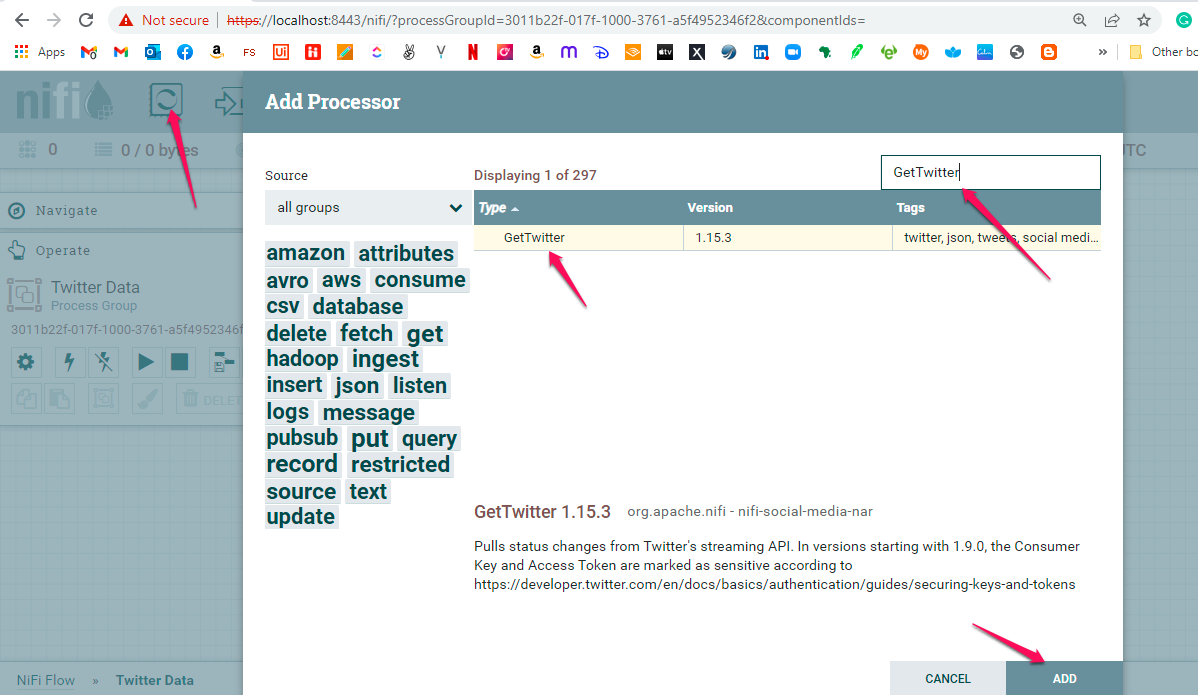
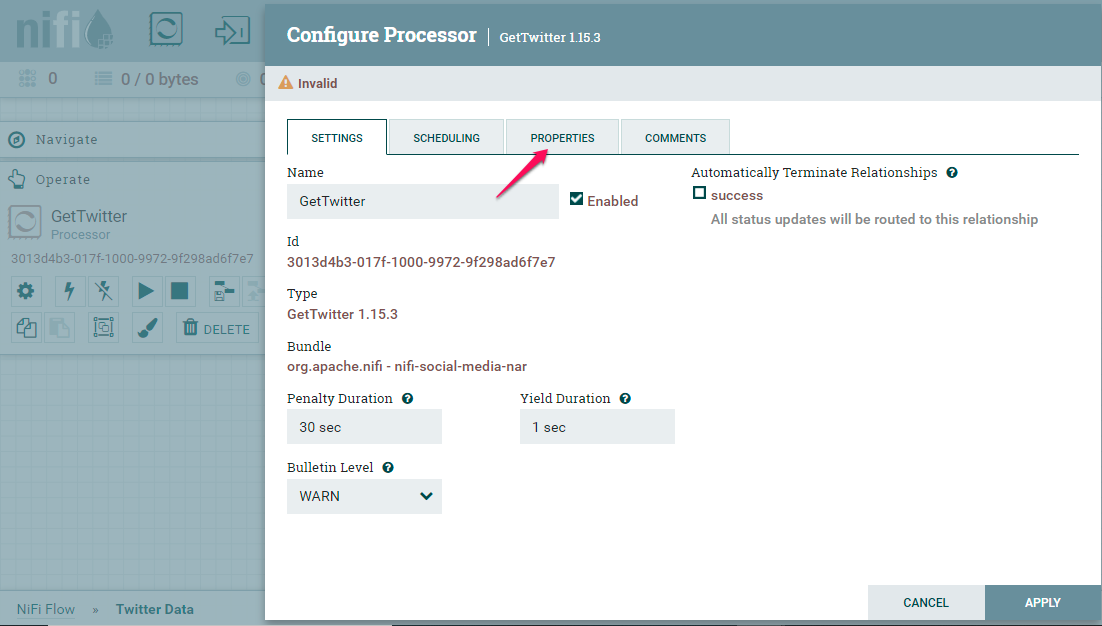
**BUAD 5722: Big Data - Lab 4 - Apache Nifi (Streaming Data)**

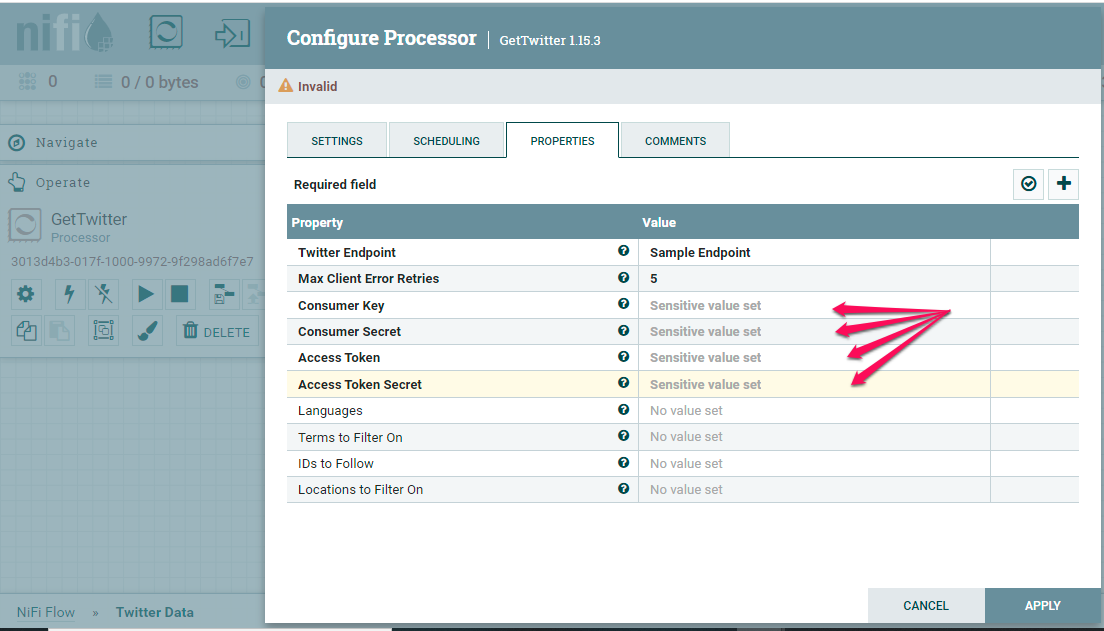
**Prof. Arturo Castellanos**

**Due: March 21st, 2022.**

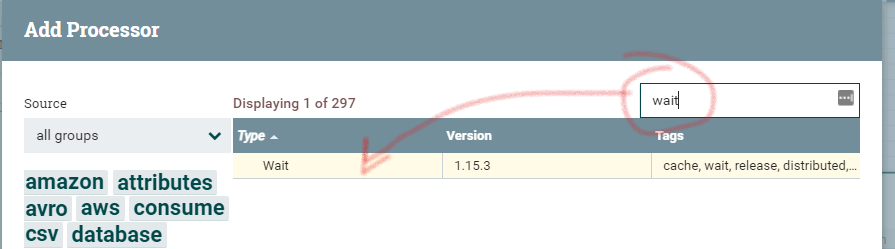
**PART II: NIFI TWITTER STREAMING LAB**

* After installing Nifi, ensure containing is running or start if stopped
* Open <https://localhost:8443/nifi/login>
* Login
  + Username is admin
  + Password is ctsBtRBKHRAx69EqUghvvgEvjnaLjFEB
* Add new Process Group called “Twitter\_Data”, drag & drop into the diagram space
* 
* Double click on Twitter Data
* Add new Processor called **GetTwitter** (drag & drop the icon into the diagram)
* To view documentation of GetTwitter, right click and select **view usage**
* 
* Goto <https://developer.twitter.com/>
* Click Dashboard
* Click Create Project
* Fill Form
  + Project Name **NifiStreamExamples**
  + How you intend to use Twitter developer platform select **Student**
  + Project description: **Nifi streaming example project for big data analytics course.**
  + App Environment - **Development**
  + Give a unique app name **- LeckanNifiStream**
  + Copy out API Key, API Key Secret and Bearer Token and save
  + Click on App Settings
    - **Generate API key and Secret**
    - **Generate Access Token and Secret**
* On Nifi, right click on GetTwitter Processor, Click configure
* Click Properties

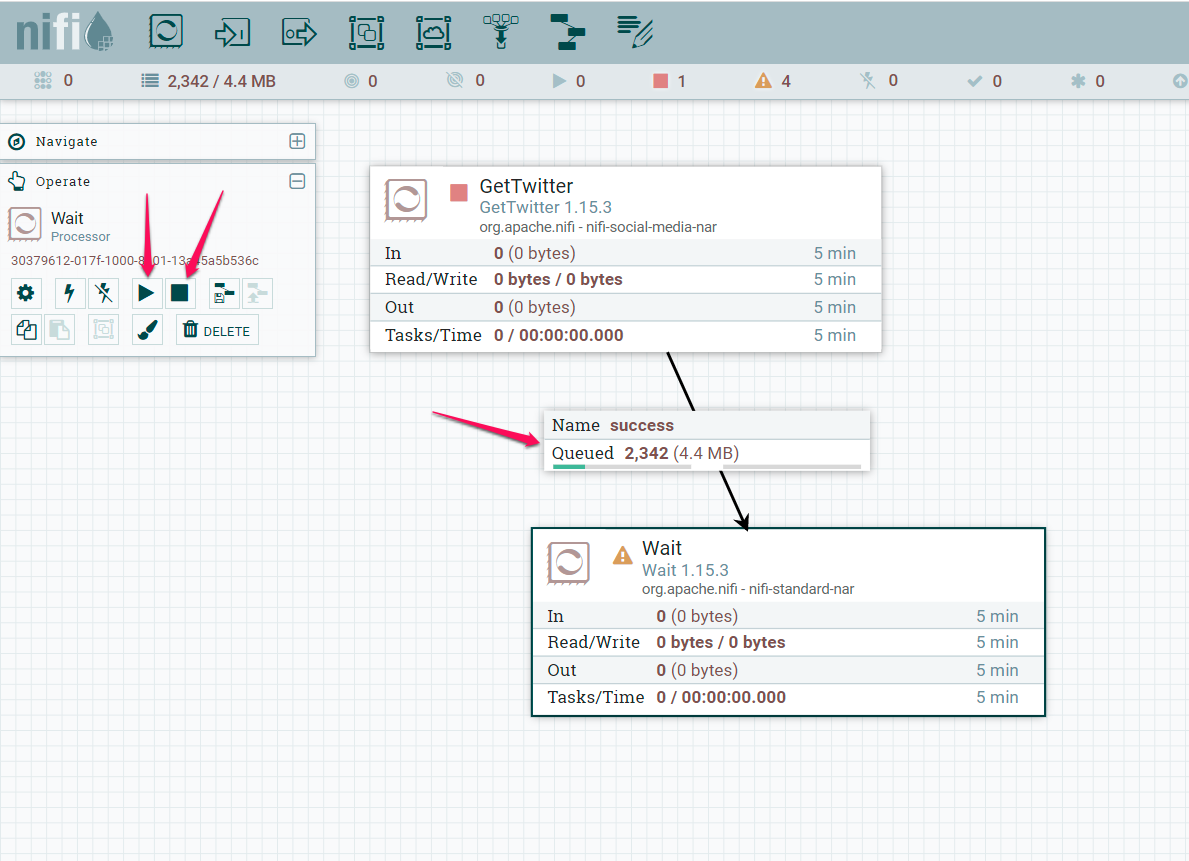
****

* Set the Consumer Key, Consumer Secret, Access Token and Access Token Secret****
* Click Apply
* Add new Processor, search for **wait** and Click Add

