

HORIZON 2020**Information and Communication Technologies****Integrating experiments and facilities in FIRE+****Deliverable 2.3****First Report on Experiment Results and
Suggestions for Improvements****Grant Agreement number: 687992****Project acronym: EMBERS****Project title: Enabling a Mobility Back-End as a Robust Service****Type of action: Innovation Action (IA)****Project website address: www.embers-project.eu****Due date of deliverable: 2017-03-31**

Dissemination Level		
PU	Public	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document properties

Leader partner	Technische Universität Berlin
Author(s)/editor(s)	Daniel Nehls (TUB), Francisco Cardoso (UW), André Duarte (UW), Olivier Fambon (INRIA), Christian Klopp (Fraunhofer)
Version	Version 1.0

Abstract

This deliverable follows up on D2.2 - "First Report on Experiment Design and Description". It aims at summarizing the results of the experiments undertaken so far (section 2) as well as introducing the next steps planned in experimentation to achieve the project's objectives such as supporting UBIWHERE in maturing the MBaaS platform. Therefore, section 3 depicts potential improvements in the experiment design based on results and identified KPIs, and defines additional experiments to be performed in the upcoming phase of the project.

Table of Contents

1	Introduction	4
2	Experiment Description and Results	5
2.1	Simple Broker Connection Experiment	5
2.2	Load Generator Experiments	6
2.3	FIT-IoT Lab Tools Experiments	15
2.3.1	Hackathon Tool wrap-up	15
2.3.2	Open-A8 Tooling	16
3	Experiment Improvements and Future	19
3.1	Experiment 1 / Broker Connection	20
3.2	Experiment 2 / FIT-IoT LAB Tools	20
3.3	Experiment 3 / Load Generator	20
3.4	Testing Infrastructure improvement	21
3.5	Future Experimentation	21
4	Conclusion	23
5	Appendix	24
5.1	FIT - IoT Lab Experiments	24
5.1.1	Iteration A - Traffic	24
5.1.2	Iteration B - Traffic	26
5.2	Load Generator Experiments	29
5.2.1	Iteration A - Traffic	29
5.2.2	Iteration A - Parking	34
5.2.3	Iteration A - Environment	38
5.2.4	Iteration B - Traffic	42
5.2.5	Iteration B - Parking	47
5.2.6	Iteration B - Environment	51
6	References	55

1 Introduction

The experiments described in the precedent deliverable D2.2 were meant to represent the first wave of tests designed and executed on the EMBERS platform.

These experiments were designed with two main objectives of the project in mind; to bring the MBaaS platform to market and contribute back to the involved FIRE+ [1] infrastructures. Following this agile approach, this deliverable summarizes the results of the first tests and describes their evolution.

The scope of these experiments is to show the integration of the platform and the work done until now. The deliverable comes right after a successful first event, the Helsinki Hackathon, which was the first real test to the platform.

All in all, this deliverable will be the first building block to further, more complex, experimentation which will enable the MBaaS to successfully reach its objectives.

2 Experiment Description and Results

This section covers the description and results of the experiments designed in the D2.2. The main purpose of these experiments is to test the MBaaS architecture at two levels: firstly, a simple broker connection to test the backend communication and in second to test the backend using the INRIA testbeds and the Fraunhofer FOKUS Load generator in order to create more realistic load scenarios. These load scenarios allow to test the MBaaS against different types of load and identify possible failures.

2.1 Simple Broker Connection Experiment

This experiment is designed as a broker connection validation test and covers the integration requirements described in D1.1 (sections 2.1.1 / protocol, and 2.3.1 / broker). For the purpose of this deliverable, we evaluated the Meshblu broker connections over HTTP.

Objectives

- Check that a sample Backend client (sensor/device, southbound interface) can register and connect to the MBaaS over HTTP
- Check that a sample Backend client can send events to the broker over HTTP
- Check that a sample Frontend client (application, MBaaS client, northbound interface) can connect to the MBaaS and receive events

Experiments

- The hackathon IoT-Lab ‘injectors’ demonstrated this (and did send events)
- The hackathon ‘Load Generator’ demonstrated this (and did send events)

Results

- The backend devices connection and event sending was successful
- The event receiving was successful, as seen via the apiary-listed API endpoints (northbound interface, http “paginated” events list)

2.2 Load Generator Experiments

This section describes how the Fraunhofer FOKUS Load Generator was used in order to test the MBaaS against different types of load scenarios, with the main purpose of getting detailed information about possible failures of the MBaaS overall architecture.

