MySQL with Hadoop:

What is Mysql :

MySQL is an open-source relational database management system (RDBMS) that enables efficient storage, retrieval, and management of data. It uses a structured query language (SQL) for interacting with databases and is widely utilized for web applications and various software applications requiring a reliable and scalable database solution.

Why we need to use Mysql with Hadoop:

MySQL is often integrated with Hadoop to leverage the strengths of both systems. Hadoop is a distributed storage and processing framework designed for handling large volumes of data, while MySQL is a relational database system known for its structured and efficient data querying capabilities.

1. **Combine Structured and Unstructured Data:** Hadoop is well-suited for processing and storing unstructured or semi-structured data, while MySQL excels in handling structured data. Integrating MySQL with Hadoop enables organizations to work seamlessly with both types of data, providing a comprehensive solution for diverse data needs.
2. **Data Processing and Analysis:** Hadoop's parallel processing capabilities are ideal for large-scale data processing and analytics. By integrating MySQL, which is proficient in handling relational data, users can perform complex analyses on structured data stored in MySQL databases, complementing the capabilities of Hadoop.
3. **Historical and Real-time Data Integration:** Hadoop is often used for storing and processing historical data, while MySQL can handle real-time data efficiently. Integrating both systems allows organizations to create a unified data environment that supports both historical and real-time analytics.
4. **Scalability:** Hadoop's distributed architecture provides scalability for handling massive amounts of data. Combining it with MySQL allows organizations to scale their data storage and processing capabilities according to their evolving needs.
5. **Ecosystem Compatibility:** Hadoop has a rich ecosystem of tools for various data processing tasks. Integrating MySQL with Hadoop allows users to benefit from this ecosystem, incorporating data from MySQL into Hadoop-based workflows and applications.

Sqoop:

Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured data stores such as relational databases. Sqoop stands for SQL-to-Hadoop. It provides a command-line interface to import data from relational databases into Hadoop Distributed File System (HDFS) or Hive, and to export data from Hadoop to relational databases. Sqoop helps bridge the gap between traditional relational databases and Hadoop, facilitating data transfer and integration between the two environments.

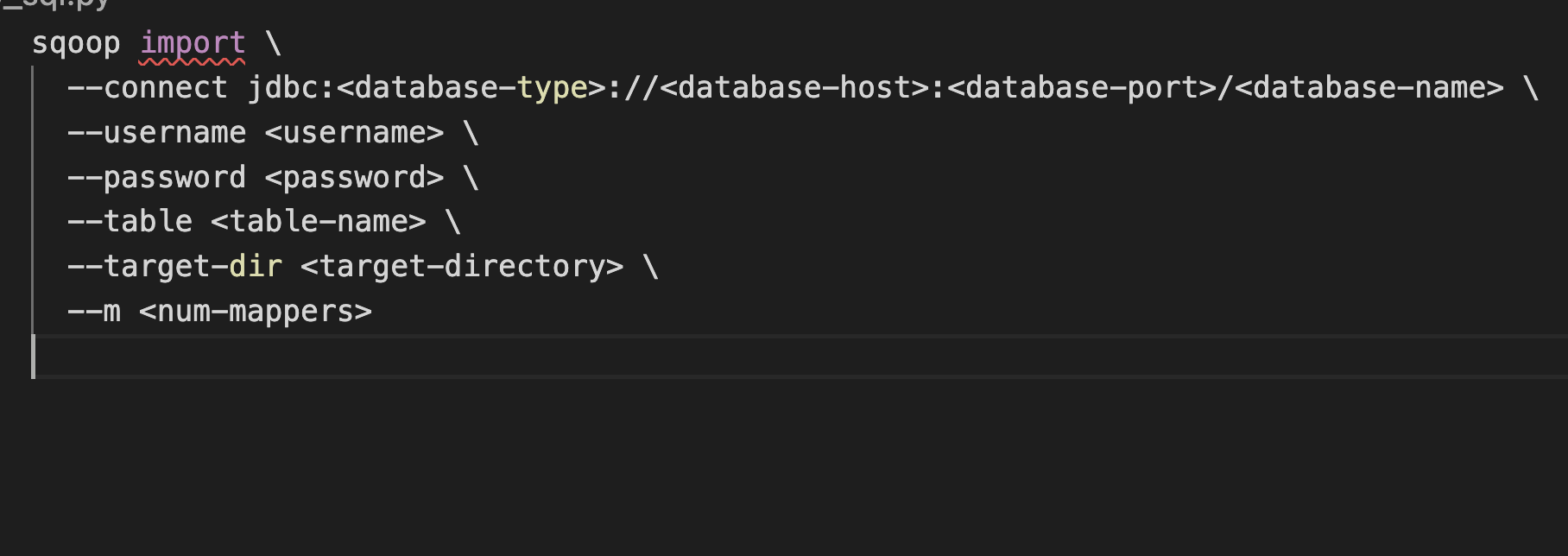
Here's a brief overview of how a Sqoop job typically works:

1. **Connection Initialization:**
   * Sqoop needs to establish a connection to both the Hadoop cluster and the relational database. This involves providing connection details such as JDBC URL, username, password, etc.
2. **Import or Export Configuration:**
   * Sqoop supports two main operations: importing data from a relational database to Hadoop and exporting data from Hadoop to a relational database. The user specifies the operation and provides details like the source and destination tables, columns, etc.
3. **MapReduce Job Generation:**
   * Sqoop generates a MapReduce job based on the specified import or export configuration. This job is responsible for efficiently moving the data between the Hadoop cluster and the relational database.
4. **Data Transfer:**
   * For import operations, Sqoop divides the data into splits and assigns each split to a Map task. These tasks run in parallel, pulling data from the relational database and storing it in HDFS or Hive.
   * For export operations, the MapReduce job reads data from HDFS or Hive, divides it into splits, and inserts it into the relational database.
5. **Data Serialization/Deserialization:**
   * Sqoop handles the serialization and deserialization of data between the relational database's format and the format used in Hadoop (usually text or Avro).
6. **Data Integrity and Consistency:**
   * Sqoop ensures data integrity during the transfer by handling retries, transaction management, and error handling.
7. **Job Completion:**
   * Once the MapReduce job is completed, Sqoop reports the status and details of the transfer, including any errors or warnings.
8. **Cleanup:**
   * Sqoop performs any necessary cleanup tasks, closing connections and releasing resources.

