**Alert System – Walk Through**

**Scenario Interpretation –**

For the sensor data flow alert system, below are the scenario interpretations done for code development:

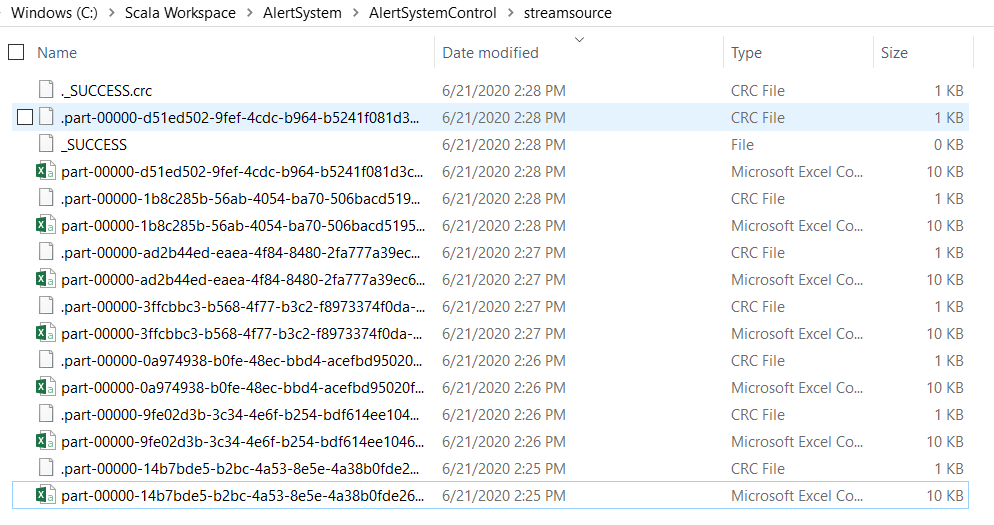
1. Sensor data will flow from a streaming source every 30 seconds
2. Spark streaming will consume the data from streaming source each and every time new batch is available, enrich the data with few fields that might be helpful for reporting purpose and dump it to the staging layer
3. Another spark application will read the data from the staging layer, impute the missing data and make sure the stream data is written with all the field values to the target layer for consumption

**Tweaks Made –**

1. To replicate a streaming scenario, the shared CSV file is sub divided into 7 partitions based on 7 days available in the file and each day is produced as an incoming stream to the source folder every 30 seconds.
2. A new field called Iterator is created inside the file to iterate the dataset against 7 different batches. Modified file available: AlertSystem 🡪 AlertSystemControl 🡪 sensor\_data.csv.
3. Alternatively, the excel could have been broken down into every 15 minutes stream interval with every single line as a single feed from sensor source, but due to time constraint this has been avoided.

**How this alert system operates –**

1. Three scala programs should run simultaneously to simulate the scenario. Each of these three programs are tightly coupled with source, stage and sink and the run steps are explained in a separate section as well as inside Readme.md file.
2. Program [createStreamSource.scala](https://github.com/saikatsengupta89/AlertSystem/blob/master/AlertSystemControl/src/main/scala/alert/system/createStreamSource.scala) will simulate a streaming source and write data in the directory **streamsource** every 30 seconds after reading from the source file sensor\_data.csv. This creates a continuous streaming source for 3.5 minutes ((30\*7)=210 seconds)