Objectives

The main purpose of this experiment is to test the MBaaS against different load scenarios and to achieve that, were used three main tools: the Load Generator from Fraunhofer FOKUS, an automation tool and a time series analytics tool from UBIWHERE, that will be shortly described next:

- **Load Generator:** This tool allows creating load scenarios against the MBaaS according to predefined specifications. It allows to easily create experiments that will send specific data to the MBaaS, simulating the data that is used in real use case scenarios.

The experiments can be configured with a variety of parameters like: duration, interval of status updates, device broker specific parameters (e.g. host, port, protocol, special headers or payloads). Especially, it is possible to define multiple sensor types with freely configurable payloads over time. This allows very specific tests. Besides the possibility to configure all those payloads manually, there are generator mechanisms to compute the different payloads automatically. This way unnecessary configuration effort can be avoided, especially in regard to tests with a huge number of emulated sensors.

An instance of the Load Generator is integrated within the FUSECO Playground at Fraunhofer FOKUS and accessible from the outside. Therefore, it is possible to run experiments without the effort to set up a running instance yourself. Additionally, it is possible to easily deploy an own instance of the Load Generator via Docker.

- **Automation tool:** This tool was created to enable easy replication of different load scenarios against the MBaaS. To do that this tool interacts with the Load Generator, which will launch experiments against the MBaaS. The automation tool is configured by different parameters that are needed by the Load Generator specifying the experiment: username, experiment name, experiment type (parking, traffic, and pollution), the number of sensors, experiment duration and interval between messages. Using these parameters, the automation tool will interact with the Load Generator in order to launch the experiment and will collect data regarding the experiment.

This tool is separated in two main services: one responsible for the interaction with the Load Generator (named performance) that will launch the experiments and other (named listener)

responsible for connecting to the MBaaS and control the data that is received from the Load Generator.

- **Time series analytics tool:** This tool allows analysing the MBaaS microservices-based architecture. Since the backend is built using Docker, is necessary to have a tool that allows to control and analyse the impact that each Docker service has. To achieve that, this tool was used, which allows collecting metrics regarding each service about CPU, RAM, Disk I/O and Network I/O usage.

With this metrics, it is possible to identify the impact that each service has in the MBaaS and possible problems in the architecture.

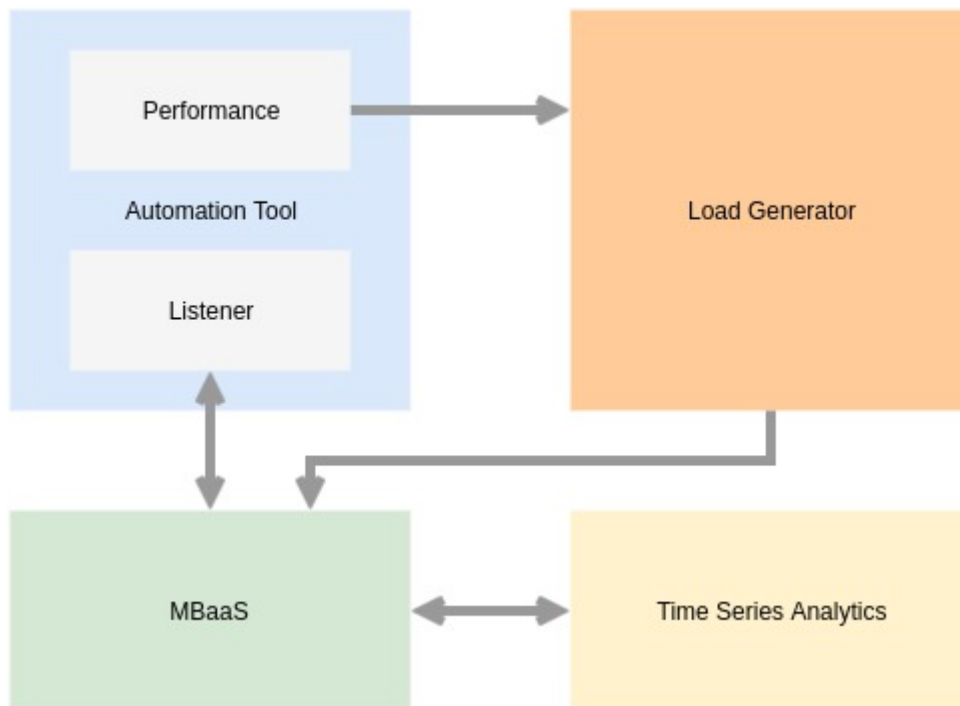


Figure 1: Tools Interaction

The image above represents how the three tools interact in order to reproduce load scenarios against the MBaaS and collect useful metrics in order to analyse the MBaaS behaviour.

