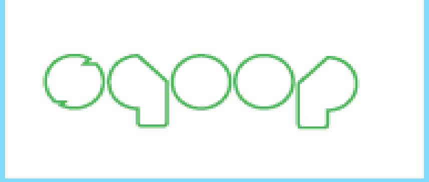
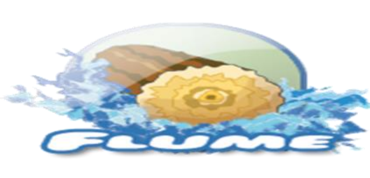
ANALYTIXLABS

Hadoop
Sqoop –Flume





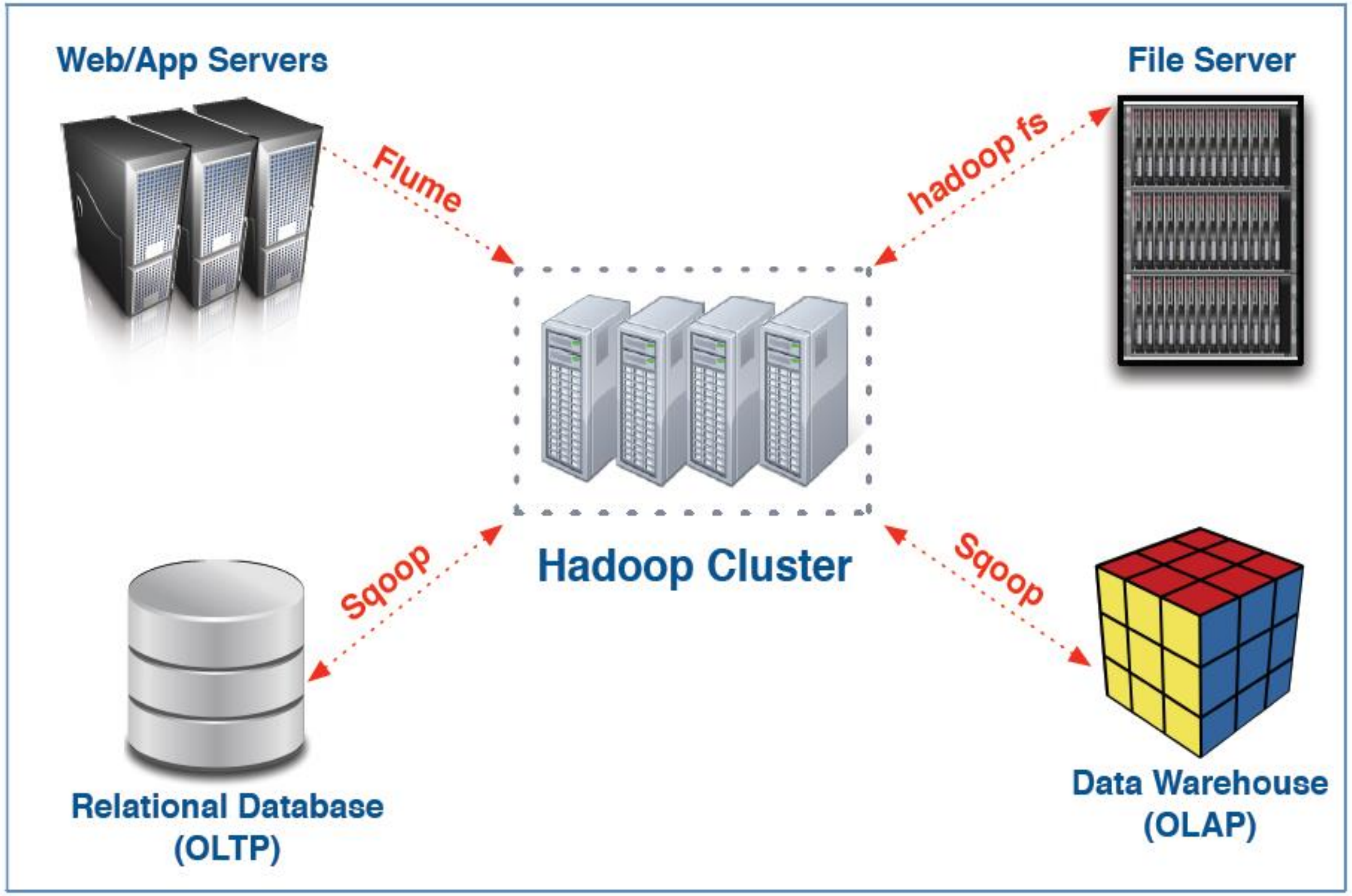
(Frame work for Data Ingestion)

Disclaimer: This material is protected under copyright act AnalytixLabs ©, 2011-2016. Unauthorized use and/ or duplication of this material or any part of this material including data, in any form without explicit and written permission from AnalytixLabs is strictly prohibited. Any violation of this copyright will attract legal actions

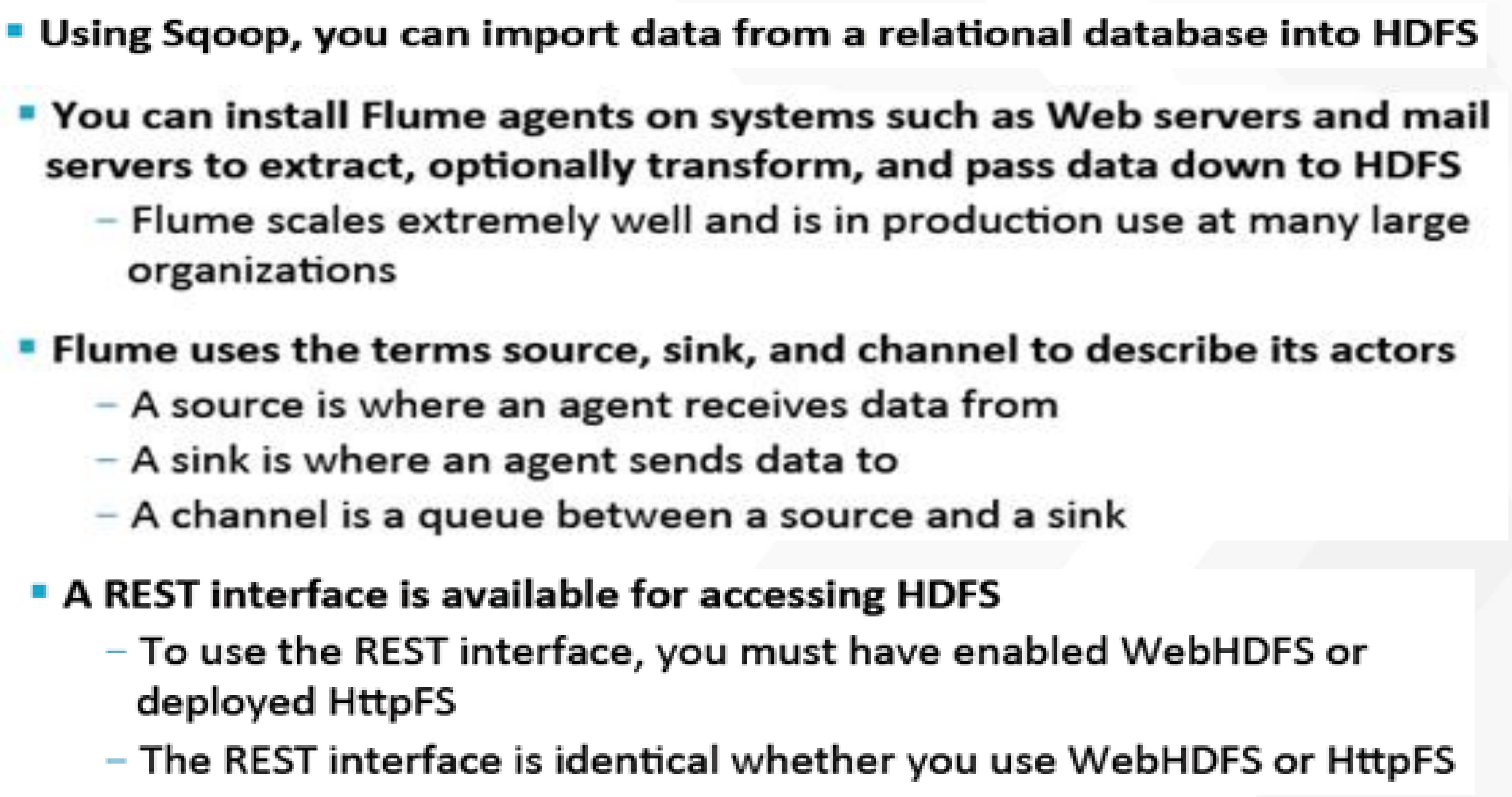
Getting Data into HDFS

In the last session, we learned how to use hadoop fs command to copy the data into and out of HDFS. Now we will see,

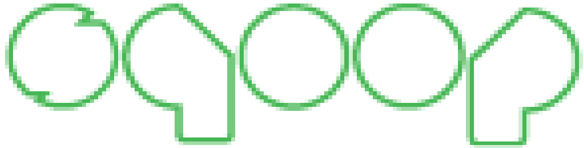
- ➢How to import data into HDFS using SQOOP?
- ➢How to import data into HDFS using Flume?
- ➢What REST interfaces Hadoop provides?



