BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. Vivek V. Jog
Dept. of Computer Engineering

innovate achieve lead

BITS Pilani

Pilani Campus

Big Data Systems (S1-24_CCZG522)

Lecture No.10

Flume

1. What is Flume?
    1. Overview
    2. Architecture
2. Down the rabbit hole
    1. Events
    2. Sources
    3. Channels
    4. Sinks
    5. Configuration
    6. Data manipulation with interceptors
    7. Multiplexing
    8. Customer Serializers
3. Use Cases

What is Flume?

Flume is a data collection and aggregation framework, which operates in distributed topologies.

It is stream - oriented, fault tolerant and offers linear scalability.

It offers low latency with high throughput

Its configuration is declarative, yet allows easy extensibility

It is quite mature – it has been around for quite a while, enjoys support from multiple vendors with thousands of large-scale deployments.

What is Flume?

Flume defines a simple pipeline structure with three roles:

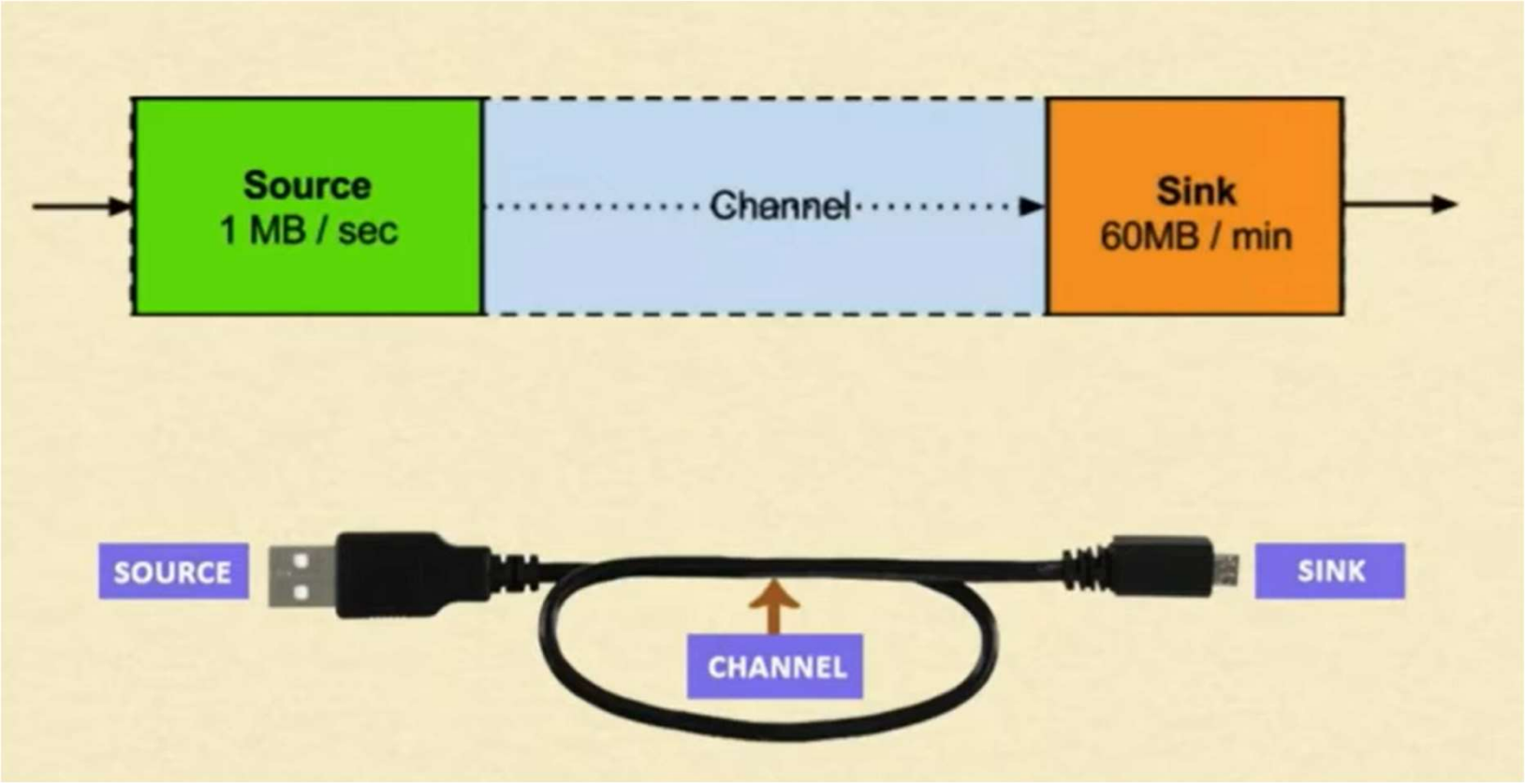
- Source
- Channel
- Sinks

Sources define where data comes from, e.g. a file, a message Queue (Kafka, JMS)

Channels are pipes connecting Sources with Sinks

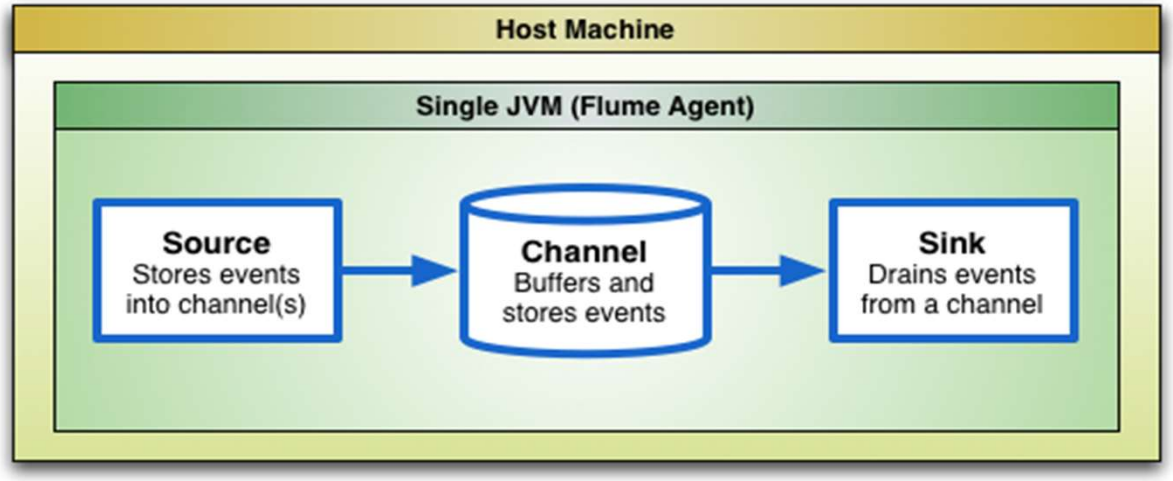
Sinks are the destination of the data pipelined from *Sources*

Channel …is like buffer

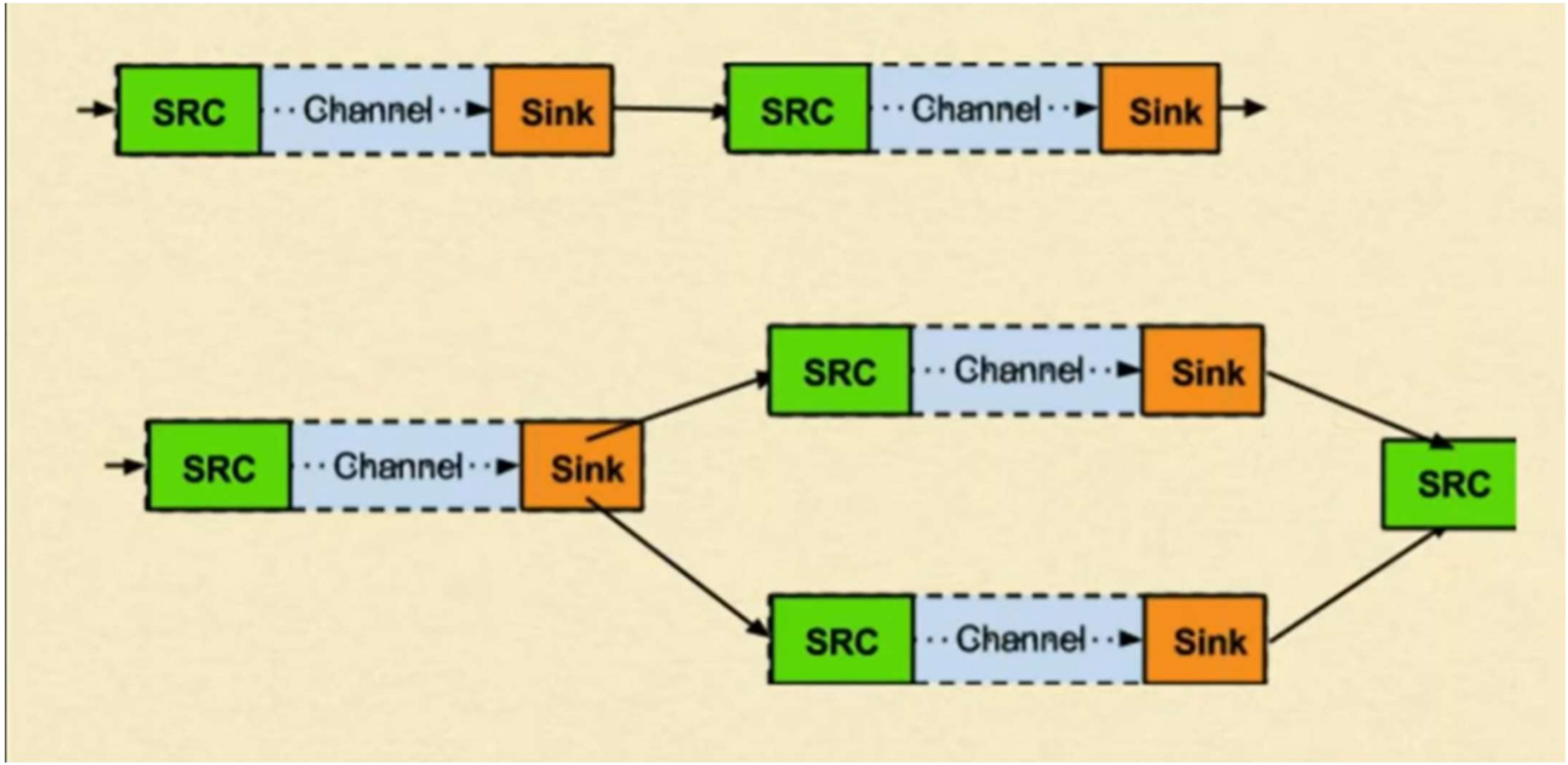


What is Flume?