After each experiment launched with this set of tools, it is possible to have a time series analysis about CPU, RAM, Disk I/O and Network I/O regarding each service. Using the time series analysis tool, it is possible to have information about the succeeded/failed steps taken by the Load Generator, detailed information about the experiment launched by the automation tool (experiment type, number of sensors, experiment duration, interval between messages, experiment start time and experiment finish time), detailed information about the experiment launched (registered sensors, messages sent, messages failed, experiment start time and experiment finish time) and information regarding the data that was received in MBaaS (expected received messages and number of received messages).

Experiments

The following table shows the two iterations planned to accomplish the experiment.

Both iterations aim to test the MBaaS against the three types of data: parking, environment, and pollution, with different numbers of sensors (100 and 300) and with different interval between messages (60 and 30 seconds).

This way the iterations represent an incremental test were firstly the MBaaS is tested against a small group of sensors sending messages between 60 and 30 seconds at the same time, and in the second iteration the MBaaS is tested against a bigger group of sensors (300) sending messages between the same interval as the previous iteration at the same time. Both iterations have the same duration (30 minutes).

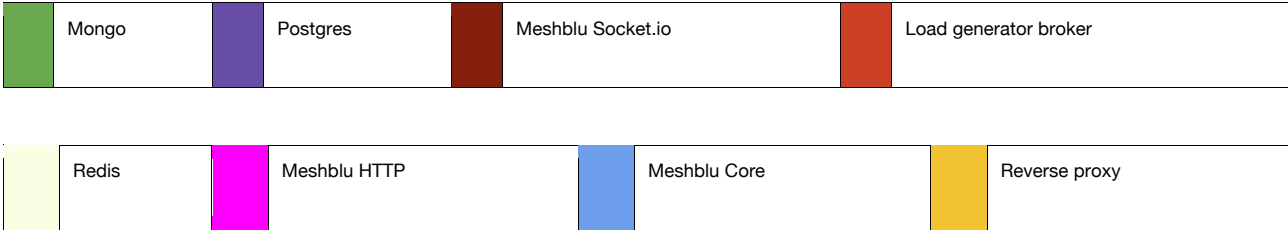
Iteration	Type	Number of devices	Duration (seconds)	Messages interval
A	Traffic	100	1800	60
				30
	Parking	100	1800	60
				30
	Environment	100	1800	60
				30
B	Traffic	300	1800	60
				30
	Parking	300	1800	60
				30
	Environment	300	1800	60
				30

During iterations information will be collected using the automation tool and the time series analytics tool. This way it will be possible to compare how the variation of the two main factors (number of

sensors and interval between messages) that can influence the system behaviour will influence the MBaaS microservice-based architecture.

Before each of the iterations it is important to set the MBaaS baseline regarding CPU, RAM, Disk I/O and Network I/O that each service consumes. Regarding this metrics, the following charts represent what each service consumes using the time series analytics tool. This baseline represents the fully functional complete MBaaS deployment without any activity.

For each one of the following charts the colours represent the MBaaS services:



- Mongo:** responsible for the database used by Meshblu to maintain the registered sensors;
- Redis:** responsible for the database used by Meshblu to maintain the received messages;
- Postgres:** responsible for main database of the MBaaS;
- Meshblu HTTP:** receives the HTTP communication to Meshblu;
- Meshblu socket.io:** receives Websocket communication to Meshblu;
- Meshblu core:** coordinates all the Meshblu containers (in this case, Meshblu HTTP and Meshblu Socket.io containers);
- Load generator broker:** responsible to collect the messages that arrive at Meshblu;
- Reverse proxy:** responsible to receive all the incoming communication to the MBaaS and redirect them to the correct service/container.

