with a black terminal screenshot or code block style.
- Possibly a note that you should see a version output if Docker is installed correctly.

Slide 9: localhost:7474

Key Point

Once Neo4j is running in Docker, you can access the Neo4j Browser at http://localhost:7474.

Explanation / Expansion

- Port 7474 is the default for the Neo4j web interface.
- Another port (7687) is used by the Bolt protocol (used by drivers and some query tools).

Slide Visual Description

- Likely a screenshot of a browser window pointing to localhost:7474, prompting for a username and password.

Slide 10: Neo4j Browser (Login Screen and Interface)

Title/Text

"localhost:7474 Then login."

Screenshot Description

- The screenshot shows the Neo4j Browser interface.
- There is a login prompt, typically requiring a username (neo4j) and a password (the one set in NEO4J_AUTH).

Additional Explanation

- After logging in, you will see the Neo4j Browser, which has:
  - A **Sidebar** (left) with Database, Favorites, Guides.
  - A **Query editor** or "Run query" bar at the top.
  - **Result frames** below the query editor showing node-relationship visualizations and data tables.
  - Ability to **Export** results, **Collapse** frames, or **Save** as a Favorite.

Slide 11: Inserting Data by Creating Nodes

Cypher Examples

```
CREATE (john:User {name: "John", birthPlace: "Dallas"})
CREATE (eve:User {name: "Eve", birthPlace: "Paris"})
```

Explanation / Expansion

- CREATE is used to add nodes (and relationships) to the graph.
- (john:User ...) means: "Create a node labeled User and assign it to the variable john."
- {name: "John", birthPlace: "Dallas"} is a property map for that node.

Slide Visual Description

- Shows sample Cypher queries for creating two users, John and Eve, with

properties.

Slide 12: Adding an Edge with No Variable Names

Cypher Example

```
CREATE (:User {name: "Bill", birthPlace: "Houston"})
CREATE (:User {name: "Alice", birthPlace: "2023-01-23"})
CREATE (:User {name: "Eve", birthPlace: "Paris"})
```

Then

```
MATCH (a:User {name: "Bill"}), (b:User {name: "Alice"})
CREATE (a)-[:KNOWS]->(b)
```

Explanation / Expansion

- We can create nodes without assigning them to variables by omitting the variable inside parentheses.
- The second query uses MATCH to find two existing nodes and creates a relationship between them.
- The :KNOWS is the relationship type.

Slide Visual Description

- Possibly code snippets in bullet points.
- Explanation that relationships are always directed in Neo4j.

Slide 13: Matching: Which Users Were Born in London?

Cypher Example

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

Explanation / Expansion

- MATCH is used to find nodes or relationships in the graph that match a pattern.
- Here, we match any User node that has the property birthPlace = "London".
- RETURN outputs the user’s name and birthplace.

Slide Visual Description

- A short snippet of the Cypher query.
- Possibly shows the result table with any matching users.

Slide 14: Download Dataset and Move to Import Folder

Instructions

- Link to a dataset: https://data.films... (example link)
- Move the CSV files into the import folder in your Neo4j instance or Docker volume.

Explanation / Expansion

- Neo4j can import CSV files located in its import directory.
- If you're using Docker, you might map a local directory to /var/lib/neo4j/import so that you can place CSV files there.

Slide Visual Description

- Possibly a screenshot showing the user copying or moving files into an import folder.
- Example CSV files might include netflix_titles.csv, imdb_titles.csv, etc.

Slide 15: Importing Data

Key Point

- You can use LOAD CSV in Cypher to import data into Neo4j.

Explanation / Expansion

- This is a built-in Neo4j command that allows reading CSV files, optionally specifying headers, field terminators, and so on.
- Must be mindful of path references (e.g., file:///filename.csv) and whether you have read/write permissions.

Slide Visual Description

- Title: "Importing Data"
- Possibly a bullet list describing the steps to use LOAD CSV.

Slide 16: Basic Data Importing

Cypher Example

```
LOAD CSV WITH HEADERS
FROM 'file:///netflix_titles.csv' AS line
CREATE (m:Movie {
  title: line.title,
  director: line.director
})
```

Explanation / Expansion

- LOAD CSV WITH HEADERS interprets the first row of the CSV as column names.
- Each row is accessible via the line variable (e.g., line.title).
- CREATE (m:Movie {...}) creates a node labeled Movie with properties from the CSV columns.

Slide Visual Description

- Shows code snippet for a basic import from netflix_titles.csv.