These three roles run within a JVM process called *Agent*s. Data is packed into *Event*s, which are a simple Avro wrapper around any type of record, typically a line from a log file.



Transfer to Single or Multiple source



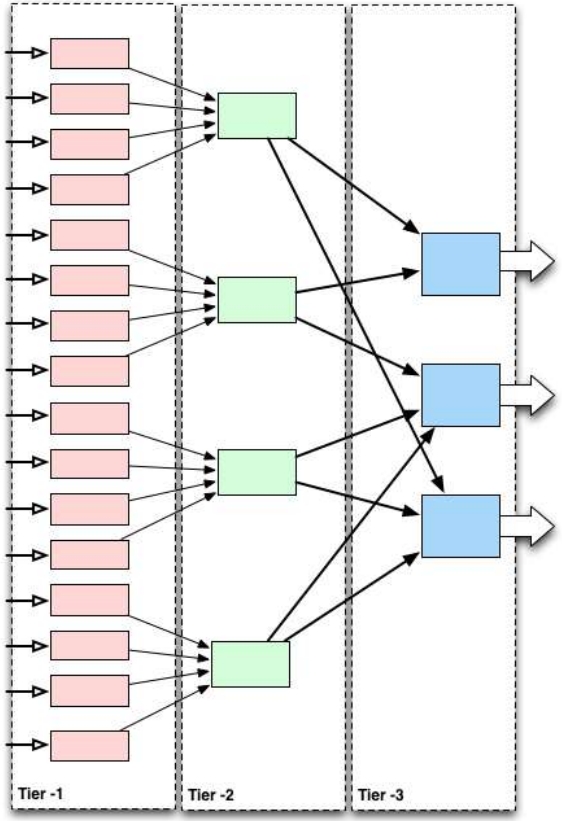
Supports a large variety of sources Including:

- tail (like unix tail -f),
- syslog,
- log4j - allowing java applications to write logs to HDFS via flume

Flume nodes can be arranged in arbitrary topologies.
Typically there is a node running on each source machine
With tiers of aggregating nodes that the data flows through on its way to HDFS.

Delivery reliability:
best-effort delivery - doesn't tolerate any node failures
end-to-end - which guarantees delivery in node failures

Contd..



Agents can be chained into a tree structure to allow multiple collection from various sources, concentrated and piped to one destination.

Shown here:

- Many agents (1 per host) at edge tier
- 4 “collector” agents in that data center
- 3 centralized agents writing to HDFS

Sources

Event-driven or polling-based

Most sources can accept batches of events

Stock source implementations:

- Spool Directory
- HTTP
- Kafka
- JMS
- Netcat
- AvroRPC

AvroRPC is use by Flume to communicate between Agents

Channels

Passive “Glue” between Sources and Sinks

Channel type determines the reliability guarantees

Stock channel types:

- JDBC – has performance issues
- Memory – lower latency for small writes, but not durable
- File – provides durability; most people use this
- New: Kafka – topics operate as channels, durable and fast

Sinks

All sinks are polling-based

Most sinks can process batches of events at a time

Stock sink implementations:

- Kafka
- HDFS
- HBase (2 variants – Sinc and Async)
- Solr
- ElasticSearch
- Avro-RPC, Thrift-RPC – Flume agent inter-connect
- File Roller – write to local filesystem
- Null, Logger– for testing purposes

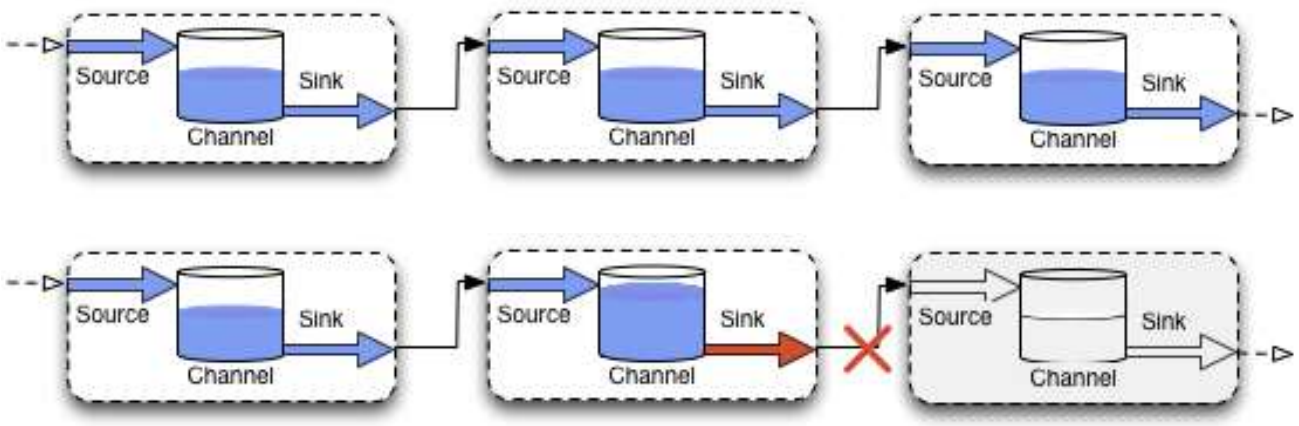
Pipeline

Source: Puts events into the local channel

Channel: Store events until someone takes them

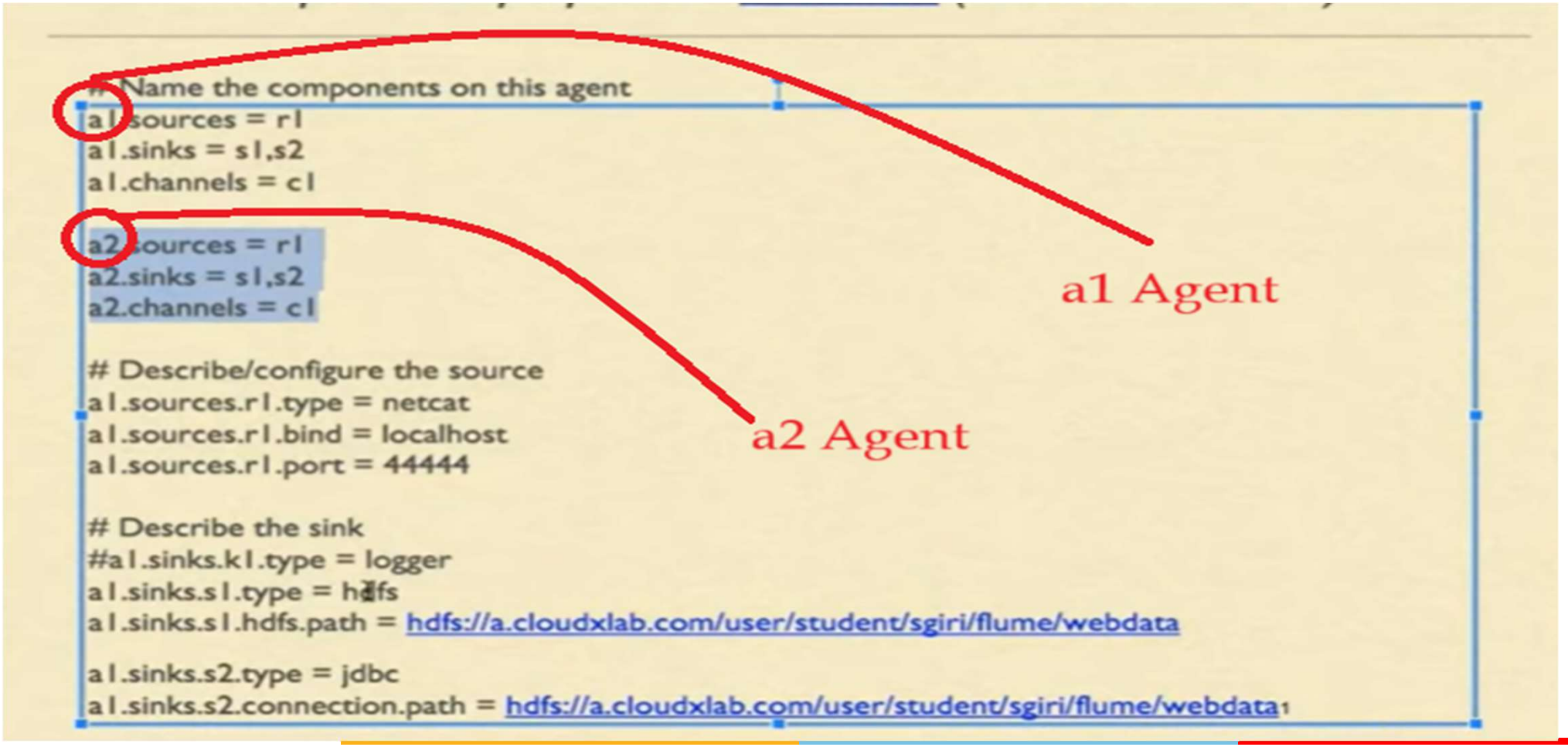
Sink: Takes events from the local channel

- On failure, sinks backoff and retry forever until success



Flume configuration…for whole cluster

Read from Port (web) Push to HDFS




```
# Name the components on this agent
a1.sources = r1
a1.sinks = s1,s2
a1.channels = c1

a2.sources = r1
a2.sinks = s1,s2
a2.channels = c1

# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

# Describe the sink
#a1.sinks.k1.type = logger
a1.sinks.s1.type = hdfs
a1.sinks.s1.hdfs.path = hdfs://a.cloudxlab.com/user/student/sgiri/flume/webdata

a1.sinks.s2.type = jdbc
a1.sinks.s2.connection.path = hdfs://a.cloudxlab.com/user/student/sgiri/flume/webdata1
```

