Technical Documentation

Big Data Analytics

PfSense Firewall Analysis

Jonas Färmann, Dena Karini, Lorenz Wackenhut

Examiner:	Prof. Dr. Renner
Submission Date:	28.01.2021

Table of Contents

Table of Figures

Tables

1 Introduction

The client Frachtwerk GmbH asked us to perform pattern recognition on a dataset of approximately 15 Mio data points of internet traffic retrieved by their firewall between Sept-Dec 2020. The deliverables were the analysis of the data, the identification of security threat patterns and a visual interface to view the most relevant results.

2 Purpose of the document

This document describes the specific features of the developed tool, e.g. to facilitate adaptation or maintenance later. In particular, the code that has been written for this tool and its functionality is explained.

3 Technical requirements

The technical requirements were agreed with the client in advance and recorded in https://moodle.htw-berlin.de/mod/resource/view.php?id=828486.

4 Development and deliverables

To ensure a reliable, performant and well documented development process 4.1) GitHub, 4.2) Docker, 4.3) Jupyter Notebooks, 4.4) Spark, 4.5) Citus and 4.6) Grafana were used. Their synergies are displayed in Figure 1.

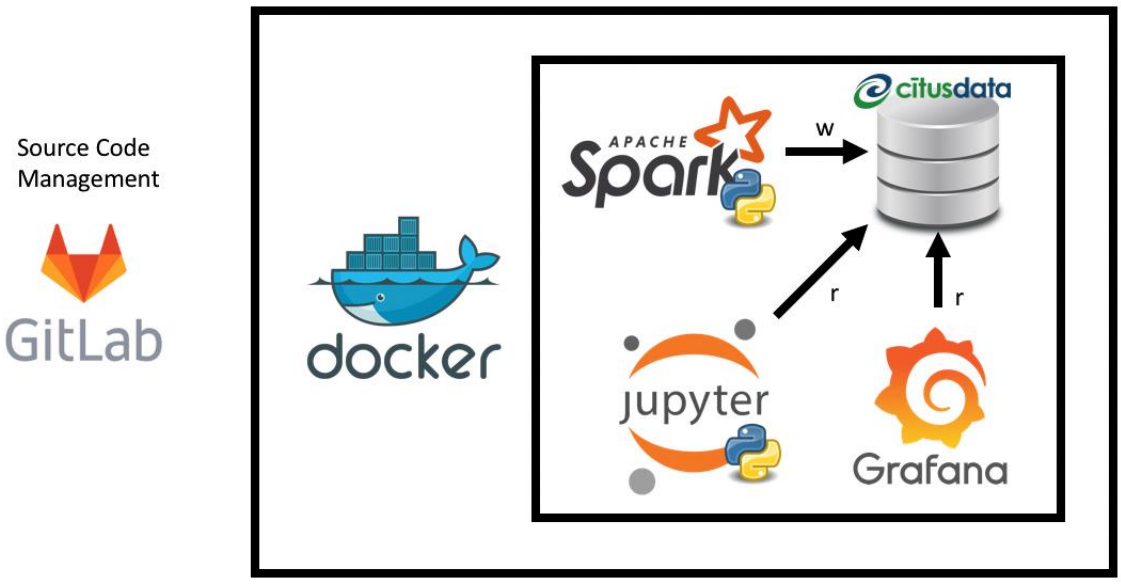


Figure 1 Tools and their synergies (r = read, w= write)

4.1 GitLab

The git repository can be found under https://gitlab.com/jonasfaehrmann/2021-05-08-Group-6-Log-Analysis-Source. As the private setting was enabled to protect the repository from public viewers, each developer needs to be added by the administrator. This can be done under https://gitlab.com/jonasfaehrmann/2021-05-08-Group-6-Log-Analysis-Source/-/project_members. If desired, the repository can also be changed to public at any time under https://gitlab.com/jonasfaehrmann/2021-05-08-Group-6-Log-Analysis-Source/edit > Visibility, project features, permissions. The git commit history shows that the development was done by three developers (Figure 2).

