# Homework 2

Aim of homework is to simulate ETL process that is common on BigData project. This homework is divided on two parts to make it easier.

## Part 1

In part 1 you will need to implement simple data generator which saves data in file, read this file in Spark job and save in Hive.

1) Install Hortonworks Data Platform Sandbox (HDP) on VirtualBox/Docker(more preferable) on your computer. You will need Spark, Hive, HDFS, YARN, Zookeeper, Kafka, HBase, Amabari. If it requires something I have not specified - install it also. It’s recommended to have 32GB of RAM on you computer and allocate 16GB for Sandbox on VirtualBox. Let me know if you have less RAM.  
Which version to install:  
- 2.6.x: previous version of HDP with little bit outdated versions of software but should be sufficient for our needs. Choose this version if you don’t want to bother yourself with configuring Spark and Hive. It should work out of the box.  
- 3.x: newest version of HDP with more up to date versions of software. Additionally, Hive Warehouse Connector was introduced(<https://docs.cloudera.com/HDPDocuments/HDP3/HDP-3.0.1/integrating-hive/content/hive_hivewarehouseconnector_for_handling_apache_spark_data.html>). It means that by default Spark is not able to interact with Hive (we will need it) and additional configuration is required. We have not tested this approach and for now we do not know how it works\behaves. Choose this version if you would like to get more experience in configuring Spark and Hive.

2) Run your word count job from homework 1 on HDP Spark cluster using spark-submit command to confirm that HDP working fine. Checkout out YARN and Spark History Server for job details.

3) Implement simple Java\Scala\Python data generator(not a Spark job). It will simulate users that submits forms(requests) for loans/credits.

Input arguments:

* Format – string. Possible values: csv, avro, parquet. Sets format for output file. For now implement only csv functionality. For other formats make it throw exception.
* Rows count – int. How many records will output file contain.
* Output file name – string. Output file format – [output\_file\_name]\_[current\_date].[format] where [current\_date] is timestamp in YYYY-MM-DD\_HH-mm-ss format and [format] is csv/ avro/parquet.
* Output file path – absolute path where to save output file

Each records should contain next fields:

* Form\_id – string. Random UUID.
* Email – string. Format – [random\_string]@[domain], where [domain] could contain next values: gmail.com, yahoo.com, hotmail.com, zimbabwe.zw, urk.net
* Age – int. Random value in diapason 18 – 99
* Country – string. Random country name, max 10 countries
* Address – string. Random string, length 10 – 30
* Zipcode – int. Random value in diapason 15000 – 15999
* Houseowner – Boolean
* WasBankrupt – string. Possible values: yes, no, y, n, 0, 1, true, false, t, f, I was, I was not
* SubmittedAt – timestamp. Format – dd/M/yyyy HH:mm:ss
* Product – string. Possible values: Student loan, Mortgage, Easy Money
* LoanAmount – decimal. Random value in diapason 300.00 – 30000.00

4) Create Hive table with similar columns.

5) Implement Spark job to read generated file and save it to Hive table. Run it on HDP cluster using spark-submit command.

As result you should be able to see content of generated file in Hive table. You can use SQL web client provided by Amabari.

## Part 2

Second part of homework is aimed to fully simulate ETL process.

1) Schedule Data generator running on Sandbox every N minutes to generate file with 100 records. You can use Sandbox's OS tools(e.g. cron) or any approach you want. As result new files should be saved to folder in HDFS. You can achieve it by using HDFS API in your Data Generator program or by using HDFS CLI to move generated file to HDFS. Choose approach you like.

2) Create Hive managed table 'loan\_form\_submit' which will contain data from generated files. Use ORC format for this table. Columns and sources:

* form\_id – string, FormId.
* email\_hash – string, hash of Email field
* email\_domain - string, domain from Email, e.g. if email is iamgroot@gmail.com then email\_domain should be 'gmail.com'
* age – int, Age
* country - string, Country
* address – string, Address
* zipcode – int, Zipcode
* own\_house - boolean, Houseowner
* was\_bankrupt\_orig - string, WasBankrupt
* was\_bankrupt - boolean, if WasBankrupt is one of (yes, y, 1, true, t, I was) then true, if one of (no, n, 0, false, f, I was not) then false, else null
* submitted\_at\_utc – timestamp. SubmittedAt in ‘yyyy-MM-dd HH:mm:ss’ format.
* submitted\_at\_est – timestamp. SubmittedAt but in EST timezone in ‘yyyy-MM-dd HH:mm:ss’ format. Treat SubmittedAt as in UTC timezone.
* product - string, Product
* loan\_amount - decimal, LoanAmount
* submitted\_at\_date - string, date part of SubmittedAt in ‘yyyy-MM-dd’, e.g. if SubmittedAt = '2020-10-10 12:35:12', then submitted\_at\_date = '2020-10-10'. Use this column as partition for Hive table.

3) Modify Spark job to read files from HDFS(use Dataframe/Dataset), perform transformation and insert them into new Hive table. Implement two approaches for transformations:

- via Spark SQL (register temp view and use SQL to query it)

- via Dataframe/Dataset API with using functions in org.spark.sql.functions.\_

4) Implement functionality for tracking which files were already processed. You can achieve it keeping this info in Hive audit table(create by yourself) and checking it from Spark job to find out which files to process. As result your Spark job should process only new files and write data in Hive table.

5) Schedule Spark job running with Oozie\*. It should run it less often than Data Generator, e.g. if Data Generator is invoked every 1-2 minutes, then Spark job should be invoked every 5 minutes. Checkout Hive table and make sure that table is populated with new data constantly.  
  
\* if you spent significant amount of time trying schedule job with Oozie without success, you can just schedule it with another approach of your choose, e.g. cron in OS where HDP is running.