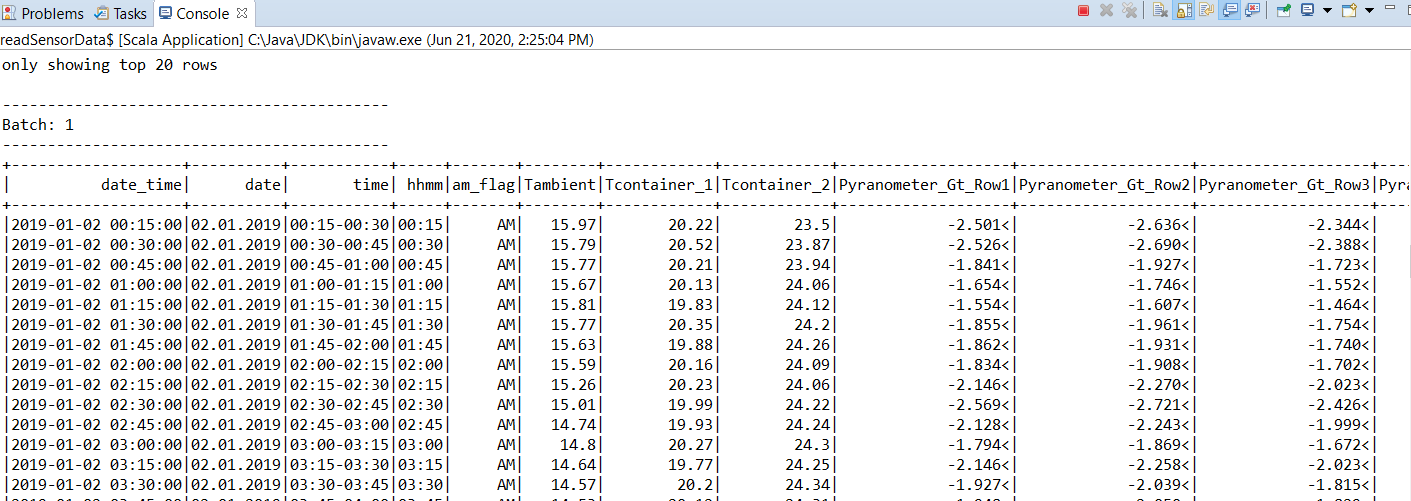
1. Program [readSensorData.scala](https://github.com/saikatsengupta89/AlertSystem/blob/master/AlertSystemControl/src/main/scala/alert/system/readSensorData.scala) is a streaming spark application which is coupled with the **streamsource** directory. As soon as data arrives inside this directory, the program will read the data and does following activities:

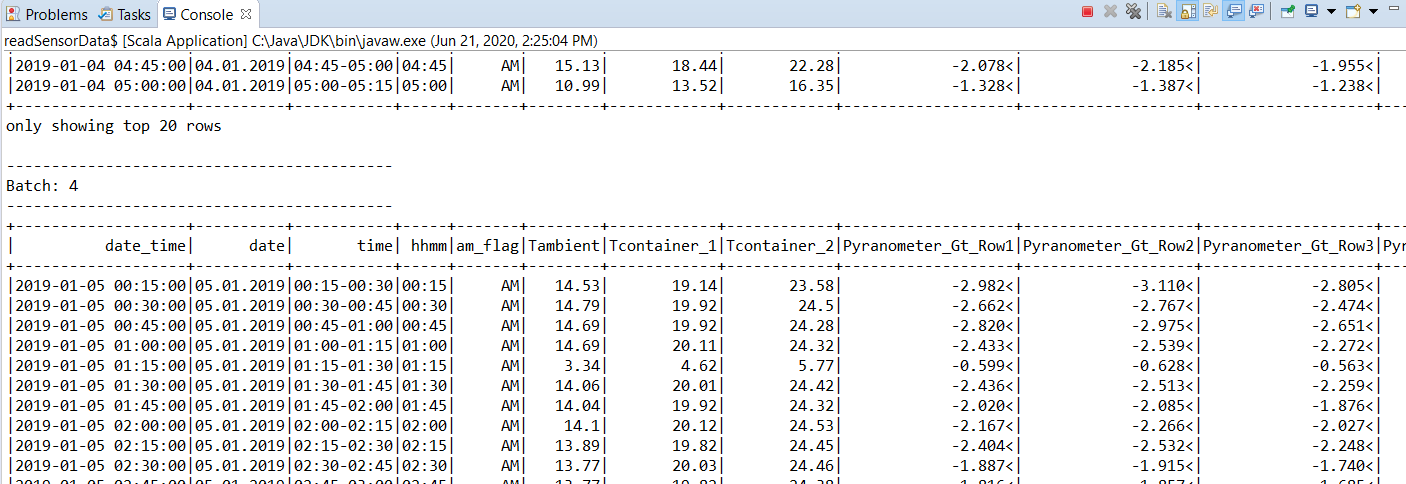
🡪 Enhance the data with additional fields like datetime, hhmm and am\_flag for end user reporting.