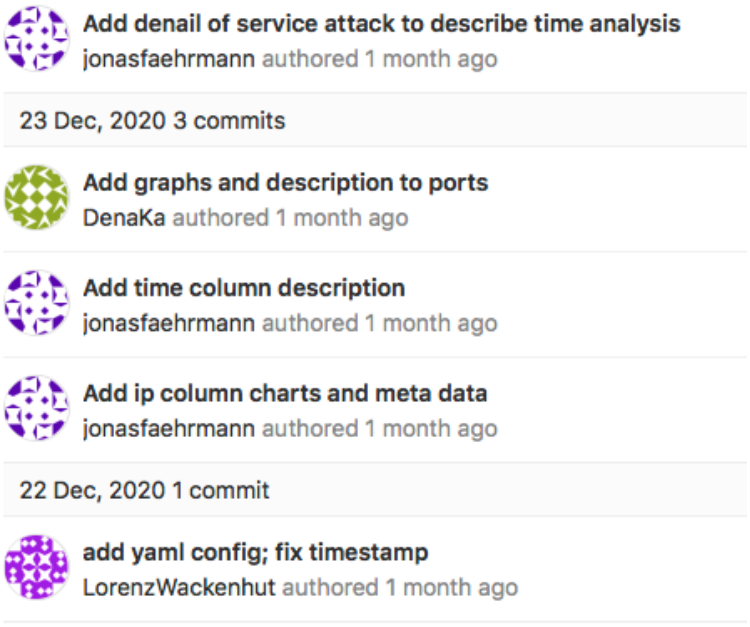


Figure 2 Git commit history

To make rollbacks easy most of the commits follow the same commit message structure as shown in Table 1.

Table 1 Git commit message structure

What	Why (optional)
Add <feature subject>	to <matter>

Due to the usage of different operating systems and tools throughout the development process the `.gitignore` was iteratively modified to remove unnecessary files. Additionally, the open-source MIT license was added to enable further development without restrictions.

Figure 3 shows the folder structure for the project and is being referenced throughout the document for every path.

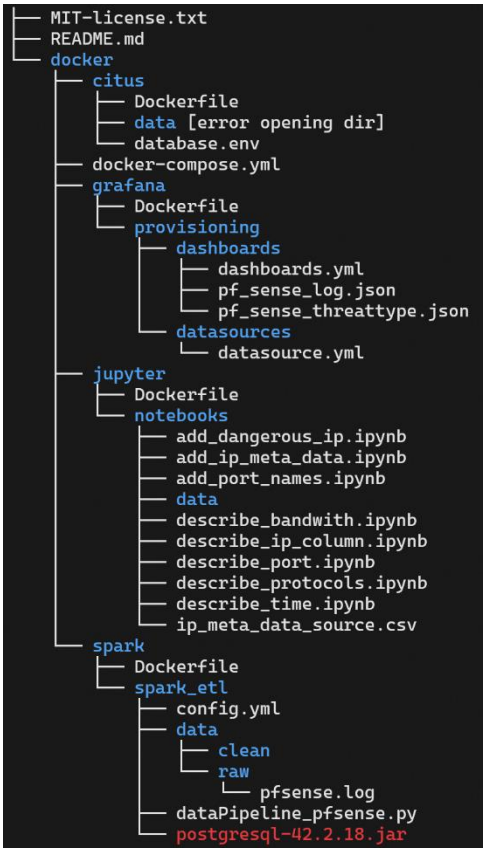


Figure 3: Folder structure

4.2 Docker

Docker is an open-source framework and the de-facto standard for the development, shipping, and running containerized applications isolated from the local infrastructure and was used for this project. (Docker 2021)

As the folder structure in Figure 3 depicts, every service folder (e.g. spark, grafana etc.) has its own dockerfile. Those files act as the build guide for the image of a docker container. In every instance an image from 'Docker Hub' servers as the main component and gets augmented with specific additional dependencies (Table 2).

Table 2: Coresponding Docker Images for every Service

Service	Docker Hub Image
spark	citusdata/citus
citus	grafana/grafana
grafana	jupyter/datascience-notebook
jupyter	godatadriven/pyspark

The ‘docker-compose’ file is the most important part of the containerization, as it orchestrates how the individual services are run and how they interact with each other. All of the containers are launched in order and wait for the completion of the previous one. This ensures that the data from the previous step is ready to load as soon as the current service is started. The ‘volume’ attribute enables to mount directories from the local machine into the docker container and vice versa. With this approach data between the host system and the dockerized application can be shared. This is especially important for submitting the spark job, the persistence of data for Citus and the provisioning for the Grafana dashboard. With the keyword ‘link’ a virtual network between the docker instances is generated, exposing the ip of the container to other docker instances.

