Contents

[**Installation** 2](#_Toc62481970)

[Docker 2](#_Toc62481971)

[Python 2](#_Toc62481972)

[Grafana 2](#_Toc62481973)

[**Execution** 2](#_Toc62481974)

[Docker 2](#_Toc62481975)

[Python 3](#_Toc62481976)

[**Prerequisites** 4](#_Toc62481977)

[Influx DB 4](#_Toc62481978)

[Prometheus 5](#_Toc62481979)

[Grafana 5](#_Toc62481980)

[**Architecture** 5](#_Toc62481981)

[**Autoencoder** 6](#_Toc62481982)

[Preprocessing 6](#_Toc62481983)

[Training 6](#_Toc62481984)

[Evaluation 7](#_Toc62481985)

[Testing 8](#_Toc62481986)

[**Grafana UI** 8](#_Toc62481987)

# **Installation**

## Docker

* Install Docker for your operating system (<https://docs.docker.com/get-docker/>)
* Build Docker container from Dockerfile (from directory, where Dockerfile is located):
  + docker build –t 5g\_anomaly\_detection –f Dockerfile .

## Python

* Tested with Python 3.7.0 and pip 10.0.1
* Required Python packages: pandas, numpy, scikit-learn, jsonschema, matplotlib, keras (2.3.1), tensorflow (2.2.1), tensorflow-cpu (2.2.1), h5py, influxdb.
* All required packages with tested versions can be installed from requirements.txt file using the command (from directory, where requirements.txt is located):
  + pip install –r requirements.txt

## Grafana

Grafana can be installed either locally (<https://grafana.com/docs/grafana/latest/installation/>) or within a Docker container (<https://hub.docker.com/r/grafana/grafana>). After installation, two data sources must be set. A Prometheus data source and an InfluxDB data source. The created dashboard can be imported from a json file (ui/threat-detection-ui.json).

# **Execution**

## Docker

* Run the docker container (in the background) with the following command:
  + docker run –d –env ${ENV\_PARAM\_NAME\_1}=${ENV\_PARAM\_VALUE\_1} –env … -p ${HOST\_PORT}:1234 5g\_anomaly\_detection
* Docker ENV params:
  + TRAIN\_MODEL: Train Model before starting it for predicting in real time. (Default: True)
  + EVAL\_MODEL: Evaluate Model in order to suggest some missing thresholds, that are not user defined. If all thresholds are set as ENV params this function can be avoid, because it will not override user-defined values. If EVAL\_MODEL is set to False and user has not set some thresholds they will be set with a default value equal to 0.1. (Default: True)
  + CPU\_TH: RMSE anomaly threshold for predicted CPU percentage rate in user mode. (Default: 0.1)
  + MEM\_TH: RMSE anomaly threshold for predicted RAM percentage rate. (Default: 0.1)
  + CPU\_RX\_TH: RMSE anomaly threshold for predicted RX CPU percentage rate. (Default: 0.1)
  + CPU\_TX\_TH: RMSE anomaly threshold for predicted TX CPU percentage rate. (Default: 0.1)
  + NET\_UP\_TH: RMSE anomaly threshold for predicted bytes transmitted rate for selected interfaces. (Default: 0.1)
  + NET\_DOWN\_TH: RMSE anomaly threshold for predicted bytes received rate for selected interfaces. (Default: 0.1)
  + NET\_5G\_UP\_TH: RMSE anomaly threshold for predicted bytes transmitted rate for 5G cell. (Default: 0.1)
  + NET\_5G\_DOWN\_TH: RMSE anomaly threshold for predicted bytes received rate for 5G cell. (Default: 0.1)
  + OVERALL\_TH: RMSE anomaly threshold for all predicted features aggregated. (Default: 0.1)
  + INFLUX\_HOST: IP of machine where InfluxDB is running. (Default: localhost)
  + INFLUX\_PORT: Port, where InfluxDB is listening. (Default: 8086)
  + INFLUX\_USER: Username for connecting in InfluxDB. (Default: admin)
  + INFLUX\_PASS: Password for connecting in InfluxDB. (Default: admin)
  + INFLUX\_DB: Database, where all collected metrics and detected anomalies are stored. (Default: metrics\_db)
  + INFLUX\_ANOMALIES\_MEASUREMENT: Measurement in ${INFLUX\_DB} where all detected anomalies will be saved. (Default: detected\_anomalies)
* Docker ports:
  + Port 1234: A simple python http server is running in order to provide docs for reading.