Config File

```
a1.sources = r1
a1.sinks = hdfs-Cluster1-sink
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

# Describe the sink
#a1.sinks.k1.type = logger
a1.sinks.hdfs-Cluster1-sink.type = hdfs
a1.sinks.hdfs-Cluster1-sink.hdfs.path = hdfs:///user/sandeepgiri9034/flume-webdata

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
#a1.sinks.k1.channel = c1
a1.sinks.hdfs-Cluster1-sink.channel = c1
```

Configuration

Declarative configuration with property files

- Define your source
- Define the sink
- Declare the channel for source and sink

```
####### Spool Config

SpoolAgent.sources = MySpooler
SpoolAgent.channels = MemChannel
SpoolAgent.sinks = HDFS

SpoolAgent.channels.MemChannel.type = memory
SpoolAgent.channels.MemChannel.capacity = 500
SpoolAgent.channels.MemChannel.transactionCapacity = 200

SpoolAgent.sources.MySpooler.channels = MemChannel
SpoolAgent.sources.MySpooler.type = spooldir
SpoolAgent.sources.MySpooler.spoolDir = /var/log/
SpoolAgent.sources.MySpooler.fileHeader = false

SpoolAgent.sinks.HDFS.channel = MemChannel
SpoolAgent.sinks.HDFS.type = hdfs
SpoolAgent.sinks.HDFS.hdfs.path = hdfs://cluster/data/spool/
SpoolAgent.sinks.HDFS.hdfs.fileType = DataStream
SpoolAgent.sinks.HDFS.hdfs.writeFormat = Text
SpoolAgent.sinks.HDFS.hdfs.batchSize = 100
SpoolAgent.sinks.HDFS.hdfs.rollSize = 0
SpoolAgent.sinks.HDFS.hdfs.rollCount = 0
SpoolAgent.sinks.HDFS.hdfs.rollInterval = 300
```

Interceptors

Incoming data can be transformed and enriched at the source

One or more interceptors can modify your events and set headers

These headers can be used for sorting within sinks and routing to different sinks

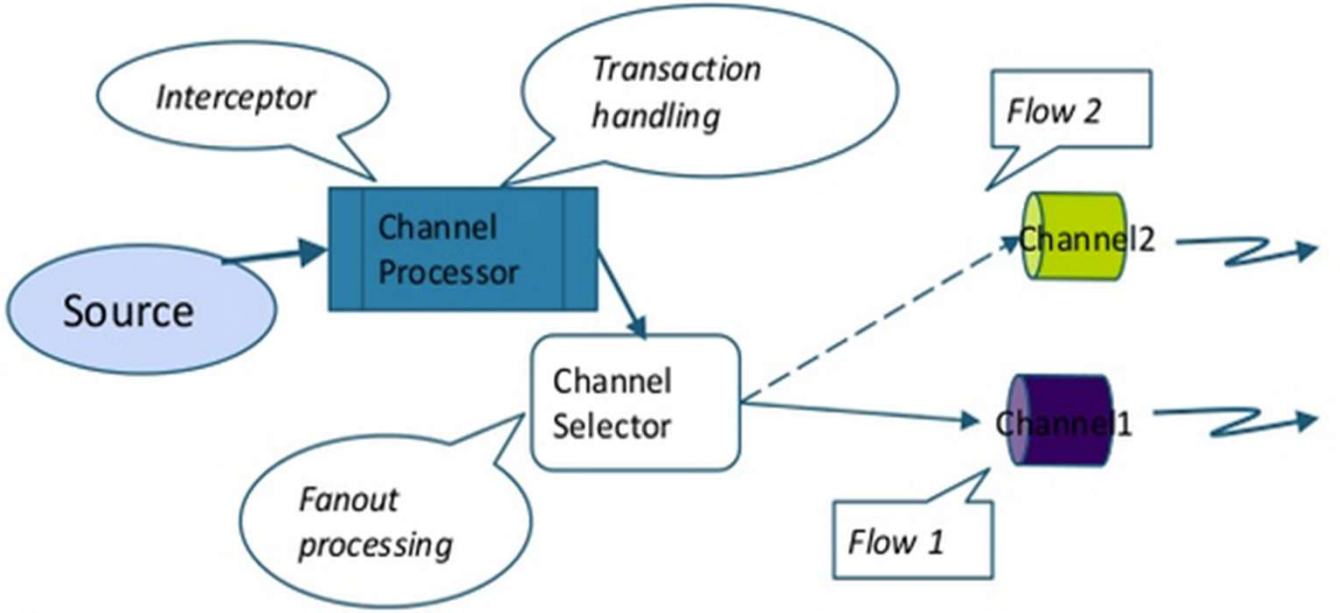
Stock Interceptor Implementations

- Timestamp (adds a timestamp header)
- Hostname (adds the flume host as a header)
- Regex (extracts the applied regex to a header)
- Static (sets a static constant header)

Selectors

Multiplexing selectors allows header sensitive dispatch to specific destination sinks.

Replicating selectors allow simultaneous writes to multiple sinks.



Serializers

Serializers customize the format written to sink.

Built-in serializers are generally fine for text

More complicated use case such as Avro require customer serializers

HDFS, HBase and FileRoller Sinks support customer Serializers.

Custom Serializers implement specific interfaces / extend abstract classes:

- AsyncHbaseEventSerializer (HBase)
- AbstractAvroEventSerializer (HDFS)
- EventSerializer (Generic)