🡪 Create three stage sinks inside directory **streamstage**: streamdata, extremedata and missingdata

🡪 Send email alerts to user configured

1. Below screenshot showing batch data getting processed by **readSensorData.scala** with enriched fields like datetime, hhmm and am\_flag



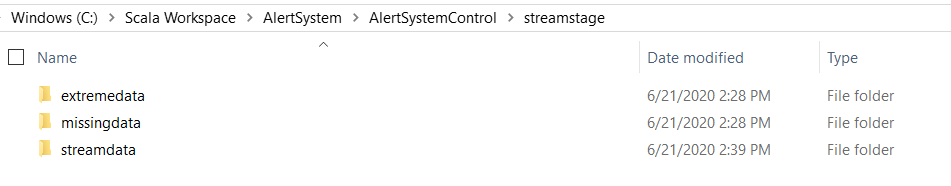


1. Once the batches are processed, the program will dump data into three separate folders.

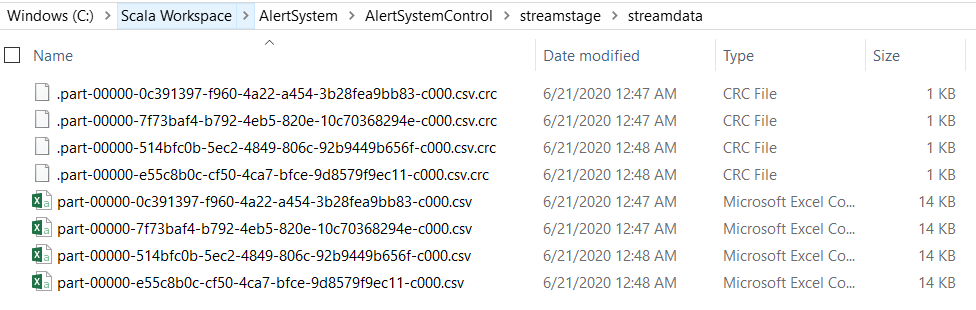
🡪 For incoming complete stream data, it dumps into **/streamstage/streamdata** folder.

🡪 For extremedata like (if a value of Tcontainer\_2 column exceed 35 or become below 10), such exception data subset is dumped into **/streamstage/extremedata**

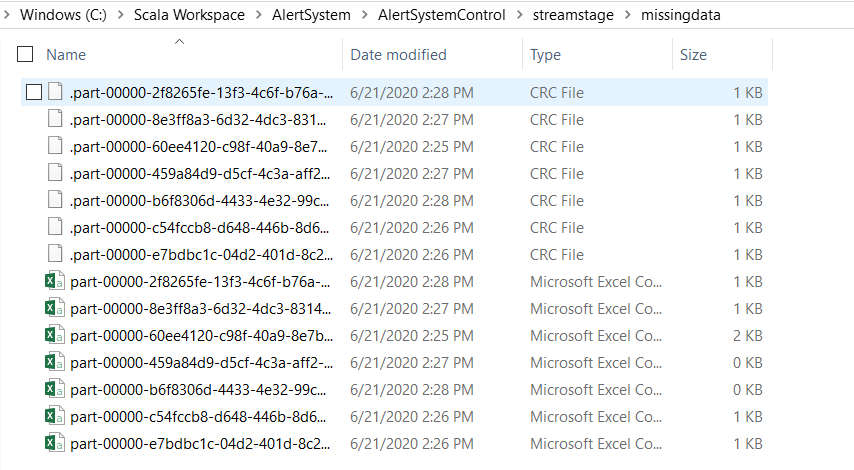
🡪 For missingdata, such subset is dumped into **/streamstage/missingdata**



