Project Athena

Done by Jakob Neuhauser, Ivonne Simic and Paul Pavlis

# Table of contents

[1. Table of contents 1](#_Toc43391847)

[2. Executive Summary 2](#_Toc43391848)

[3. System Architecture 3](#_Toc43391849)

[Diagram 3](#_Toc43391850)

[Procedure & Technologies 3](#_Toc43391851)

[Summary of the overall procedure 3](#_Toc43391852)

[Apache Nifi 4](#_Toc43391853)

[Elasticsearch / Kibana 4](#_Toc43391854)

[Hadoop HDFS & Apache Spark 4](#_Toc43391855)

[Apache Kafka 4](#_Toc43391856)

[Greenhouse Data 5](#_Toc43391857)

[Data Picture 5](#_Toc43391858)

[Description 5](#_Toc43391859)

[Field Description 5](#_Toc43391860)

[4. Steps to reproduce (from readme.txt) 7](#_Toc43391861)

[Preamble 7](#_Toc43391862)

[Procedure 7](#_Toc43391863)

# Executive Summary

TODO

# System Architecture

## Ein Bild, das Text, Karte enthält. Automatisch generierte BeschreibungDiagram

## Procedure & Technologies

### Summary of the overall procedure

To start the whole process, we need to receive the sensors data via messages, so that we can transform and use them for the following steps. This is being achieved with Apache Nifi. The Program listens on a specific port and each sensor data entry that is sent there, will be converted into a usable file (flowfile in Nifi). These flowfiles then undergo a check and are transformed to the right json format for later use.

We use these sensor data sets for three different workflows:

* (Near) Real-Time Analysis and Dashboard
  + The data is transported (with Apache Nifi) to Elasticsearch, where it will be stored at an index (index can basically be associated with a table in SQL). From there the data will be accessed with Kibana and mapped onto different Graphs. These Graphs show the flow of the current (and near past) sensor data and the different values for e.g. the temperature or soil moisture.
* Long-term storage and queries on the dataset
  + First, the data from the different Nifi flowfiles are merged into fewer larger files, so that the following technology is more efficient in handling large amounts of data. The data is then stored in HDFS (Hadoop distributed file system), which basically just stores the data for later use. Then a Query Engine (Apache Spark) is used to make queries on the whole dataset to get e.g. the average temperature of the year 2019. The data from the analysis can be used to correct different aspects of the greenhouse.
* Check for critical values and changing the hardware accordingly
  + In this pipeline, the values of the sensor data are checked (in Nifi) against predefined maximal and minimal values. If the value is above or below the threshold, the file will be sent to a message broker (Apache Kafka). From there the critical values can be extracted and sent to the hardware to change these values on e.g. the flowerpots. (The Hardware aspect is out of scope for this project. The data will only be provided, not used)

### Apache Nifi

TODO

> Beschreibung der Technologie (per se)  
> Relativ genaue Beschreibung und Ablauf der Verwendung  
> Begründung, warum diese Technolgie

### Elasticsearch / Kibana

TODO

### Hadoop HDFS & Apache Spark

TODO

### Apache Kafka

TODO

## Greenhouse Data

### Data Picture

### Description

This part explains the data that is being used for the whole project. You can see a snippet of the data in the previous paragraph. Each of those rows represents one entry of a sensor from the greenhouse. It includes a timestamp and the fields “fertilizer\_level”, “light”, “soil\_moisture\_percent” and “air\_temperature\_celsius”.

At the start we were looking for an open-API, which would send us sensor data as a stream of data, but we could not find one for free. To compensate for that, we chose to write our own program, that sends data as a stream to a specific internet address and port. The data that is being used for that purpose, is a real data set from kaggle: [https://www.kaggle.com/ludwa6/soil-sensor](https://www.kaggle.com/ludwa6/test-soil-sensor)

### Field Description

The following description is for the specific dataset, that we are sending from our script to the address to simulate real sensor data. Another field has been added so that a distinction between different sensors is possible. Example data entry from a sensor in json format:

{"name": "sensor1", "time\_measured": "2020-06-12T22:34:22Z", "fertilizer": 1.02, "light": 8.83, "soil\_moisture\_percent": 12.02, "air\_temperature": 24.25}

* Field: name
  + This field includes the name of the sensor that recorded and send the data. In our example we are just using one kind of sensor and the sensor are named with “sensor” + [number].
  + Type: String
* Field: time\_measured
  + This field describes the exact time, where the sensor recorded the data. Additionally, the data is then formatted to conform to [ISO\_8601](https://en.wikipedia.org/wiki/ISO_8601), which is a universally accepted time format.
  + Type: Timestamp
* Field: fertilizer
  + This field describes the level, at which the fertilizer is (e.g. in a flowerpot).
  + Type: Decimal / Float
* Field: light
  + This field describes the amount of light that the plant (and sensor) is getting. This varies heavily if the light is turned of or if a cloud is blocking the sun.
  + Type: Decimal / Float
* Field: soil\_moisture\_percent
  + This field describes the percent of moisture (wetness) in the soil/ground. This variable degrades lightly over time and then makes a jump to high number again (when it is watered)
  + Type: Decimal / Float
* Field: air\_temperature
  + This field describes the air temperature that the plant (and sensor) is getting. If the value is too high or too low, rapid adjustments must be made.
  + Type: Decimal / Float

# Steps to reproduce (from readme.txt)

## Preamble

The following paragraph was written with GitHub Syntax and is better viewed there. You can choose to view it here as well, but for the correct highlighting you should consider going to: <https://github.com/h1v3r/Athena/>

Github is a place where you can store and get projects from people that are using the technology git. Git is a version control system, which helps you and your team to manage the project so that there is not a conflict with different version of the same document.