If there is no change in training process or in the training data, TRAIN\_MODEL param can be set to False, as the root folder contains a trained and evaluated model in the given data.

## Python

If the algorithm is executed from Python command line in testing mode, the file thresholds.json in data folder must exists. If there is no such file in the data folder, it must be created manually. Its structure is the following:

{"cpu\_threshold": 0.1, "mem\_threshold": 0.1, ...}

The required keys for thresholds.json file are the following: cpu\_threshold, mem\_threshold, cpu\_tx\_threshold, cpu\_rx\_threshold, net\_up\_threshold, net\_down\_threshold, net\_5g\_up\_threshold, net\_5g\_down\_threshold, overall\_threshold.

* Training Mode (from root folder, where find\_anomalies.py is located):

python find\_anomalies.py \

            --mode train \

            --model model/5g\_autoencoder.h5 \

            --evaluate false \

            --thresholds\_file data/thresholds.json

* Training Mode with evaluation (from root folder, where find\_anomalies.py is located):

python find\_anomalies.py \

            --mode train \

            --model model/5g\_autoencoder.h5 \

            --evaluate true \

            --thresholds\_file data/thresholds.json

* Testing Mode (from root folder, where find\_anomalies.py is located):

The following parameters must be set either as variables in shell or directly in the following command to execute the algorithm in test mode:

* + ${INFLUX\_HOST}: IP of machine where InfluxDB is running.
  + ${INFLUX\_PORT}: Port, where InfluxDB is listening.
  + ${INFLUX\_USER}: Username for connecting in InfluxDB.
  + ${INFLUX\_PASS}: Password for connecting in InfluxDB.
  + ${INFLUX\_DB}: Database, where all collected metrics and detected anomalies are stored.
  + ${INFLUX\_ANOMALIES\_MEASUREMENT}: Measurement in ${INFLUX\_DB} where all detected anomalies will be saved.

python find\_anomalies.py \

    --mode test \

    --model model/5g\_autoencoder.h5 \

    --thresholds\_file data/thresholds.json \

    --influx\_host ${INFLUX\_HOST} \

    --influx\_port ${INFLUX\_PORT} \

    --influx\_user ${INFLUX\_USER} \

    --influx\_pass ${INFLUX\_PASS} \

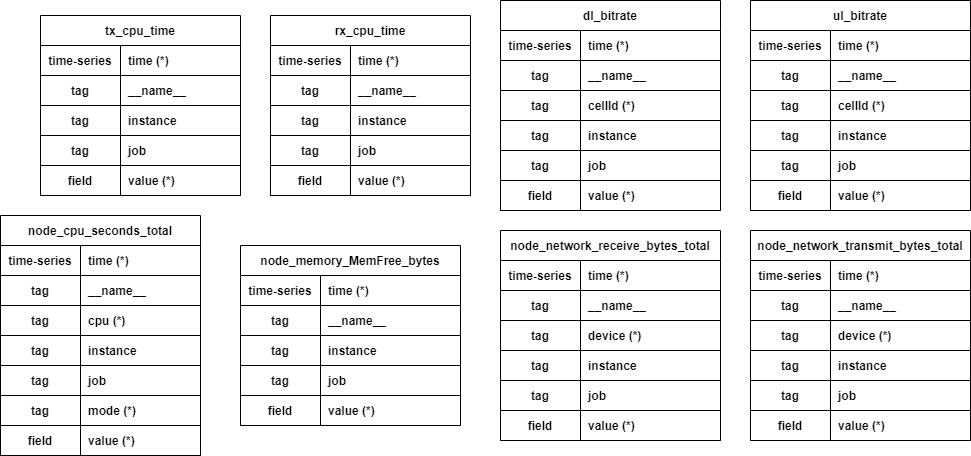
    --influx\_db ${INFLUX\_DB} \

    --influx\_measurement ${INFLUX\_ANOMALIES\_MEASUREMENT}

# **Prerequisites**

## Influx DB

Influx DB must has a specific schema for each table. Columns with star (**\***) are required to exist.



