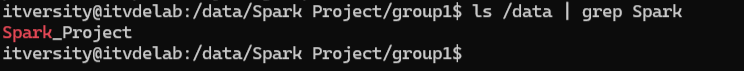
**Business Assumptions**

* We will assume that data (group of files) arrives to Local File System inside a folder named *whatever*, which means we will have a new folder every hour.

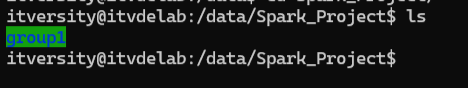
A screenshot of a computer

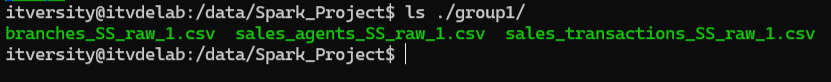
Description automatically generatedThis is how the folder will look like after 6 hours

* We will assume source data is stored locally on a folder named *“Spark\_Project”* which itself is inside a Folder named *“/data”*.



* *“Spark Project”* Folder contains a Single Folder for each group of files (Batch), initially it contains a single folder.



* Each folder contains the 3 files coming from source systems.
* After each group (Folder) is uploaded into the Data Lake *it will be archived*:
  + We will keep archived data in a folder named *“Archived”*.
  + We will use a specific data retention policy for this folder, let’s say we will keep data for 1 week for disaster recovery.
* Data comes to Local File System in a **non-idempotent** manner (Same group/batch can be pushed twice into LFS).

**Load Data Into HDFS**

* Source system pushes 3 files every hour which means we have various portioning options for storing Row Data in the Data Lake.
* We will divide the data into **3 Folders** *(one for each file).*
* Each Folder will store the historical data for each file.
* We will add some *quality columns* to the files before uploading them into the Data Lake.
* We will store the Row Data inside HDFS in a folder named *“/Spark\_Project/data/Q\_company/”*
* The structure of *“HDFS:/ Spark\_Project/data/Q\_company/”* will go something like this:

/Spark\_Project/data/Q\_company/  
├── branches/  
│ ├── 2024-06-09-00  
│ ├── 2024-06-09-01  
│ ├── ...

│ ├── 2024-06-09-23

│ ├── ...  
├── sales\_agents/  
│ ├── 2024-06-09-00  
│ ├── 2024-06-09-01  
│ ├── ...

│ ├── 2024-06-09-23/

│ ├── ...  
├── sales\_transactions/

│ ├── 2024-06-09-00

│ ├── 2024-06-09-01

│ ├── ...

│ ├── 2024-06-09-23

│ ├── ...

* This structure will make it easier for us to read the data for each file and clean/process it.
* We will add **3 Quality** Columns to each file:

|  |  |
| --- | --- |
| * + Load Time. | * + File Group. |
| * + Load Source. | * + Source Path. |

* The log file will follow the following structure: ***TIMESTAMP LOGLEVEL MESSAGE***

**Loading Script:**

* The automation script to upload the files to HDFS can be found in a file named *“Data\_Load.PY”.*

Create HDFS Directories

Get Files to Add to HDFS

Upload Files to HDFS

Archive Files Locally

* Now we will create an AirFlow Script to trigger the Python Automation code whenever a new files group arrives.
  + - The airflow code can be found in a file named *“airflow\_script.py”.*
    - The script runs every hour and triggers a bash script to fetch the data to "/data/Spark\_Project".
    - It listens for any incoming files/directories arriving and when a new file group arrives, it triggers the Python Automation Script to upload the group to Data Lake.
    - This moves files from LFS into HDFS.
* The following is the Bash Script to fetch the data to "/data/Spark\_Project" (From LFS to LFS)

# !/bin/bash  
  
# Source directory  
SOURCE\_DIR="/Source\_To\_File\_Groups"  
  
# Destination directory  
DEST\_DIR="/data/Spark\_Project/"  
  
# Log file path  
LOG\_FILE="/data/Spark\_Project/log\_file.log"  
  
# Function to move directories  
move\_directories()  
{  
 echo "$(date '+%Y-%m-%d %H:%M:%S') INFO: Starting to move directories from $SOURCE\_DIR to $DEST\_DIR" >> "$LOG\_FILE"  
  
mv "$SOURCE\_DIR"/\* "$DEST\_DIR"  
  
echo "$(date '+%Y-%m-%d %H:%M:%S') INFO: Moved directory $dir\_name to $DEST\_DIR" >> "$LOG\_FILE"  
  
  
echo "$(date '+%Y-%m-%d %H:%M:%S') INFO: Finished moving directories" >> "$LOG\_FILE"  
}  
  
# Run the function  
move\_directories

**PS**

* When you run the automation script to upload multiple file groups it won’t upload duplicate files, so you may find some non-uploaded files.
* The code is designed this way to avoid duplicate files uploaded by mistake.
* To disable this, it is a manner of commenting a single line of code!
* But we thought as long as the script is triggered each hour (for a single file group) this feature may be useful.

**Batch Processing**