The command `docker-compose up` starts the whole datapipeline by first checking if all the defined images are already download. If not, the latest version of the service is build using the respective Dockerfile. This process can take several minutes, as PySpark and Jupyter are relatively heavy in their memory footprint. After all images are build, the containers can be launched.

The command ‘`docker-compose down`’ tears down the whole architecture.

“By default, the only things removed are:

- Containers for services defined in the Compose file
- Networks defined in the networks section of the Compose file
- The default network if one is used” (Docker 2021)

```
version: "3.5"
services:
  citus:
    build:
      context: ./citus
      shm_size: '256mb'
    container_name: citus
    env_file:
      - ./citus/database.env
    ports:
      - 5432:5432
    volumes:
      - ./citus/data/:/var/lib/postgresql/data/
    shm_size: '256mb'

  spark:
    build:
      context: ./spark
    container_name: spark
    depends_on:
      - "citus"
    ports:
      - 4040:4040
    volumes:
      - ./spark:/job
      - ./jupyter/notebooks/data/:/jupyter/notebooks/data/
    links:
      - citus
    command: ["--driver-class-path",
              "job/spark_etl/postgresql-42.2.18.jar",
              "job/spark_etl/dataPipeline_pfsense.py"]

  grafana:
    build:
      context: ./grafana
    container_name: grafana
    environment:
      - GF_INSTALL_PLUGINS=grafana-worldmap-panel
    ports:
      - 3000:3000
    links:
      - citus
    depends_on:
      - "citus"
      - "spark"

  jupyter:
    build:
      context: ./jupyter
    container_name: jupyter
    links:
      - citus
    depends_on:
      - "citus"
      - "spark"
    ports:
      - 8888:8888
    environment:
      - JUPYTER_TOKEN=bda
```

Figure 4: Docker Compose

4.3 Jupyter Notebooks

Jupyter Notebooks were used to document the initial analysis process. After the "`docker-compose up`" command (section 4.2 Docker) was run, the notebooks can be accessed under http://localhost:8888?token=bda (Figure 4).

Figure 5 Jupyter notebooks

If a notebook shows up as non-modifiable, the trust button in the top right must be clicked (Figure 4).

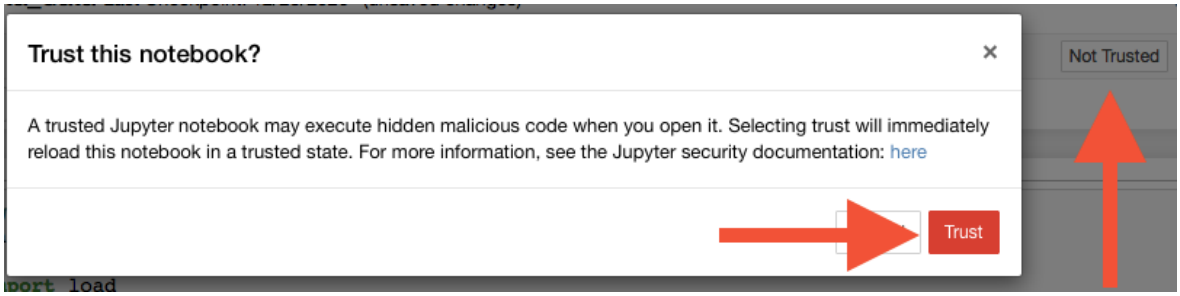


Figure 6 Trust jupyter notebook

To modify and edit the Jupyter notebooks locally, without Docker, Python 3.7 (https://www.python.org/downloads/) has to be installed. Additionally, the libraries matplotlib, pandas, pyarrow, numpy and seaborn need to be retrieved using:

```
# Python 3.7 required
pip install matplotlib pandas pyarrow numpy seaborn
```

Finally, the installation instructions on jupyter notebook need to be followed (https://jupyter.org/install). After this navigate to the path "`./docker/jupyter/notebooks/`" and run "jupyter notebook ." with the terminal of your choice.