Getting Data into HDFS

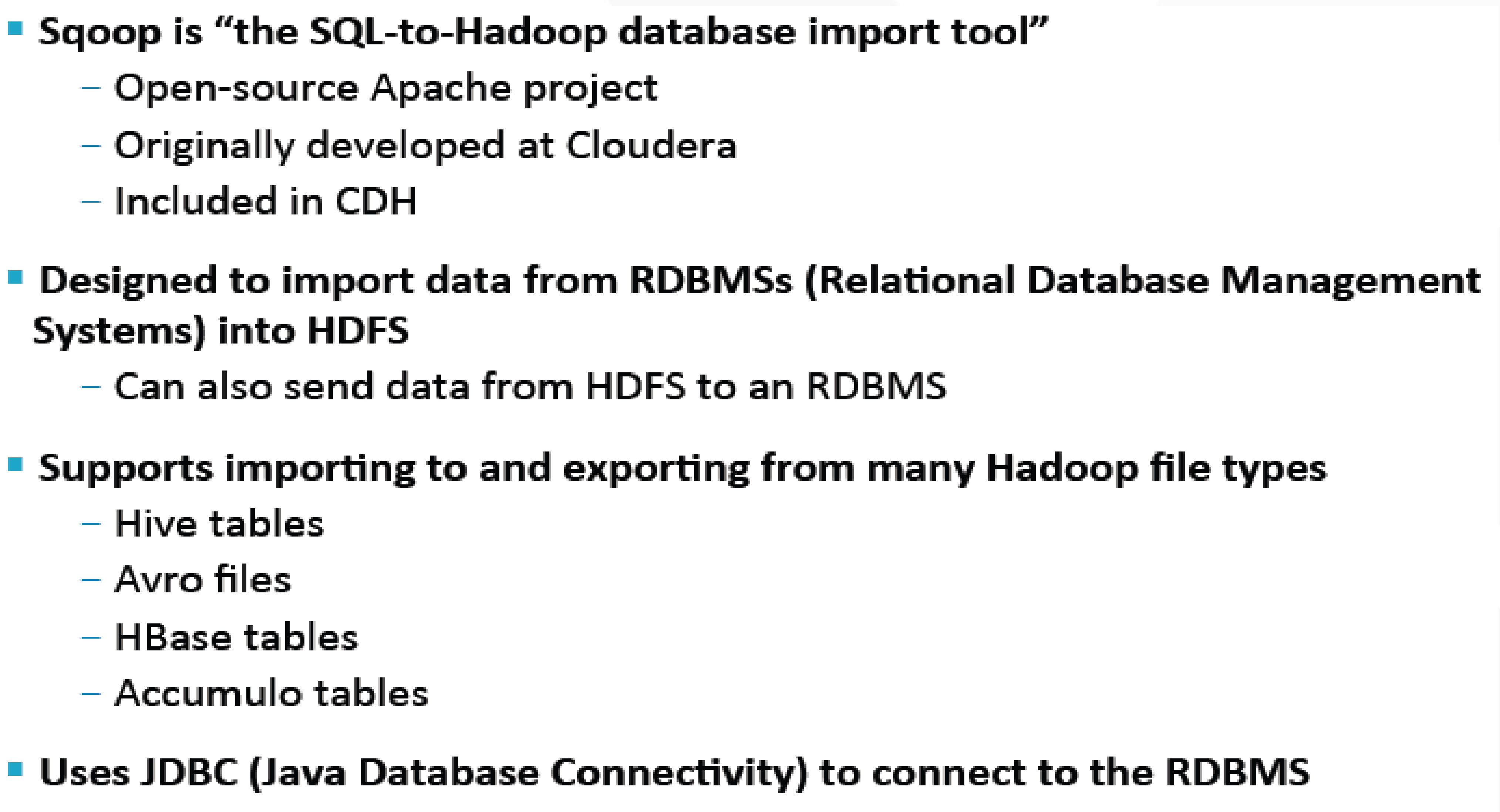


- Using Sqoop, you can import data from a relational database into HDFS
- You can install Flume agents on systems such as Web servers and mail servers to extract, optionally transform, and pass data down to HDFS
  - Flume scales extremely well and is in production use at many large organizations
- Flume uses the terms source, sink, and channel to describe its actors
  - A source is where an agent receives data from
  - A sink is where an agent sends data to
  - A channel is a queue between a source and a sink
- A REST interface is available for accessing HDFS
  - To use the REST interface, you must have enabled WebHDFS or deployed HttpFS
  - The REST interface is identical whether you use WebHDFS or HttpFS



http://sqoop.apache.org/

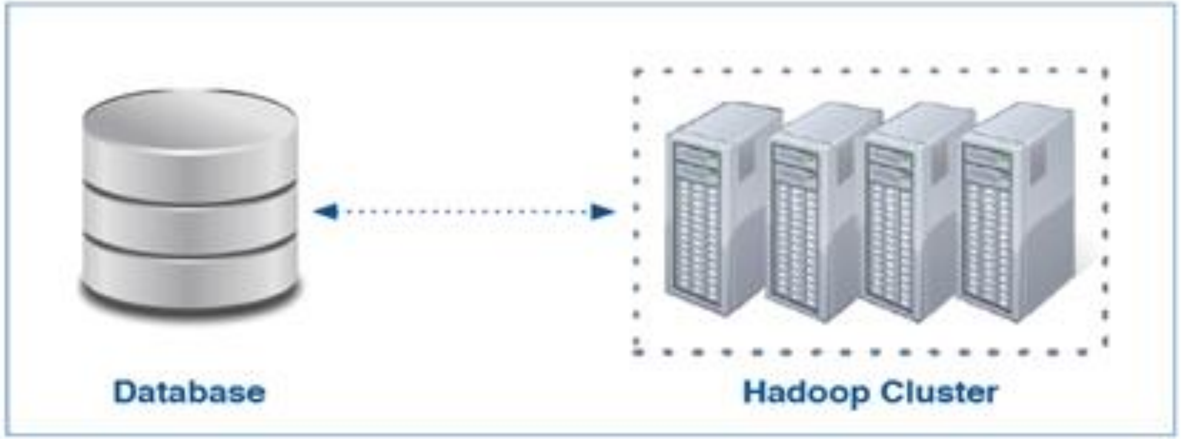
What is Sqoop?

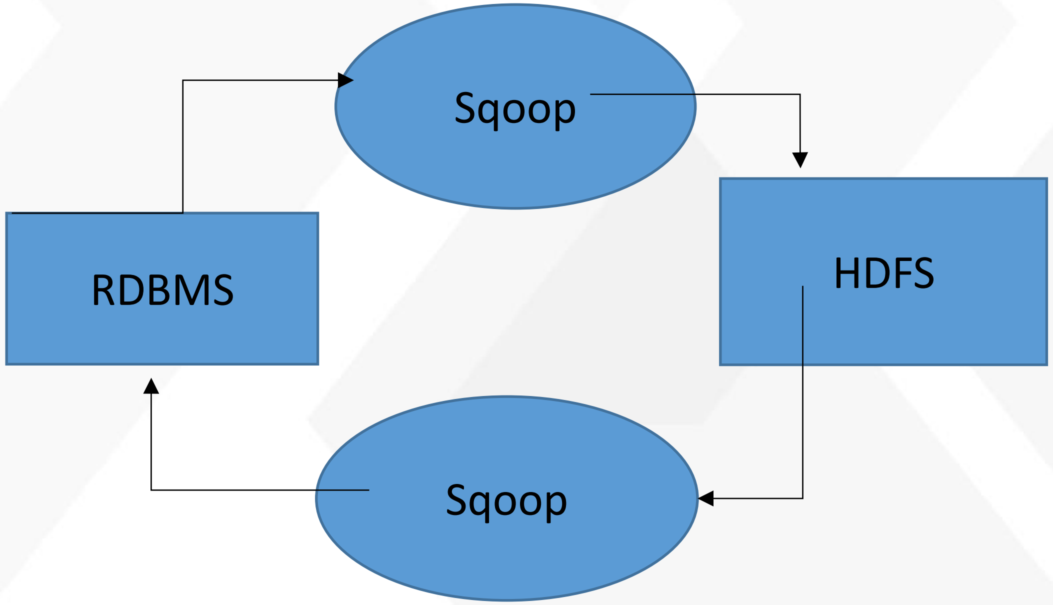


- **Sqoop is “the SQL-to-Hadoop database import tool”**
  - Open-source Apache project
  - Originally developed at Cloudera
  - Included in CDH
- **Designed to import data from RDBMSs (Relational Database Management Systems) into HDFS**
  - Can also send data from HDFS to an RDBMS
- **Supports importing to and exporting from many Hadoop file types**
  - Hive tables
  - Avro files
  - HBase tables
  - Accumulo tables
- **Uses JDBC (Java Database Connectivity) to connect to the RDBMS**

Basic usage of Sqoop

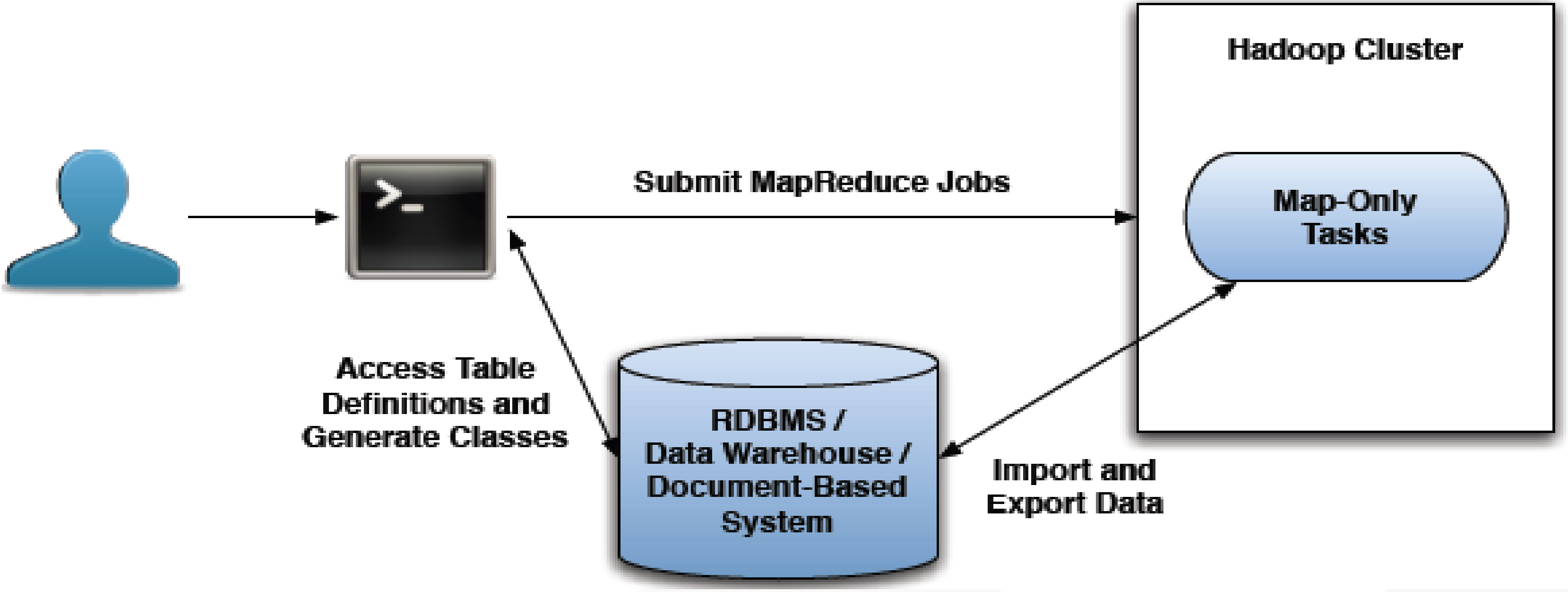
- Sqoop uses MapReduce to import and export the data which provides parallel operation as well as fault tolerance
- Sqoop will read table row-by-row into HDFS. The output of this import process is set of files containing a copy of the imported table
- The import process is performed in parallel hence there will be multiple files
- After manipulating the imported records, result dataset can be exported back to RDBMS
- Sqoop's export process will read a set of delimited text files from HDFS in parallel, parse them into record, and insert them as new row





How does sqoop works?

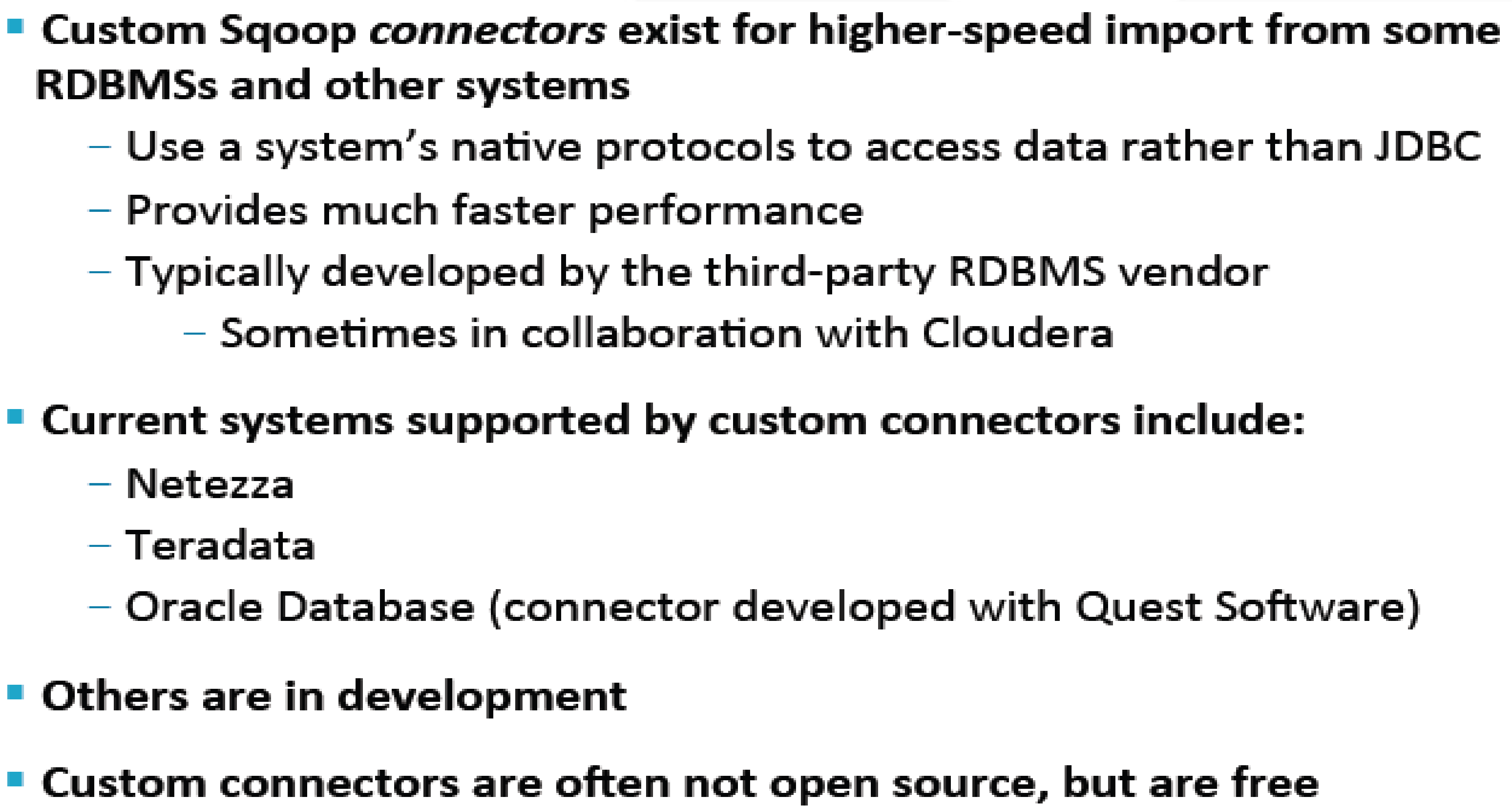
- **Sqoop examines each table and automatically generates a Java class to import data into HDFS**
- **It then creates and runs a Map-only MapReduce job to import the data**
  - By default, four Mappers connect to the RDBMS
  - Each imports a quarter of the data


Hadoop Cluster
Submit MapReduce Jobs
Map-Only Tasks
Access Table Definitions and Generate Classes
RDBMS / Data Warehouse / Document-Based System
Import and Export Data

Sqoop Features

- Imports a single table, or all tables in a database
- Can specify which rows to import
  - Via a WHERE clause
- Can specify which columns to import
- Can provide an arbitrary SELECT statement
- Sqoop can automatically create a Hive table based on the imported data
- Supports incremental imports of data
- Can export data from HDFS to a database table

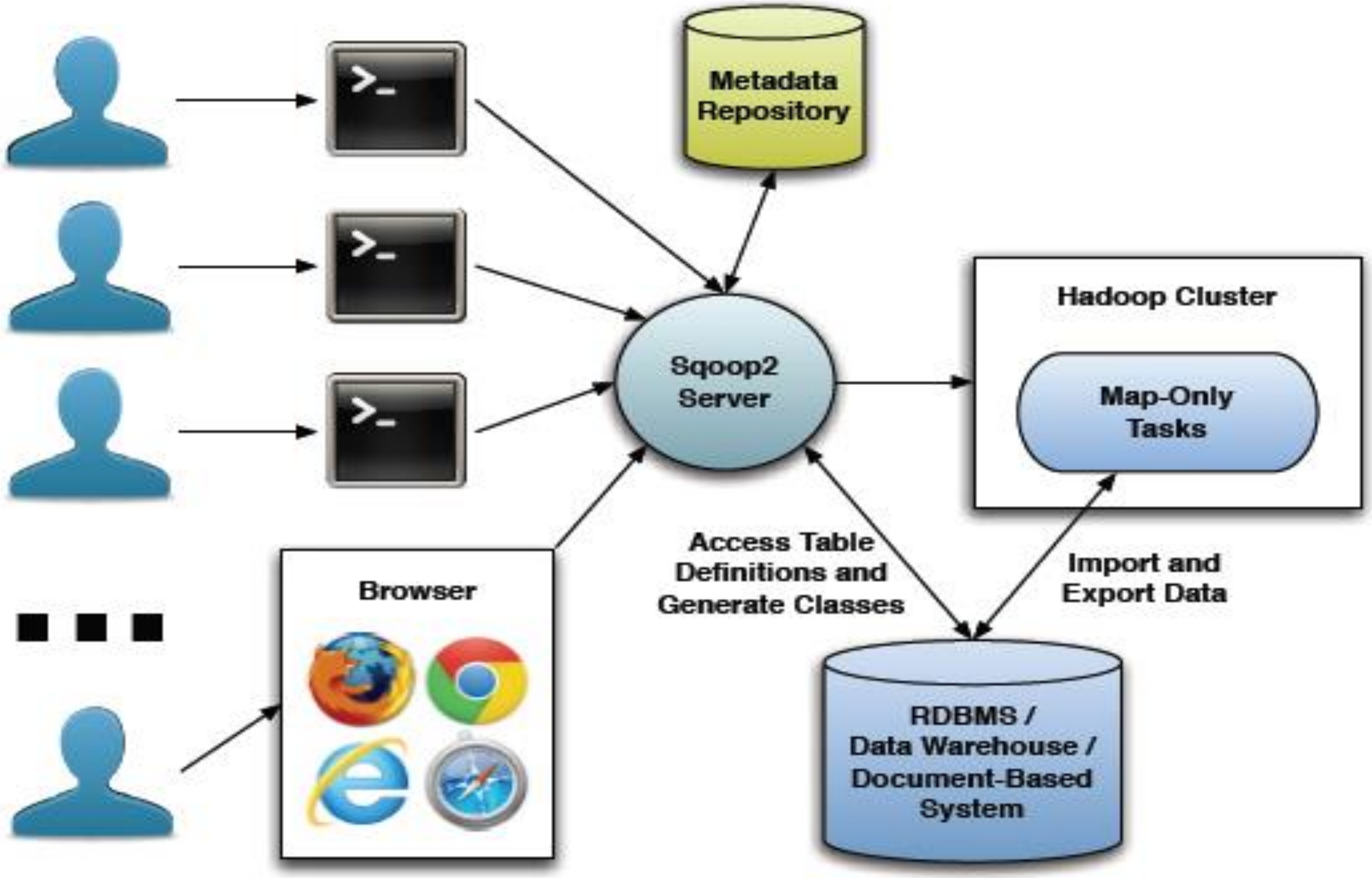
Sqoop Connectors



- Custom Sqoop *connectors* exist for higher-speed import from some RDBMSs and other systems
  - Use a system's native protocols to access data rather than JDBC
  - Provides much faster performance
  - Typically developed by the third-party RDBMS vendor
    - Sometimes in collaboration with Cloudera
- Current systems supported by custom connectors include:
  - Netezza
  - Teradata
  - Oracle Database (connector developed with Quest Software)
- Others are in development
- Custom connectors are often not open source, but are free

