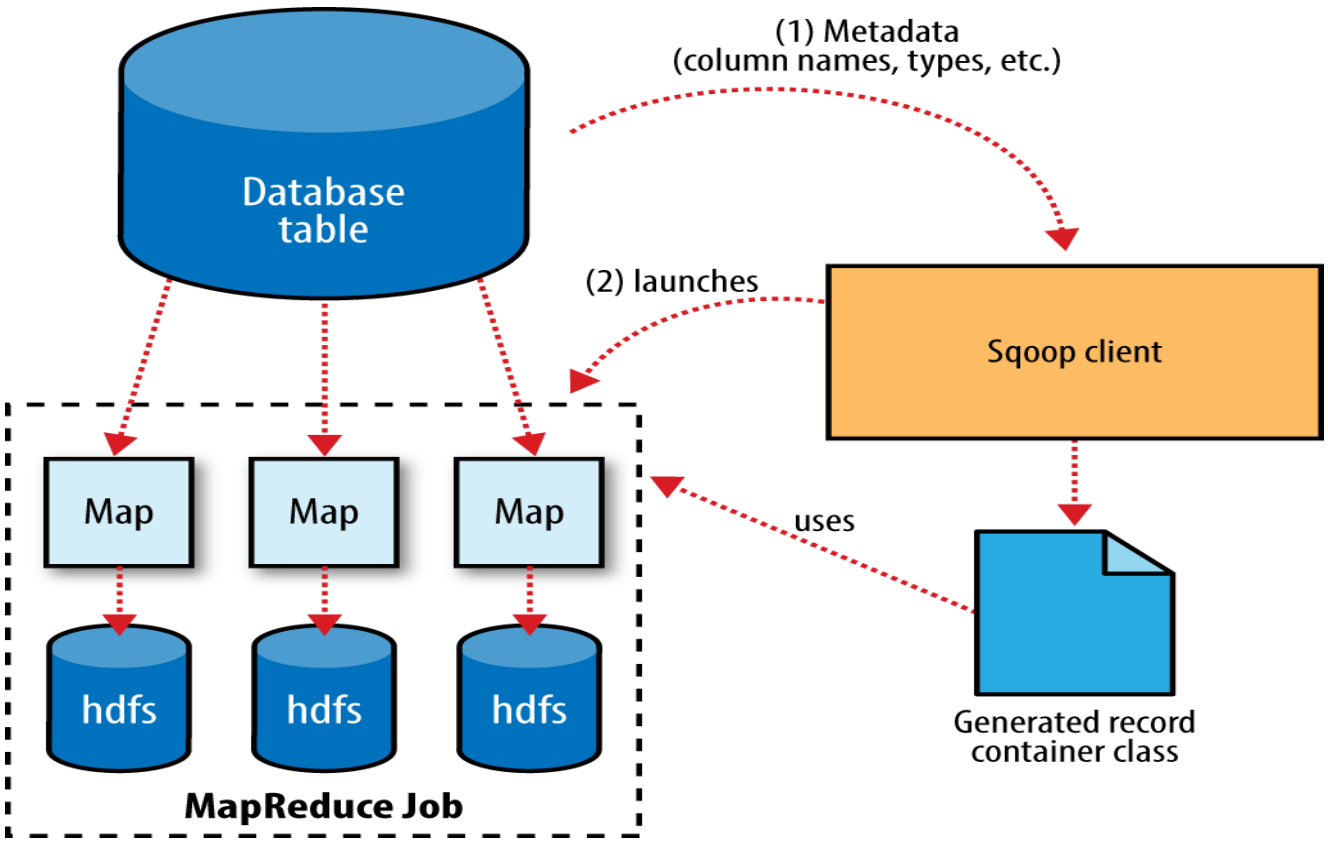
Q. The workflow of Sqoop and its Benefits

