Cloud Computing 2022/23 spring

Main
Lectures
Practicals
Plagiarism Policy
Results
Submit Homework

Practice 9 - Introduction data pipelines using Apache NiFi

In this Practice session you will work with Apache NiFi. You will learn how to set up NiFi in cloud, how to create NiFi pipelines and how to use it to manage data streams.

References

- Apache NiFi In Depth https://nifi.apache.org/docs/nifi-docs/html/nifi-in-depth.html#flowfile-repository
- Apache NiFi documentation http://nifi.apache.org/docs.html

Exercise 9.1. Installation of Apache NiFi!

In this task we will use OpenStack to run Apache NiFi inside an instance using Docker.

Create an OpenStack VM instance:

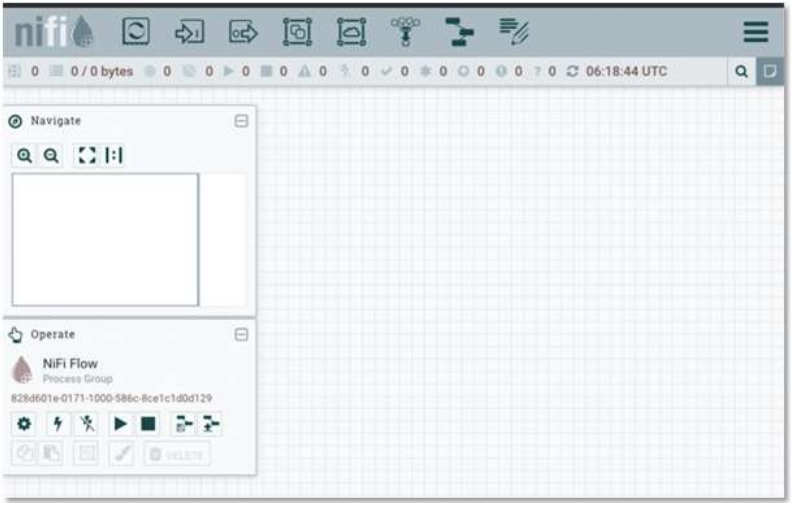
- **Source**: Instead of an *Image*, use **Volume Snapshot**, choose `Ubuntu22+Docker`
  - in this Ubuntu-based snapshot, the installation of Docker (as we did in Lab 2) has already been done for us.
  - Enable "*Delete Volume on Instance Delete*"
- **Flavour**: should be `m2.tiny`
- Select security group with name *Graphana InfluxDB*

Run Apache NiFi using Docker

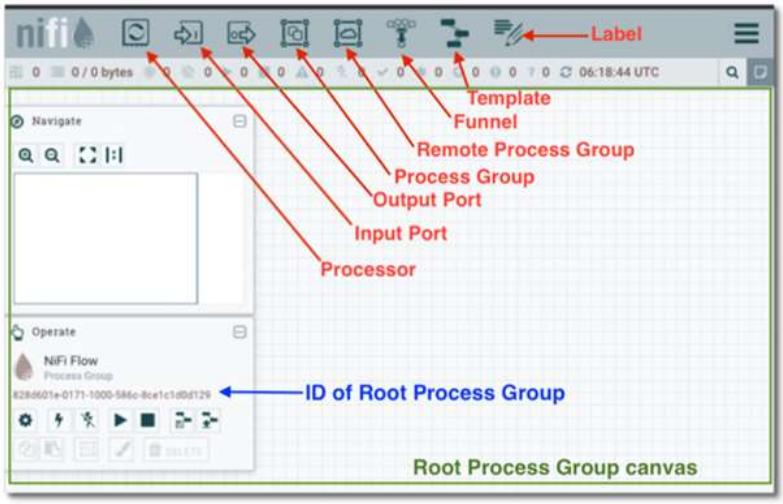
- Create a Docker container with the NiFi image from Docker Hub using the following command line command:

```
docker run --name nifi   -p 443:8443  \
 -d   --hostname 193.40.11.178.nip.io \
  -e SINGLE_USER_CREDENTIALS_USERNAME=lab09  \
 -e SINGLE_USER_CREDENTIALS_PASSWORD=tartunifi2023 \
  apache/nifi:latest
```

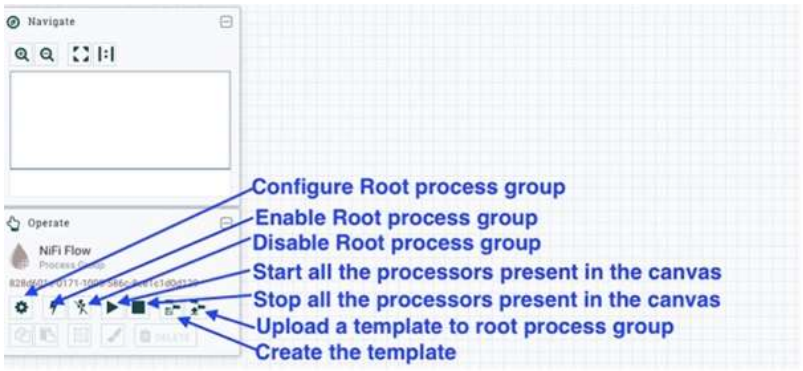
  - SINGLE_USER_CREDENTIALS_USERNAME=lab09 - defines the username (Change to your own username)
  - SINGLE_USER_CREDENTIALS_PASSWORD=tartunifi2023 - defines the password (Change to your own password with more than 12 characters)
  - --hostname 193.40.11.178.nip.io - (PS!! 193.40.11.178 should be replaced with your VM IP) defines that the computer/server corresponds to the dynamic host address 193.40.11.178.nip.io, where the first part (193.40.11.178) MUST be the computer/IP address (Otherwise a certificate error will occur)
- After that, NiFi can be reached at https://193.40.11.178.nip.io/nifi (PS!! 193.40.11.178 should be replaced with VM IP)
- It takes some time for nifi server to be up and running (For me 5 minutes). You can check for the nifi container logs if needed.
- The web interface looks something like this:



- Description of the main Web interface elements:



- Description of the main control elements:



- Now you are ready to start creating NiFi pipelines

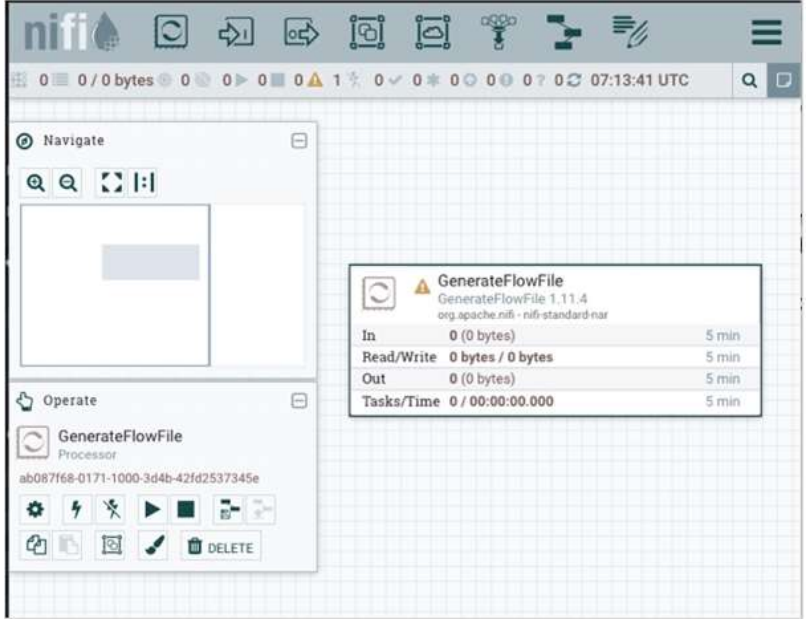
Exercise 9.2. Generate flow files and send to local directory

Let's create a simple pipeline which generates FlowFiles with random content and stores them as files into the filesystem. We will use 2 NiFi processors:

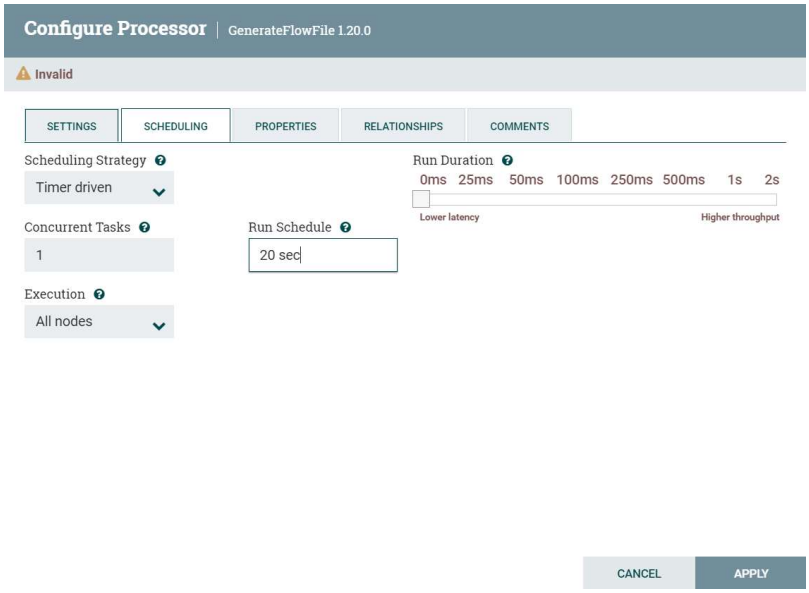
- ***GenerateFlowFile*** : This processor creates FlowFiles with random data or custom content. It is useful for load testing, configuration, and simulation.
- ***PutFile*** : This Processor can used to store incoming FlowFiles into user configured folder in the local filesystem.

Lets create both of these processors and connect them:

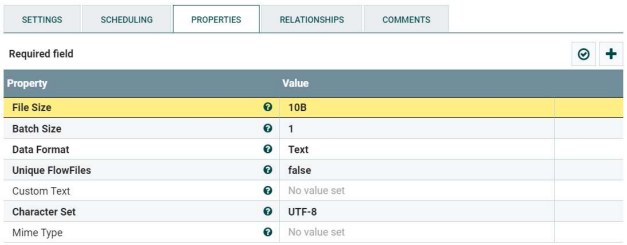
- Add ***GenerateFlowFile*** processor to the NiFi canvas
  - Drag and drop the NiFi processor icon (top left on the NiFi web interface) to the canvas. Nifi will display a table of available Processors.
  - Type ***GenerateFlowFile*** in the search box.
  - Double click on the processor to add to the canvas.
  - Now ***GenerateFlowFile*** processor will be added to the main canvas.



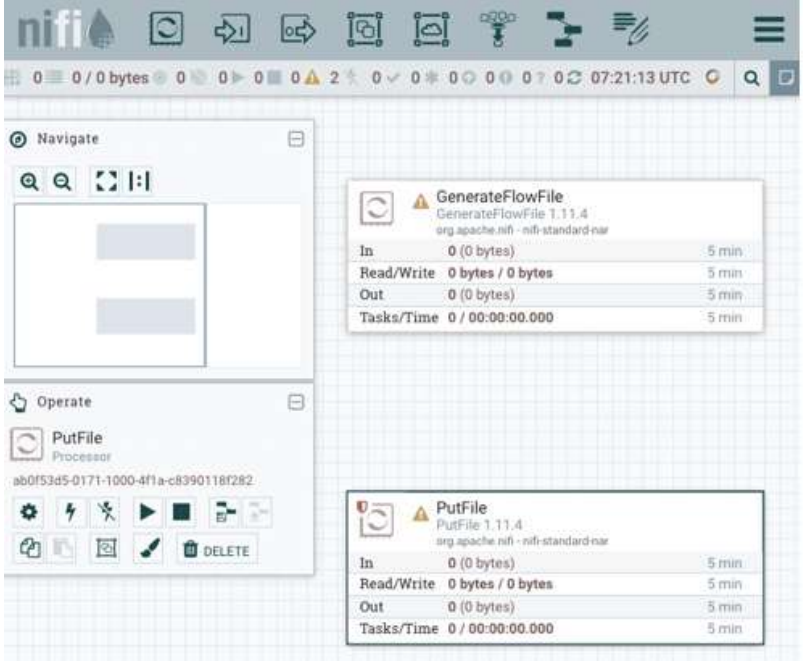
- Double click on the newly added ***GenerateFlowFile*** Processor to get the configure processor window.
- Configure the Scheduling tab
  - Schedule this processor to run in every `20 sec`. This allow us to limit the number of FlowFiles that are generated.
  - NB! Make sure that the **Run schedule** is not set to 0 sec, as this will make NiFi to schedule the Processor without limit and huge number of files will be generated at once.
  - ***Scheduling Tab*** should now look like this:



- Configure the Properties tab with the following info:
  - File size: `10B`
  - ***Properties Tab*** should look like this:

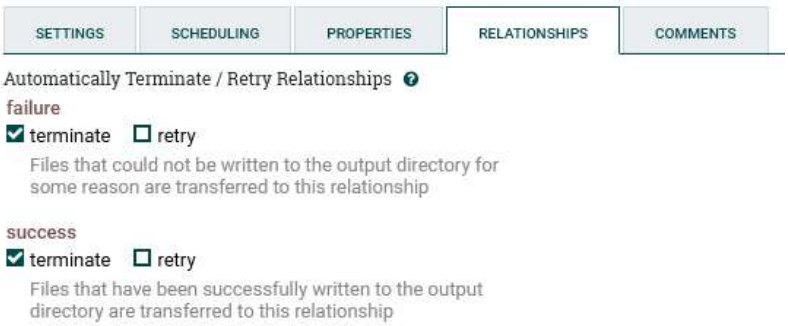


Lets now add the second NiFi processor: ***PutFile***

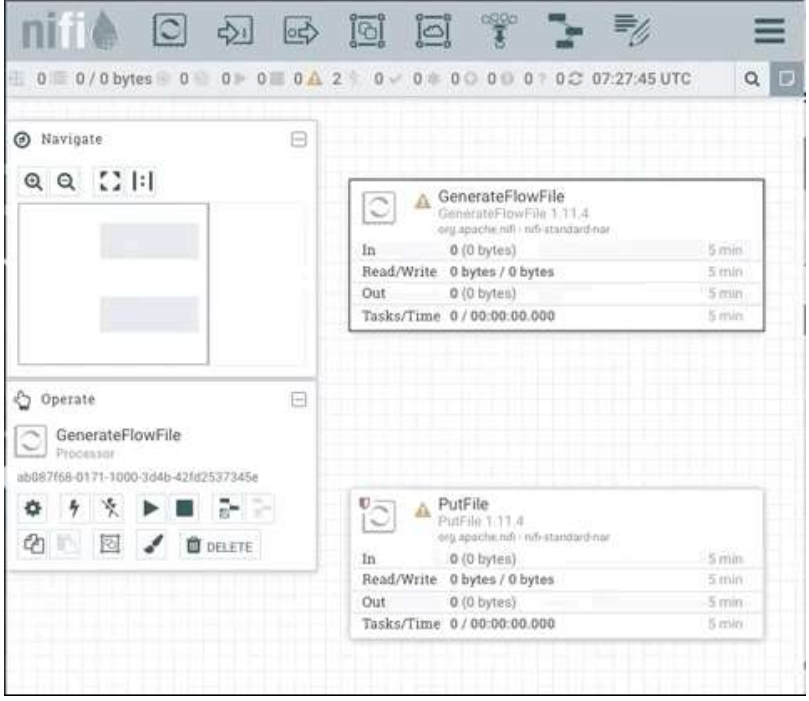


- Configure the ***PutFile*** processor:
  - In the ***Properties Tab*** set:
    - Directory: `/tmp/NiFi_ex1/`
  - The Properties tab should look like this:

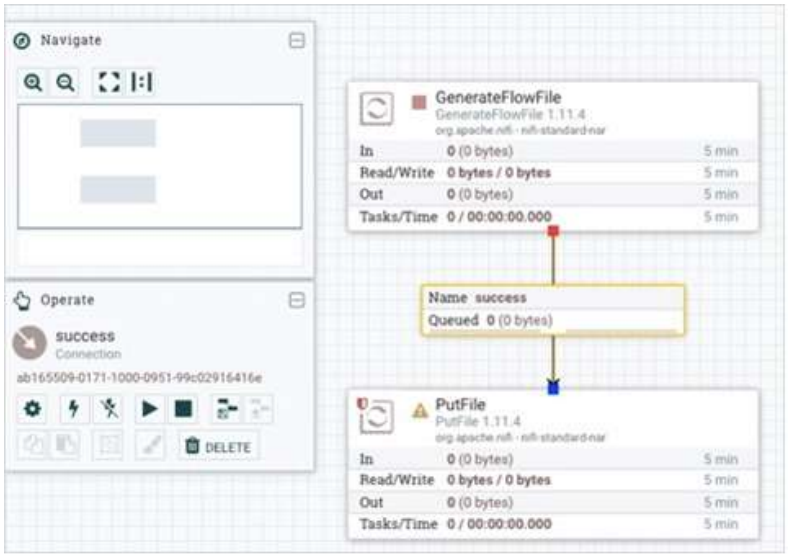
- In the ***Relationships Tab***:
  - Relationships Tab allows us to configure which outgoing relationships are not used, meaning where data is not sent and which outgoing pipes are **Automatically Terminated**.
  - This is very important, as every processor can have many outgoing relationships and NiFi will not allow us to start Processors when it is not clear how every outgoing relationship is used.
  - You will need to configure this for every Processor, setting any unused outgoing relationships as **Automatically Terminated**
  - Configure this processor to mark **Failure** and **Success** relationships as Automatically Terminated.
  - The Relationships Tab should look like this:



- Establishing connection between two processors.
  - Hover over the **''GenerateFlowFile'**Processor and drag the appearing arrow over to the other processor to establish relationship between them.
  - NiFi usually asks, which outgoing relationship to use, but there will only be one option for the **''GenerateFlowFile'**processor: **success**, which should already be selected.



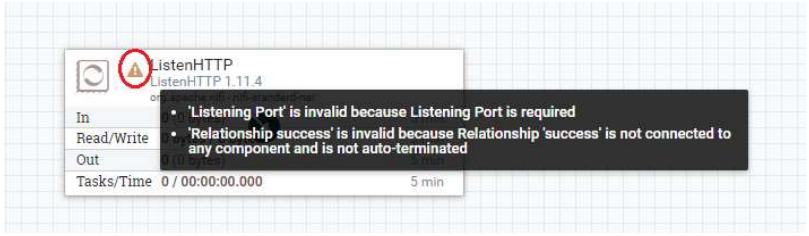
- The resulting pipeline should look like this:



Lets now start the pipeline and verify that data is being generated and stored to the file system:

- Right click on the "**GenerateFlowFile**' processor and select **Start** menu item.
- Similarly right click on the ***PutFile*** processor and select **Start** menu item.
- You verify the output of data pipeline in two ways:
    1. Through ***Data Provenance***:
        - For this, right click on ***PutFile*** processor.
        - Select ***View data provenance*** menu item
        - This will show the list of flow files that are handled by this processor.
        - Click on the ***i*** button in the first column of each record.
        - Goto ***CONTENT*** tab.
        - Click on ***View*** button.
    2. The second way could be to login VM and exec inside container `docker exec -it container-name /bin/bash` and verifying the files in the given directory `ls /tmp/NiFi_ex1/`
- Take a screenshot which displays the created pipelines (After starting them and testing them. PS! IP of the instance should be visible in the screenshot)
- Take a screenshot of the result (either through Data Provenance view, or by checking the output folder from the command line)

PS! To check issues related to NiFi Processors, you can hover your mouse over the Error icon:



Exercise 9.3. Creating NiFi templates

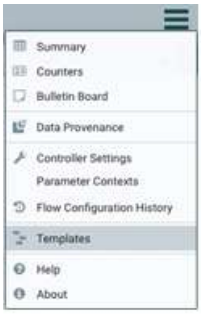
NiFi templates can be used to save NiFi pipelines as re-useable software artifacts that can be imported multiple times into NiFi canvas, downloaded as XML files, shared and uploaded into other NiFi deployments.

Lets create a NiFi template from the previous pipeline you created.

- Select the components you want to put into a template. In this case select all the components (***GenerateFlowFile***, **'PutFile*, and the* connection queue**'').
  - You can use shift-clicking to select multiple items.
- Right click on any selected component.
- Select ''**Create template**' option.
- Give a template name, and the description (optional).
- Now click on ''**create**' button followed by '**Ok**' button.

Steps to download a template

- Click on the icon present in the right top corner
- Select Template option.



- Now find/select the template you want to download.
- In the last column, click on the download icon.

Importing template

- First it is required to upload the template

- Click on search Icon -> select the file -> click on Upload button

- Drag and Drop Template Icon to the canvas

- Select the template name from the drop down list.
- Now click on *Add* button.
- Make sure to save the downloaded template, this will be one of the lab submissions.

Exercise 9.4. More advanced NiFi pipeline

This task focused on using NiFi pipelines to access the data from one service and store/move it to another location/service. This essentially gives an example of using NiFi for data movement/integration between two different services.

In this task, we will create a NiFi pipeline that:

1. Periodically queries the weather data of **Tartu city** from OpenWeatherMap API Service
2. Parses the JSON response data from Weather API
3. Stores the data in influxdb

Further, we will visualize the data using Grafana. We need to deploy Inflxudb and Grafana services in the VM for storage and visualization:

- Create a docker container of Influxdb

```
docker run -d --name influxdb -e INFLUXDB_DB=openweather -e INFLUXDB_ADMIN_USER=admin -e INFLUXDB_ADMIN_PASSWORD=CHOOSE_FREELY -p 8086:8086 influxdb:1.8
```

- Once the container is created, exec to the influxdb container and check the database created with the name *openweather* and play around with influx commands
- Exec the container `docker exec -it influxdb /bin/bash`
  - Open Influx client `influx`
  - Use command `show databases`
- Similarly create Grafana container `docker run -d --name grafana -p 3000:3000 grafana/grafana` .

9.4.1 Getting API token

We need to get API Token to access the weather data from the OpenWeatherMap Service

- Create an OpenWeatherMap service account if you do not already: https://home.openweathermap.org
- Confirm the email notification sent by OpenWeatherMap.
- After logging in and go to the API keys page to copy the API access information: `https://home.openweathermap.org/api_keys`
- Note!! We'll use the API key a little later in the below task.

9.4.2 Querying data from Weather API Service

In this task, we configure the **invokeHTTP** processor so that it sends an API request to the OpenWeatherMap service to request current weather data for Tartu City.

- Create a processor group with the name **OpenWeather Data** and we will use this processor group to create the NiFi pipeline.
- In this processor group, drag the invoke HTTP processor from the processor panel.
- Open the **Properties** view of the invokeHTTP processor
  - **HTTP Method** defines what type of HTTP API request, we leave it as it is: GET
  - **HTTP URL**: API, web service address. We set the value to `http://api.openweathermap.org/data/2.5/weather?lat=58.385835&lon=26.725940&appid=${token}&units=metric`
    - Tartu city coordinates are lat=58.385835 and lon=26.725940
    - The OpenWeatherMap API key goes with *appid* as a parameter:appid=${token}. The ${token} is a NiFI variable that is replaced by a variable string. We configure this variable separately to make it easier to manage/change.
  - Leave the rest of the values the same

- We configure invokeHTTP to send requests every 5 seconds
  - Open the *Scheduling* view of the invokeHTTP processor
  - Change the *Run Schedule* value to: 5 sec
- Under the RELATIONSHIPS settings of the processor, under *Original* outgoing connection, set this connection to closed (select *terminate*) and save (*Apply*)
  - For a correct processor, all outgoing connections must point to another processor, port or elsewhere, or be set to close. Otherwise, you will get an error message at startup.
  - We forward the remaining outgoing connections in the further task.

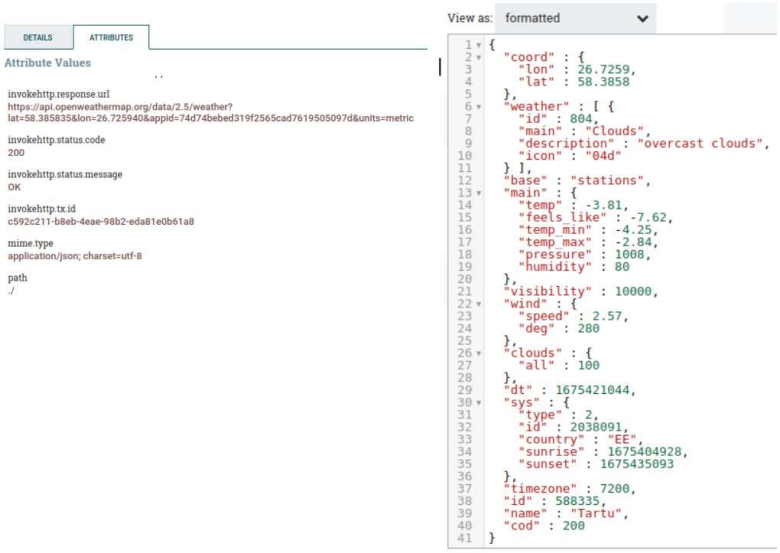
- Now let us set the API key token value
  - Exit your Processor group - back to NiFi's main view.
  - Right-click on your processor group and select *Variables*
    - Press the Plus button and add a new variable "token"
    - Set its value to the OpenWeatherMap API key you found earlier at Task 9.4.1 (https://home.openweathermap.org/api_keys)

9.4.4 Reading data from the received JSON response from OpenWeatherAPI

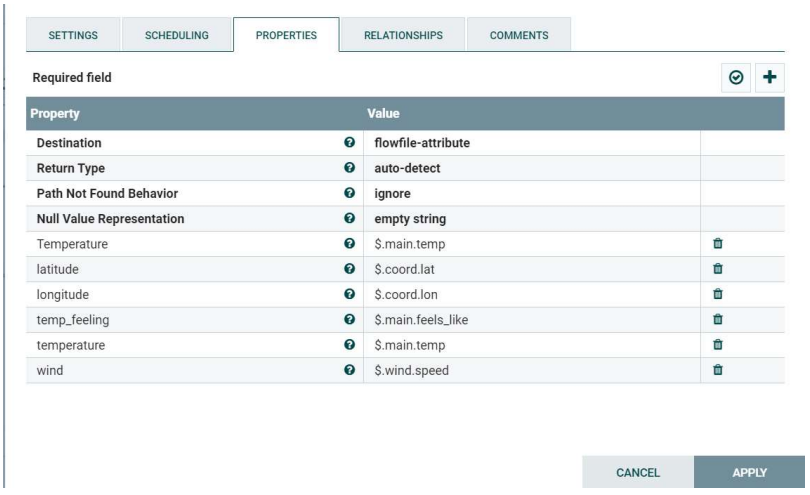
The goal now is to convert OpenWeatherMap output -> Influxdb Service input. We read out the necessary values (for example, temperature, humidity) in the JSON structure received by OpenWeatherMap from the object of the API response, and store them in FlowFile metadata as attributes, which we process further in the following tasks.

- Creating and connecting **EvaluateJsonPath** processor to the **InvokeHTTP** processor:
  - Add a new processor of type **EvaluateJsonPath**
  - Connect the output of the previous **InvokeHTTP** type processor *Response* to the input of this **EvaluateJsonPath** processor
  - A queue is now created between these two processors where FlowFiles that have been processed by the previous one and not yet processed by the next processor are stored.
- Testing the **InvokeHTTP** processor: Now that we have a queue between the two processors, we can test the previous **invokeHTTP** type processor. The name of the queue is *Response*, i.e. the name of the output stream of the invokeHTTP processor.
  - Add a new Output Port to the NiFi desktop
    - Name it: *errors*
    - Pull the InvokeHTTP connection with the mouse between the processor and the *errors* output port.
    - Activate relationships: *Failure*, *No Retry*, *Retry*. As a result, errors can be viewed if necessary content and attributes of generated FlowFiles.

- Check that the **invokeHTTP** processor is not a yellow triangle-shaped Error or warning icon.
- Verify that all invokeHTTP outputs are set to terminate or directed somewhere.
- Right-click on the invokeHTTP processor and select *Run Once* to run it manually for one time testing
- After testing it can be started by selecting *Start*. As a result, one FlowFile should appear in the queue.
- View the attributes and contents of the resulting FlowFile
  - Right-click on the queue and select "List queue"
  - There you will see a table of FlowFiles
  - By pressing either the "i" info icon or the eye icon, you can view the FlowFile properties and its contents.
- Attributes store FlowFile metadata. Since it comes from the InvokeHTTP processor, there is information about which request this object came from, what the HTTP code was, what format it is in, etc. This helps a lot in debugging data integrations and finding errors.
  - By pressing the *View* button (small eye icon), can view the contents of the FlowFile. Clicked on the *view as* option ''Formatted", to see the contents of the JSON object in a more beautiful form.



- Now let us continue configuring the **EvaluateJsonPath** processor,
  - Configuring processor parameters (PROPERTIES)
    - Destination: flowfile-attribute
    - We specify that values read from JSON are written as metadata. Add new parameters by pressing the small "+" button at the top right:
      - Add the key and the corresponding value
      - latitude `$.coord.lat`
      - longitude `$.coord.lon`
      - temp_feeling `$.main.feels_like`
      - temperature `$.main.temp`
      - wind `$.wind.speed`
      - humidity `$.main.humidity`
      - pressure `$.main.pressure`
    - As a result, the input JSON is searched values from within the data object with the corresponding JSON path:
      - $.coord.lat - is a JSON-based query that searches for the value under the "lat" key within the structure under the "coord" key in the JSON and set them in FlowFile metadata as correspondingly named attributes (latitude, temperature, ...) that can be used as variables in subsequent processors.
    - Leave the rest of the options the same
      - Direct the *failure*, *unmatched* output streams of this processor to the *errors* output port, as you did with the previous processor.
    - After setting the parameters, the result should be like this:



- The processor can be tested after we connect the "matched" output of this EvaluateJsonPath type processor to the input of the **ReplaceText** processor of the next task and start the processor.
- In addition, care must be taken to ensure that all outputs of this processor are closed or directed to the "errors" output port.

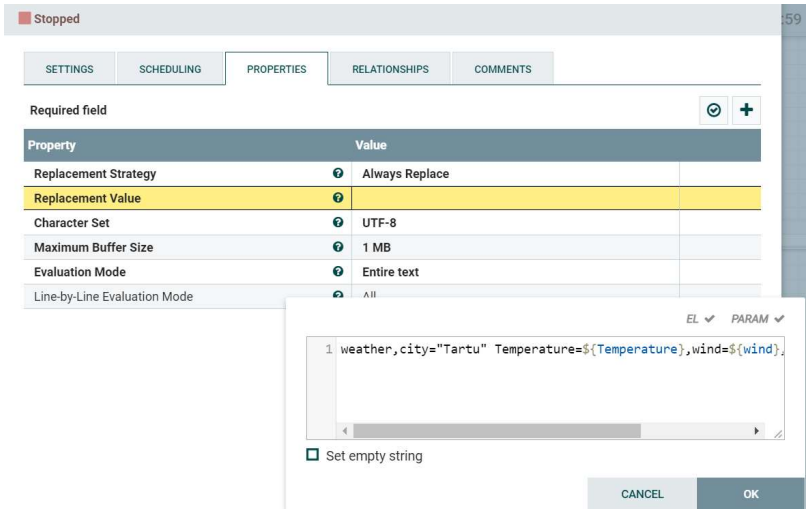
9.4.5 Prepare the data and save it to influxdb

- Now let us add the **Replace Text** processor to prepare the output data from the **EvaluationJSON** processor in to the line protocol format. The line protocol format is used by the Inflxudb to store the time series data. It has a format like

`measurement, tag field1,field2,filed3 timestamp`.

- Add the **Replace Text** processor from the panel and modify the properties.
  - Change **Evaluation Mode**to `Entire Text`
  - Change the **Replace Value** (Add the keys as recorded in EvaluatedJSON processor.):

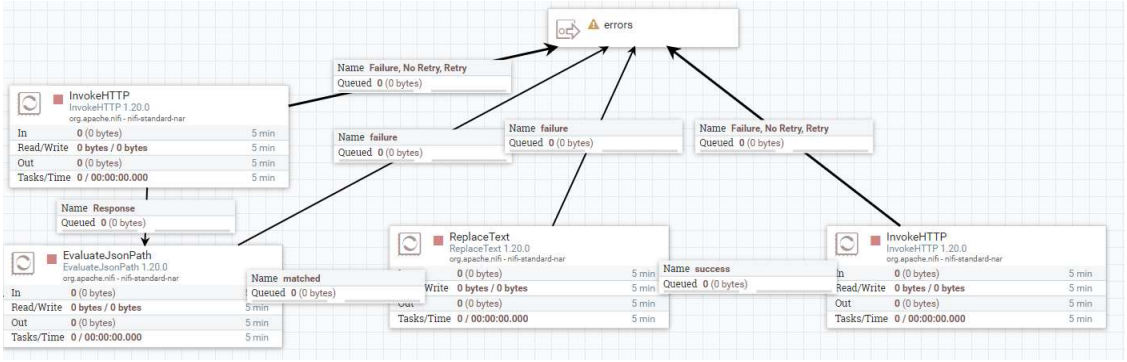
```
weather,city="Tartu" Temperature=${temperature},Wind=${wind},Tempfeeling=${temp_feeling},Humidity=${humidity},Pressure=${pressure}
```



- The processor can be tested after we connect the *success* output of this ReplaceText type processor to the input of the InvokeHTTP processor of the next task and start the processor.
- In addition, care must be taken to ensure that all outputs of this processor are closed or directed to the "errors" output port.

- Let us add the new **InvokeHTTP** processor to store the data into the influxdb.
  - Modify the properties as below:
    - HTTP Method: `POST`
    - HTTP URL: `http://172.17.65.188:8086/write?db=openweather` change the IP address to your VM IP.
    - Request Username: admin
    - Request Password: Add password while given in inflxudb
  - All outputs of this processor are closed or directed to the "errors" output port.

The final nifi pipeline looks like this:



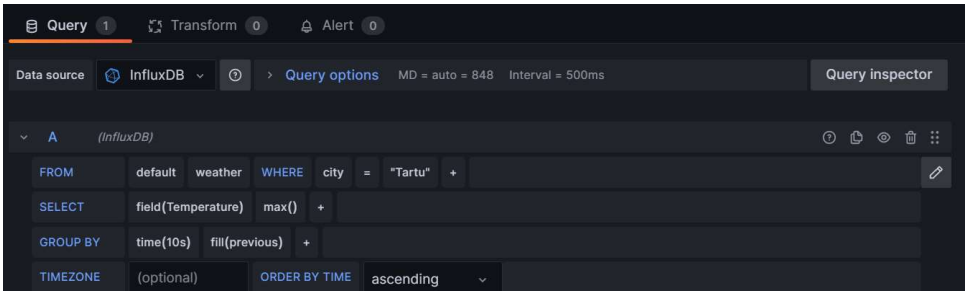
- Test the working of the complete pipeline and finally data should be stored in the influxdb.
  - Check the data stored in the influxdb by exec to the container and use the influx CLI and query the database.
    - List the database "show databases"
    - Use the database "Use openweather"
    - Query the db "Select * from weather"

9.4.6 Weather data visualization.

In this task, the Grafana service is used to create a dashboard and visualize the data.

- Access the Grafana service over browser `http://VM IP:3000`
  - Default username:admin and password:admin
- Add a data source
  - Move your cursor to the cog icon on the side menu which will show the configuration options. Click on the data sources -->Search for Influxdb-->Add with the following parameters
    - URL: http://VM IP:8086
    - Auth-->Basic auth, Add the username and password under **Basic Auth Details** (PS!! Influxdb username and password)
    - Database: openweather
    - Save and Test

- Create the dashboard (Guide is here)
  - Mouse over to Dashboard on left side, Click on + **New Dashboard**. Then click on Add a new panel.
    - Add the query to fetch the data from influxdb data source as shown below



- Save the panel and should see the data visualization with the graph as shown below:

- Similarly, you can create a dashboard for wind speed that feels like temperature.

- Take a screenshot of grafana dashboard for temperature and wind speed graphs (PS!! IP Should be visible) %
- Take a screenshot that displays the created pipeline (After starting them and testing them. PS! IP of the instance should be visible in the screenshot)

Exercise 9.5. Collecting and storing the weather data in the CSV format locally

In this task, you're going to collect and save the weather data in CSV format every 5 minutes (You can keep a longer time and 5 min is for testing). Most of the tasks are to be carried out by you based on the knowledge gained in the previous tasks.

The goal is to read the JSON data from OpenWeather, collect it in the CSV format for every 5 minutes, and store it in the local drive under the directory `/tmp/nifi`.

To perform this task, you need to use four processors. It continues the pipeline from the **EvaluateJSON** processor from the previous task.

- **Replace Text** Processor: This is used to format the data that is required to store in CSV.
  - Here, you need to change the Replacement Value:

```
${now():toNumber()},city="Tartu",Temperature=${temperature},Tempfeeling=${temp_feeling},Humidity=${humidity},Pressure=${pressure},Wind=${wind}
```

- We add a timestamp, city, and weather data as a columns in the csv.

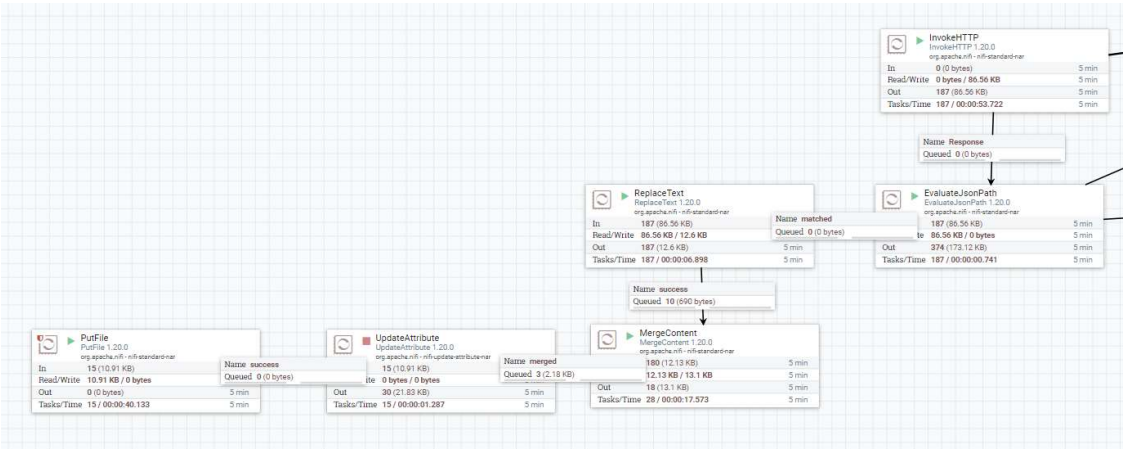
- **MergeContent** processor: This process is used to merge the data points together to fit in the CSV format. You should also add the header and newlines as necessary.
  - Change the Scheduling
    - Update the Run Schedule of this processor to *5 m*
  - Change the properties :
    - Delimiter Strategy: Text
    - Header: `timestamp, city, field1, field2 ...` (PS!! field means your recorded openweather data)
      - This is the column names in the CSV.
      - To get a newline after the last field, please press Shift+Enter, otherwise, you will see the data continuously merged with headers.
    - Demarcator: `${literal(' '):unescapeXml()}` (This will add new line to each row)
  - Here is an example of data after merging (PS!! Fields are not the same as yours, this is an just example):

```
timestamp,city,Temperature,Pressure,Humidity
1680555664546,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555659300,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555661406,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555665582,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555666633,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555667675,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555669767,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555672897,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555673939,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
1680555668730,city=Tartu,Temperature=-1.66,Pressure=1022,Humidity=71
```

- **UpdateAttribute** processor : This is used to update the **filename** attribute with value having `.csv` extension. Add a attribute to update using + in the Properties tab. Key as `filename` and value as `weather_${now():toNumber()}.csv`.
- **PutFile** processor: This is used to store the CSV files in the local directory `/tmp/nifi` of the nifi container. The properties are

- Directory: `/tmp/nifi`
  - This will store the filename attribute as a name in the directory.