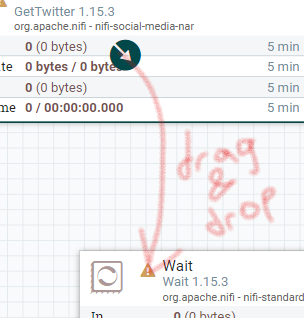


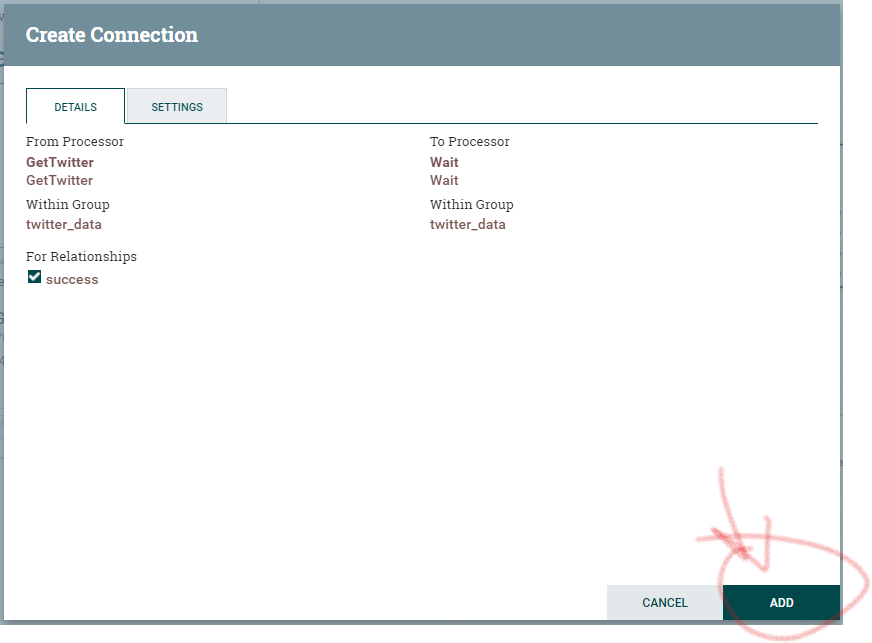


****

****

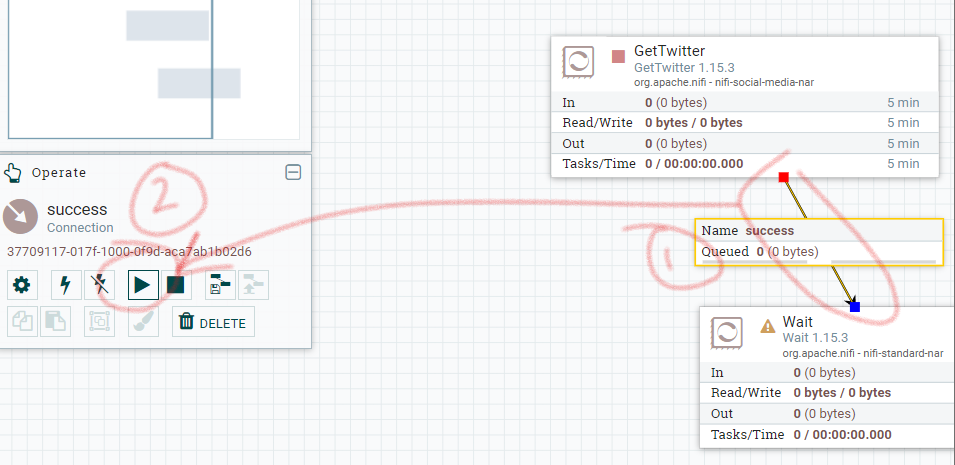
* To view documentation of wait, right click and select **view usage**
* Create a connection between GetTwitter and wait



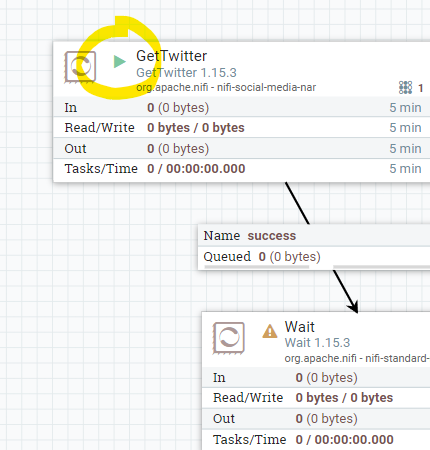


**Note: Moving forward, whenever we create connections between processors, we will select the success relationship. I won’t include this screenshot for future connections in the lab.**

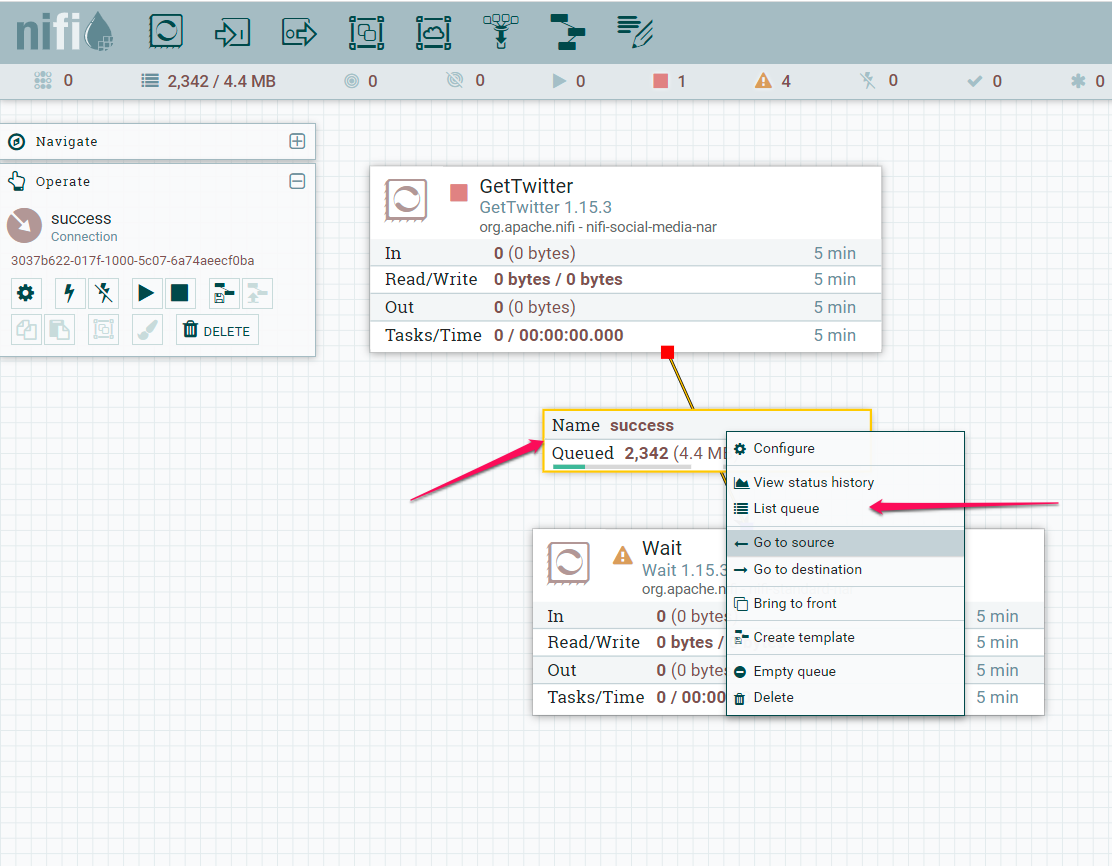
* You will now see a connection between the ‘GetTwitter’ and the ‘Wait’ operator [see (1) below]. Click the play button [see (2) below] and click the stop button after 30secs to review data gotten



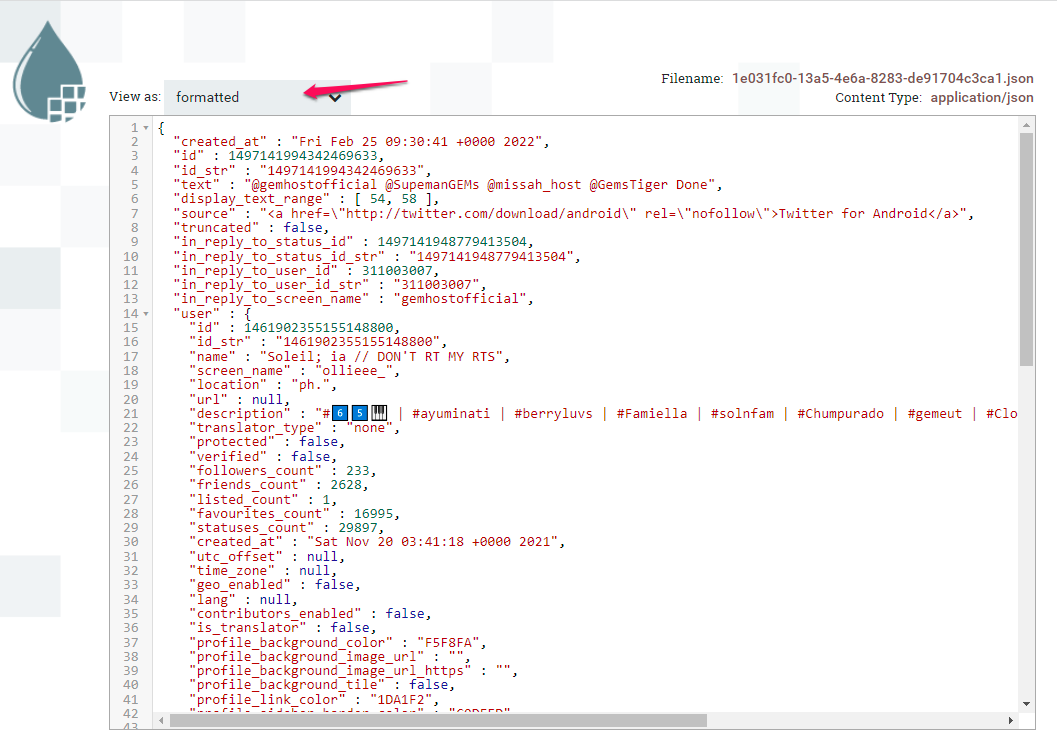
While is ‘playing’ you will see the green arrow (see below). After you stop, it will go back to a red square.



* Right click on grid click refresh
* Right click on the success connector and click list queue
* Select one record and click in the eye (to view content)
* Click in drop-down menu view as: **Formatted**
* Copy out data as sample if it starts with **{“created\_at”**, view another record otherwise

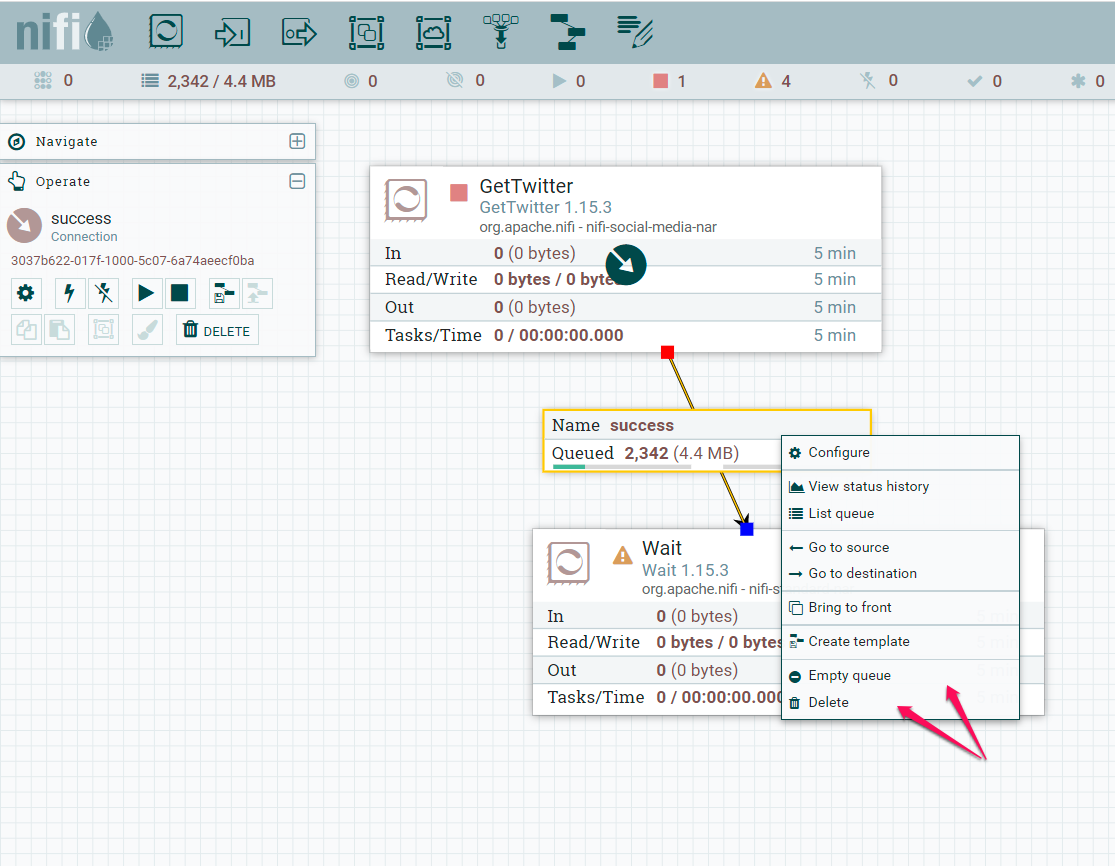
****

****

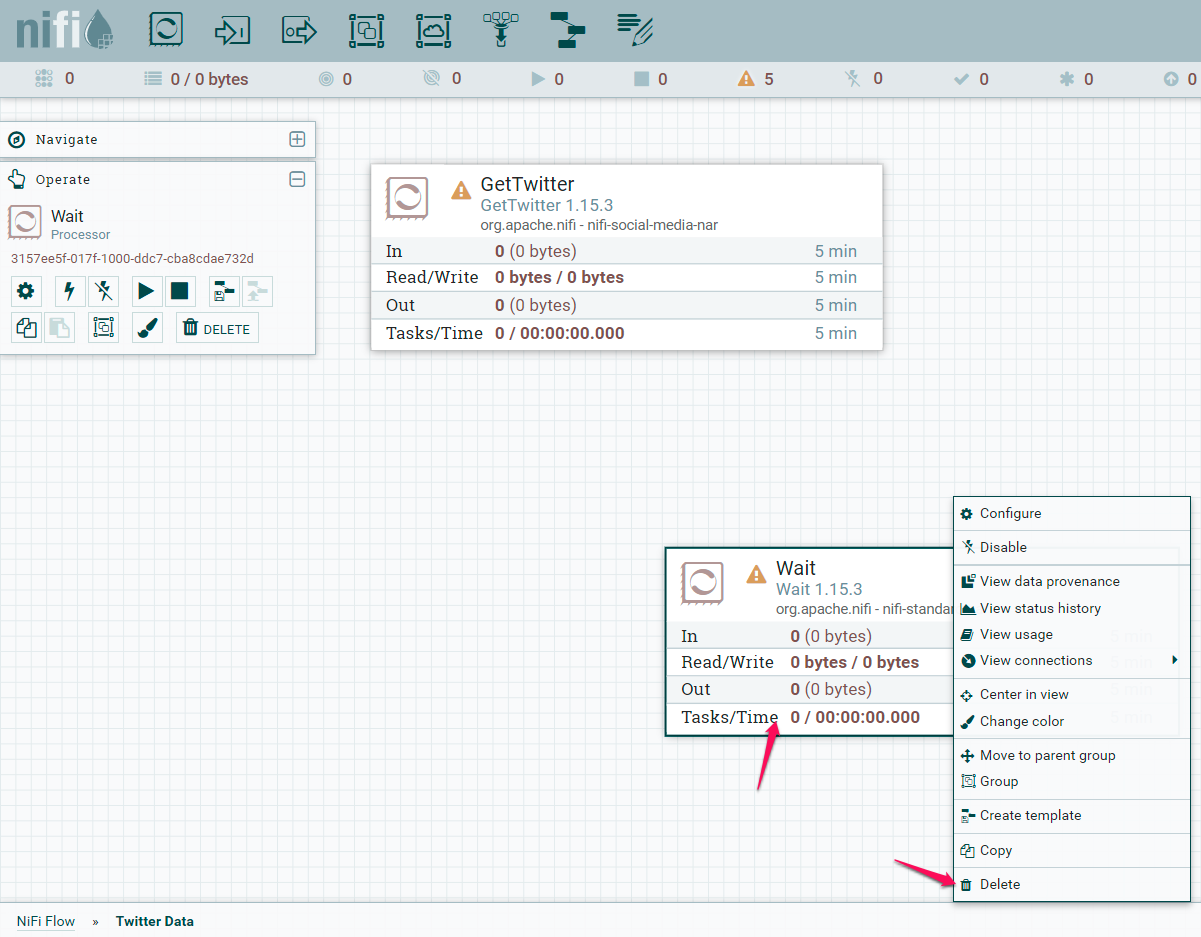
****

**To Trim the response to only show relevant data**

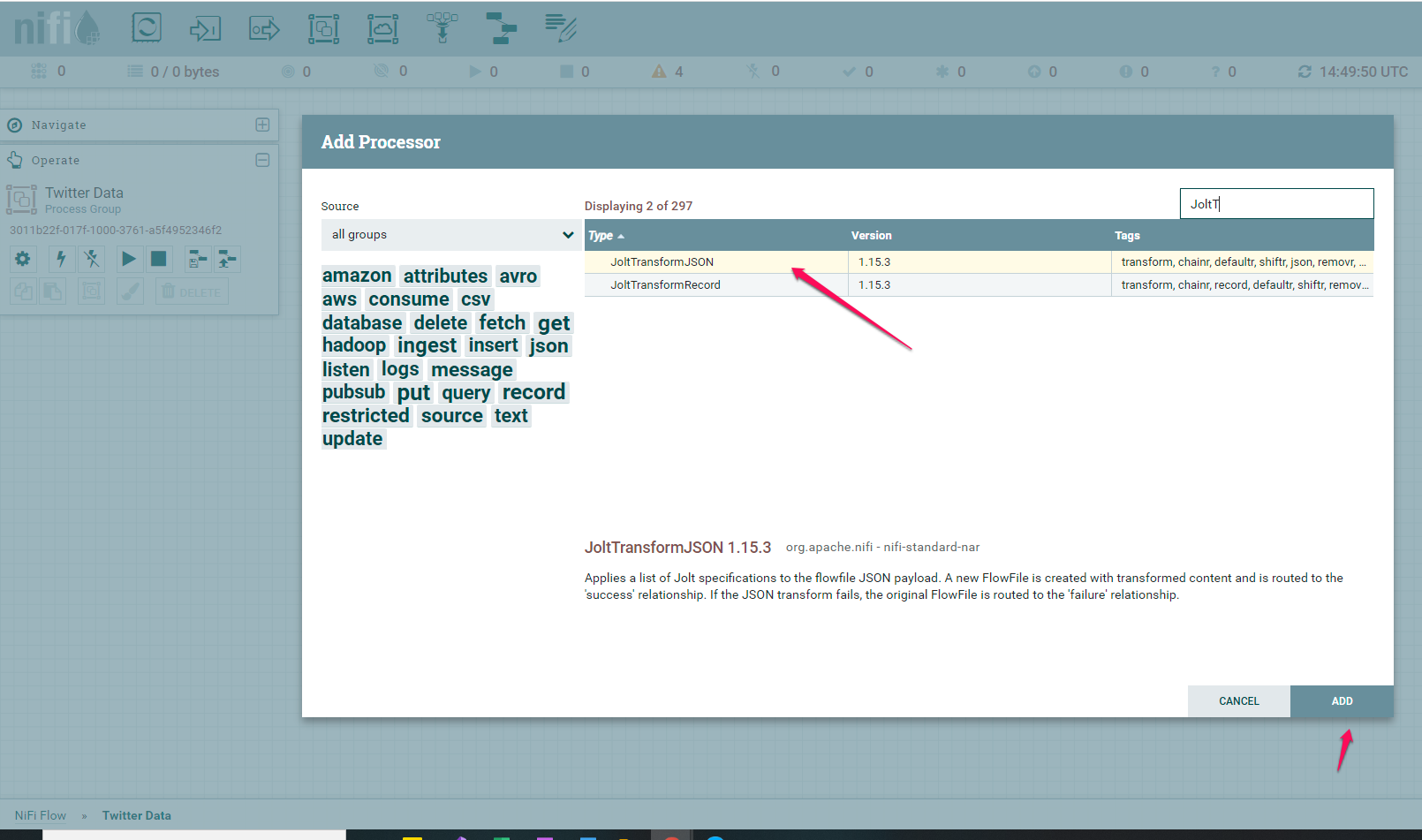
* Copy a sample response from a successful run of GetTwitter
  + Ensure the sample record gotten is one that starts with {“created\_at”
* Right click on the connection between GetTwitter and wait and select emptyqueue



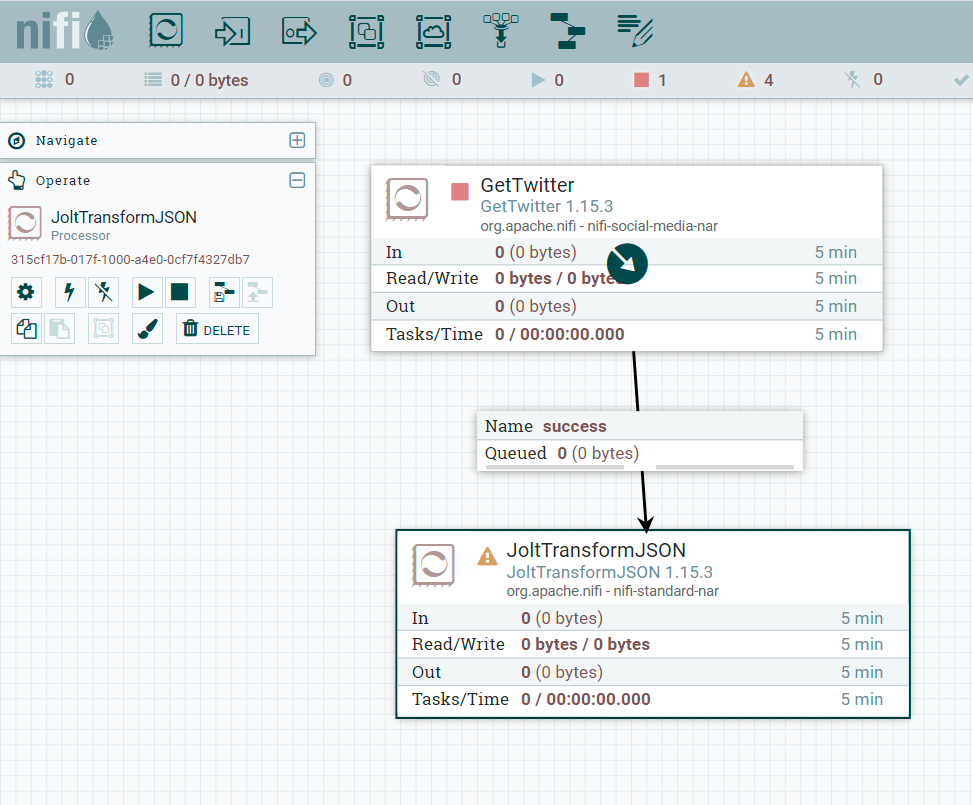
* Right click on the connection and select delete
* Right click on wait and select delete



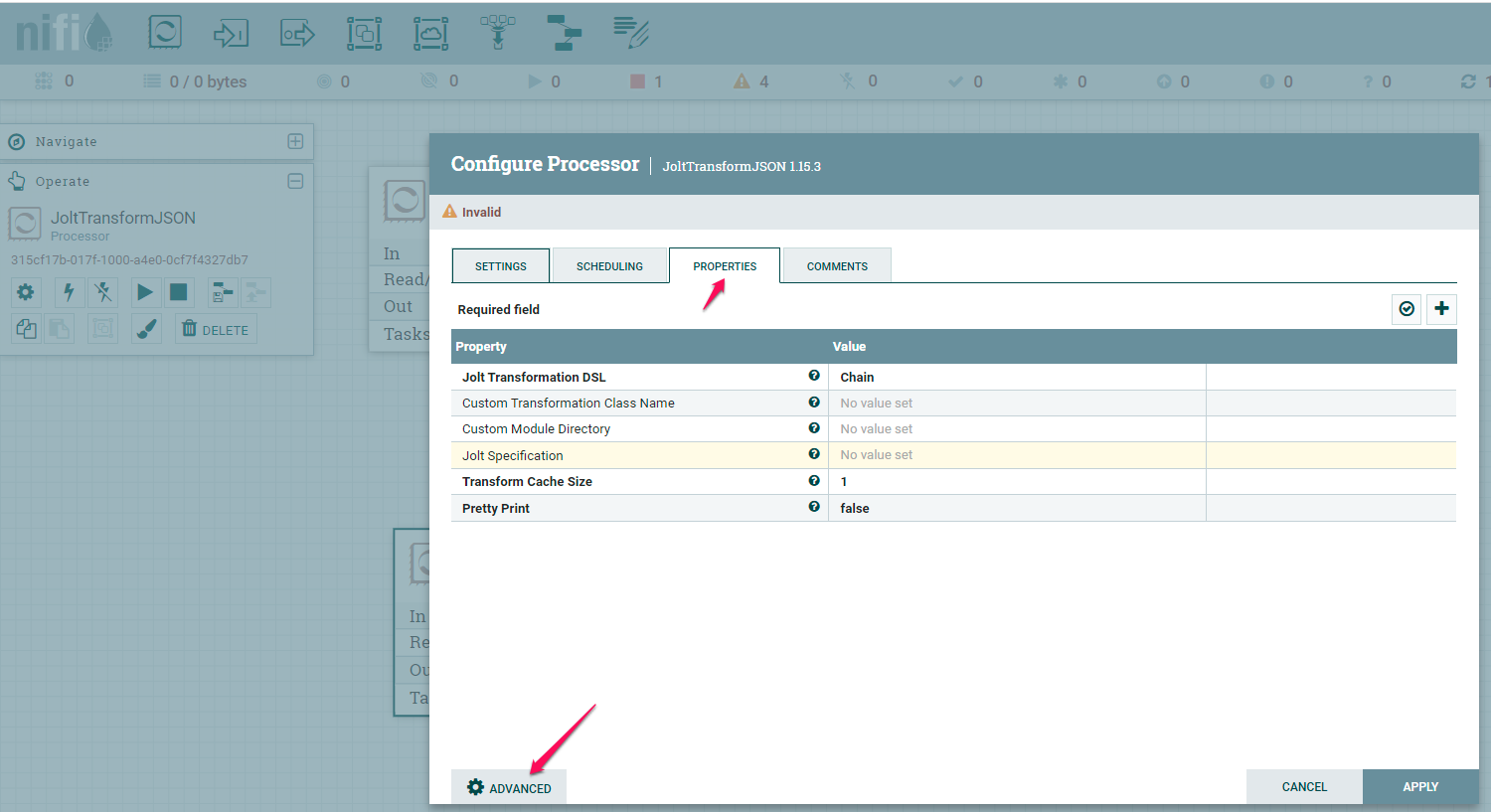
* Add new processor, search and select **JoltTransformJSON**



* To view documentation of **JoltTransformJSON**, right click and select **view usage**
* Create a connection between **GetTwitter** and **JoltTransformJSON,** click **add**



* Right click on **JoltTransformJSon** and select **configure** click on **properties**
* Click on **advanced**



* **Under Json Input**, paste the sample request from the earlier queue from view GetTwitter
* Under Jolt Specification paste json below

**[**

**{**

**"operation":"shift",**

**"spec":{**

**"created\_at":"created\_at",**

**"id\_str":"id\_str",**

**"text":"text",**

**"source":"source",**

**"geo":"geolocation",**

**"coordinates":"coordinates",**

**"place":"place",**

**"user" : {**

**"id":"id",**

**"name":"name",**

**"screen\_name":"screen\_name",**

**"location":"location",**

**"followers\_count":"followers\_count",**

**"friends\_count":"friends\_count",**

**"lang":"language"**

**}**

**}**

**}**

**Where did we get the JOLT Spec from?**

You can use jolt-demo.appspot.com as a way to validate your Jolt Spec. Note that you need to use the format:

[

{

“operatiion”:”shift”,

“spec”:{

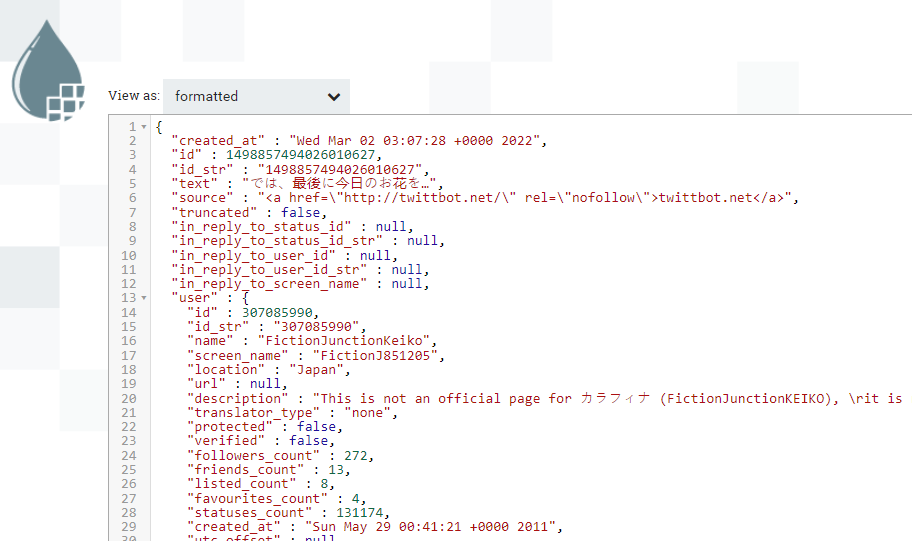
###ATTRIBUTES HERE###

}

}

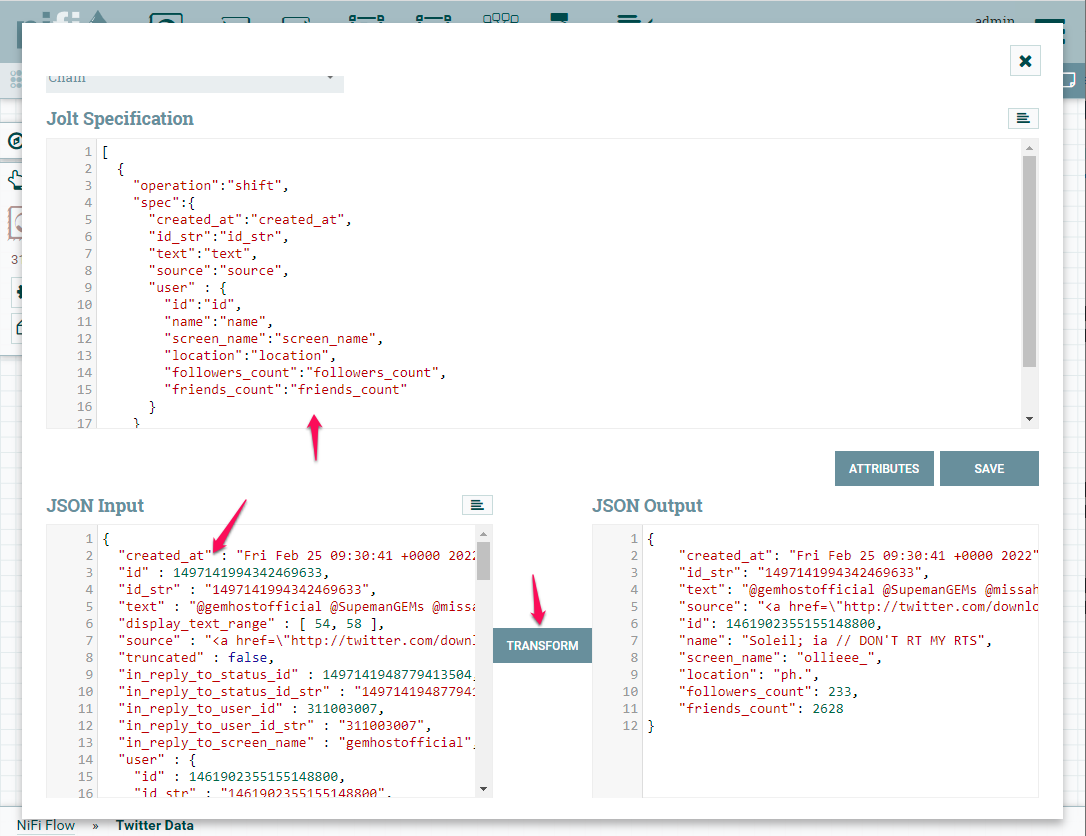
]

You would need a sample tweet coming from the “Get\_Twitter” processor to test the Jolt Spec agains (first column in the image below):

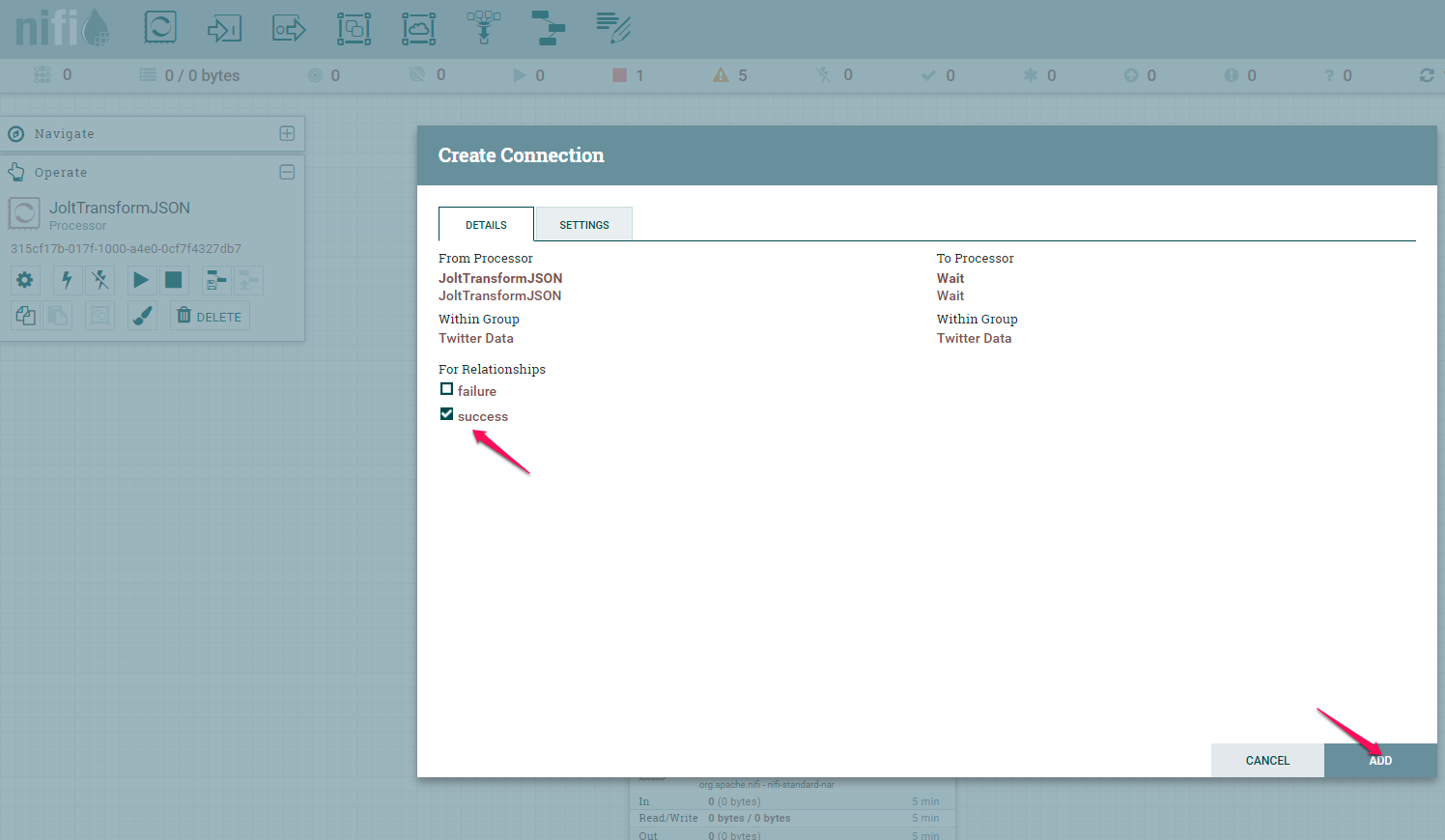


****

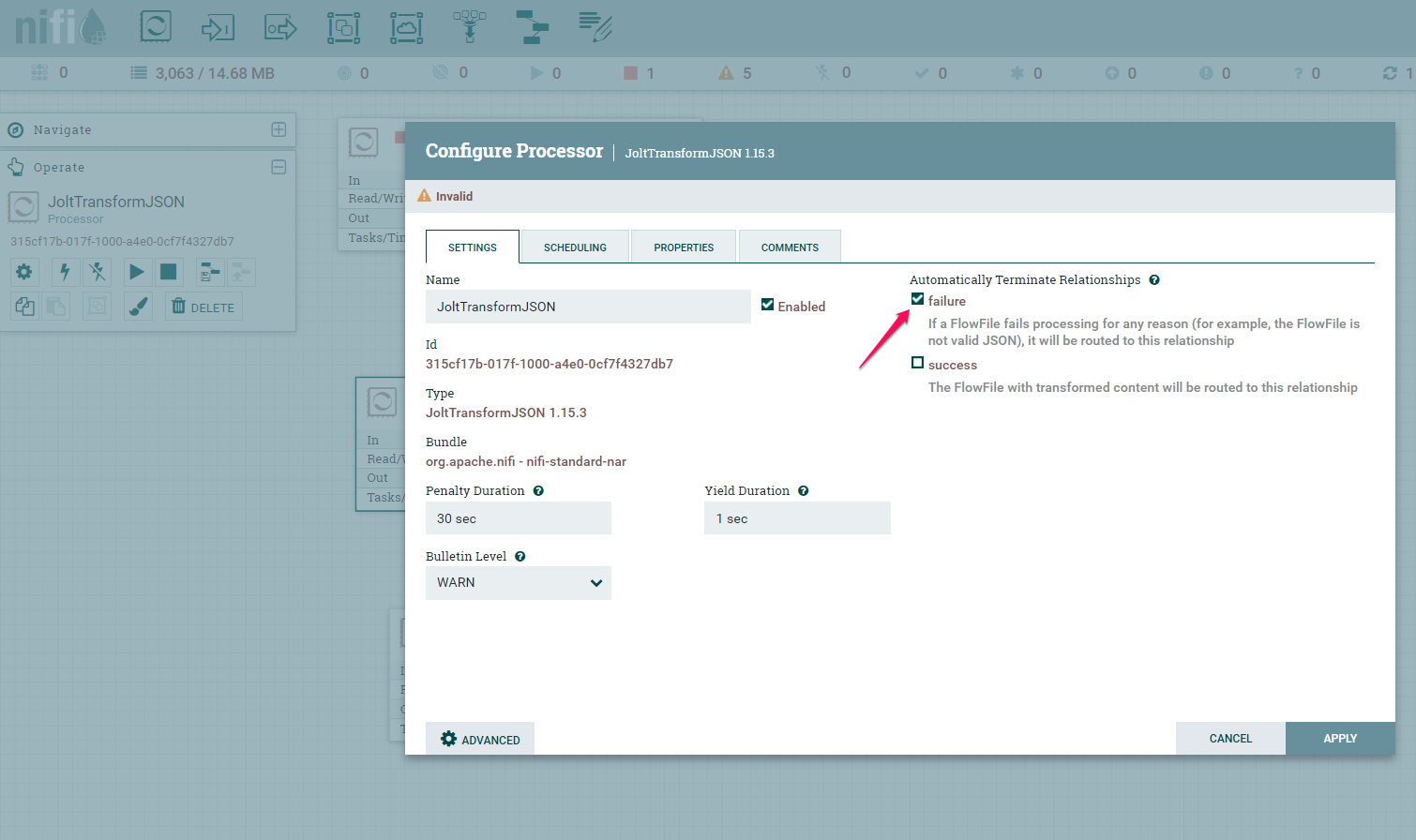
* **Click Transform** and confirm Json Output generated result



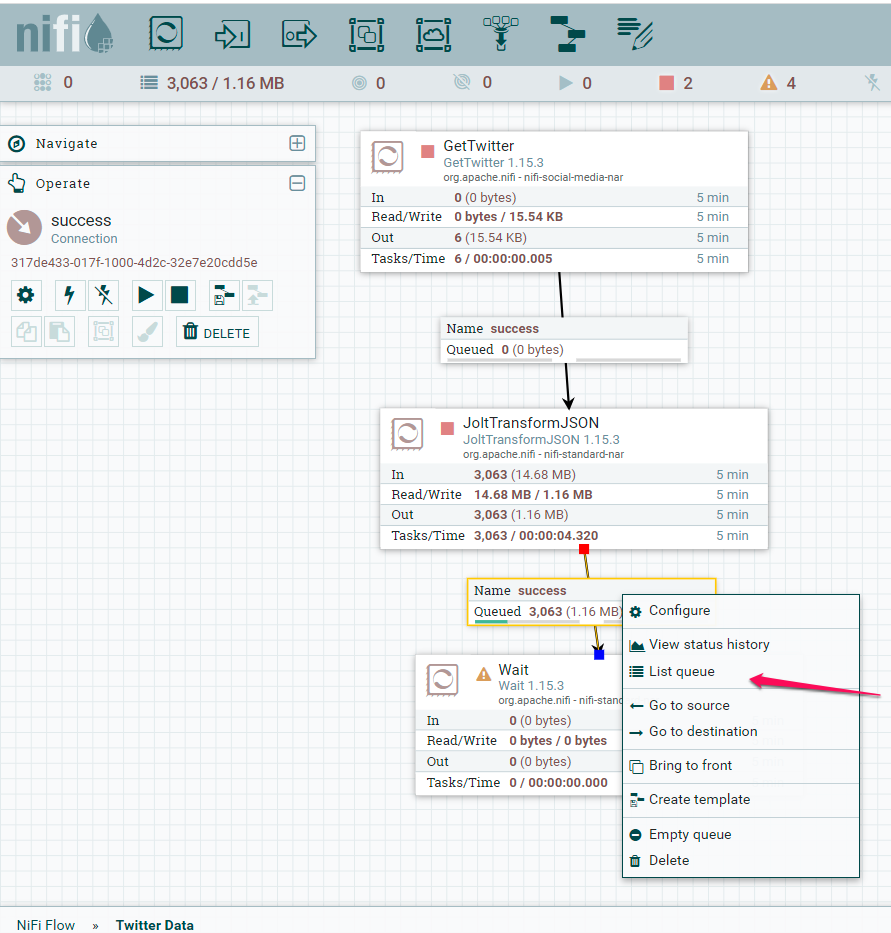
* **Click save**
* **Close Jolt advanced menu window (hit X)**
* Create a new processor - **wait**
* Create connection from **JoltTransformJSON to wait**
* In the Connection details page **select success checkbox**