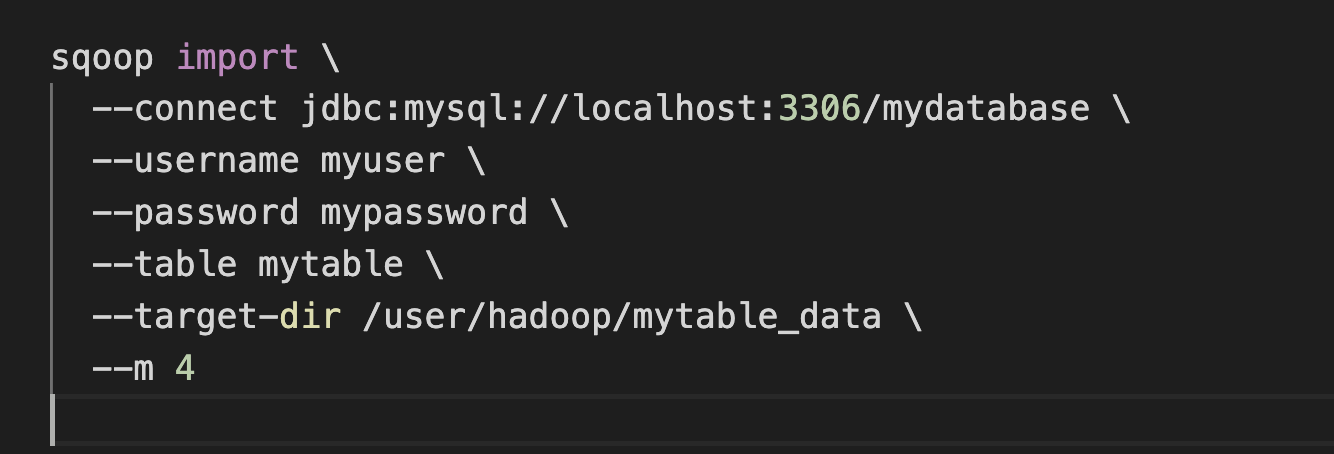
JDBC:

JDBC stands for Java Database Connectivity, and a JDBC connector is a software component or driver that enables Java applications to interact with and access relational databases. It acts as an interface between the Java application and the database, allowing the application to send SQL queries, statements, and updates to the database and receive the results.

Sqoop Import Commands:



Sqoop Export Commands:



* **<database-type>**: The type of the database (e.g., **mysql**, **oracle**, **sqlserver**).
* **<database-host>**: The hostname or IP address of the database server.
* **<database-port>**: The port on which the database server is listening.
* **<database-name>**: The name of the database.
* **<username>**: The username for connecting to the database.
* **<password>**: The password for connecting to the database.
* **<table-name>**: The name of the table to import or export.
* **<target-dir>**: The HDFS directory where the imported data will be stored.
* **<num-mappers>**: The number of parallel map tasks to use during import.

For export, additional parameters like **--input-fields-terminated-by**, **--update-key**, and **--update-mode** are used to specify the input field delimiter, the primary key column for updates, and the update mode, respectively

Top of Form

Bottom of Form

Setting up MySQL :

Run below queries:

start it with specific options (like skipping grant tables and networking), change the root user's password, and then restart the MySQL service without those special options.

1. **systemctl stop mysqld**: Stops the MySQL service.
2. **systemctl set-environment MYSQLD\_OPTS="--skip-grant-tables --skip-networking"**: Sets environment variables for the MySQL service, instructing it to skip grant tables and networking. This effectively allows you to access MySQL without a password.
3. **systemctl start mysqld**: Restarts the MySQL service with the specified environment options.
4. **mysql -uroot**: Connects to MySQL as the root user without a password due to the skip-grant-tables option.
5. **FLUSH PRIVILEGES;**: Reloads the grant tables, applying changes.
6. **alter user 'root'@'localhost' IDENTIFIED BY 'hadoop';**: Changes the root user's password to 'hadoop'.
7. **FLUSH PRIVILEGES;**: Reloads the grant tables again to apply the password change.
8. **QUIT;**: Exits the MySQL shell.
9. **systemctl unset-environment MYSQLD\_OPTS**: Unsets the previously set environment variables.
10. **systemctl restart mysqld**: Restarts the MySQL service without the skip-grant-tables and skip-networking options.

After running these commands, you should be able to log in to MySQL with the new password 'hadoop'. Note that using **--skip-grant-tables** and altering the password in this manner should be done with caution, especially in a production environment, as it temporarily bypasses normal authentication mechanisms. Always ensure that you understand the security implications of such actions.

Top of Form

Bottom of Form

su root

systemctl stop mysqld

systemctl set-environment MYSQLD\_OPTS="--skip-grant-tables --skip-networking"

systemctl start mysqld

mysql -uroot

------- Mysql cmd

FLUSH PRIVILEGES;

alter user 'root'@'localhost' IDENTIFIED BY 'hadoop';

FLUSH PRIVILEGES;

QUIT;

------ CMD

systemctl unset-environment MYSQLD\_OPTS

systemctl restart mysqld