The final nifi pipeline for this task looks like this:



- Start the pipeline and test by checking the files stored in the local directory of nifi.
  - You can exec to the container and list the files under /tmp/nifi

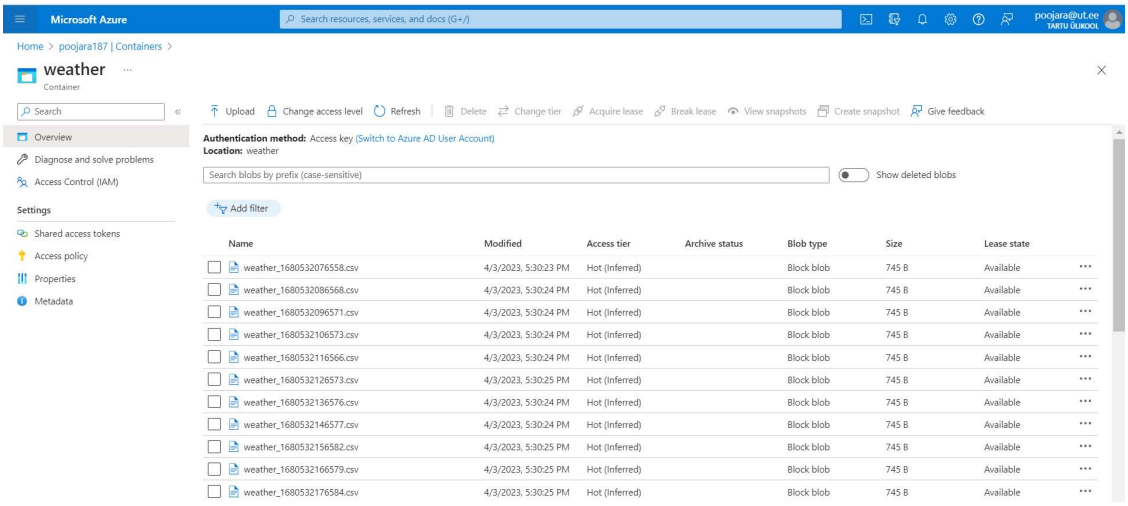
- Take a screenshot of the docker exec command that shows the listing of stored csv files %
- Take a screenshot which displays the created pipeline (After starting them and testing them. PS! IP of the instance should be visible in the screenshot)

Bonus exercise

The goal of the bonus task is to move the local csv files stored in `/tmp/nifi` directory to Azure Storage Service (blob Storage). To perform this task, you need to use three processors namely (The corresponding pipeline should look like ListFile-->FetchFile-->PutAzureBlobStorage):

- **ListFile** processor: This is used to list the latest entries of files in the directory (With the option **Listing Strategy** in properties). List and send the list of files to next processor.
- **FetchFile** processor: This is used the get the specified file with content as a flow file.
- **PutAzureBlobStorage** processor: This is used to store the files in the Azure blob storage.
  - Here, you need note down the storage account name and key1 from Azure Storage Account.
  - Update the corresponding keys and values in the Properties of the processor (Storage Account Name, Storage Account Key).
  - Further, update **Container Name** (This you can choose freely) and **Blob** (This should be ${filename}) in the properties.

- Connect the three processors and test the pipeline.
- You should see the files stored in the azure storage as shown below:



- Take a screenshot of the azure storage with a list of blobs as shown above %
- Create a and download as a template of the pipeline %

Deliverables:

1. Screenshots from tasks 9.2 and 9.4, 9.5
2. Templates from tasks 9.3 and 9.4, 9.5
   - Please try importing your saved templates to the canvas by yourself before submitting to verify you did not miss anything when saved the template.
3. Answer the following question:
   1. Which of the available NiFi processors looks most interesting or useful to you (which was NOT covered in this lab)
      - Why do you find it interesting or useful?
      - You can see the list of NiFi processors here: http://nifi.apache.org/docs.html
   2. Explain what is copy-on-write paradigm that NiFi Content Repository uses.
      - Read about the NiFi content Repository here: https://nifi.apache.org/docs/nifi-docs/html/nifi-in-depth.html#content-repository
      - Why is copy-on-write useful when dealing with large amount of streaming data?

4. Don't forget to delete your VM

Task	Lab 9 - nifi
Current submission	ZIP (21:10 05.04.2023)
	If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
File	Choose File No file chosen
Comment	

Submit

Troubleshooting

If a Processor is not behaving as expected (e.g. RouteOnAttribute runs, but nothing is matched), one way to get extra information is to check *data provenance*

- Right-click on Processor, choose *data provenance*.
- You will see a list of data the processor has handled, try clickin on the "i" icon to view details of a single item.
    - Here you can find Attributes of the FlowFile, which may contain useful information such as the message contents in case of this example.
- If you have started the QueryDatabaseTable processor, it keeps track of the database entry ID-s to query only for fresh entries. If for testing purposes, you would like to reset the "ID" counter and re-test with older messages, stop the processor, right click and select "View State", and select "Clear state" to reset the ID counter.
- The guide to creating a data source in the grafana is (Guide is here)