Slide 17: Loading CSVs – General Syntax

General Syntax

```
LOAD CSV [WITH|WITHOUT] HEADERS
FROM 'file:///some_file.csv' AS line
FIELDTERMINATOR ';'
[do stuff with line]
```

Explanation / Expansion

- WITH HEADERS or WITHOUT HEADERS depends on your CSV.
- FIELDTERMINATOR is optional if your CSV uses a non-default delimiter (e.g., semicolon).
- Inside the query, you can manipulate each row (line) to create nodes, relationships, or update existing data.

Slide Visual Description

- A bullet list or code snippet showing the general structure of a LOAD CSV statement.

———

Slide 18: Importing with Directors This Time

Cypher Example

```
LOAD CSV WITH HEADERS
FROM 'file:///netflix_titles.csv' AS line
MATCH (p:Person {name: line.director})
MERGE (m:Movie {title: line.title})
CREATE (p)-[:DIRECTED]->(m)
```

Explanation / Expansion

- Demonstrates how you can match or merge existing persons (:Person) and create relationships to newly merged movies.
- MERGE acts like an "upsert" — if the node doesn't exist, it will be created; if it does exist, it is matched.

Slide Visual Description

- Code snippet focusing on the creation of DIRECTED relationships.
- Possibly a bullet or highlight on MERGE vs. CREATE.

———

Slide 19: Matching and Removing Directors Merged

Cypher Example

```
MATCH (p:Person)
DELETE p
```

(Hypothetical example if you needed to remove incorrectly merged data.)

Or:

```
MATCH (p:Person)-[r:DIRECTED]->(m:Movie)
DELETE r
```

(Removes relationships but keeps the nodes.)

Explanation / Expansion

- MATCH + DELETE is how you remove nodes or relationships in Neo4j.

- If a node has relationships, you cannot delete the node unless you also delete or detach those relationships (DETACH DELETE p).

Slide Visual Description

- Possibly a snippet showing the commands to remove data from the graph.
- Could be used for cleaning up mistakes before re-importing.

Slide 20: Adding Edges

Cypher Example

```
LOAD CSV WITH HEADERS
FROM 'file:///some_titles.csv' AS line
MATCH (m:Movie {title: line.title})
MATCH (p:Person {name: line.director_name})
CREATE (p)-[:DIRECTED]->(m)
```

Explanation / Expansion

- You can do multiple MATCH clauses to find existing nodes (movies and people) and then create relationships between them.
- This step is often done after the nodes themselves have been created/merged in prior steps.

Slide Visual Description

- Another code snippet with bullet points.
- Emphasizes that relationships are created after we have the needed nodes in the graph.

Slide 21: Gut Check

Query Example

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

Explanation / Expansion

- This is a test query to see if the movie *Ray* and its director(s) were successfully imported.

- m, p in the return statement will show the movie node and the related person node in both table and graph view.

Screenshot Description

- The Neo4j Browser screenshot likely shows a node labeled “Ray” connected via a DIRECTED relationship to a person node labeled with the director’s name (e.g., “Taylor Hackford”).

Additional Notes and Best Practices

1. **Data Modeling**
   - Plan your labels (e.g., :Movie, :Person, :User) carefully.
   - Decide which properties each node should contain and which relationships to create.
2. **Indexes and Constraints**
   - For better performance, especially on large datasets, consider creating indexes or constraints (e.g., CREATE CONSTRAINT FOR (m:Movie) REQUIRE m.title IS UNIQUE).
3. **Docker Tips**
   - Always ensure your volumes are mapped properly so data persists.
   - .env is a best practice for storing credentials.
   - Use docker-compose logs -f neo4j to view logs in real time if something goes wrong.
4. **Security**
   - Don’t leave default credentials in production environments.
   - Limit external access to Neo4j’s ports if not needed.
5. **Learning Resources**
   - The Neo4j Documentation covers more advanced topics like indexing, constraints, performance tuning, APOC usage, etc.
   - The Neo4j Browser Guide is helpful for new users.

Conclusion

This write-up provides an expanded overview of the slides regarding Neo4j basics, how to run Neo4j in Docker Compose, how to use .env files, and how to import data with Cypher commands such as LOAD CSV. The screenshots have been described in text so you can fully understand the setup, commands, and the Neo4j

Browser interface.

With this information, you should be able to:

- Spin up a Neo4j container via Docker Compose,
- Log into the Neo4j Browser at localhost:7474,
- Create nodes and relationships in Cypher,
- Import CSV data into Neo4j, and
- Verify your data by running test queries.