Tools for Data Input/Output

HBD

Technology Map

“EXTRACT” (Acquire)

“TRANSFORM” (Process)

“LOAD” (STORE)

APACHE HBASE

Introduction

Applications implemented on **Hadoop** involve ingestion of disparate data types from **multiple sources** and with **differing requirements for frequency of ingestion**.

Common data sources for Hadoop include:

- Traditional data management systems such as relational databases and mainframes
- Logs, machine-generated data, and other forms of event data
- Files being imported from existing enterprise data storage systems

Hadoop (and other Big data tool) **start with the data in an HDFS** (or equivalent) so we need to **upload the data** to it.

Usually the source/destination systems outside the cluster are the bottleneck.

Considerations

- **Timeliness of data ingestion and accessibility:** What are the requirements around how often data needs to be ingested? How soon does data need to be available to downstream processing?
- **Incremental updates:** How will new data be added? Does it need to be appended to existing data? Or overwrite existing data?
- **Data access and processing:** Will the data be used in processing? If so, will it be used in batch processing jobs? Or is random access to the data required?

SQOOP

Command line tool designed to transfer data between Hadoop and relational databases.

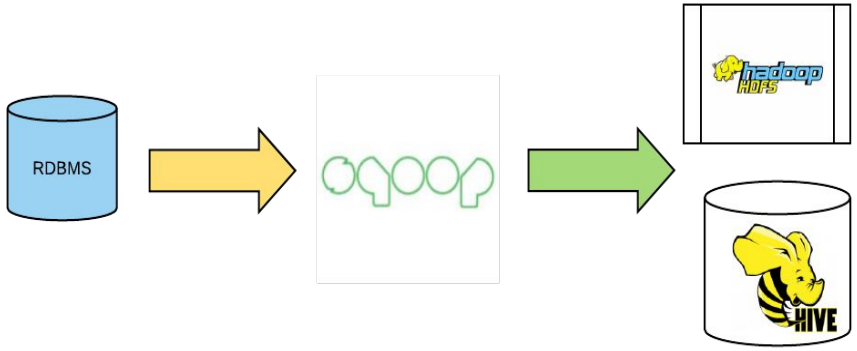
Apache Sqoop

Usual workflow:

- Import data from a relational database management system (RDBMS) into the Hadoop Distributed File System (HDFS) with Sqoop.
- Transform the data with Hadoop MapReduce/Hive/Pig/etc.
- Export data into an RDBMS with Sqoop.

Uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.

Currently Sqoop 2 is available but not widely used and not feature complete. Sqoop 2 moves from shell client submit maps job to import, to a server with map reduce jobs and security. **We will talk about Sqoop 1.**



Sqoop Import

From database to HDFS

Apache Sqoop Import

- The **input** to the import process is a **database table**.
- Sqoop will **read** the table **row-by-row into HDFS.**
- The **output** of this import process **is a set of files** containing a copy of the imported table.
- The import process **is performed in parallel**. For this reason, the output will be in **multiple files**.
- These files may be delimited **text files** (for example, with commas or tabs separating each field), **or binary** (Avro or SequenceFiles) containing serialized record data.

Apache Sqoop Import Examaple

```
$./sqoop import \
--append --check-column "modified_date" --incremental "lastmodified" \
--connect 'jdbc:mysql://db-host:3306/db' \
--username my_sql_user --password-file '/user/hadoop/my_sql_user.password' \
--table transactions --split-by "transaction_id" \
--direct -m 8 \
--hive-home "/home/hadoop/hive/" --hive-import \
--hive-table "staging.transactions" --null-string '\\N' --null-non-string '\\N'
 \
--hive-partition-key day --hive-partition-value ${current_date}
```

Apache Sqoop Import Parameters

- **--connect:** jdbc connection string for the database.
- **--username**: username to connect to the database.
- **-P|--password-file|--password**: way to supply the password for the db connection.
- **--append**: Append data to an existing dataset in HDFS.
- **--direct**: use database specific driver (e.g. mysqldump for MySQL).
- **--columns <col,col,col...>**: Columns to import from table.
- **-m,--num-mappers <n>**: Use n map tasks to import in parallel.
- **-e,--query <statement>**: Import the results of statement.
- **--table <table-name>**: Table to read.
- **--where <where clause>**: WHERE clause to use during import.
- **--hive-import,--hive-overwrite**: additional options to import to Hive.

Apache Sqoop Incremental Import

Retrieve **only rows newer** than some previously imported set of rows.