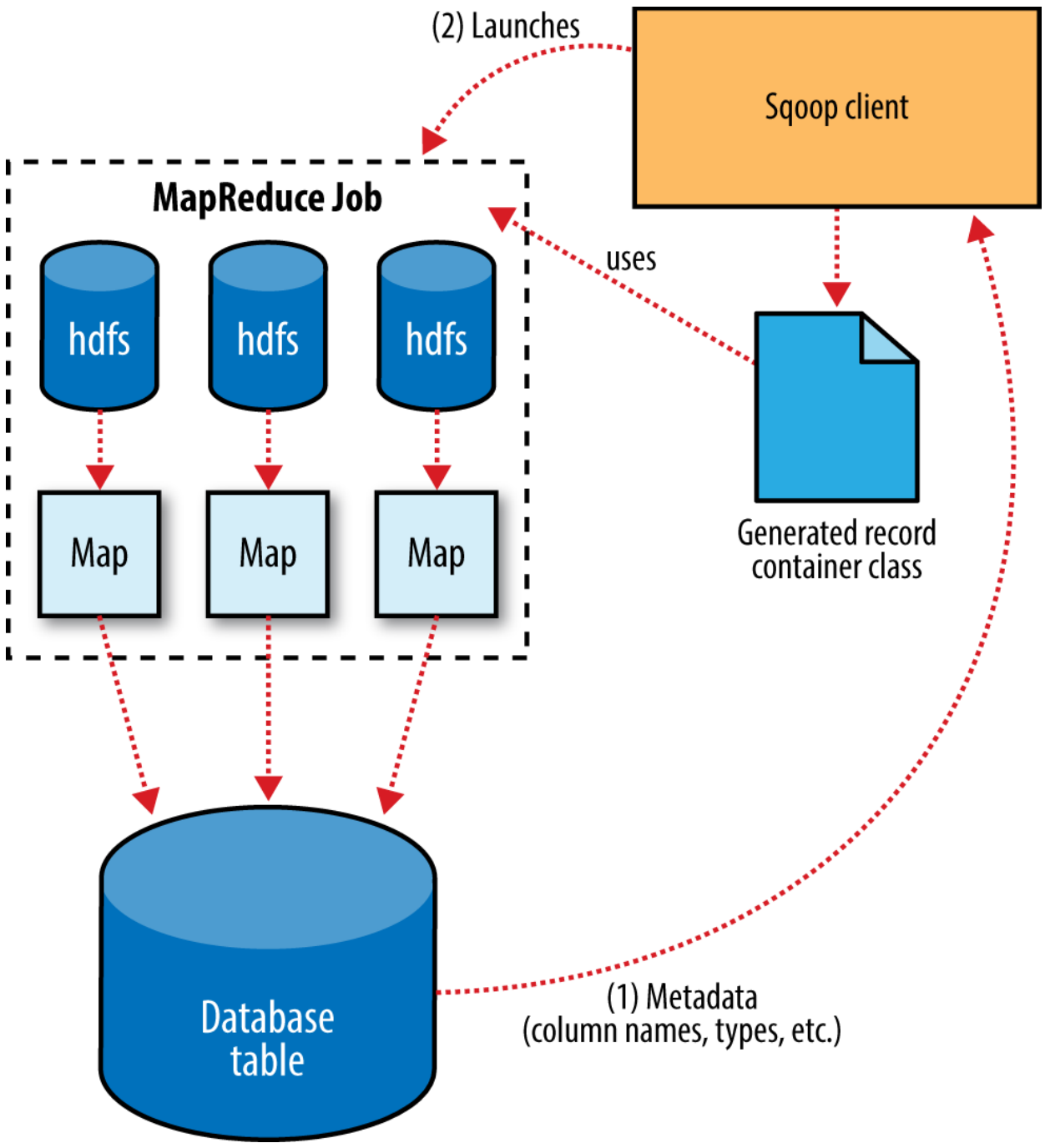
Ans. Apache Sqoop is a Hadoop tool used for importing and exporting data between relational databases MySQL, Oracle, etc. and Hadoop clusters. Sqoop commands are structured around connecting to and importing or exporting data from various relational databases. It often uses JDBC to talk to these external database systems. The major operations which we do via sqoop is import and export.

Import- Sqoop imports a table from a database by running a MapReduce job that extracts rows from the table, and writes the records to HDFS. Sqoop is written in Java. Java provides an API called Java Database Connectivity, or JDBC, that allows applications to access data stored in an RDBMS as well as to inspect the nature of this data. Most database vendors provide a JDBC driver that implements the JDBC API and contains the necessary code to connect to their database servers. Before the import can start, Sqoop uses JDBC to examine the table it is to import. It retrieves a list of all the columns and their SQL data types. These SQL types (VARCHAR, INTEGER, etc.) can then be mapped to Java data types (String, Integer, etc.), which will hold the field values in MapReduce applications. Sqoop's code generator will use this information to create a table-specific class to hold a record extracted from the table. Below diagram shows the workflow of import process-



Export- In Sqoop, an import refers to the movement of data from a database system into HDFS. By contrast, an export uses HDFS as the source of data and a remote database as the destination. Before exporting a table from HDFS to a database, we must prepare the database to receive the data by creating the target table. Although Sqoop can infer which Java types are appropriate to hold SQL data types, this translation does not work in both directions. This generated class has the ability to parse records from text files and insert values of the appropriate types into a table (in addition to the ability to read the columns from a ResultSet). A MapReduce job is then launched that reads the source datafiles from HDFS, parses the records using the generated class, and executes the chosen export strategy. The JDBC-based export strategy builds up batch INSERT statements that will each add multiple records to the target table. Inserting many records per statement performs much better than executing many single-row INSERT statements on most database systems. Separate threads are used to read from HDFS and communicate with the database, to ensure that I/O operations involving different systems are overlapped as much as possible.