CPU usage by container

Represent the usage by container of the host machine CPU in percentage (0-100%)

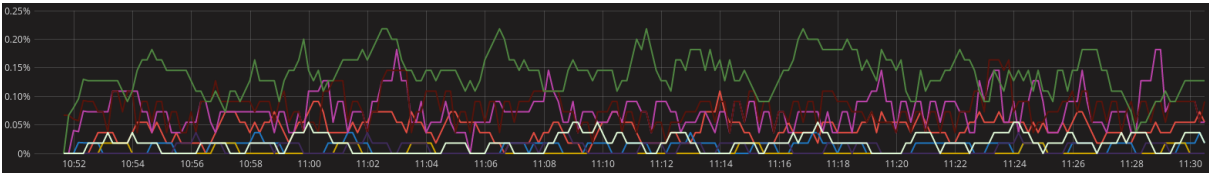


Figure 2: CPU usage baseline by container (0-100%)

Memory RAM usage by container

Represent the usage by container of the host machine memory RAM in Megabytes

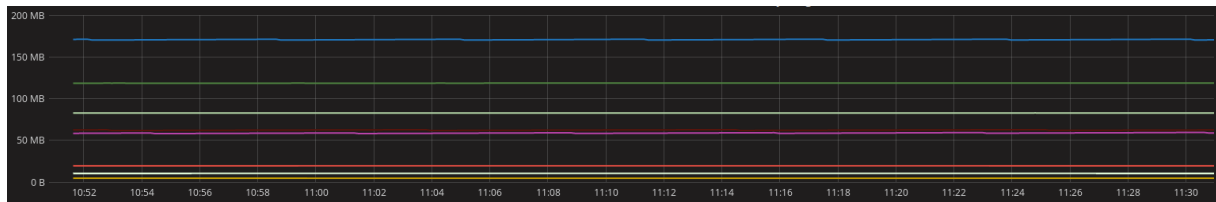


Figure 3:Memory RAM usage baseline by container (MB)

Disk input/output by container

Represent the total disk input and output for each container in Megabytes

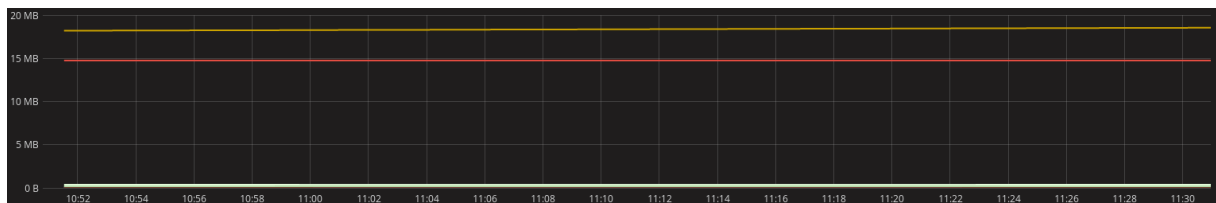


Figure 4: Disk I/O baseline by container (MB)

Network output by container

Represent each container network output in Megabytes



Figure 5: Network output baseline by container (MB)

Network input by container

Represent each container network input in Megabytes



Figure 6: - Network input baseline by container (MB)

It is important to refer that in order to collect the time series analytics regarding each service it is necessary to have two additional services running in the MBaaS. These two services are responsible for collecting metrics about the host machine and each one of the services. Since these two must be running on the same host as the MBaaS and they are consuming resources these two will influence the MBaaS performance.

The automation tool described above, will be used to interact with the Load Generator and collect some metrics regarding the experiment.

```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-20 12:19:44.602392

[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to performance, listener
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener     | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
listener     | Downloading socketIO-client-0.7.2.tar.gz
performance | Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
listener     | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener     | Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
performance | Successfully installed requests-2.7.0
listener     | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener     | Downloading six-1.10.0-py2.py3-none-any.whl
listener     | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener     | Downloading websocket-client-0.40.0.tar.gz (196kB)
listener     | Installing collected packages: requests, six, websocket-client, socketIO-client
listener     | Running setup.py install for websocket-client: started
listener     | Running setup.py install for websocket-client: finished with status 'done'
listener     | Running setup.py install for socketIO-client: started
listener     | Running setup.py install for socketIO-client: finished with status 'done'
listener     | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener     |
listener     | Listener Process:
listener     |
listener     | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener     |
listener     | Listener results:
listener     |
listener     | Expected received messages: 3000.0
listener     |
listener     | Received messages: 3000
listener     |
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: parking
performance | Number of sensors: 100
performance | Experiment duration: 1800s
performance | Messages interval: 60s
performance |
performance | Start Time: 2017-03-20 12:20:51.157268
performance | Finish Time: 2017-03-20 12:51:51.643546
performance | Elapsed Time: 0:31:00.486278
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 100
performance | Unregistered sensors: 100
performance | Registration ended: 2017-03-20 12:20:51.153902
performance | Unregistration ended: 2017-03-20 12:51:46.641152
performance | Messages sent: 3000
performance | Messages failed: 0
performance | Experiment start: 2017-03-20 12:19:49.463852
performance | Experiment end: 2017-03-20 12:50:44.857478
performance |
performance | performance exited with code 0
performance | listener exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done

Stopping performance test at 2017-03-20 12:51:53.290681

Performance test duration: 0:32:08.688289

Done.
```