- **--check-column (col):** Specifies the column to be examined when determining which rows to import.
- **--incremental (mode):** Specifies how Sqoop determines which rows are new. Legal values for mode include **append** (if we only insert and we need to check by id) and **lastmodified** (if we update and we can check a last modified field).
- **--last-value (value):** Specifies the maximum value of the check column from the previous import (last id or last timestamp).

At the end of an incremental import, the value which should be specified as ***--last-value*** *for a subsequent import* ***is printed to the screen****.*

Apache Sqoop Import Internals

- A Java class is generated during the import process, which can encapsulate one row of the imported table.
- This class is used during the import process.
- The Java source code for this class is also provided to you, for use in subsequent MapReduce processing of the data.
- This class can serialize and deserialize data to and from the SequenceFile format.
- It can also parse the delimited-text form of a record.

Apache Sqoop import all tables in database

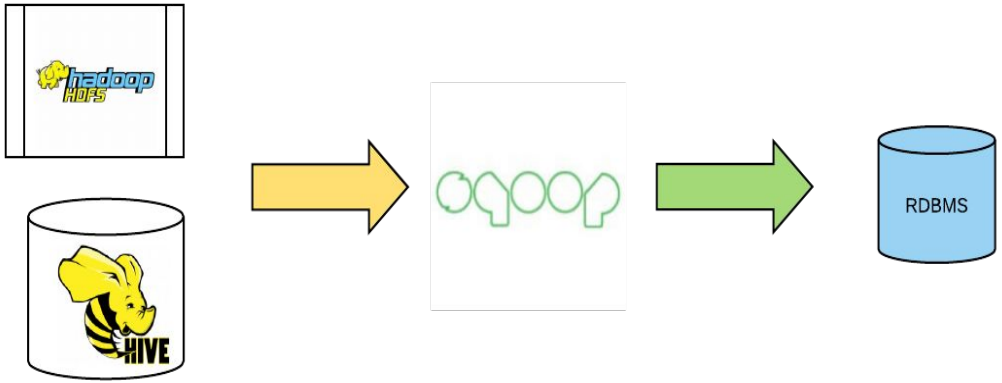
The following conditions must be met:

- Each table must have a single-column primary key.
- You must intend to import all columns of each table.
- You must not intend to use non-default splitting column, nor impose any conditions via a WHERE clause.

```
$ sqoop import-all-tables --connect jdbc:mysql://db.foo.com/corp
```

After the import you would found in HDFS the following folders (on per table):

- **/user/someuser/EMPLOYEES**
- **/user/someuser/PAYCHECKS**
- **/user/someuser/OFFICE_SUPPLIES**



Sqoop Export

From HDFS to RDBMS

Apache Sqoop Export

Sqoop's export process will:

- **Read** a set of delimited **text files from HDFS** in parallel.
- **Parse** them into records.
- **Insert** them as new rows in a **target database table**.

Considerations

- The **target table must already exist** in the database.
- The **default operation** is to transform these into a set of **INSERT statements** that inject the records into the database.
- In "**update mode**", Sqoop will generate **UPDATE statements** that replace existing records in the database.
- In "**call mode**" Sqoop will make a **stored procedure call** for each record.

Apache Sqoop Export Example

```
sqoop export \
    -Dsqoop.export.records.per.statement=50 \
    -Dsqoop.export.statements.per.transaction=50 \
    --connect 'jdbc:mysql://db-host/db'
    --username my_sql_user --password 'pass123' \
    --table transactions \
    -m 12 \
    --fields-terminated-by '\01'
    --input-null-string '\\N' --input-null-non-string '\\N' \
    --export-dir /user/hive/warehouse/database.db/transactions \
    -- --default-character-set=utf8mb4
```

Apache Sqoop Export arguments

- **--export-dir <dir>**: HDFS source path for the export.
- **-m,--num-mappers <n>**: Use n map tasks to export in parallel.
- **--table <table-name>**: Table to populate.
- **--update-mode <mode>**: Specify how updates are performed when new rows are found with non-matching keys in database. Legal values for mode include **updateonly** (default) and **allowinsert**.

Apache Sqoop Export staging table

Sqoop **breaks down export process into multiple transactions** which can mean partial data being committed if a job fails in the middle.

The **solution is specifying a staging table via the --staging-table** option which acts as an auxiliary table that is used to stage exported data. The staged data is finally moved to the destination table in a single transaction.