Some practical

Key word

Directory

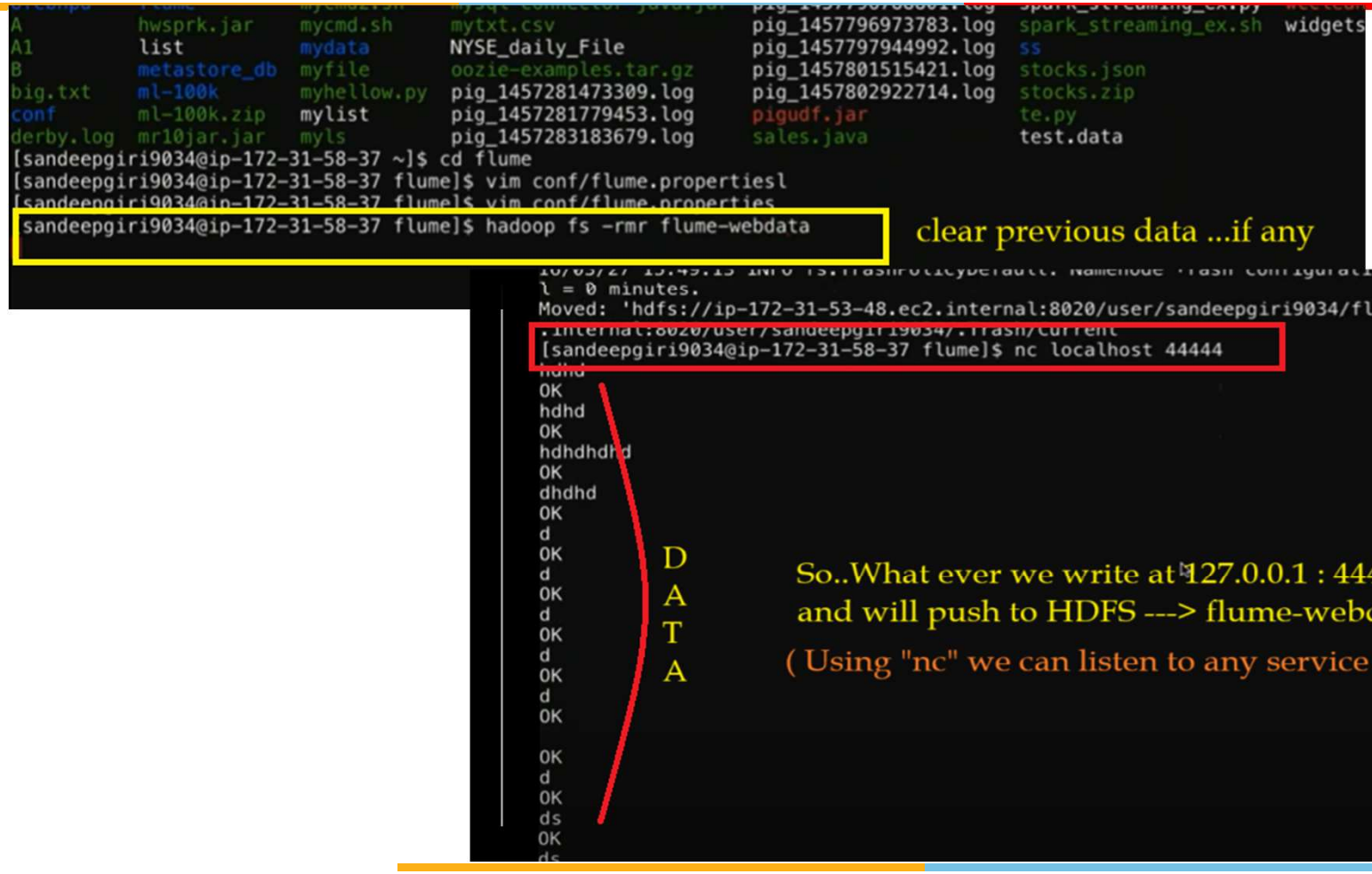
File

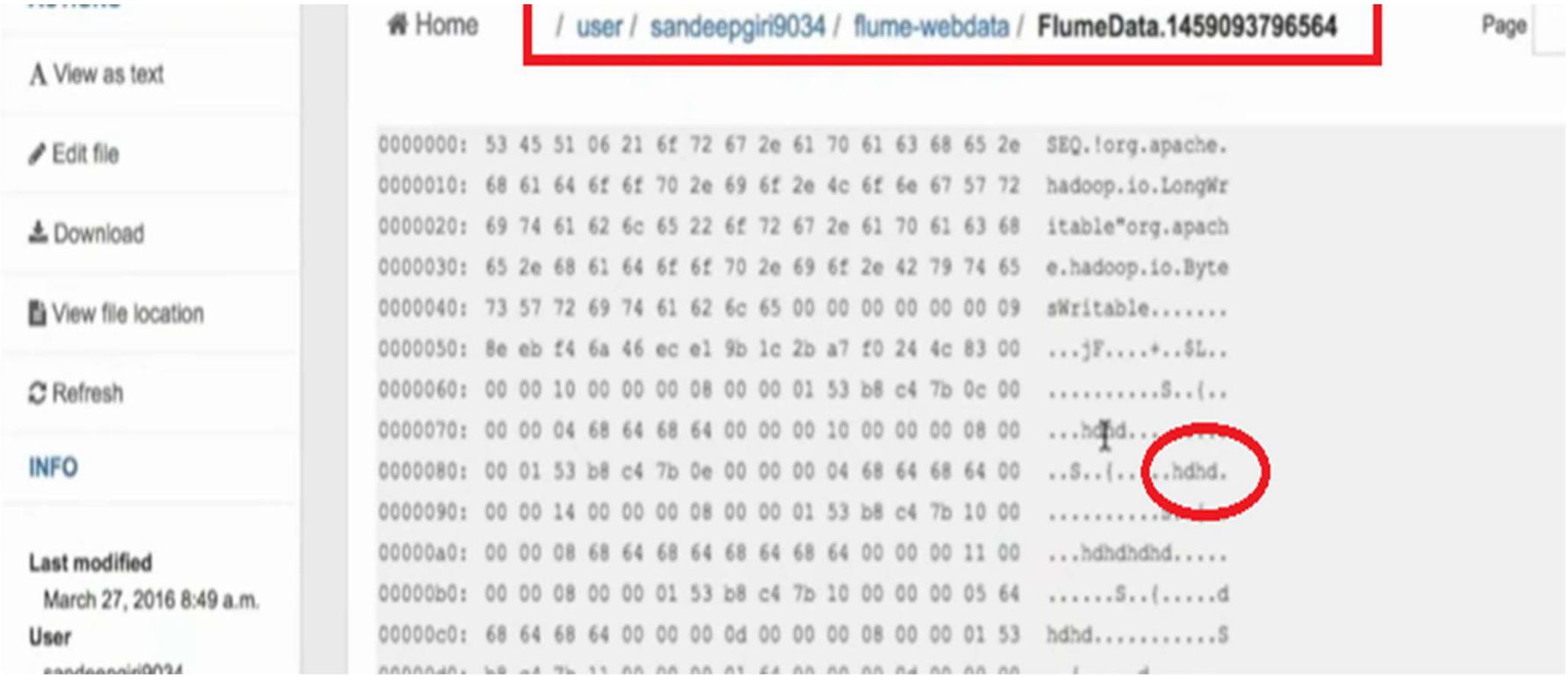
Agent Name

```
flume-ng agent --conf conf --conf-file conf/flume.properties -name a1 Dflume.root.logger=INFO,console
```

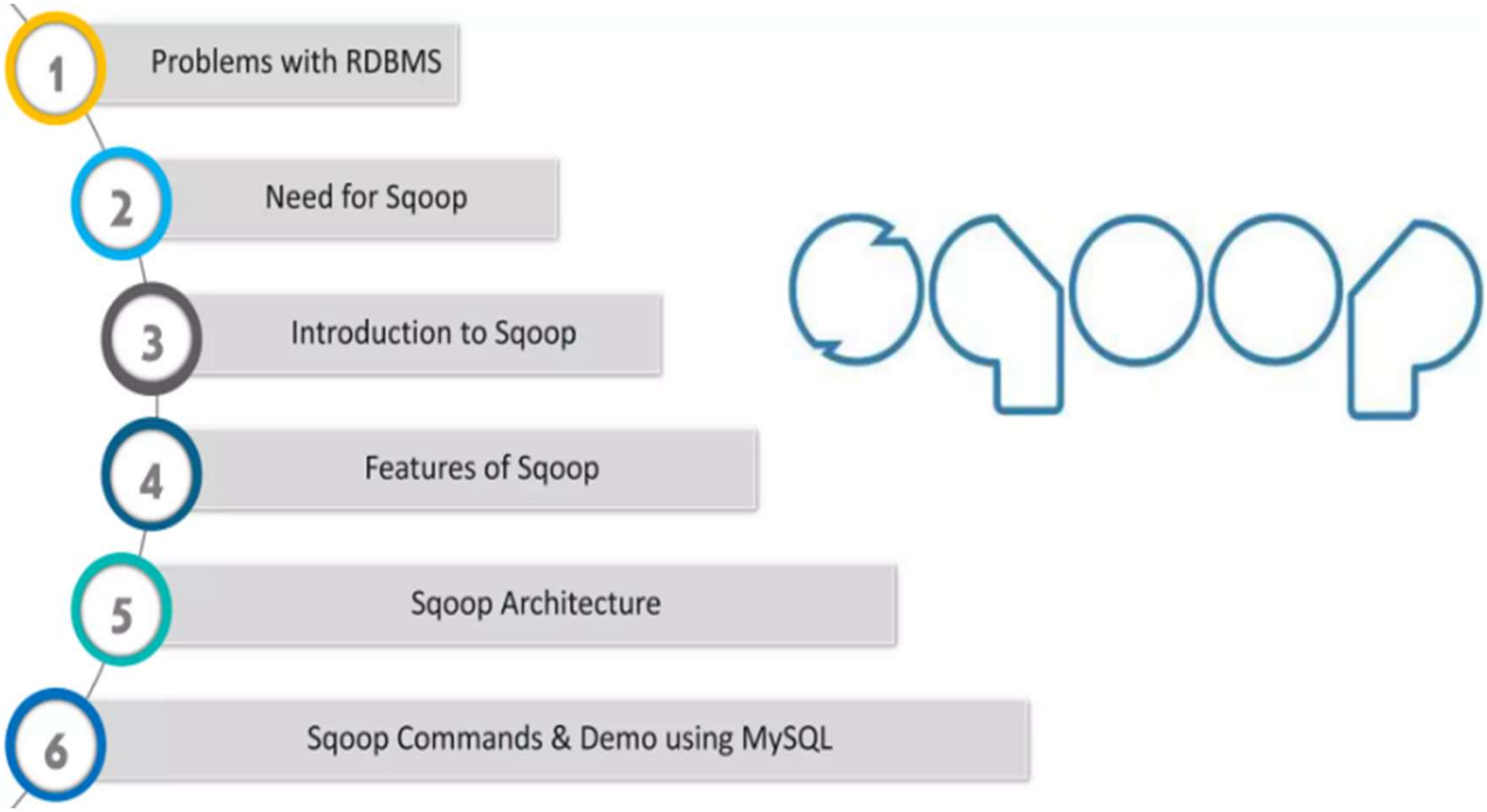
```
16/03/27 15:46:06 INFO source.NetcatSource: Source starting
16/03/27 15:46:06 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SINK, name: hdfs-Cluster1-s
nk: Successfully registered new MBean.
16/03/27 15:46:06 INFO source.NetcatSource: Created serverSocket:sun.nio.ch.ServerSocketChannelImpl[/127.0.0.1:44444]
```

Socket is created that will continuously listen

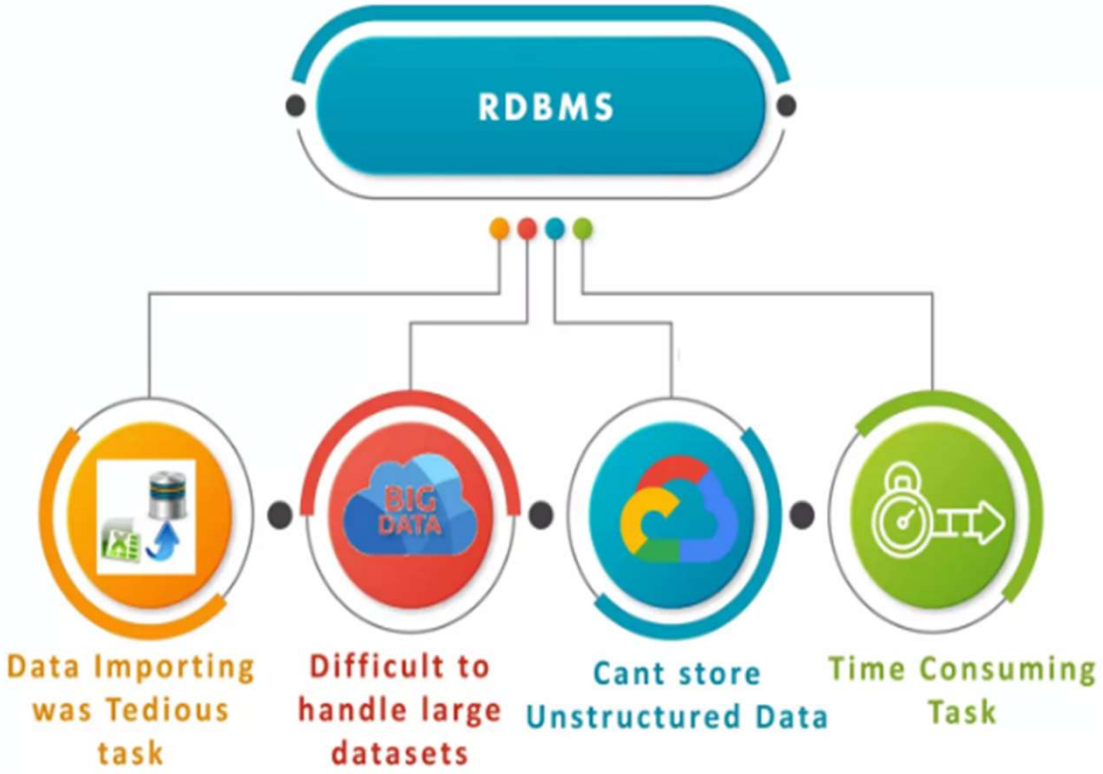

A hwsprk.jar mycmd.sh mytxt.csv pig_1457796973783.log spark_streaming_ex.sh widgets
A1 list mydata NYSE_daily_File pig_1457797944992.log ss
B metastore_db myfile oozie-examples.tar.gz pig_1457801515421.log stocks.json
big.txt ml-100k myhellow.py pig_1457281473309.log pig_1457802922714.log stocks.zip
conf ml-100k.zip mylist pig_1457281779453.log pigudf.jar te.py
derby.log mr10jar.jar myls pig_1457283183679.log sales.java test.data
[sandeepgiri9034@ip-172-31-58-37 ~]$ cd flume
[sandeepgiri9034@ip-172-31-58-37 flume]$ vim conf/flume.propertiesl
sandeepgiri9034@ip-172-31-58-37 flume]$ hadoop fs -rmr flume-webdata
clear previous data ...if any
l = 0 minutes.
Moved: 'hdfs://ip-172-31-53-48.ec2.internal:8020/user/sandeepgiri9034/flume-webdata' to trash at: hdfs://ip-172-31-53-48.
[sandeepgiri9034@ip-172-31-58-37 flume]$ nc localhost 44444
OK
hdhd
OK
hdhdhdhd
OK
dhdhd
OK
d
OK
d
OK
d
OK
d
OK
d
OK
OK
d
OK
ds
OK
DATA
So..What ever we write at 127.0.0.1 : 4444 our agent will read that
and will push to HDFS ---> flume-webdata
(Using "nc" we can listen to any service .....may be web or anything)