In addition, some columns in some measurements must have specific values as follows:

* node\_cpu\_seconds\_total:
  + cpu: must has values 0 and 1. These two are used for collecting and aggregating metrics.
* node\_network\_receive\_bytes\_total/ node\_network\_transmit\_bytes\_total:
  + device: enp1s0, enp0s20u1 and ppp0 are used for collecting metrics and aggregating received and transmitted bytes and their rate.
* ul\_bitrate/dl\_bitrate:
  + cellId: There are two cellIds: 1 and 2. cellId 1 is used for 4g connections and cellId 2 is used for 5g connections. The algorithms collects metrics only from cellId 2, which is used for 5g connections.

## Prometheus

Prometheus can be installed either standalone (<https://prometheus.io/docs/prometheus/latest/getting_started/>) or using Docker image (<https://hub.docker.com/r/prom/prometheus/>). Prometheus must be configured using .yml file (<https://prometheus.io/docs/prometheus/latest/configuration/configuration/>). In this configuration is set the url of the InfluxDB, where the collected metrics will be stored, scrape sources and metrics relabel.

An example of this config .yml file is the following:

# Remote write configuration for Influx

remote\_write:

- url: "http://{influx\_host}:{port}/api/v1/prom/write?db={db}&u={user}&p={pass}"

scrape\_configs:

- job\_name: ‘amari exporter’

scrape\_interval: 5s

static\_configs:

- targets: ["{job\_ip}:{job\_port}"]

- job\_name: 'node\_exporter\_dellEdge'

scrape\_interval: 15s

static\_configs:

- targets: ["{job\_ip}:{job\_port}"]

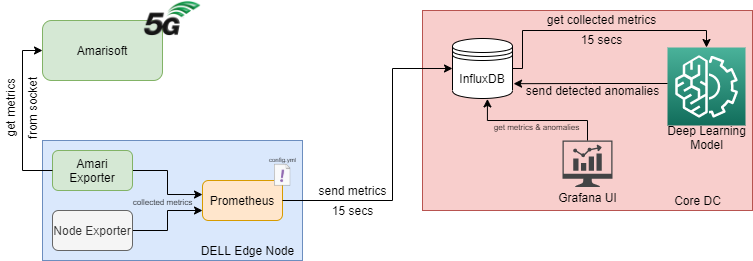
Prmotheus uses 2 data sources in order to collect metrics, Node Exporter (<https://prometheus.io/docs/guides/node-exporter/>) and Amari Exporter (<https://github.com/5genesis/Security-Framework/tree/main/amarisoft%20exporter>). Node Exporter collects OS metrics from Edge Machine. Amari Exporter collects metrics from Amarisoft relative to 5G.

## Grafana

Deployed Grafana UI uses both Prometheus and InfluxDB for showing collected metrics and detected anomalies. A Prometheus and an InfluxDB data sources must be set in Grafana before importing the dashboard. During import Grafana will ask to match the imported data sources from .json file to existing data sources in Grafana.

# **Architecture**

In the figure below the whole architecture of deployment is shown:



The are three main components (Amarisoft, Edge Node and Core DC). Amarisoft is the device that works as 5G transmitter. In Edge node Prometheus and two exporters are running. In Core DC an Influx DB is deployed. Also, the anomaly detection software and a provided UI through Grafana are deployed. Amari Exporter collets metrics from Amarisoft through network sockets. Node Exporter collects metrics from Edge Node. Both exporters are send collected metrics to Prometheus. These collected metrics are sent to Influx DB every 15 seconds. Anomaly Detection model fetches ingested data from Influx DB and decides if a record is an anomaly compared to normal records or not. Detected anomalies are saved to Influx DB. Grafana UI monitors some metrics from Edge Node and Amarisoft, and also has a table with all detected anomalies. For each anomaly there is the feature to see more info about collected and monitored metrics.