****

* Right click the JoltTransformJson, click configure and check failure under settings

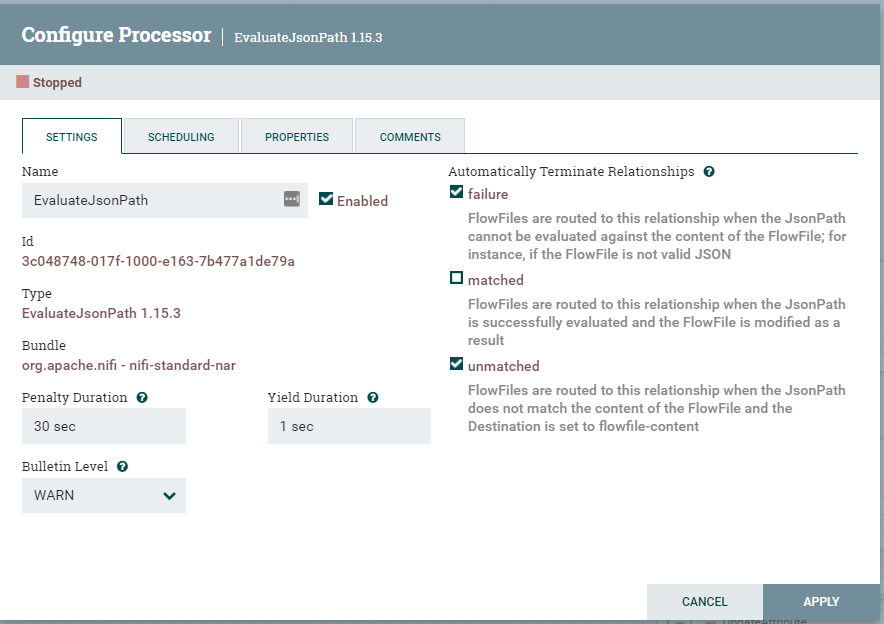
****

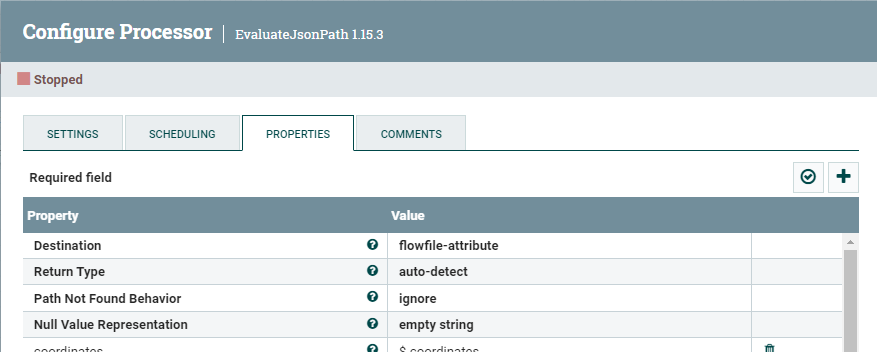
* Start the GetTwitter and JoltTransformJson for leave running for about 20secs
* Stop the GetTwitter and JoltTransformJson processors
* Right click and click ‘Refresh’
* Right click the connection between JoltTransformJson and Wait, click **list queue**



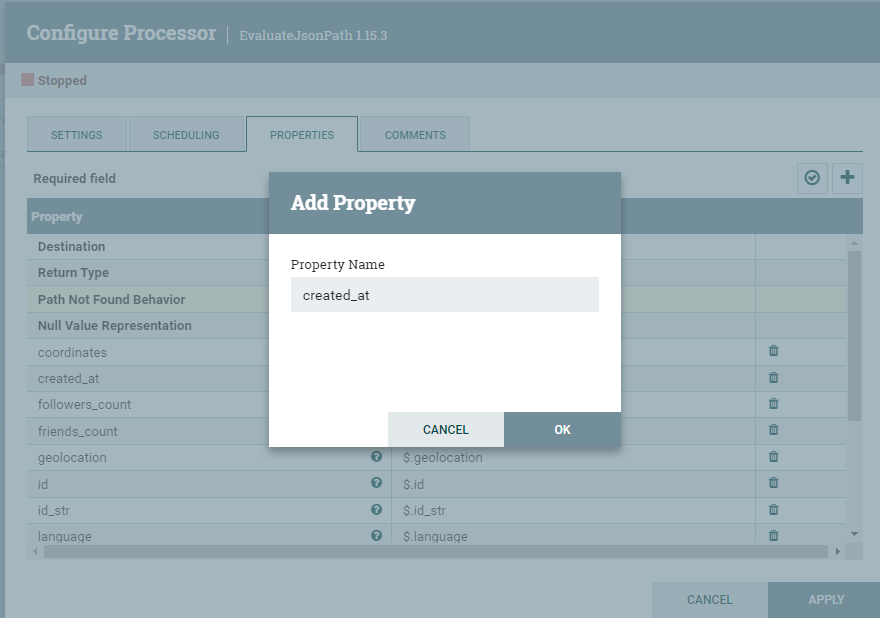


We will now store the JSON file as a CSV file. We will first add a EvaluateJsonPath processor:

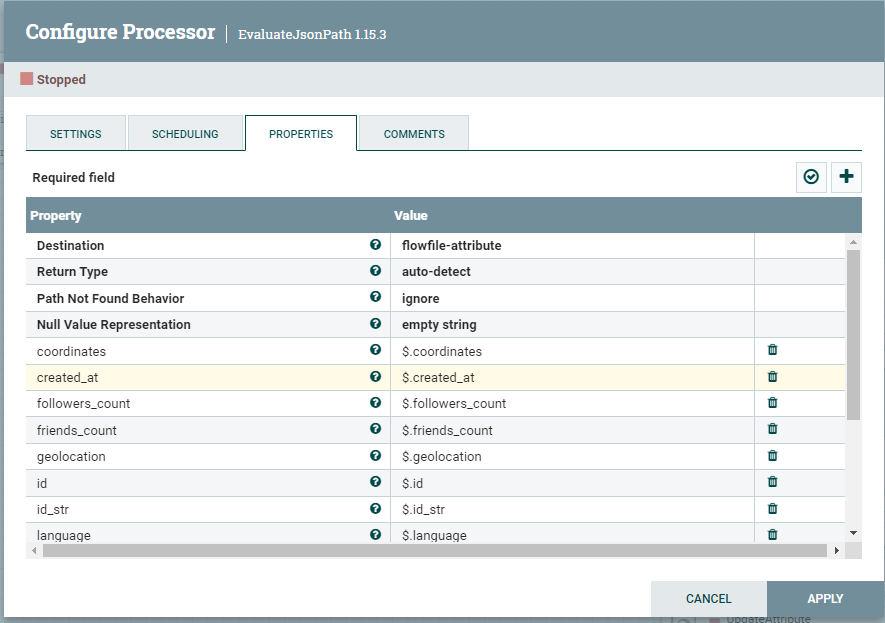




You can add new properties by clicking on the ‘+’ sign. We will add the attributes one by one:



Make sure you add the other attributes that you are storing in the csv. In the value you can use $.variable\_name to access that data from the stream. Do the same for the rest of the attributes you selected in the Jolt transform: **coordinates, followers\_count, friends\_count, geolocation, id, id\_str, language, location, name, place, screen\_name, source, text** [these are the attributes that we decided to retrieve from the Jolt spec]

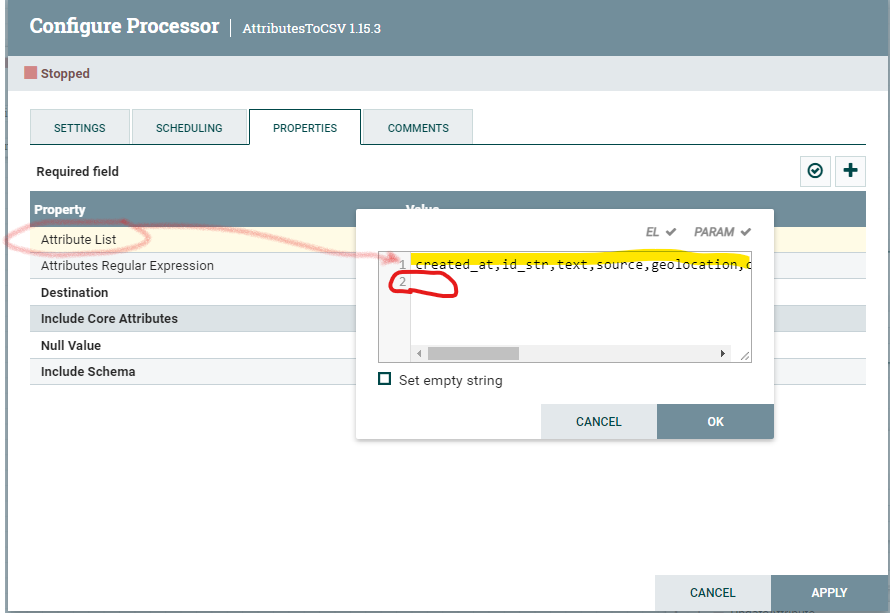


These is the rest of the attributes:

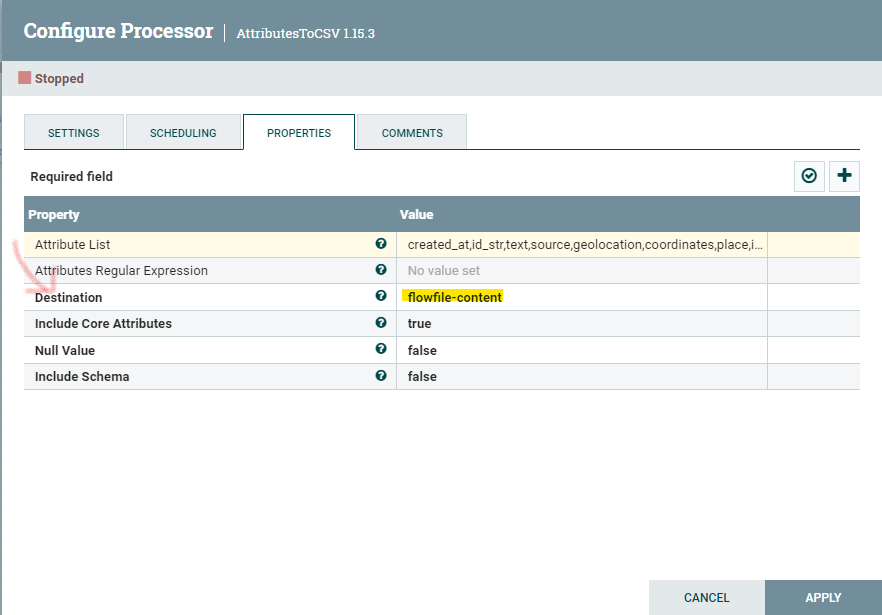


Make sure in the properties, you add the list of attributes you want to save (make sure you leave a carriage return (new line) in the second row (see red circle below):

**The attributes are:** created\_at,id\_str,text,source,geolocation,coordinates,place,id,name,screen\_name,location,followers\_count,friends\_count,language



In the **Destination** property select: flowfile-content



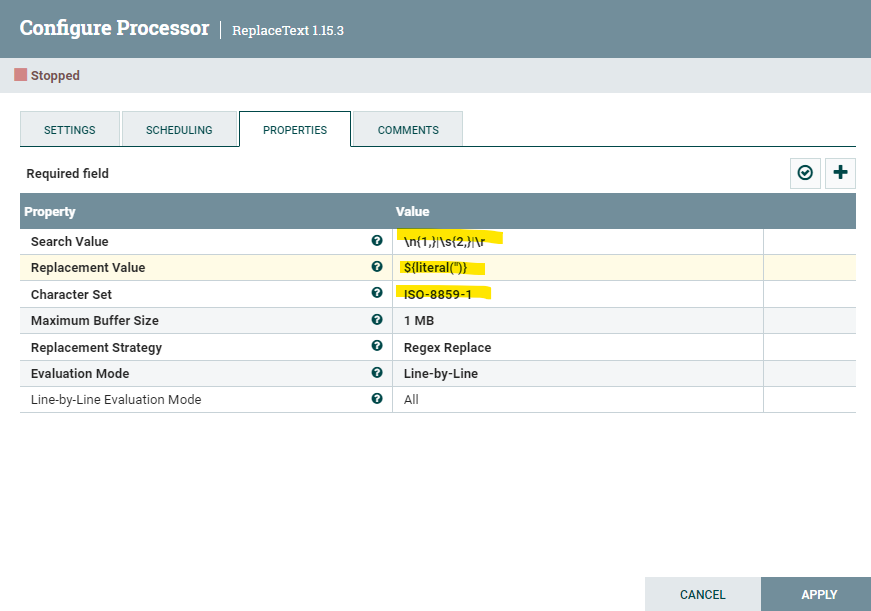
We are going to use a regular expression to replace carriage returns or line breaks with nothing:

Change the following properties:

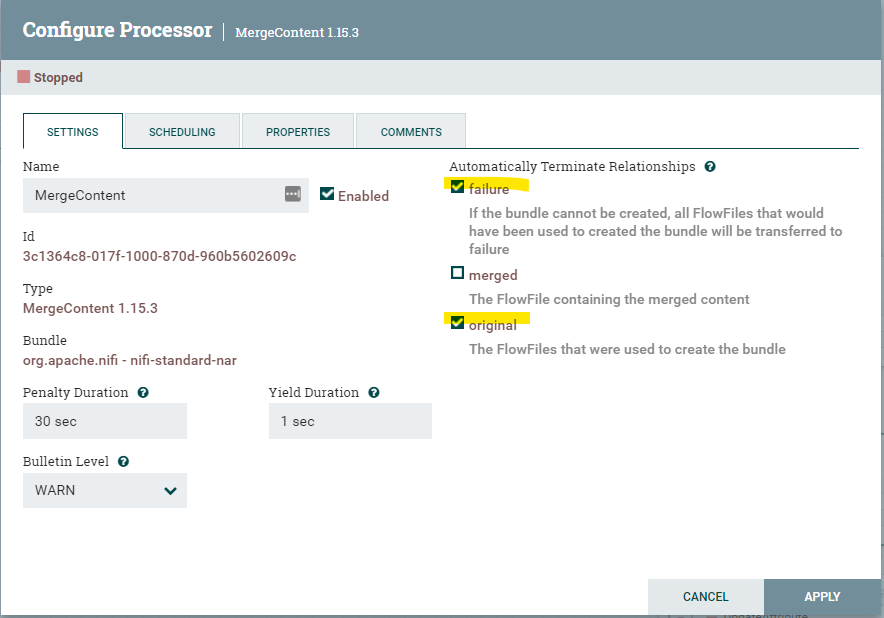
search value: \n{1,}|\s{2,}|\r

replacement value: ${literal('')}

Character set: ISO-8859-1



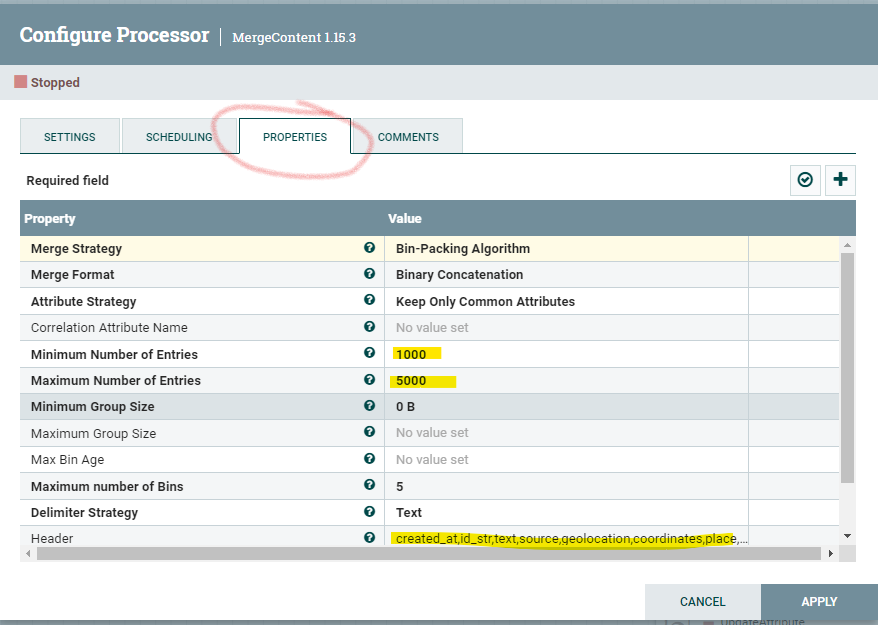
The **AttribtesToCSV** processor would create a csv file for each row that goes through the flowfile so now we will add a **MergeContent** processor into the canvas and we will connect it to the replace text processor we created before.



We will now change a few properties in the MergeContent.

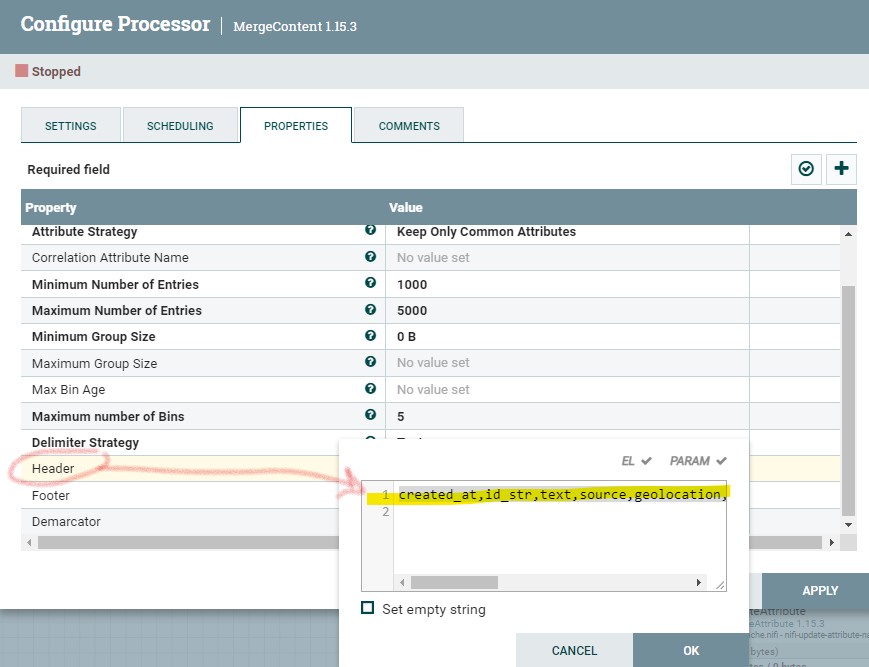
Minimum number of entries: 1000

Maximum number of entries: 5000

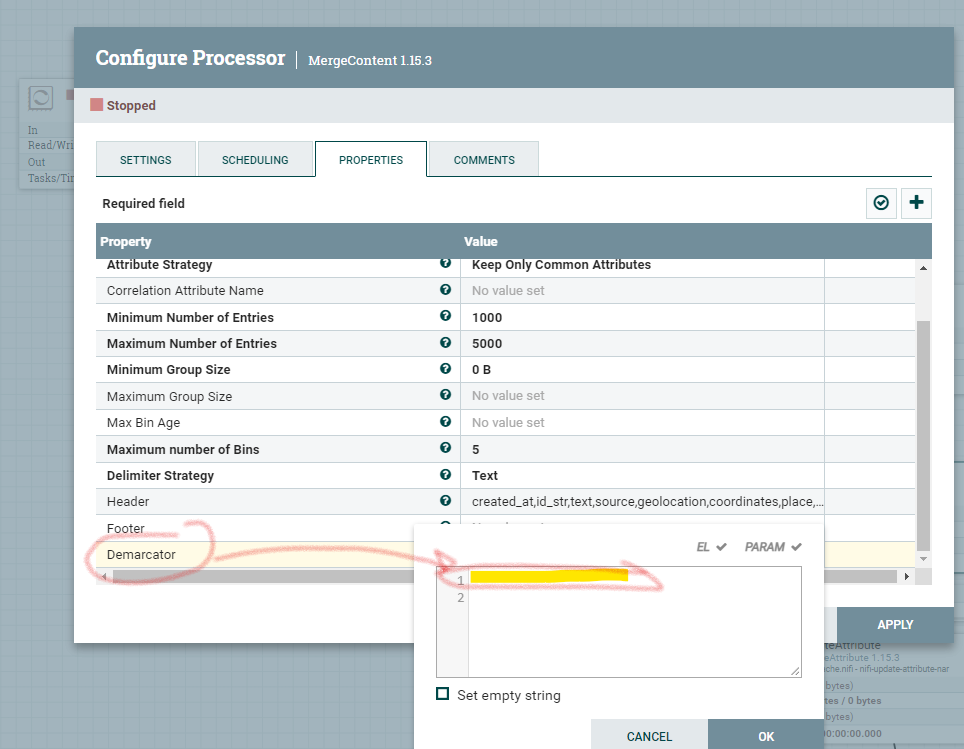


The minimum number of entries and maximum number of entries control how big we want the csv files to be.

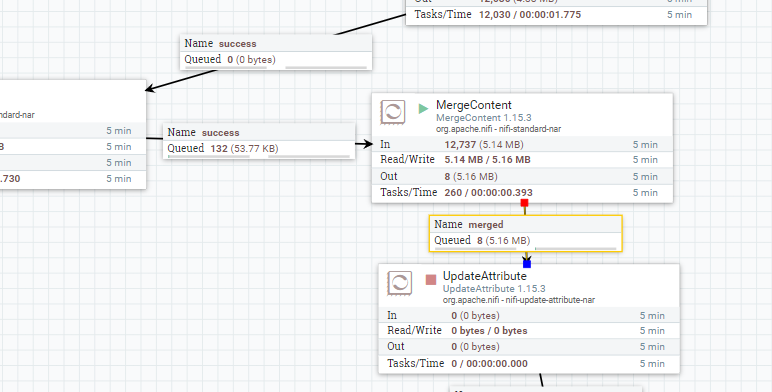
Header: all the attribute names separated by a comma, this will print the header of the csv file



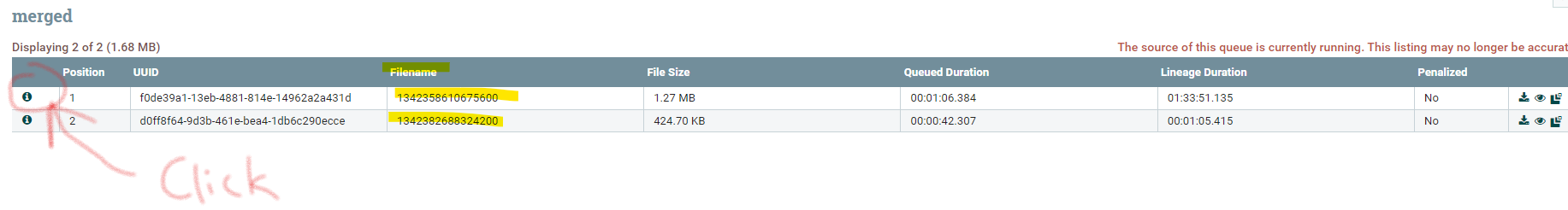
**Demarcator**: Create a new line (line break) by hitting enter in your keyboard

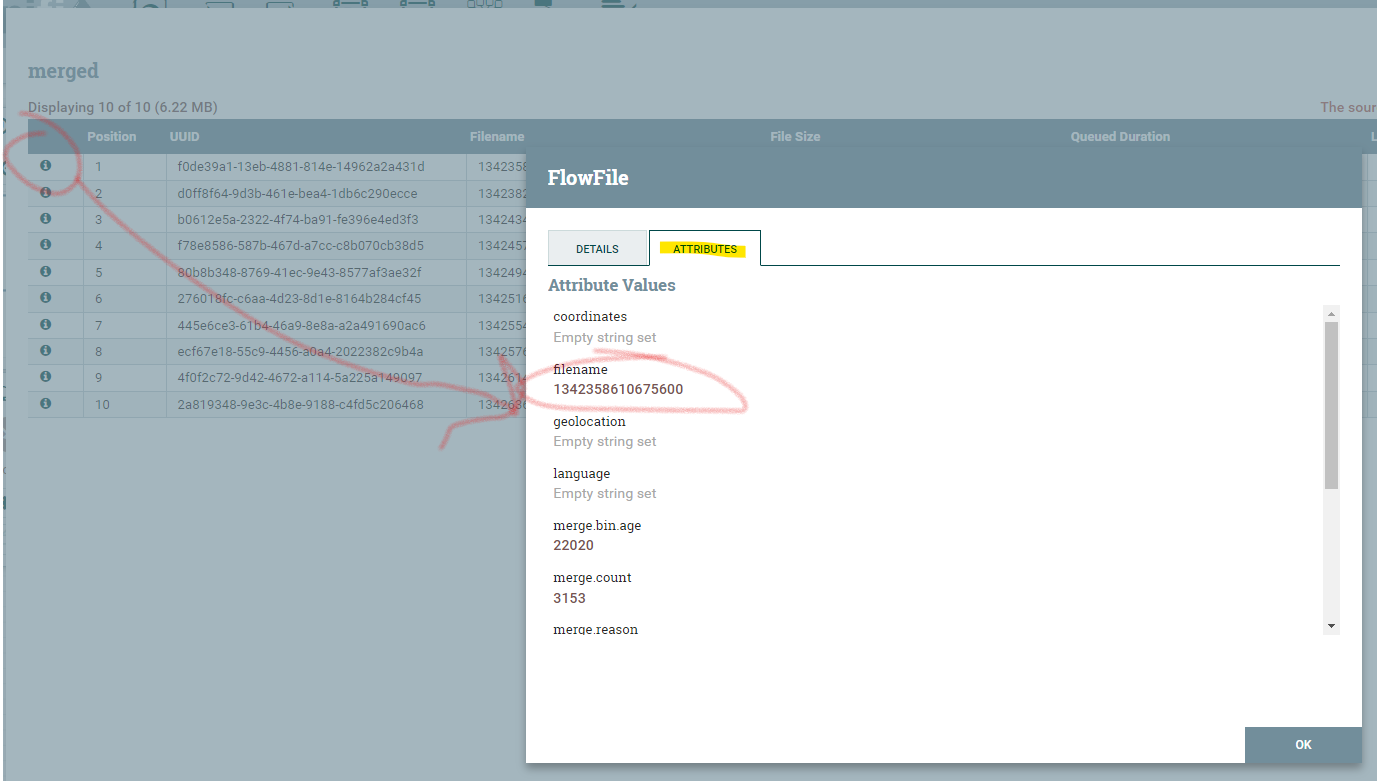


We can run the flow and we will see that the Merged queue will only write files whenever the Min/Max number of entries conditions from the previous step are fulfilled (box in yellow):

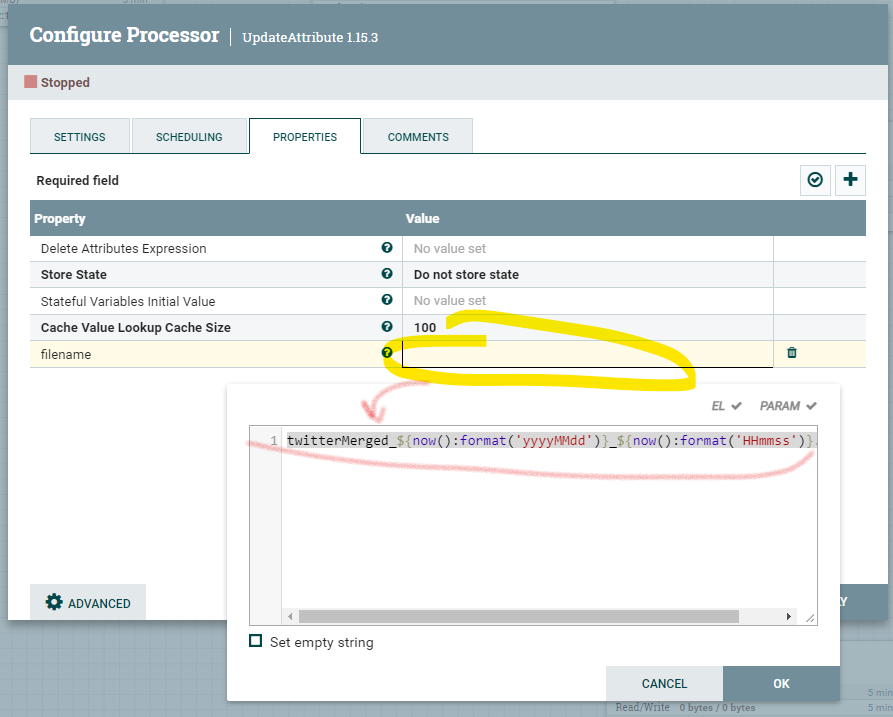


We can see the by right clicking on the merged queue and click on “list queue”:

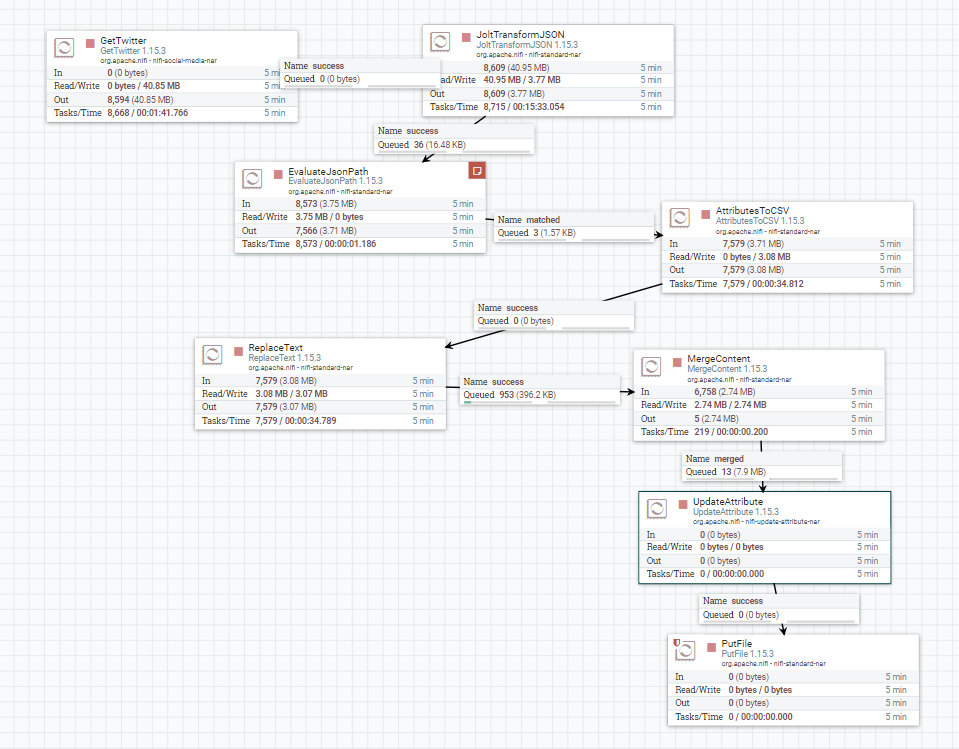




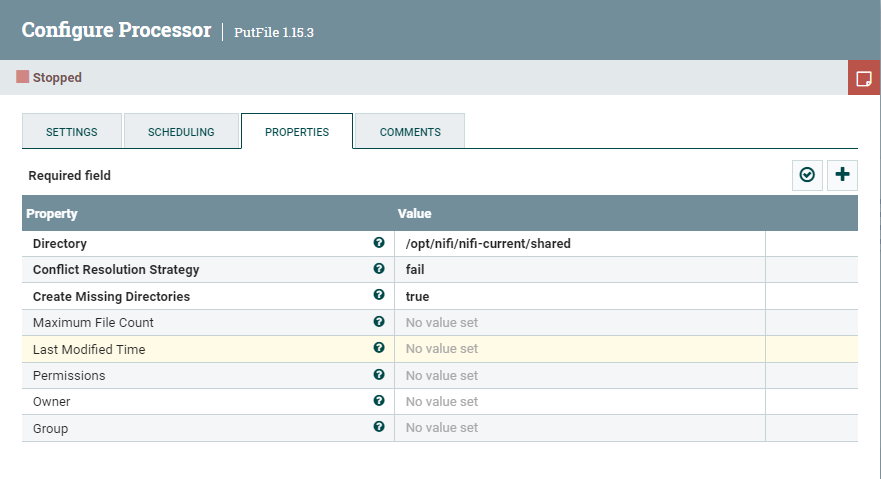
It is creating the csv files with random names (not really random! cron time) but we want to do something a little more meaningful using the date and time the file was created:



Final diagram:



Last, we select the directory where we want to save the CSV files:



**Summary of what you achieved in the lab:**

We are streaming data from twitter in real-time, we receive JSON output from Twitter, we use JoltTransformJSON to only get a few attributes from the JSON output, we use the EvaluateJSONPath to evaluate multiple expressions against the content of the FlowFile. The results are assigned to attributes by adding properties that map to the attribute name into which the result will be placed. The AttributesToCSV denormalizes the data from JSON format to a comma delimited output. The ReplaceText uses Regex to remove new lines (carriage returns) to avoid having formatting issues in the csv. The MergeContent operator merges all the csv outputs from the previous operator into a CSV that merges multiple outputs into one csv. We control how big we want this file to be using the Min/Max number of entries. Make sure you add a header to the csv file and that you add a carriage return in the “Demarcator” property. Otherwise all records will be put next to each other in the document. The update attribute allows us to change the “filename” attribute from the MergeContent Processor with something more meaningful (date-time when the file was created). Last, we select the directory where we want the file to be stored.