4.3.1 Preloading Data

In case the pipeline should not be run through and only the analysis is of interest, preloaded data can be insert into the notebooks. We uploaded a file called `'2021-05-08-Group-6-Log-Analysis-Data.zip'` on Frachtwerks storage account. The file has to be unzipped and contains two more archives called citus.zip and parquet.zip. The latter has to be placed in the folder `'./docker/jupyter/notebooks/data'`. Here the file must be unzipped. Now only the database and the desired analytical service can be run with the command:

```
>> docker-compose up jupyter
```

4.4 Spark

'PySpark' was used to build a performant ETL-pipeline that can easily be deployed on a local machine or in cluster mode with the change of the spark submit parameter. The Spark container is started within the docker-compose script and submits a job via the command attribute in docker. In the case of this particular use case the application is submitted with the statement: `'command: ["--driver-class-path", "job/spark_etl/postgresql-42.2.18.jar", "job/spark_etl/dataPipeline_pfsense.py"]'` (Figure 4). Additionally, a jdbc-driver is supplied, as it is needed to write to Citus.

4.4.1 Configuration

The spark job can be configured using the `config.yml` in the same folder as datapipeline_pfSense.py (Figure 4). The following attributes can be modified:

- `sample_size: int`
  Defines the number of rows that should be processed. Running the pipeline with all entries takes several hours. Running with 10000 entries is recommended for testing purposes
- `source_path: string`
  Path to raw log files.
- `write_mode: string`
  Defines the write mode. Possible values: "parquet"; "database"
- `db_url: string`
  jdbc connection string for Citus.
- `db_properties: dictionary`
  User credentials to connect to Citus.
- `dest_path_v4: string`
  Output folder.
- `dest_path_v6: string`
  Output folder.

- `dest_path_log: string`
  Output folder.
- `schema_v4: dictionary`
  Used to cast columns into correct datatypes.
- `schema_v6: dictionary`
  Used to cast columns into correct datatypes.
- `final_cols_v4: list`
  Columns to be selected and reordered
- `final_cols_v6: list`
  Columns to be selected and reordered

4.4.2 Class Design

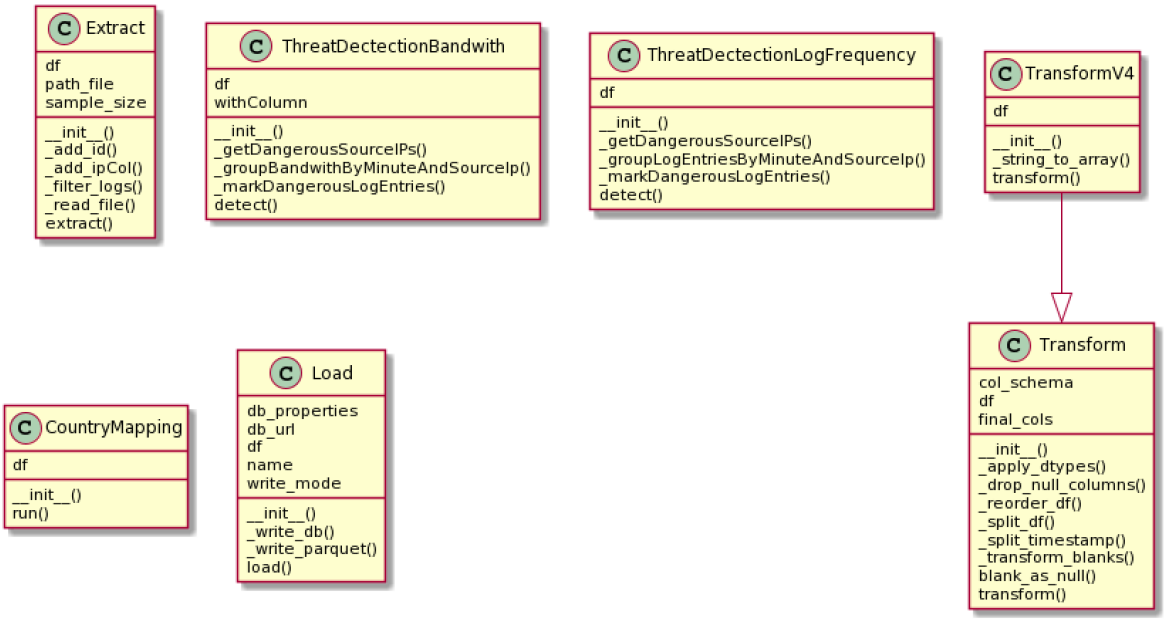


Figure 7: UML Class Diagram Spark ETL-Job

All explanations in the following section reference the class diagram in Figure 7.