Figure 7: Automation tool example result

In the results of the automation tool, as we can see in Figure 7, it is possible to collect data regarding the automation tool itself, the launcher and listener services of the automation tool, the Load Generator and the MBaaS:

1. **Automation tool process:** The automation tool starts by creating the two services (performance and listener) and everything needed by each one, like requirements and environment variables.
2. **Listener process:** The listener service is responsible for connecting to a predefined gateway and receives all the messages sent to it. This way is possible to compare the expected received messages and the number of messages that actually arrived at the MBaaS and consequently to the listener service.
3. **Performance process and results:** The performance service of the automation tool is responsible for interacting with the Load Generator to launch the experiments. In the process of this service, it is possible to recognize what interactions were done with the Load Generator and if they succeeded or failed. Regarding the results, it is possible to collect the following metrics:
 - **Experiment type:** type of data that the experiment sent to the MBaaS;
 - **Number of sensors:** number of sensors that sent messages to the MBaaS;
 - **Experiment duration:** time that the experiment was running;
 - **Messages interval:** interval between messages sent by sensors to the MBaaS;
 - **Start time:** time that the load generator started the experiment (sensors registration not included);
 - **Finish Time:** time that the load generator finished the experiment (sensors unregistration not included);
 - **Elapsed time:** Time that the experiment was running (finish time - start time).
4. **Load generator results:** These results are exported by the Load Generator itself after completing each experiment and provide the following metrics:
 - **Registered sensors:** number of sensors registered before the experiment;
 - **Unregistered sensors:** number of sensors unregistered after the experiment;
 - **Registration ended:** time when the sensors registration ended;
 - **Unregistration ended:** time when the sensors unregistration ended;
 - **Messages sent:** number of messages sent during the experiment;
 - **Messages failed:** number of failed messages during the experiment;
 - **Experiment start:** time when the experiment started;
 - **Experiment ended:** time when the experiment ended.

To understand each one of the baseline charts and the results collected from the experiment it is important to know the host machine characteristics:

CPU	Memory RAM	Disk Size	Disk IOPS Limit	Operative System
1 Core	3.5 GB	30 GB	3200	Ubuntu 16.04

This machine host was created with this characteristic specifically for tests purposes in order to separate the production environment from the test's environment. This way it is possible to have a clean deployment that can be used for tests without any impact on the production environment.

By analysing the baseline times series charts for the machine and accordingly its characteristics it is possible to say that, in general, all the observed containers/services consume a very small part of the machine resources when there is no activity. This small resource consumption is a result of the microservices-based architecture that allows that all the MbaaS is built in small services that work individually and communicate with each other in order to work as one single piece.

Due to the cloud provider that was used to host the MBaaS, the machine characteristics needed to be changed (from 2GB to 3.5GB of RAM) to match the available options.

Results

The following table represents the results obtained for each experiment regarding the number of messages sent from the load generator and received at the MBaaS and the experiment duration. These two parameters are important to measure, because they represent the success or failure of the experiment. The experiment duration refers to all steps of the automation tool process. So, this time is more than actually the time that the experiment was running because it includes intermediate steps like automation tool services preparation, sensors registration and unregistration by the Load Generator and the time needed to interact the load generator to launch the experiments.