Considerations:

You must create **the staging table prior** to running the export job.

This table must be **structurally identical** to the target table.

This table should either **be empty** before the export job runs, or the **--clear-staging-table option** must be specified.

Apache Sqoop Export Consideration

By default **sqoop-export appends new rows** to a table using **an INSERT statement**.

The export process will fail if an INSERT statement fails.

If you specify the **--update-key** argument, Sqoop will instead **modify an existing dataset in the database**.

Each input record is treated as an UPDATE statement that modifies an existing row.

The row modified by a statement is determined by the column name(s) specified with --update-key.

Apache Sqoop Tools

```
$ sqoop <tool>
```

- **codegen**: Generate code to interact with database records.
- **create-hive-table**: Import a table definition into Hive.
- **eval**: Evaluate a SQL statement and display the results.
- **export**: Export an HDFS directory to a database table.
- **import**: Import a table from a database to HDFS.
- **import-all-tables**: Import tables from a database to HDFS.
- **list-databases**: List available databases on a server.
- **list-tables**: List available tables in a database.

Apache Sqoop Metastore

The **Sqoop Metastore** is a powerful part of Sqoop that allows you to **retain your job definitions and to easily run them anytime**.

The most important use of the metastore is for running incremental jobs and not having to keep track of the last-value imported.

```
Create job: % sqoop job --create visits -- import --connect ..
Execute job: % sqoop job --exec visits
List jobs defined in metastore: % sqoop job --list
Show saved job configuration: % sqoop job --show visits
Delete job in metastore:  % sqoop job --delete visits
```

FLUME

Stream data to HDFS

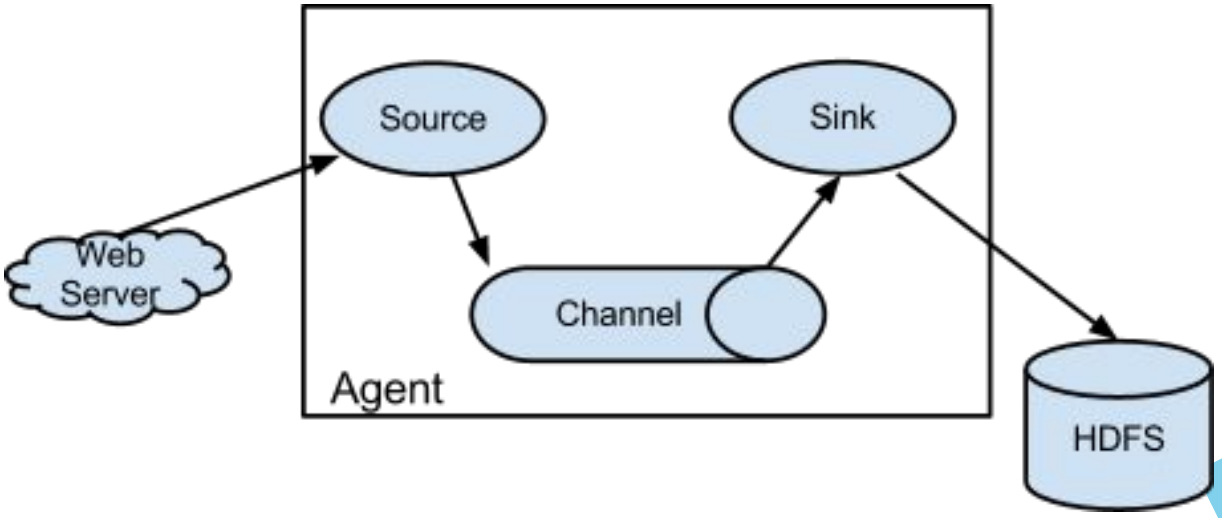
Apache Flume

Apache Flume is a **distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data** from many different sources to a centralized data store.

Features:

- **Stream of data**: Ingest streaming data from multiple sources into Hadoop for storage and analysis.
- **Insulate systems**: Buffer storage platform from transient spikes, when the rate of incoming data exceeds the rate at which data can be written to the destination.
- **Guarantee data delivery**: uses channel-based transactions to guarantee reliable message delivery.
- **Scale horizontally**: To ingest new data streams and additional volume as needed.