4.4.2.1 Extraction

The extraction process ingests the raw firewall log and performs light pre-processing steps to enable the transformation in the next step. The corresponding class is called 'Extract'. The following steps performed:

- `_add_id()` : Adds a monotonically increasing ID which enables the separation of the data into ip4, ip6 and log entries.
- `_add_ipCol()` : Extracts and adds the ip information in order to support splitting later on.

- `_filter_logs()` : Filters log rows from data rows.
- `_read_file()` : Reads the raw text file.
- `extract()` : Queues all previous methods and executes them.

4.4.2.2 Transformation

The transformation encompasses all methods that are used to clean and format the data-frames. This step is separated into two classes, as ip4 and ip6 entries inherit different columns. The only method which is different is `_string_to_array()`.

- `_apply_dtypes()` : Casts columns according to col_schema in data types.
- `_drop_null_columns()` : Drops columns with nothing but null value.
- `_reorder_df()` : Selects and reorders columns according to final_cols
- `_split_df()` : Extracts columns from text and adds them do the dataframe
- `_split_timestamp()` : Extracts the timestamp and formats it.
- `_transform_blanks()` : Transform blank strings into null values so they can be dropped.
- `_string_to_array()` : Formats the column 'options' in ip6 entries as list.
- `Transform()` : Queues all previous methods and executes them.

4.4.2.3 Enrichment

The Enrichment step incorporates all tasks which add additional information to the cleaned and processed data. Three classes belong to this step:

- `ThreatDetectionLogFrequency()`
  - `_getDangerousSourceIPs()` : Counts requests by same IP in minute.
  - `_groupLogEntriesByMinuteAndSourceIp()` : Groups entries by minutes.
  - `_markDangerousLogEntries()` : Flags entries.
  - `detect()` : ueues all previous methods and executes them.
- `ThreatDetectionBandwith()`
  - `_getDangerousSourceIPs()` : Counts bandwith by same IP in minute.
  - `_groupLogEntriesByMinuteAndSourceIp()` : Groups entries by minutes.
  - `_markDangerousLogEntries()` : Flags entries.
  - `detect()` : ueues all previous methods and executes them.
- `CountryMapping()`
  - `run()` : Enriches entries by requesting information from https://api.ip.sb/geoip/

4.4.2.4 Loading

The last step in the ETL job is the loading of the data. According to the write mode the data is either loaded into Citus or saved as Parquet files.

- `_write_db()`: Writes dataframe into Citus
- `_write_parquet()`: Writes dataframe as Parquet
- `load()`: Queues all previous methods and executes them.

4.5 Citus

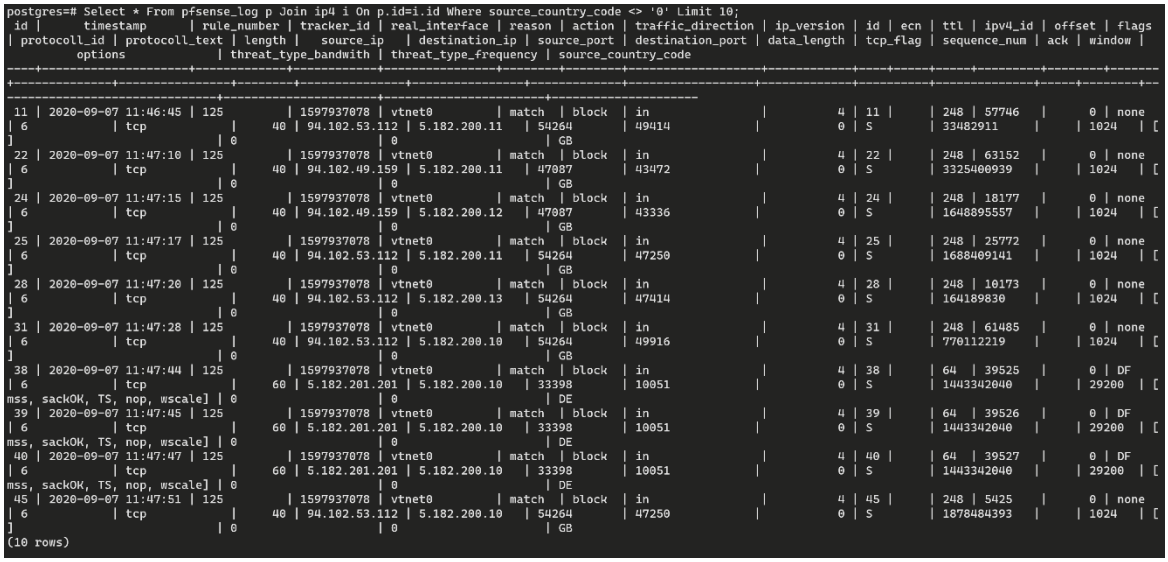
Citus is a database system bases on Postgres and enables the distributed storage of relational tables (Citusdata 2021). The dataframes from the previous step are stored in three tables (Figure 8) named pfsense_log (Figure 9), ip4 and ip6. The tables can be joined using the ID column (Figure 10).

```
                List of relations
 Schema |     Name      | Type  |  Owner
--------+---------------+-------+----------
 public | ip4           | table | postgres
 public | ip6           | table | postgres
 public | pfsense_log   | table | postgres
(3 rows)
```