Iteration	Type	Devices	Duration (s)	Messages interval	Sent/Received messages	Duration
A	Traffic	100	1800	60	3000/3000	0:32:12
				30	6000/6000	0:32:15
	Parking	100	1800	60	3000/3000	0:32:08
				30	6000/6000	0:32:14
	Environment	100	1800	60	3000/3000	0:32:10

				30	6000/6000	0:32:17
B	Traffic	300	1800	60	9000/9000	0:36:12
				30	18000/18000	0:36:17
	Parking	300	1800	60	9000/9000	0:36:07
				30	18000/18000	0:36:16
	Environment	300	1800	60	9000/9000	0:36:24
				30	18000/18000	0:36:33

Regarding the automation tool and the time series analytics, the results obtained for each iteration can be found in the appendix (5.2). The number of sent/received messages and duration presented in the previous table were collected from the automation tool results.

The main purpose of these experiments was to test the MBaaS against three types of data: traffic, parking and environment and check the behaviour of each MBaaS service. Accordingly, after analysing the table, it is possible to confirm that all the iterations/experiments were successful since all data sent from the Load Generator arrived successfully at the MBaaS as it was supposed.

Regarding the duration of the experiment, the obtained times are always bigger than the set experiment duration. This is because this time includes intermediate steps that were already referred, with sensor registration before an experiment and the sensor unregistration after the experiment being the biggest influences as each group of 100 sensors needs 1 minute to be registered and unregistered at the MBaaS.

Regarding the time series analytics tool results, it is possible to say that the data type didn't make any difference in the collected results, leaving the number of sensors and the interval between messages as variables. These two variables influenced the device broker in different ways. The number of sensors was responsible for the initial resources consumption, especially CPU. This means that the difference between 100 and 300 sensors is the initial resources consumption needed to register all sensors. The difference in the messages interval from 60 to 30 seconds is that the sensors communicate more times which will force the device broker to receive and process more messages. Therefore, it will consume more resources to dispatch the number of messages arriving.

Briefly, comparing with the baseline metrics, a high number of sensors result in an increasing consumption of resources (especially CPU and Disk) and network input and output since the number of requests for the sensors registration is bigger.

Reducing the interval between messages (from 60 to 30) has almost the same effect as increasing

the number of sensors. The CPU consumption increases since there will be more messages in short period of time and the network input and output increases for the same reason.

Regarding the disk I/O, this will increase from the first experiment to the last one since it is cumulative. The RAM usage doesn't have bigger variations. This is a favourable point to the MBaaS since it is able to maintain the RAM (that is the most critical resource) consumption independently from the experiment.

2.3 FIT-IoT Lab Tools Experiments

2.3.1 Hackathon Tool wrap-up

Although not foreseen originally as a part of the testing plan, the Hackathon tool was in effect the first realistic use-case of the MBaaS and covered most of the testing requirements defined in D2.2. The tool was used with success back in December to send Citypulse [2] datasets events (parking, traffic, pollution) to the MBaaS via Meshblu [3] over HTTP in quasi-real-time. The experiment was setup to send 1 event per minute for each of the 180 devices involved (60 devices x 3 datasets) over 2 days.

Overall, the system worked as expected during two days, with a total of 180 IoT-LAB M3 nodes sending data to Meshblu via a single gateway implemented on the IoT-LAB SSH frontend. Events we published both on a “paginated” http interface and on a real-time publish-subscribe interface. Hackathon users successfully used the published events.

Below is a quick recap of the experiment.

Objectives:

- Register a fixed set of devices to the Meshblu broker
- Setup metadata (GPS coordinates, names, etc.) for devices when registering
- Use fixed pre-configured/pre-registered “gateway” devices, one per data type (parking, traffic, pollution) as targets for sending events (gateways registered by UW)
- Send events to gateways
- Assert events are received on the northbound interface (http “paginated” events list)

Experiments:

- Run 180 nodes + 1 “frontend” (60 injectors nodes x 3 datasets) for 2 days
- Manual monitoring of events flow (no report) via the HTTP “paginated” interface

Results:

- It worked seamlessly for the 2 days
- Participants successfully used published events

2.3.2 Open-A8 Tooling

For the purpose of this deliverable and to further extend the use of IoT-LAB as a “real devices” provider for the MBaaS, we developed a new set of tools targeted at the IoT-LAB A8 nodes [4]. The IoT-LAB A8 nodes are small ARM-based embedded Linux machines, typically used as IoT “gateways” or for “big sensors” applications. IoT-LAB offers 200+ such devices in Grenoble and another 150+ in Saclay [5].