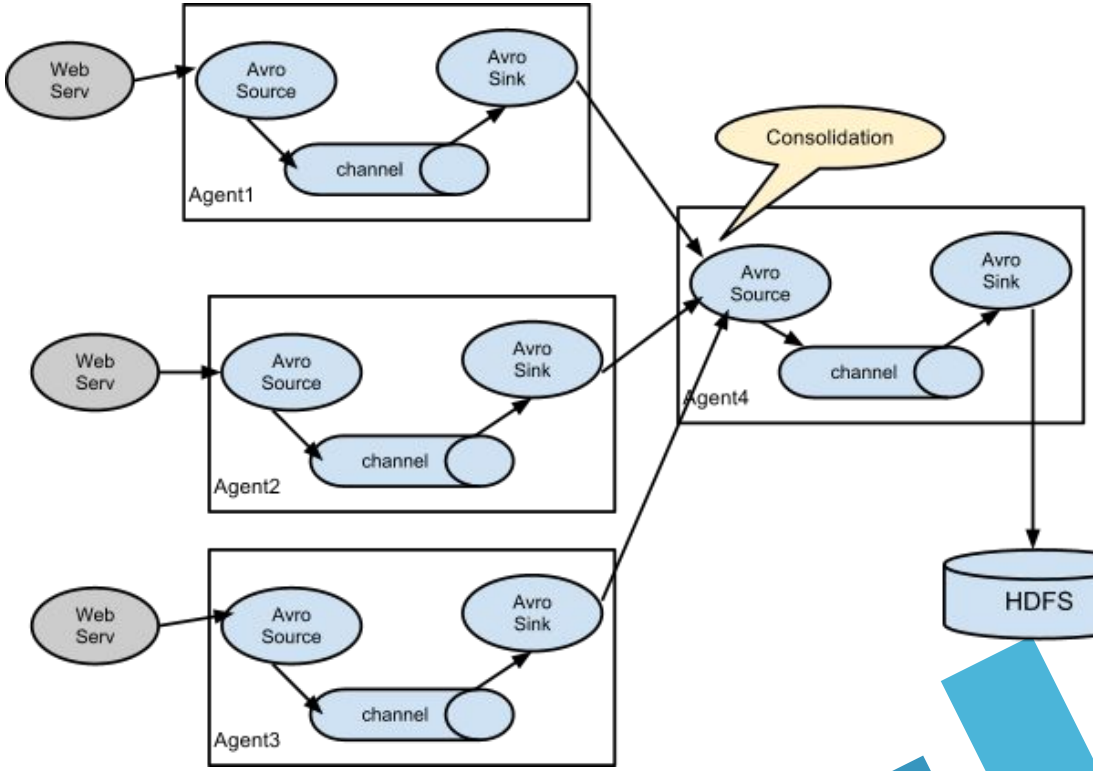
Apache Flume Agent



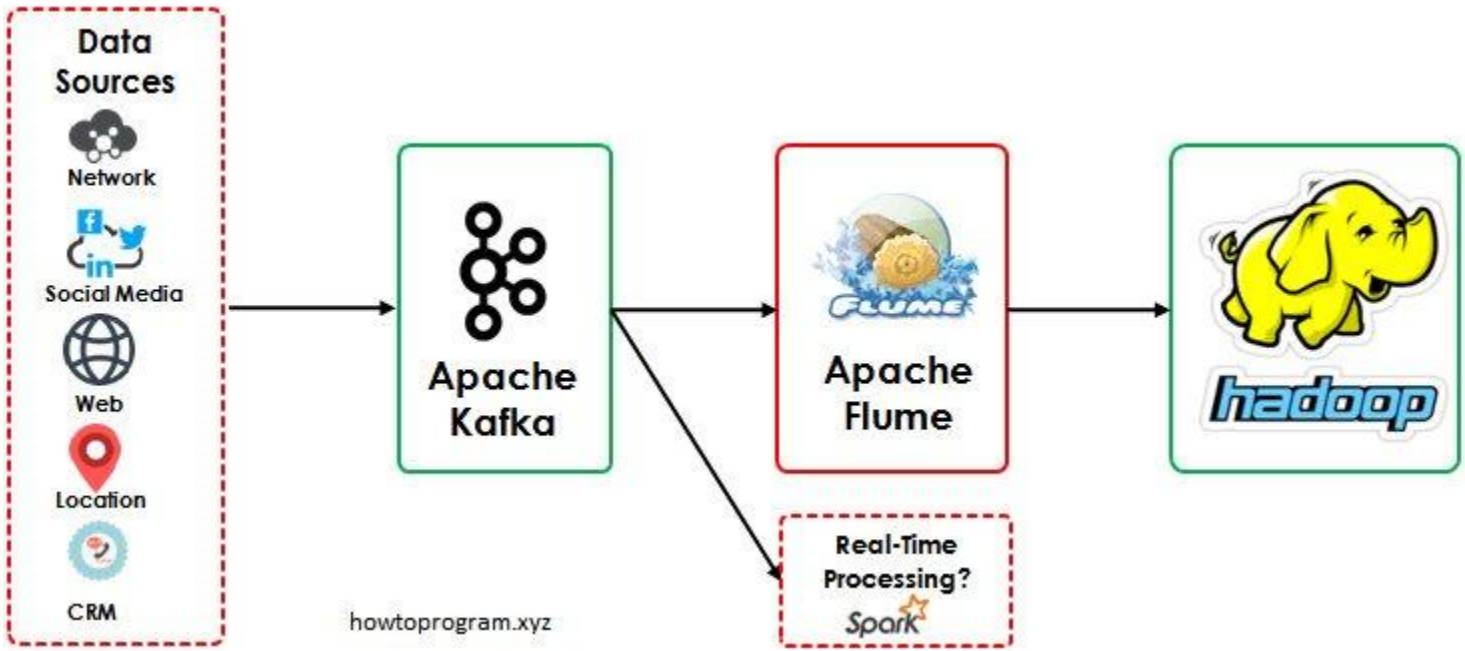
Apache Flume Components

- **Event**: a singular unit of data that is transported by Flume (typically a single log entry).
- **Source**: the entity through which data enters into Flume. Sources either actively poll for data or passively wait for data to be delivered to them. A variety of sources allow data to be collected, such as log4j logs and syslogs.
- **Sink**: the entity that delivers the data to the destination. A variety of sinks allow data to be streamed to a range of destinations. One example is the HDFS sink that writes events to HDFS.sin
- **Channel**: the conduit between the Source and the Sink. Sources ingest events into the channel and the sinks drain the channel.
- **Agent**: any physical Java virtual machine running Flume. It is a collection of sources, sinks and channels.
- **Client**: the entity that produces and transmits the Event to the Source operating within the Agent.