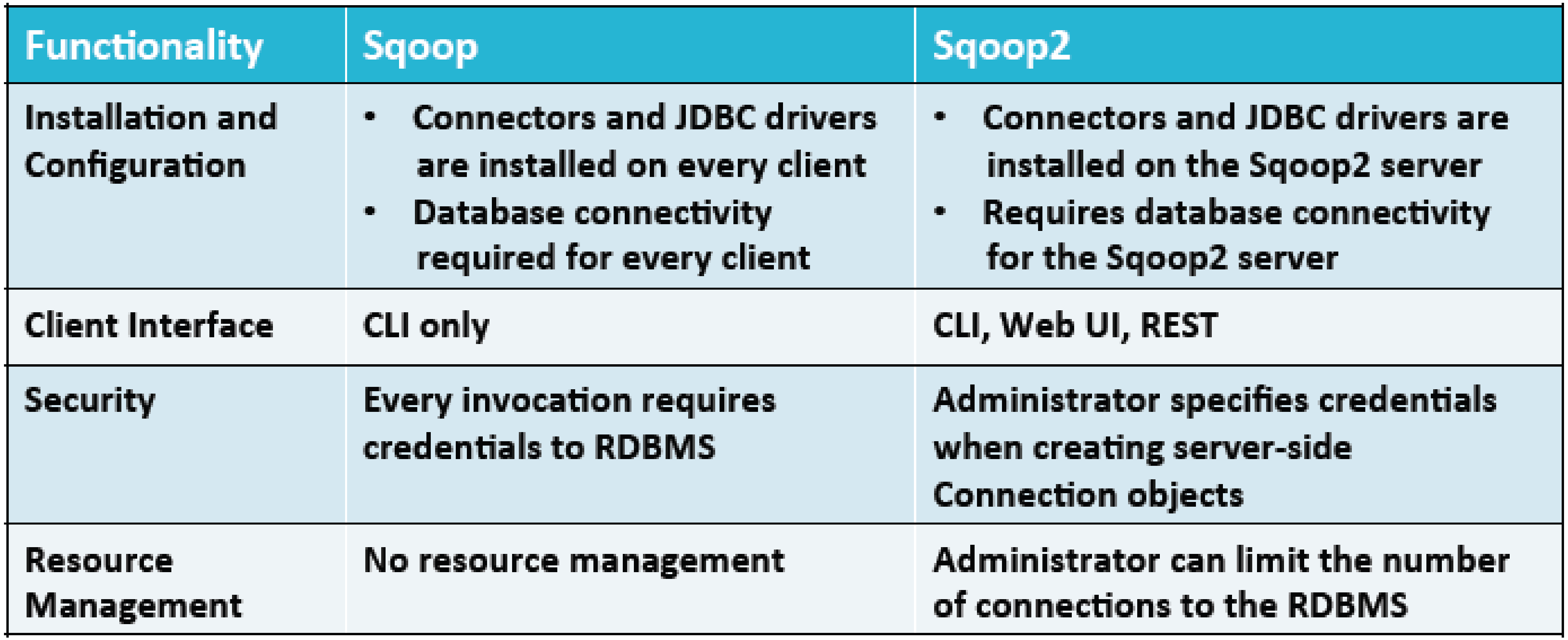
Sqoop 2 – Sqoop as service

- **New version of Sqoop can be run as a service on a centrally-available machine**



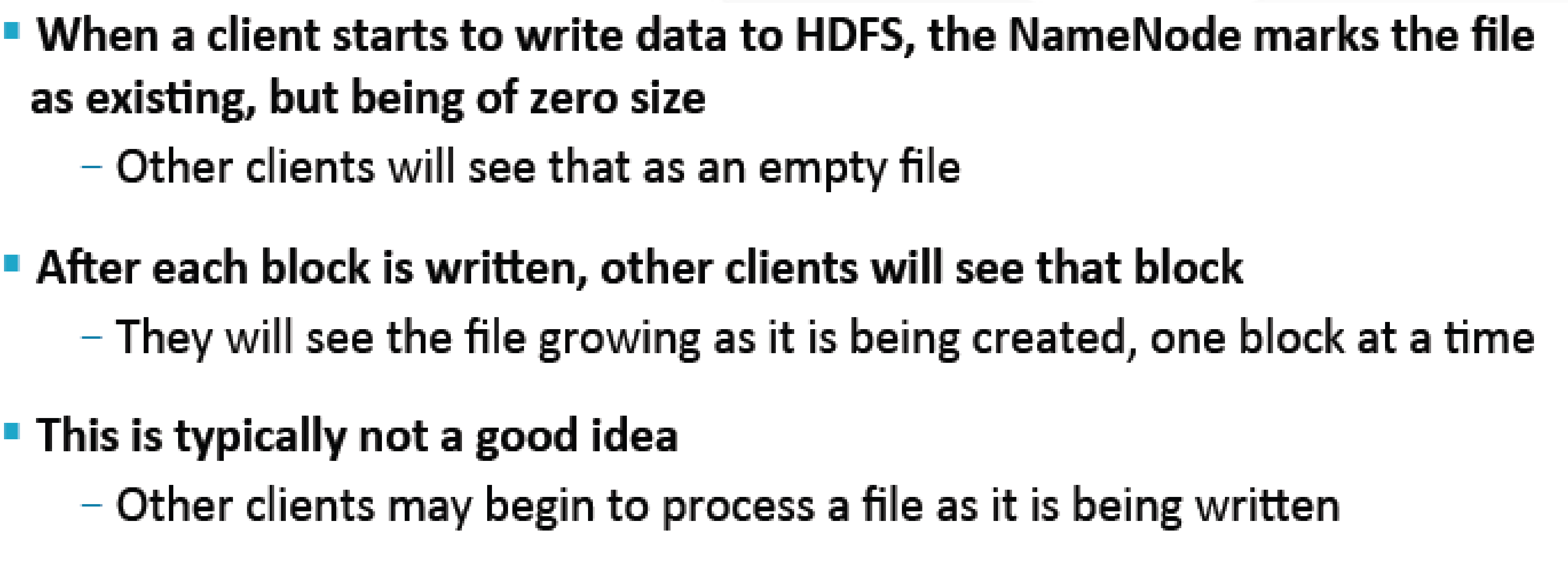
Metadata Repository
Hadoop Cluster
Sqoop2 Server
Map-Only Tasks
Access Table Definitions and Generate Classes
Import and Export Data
Browser
RDBMS / Data Warehouse / Document-Based System

Sqoop vs. Sqoop2



Functionality	Sqoop	Sqoop2
Installation and Configuration	• Connectors and JDBC drivers are installed on every client • Database connectivity required for every client	• Connectors and JDBC drivers are installed on the Sqoop2 server • Requires database connectivity for the Sqoop2 server
Client Interface	CLI only	CLI, Web UI, REST
Security	Every invocation requires credentials to RDBMS	Administrator specifies credentials when creating server-side Connection objects
Resource Management	No resource management	Administrator can limit the number of connections to the RDBMS

What do the others see as data is imported?

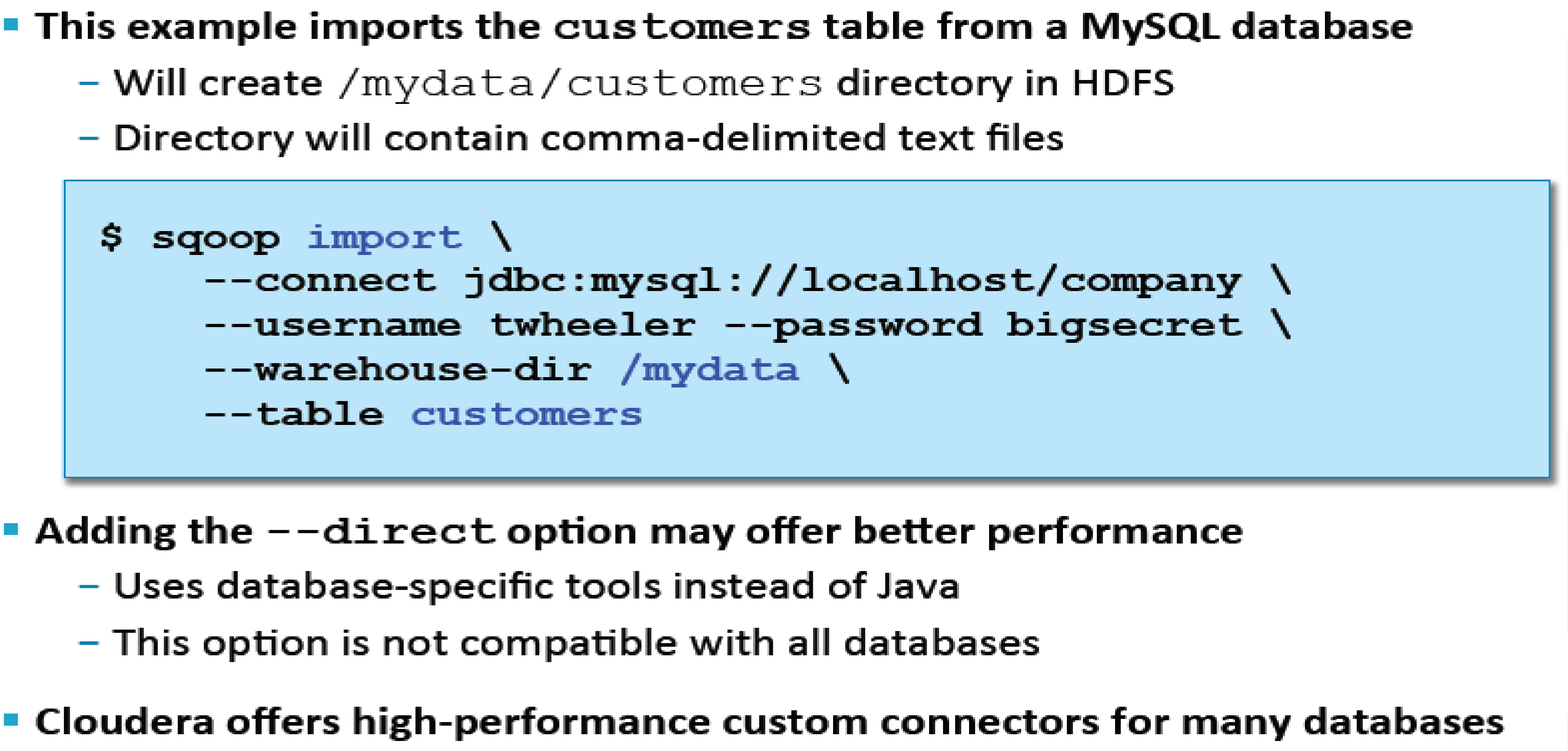


- **When a client starts to write data to HDFS, the NameNode marks the file as existing, but being of zero size**
  - Other clients will see that as an empty file
- **After each block is written, other clients will see that block**
  - They will see the file growing as it is being created, one block at a time
- **This is typically not a good idea**
  - Other clients may begin to process a file as it is being written