Home
/ user / sandeepgiri9034 / flume-webdata / FlumeData.1459093796564
Page
View as text
Edit file
Download
View file location
Refresh
INFO
Last modified
March 27, 2016 8:49 a.m.
User
0000000: 53 45 51 06 21 6f 72 67 2e 61 70 61 63 68 65 2e SEQ.!org.apache.
0000010: 68 61 64 6f 6f 70 2e 69 6f 2e 4c 6f 6e 67 57 72 hadoop.io.LongWr
0000020: 69 74 61 62 6c 65 22 6f 72 67 2e 61 70 61 63 68 itable"org.apach
0000030: 65 2e 68 61 64 6f 6f 70 2e 69 6f 2e 42 79 74 65 e.hadoop.io.Byte
0000040: 73 57 72 69 74 61 62 6c 65 00 00 00 00 00 00 09 sWritable.....
0000050: 8e eb f4 6a 46 ec e1 9b 1c 2b a7 f0 24 4c 83 00 ...jF....+..$L..
0000060: 00 00 10 00 00 00 08 00 00 01 53 b8 c4 7b 0c 00 ....S..{..
0000070: 00 00 04 68 64 68 64 00 00 00 10 00 00 00 08 00 ...hdhd.....
0000080: 00 01 53 b8 c4 7b 0e 00 00 00 04 68 64 68 64 00 ..S..{.....hdhd.
0000090: 00 00 14 00 00 00 08 00 00 01 53 b8 c4 7b 10 00 ....S..{..
00000a0: 00 00 08 68 64 68 64 68 64 68 64 00 00 00 11 00 ...hdhdhdhd.....
00000b0: 00 00 08 00 00 01 53 b8 c4 7b 10 00 00 00 05 64 ......S..{.....d
00000c0: 68 64 68 64 00 00 00 0d 00 00 00 08 00 00 01 53 hdhd...........S

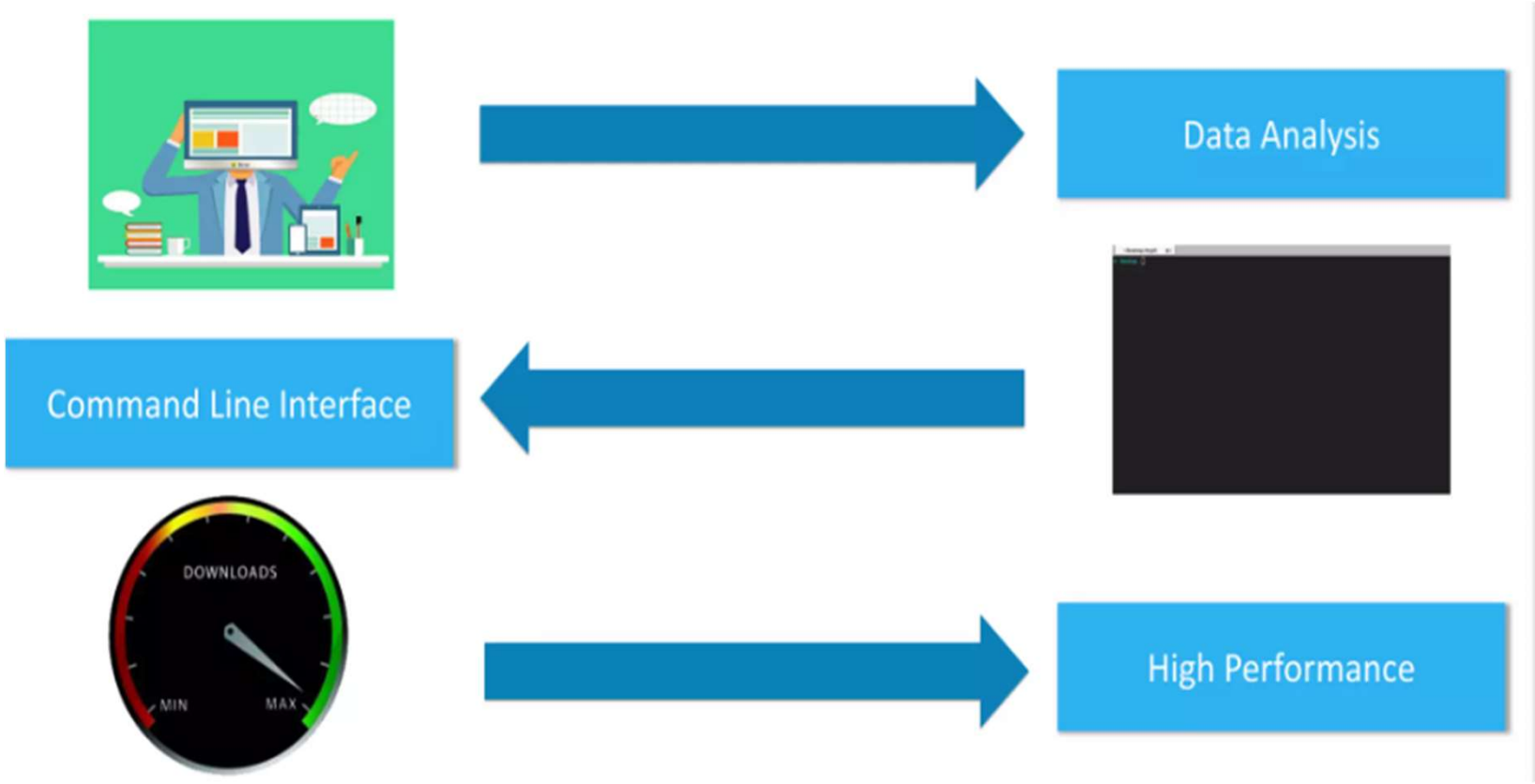
Sqoop


1
Problems with RDBMS
2
Need for Sqoop
3
Introduction to Sqoop
4
Features of Sqoop
5
Sqoop Architecture
6
Sqoop Commands & Demo using MySQL

Problem with RDBMS



Why Sqoop?


Data Analysis
Command Line Interface
DOWNLOADS
MIN
MAX
High Performance

Why Sqoop cont..

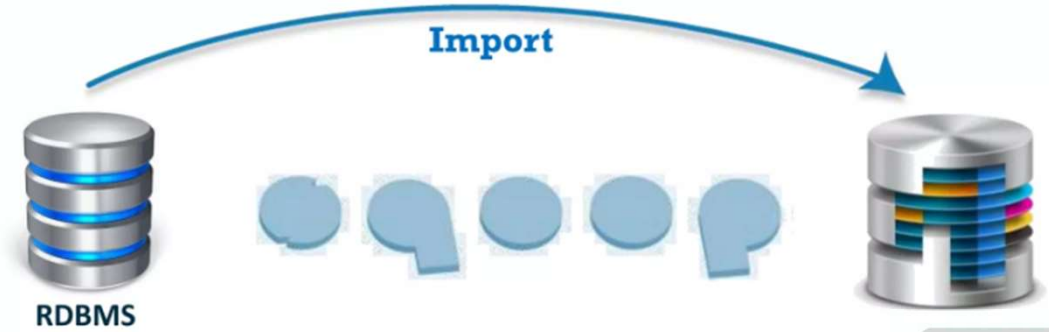
- SQL Servers are already deployed opulent worldwide
- Nightly processing is done on SQL servers for years
- As Hadoop making ways into enterprise, there was a need to move certain part of data from traditional SQL DB (RD) to Hadoop
- Transferring data using scripts is inefficient and time consuming.
- Traditional DB already have reporting, data visualization etc. applications built in enterprise
  - Bringing processed data from Hadoop to those application is the need

Why Sqoop cont..

- From RDB to Hadoop
  - Users must consider details like ensuring consistency of data, the consumption of production system resources, data preparation for provisioning downstream pipeline.
- Hadoop to RDB
  - Directly accessing data residing on external systems from within the map reduce applications complicates applications and exposes the production system to the risk of excessive load originating from cluster nodes.

What is Sqoop?

Tool used to transfer bulk data between HDFS & Relational Database Servers



Tool used to transfer bulk data between HDFS & Relational Database Servers

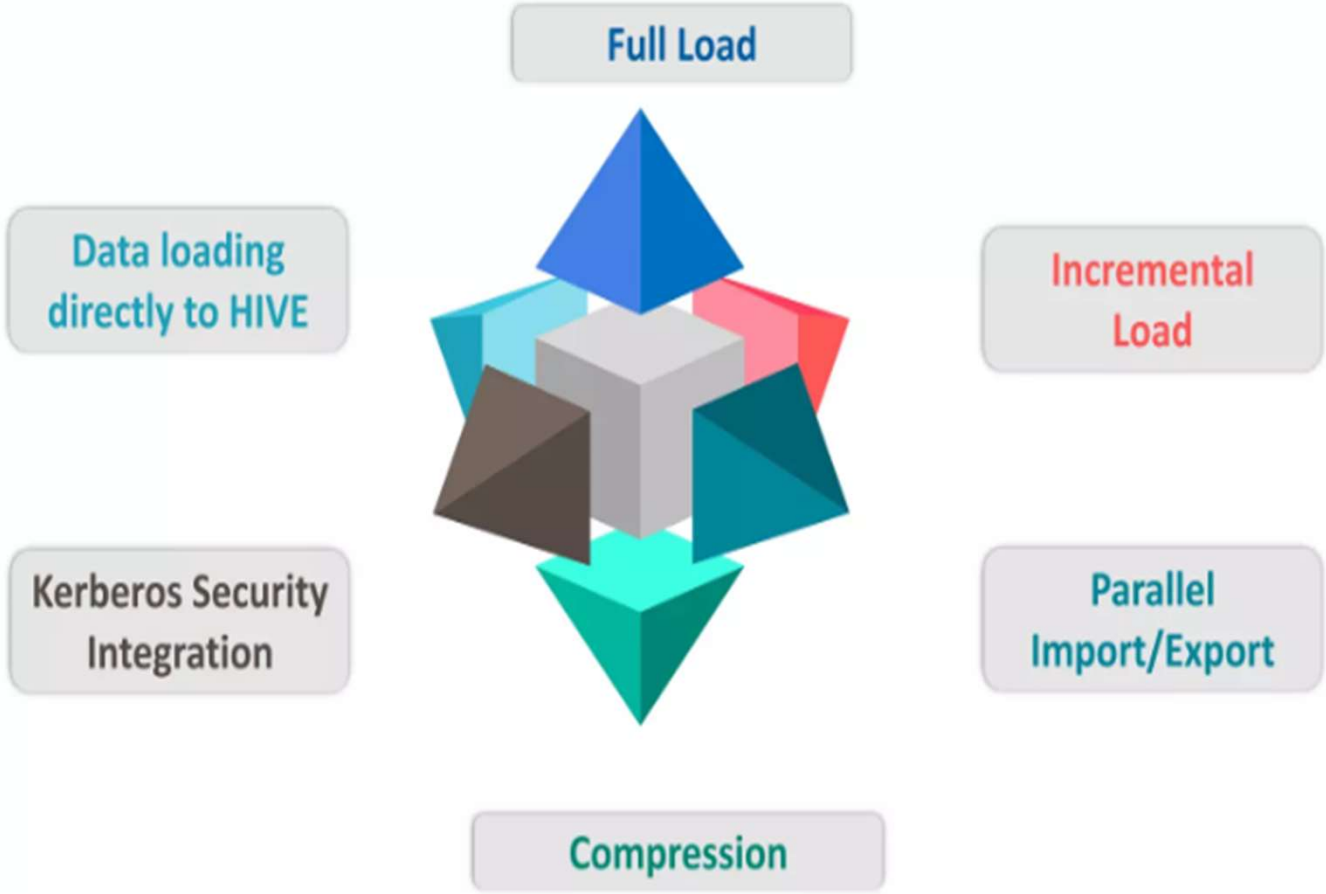
What Sqoop Provides?

- Sqoop allows easy import and export of data from structured data stores
  - RD, Enterprise data warehouses, and NoSQL systems
- Provision data from external system on to HDFS
  - Once data is moved populate tables in Hive and HBase.
- Sqoop integrates with Oozie, allowing you to schedule and automate import and export tasks.
- Sqoop uses a connector based architecture which supports plugins that provide connectivity to new external systems.

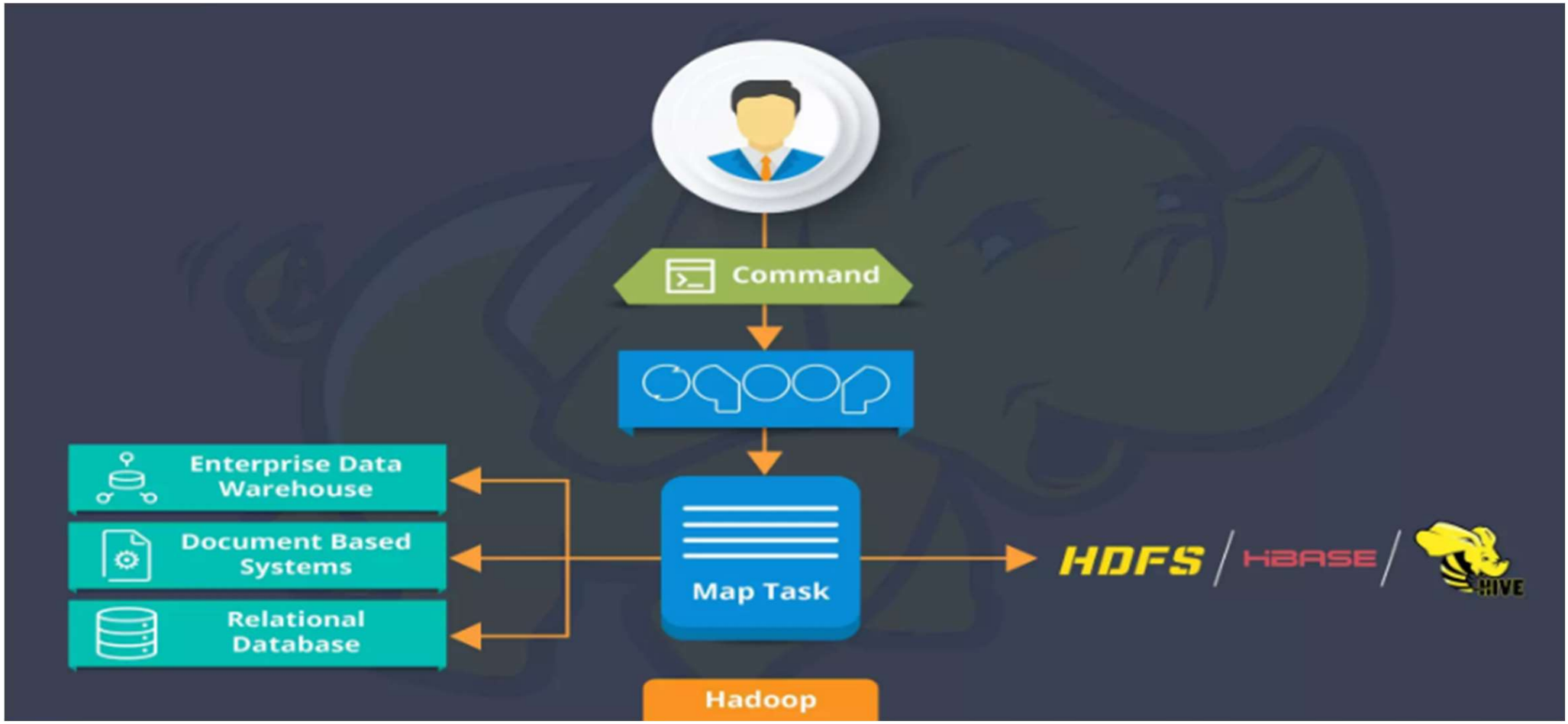
What Sqoop Provide?

- Sqoop runs in Hadoop Cluster
- Sqoop has access to Hadoop Core
  - Sqoop use Mappers to slice the incoming
  - Data is placed to HDFS
- Sqoop Import
  - From RD/NoSql DB to Hadoop
- Export
  - From Hadoop to RD/NoSql DB

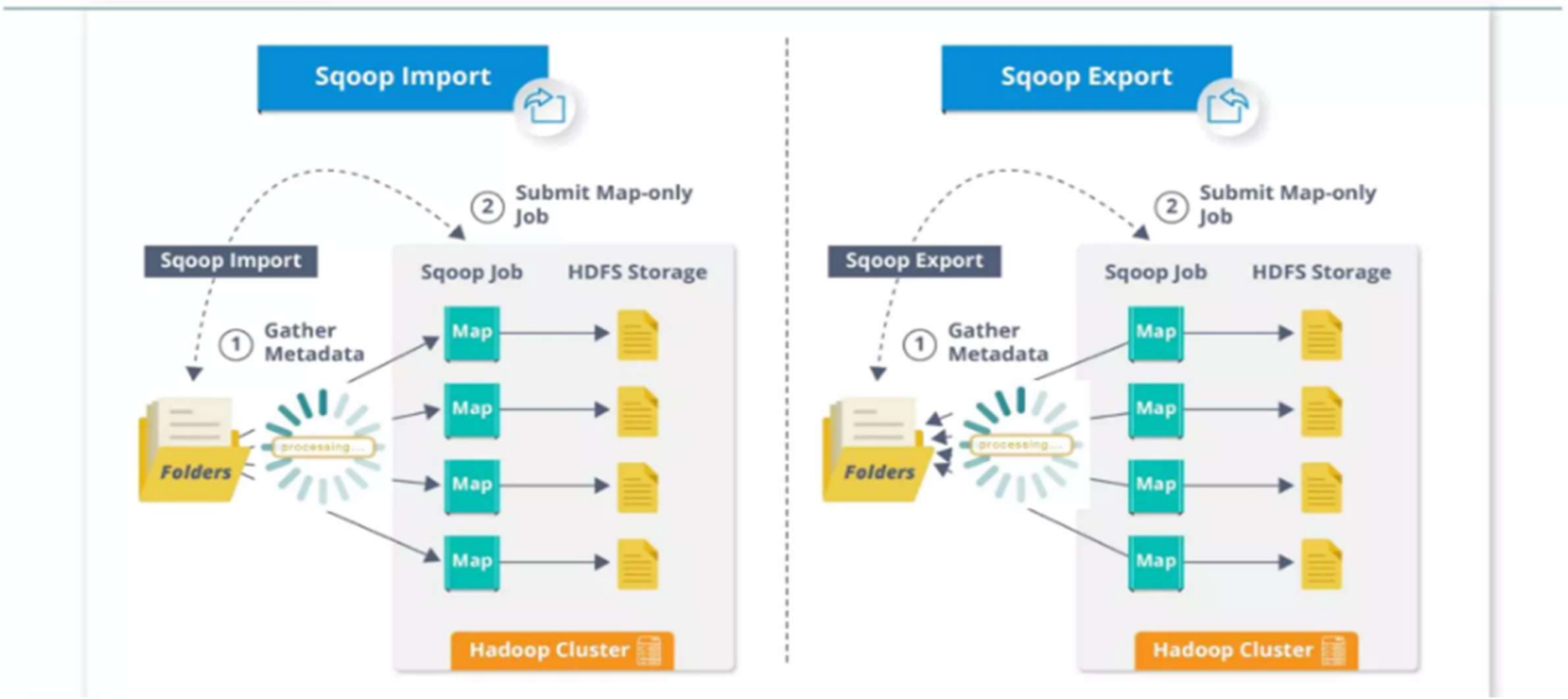
Features


Full Load
Data loading directly to HIVE
Incremental Load
Kerberos Security Integration
Parallel Import/Export
Compression

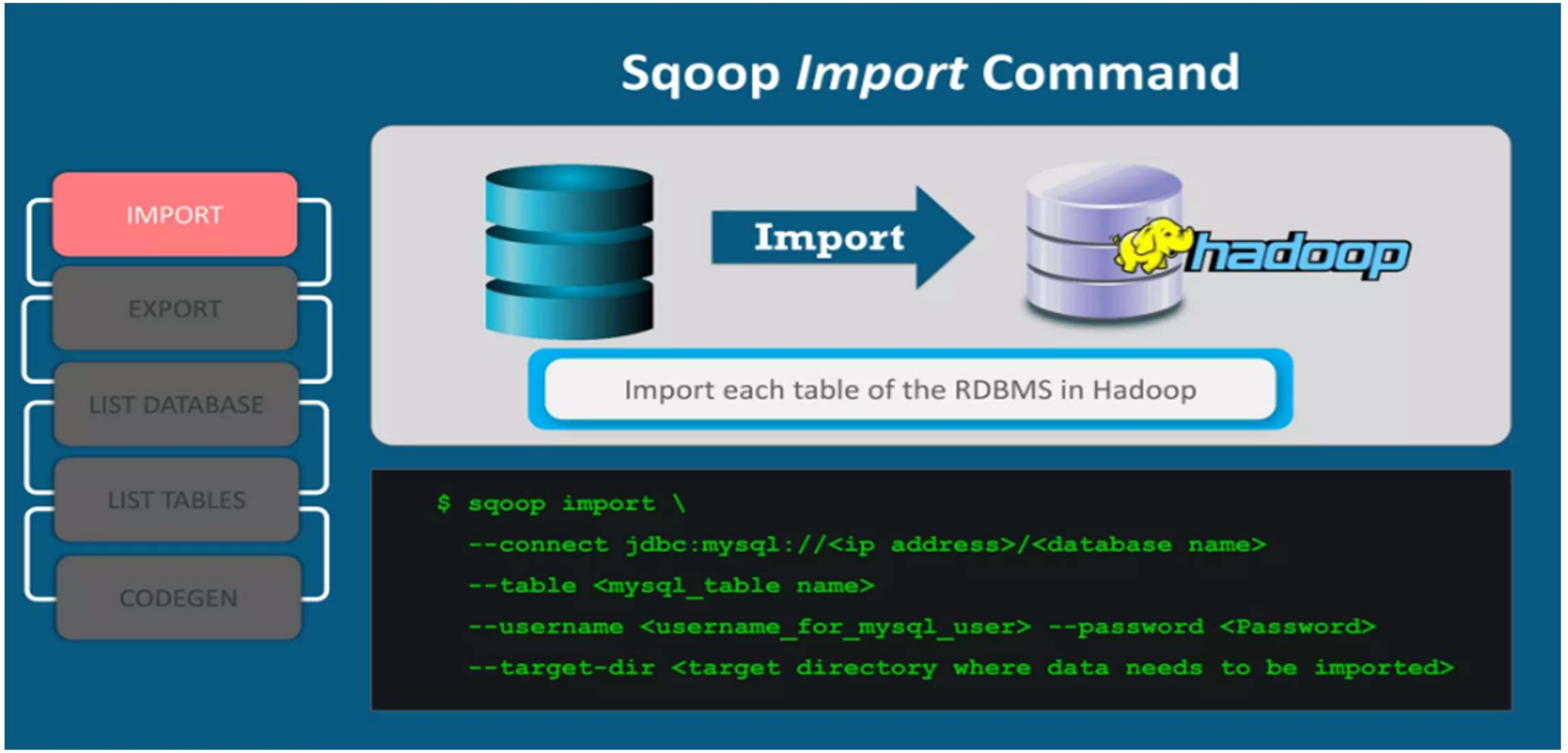
Architecture



Sqoop Import & Export



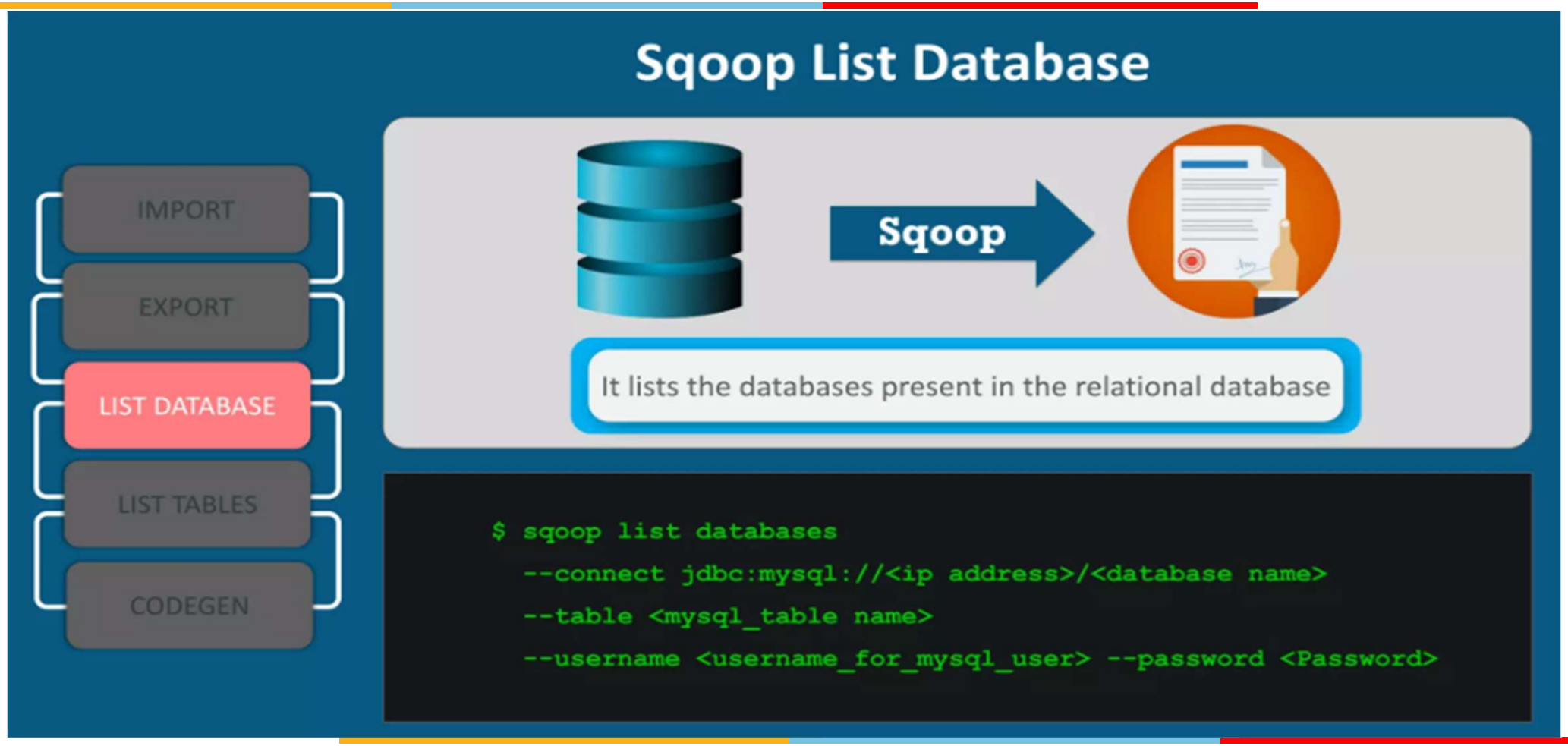
Cont.…


Sqoop Import Command
IMPORT
EXPORT
LIST DATABASE
LIST TABLES
CODEGEN
Import
hadoop
Import each table of the RDBMS in Hadoop
$ sqoop import \
--connect jdbc:mysql://<ip address>/<database name>
--table <mysql_table name>
--username <username_for_mysql_user> --password <Password>
--target-dir <target directory where data needs to be imported>

Cont.…


Sqoop Export Command
IMPORT
EXPORT
LIST DATABASE
LIST TABLES
CODEGEN
Export
hadoop
Exports data from HDFS to RDBMS
$ sqoop export \
--connect jdbc:mysql://<ip address>/<database name>
--table <mysql_table name>
--username <username_for_mysql_user> --password <Password>
--export-dir <directory name where data needs to be exported>

List


Sqoop List Database
IMPORT
EXPORT
LIST DATABASE
LIST TABLES
CODEGEN
Sqoop
It lists the databases present in the relational database
$ sqoop list databases
--connect jdbc:mysql://<ip address>/<database name>
--table <mysql_table name>
--username <username_for_mysql_user> --password <Password>

List Cont.…


Sqoop List Tables
IMPORT
EXPORT
LIST DATABASE
LIST TABLES
CODEGEN
ID
Name
Age
Address
1
Jino
25
Kottayam
2
Neha
22
Dharwad
3
Chaitra
23
Bangalore
It lists the available tables in the database
$ sqoop list tables
--connect jdbc:mysql://<ip address>/<database name>
--table <mysql_table name>
--username <username_for_mysql_user> --password <Password>

Import All Tables


```
$ sqoop import-all-tables (generic-args) (import-args)
$ sqoop-import-all-tables (generic-args) (import-args)
```

Example

Let us take an example of importing all tables from the **userdb** database. The list of tables that the database **userdb** contains is as follows.

```
+--------------------+
|     Tables         |
+--------------------+
|     emp            |
|     emp_add        |
|     emp_contact    |
+--------------------+
```

The following command is used to import all the tables from the **userdb** database.

```
$ sqoop import-all-tables \
--connect jdbc:mysql://localhost/userdb \
--username root
```

Note – If you are using the import-all-tables, it is mandatory that every table in that database must have a primary key field.