Below diagram shows the workflow of Export-



For MySQL, Sqoop can employ a direct-mode strategy using **mysqlimport**. Each map task spawns a **mysqlimport** process that it communicates with via a named FIFO file on the local file system. Data is then streamed into **mysqlimport** via the FIFO channel, and from there into the database. Whereas most MapReduce jobs reading from HDFS pick the degree of parallelism (number of map tasks) based on the number and size of the files to process, Sqoop's export system allows users explicit control over the number of tasks. The performance of the export can be affected by the number of parallel writers to the database, so Sqoop uses the CombineFileInputFormat class to group the input files into a smaller number of map tasks.

Now apart from above we can run Sqoop's job also. Oozie's sqoop action helps users run Sqoop jobs as part of the workflow. The following elements are part of the Sqoop action

1. job-tracker (required)
2. name-node (required)
3. prepare
4. job-xml
5. configuration
6. command (required if arg is not used)
7. arg (required if command is not used)
8. file
9. archive

The action needs to know the JobTracker and the NameNode of the underlying Hadoop cluster where Oozie has to run the sqoop action .The <prepare> section is optional and is typically used as a preprocessor to delete output directories or HCatalog table partitions or to create some directories required for the action. This delete helps make the action repeatable and enables retries after failure.
The <job-xml> element or the <configuration> section can be used to capture all of the Hadoop job configuration properties.For hive action we will be using the <job-xml> tag to pass the hive-site.xml.This way, the hive-site.xml is just reused in its entirety and no additional configuration settings or special files are necessary.
Oozie also supports the <file> and <archive> elements for actions that need them. This is the native, Hadoop way of packaging libraries, archives, scripts, and other data files that jobs need, and Oozie provides the syntax to handle them. The arguments to Sqoop are sent either through the <command> element in one line or broken down into many <arg> elements.

The sqoop action runs a Sqoop job. The workflow job will wait until the Sqoop job completes before continuing to the next action. To run the Sqoop job, you have to configure the sqoop action with the =job-tracker=, name-node and Sqoop command or arg elements as well as configuration. A sqoop action can be configured to create or delete HDFS directories before starting the Sqoop job. Sqoop configuration can be specified with a file, using the job-xml element, and inline, using the configuration elements.

The Sqoop command can be specified either using the command element or multiple arg elements. When using the command element, Oozie will split the command on every space into multiple arguments. When using the arg elements, Oozie will pass each argument value as an argument to Sqoop. The arg variant should be used when there are spaces within a single argument.
The counters of the map-reduce job run by the Sqoop action are available to be used in the workflow via the hadoop:counters() EL function . If the Sqoop action run an import all command, the hadoop:counters() EL will return the aggregated counters of all map-reduce jobs run by the Sqoop import all command.

Sqoop action logs are redirected to the Oozie Launcher map-reduce job task STDOUT/STDERR that runs Sqoop. From Oozie web-console, from the Sqoop action pop up using the 'Console URL' link, it is possible to navigate to the Oozie Launcher map-reduce job task logs via the Hadoop job-tracker web-console.
The logging level of the Sqoop action can set in the Sqoop action configuration using the property oozie.sqoop.log.level . The default value is INFO .

Below are the benefits of Sqoop-

- Allows the transfer of data with a variety of structured data stores like Postgres, Oracle, Teradata, and so on.
- Since the data is transferred and stored in Hadoop, Sqoop allows us to offload certain processing done in the ETL (Extract, Load and Transform) process into low-cost, fast, and effective Hadoop processes.
- Sqoop can execute the data transfer in parallel, so execution can be quick and more cost effective.
- Helps to integrate with sequential data from the mainframe. This helps not only to limit the usage of the mainframe, but also reduces the high cost in executing certain jobs using mainframe hardware.
- Bulk import: Big Data Sqoop facilitates the import of singular tables and comprehensive databases into HDFS. The information is saved in the native directories and files in the HDFS file system.
- Data interaction: Big Data Sqoop is capable of generating Java classes so that you can interact with the data in the scope of programming.
- Ease of Use - Sqoop lets connectors to be configured in one place, which can be managed by the admin role and run by the operator role. This centralized architecture helps in better deployment of Big Data analytics and solutions.
- Ease of Extension - The connectors of Sqoop are not restricted to just the JDBC model. It has the competencies to extend and define its own vocabulary without having the need to mention a table name.
- Security - The fact that Sqoop operates as server based application that secures access to external systems and does not allow code generation, makes its security to go by.