Importing Data: Best Practices

- **Best practice is to import data into a temporary directory**
- **After the file is completely written, move data to the target directory**
  - This is an atomic operation
  - Happens very quickly since it merely requires an update of the NameNode's metadata
- **Many organizations standardize on a directory structure such as**
  - `/incoming/<import_job_name>/<files>`
  - `/for_processing/<import_job_name>/<files>`
  - `/completed/<import_job_name>/<files>`
- **It is the job's responsibility to move the files from `for_processing` to `completed` after the job has finished successfully**

Sqoop Examples



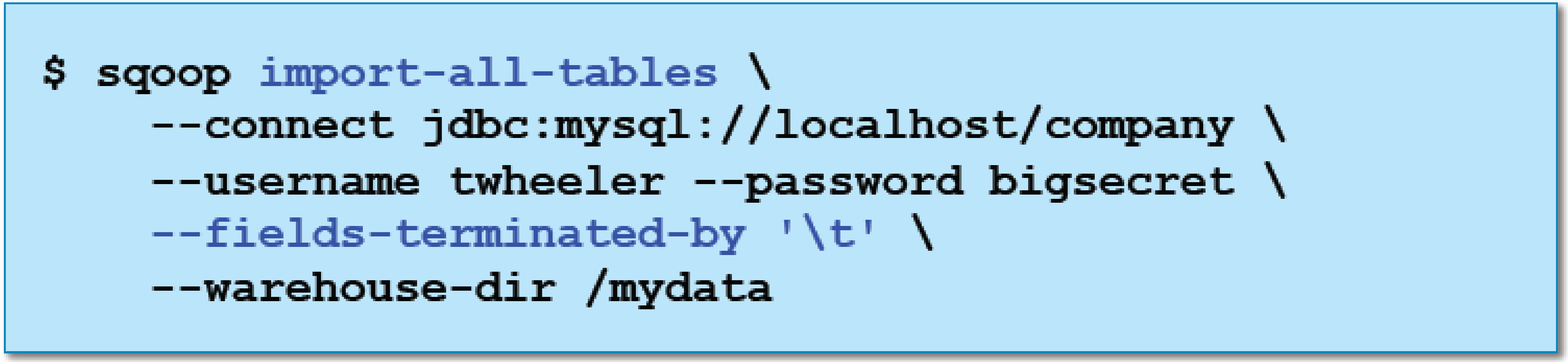
- This example imports the `customers` table from a MySQL database
  - Will create `/mydata/customers` directory in HDFS
  - Directory will contain comma-delimited text files

```
$ sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username twheeler --password bigsecret \
    --warehouse-dir /mydata \
    --table customers
```

- Adding the `--direct` option may offer better performance
  - Uses database-specific tools instead of Java
  - This option is not compatible with all databases
- Cloudera offers high-performance custom connectors for many databases

Sqoop Examples

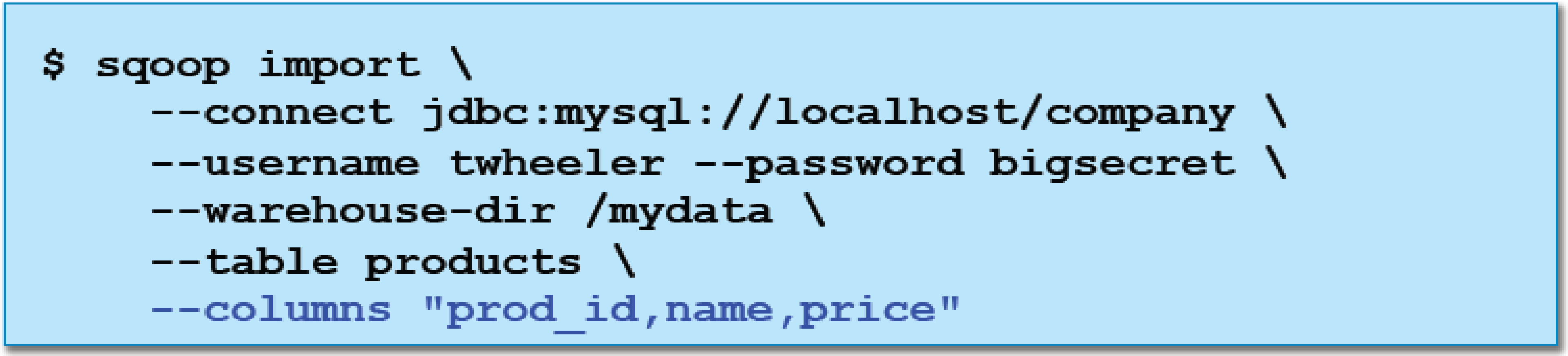
- **Import all tables from the database (fields will be tab-delimited)**



```
$ sqoop import-all-tables \
    --connect jdbc:mysql://localhost/company \
    --username twheeler --password bigsecret \
    --fields-terminated-by '\t' \
    --warehouse-dir /mydata
```

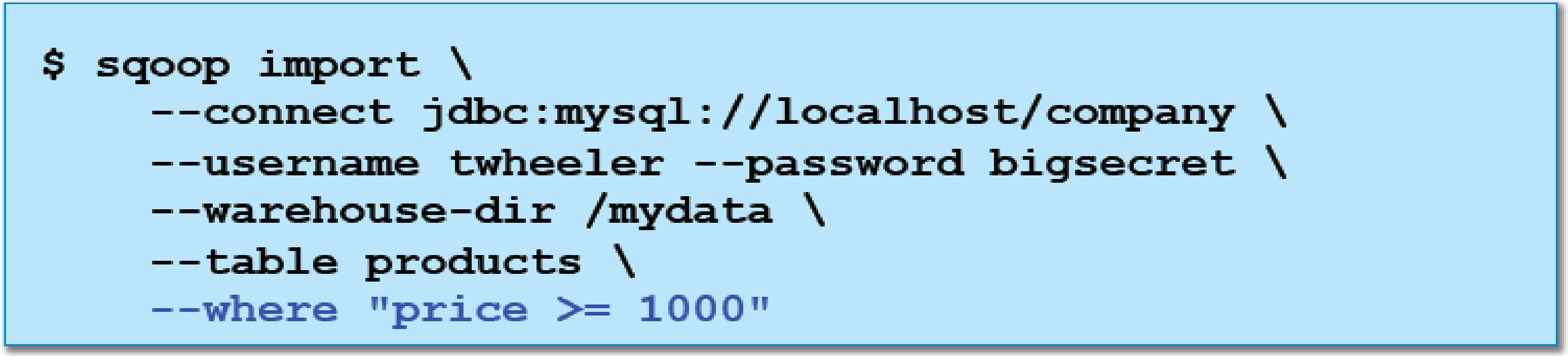
Sqoop Examples

- **Import only specified columns from products table**



```
$ sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username twheeler --password bigsecret \
    --warehouse-dir /mydata \
    --table products \
    --columns "prod_id,name,price"
```

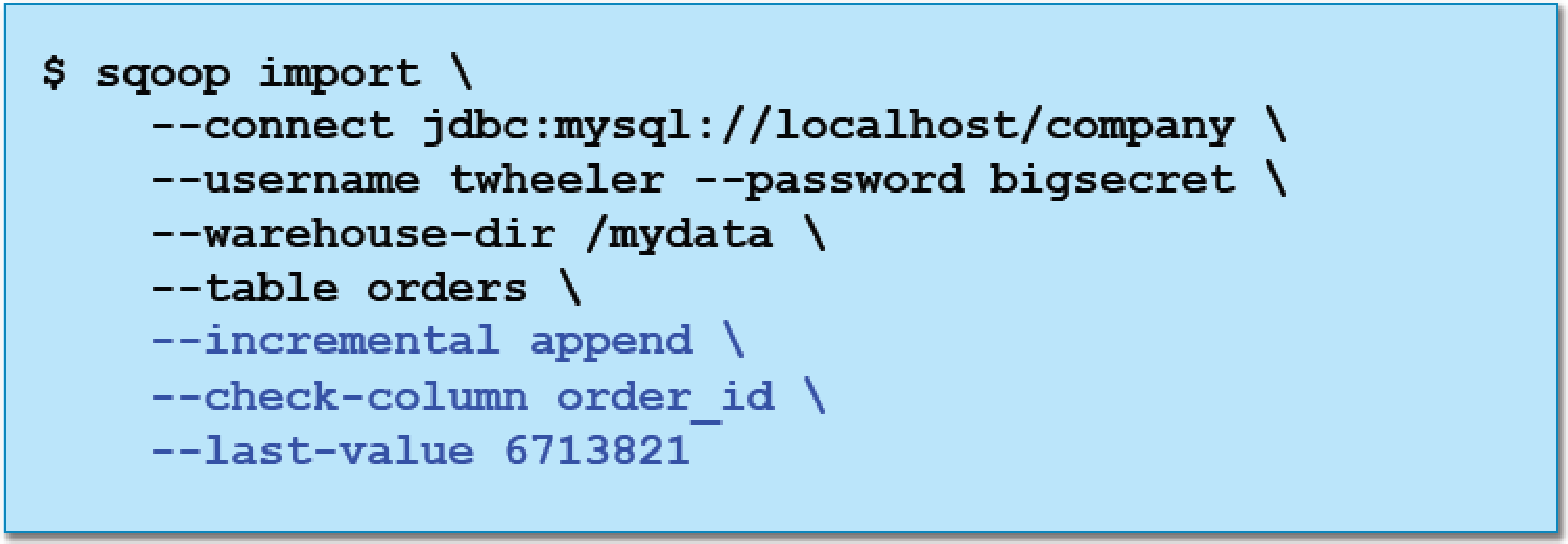
- **Import only matching rows from products table**



```
$ sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username twheeler --password bigsecret \
    --warehouse-dir /mydata \
    --table products \
    --where "price >= 1000"
```

