**Pig: Pig is a data flow language**

Pig has two components:

* **Pig Latin:** Data flow language for PIG
* **Environment**: pig-x.y.z.tar. It can be set to run in Local and MapR mode. In local mode it runs on a single JVM, while in MR mode it runs on Hadoop cluster. Pig can run on TEZ too.

Pig Latin can be coded in three ways:

* **Scripts**: Bunch of commands for specific task. Save in file and run using pig command.
* **Grunt Mode**: Interactive Grunt shell prompt, type pig at master node CLI to start it.
* **Embedded Mode:** Embedded in java program

Pig can be run in following modes:

* **Local Mode:** *pig –x local*
* **MapReduce mode:** Pig need to know location of cluster NameNode and JobTracker *$ pig or*

*$ pig –x mapreduce*

MR takes time and effort in coding Pig is an alternative to it, and it is quite extensible using UDF’s. It is very efficient in case of complex join operation.

MR are highly optimized, so if we need to write a reporting job on larger data set and execute it frequently then Map reduce is ideal choice.

Starting pig: Type pig at command line from master node and grunt will start.

**ratings** = **Load** ‘/usr/maria\_dev/ml-100k/u.data’ AS (userID:int, rating:int, ratingTime:int);

**metadata**= **Load** ‘/usr/maria\_dev/ml-100k/u.item’ **USING PigStorage (‘|’)** AS (movieID: int, movieTitle: chararray, releaseDate: chararray, videoRelease: chararray, imdbLink:chararray);

**DUMP** metadata**;**

**nameLookup** = FOREACH metadata GENERATE movieID, movieTitle, ToUnixTime(ToDate(releaseDate, ‘dd-MMM-yyyy’)) AS releaseTime;

**ratingsByMovie =** GROUP ratings BY movieID;

**DUMP** ratingsByMovie**;**

**avgRatings = FOREACH** ratingsByMovieGENERATE group AS movieID, AVG(ratings.rating) AS avgRating;

**DUMP avgRatings;**

**FiveSatrMovies = FILTER** avgRatings BY avgRating > 4.0;

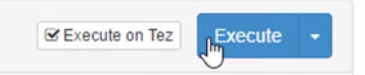
**DESCRIBE ratings;**

**DESCRIBE ratingsByMovie;**

**DESCRIBE avgRatings;**

**fiveStarWithData** = **JOIN** fiveStarMovies BY movieID, nameLookup BY movieID;

**oldestFiveStarMovies = ORDER fiveStarWithData BY nameLookup :: releaseTime;**



We can run above script form Ambari new script option. PIG can run faster on TEZ because TEZ uses DAG to execute it in best order.

**Important PIG Latin commands:**

**LOAD:** To read data

**STORE:** STORE ratings INTO ‘outRatings’ USING PigStore(‘:’);

**DUMP:** To display data to console

**FILTER:** Filtering

**DISTINCT:**

**FOREACH/GENERATE:**

**MAPREDUCE:**

**STREAM:** Stream result to processes

**SAMPLE:** create random sample

**JOIN, COGROUP, GROUP**

**CROSS:** Cartesian product

**CUBE:** Complex than CROSS

**ORDER**

**RANK**

**LIMIT**

**UNION, SPLIT**

**DESCRIBE, EXPLAIN, ILLUSTRATE**

**Relations, Tuple, Bag and field**

Pig Latin statements work with relations. A relation is a bag (more specifically, an outer bag). In above example ratings, metadata etc. are relations

A bag is a collection of tuples and a tuple is an ordered set of fields. A field is a piece of data.

Above code creates a relation named ratings with a given schema **USING PigStorage (‘|’)** is used to define delimiter. **FOREACH/ GENERATE**

Is used to create a relation from another relation.

UDF’s, Loaders and other functions.

**REGISTER: Registers jar file with UDF for importing.**

**DEFINE: Assigns names to function to be used in Pig scripts.**

**IMPORT: Used for importing macros.**

**AVG, CONCAT, COUNT, MAX, MIN, SIZE, SUM.**

* **PigStorage**
* **TextLoader**
* **JsonLoader**
* **AvroStorage**
* **ParaquetLoader**
* **OrcStorage**
* **HBaseStorage**

All UDF should extend a Filter function and has to contain a method called exec, which contains a Tuple.

public class IsOfAge extends FilterFunc {

@Override

publicBoolean exec(Tuple tuple) throwsIOException {

if(tuple == null|| tuple.size() == 0) {

returnfalse;

}

try{

Object object= tuple.get(0);

if(object == null) {

returnfalse;

}

inti = (Integer) object;

if(i == 18 || i == 19 || i == 21 || i == 23 || i == 27) {

returntrue;

} else{

returnfalse;

}

} catch(ExecExceptione) {

thrownewIOException(e);

}

}

}

Once a UDF is created, the following command has to be used to register the JAR file.

register myudf.jar;

X = filter A by IsOfAge(age);

**Pig Latin Example**: Get the list of pages visited by users whose age is between 20 and 25 years.

1. **users** = load 'users' as (name, age);
2. **users\_18to\_25** = filter users by age > 20 and age< 25;
3. **page\_views** = load 'pages' as (user, url);
4. **page\_views\_u18to25** = join users\_18to25 by name, page views by user;

**Hive**

Hive was developed at Facebook and is known for its SQL like language. Hive can be written in

* **Script mode:** hive -f
* **Interactive mode:** hive -e
* **Embedded mode:** embedded to java code

From Ambari: http://127.0.0.1:8888 menu click on Hive View to interact with hive.

**Meta-store:** Central repository of Apache Hive metadata. It stores metadata for Hive tables and partitions in a relational database. It provides client access to this information by using metastore service API.Any JDBC compliant DB can be used for the metastore for Hive.

**Managed vs External Table:** Every Object in Hive is a reference to the HDFS. Database points to a folder in HDFS, table points under folder, partition points a sub-folder under table folder and it goes on till the leaf node which is the file or data.

Hive table is an **internal table** unless specified explicitly while creating. Internal table has strong reference to the inline file system, means if we drop the table in Hive, the data it is referencing will also be deleted. The **external table** does have reference to the data but has a loose coupling with the data. When you drop the table in Hive the data remains intact.

**File Format:** Text File, Sequence File, RC File, AVRO File, ORC File, Parquet File

hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String, salary String, destination String)

COMMENT ‘Employee details’

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ‘\t’

LINES TERMINATED BY ‘\n’

STORED AS TEXTFILE;

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename

[PARTITION (partcol1=val1, partcol2=val2 ...)]

CREATE EXTERNAL TABLE my\_avro\_tbl

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'

STORED as INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'

OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'

LOCATION '/user/...'

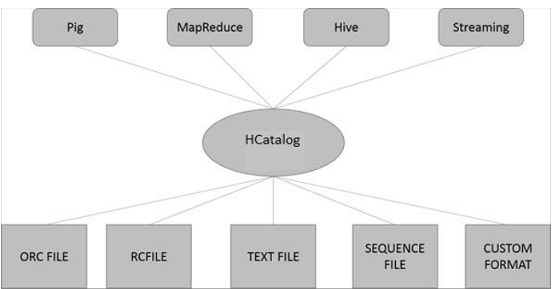
TBLPROPERTIES ('avro.schema.url'='hdfs://name-node.fqdn:8020/user/.../schema.avsc');

**Row Format Delimited**: Instruction that file contain one row per line. New line character means new records.

**Fields Terminated by ‘,’:** To specify delimiter If none is set the default will be used which is ctrl-A

**Stored As TextFile:** Type of file to expect. The other type of file that can be consumed is Sequence Files (Hadoop’s binary file format).

**HCatalog**: Is a table storage management tool for Hadoop that exposes the tabular data of Hive metastore to other Hadoop applications. It enables users with different data processing tools (Pig, MapReduce) to easily write data onto a grid. HCatalog ensures that users don’t have to worry about where or in what format their data is stored.



**Schema on Read:** Makes faster to write as there is no checking of schema during write like RDBMS.

**Hive Load:** LOAD DATA command moves data from DFS to Hive. LOAD DATA LOCAL copies data from local file system into Hive.

**Partitioning**: Partitioning will create subdirectory to store data, based on partioned column.

**To save HQL**: hive –f /somepath

Hive Thrift Server (Phoenix):

**Hive UDF’s**

**Sqoop**

Sqoop is a command-line interface application effectively for transferring bulk data between relational databases and Hadoop. It does it efficiently by doing copy process in parallel (MR framework).

Import: RDBMS to HDFS

*$ sqoop import \*

*--connect jdbc:mysql://localhost/userdb \*

*--username root \*

*--table emp --m 1*

Import directly to hive:

sqoop import --connect jdbc:mysql://localhost:3306/sqoop

--username root

-P

--split-by id

--columns id,name

--table customer

--target-dir /user/cloudera/ingest/raw/customers

--fields-terminated-by ","

--hive-import

--create-hive-table

--hive-table sqoop\_workspace.customers

Apart from RDBMS, Database dump in flat-file Delimited text, Avro, Sequence Files.can also be imported through Sqoop.

sqoop import --connect connectionString \

--username userName –P --table tableName --as-sequencefile

sqoop import --connect connectionString \

--username userName –P --table tableName --as-avrodatafile

Export: HDFS to RDBMS

$ sqoop export \

--connect jdbc:mysql://localhost/db \

--username root \

--table employee \

--export-dir /emp/emp\_data

**Target** can be Hive, Hbase, HDFS

**Incremental Imports:**

* Append Mode
* Last Modified Mode

We control number of map task by –m

**Commands:**

--direct (For higher efficiency) mysql supports

--class-name --as-sequencefile

--Fields-terminated-by

--lines-terminated-by

--hive-import

--where

--where –target-dir --append

--append \

--check-column <primary key> \

--incremental lastmodified \

--last-value <Last Value of primary key which sqoop job has inserted in last run>

sqoop import

--connect jdbc:oracle:thin:@server1.companyxyz.com:4567/prod/DATABASE=schema1

--username xyz

--password xyz

--table customers

--columns cust\_id, name, address, date, history, occupation

--where item>=1234

--target-dir /tmp//customers

--m 8

--split-by cust\_id

--fields-terminated-by ,

--escaped-by \

--hive-drop-import-delims

--map-column-java

cust\_id=string, name=string, address=string, date=string, history=string, occupation=string

**Oozie:**

Oozie is a workflow scheduler system to manage Apache Hadoop jobs. Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions.

**Points:** Make sure each action works on its own.

Make a directory in HDFS for your job.

Create workflow.xml file and put it in HDFS folder

Define variables needed by workflow.xml in job.properties file which goes on your local filesystem where you'll launch the job from

We can also set these properties within your XML.

nameNode:hdfs://sandbox.hortonworks.com : 8020

jobTracker:http://sandbox.hortonworks.com : 8050

queueName:default

oozie.use.system. libpath:true

oozie job --oozie http://localhost:11000/oozie -config

/home/maria\_dev/job.properties -run

Monitor progress at <http://127.0.0.1:11000/oozie>

**Oozie Coordinator:** Schedules workflow execution.

Launches workflows based on start time & frequency. Can be triggered based on data being available. Runs exactly same way as a workflow.

*<coordinator-app xmlns = "uri:oozie:coordinator:0.2" name =*

*"coord\_copydata\_from\_external\_orc" frequency = "5 \* \* \* \*" start =*

*"2016-00-18T01:00Z" end = "2025-12-31T00:00Z"" timezone = "America/Los\_Angeles">*

*<controls>*

*<timeout>1</timeout>*

*<concurrency>1</concurrency>*

*<execution>FIFO</execution>*

*<throttle>1</throttle>*

*</controls>*

*<action>*

*<workflow>*

*<app-path>pathof\_workflow\_xml/workflow.xml</app-path>*

*</workflow>*

*</action>*

*</coordinator-app>*

**Oozie Bundle:** New in Oozie 3.0 A bundle is a collection of coordinators that can be managed together

Example: you may have a bunch of coordinators for processing log data in various ways By grouping them in a bundle, you could suspend them all if there were some problem with log collection.

**Hue** comes with GUI based Oozie editer.