Export


```
$ sqoop export (generic-args) (export-args)
$ sqoop-export (generic-args) (export-args)
```

Example

Let us take an example of the employee data in file, in HDFS. The employee data is available in **emp_data** file in 'emp/' directory in HDFS. The **emp_data** is as follows.

```
1201, gopal,     manager, 50000, TP
1202, manisha,   preader, 50000, TP
1203, kalil,     php dev, 30000, AC
1204, prasanth,  php dev, 30000, AC
1205, kranthi,   admin,   20000, TP
1206, satish p,  grp des, 20000, GR
```

It is mandatory that the table to be exported is created manually and is present in the database from where it has to be exported.

Export Cont.…

The following query is used to create the table 'employee' in mysql command line.

```
$ mysql
mysql> USE db;
mysql> CREATE TABLE employee (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(20),
    deg VARCHAR(20),
    salary INT,
    dept VARCHAR(10));
```

The following command is used to export the table data (which is in **emp_data** file on HDFS) to the employee table in db database of Mysql database server.

```
$ sqoop export \
--connect jdbc:mysql://localhost/db \
--username root \
--table employee \
--export-dir /emp/emp_data
```

The following command is used to verify the table in mysql command line.

```
mysql>select * from employee;
```

If the given data is stored successfully, then you can find the following table of given employee data.

```
+------+--------------+-------------+-------------------+--------+
| Id   | Name         | Designation | Salary            | Dept   |
+------+--------------+-------------+-------------------+--------+
| 1201 | gopal        | manager     | 50000             | TP     |
| 1202 | manisha      | preader     | 50000             | TP     |
| 1203 | kalil        | php dev     | 30000             | AC     |
| 1204 | prasanth     | php dev     | 30000             | AC     |
| 1205 | kranthi      | admin       | 20000             | TP     |
| 1206 | satish p     | grp des     | 20000             | GR     |
+------+--------------+-------------+-------------------+--------+
```

Sqoop JOB

Sqoop job creates and saves the import and export commands. It specifies parameters to identify and recall the saved job.

This re-calling or re-executing is used in the incremental import, which can import the updated rows from RDBMS table to HDFS.

Create & Verify job with the name "**myjob**", which can import the table data from RDBMS table to HDFS

Create Job (--create)

```
$ sqoop job --create myjob \
-- import \
--connect jdbc:mysql://localhost/db \
--username root \
--table employee --m 1
```

Verify Job (--list)

'**--list**' argument is used to verify the saved jobs. The following command is used to verify the list of saved Sqoop jobs.

```
$ sqoop job --list
```

It shows the list of saved jobs.

```
Available jobs:
  myjob
```

Inspect & Execute JOB

Inspect Job (--show)

'**--show**' argument is used to inspect or verify particular jobs and their details. The following command and sample output is used to verify a job called **myjob**.

```
$ sqoop job --show myjob
```

It shows the tools and their options, which are used in **myjob**.

```
Job: myjob
 Tool: import Options:
 ----------------------------
 direct.import = true
 codegen.input.delimiters.record = 0
 hdfs.append.dir = false
 db.table = employee
 ...
 incremental.last.value = 1206
 ...
```

Execute Job (--exec)

'**--exec**' option is used to execute a saved job. The following command is used to execute a saved job called **myjob**.

```
$ sqoop job --exec myjob
```

It shows you the following output.

```
10/08/19 13:08:45 INFO tool.CodeGenTool: Beginning code generation
...
```

Sqoop Eval Tool

It allows users to execute user-defined queries against respective database servers and preview the result in the console. So, the user can expect the resultant table data to import. Using eval, we can evaluate any type of SQL query that can be either DDL or DML statement.

Example: Select & Insert Query Evaluation


```
$ sqoop eval \
--connect jdbc:mysql://localhost/db \
--username root \
--query "SELECT * FROM employee LIMIT 3"
```

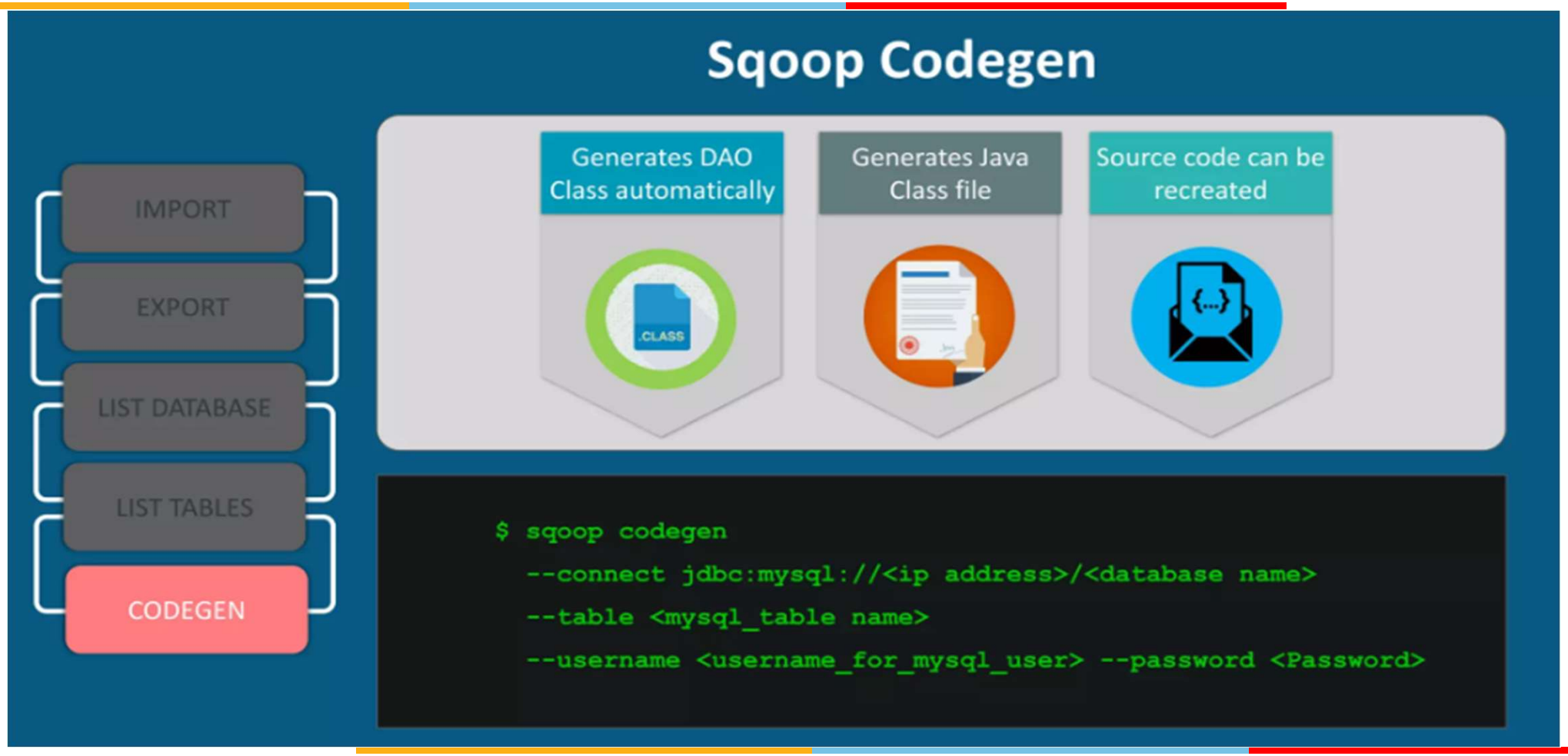
If the command executes successfully, then it will produce the following output on the terminal.

```
+------+-------------+------------+------------------+--------+
| Id   | Name        | Designation | Salary          | Dept   |
+------+-------------+------------+------------------+--------+
| 1201 | gopal       | manager    | 50000            | TP     |
| 1202 | manisha     | preader    | 50000            | TP     |
| 1203 | khalil      | php dev    | 30000            | AC     |
+------+-------------+------------+------------------+--------+
```

```
$ sqoop eval \
--connect jdbc:mysql://localhost/db \
--username root \
-e "INSERT INTO employee VALUES(1207,'Raju','UI dev',15000,'TP')"
```

If the command executes successfully, then it will display the status of the updated rows on the console.

Sqoop Codegen



Codegen Tool

It generates DAO class in Java, based on the Table Schema structure. The Java definition is instantiated as a part of the import process. The main usage of this tool is to check if Java lost the Java code. If so, it will create a new version of Java with the default delimiter between fields.

Example

The following command is used to execute the given example.

```
$ sqoop codegen \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp
```

If the command executes successfully, then it will produce the following output on the terminal.

```
14/12/23 02:34:40 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
14/12/23 02:34:41 INFO tool.CodeGenTool: Beginning code generation
.....
14/12/23 02:34:42 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/local/hadoop
Note: /tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/emp.java uses or
  overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

14/12/23 02:34:47 INFO orm.CompilationManager: Writing jar file:
  /tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/emp.jar
```

Let us take a look at the output. The path, which is in bold, is the location that the Java code of the **emp** table generates and stores. Let us verify the files in that location using the following commands.

```
$ cd /tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/
$ ls
emp.class
emp.jar
emp.java
```

If you want to verify in depth, compare the **emp** table in the **userdb** database and **emp.java** in the following directory

/tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/.

Thank You All