Figure 8: Tables in Citus

```
postgres=# Select * From pfsense_log Limit 10;
 id |      timestamp      | rule_number | tracker_id | real_interface | reason | action | traffic_direction | ip_version
----+---------------------+-------------+------------+----------------+--------+--------+-------------------+------------
-
  0 | 2020-09-07 11:45:50 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
  1 | 2020-09-07 11:45:52 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
  2 | 2020-09-07 11:46:00 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
  3 | 2020-09-07 11:46:00 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
  4 | 2020-09-07 11:46:12 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
  5 | 2020-09-07 11:46:12 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
  6 | 2020-09-07 11:46:13 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
  7 | 2020-09-07 11:46:16 | 5           | 1000000003 | vtnet3         | match  | block  | in                |          6
  8 | 2020-09-07 11:46:35 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
  9 | 2020-09-07 11:46:35 | 200         | 1599469328 | vtnet3         | match  | block  | in                |          4
(10 rows)
```

Figure 9: Example Select in Citus



```
postgres=# Select * From pfsense_log p Join ip4 i On p.id=i.id Where source_country_code <> '0' Limit 10;
 id |      timestamp      | rule_number | tracker_id | real_interface | reason | action | traffic_direction | ip_version | id | ecn | ttl | ipv4_id | offset | flags
| protocoll_id | protocoll_text | length |    source_ip    | destination_ip | source_port | destination_port | data_length | tcp_flag | sequence_num | ack | window |
        options             | threat_type_bandwith | threat_type_frequency | source_country_code
----+---------------------+-------------+------------+----------------+--------+--------+-------------------+------------+----+-----+-----+---------+--------+-------
+--------------+----------------+--------+-----------------+----------------+-------------+------------------+-------------+----------+--------------+-----+--------+--
-------------------------+----------------------+-----------------------+---------------------
 11 | 2020-09-07 11:46:45 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 11 |     | 248 | 57746   |      0 | none
| 6            | tcp            |     40 | 94.102.53.112 | 5.182.200.11   | 54264       | 49414            |             |        0 | S        | 33482911     |      | 1024   | [
]                          | 0                    | 0                     | GB
 22 | 2020-09-07 11:47:10 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 22 |     | 248 | 63152   |      0 | none
| 6            | tcp            |     40 | 94.102.49.159 | 5.182.200.11   | 47087       | 43472            |             |        0 | S        | 3325400939   |      | 1024   | [
]                          | 0                    | 0                     | GB
 24 | 2020-09-07 11:47:15 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 24 |     | 248 | 18177   |      0 | none
| 6            | tcp            |     40 | 94.102.49.159 | 5.182.200.12   | 47087       | 43336            |             |        0 | S        | 1648895557   |      | 1024   | [
]                          | 0                    | 0                     | GB
 25 | 2020-09-07 11:47:17 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 25 |     | 248 | 25772   |      0 | none
| 6            | tcp            |     40 | 94.102.53.112 | 5.182.200.11   | 54264       | 47250            |             |        0 | S        | 1688409141   |      | 1024   | [
]                          | 0                    | 0                     | GB
 28 | 2020-09-07 11:47:20 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 28 |     | 248 | 10173   |      0 | none
| 6            | tcp            |     40 | 94.102.53.112 | 5.182.200.13   | 54264       | 47414            |             |        0 | S        | 164189830    |      | 1024   | [
]                          | 0                    | 0                     | GB
 31 | 2020-09-07 11:47:28 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 31 |     | 248 | 61485   |      0 | none
| 6            | tcp            |     40 | 94.102.53.112 | 5.182.200.10   | 54264       | 49916            |             |        0 | S        | 770112219    |      | 1024   | [
]                          | 0                    | 0                     | GB
 38 | 2020-09-07 11:47:44 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 38 |     | 64  | 39525   |      0 | DF
| 6            | tcp            |     60 | 5.182.201.201 | 5.182.200.10   | 33398       | 10051            |             |        0 | S        | 1443342040   |      | 29200  | [
mss, sackOK, TS, nop, wscale] | 0                    | 0                     | DE
 39 | 2020-09-07 11:47:45 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 39 |     | 64  | 39526   |      0 | DF
| 6            | tcp            |     60 | 5.182.201.201 | 5.182.200.10   | 33398       | 10051            |             |        0 | S        | 1443342040   |      | 29200  | [
mss, sackOK, TS, nop, wscale] | 0                    | 0                     | DE
 40 | 2020-09-07 11:47:47 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 40 |     | 64  | 39527   |      0 | DF
| 6            | tcp            |     60 | 5.182.201.201 | 5.182.200.10   | 33398       | 10051            |             |        0 | S        | 1443342040   |      | 29200  | [
mss, sackOK, TS, nop, wscale] | 0                    | 0                     | DE
 45 | 2020-09-07 11:47:51 | 125         | 1597937078 | vtnet0         | match  | block  | in                |          4 | 45 |     | 248 | 5425    |      0 | none
| 6            | tcp            |     40 | 94.102.53.112 | 5.182.200.10   | 54264       | 47250            |             |        0 | S        | 1878484393   |      | 1024   | [
]                          | 0                    | 0                     | GB
(10 rows)
```

Figure 10: Example Join on ID in Citus

While the container is running, PSQL in Citus can be accessed using the following commands. The default password is: 'bda'

```
>> docker exec -it citus bash
```

```
>> psql -h localhost -p 5432 -U postgres -W
```

The data stored in Citus is persisted through a volume mount in the docker-compose file. The default storage location is `'./citus/data/:/var/lib/postgresql/data/'` (Figure 4).

4.5.1 Preloading Data

In case the pipeline should not be run through and only the analysis is of interest, preloaded data can be insert into the database. We uploaded a file called '`2021-05-08-Group-6-Log-Analysis-Data.zip`' on Frachtwerks storage account. The file has to be unzipped and contains two more archives called citus.zip and parquet.zip. The first has to be placed in the folder `'./docker/citus'`. Here the file must be unzipped, and the old data folder has to be deleted. Now only the database and the desired analytical service can be run with the command:

```
>> docker-compose citus <analytical service>
```

4.6 Grafana

Grafana is used to display the key results of the data analysis and pattern recognition. After the "`docker-compose up`" command (section 4.2 Docker) was run, the corresponding Grafana server can be accessed under http://localhost:3000 and the dashboard under

http://localhost:3000/d/JGhRWUBGz/2021-05-08-group-6-log-analysis?orgId=1 (Figure 11).

Figure 11 Grafana dashboard

The provisioning feature of Grafana enables the automatic initialization of predefined dashboards and datasources during startup (https://grafana.com/docs/grafana/latest/administration/provisioning/ (Grafana Labs 2021)). To edit and add dashboards or datasources navigate to "`./docker/grafana/provisioning/`".

Publication bibliography

Citusdata (2021): Citus. Citusdata. Available online at https://www.citusdata.com/, checked on 1/28/2021.

Docker (2021): Get Started Overview. Docker. Available online at https://docs.docker.com/get-started/overview/, checked on 1/28/2021.

Grafana Labs (2021): Grafana. Grafana Labs. Available online at https://grafana.com/, checked on 1/28/2021.