## Procedure

# Athena

## How to start up Athena

### Prerequirements

You need to install [\*\*docker\*\*](https://www.docker.com/) as well as [\*\*docker-compose\*\*](https://docs.docker.com/compose/). On an [\*\*Arch Linux\*\*](https://i.redd.it/tcdhu46p4y451.jpg) System this can be done with:

`sudo pacman -S docker docker-compose`

### Clone from GitHub

First you need to download the project from GitHub.

`git clone https://github.com/h1v3r/Athena.git`

After that you change into the git folder you just downloaded.

`cd Athena`

### Start the containers

Before you can start everything up, you need to change the permissions for two scripts. You will need root privileges for that.

`sudo chmod 645 init\_athena.sh rm\_athena.sh`

Now you can start the "init\_athena" script. You need to do this as root too.

`sudo ./init\_athena.sh`

Now everything is starting up. If there is a problem with Elasticsearch and permissions, try:

`sudo chmod 777 ../Athena\_Data`

in the git folder. It may solve the problem.

### Set up NiFi

Next you need to implement the template for NiFi. NiFi will be available at:

`localhost:8080/nifi/`

It may take a bit till it is up and running. On the NiFi web interface you need to navigate you curse over the grid. Then right click on it and select “Upload template”. A window will pop up where you can select the template you want to upload. Click on the icon with the magnifying glass. If you have done that, a window will open up where you can select the template you want to upload. Simply navigate to your git folder and select “NiFi-Template.xml”. After this you must click “upload” and the template will be uploaded.

Now navigate to the icon with the name “Template” in the top bar and drag and drop it into the grid. Now you can choose what template you want to use. Select “NiFi-Template” and click “ADD”. The Template should appear on the grid.

First you need to click somewhere on the grid to dis-select everything. The template consists of many processors (large and rectangular). At every processor you can find a red square which indicates that the processor is turned off. To turn a processor on you right click on it and select start. You need to do this for every processor.

### Start the API

After you have set up NiFi you can start the API with:

`python python\_SendData\_simultan.py`

To accept the default settings of the script click enter.

### Pause the Program

The following steps "Pause the Program", "Remove Athena" and "Portainer" are optional and not needed for the workflow. It provides additional information’s to maximize your docker experience or stop and remove the whole application when you are done.

You can bring all containers down by executing

`sudo docker-compose down`

inside the git folder. Use

`sudo docker-compose up -d`

to bring them up again. However, you need to setup NiFi again.

### Remove Athena

First you want to stop the API. Simply click into the shell where you have started the API and press "Ctrl-C".

Next you simply have to execute the "rm\_athena.sh" script.

`sudo ./rm\_athena.sh`

This will delete all data related to the project except the git repo and date in relation with docker.

### Portainer

The "docker-compose" includes a Portainer container which can be accessed at:

`localhost:9000`

## Analyse the data with Kibana

Kibana will be available at:

`localhost:5601`

After you have entered the Kibana web interface you need to click on “Dashboard” at the menu on the left. At “Dashboards” select “Greenhouse\_Dashboard”.

If you are not able to see the dashboard and you get the message that you need to define an index first, then click on "Management" at the menu on the left. Under the header "Kibana" you will find the point "Saved Objects". After clicking that, there should be an option to import a kibana file which is located in your directory at "./zz-Archive/Kibana-save-1.ndjson

Now you should see a few rows added to the list including the index "testindex2" where our data is stored and "Greenhouse\_Dashboard" where the Dashboard is located. Either click on it or go the menu point "Dashboards" at the menu on the left and choose the correct Dashboard.

Now you are at the Dashboard. On the top you will find a console where you can group by “Sensor Name” or select ranges for the parameters. Below that you can find diagrams for the count of measurements per time interval and a pie chart where you can see the shares of each sensor. Under those diagrams you can find four more, each representing one parameter (Air Temperature, Fertilizer, Light and Soil Moisture (in percent)). Each graph displays the maximum, minimum and median per time interval for the corresponding reading.

If you get the message "No data available" at the dashboard screen, try changing the time window at the top right of the window to a different interval (e.g. today).

## Analyse HDFS with Spark (example)

To open the spark shell you need to access the Spark container with:

`sudo docker exec -it Athena-spark-master-1 /bin/bash`

Then you can open the spark shell.

`./spark/bin/spark-shell`

On the spark shell you first need to create a data frame.

`val sensorDF = spark.read.json("hdfs://namenode:9000/GreenhouseArchiveTest1")`

With this data frame you can create a view:

`sensorDF.createOrReplaceTempView("sensors")`

You can select from this view with sql statements and print the result.

`val all\_data = spark.sql("select \* from sensors")`

`all\_data.show()`

You can make an example display of all data in a certain time interval (in this example all data on "2020-06-14"),

`val max\_for\_time = spark.sql("select max(air\_temperature), max(fertilizer), max(light), max(soil\_moisture\_percent) from sensors where time\_measured like '2020-06-14%'")`

`max\_for\_time.show()`

or group by sensor.

`val avg\_per\_sensor = spark.sql("select name, avg(air\_temperature), avg(fertilizer), avg(light), avg(soil\_moisture\_percent) from sensors group by name order by name")`

`avg\_per\_sensor.show()`

If you want to export the data you collected you can do that by saving the data at the “/spark-data” directory. This directory is mounted at “../Athena\_Data/spark-data” at you host machine.

`val op= avg\_per\_sensor.rdd.map(\_.toString().replace("[","").replace("]", "")).saveAsTextFile("/spark-data/test")`