Sqoop Examples

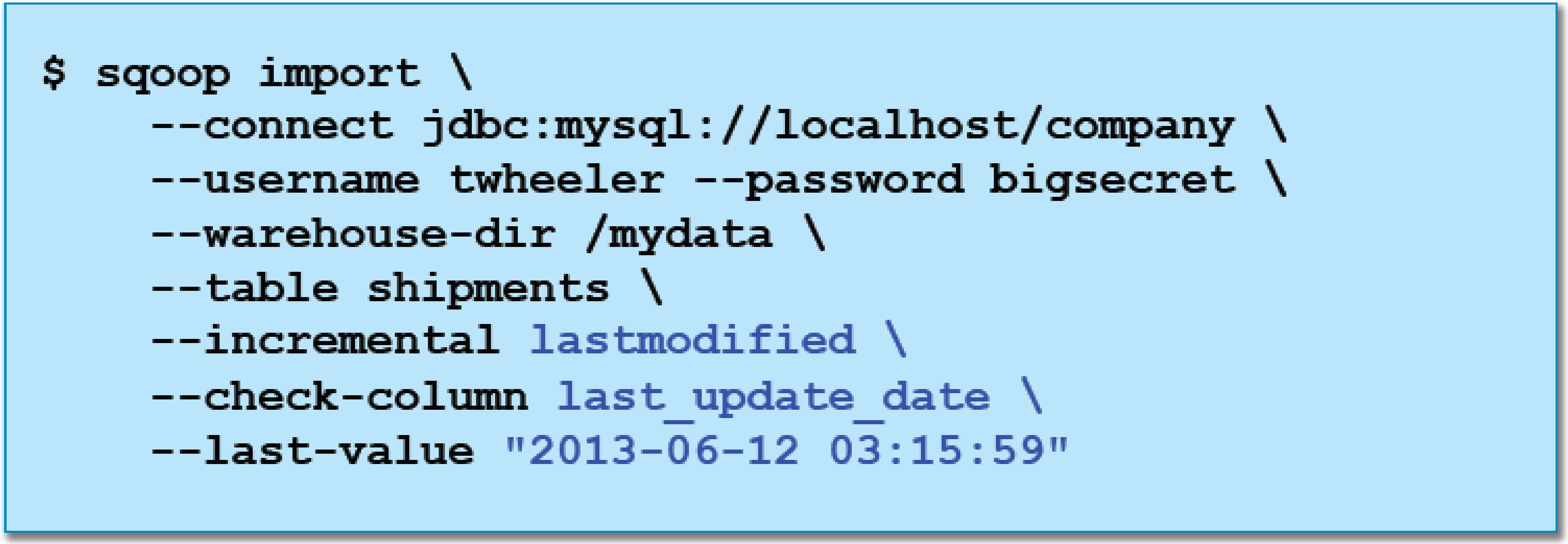
- **What if new records are added to the database?**
  - Could re-import all records, but this is inefficient
- **Sqoop's incremental append mode imports only *new* records**
  - Based on value of last record in specified column



```
$ sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username twheeler --password bigsecret \
    --warehouse-dir /mydata \
    --table orders \
    --incremental append \
    --check-column order_id \
    --last-value 6713821
```

Sqoop Examples

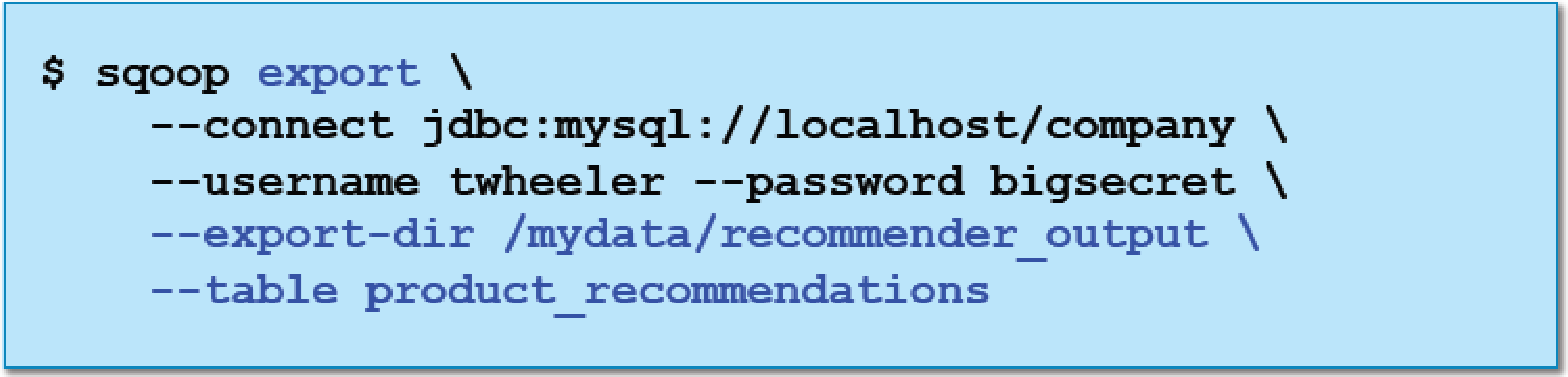
- **What if existing records are also modified in the database?**
  - Incremental append mode doesn’t handle this
- **Sqoop’s `lastmodified` append mode adds *and* updates records**
  - Caveat: You must maintain a timestamp column in your table



```
$ sqoop import \
    --connect jdbc:mysql://localhost/company \
    --username twheeler --password bigsecret \
    --warehouse-dir /mydata \
    --table shipments \
    --incremental lastmodified \
    --check-column last_update_date \
    --last-value "2013-06-12 03:15:59"
```

Sqoop Examples

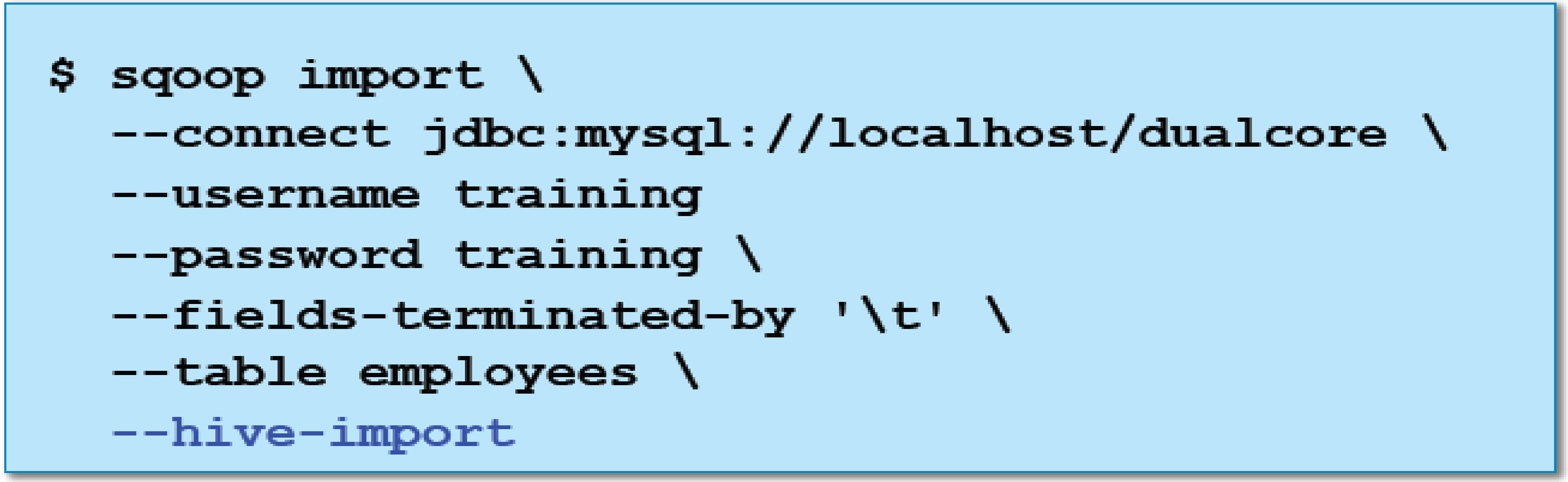
- **We've seen several ways to pull records from an RDBMS into Hadoop**
  - It is sometimes also helpful to *push* data in Hadoop back to an RDBMS
- **Sqoop supports this via `export`**



```
$ sqoop export \
    --connect jdbc:mysql://localhost/company \
    --username twheeler --password bigsecret \
    --export-dir /mydata/recommender_output \
    --table product_recommendations
```

Sqoop Examples

- **Sqoop has built-in support for importing data into Hive**
- **Just add the `--hive-import` option to your Sqoop command**
  - Creates the table in Hive (metastore)
  - Imports data from RDBMS to table's directory in HDFS



```
$ sqoop import \
   --connect jdbc:mysql://localhost/dualcore \
   --username training
   --password training \
   --fields-terminated-by '\t' \
   --table employees \
   --hive-import
```

Flume

What is Flume

Flume is a distributed, reliable, available service for efficiently moving large amounts of data as it is produced

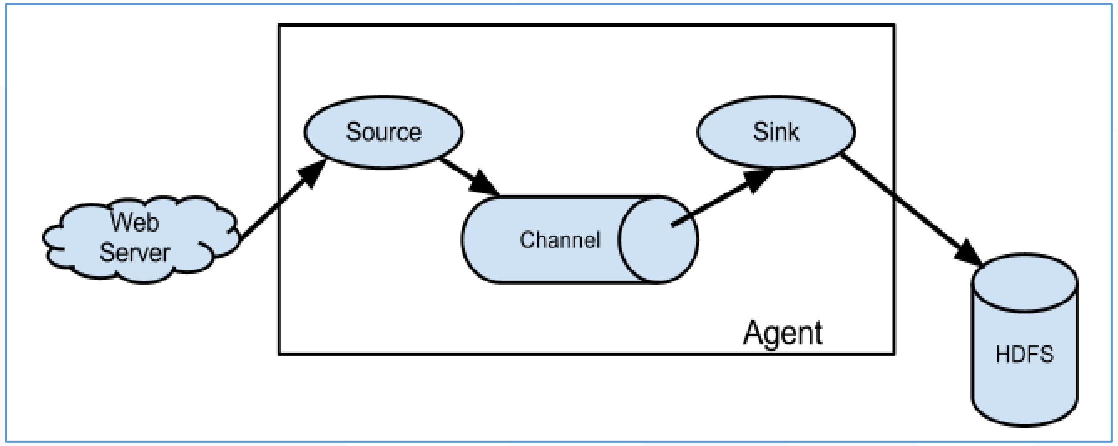
– Ideally suited to gathering logs from multiple systems and inserting them into HDFS as they are generated

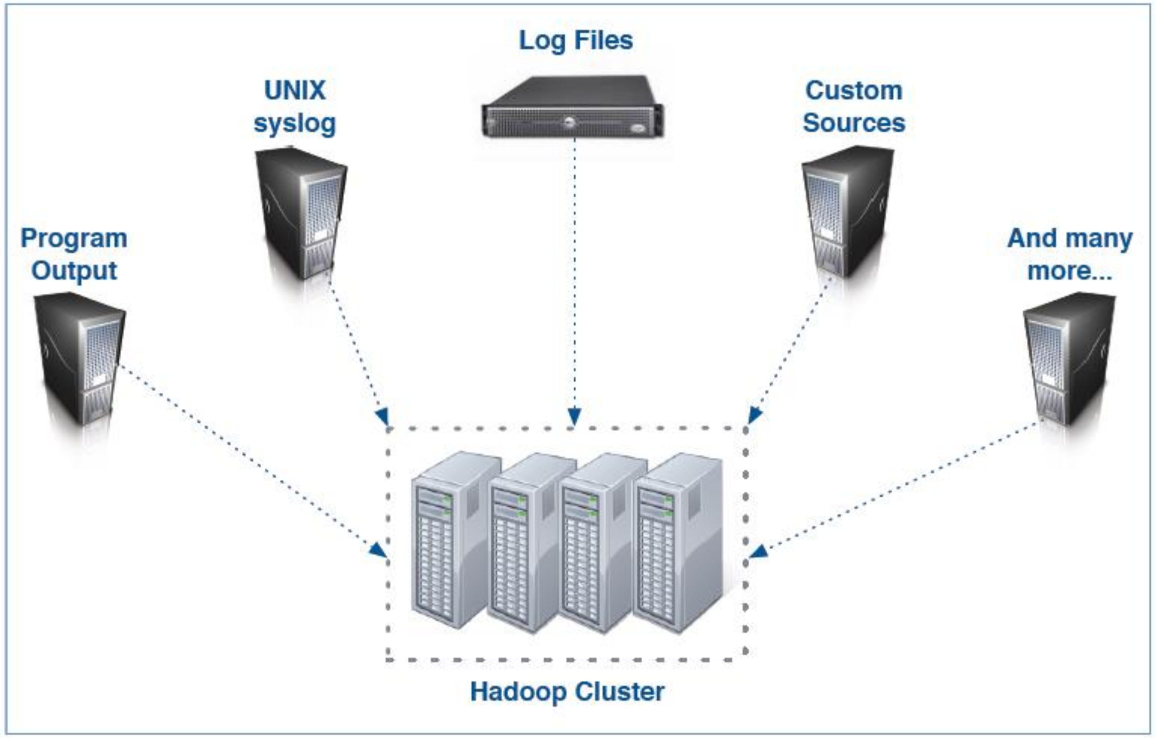
Flume is an open source Apache project

- – Initially developed by Cloudera
- – Included in CDH

Flume's design goals:

- – Reliability
- – Scalability
- – Extensibility





How Flume helps Hadoop to get data from live streaming?

Flume allows the user to do the following:

✓Flume is typically used to ingest log files from real time systems such as Web servers, firewalls, and mail servers into HDFS

✓Currently in use in many large organizations, ingesting millions of events per day

✓It acts as a buffer when the rate of incoming data exceeds the rate at which the data can be written. Thereby preventing data loss.

✓Guarantees data delivery.

✓Scales horizontally (connects commodity system in parallel) to handle additional data volume.

High Level Overview

Each Flume agent has a source and a sink

Source

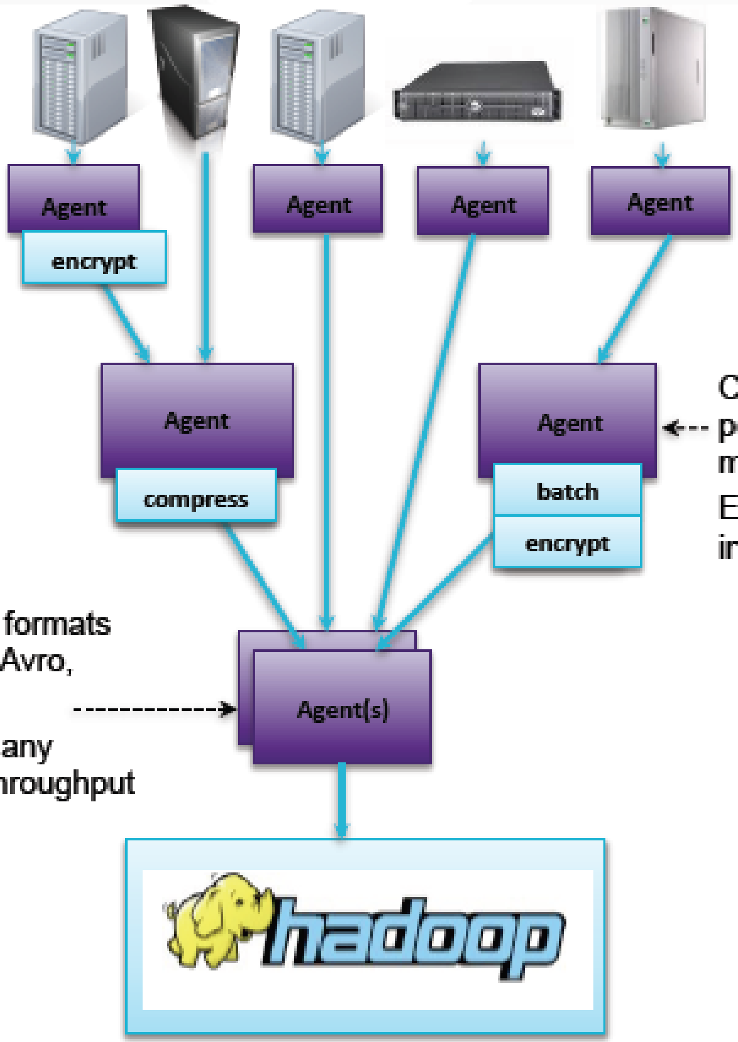
- Tells the node where to receive data from

Sink

- Tells the node where to send data to

Channel

- A queue between the Source and Sink
- Can be in memory only or ‘Durable’
- Durable channels will not lose data if power is lost



Optionally pre-process incoming data: perform transformations, suppressions, metadata enrichment

Each agent can be configured with an in-memory or durable channel.

Writes to multiple HDFS file formats (text, SequenceFile, JSON, Avro, others)

Parallelized writes across many collectors – as much write throughput as required

Essential Components Involved in Getting Data from a Live-Streaming Source

There are 3 major components, namely: Source, Channel, and Sink, which are involved in ingesting data, moving data and storing data, respectively.

Below is the breakdown of the parts applicable in this scenario:

- ✓**Event** – A singular unit of data that is transported by Flume (typically a single log entry).
- ✓**Source** – The entity through which data enters into the Flume. Sources either actively samples the data or passively waits for data to be delivered to them. A variety of sources such as log4j logs and syslogs, allows data to be collected.
- ✓**Sink** – The unit that delivers the data to the destination. A variety of sinks allow data to be streamed to a range of destinations. Example: HDFS sink writes events to the HDFS.
- ✓**Channel** – It is the connection between the Source and the Sink. The Source ingests Event into the Channel and the Sink drains the Channel.
- ✓**Agent** – Any physical Java virtual machine running Flume. It is a collection of Sources, Sinks and Channels.
- ✓**Client** – It produces and transmits the Event to the Source operating within the Agent
- ✓**Flow:** Movement of events from the point of origin to their final destination

Flume Sources & Sinks

- Avro Source
- Thrift Source
- Exec Source
- JMS Source
- Spooling Directory Source
    - Event Deserializers: LINE, AVRO, BlobDeserializer
- Twitter 1% firehose Source (experimental)
- Kafka Source
- NetCat Source
- Sequence Generator Source
- Syslog Sources
    - Syslog TCP Source, Multiport Syslog TCP Source
    - Syslog UDP Source, HTTP Source
    - JSONHandler, BlobHandler
- Stress Source
- Legacy Sources
    - Avro Legacy Source, Thrift Legacy Source
- Custom Source
- Scribe Source

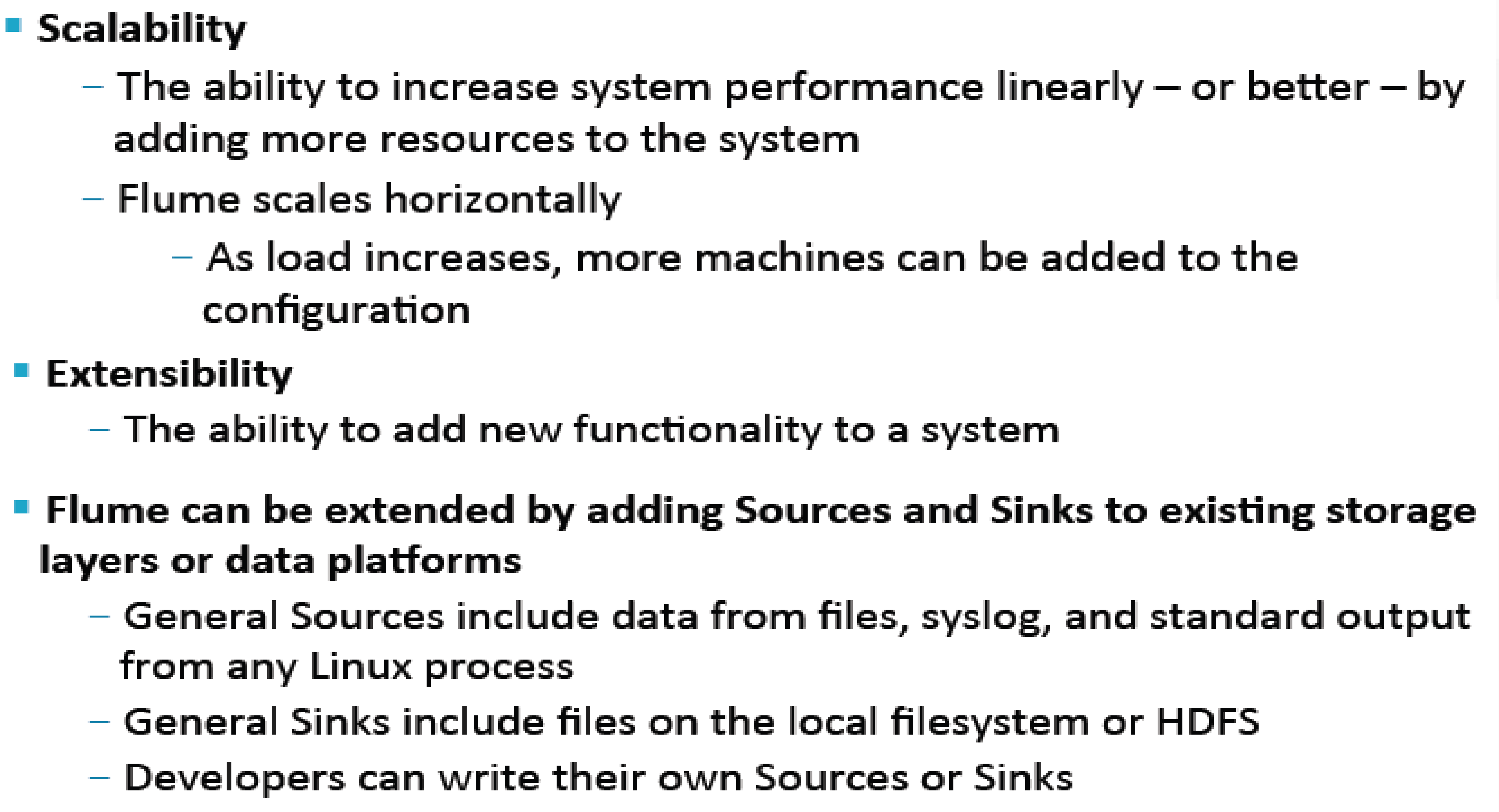
- HDFS Sink
- Hive Sink
- Logger Sink
- Avro Sink
- Thrift Sink
- IRC Sink
- File Roll Sink
- Null Sink
- HBaseSinks
    - HBaseSink
    - AsyncHBaseSink
- MorphlineSolrSink
- ElasticSearchSink
- Kite Dataset Sink
- Kafka Sink
- Custom Sink

- Memory Channel
- JDBC Channel
- Kafka Channel
- File Channel
- Spillable Memory Channel
- Pseudo Transaction Channel
- Custom Channel

Flume Design Goals: Reliability

- **Channels provide Flume's reliability**
- **Memory Channel**
  - Data will be lost if power is lost
- **Disk-based Channel**
  - Disk-based queue guarantees durability of data in face of a power loss
- **Data transfer between Agents and Channels is transactional**
  - A failed data transfer to a downstream agent rolls back and retries
- **Can configure multiple Agents with the same task**
  - e.g., 2 Agents doing the job of 1 'collector' – if one agent fails then upstream agents would fail over

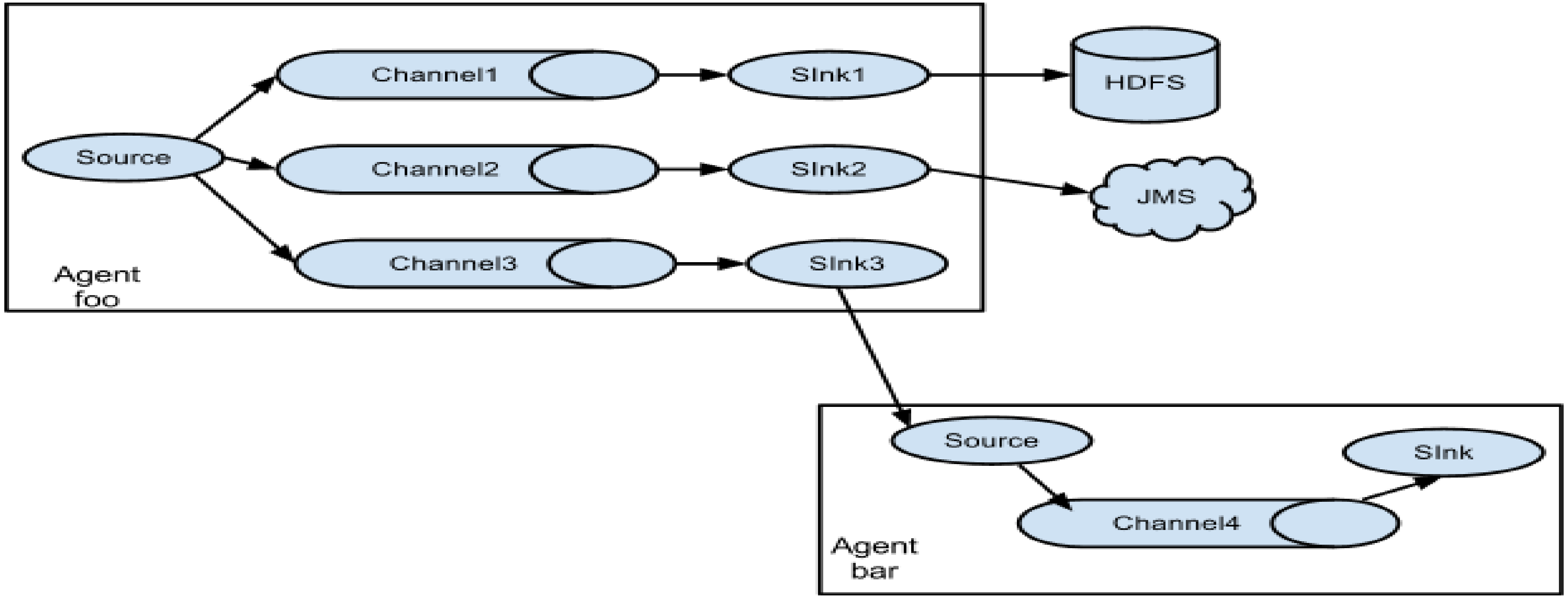
Flume Design Goals: Scalability - Extensibility



- **Scalability**
  - The ability to increase system performance linearly – or better – by adding more resources to the system
  - Flume scales horizontally
    - As load increases, more machines can be added to the configuration
- **Extensibility**
  - The ability to add new functionality to a system
- **Flume can be extended by adding Sources and Sinks to existing storage layers or data platforms**
  - General Sources include data from files, syslog, and standard output from any Linux process
  - General Sinks include files on the local filesystem or HDFS
  - Developers can write their own Sources or Sinks

Flow Pipeline

- The client transmits the event to its next hop destination
- The source receiving this event will then deliver it to one or more channels
- The channels that receive the event are drained by one are more sinks operating within same agent
- Sink will forward event to final destination



Sentiment Analysis using Social Media data

Steps for data Streaming from Twitter to HDFS

Create Access tokens from twitter.com

- ✓ **Step 1:** Open a Twitter account
- ✓ **Step 2:** Go to the following link and click on ‘create app’. (**https://apps.twitter.com/app**)
- ✓ **Step 3:** Fill in the necessary details.
- ✓ **Step 4:** Accept the agreement and click on ‘create your Twitter application’.
- ✓ **Step 5:** Go to ‘Keys and Access Token’ tab.
- ✓ **Step 6:** Copy the consumer key and the consumer secret.
- ✓ **Step 7:** Scroll down further and click on ‘create my access token’.
- ✓ Step 8: Copy the Access Token and Access token Secret.

Setting up raw data folders in HDFS and copy the data

Extract Data from Twitter using Flume

Import data into Hive and perform analysis

Integrate Excel-2013 or Tableau with Hiveserver2

High Level Steps for extracting data from twitter

Step 1: Download file flume-sources-1.0-SNAPSHOT.jar from the url
http://www.thecloudavenue.com/2013/03/analyse-tweets-using-flume-hadoop-and.html
or from http://files.cloudera.com/samples/flume-sources-1.0-SNAPSHOT.jar

create folder in myjars in /usr/lib/flume-ng.

```
#sudo mkdir /usr/lib/flume-ng/myjars
```

Put this file in /usr/lib/flume-ng/myjars

```
#sudo cp /home/cloudera/Desktop/Projects/flume-sources-1.0-SNAPSHOT.jar /usr/lib/flume-ng/myjars/
```

Step 2: Create a new app with apps.twitter.com. Generate the access tokens. Copy the consumer tokens and access tokens and use them in the twitter_conf3.conf file below.

Step 3: Create a config file called twitter.conf with below data and put it in the folder and provide full access to it (i.e full read and write access) /usr/lib/flume-ng/conf

```
#sudo cp /home/cloudera/Desktop/Projects/twitter_conf3.conf /usr/lib/flume-ng/conf/
#sudo chmod -777 /usr/lib/flume-ng/conf/twitter_conf3.conf
```

Step 4: Modify the file /usr/lib/flume-ng/conf/flume-env.sh file. Add the below line

```
FLUME_CLASSPATH="/usr/lib/flume-ng/myjars/flume-sources-1.0-SNAPSHOT.jar"
#vi /usr/lib/flume-ng/conf/flume-env.sh (enter i , copy the class path, enter ctrl+esc, enter :wq!)
```

Step 5: Run the following commands

```
cd /usr/lib/flume-ng/bin
./flume-ng agent -n TwitterAgent -c conf -f ../conf/twitter_conf3.conf   (or)
/usr/lib/flume/bin/flume-ng agent -conf ./conf/ -f/etc/flume/conf/twitter_conf.conf -Dflume.root.logger=DEBUG,  console -n TwitterAgent
```

Step 6: Check the downloaded twitter docs in HDFS

```
hdfs dfs -ls /user/cloudera/data/tweets_raw/flume-23423432453
```

Contact Us

Visit us on: http://www.analytixlabs.in/

For more information, please contact us: http://www.analytixlabs.co.in/contact-us/

Or email: info@analytixlabs.co.in

Call us we would love to speak with you: (+91) 9910509849

Join us on:

Twitter - http://twitter.com/#!/AnalytixLabs

Facebook - http://www.facebook.com/analytixlabs

LinkedIn - http://www.linkedin.com/in/analytixlabs

Blog - http://www.analytixlabs.co.in/category/blog/