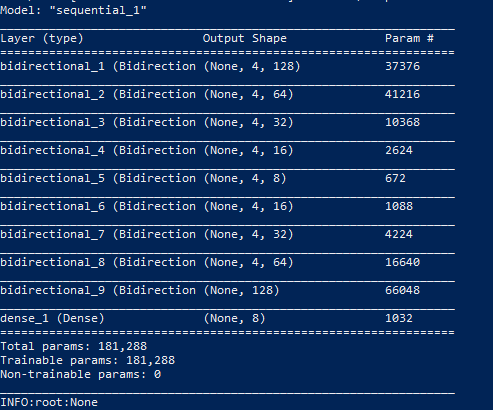
# **Autoencoder**

## Preprocessing

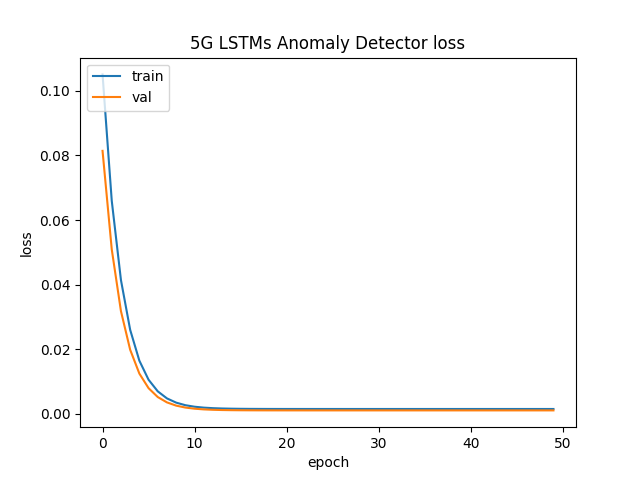
Training data have been collected from both Node Exporter and Amari Exporter. Because of these two different systems, there is a small difference in timestamps for these metrics. For this reason the training dataset is resampled every 15 seconds, to synchronize records from these two exporters. For training, the following features have been selected: CPU percentage rate, RAM percentage rate, RX/TX CPU percentage rate, rate of transmitted/received bytes and rate of bytes downloaded/uploaded from 5G interface. These features are normalized using Min-Max Normalization. The min & max values that will be used for normalization will be saved in a JSON file. After resampling and normalize the dataset is split in sequences, with four steps per sequence, where the first three records will be used to predict the fourth.

## Training

For training the architecture of an Autoencoder has been used. Model consisted of 9 Bidirectional LSTM layers and one Dense layer at the end.



As activation function ReLu is used. SGD with Nesterov momentum has been selected as optimizer, with learning rate equal to 0.01. As validation dataset 10% of the trainset has been used. The model is trained for 50 epochs.



## Evaluation

If evaluation is enbled, trained model will be loaded and used for testing in three different datasets. The first dataset has been collected during a CPU stress test attack. The second dataset has been collected during an iperf stress test attack. The third dataset is the training set. All these datasets will be normalized with saved normalization values during training. Using these datasets the RMSE between the actual and predicted values will be calculated. Network features’ thresholds, will not be affectd from CPU stress dataset and CPU feratures’ thresholds will not be affected from iperf stress test dataset. The 99th percentile of each feature’s RMSE will be considered as threshold from anomaly detection algorithm. The 99th percentile has been selected compred with the max value in order to avoid outliers, that may exist in these datasets. User-defined thresholds will not be overridden from calculated RMSEs.

## Testing

Trained model will be used in order to predict the next features’ values in the time series. Because it is trained in normal traffic it will predict these values considering that the incoming traffic is normal. If the RMSE between the predicted and actual value is above a set threshold this record will be considered as anomaly. Algorithm fetches the last recird from an InfluxDB every 15 seconds. For predicting the next value a sliding window, with size equal to 30, keeping only last records is used. Fetched records are been preprocessed and normalized before used by the model for predicting. After prediction the RMSEs between the actual and predicted values are calculated and compared with set thresholds, to identify if the record considers an anomaly or not. A possible cause of the anomaly is also saved in the database with the detected anomalies or an ‘unknown cause’ will be set if the cause cannot be identified. A proposed thresholds JSON file for testing is the follow:

{"cpu\_threshold": 0.05, "mem\_threshold": 0.1, "cpu\_tx\_threshold": 0.1,

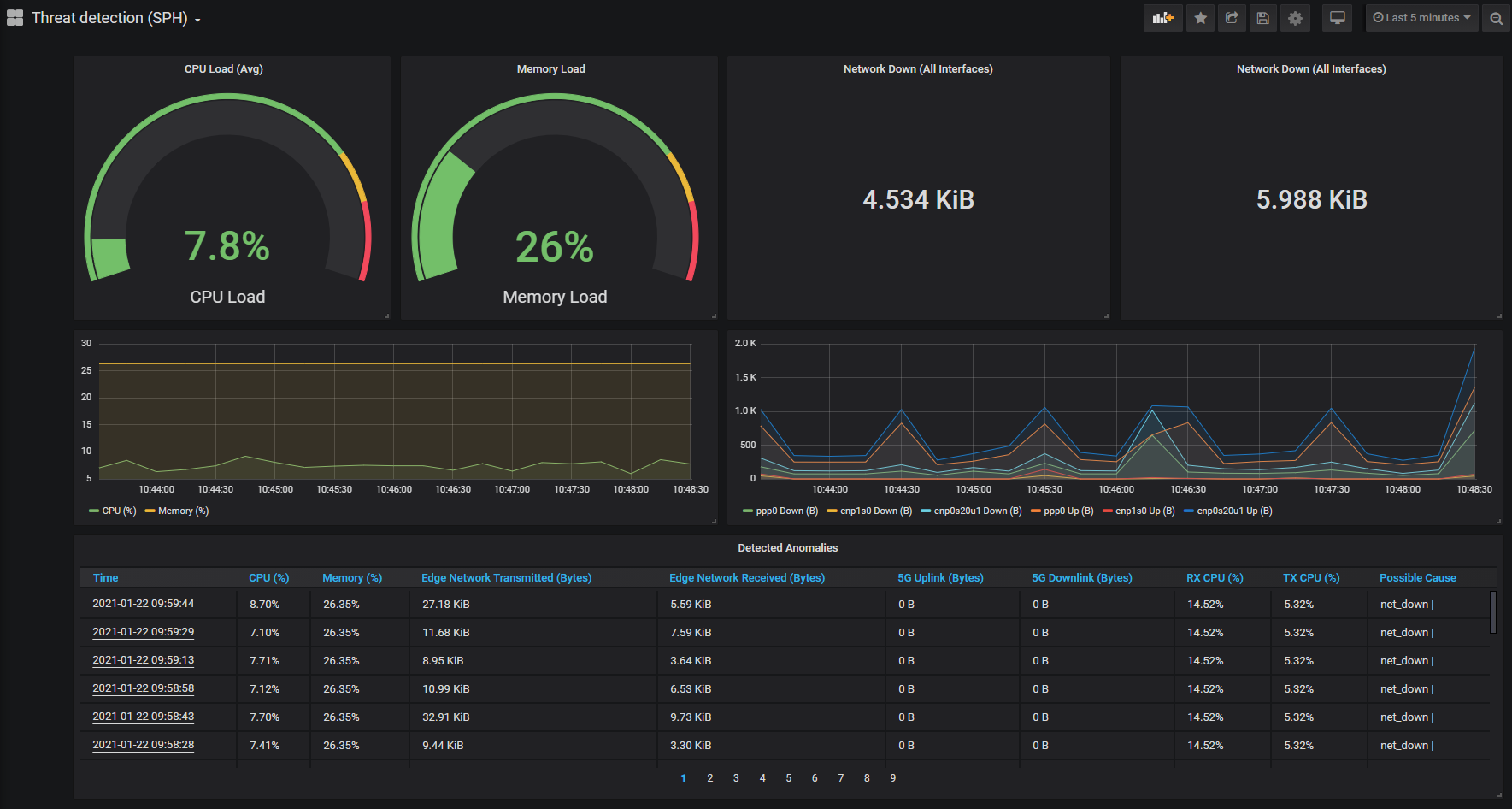
"cpu\_rx\_threshold": 0.1, "net\_up\_threshold": 0.501,

"net\_down\_threshold": 0.501, "net\_5g\_up\_threshold": 0.501,

"net\_5g\_down\_threshold": 0.501, "overall\_threshold": 0.15}

# **Grafana UI**

A Grafana UI is also provided for both monitoring some useful metrics and also list all detected anomalies. The provided UI is shown in the following image:



The first two panels in top row are monitoring the percentage usage for all CPU cores in average and RAM. The next two panels in the top row are monitoring the sum of all received and transmitted bytes for all network interfaces. In the second row there are two panels. The left one show the percentage of used CPU and RAM and the second one the network bytes that received and transmitted from three specific interfaces. Finally, in the bottom row there is the table with all detected anomalies. The table contains the time the anomaly detected and values for some features the time of anomaly, like CPU and RAM usage, network metrics and 5G metrics. Also, there is a message with possible causes if the cause of anomaly can be detected.