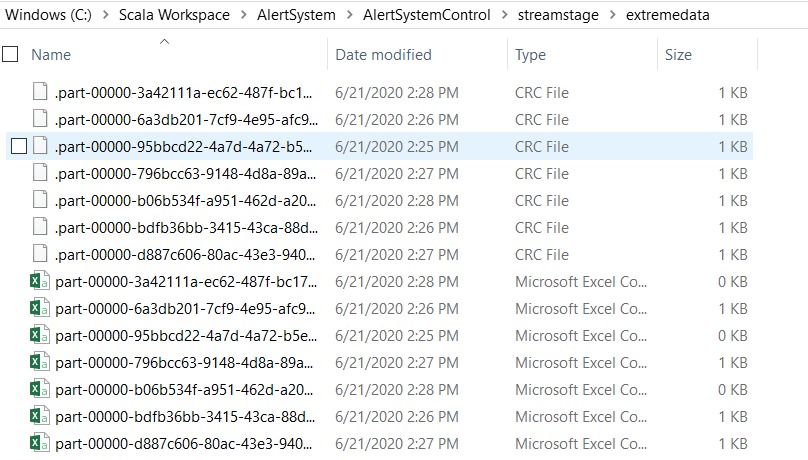
Inside stage - streamdata,



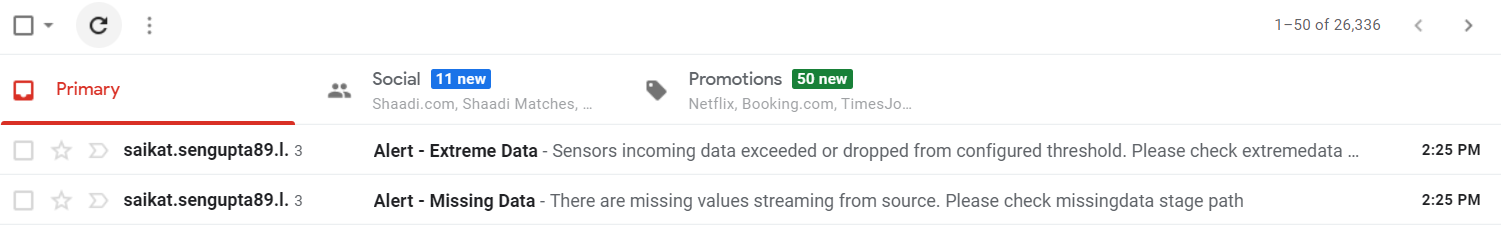
Inside stage – missingdata

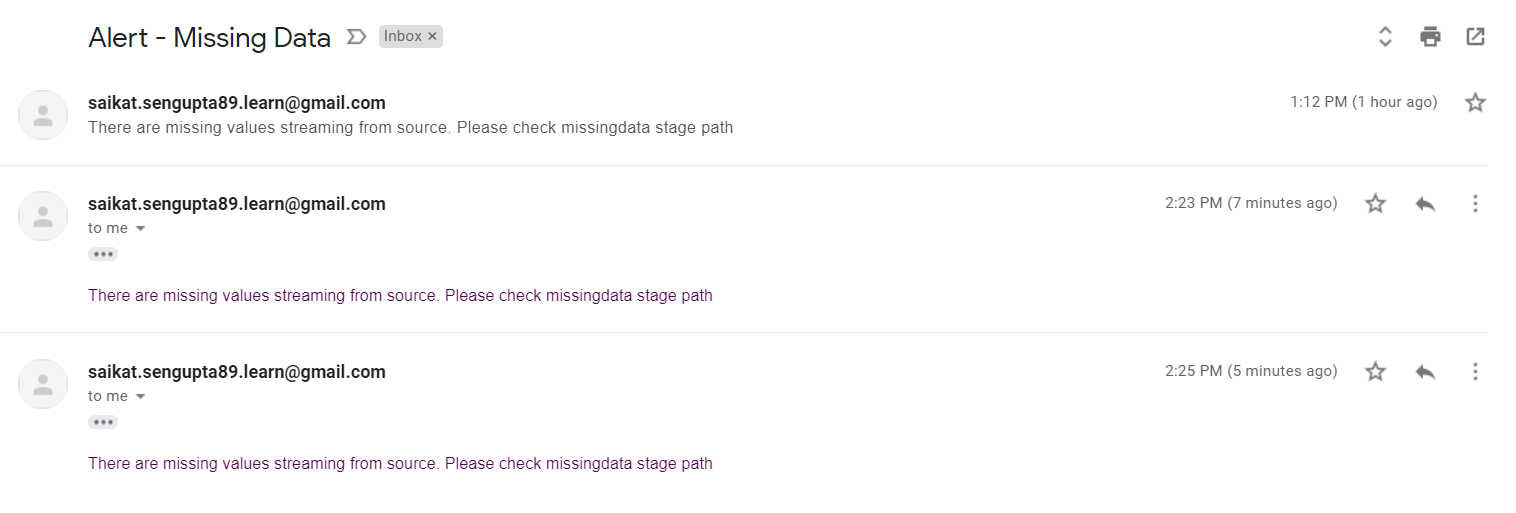


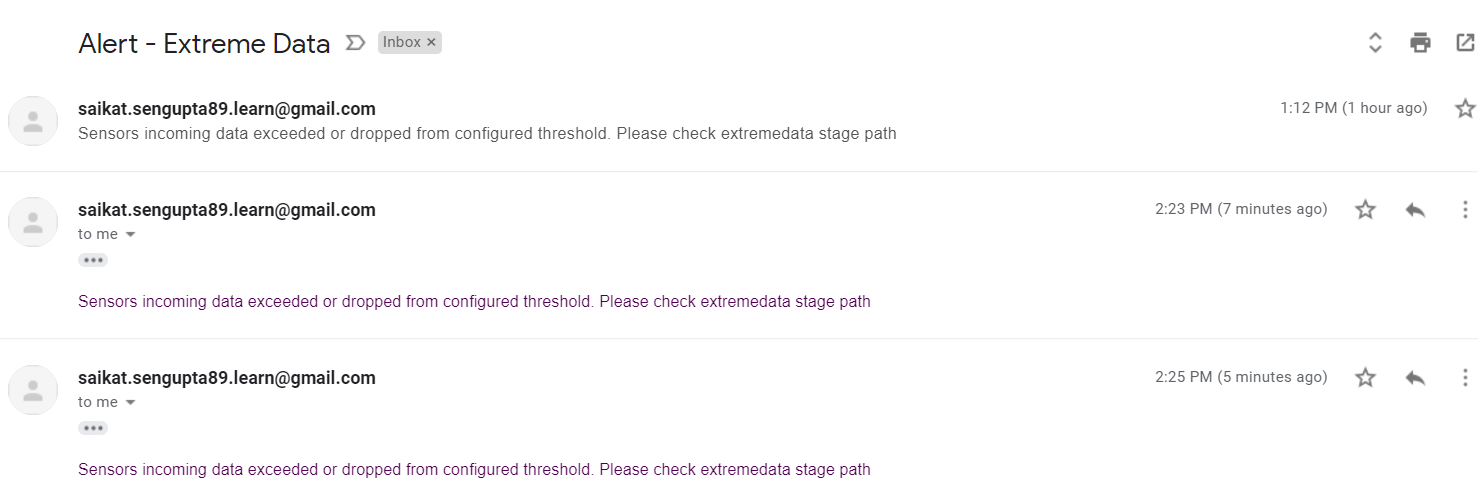
Inside stage – extremedata,



1. While processing missing data or extreme data, those exceptions are send as email alerts as monitoring purpose if required. Below screenshot on the same.







1. Program [imputeData.scala](https://github.com/saikatsengupta89/AlertSystem/blob/master/AlertSystemControl/src/main/scala/alert/system/imputeData.scala) will read the stream data, missing data and extreme data from the staging folders every 10 seconds, do imputation on the sensor data to substitute missing values and write sensor data, extreme data and missing data into three different target folders.

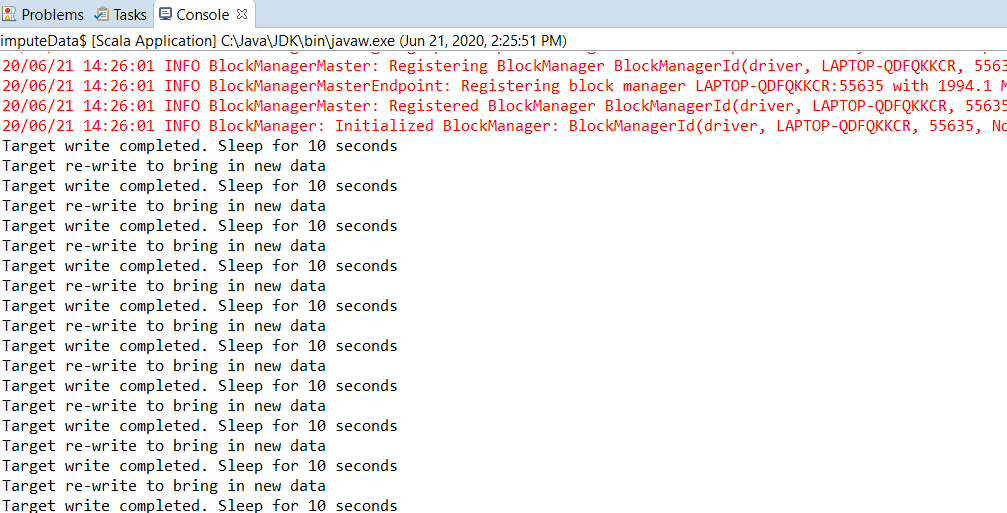
**Final target results:**

**/streamtarget/streamdata** – Will have full data from sensors with missing values being imputed based on below logic:

🡪 If values are missing for fields in a specific row, am using **lag** function to consider the previous value. This can be changed based on business definition

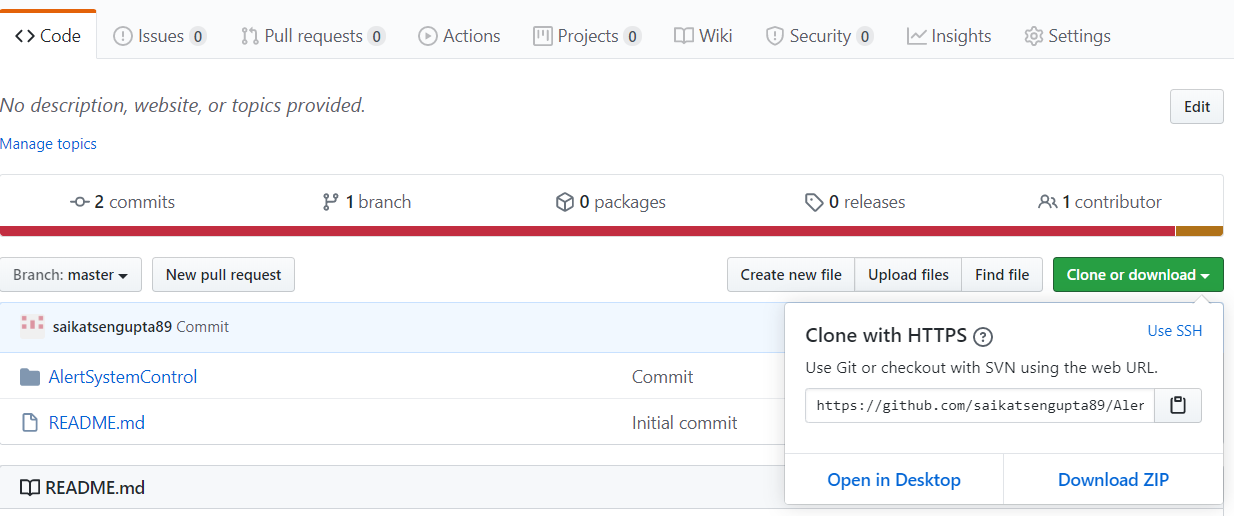
🡪 If multiple consecutive rows are missing, then lag function won’t help. This is achieved using mean of all those individual fields for that hour and putting the same mean value for all the fields in the missing rows.

Below screenshot showing the target data load gets triggered every 10 seconds.



**Steps to run alert system operates –**

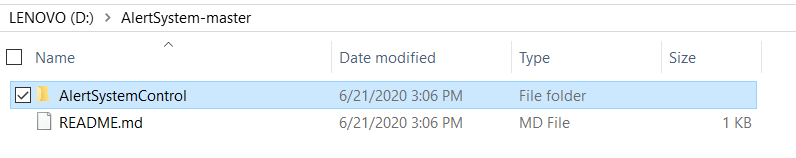
1. Download the zip as shown below.



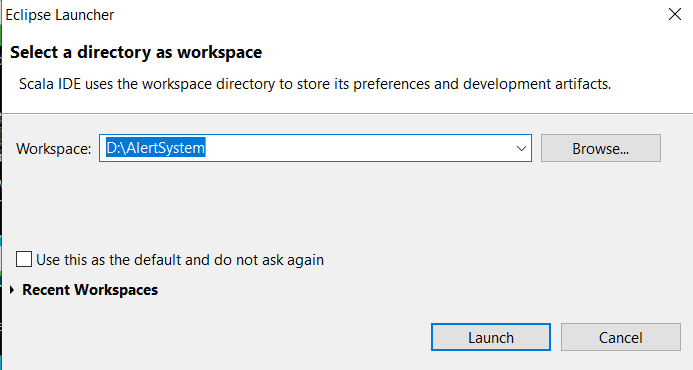
1. Extract the zip. Below folder will get created.



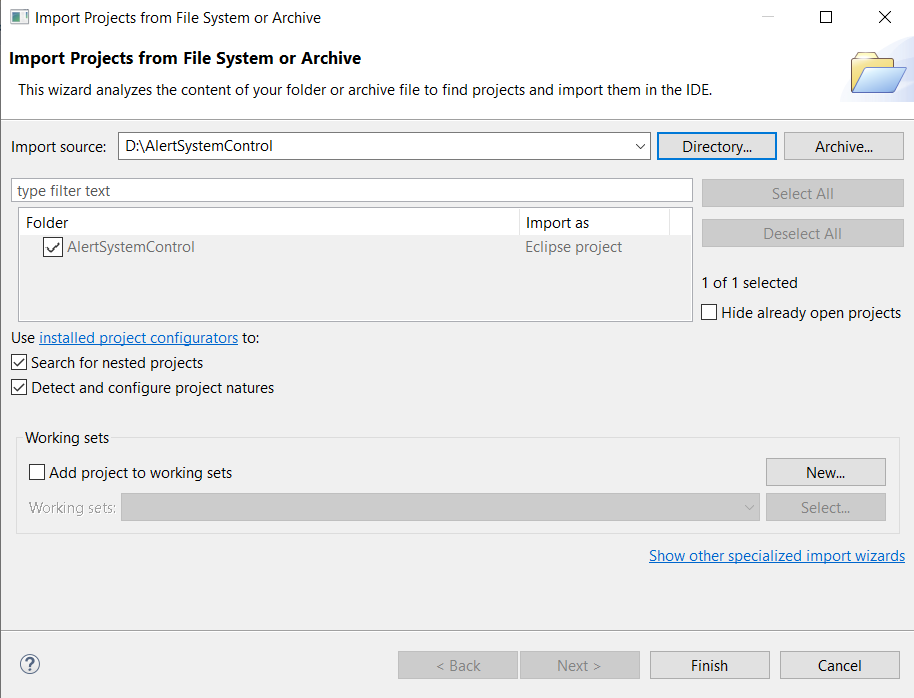
1. Take the AlertSystemControl project folder out and place it in c drive or d drive.



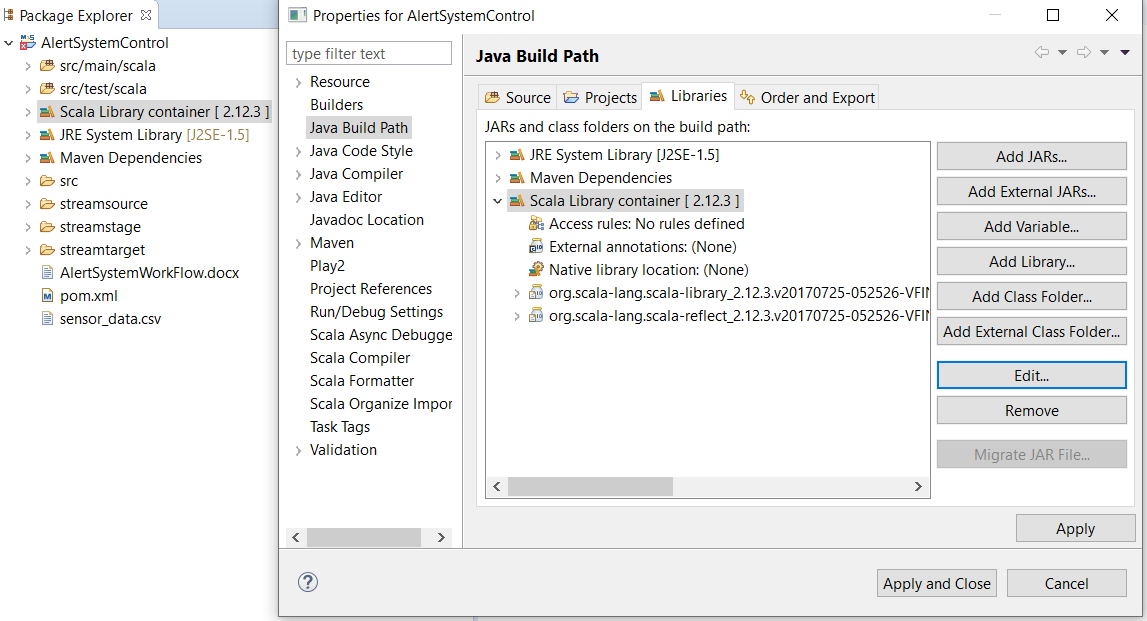
1. Then launch eclipse – scala and create a sample workspace as below screenshot. Here, the sample workspace is AlertSystem in D drive.

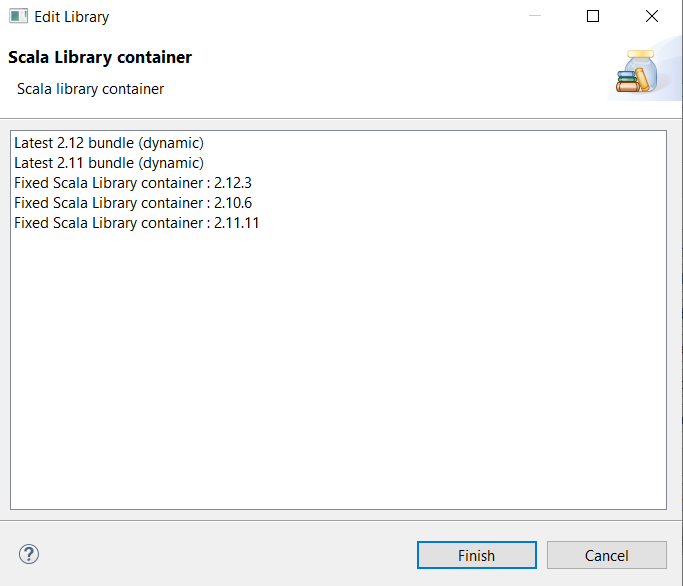


1. Then go to File 🡪 Open Projects from file system and browse the path where the project folder has been extracted and click create as shown below.



1. Since this is a maven project, all necessary jars will get downloaded w.r.t. the pom.xml file. Next go to the build path and change the scala library from 2.12.3 to 2.11.11 version as spark is supported on 2.11.11





1. Now execute the below programs in the following order:
2. readSensorData.scala
3. createStreamSource.scala
4. imputeData.scala

Below screenshot showing all the three programs running simultaneously to produce the control system with alert configuration.