Apache Flume Topology

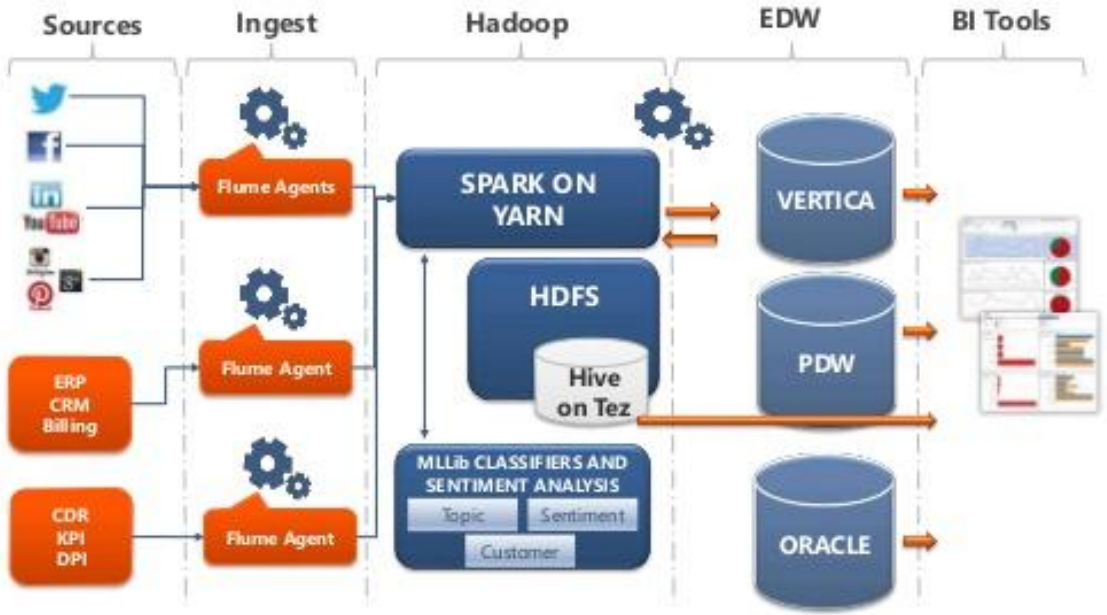


Lambda with Flume



Data Worhouses with Flume

Next Generation Architecture



Apache Flume Components Interactions

1. A **flow** in Flume **starts from the Client**.
2. The **Client** t**ransmits the Event to a Source** operating within the Agent.
3. The **Source** receiving this Event then **delivers it to one or more Channels**.
4. One or more **Sinks** operating within the same Agent **drain these Channels**.
5. **Channels decouple the ingestion rate** from drain rate using the familiar producer-consumer model of data exchange.
6. **When spikes in client side activity** cause data to be generated faster than can be handled by the provisioned destination capacity can handle, **the Channel size increases**. This allows sources to continue normal operation for the duration of the spike.
7. **The Sink of one Agent can be chained to the Source of another Agent**. This chaining enables the creation of complex data flow topologies.

Apache Flume Example netcat 2 log

```
# Archivo de configuración (example.conf)
# Name the components on this agent
    aNet2Log.sources = netcat-source
    aNet2Log.channels = memory-channel
    aNet2Log.sinks = logger-sink

# Describe/configure Source
    aNet2Log.sources.netcat-source.type = netcat
    aNet2Log.sources.netcat-source.bind = localhost
    aNet2Log.sources.netcat-source.port = 44444

# Describe the sink
    aNet2Log.sinks.logger-sink.type = logger

# Use a channel which buffers events in memory
    aNet2Log.channels.memory-channel.type = memory
    aNet2Log.channels.memory-channel.capacity = 1000
    aNet2Log.channels.memory-channel.transactionCapacity =
      100

# Bind the source and sink to the channel
    aNet2Log.sources.netcat-source.channels = memory-channel
    aNet2Log.sinks.logger-sink.channel = memory-channel
```

```
Para probar (correr en 2 consolas):
consola-1> bin/flume-ng agent --name aNet2Log --conf-file
  example.conf -Dflume.root.logger=INFO,console
consola-2> curl telnet://localhost:44444
```

Apache Flume Example netcat 2 HDFS

```
# Name the components on this agent
aNet2HDFS.sources = netcat-source
aNet2HDFS.channels = memory-channel
aNet2HDFS.sinks = hdfs

# Describe/configure Source
aNet2HDFS.sources.netcat-source.type = netcat
aNet2HDFS.sources.netcat-source.bind = localhost
aNet2HDFS.sources.netcat-source.port = 44445

# Describe the sink
  aNet2HDFS.sinks.hdfs.type = hdfs
  aNet2HDFS.sinks.hdfs.hdfs.path = hdfs:/log_data/
  aNet2HDFS.sinks.hdfs.hdfs.fileType = DataStream
  aNet2HDFS.sinks.hdfs.hdfs.writeFormat = Text
  aNet2HDFS.sinks.hdfs.hdfs.batchSize = 1000
  aNet2HDFS.sinks.hdfs.hdfs.rollSize = 0
  aNet2HDFS.sinks.hdfs.hdfs.rollCount = 10000

# Use a channel which buffers events in memory
   aNet2HDFS.channels.memory-channel.type = memory
   aNet2HDFS.channels.memory-channel.capacity = 1000
   aNet2HDFS.channels.memory-channel.transactionCapacity =
    100

# Bind the source and sink to the channel
  aNet2HDFS.sources.netcat-source.channels = memory-channel
 aNet2HDFS.sinks.hdfs.channel = memory-channel
```

Key Take Aways

Tools for ingressing and exporting data to and from HDFS **are fundamental** for any Big Data adoption strategy.

The Hadoop ecosystem offers many different **tools according to the kind of datasource** we are transferring data to and/or from.

It is important to keep in mind that **many datasources are not prepared for the level of parallelization** that Hadoop is able and therefore configurations need to be adjusted accordingly.

The tools described during this presentation are **mostly of batch nature with the exception of Apache Flume** which can handle streaming events of data.

References

- Mark Grover, Ted Malaska, Jonathan Seidman, Gwen Shapira, **Hadoop Application Architectures**, O'Reilly Media, July 2015
- http://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html
- Kathleen Ting and Jarek Jarcec Cecho, **Apache Sqoop Cookbook**, O'Reilly Media, July 2013
- http://hortonworks.com/hadoop/flume/
- https://flume.apache.org/FlumeUserGuide.html

CREDITS

Content of the slides:

- Big Data Tools - ITBA

Images:

- Big Data Tools - ITBA
- obtained from: commons.wikimedia.org

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash