# Introduction to Hadoop

SMU, 2019

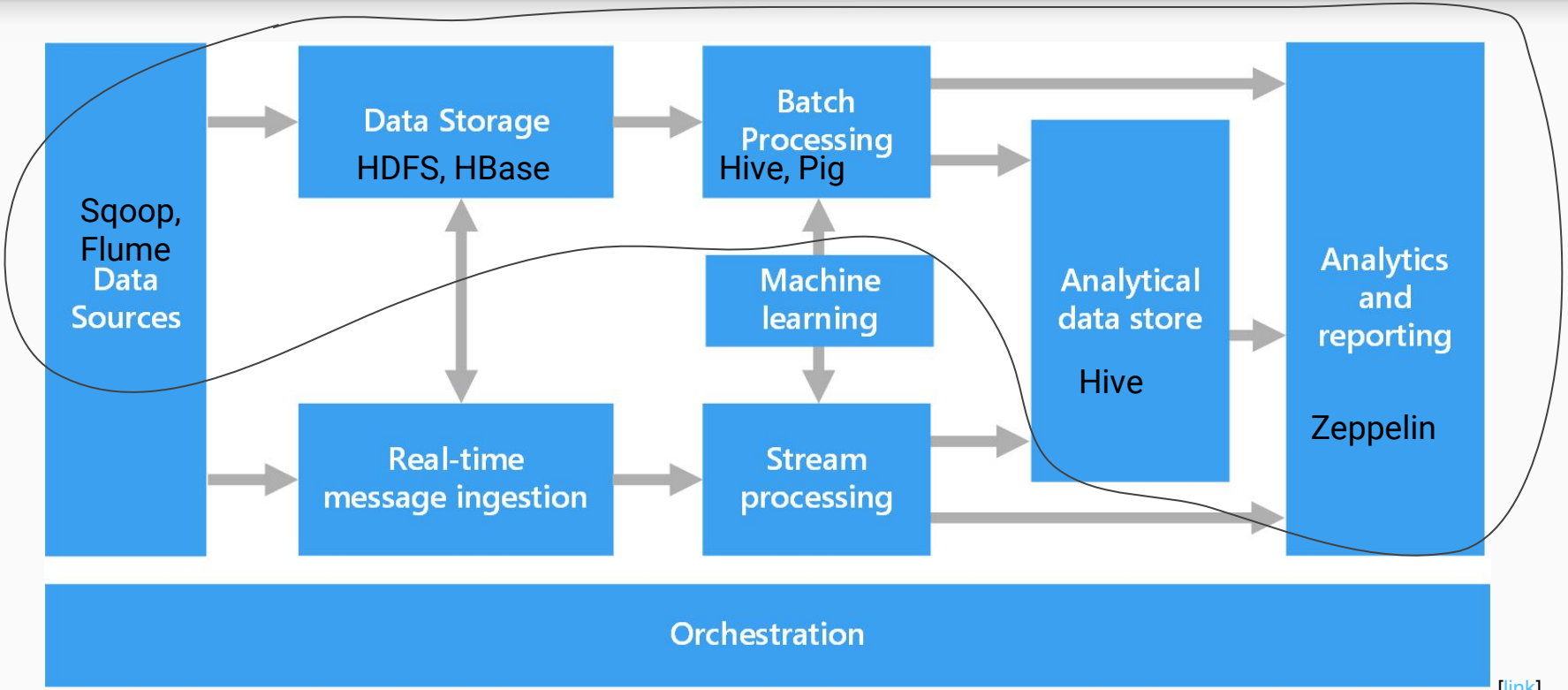
Nikita Neveditsin
*nikita.neveditsin@smu.ca*

# Workshop outline

**Target: learn basic components of Hadoop. By the end of the workshop students should be able to load data into Hadoop, work with HDFS, understand how Hive works, perform basic data visualizations with Zeppelin**

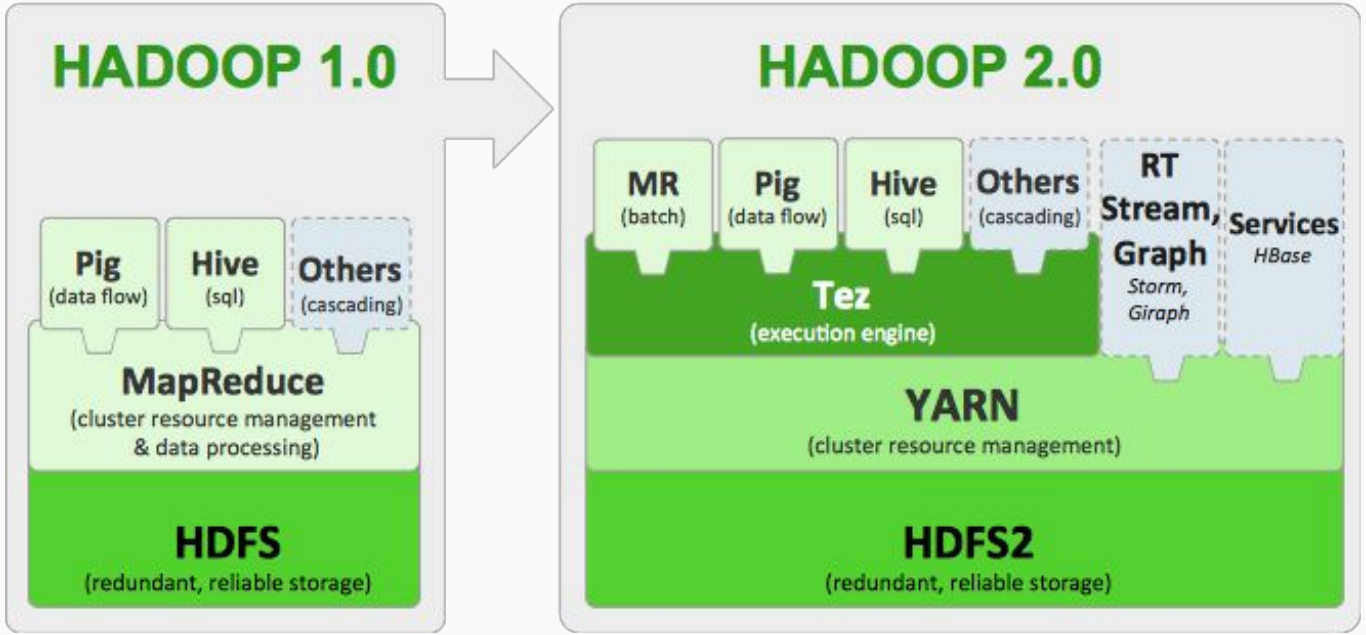
1. HDP Interface: Ambari overview, CLI
2. HDFS: working with files and basic commands
    - Exercise
3. Sqoop
    - Exercise
4. Hive
    - Exercises
5. Zeppelin
    - Exercise

# From lecture:



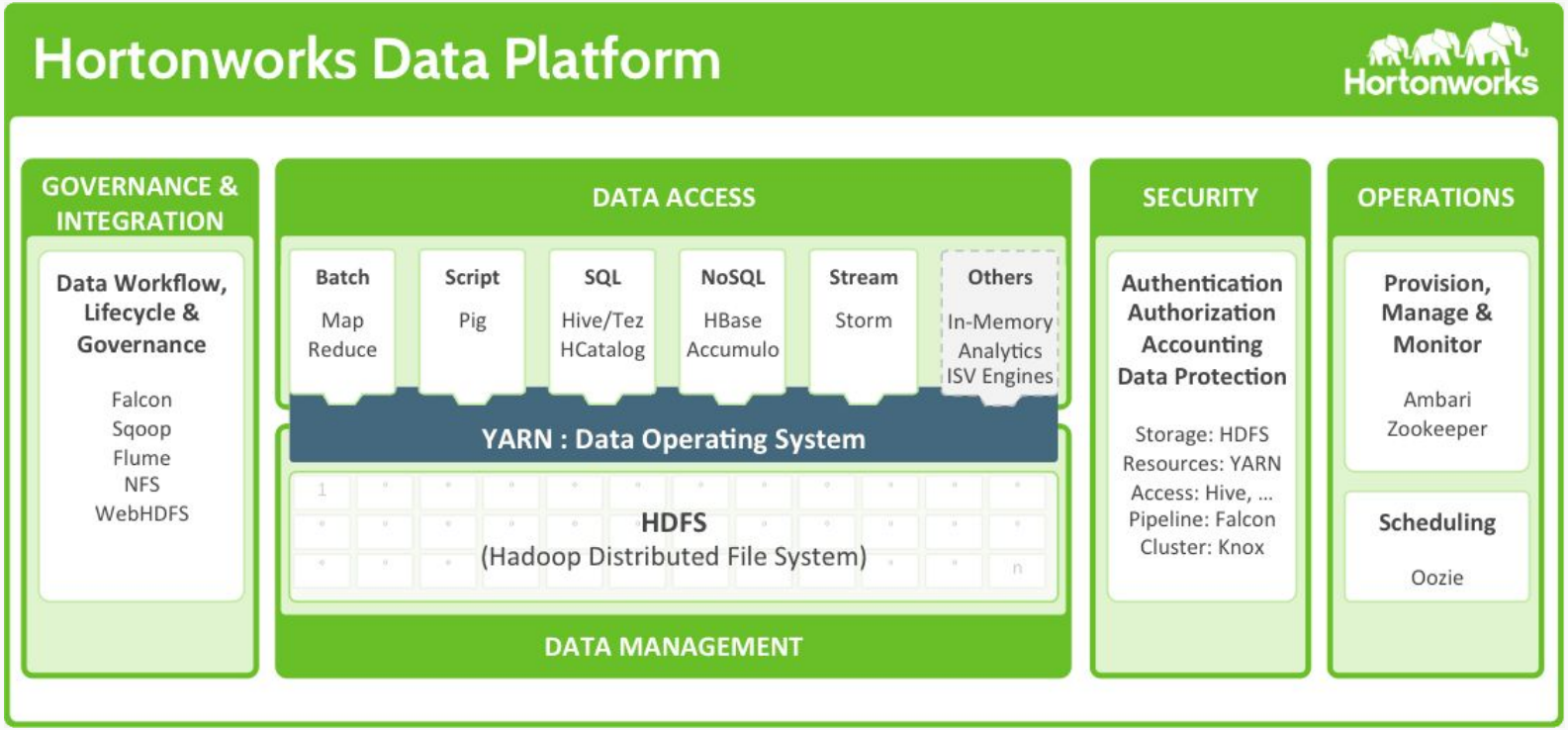

[link]

# Hadoop 2 architecture



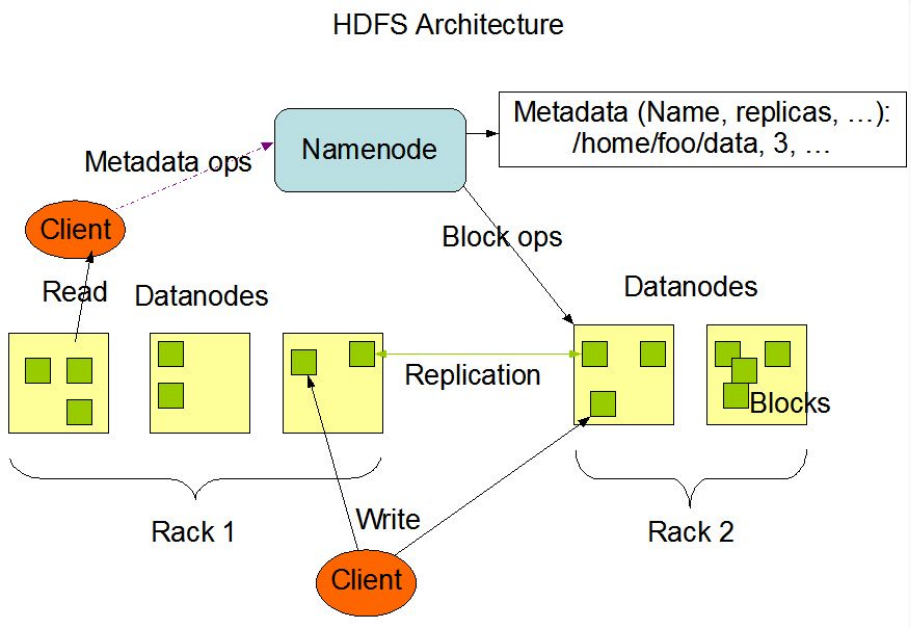

[link]

# Hortonworks: overview




[link]

# HDFS Architecture (from Lecture)

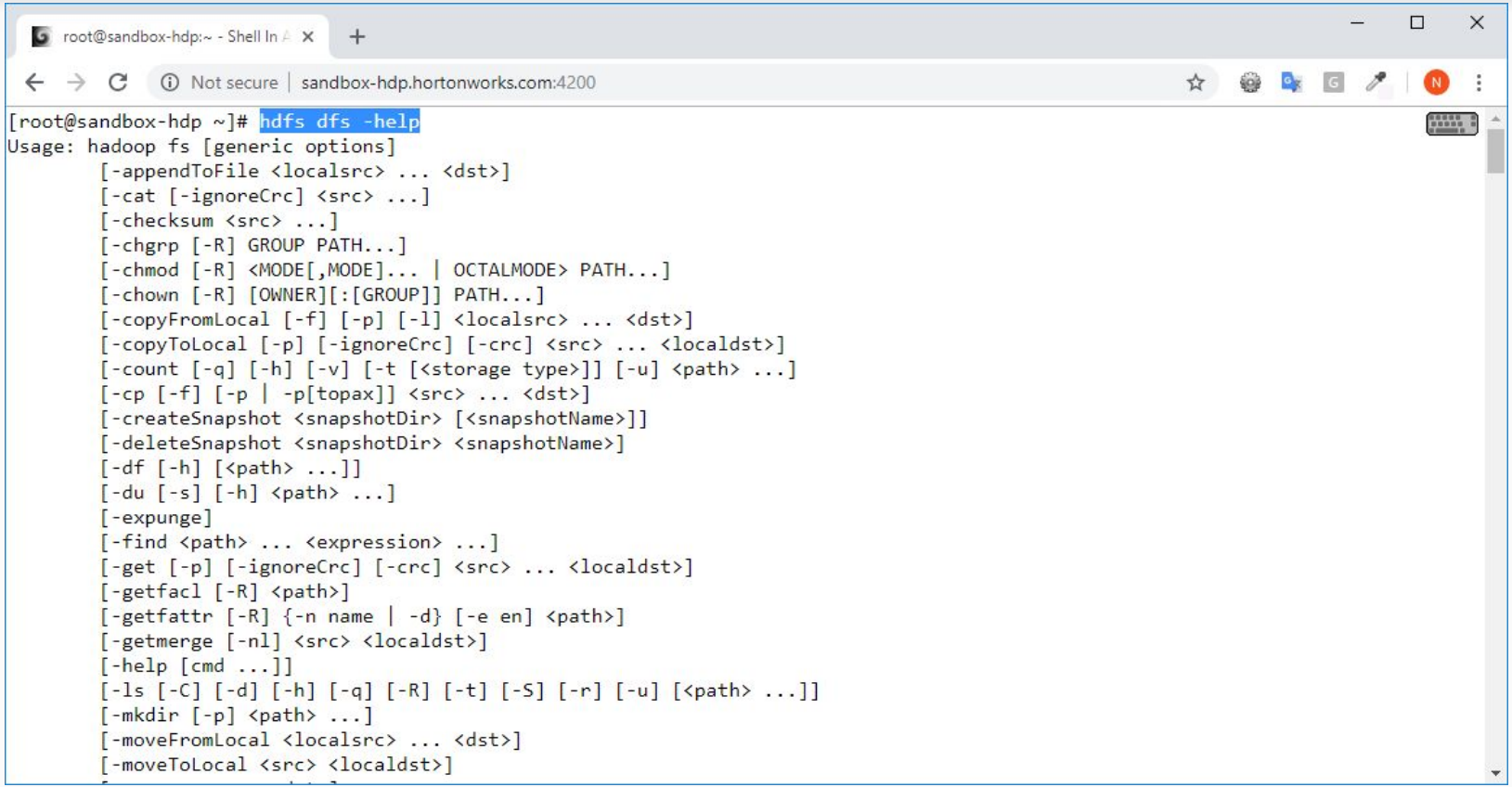



[link]

Two major types on **nodes** (servers):

- **Namenode** stores metadata i.e. number of data blocks, replicas and other details. This metadata is available in memory in the master for faster retrieval of data
- **Datanodes** store actual data in HDFS. They perform read and write operations as per the request of the client [I]

# HDFS Interface

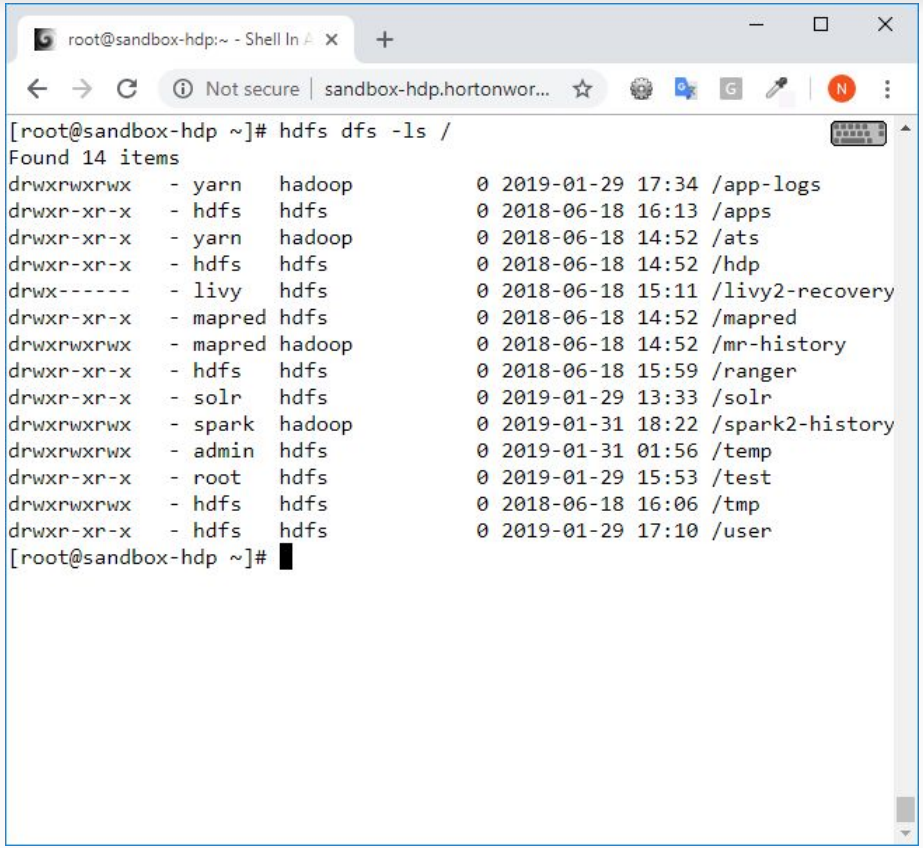


# HDFS Interface (cont-d)

**Syntax:**

**hadoop fs -command args**

- **A lot of unix-like commands (ls, cp, rm, mv, mkdir, etc.)**



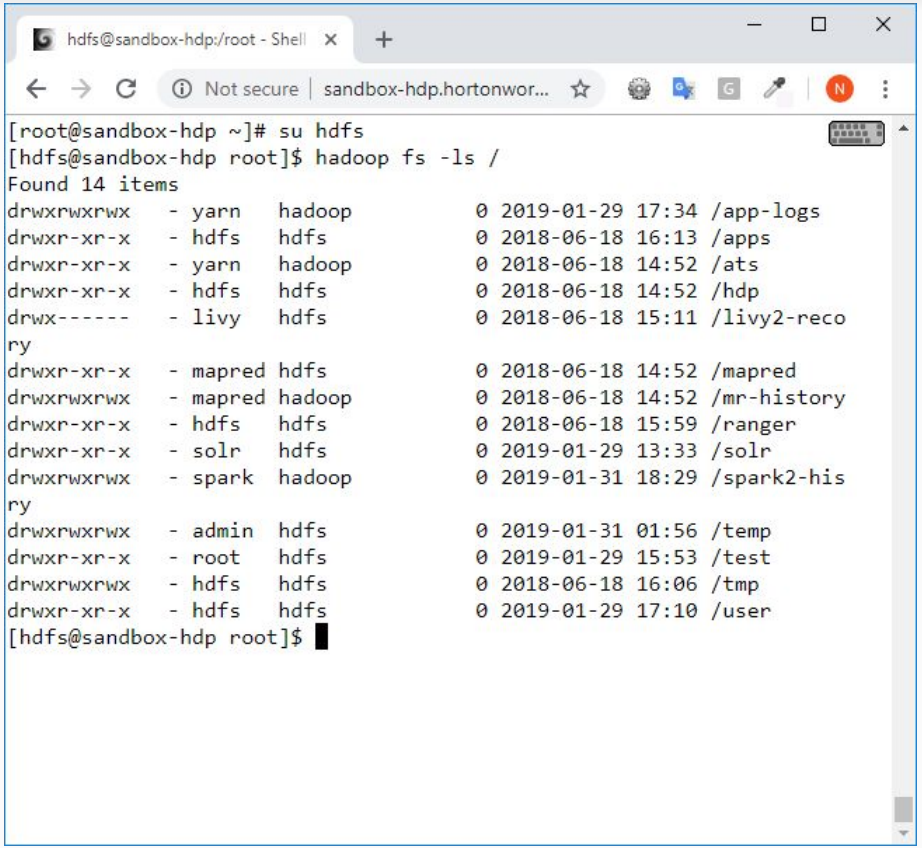
# HDFS: superuser

**hdfs** has “hdfs” superuser (hdfs user can do **everything** inside the file system)

From sandbox type:

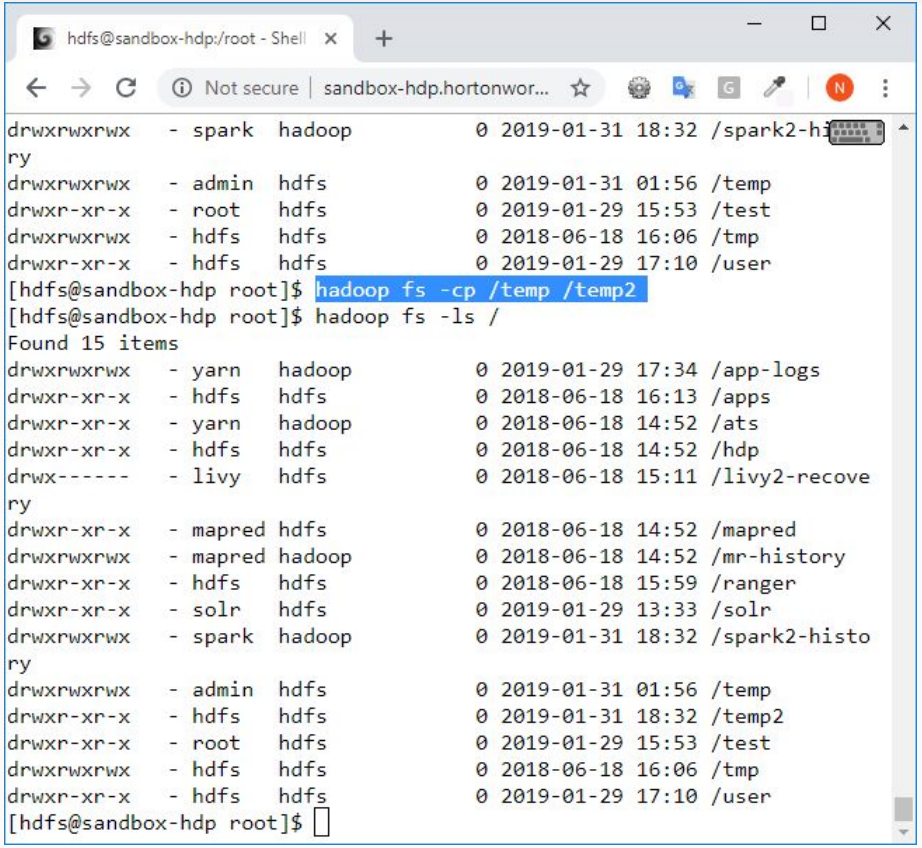
**su hdfs**

This will switch your user from root to hdfs



# HDFS: copy files

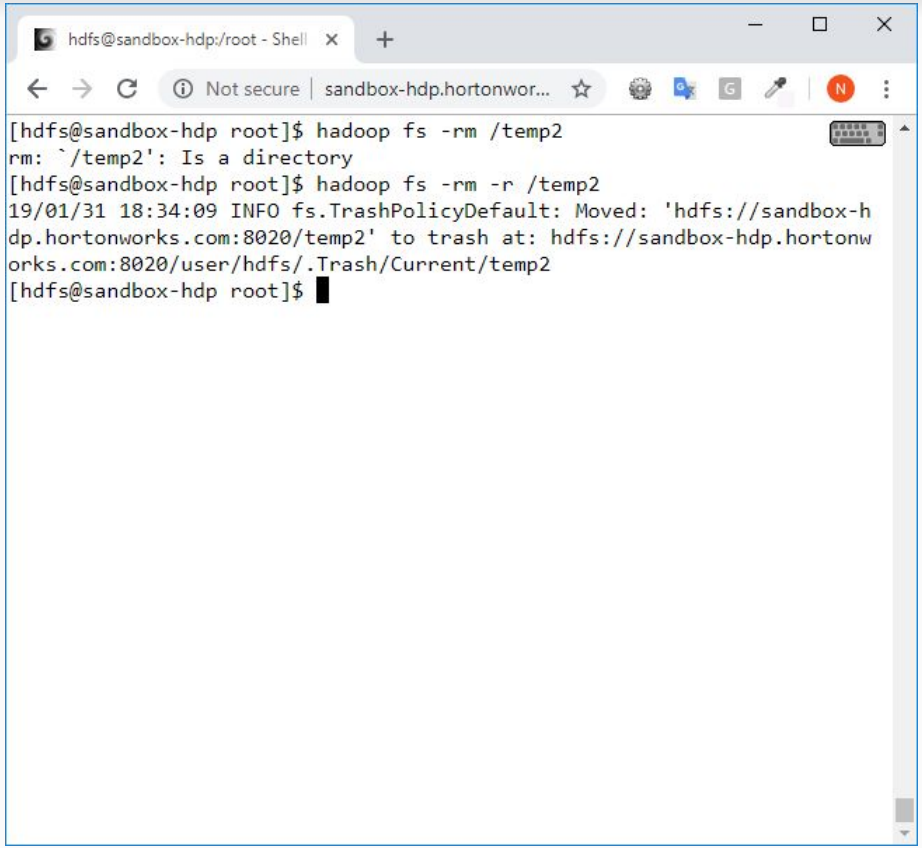
hadoop fs -cp **src dst**



# HDFS: remove files

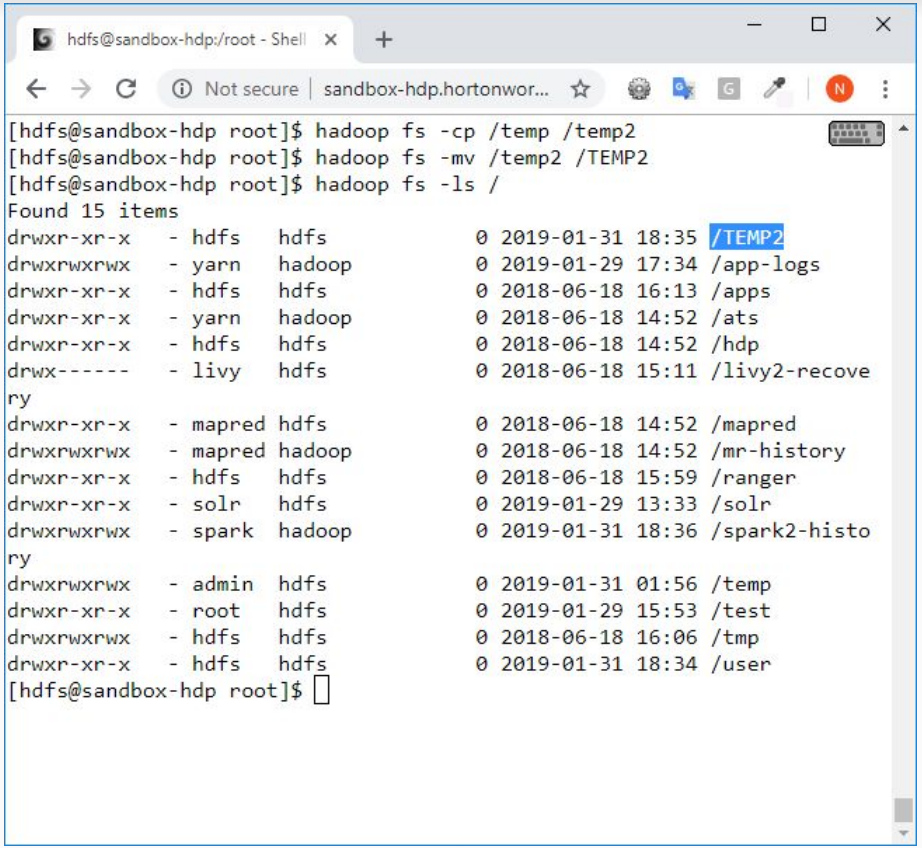
hadoop fs -rm **file**

- **-r** if need to remove directory



# HDFS: move or rename

hadoop fs -mv **file1 file2**

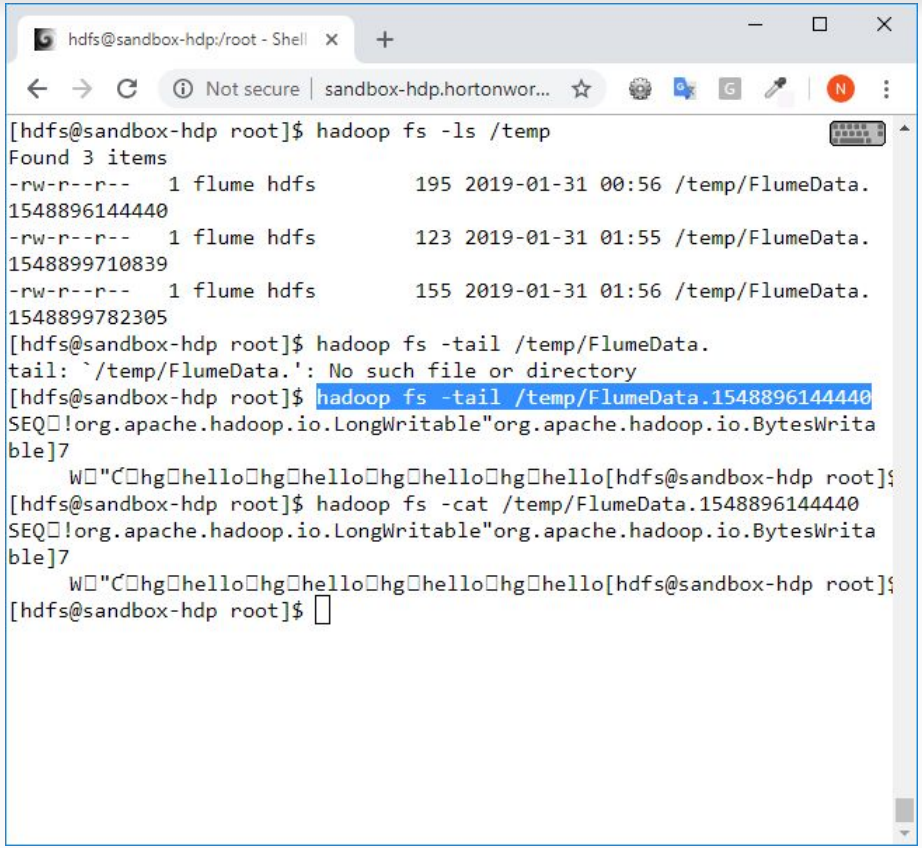


# HDFS: cat and tail

hadoop fs -cat **file**

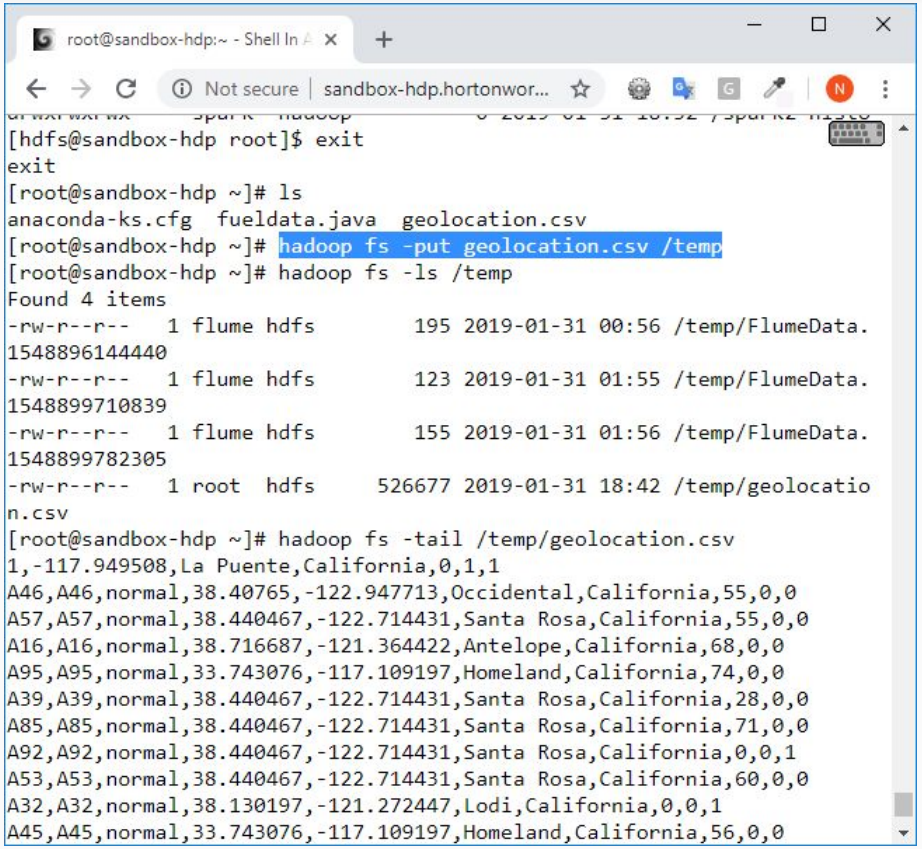
hadoop fs -tail **file**

Like in Linux/*NIX



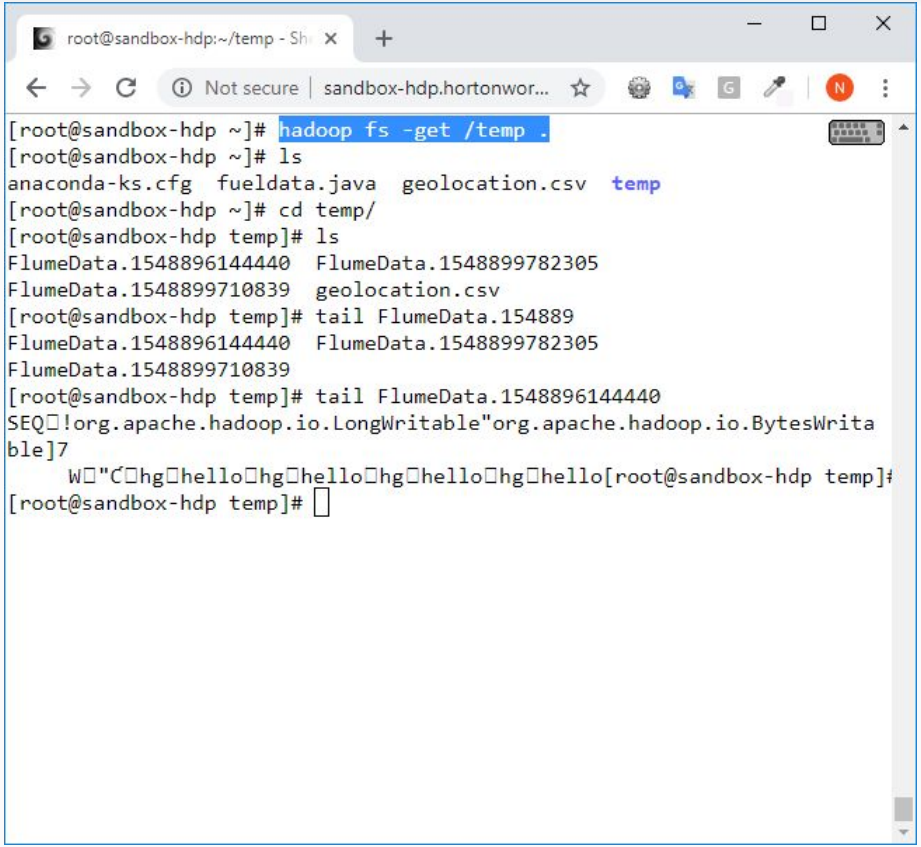
# HDFS: copy from local file system

hadoop fs -put **local_file HDFS_path**



# HDFS: copy from HDFS to local file system

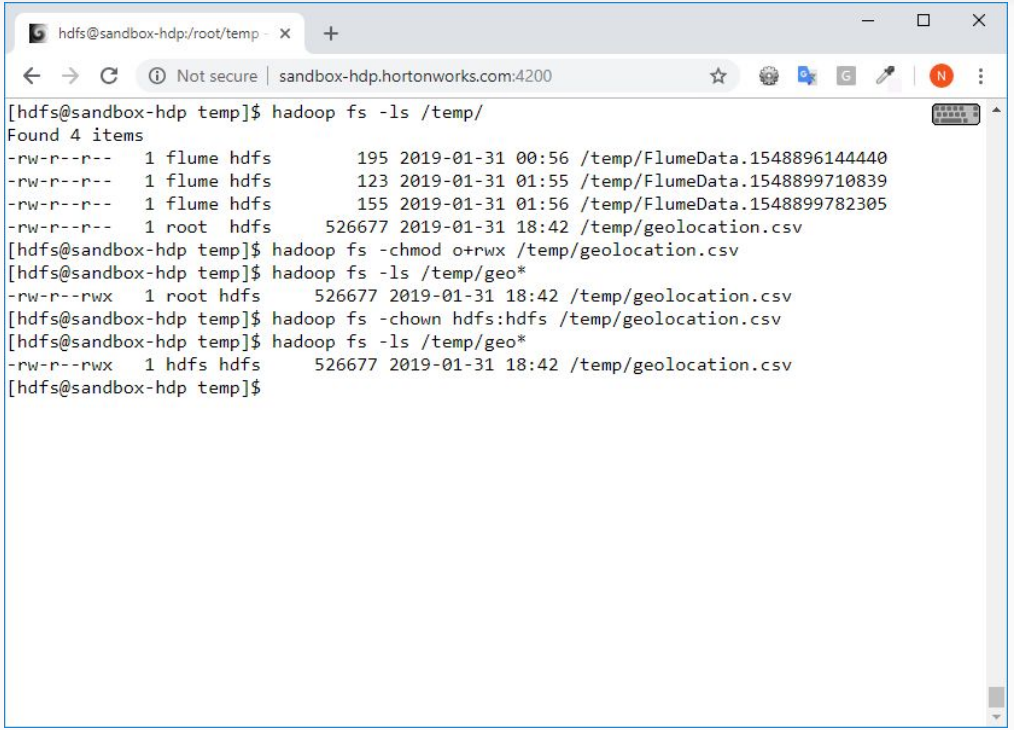
hadoop fs -get **HDFS_path local_file**



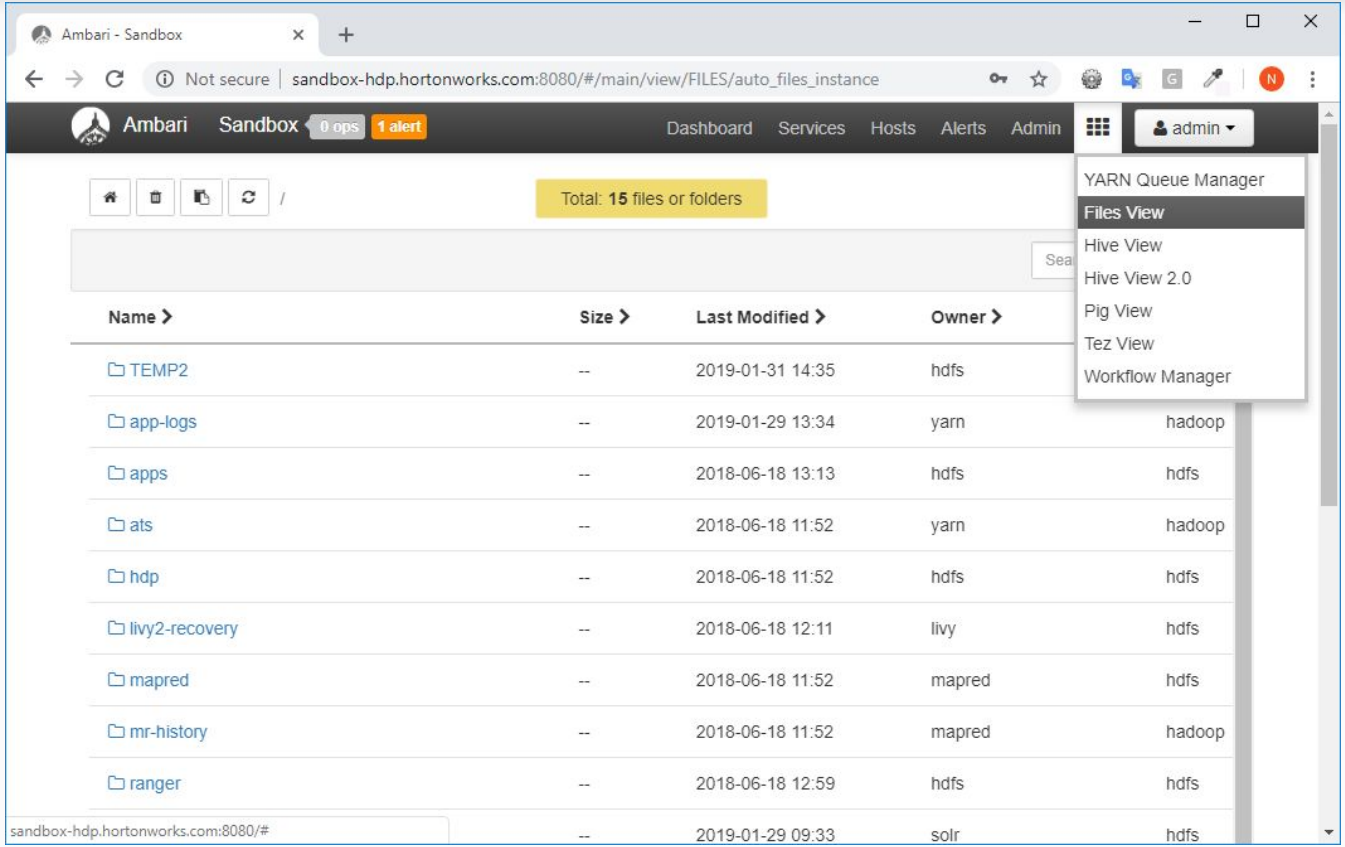
# HDFS: permissions

hadoop fs -chown **user:group file**

hadoop fs -chmod **o+rx file**



# HDFS: “Files View” in Ambari



# Exercise

1. Download a dataset (US census, 1996):

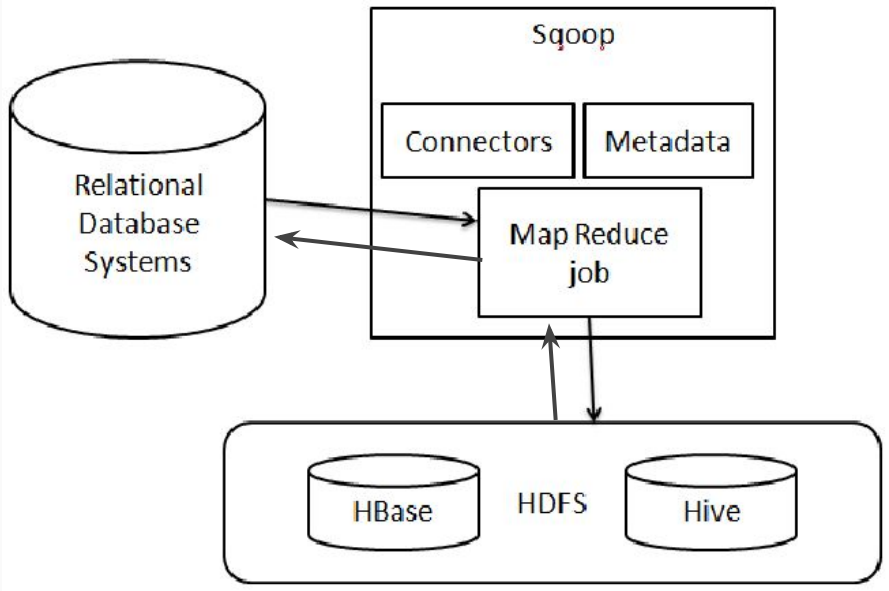
http://wolly.cs.smu.ca/tmp/adult.data

2. Create a folder "**/dsets/census**" in HDFS, put the data there

3. Permissions:

- The dataset file should have **read/write** permissions for everyone
- **dsets** folder should have read/write permissions for everyone as well

# Sqoop




**Apache Sqoop** is a tool designed for efficiently ***transferring*** bulk data between Apache Hadoop and structured datastores such as **relational databases**.

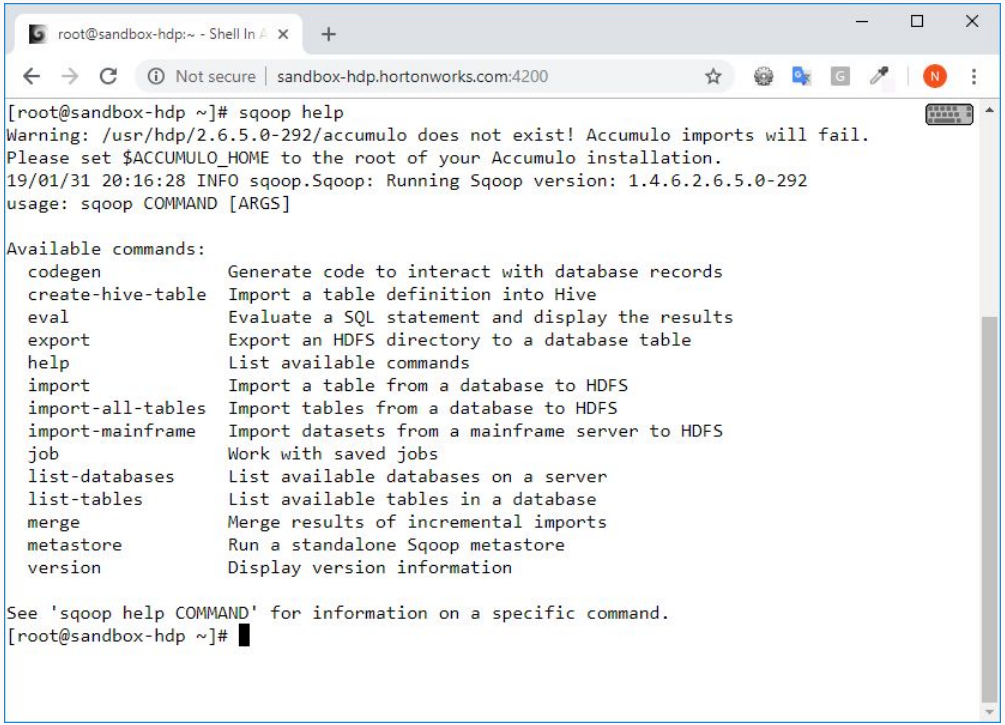
[link]

# Sqoop is a command-line tool

**sqoop** command args

Most commonly used commands:

- import
- export (Note: a table with schema have to be defined in the target database)

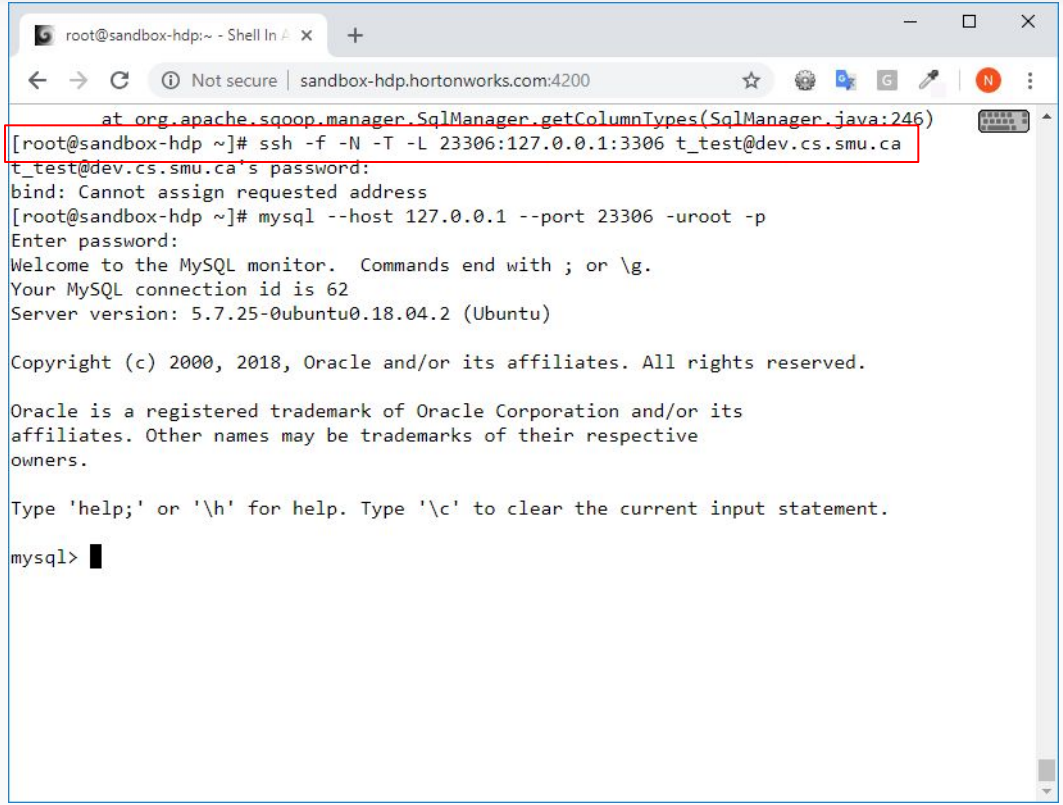


# Sqoop: simple export example

**Step 1:**

Make sure that you can connect to the database you are pulling data from

In our example we set an ssh tunnel to **dev.cs.smu.ca** server to access a database there

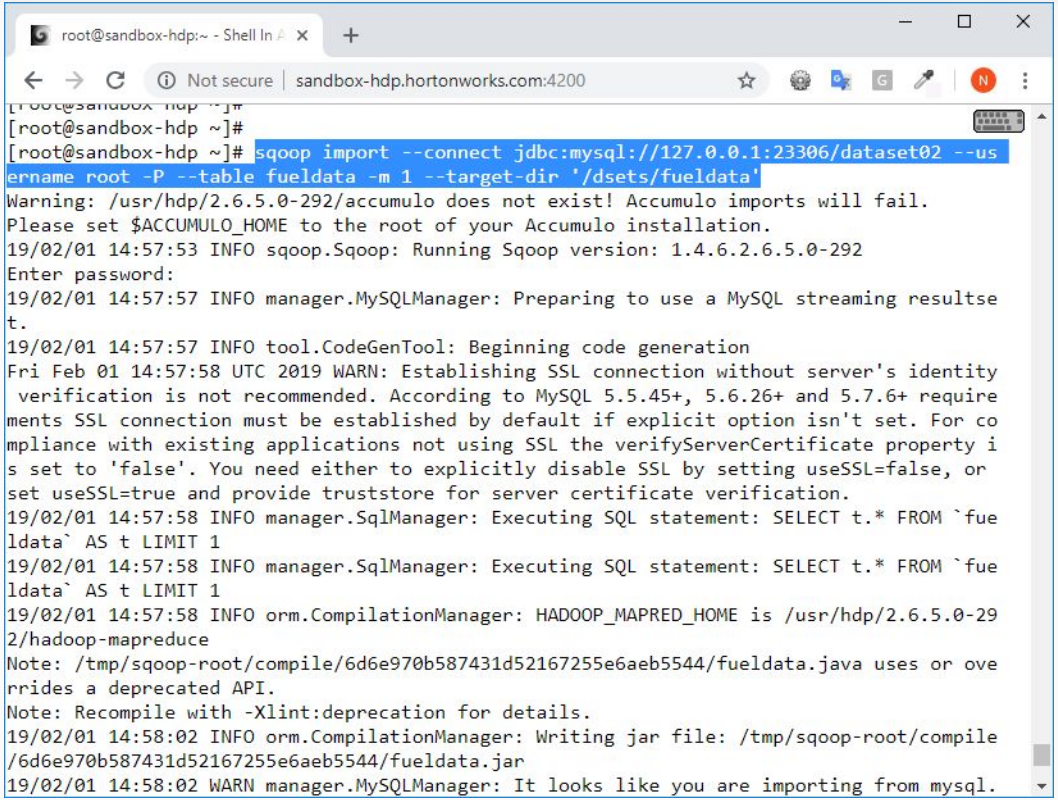


# Sqoop: simple export example (cont-d)

**Step 2:**

Run sqoop import command:

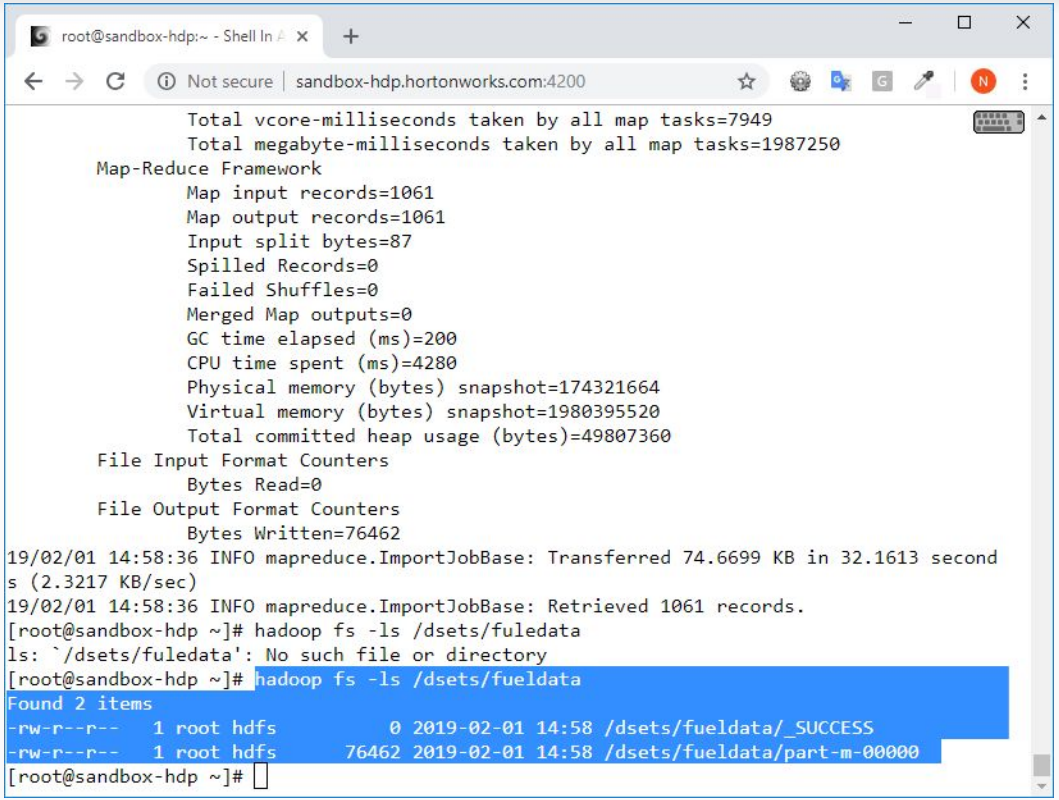
**sqoop import** *--connect*
**jdbc:mysql://127.0.0.1:23306/dataset02**
*--username* **root** -P *--table* fueldata *-m 1*
*--target-dir* '/dsets/fueldata'



# Sqoop: simple export example (cont-d)

**You'll see results of the command.**

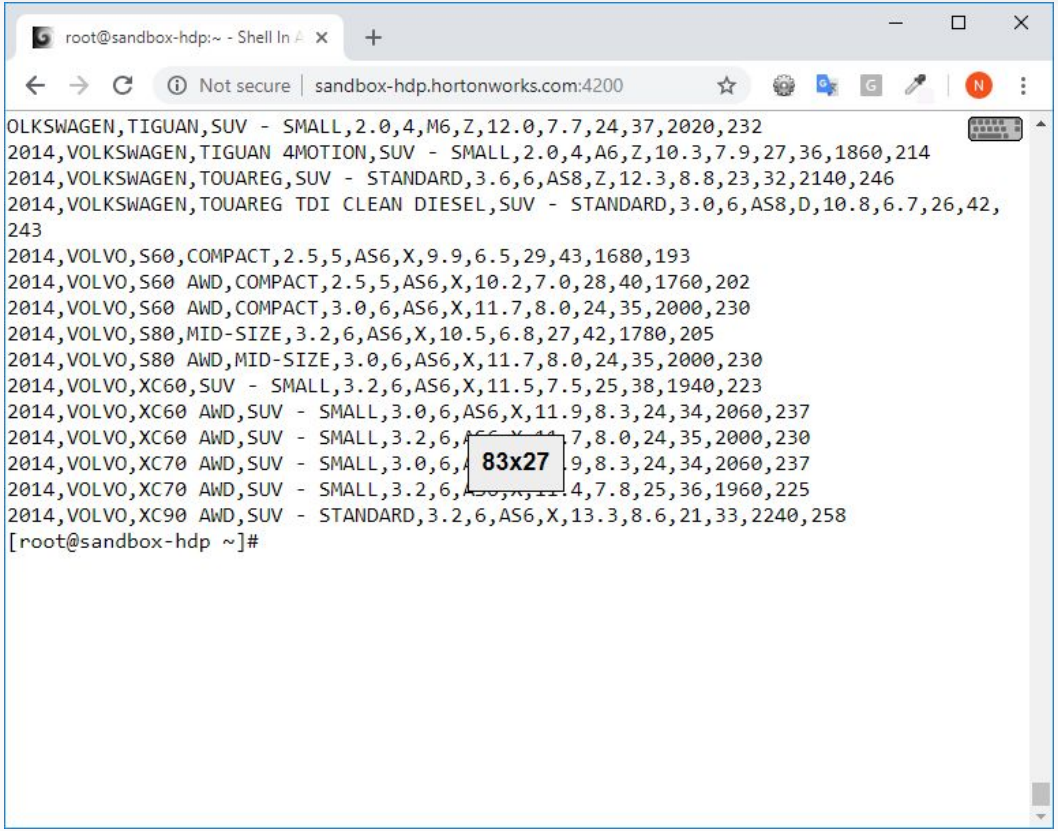
**fueldata directory is created and has files there**



# Sqoop: simple export example (cont-d)

By default, the data is saved as textfile in HDFS (rows are separated by a newline, columns are separated by comma or other separator)

You may also import data directly to Hive (later this day)



# Sqoop: simple export example (cont-d)

## Command line arguments explanation:

**sqoop import** *--connect* **jdbc:mysql://127.0.0.1:23306/dataset02** *--username* **root** -P *--table* fueldata *-m 1* *--target-dir* '/dsets/fueldata'

- import - we **import** data (straightforward)
- After --connect we specify a **jdbc connection string** (where 127.0.0.1: **host**, 23306: **port**, dataset02: **database**)
- After --username we specify username for the mysql database connection
- -P means prompt for a password for the database connection
- After --table we specify a table name in our remote mysql database (you can also use *import-all-tables* command to import the whole database)
- After --target-dir we specify a directory in HDFS where to store the data
- ***-m 1*** means that we use **one mapper** rather than many. If you specify many mappers (>1), make sure that the table in database has a primary key (as it splits data by primary key)

# Exercise

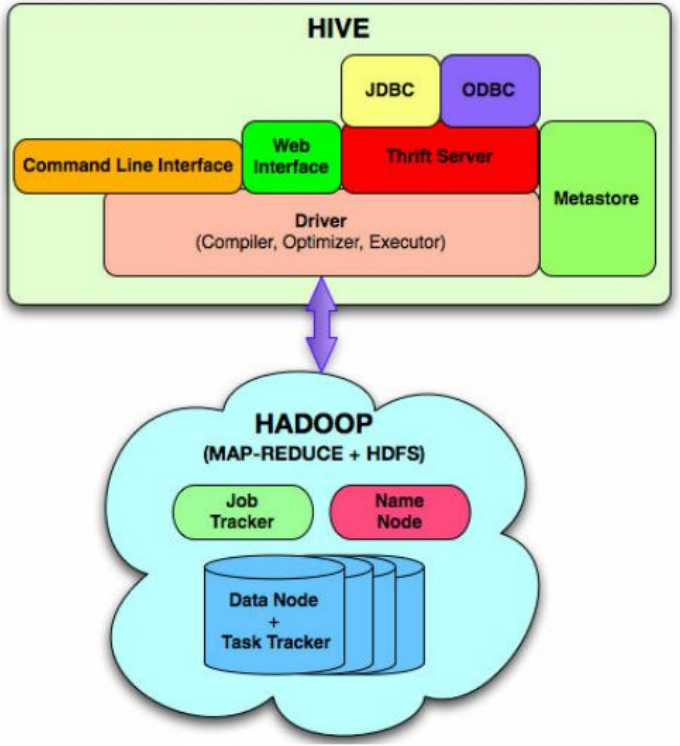
- Transfer the **fueldata** table from **dataset02** database from MySQL at dev.cs.smu.ca to HDFS folder **/dsets/fuel** using sqoop
- Make sure that is has been transferred correctly (you may use *hadoop fs -tail* command)

Hint: if you forgot how to create an SSH tunnel:

**ssh -f -N -T -l 23306:127.0.0.1:3306 username@dev.cs.smu.ca**

You all can use 23306 port (no birthdays are necessary - you are on your own VM)

# Hive Architecture



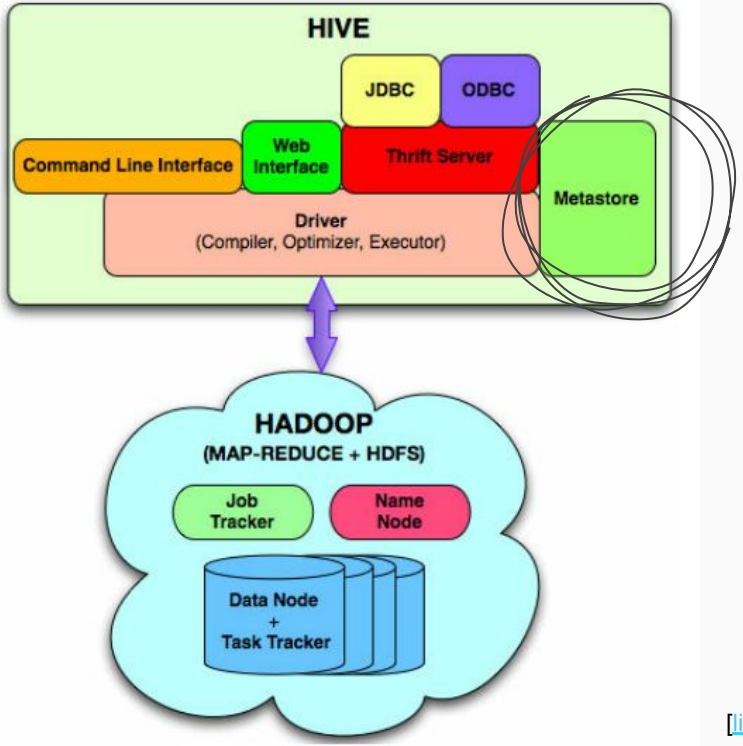

[link]

**Apache Hive** is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Introduced initially at Facebook around 2008 [wiki]

Important things:

- Hive is **not a database** but it provides a SQL-like interface to querying data (called HiveQL)
- In Hadoop, Hive works on top of HDFS (data is stored in HDFS) and uses MapReduce/Tez to execute queries

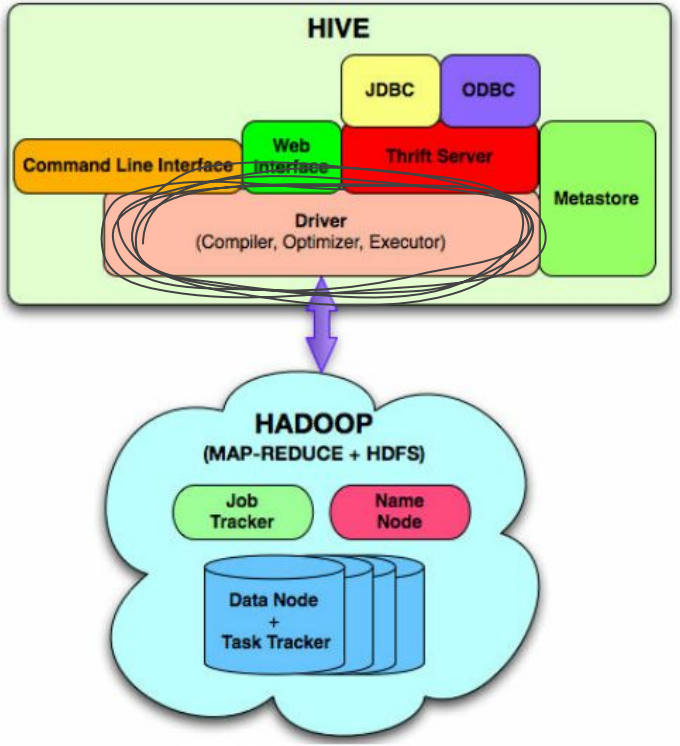
# Hive Architecture: metastore




**Metastore**: Stores metadata for each of the tables such as their schema and location. It also includes the partition metadata which helps the driver to track the progress of various data sets distributed over the cluster. **The data is stored in a traditional RDBMS format (MySQL in our case)** [wiki].

[link]

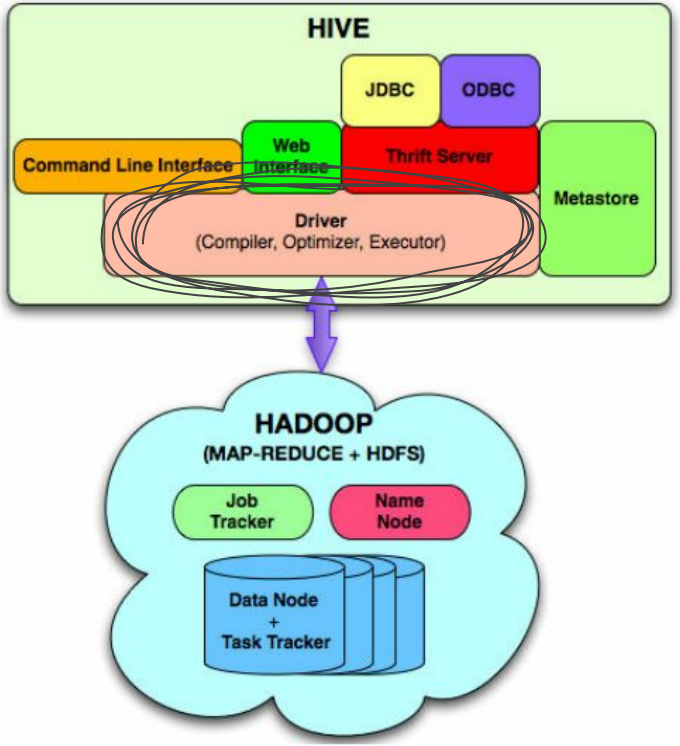
# Hive Architecture: driver




**Driver**: Acts like a controller which receives the **HiveQL** statements. It starts the execution of the statement by creating sessions, and monitors the life cycle and progress of the execution. It stores the necessary metadata generated during the execution of a HiveQL statement. The driver also acts as a collection point of data or query results obtained after the Reduce operation [I]

[link]

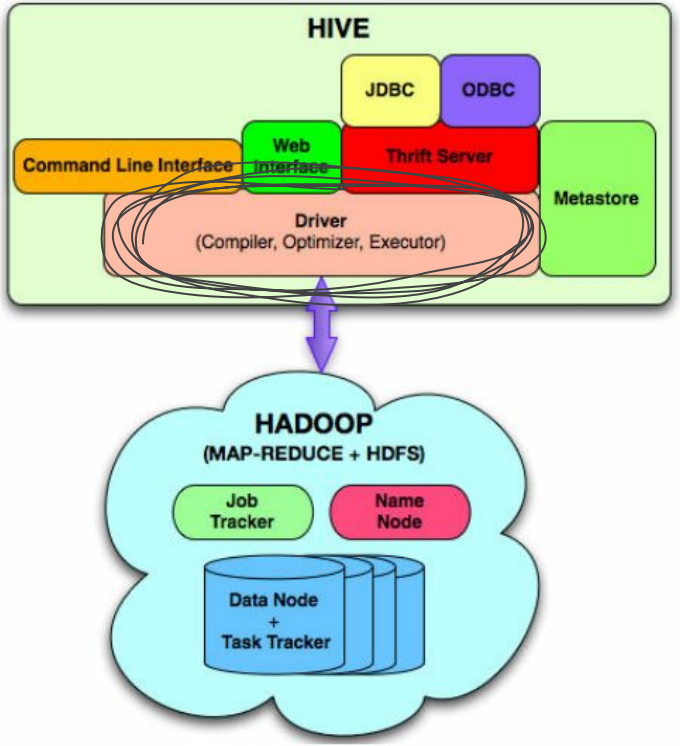
# Hive Architecture: compiler




[link]

**Compiler**: Performs compilation of the HiveQL query, which converts the query to an execution plan. This plan contains the tasks and steps needed to be performed by the Hadoop MapReduce to get the output as translated by the query. The compiler converts the query to an abstract syntax tree (AST). After checking for compatibility and compile time errors, it converts the AST to a directed acyclic graph (DAG). The DAG divides operators to MapReduce stages and tasks based on the input query and data [1]

# Hive Architecture: optimizer and executor




**Optimizer**: Performs various transformations on the execution plan to get an optimized DAG. Transformations can be aggregated together, such as converting a pipeline of joins to a single join, for better performance. It can also split the tasks, such as applying a transformation on data before a reduce operation, to provide better performance and scalability. However, the logic of transformation used for optimization used can be modified or pipelined using another optimizer.

**Executor**: After compilation and optimization, the executor executes the tasks. It interacts with the job tracker of Hadoop to schedule tasks to be run.

[link]

# Hive vs Relational Database

**1. Schema on read:**

Traditionally the table's schema is enforced at data load time (schema on write). Hive enforces it at query time (a load operation is simply a quick file move)

**2. Updates:**

Table updates are only possible by transforming all the data into a new table (i.e. no appends)

**3. Transactions :**

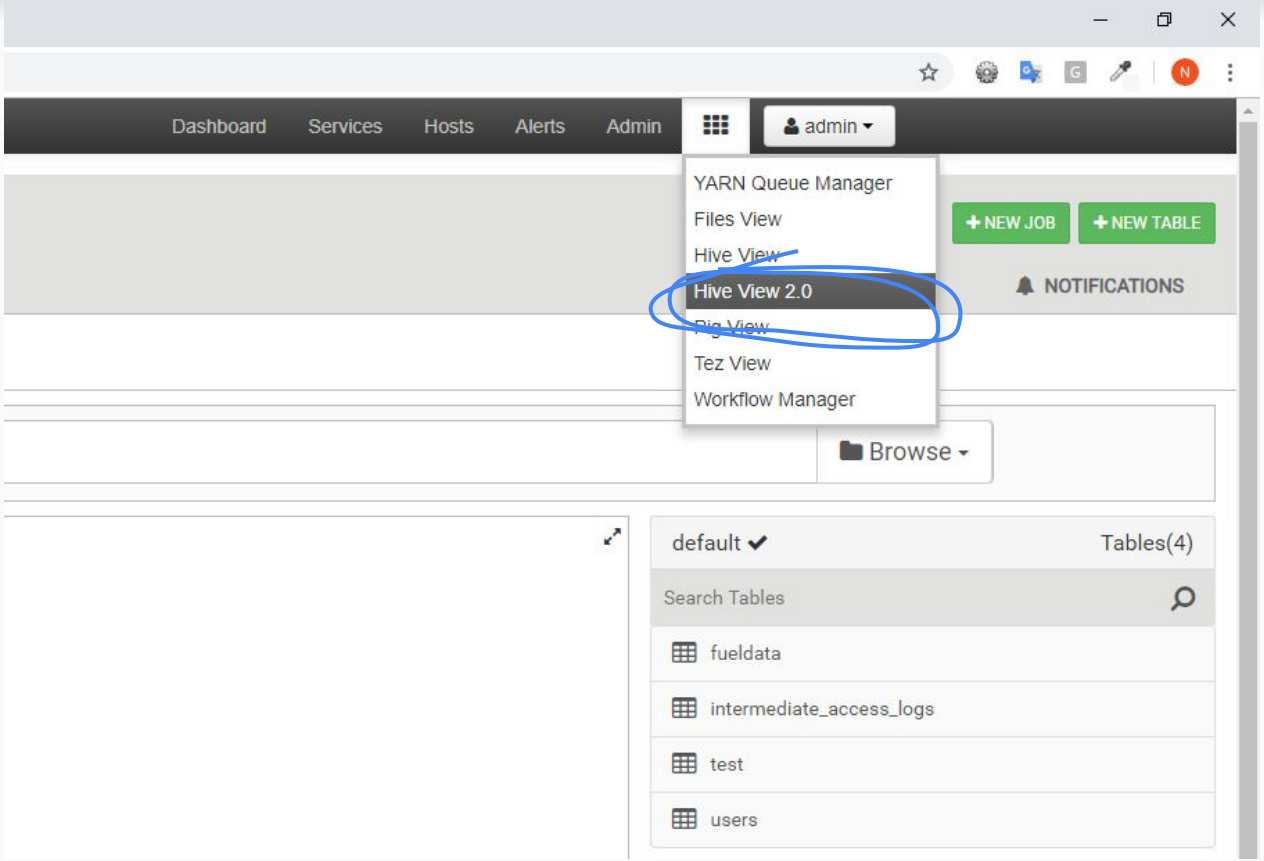
Hive does not support concurrent accesses to tables and hence application-level concurrency and locking mechanisms are needed.

**4. Indexes:**

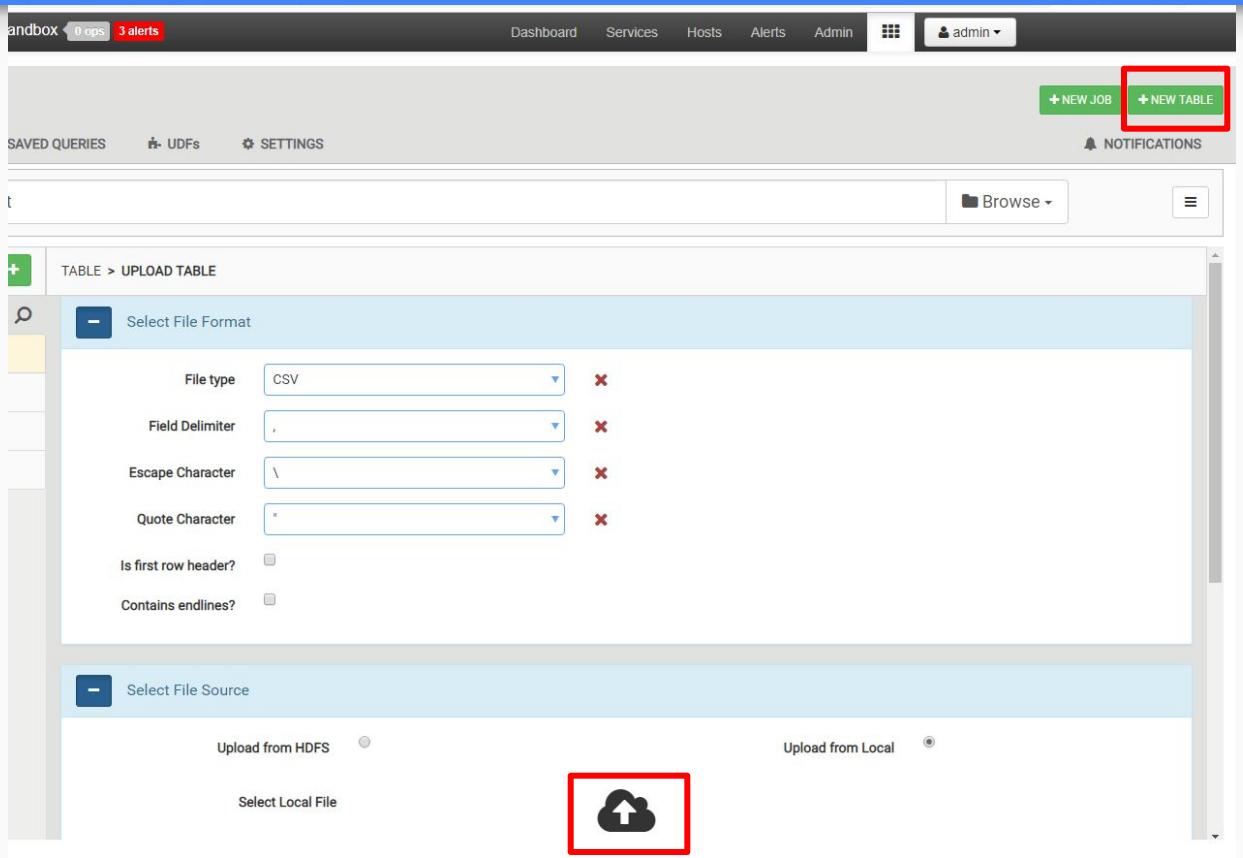
Support provided but relatively immature

[link]

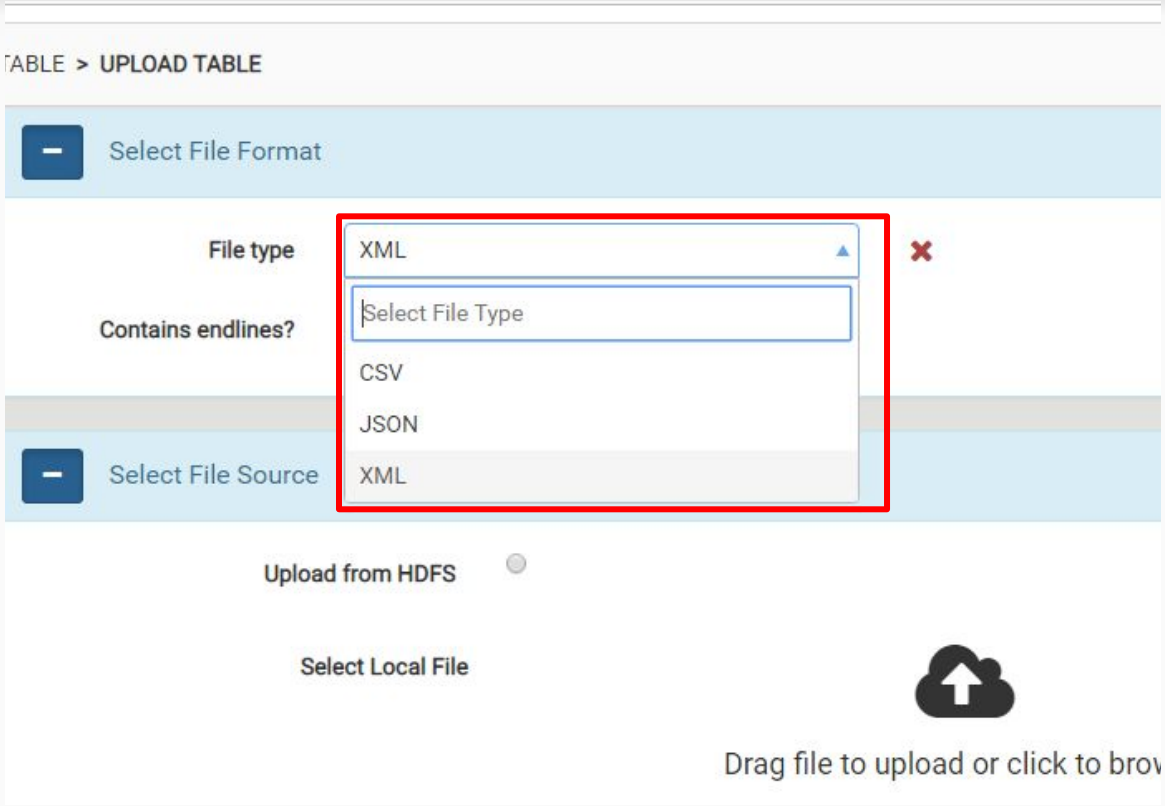
# Hive: let’s start



# Hive: let’s start (cont-d)



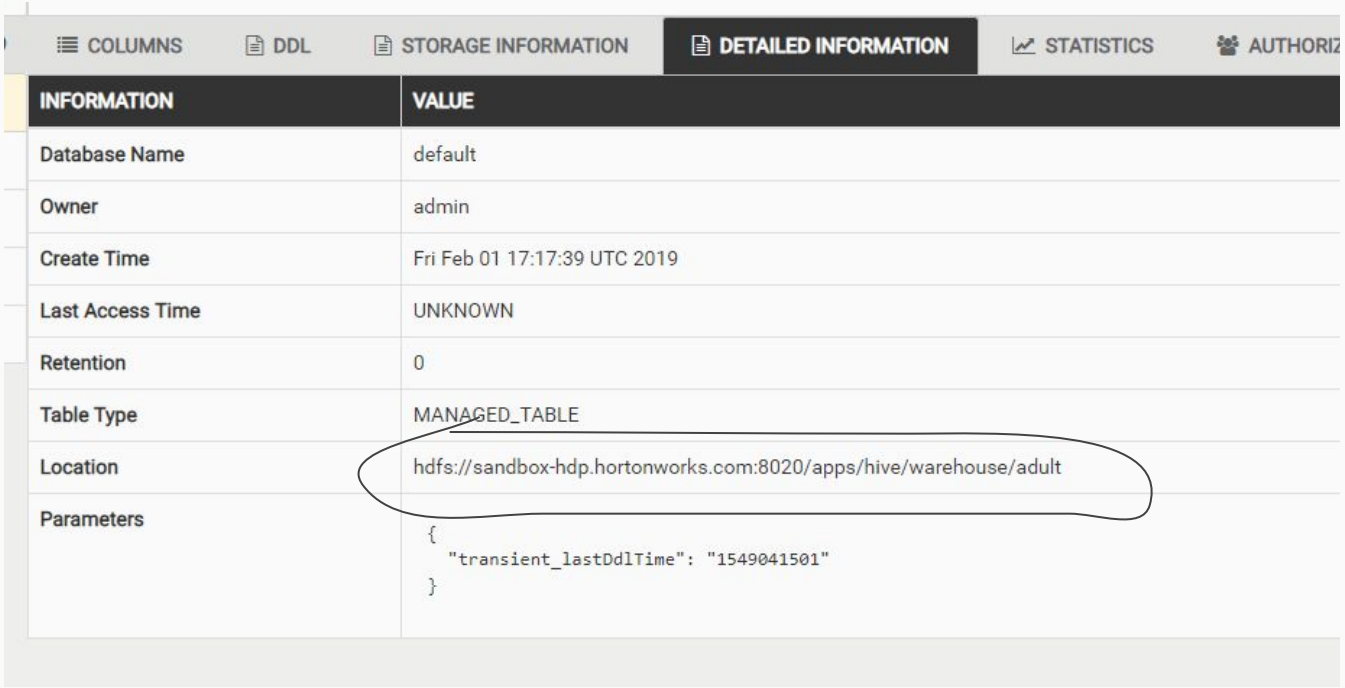
# Hive: let’s start (cont-d)



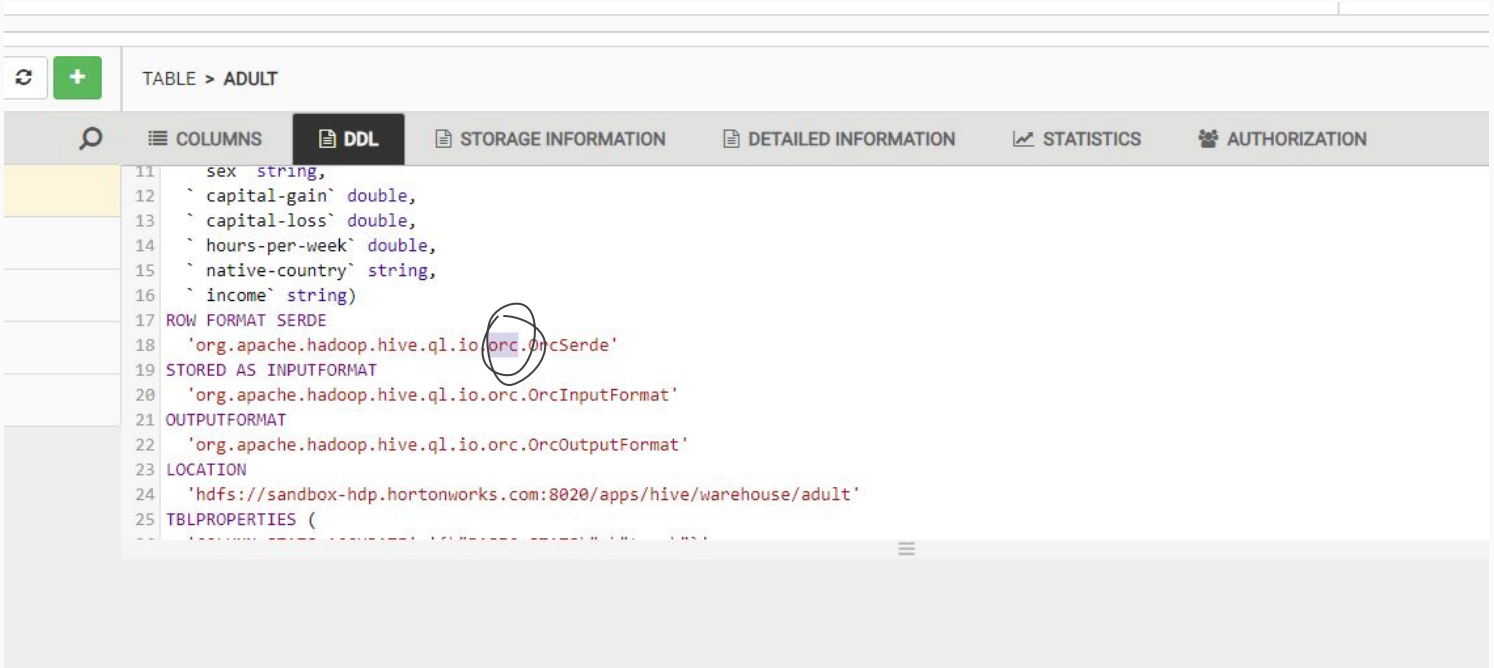
# Exercise

- Create a table in Hive from the data downloaded into HDFS in the first Exercise (**/dsets/census/adult.data)**
- You can either use HDFS path or upload it from your local machine (http://wolly.cs.smu.ca/tmp/adult.data)
- Don't forget to use first row as headers
- Once uploaded, try the following queries:
  - **SELECT * FROM adult LIMIT 10**
  - **SELECT count(*) FROM adult**

# What was done?

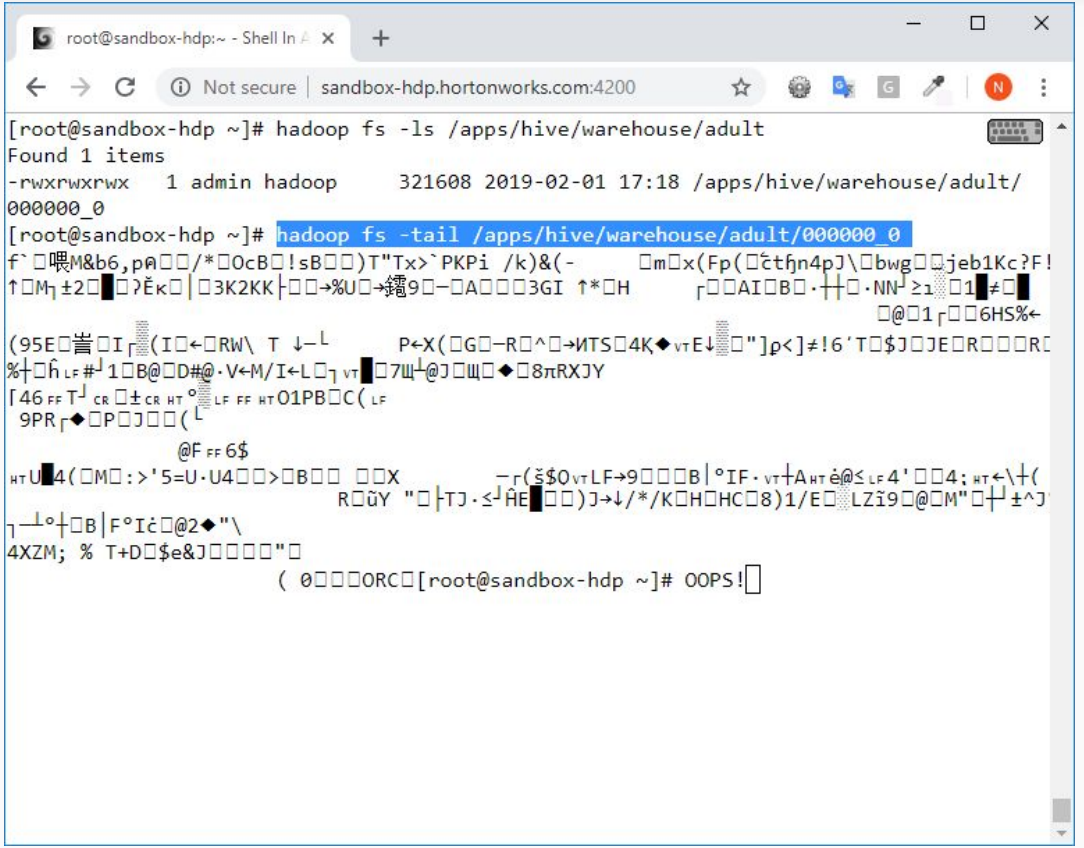


# What was done? (cont-d)



# What’s in the file?

**ORC file was created instead of just a simple text file. That’s a default Hive behaviour when we create tables from GUI from .CSV files**



# What is ORC?

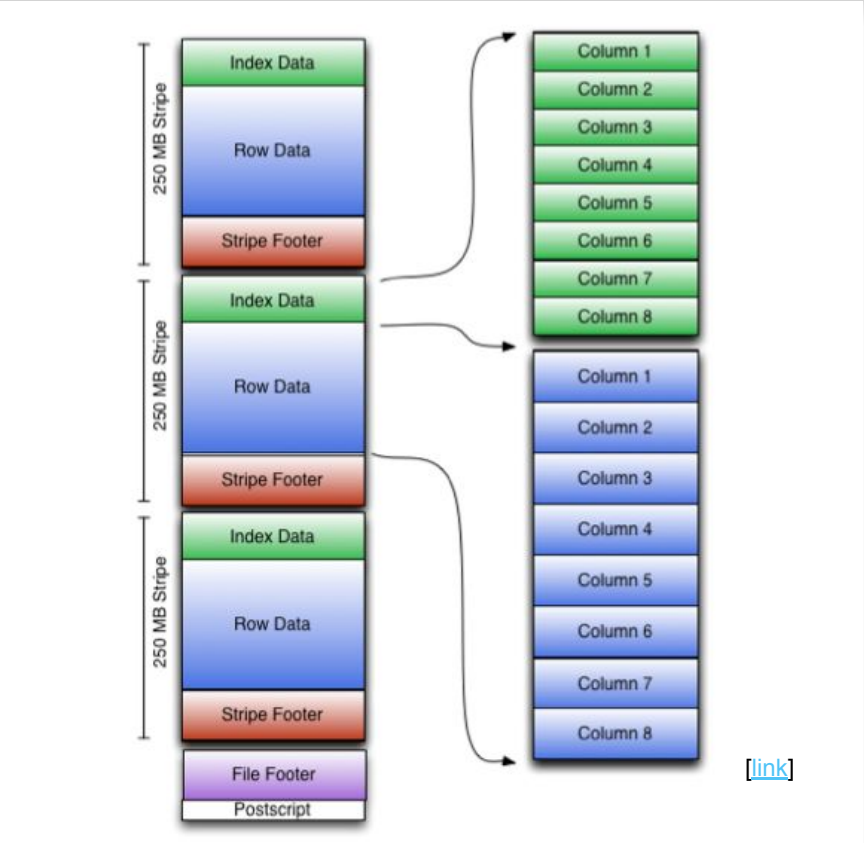
**Hive** (and some Hadoop ecosystem components) supports different types of files:

- Text files (as with Sqoop)
- AVRO
- ORC
- Parquet
- ..etc

# What is ORC? (cont-d)

**The Optimized Row Columnar (ORC)** file format provides a highly efficient way to store Hive data

More about the format: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC



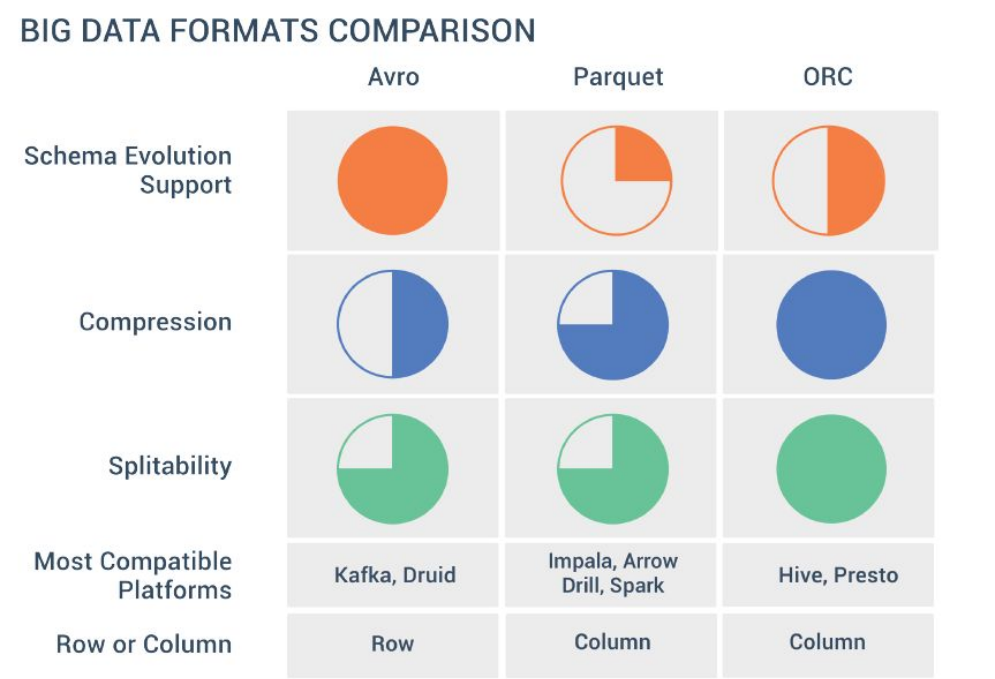

[link]

# More on file formats

Can we store data in text format?

Yes!

These formats were created just to speedup applications and to use disk space more efficiently



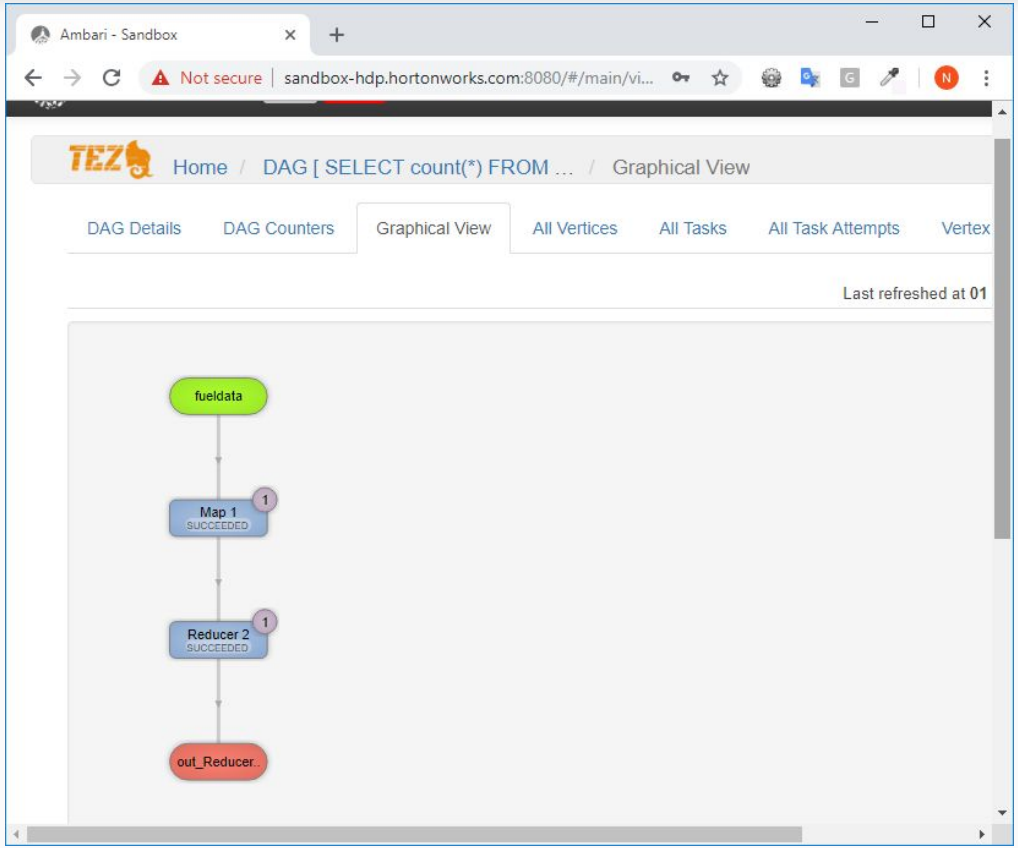

[link]

# Hive + sqoop: Exercise

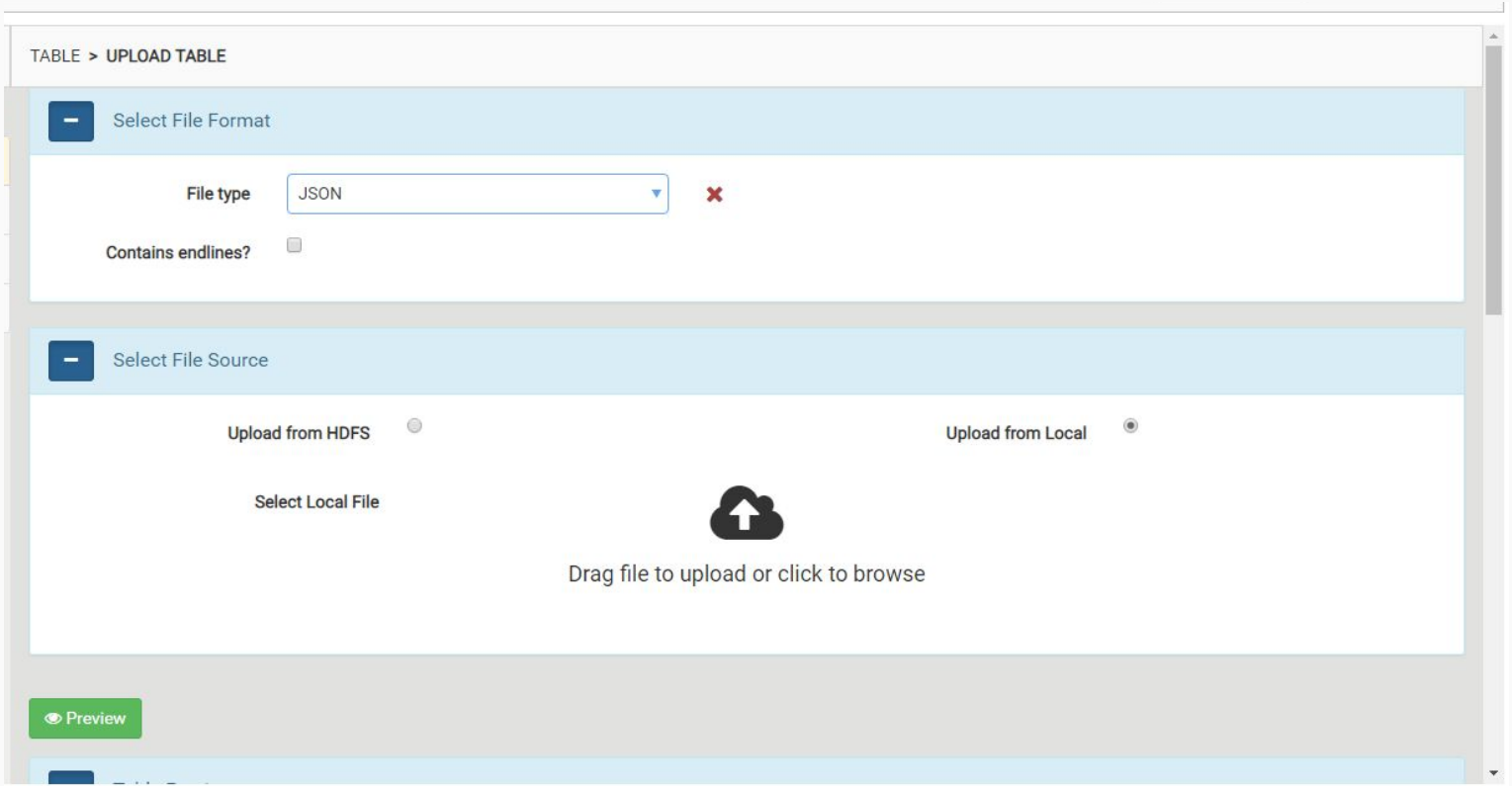
- Transfer the **fueldata** table from **dataset02** database from MySQL at dev.cs.smu.ca to **Hive** using sqoop
- Command: sqoop import --connect jdbc:mysql://127.0.0.1:23306/dataset02 --username your_username -P --table fueldata -m 1 ***--hive-import --create-hive-table --hive-table fueldata --target-dir '/apps/hive/warehouse/fueldata'***
- Important: **--target-dir '/apps/hive/warehouse/fueldata'**: warehouse dir is the Hive's default storage

# Hive + sqoop: Exercise

- What is a format of the imported dataset?
- Try: **SELECT count(*) FROM fueldata**
- Go to Tez View in Ambari and find a DAG for the last query



# Hive: working with semi-structured data

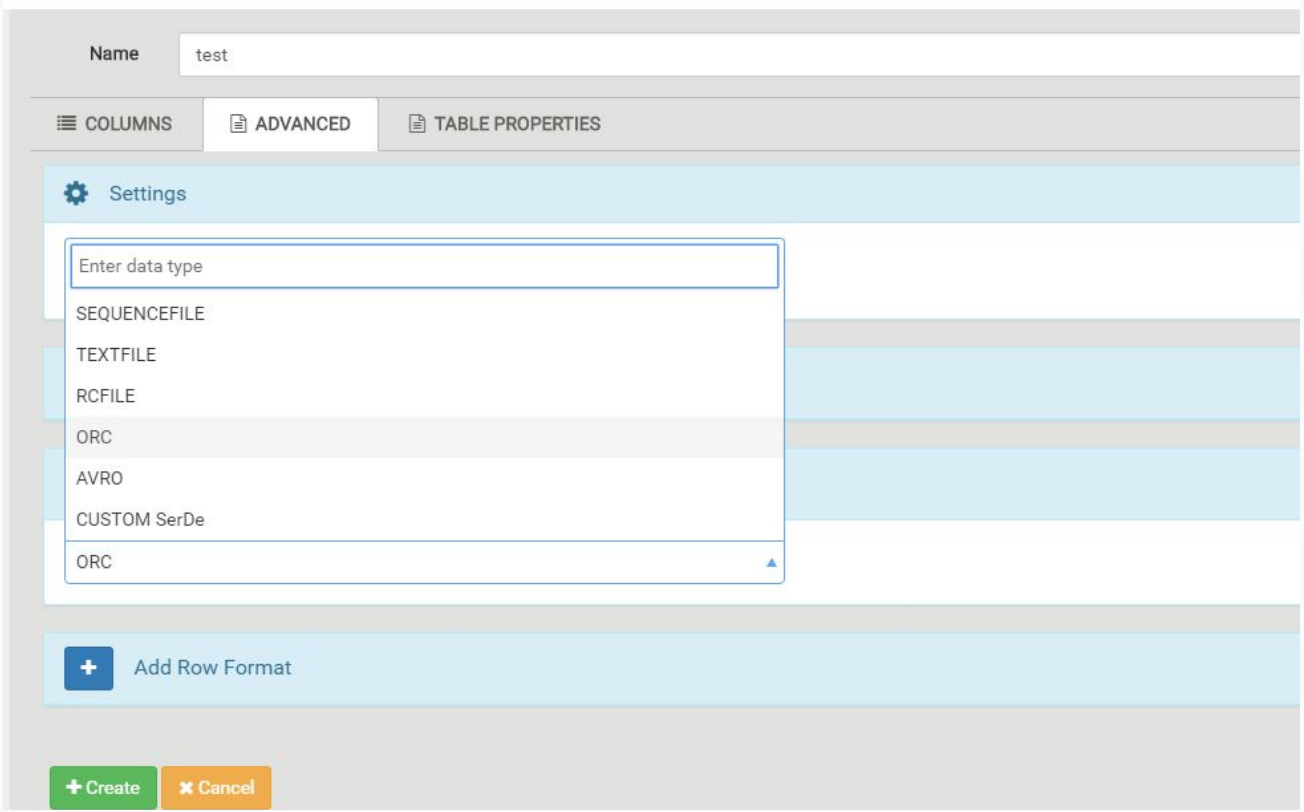


# Hive: working with semi-structured data

Table Preview

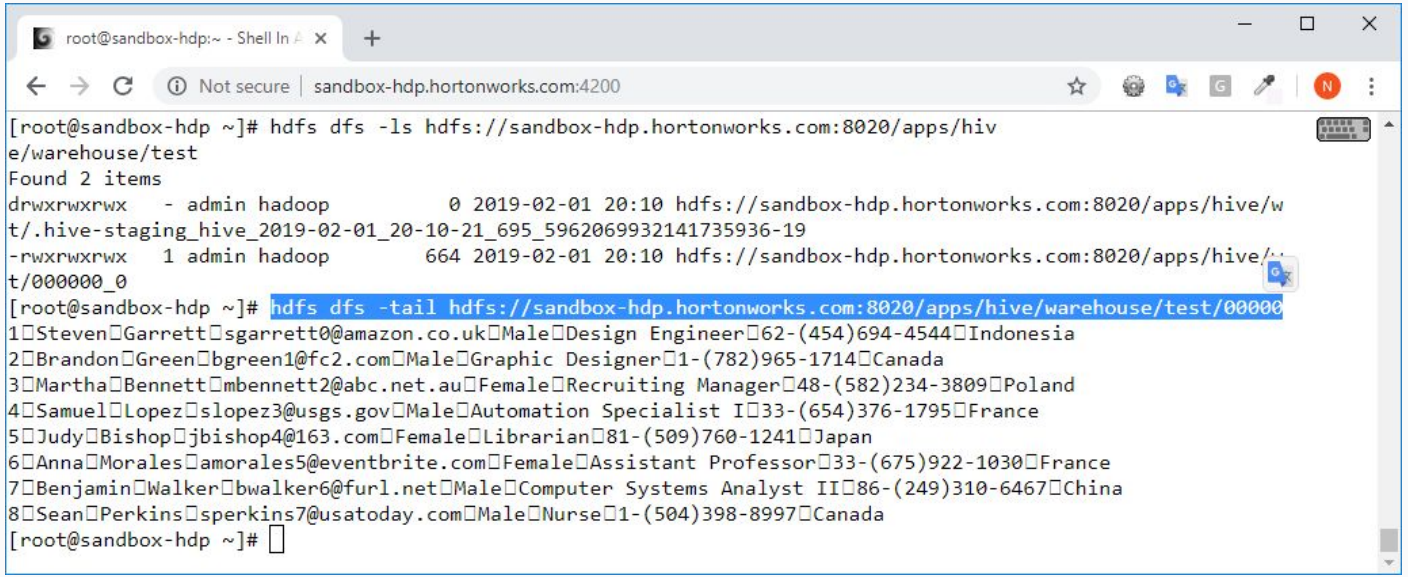
| id | first_name | last_name | email | gender | designation | phone | country |
|---|---|---|---|---|---|---|---|
| 1 | Steven | Garrett | sgarrett0@amazon.co.uk | Male | Design Engineer | 62-(454)694-4544 | Indonesia |
| 2 | Brandon | Green | bgreen1@fc2.com | Male | Graphic Designer | 1-(782)965-1714 | Canada |
| 3 | Martha | Bennett | mbennett2@abc.net.au | Female | Recruiting Manager | 48-(582)234-3809 | Poland |
| 4 | Samuel | Lopez | slopez3@usgs.gov | Male | Automation Specialist I | 33-(654)376-1795 | France |
| 5 | Judy | Bishop | jbishop4@163.com | Female | Librarian | 81-(509)760-1241 | Japan |
| 6 | Anna | Morales | amorales5@eventbrite.com | Female | Assistant Professor | 33-(675)922-1030 | France |
| 7 | Benjamin | Walker | bwalker6@furl.net | Male | Computer Systems Analyst II | 86-(249)310-6467 | China |
| 8 | Sean | Perkins | sperkins7@usatoday.com | Male | Nurse | 1-(504)398-8997 | Canada |

# Hive: working with semi-structured data



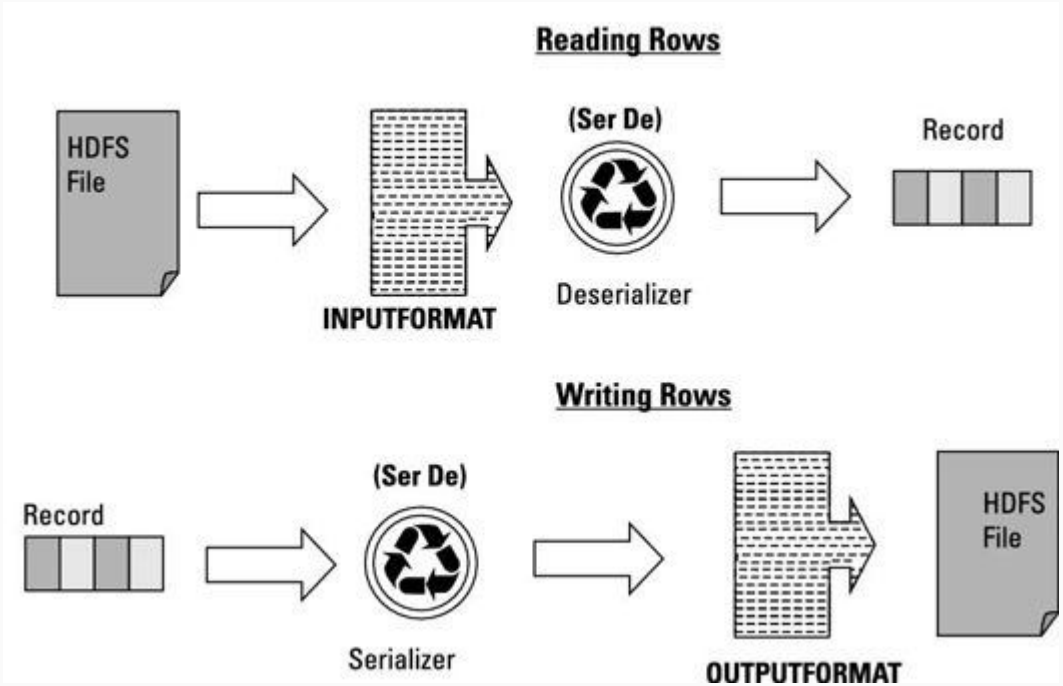
# Exercise

- Download a **json** file from http://wolly.cs.smu.ca/tmp/test.json
- Create a table in Hive based on this file. If you choose Text format you'll see that the new file in HDFS contains this data:



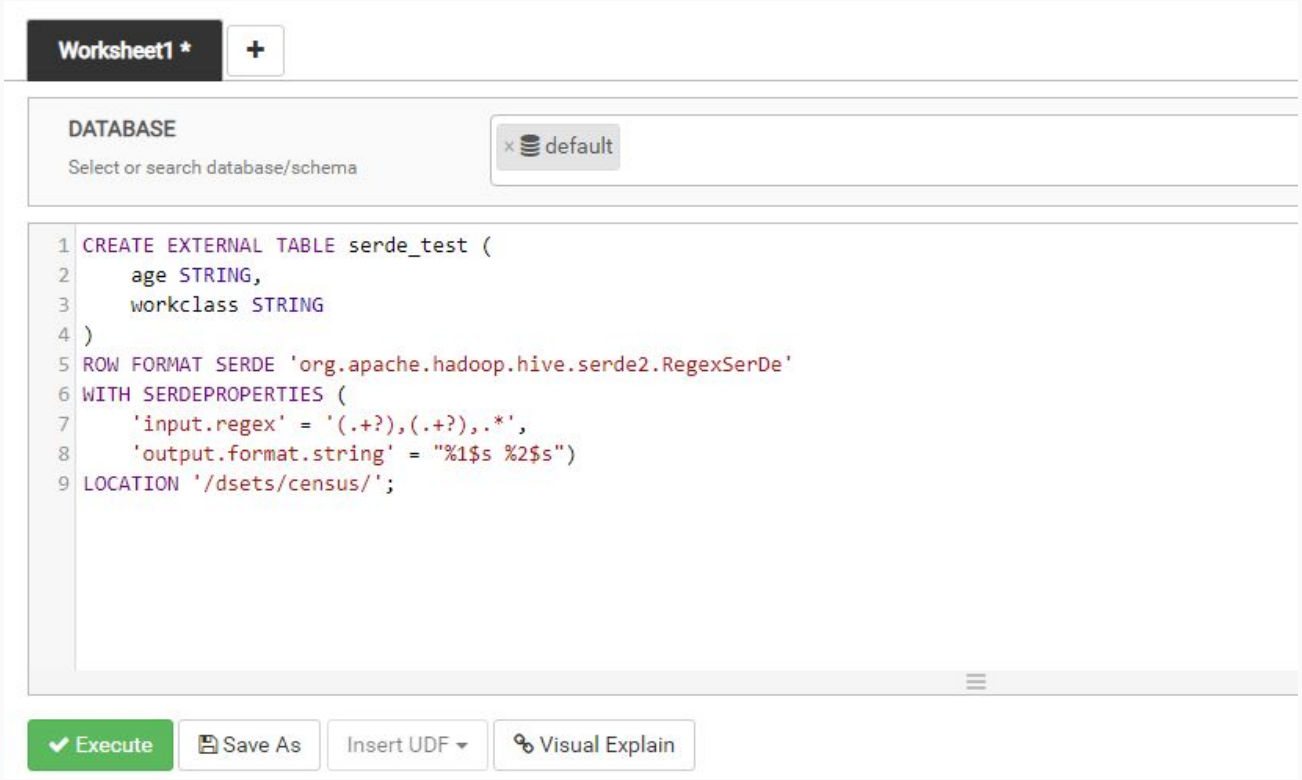
# Hive: SerDe

- SerDe = Serializer + Deserializer
- It is used by Hive to “convert” text (or binary) files to records that Hive works with and vice versa
- You can **write your own SerDe** in Java or you can use one of many predefined ones:
  - Avro
  - ORC
  - RegEx
  - Thrift
  - Parquet
  - CSV
  - JsonSerDe

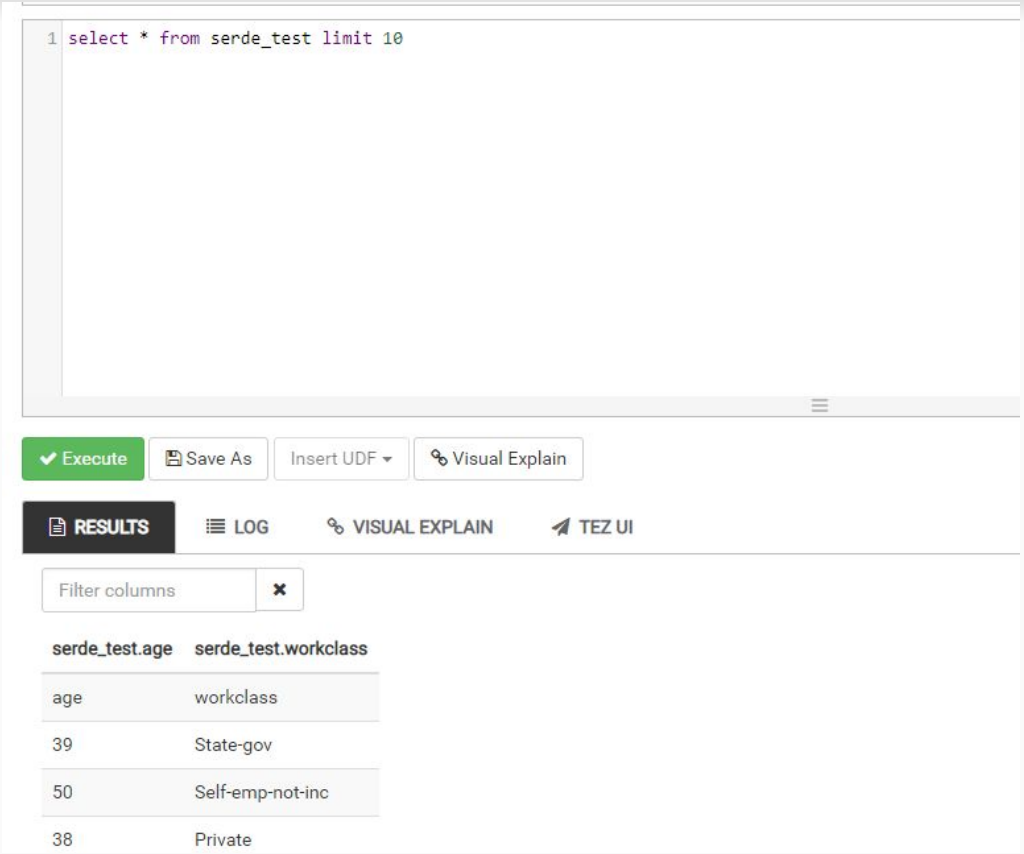



[link]

# Hive: SerDe: RegEx SerDe example



# Hive: SerDe: RegEx SerDe example



# Hive: SerDe: RegEx SerDe example explained

```
CREATE EXTERNAL TABLE serde_test (
    age STRING,
    workclass STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
    'input.regex' = '(.+?),(.+?),.*',
    'output.format.string' = "%1$s %2$s") -- remember capturing groups
in regex? Here they are!
LOCATION '/dsets/census/';
```

# Exercise: real use case

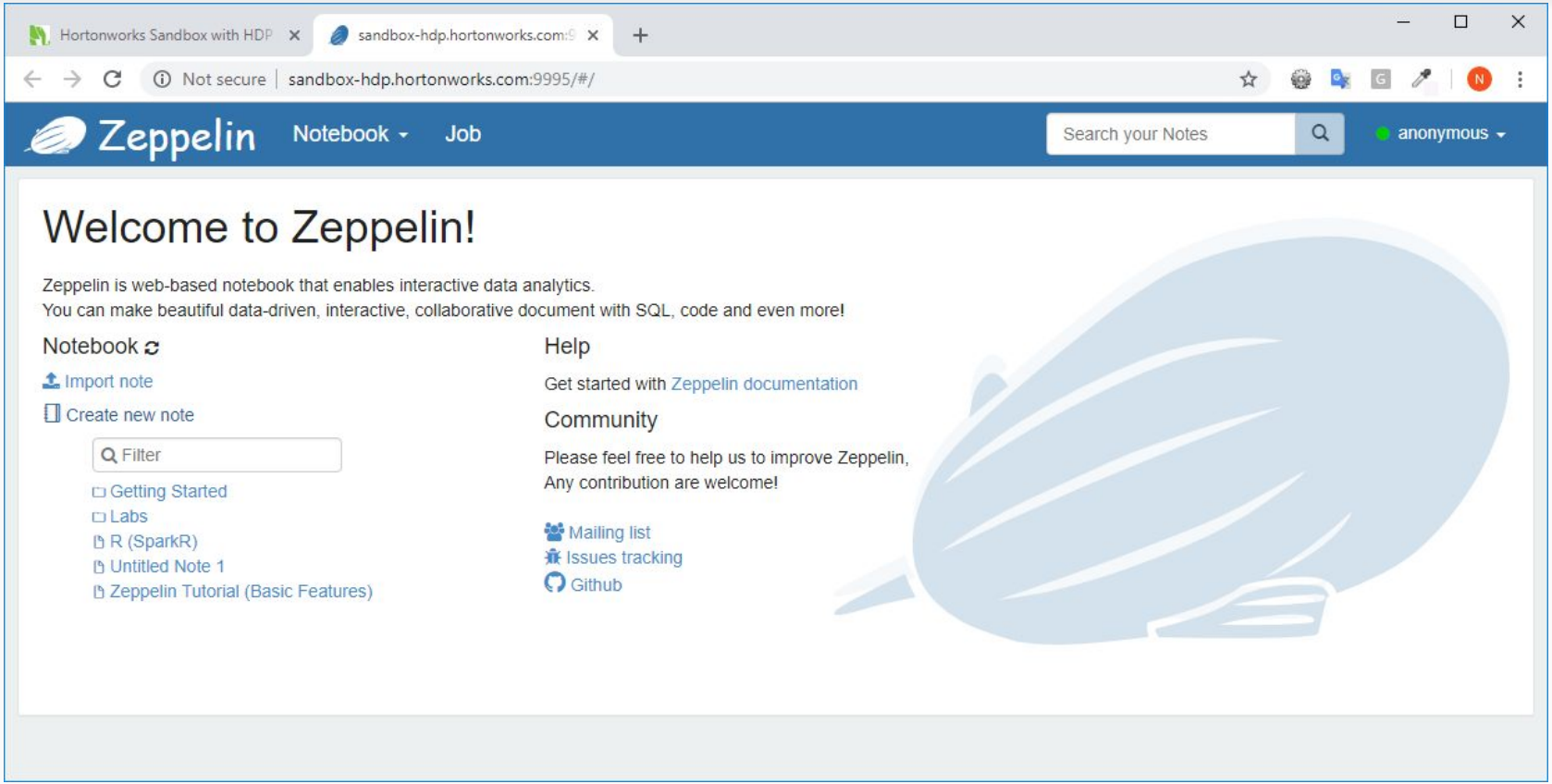
**Purpose: identify Linux usernames that are most likely to be hacked**

- Download an **archive** with auth logs from http://wolly.cs.smu.ca/tmp/auth.tar.gz
- Uncompress and put the files into hdfs folder "**/dsets/logs**"
- Create a Hive table based on the logs (using RegEx SerDe):
  - Name of invalid user
  - Ip address
- Using Hive identify top 10 usernames that hackers try to hack

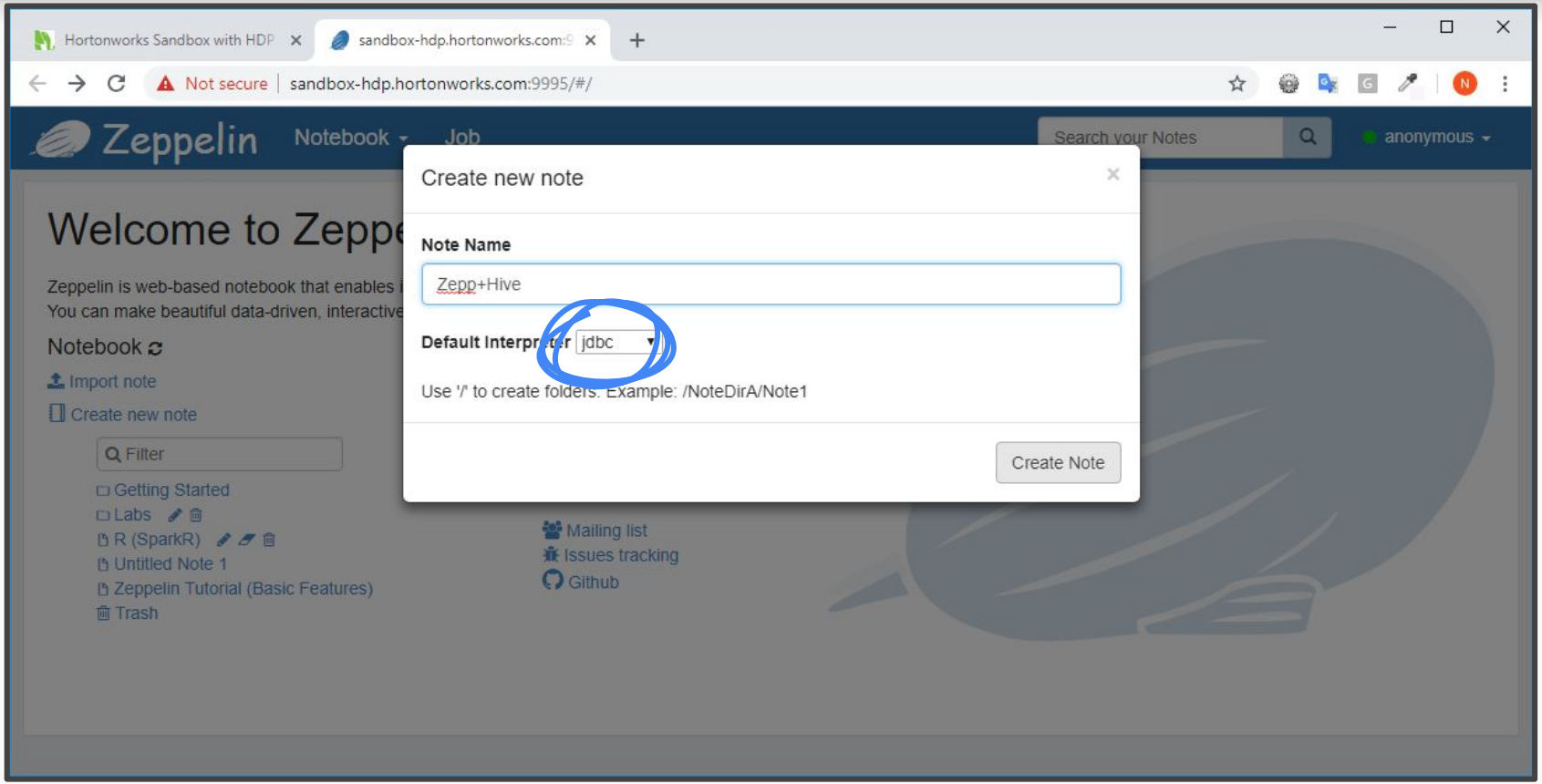
***Hint*: example of line with invalid username (format):**

```
Failed password for invalid user wi from 184.82.9.233 port 45796 ssh2
```

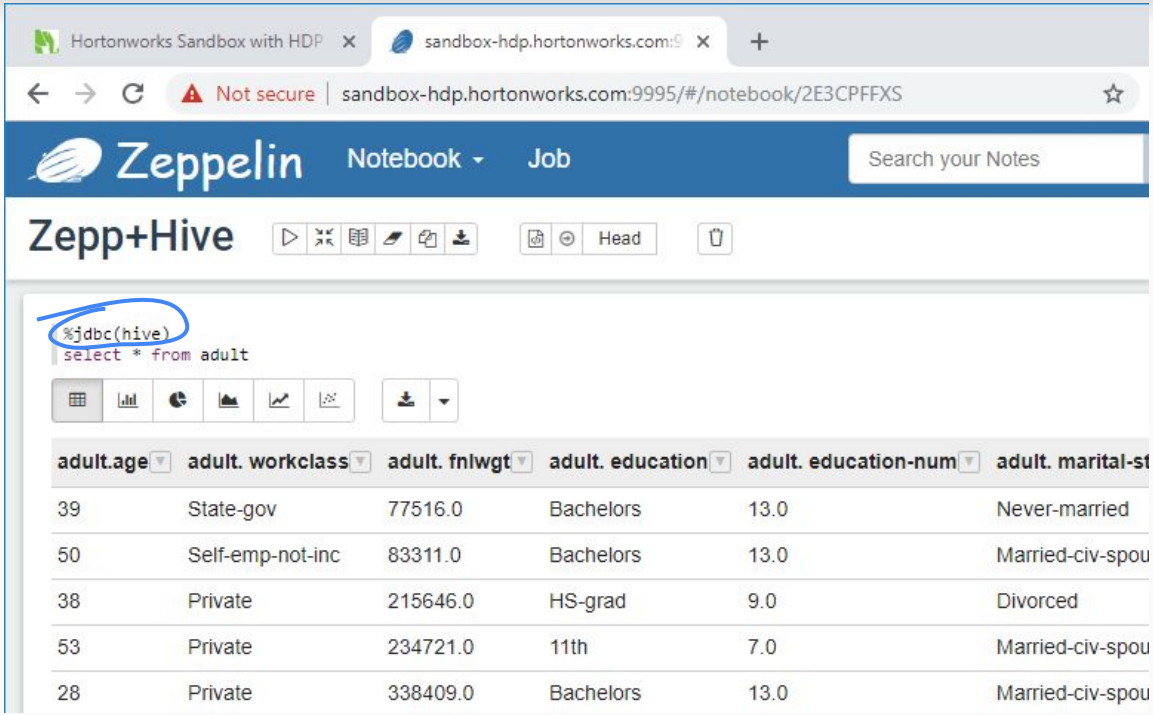
# Zeppelin


Hortonworks Sandbox with HDP
sandbox-hdp.hortonworks.com:9
Not secure | sandbox-hdp.hortonworks.com:9995/#/
Zeppelin
Notebook
Job
Search your Notes
anonymous
Welcome to Zeppelin!
Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!
Notebook
Import note
Create new note
Filter
Getting Started
Labs
R (SparkR)
Untitled Note 1
Zeppelin Tutorial (Basic Features)
Help
Get started with Zeppelin documentation
Community
Please feel free to help us to improve Zeppelin,
Any contribution are welcome!
Mailing list
Issues tracking
Github

# Zeppelin

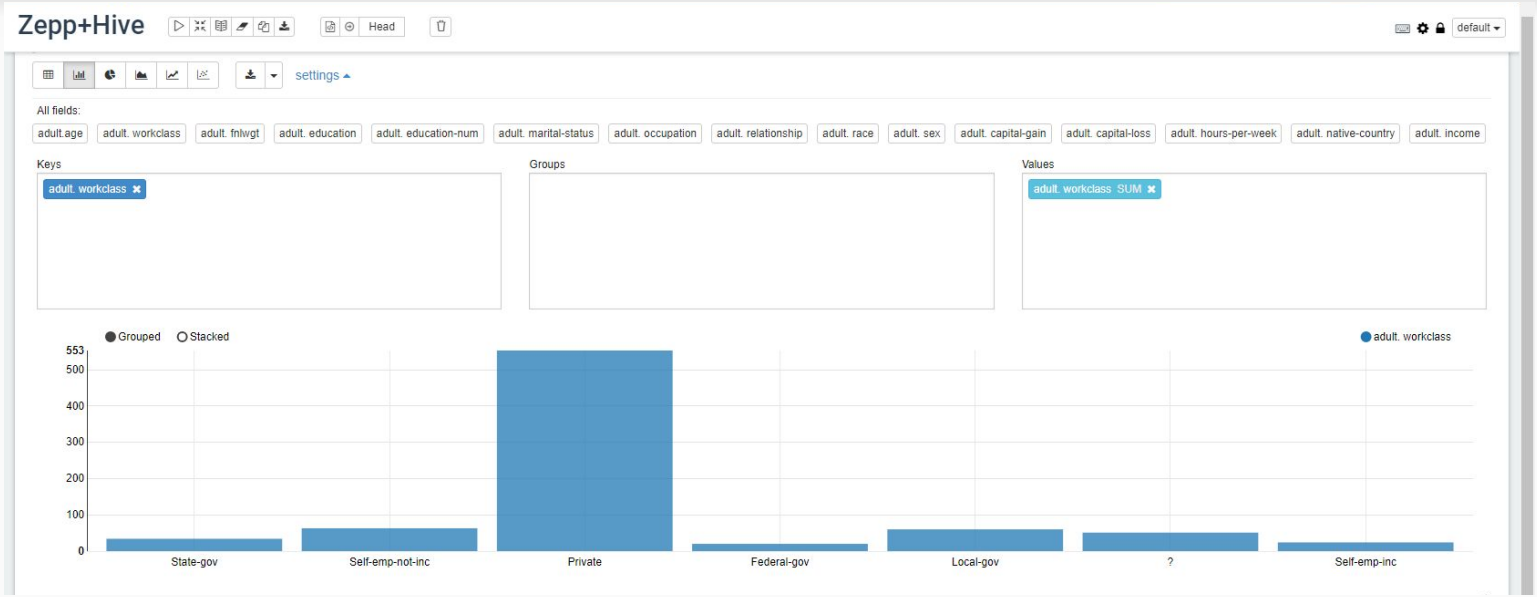

Hortonworks Sandbox with HDP
sandbox-hdp.hortonworks.com
Not secure
sandbox-hdp.hortonworks.com:9995/#/
Zeppelin
Notebook
Job
Search your Notes
anonymous
Welcome to Zeppe
Zeppelin is web-based notebook that enables
You can make beautiful data-driven, interactive
Notebook
Import note
Create new note
Filter
Getting Started
Labs
R (SparkR)
Untitled Note 1
Zeppelin Tutorial (Basic Features)
Trash
Mailing list
Issues tracking
Github
Create new note
Note Name
Zepp+Hive
Default Interpreter
jdbc
Use '/' to create folders. Example: /NoteDirA/Note1
Create Note

# Zeppelin



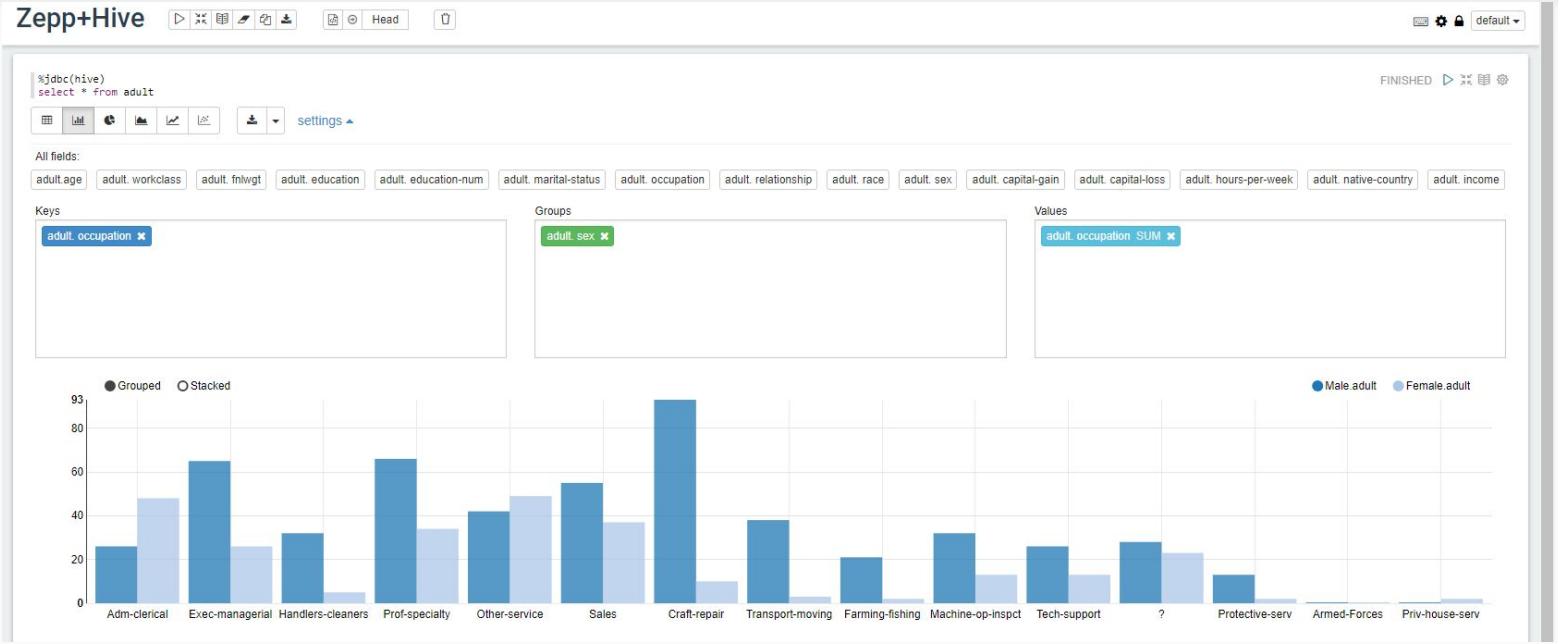

Don’t forget about “magic” statement **%jdbc(hive)** before **each** SQL-like statement

# Zeppelin




You generally don’t need to write complex SQL statements to get graphs. Just use UI

# Zeppelin




You generally don’t need to write complex SQL statements to get some basic insights from data. Just use UI

# Exercise

Open Zeppelin, use any available table in Hive from today's exercises (e.g., "adult"). Build multiple graphs that can give some insights from the data:

- Use at least 2 types of graphs
- Try to run different queries

# EOW

EOW

Thanks for your attention!