The tools are designed to be deployed and run on the IoT-LAB A8 nodes, but may be used on any Linux machine that is able to interpret Python. The source code for the tools is available on GitHub [6]. The toolset is built around two sub-components: a simple multi-protocol “meshblu-clients” library and a data-sources abstraction layer. The multi-protocol library provides for devices registration via the Meshblu HTTP API and for events sending to Meshblu over HTTP, MQTT [7] and CoAP [8].

The toolset is composed of the three high-level command-line tools outlined below:

- the ‘registry’ tool: register/unregister/list meshblu devices
- the ‘injectors’ tool: send events to “gateway” device (e.g. from an IoT-lab A8 node)
- the ‘deploy’ tool: automate the IoT-LAB experiment: reserve nodes, deploy and run injectors

The ‘registry’ tool is used to initialize the test session to Meshblu: we register a first “authentication” device which is required for most subsequent requests to the broker. The ‘injectors’ tool then directly registers devices as needed and unregisters them upon completion.

Each test session simply consists in running the ‘deploy’ tool. This results in allocating a set of IoT-LAB A8 nodes, deploying the ‘injectors’ tool on allocated nodes, running the ‘injectors’ tool in parallel on all nodes, and collecting summary outputs. Each ‘injectors’ tool performs the sequence: register devices, send event messages to Meshblu at specified rate for specified duration, unregister devices.

Objectives

The objective of this round of experiments is to validate that the MBaaS can handle messages sent by IoT-LAB devices over HTTP. The test scenarios are a subset of the scenarios described in section “Load Generator”. To validate that messages are received as expected, we used the same subscriber tool. For this round of tests, we used a single IoT-LAB A8 node, running a single injector.

The injector parameters were set to:

```
--protocol http           (for all test iterations)
--events traffic          (for all test iterations)
--nb-devices 100 or 300   (iterations A and B respectively)
--ev-per-hour 120 or 60  (iteration A+B, case 1 and 2 respectively)
--duration 30             (for all test iterations)
```


Experiments

Iteration	Type	Number of devices	Duration (seconds)	Messages interval
A	Traffic	100	1800	60
				30
B	Traffic	300	1800	60
				30

For both iterations, we collected server-side information using the time series analytics tool and checked that sent messages were received by comparing counts. The results are shown in the next section.

Results

Iteration	Type	Devices	Duration (s)	Messages interval	Sent/Received messages	Injection Duration
A	Traffic	100	1800	60	3000/3000	0:29:09
				30	6000/6000	0:29:38
B	Traffic	300	1800	60	9000/9000	0:29:25
				30	18000/18000	0:29:53

The detailed results regarding the time series analytics tool can be found in the appendix 5.1.

Overall, the four experiments went seamlessly. Disk IO is not impacted by the tests. Meshblu RAM usage is stable throughout the tests, after the expected initial increase due to the load (new devices, first messages). CPU load is caused by Meshblu only and moving in conjunction with incoming messages as expected. Network IO is caused by Meshblu and Redis and also moving with incoming messages.

The following set of images represent the automation process to start the experiments.

```
(emblers-dev) fambon@orval:~/dev/iotlab-injectors$ ./deploy.sh 1 100
start_experiment          [done]
wait_for_boot             [done]
deploy_injectors          [done]
run_injectors             0

2017-03-22 16:25:34 === starting ====
+ NB_NODES=1
+ DURATION=100
+ BROKER=msg.test.emblers.city
+ IOTLAB_SITE=grenoble
2017-03-22 16:25:34 === start_experiment
2017-03-22 16:25:55 === wait_for_boot
2017-03-22 16:26:34 === deploy_injectors
2017-03-22 16:27:47 === run_injectors
running 100 'traffic+http' injectors on local node
sending 120 events/h (per injector) for 30.0 min.
total: 6000 events sent in 1778.4 sec.
2017-03-22 16:58:11 === complete ===

BROKER="msg.test.emblers.city"

run_injectors() {
    for node in $nodes; do
        ssh $node injectors --run \
            --protocol http \
            --nb-devices 100 \
            --duration 30 \
            --events traffic \
            --ev-per-hour 120 \
            &
    done
    wait
}
~
"deploy.local.sh" 16 lines --6%--          1,1          All

Listener Process:
Listening to Gateway 7ec71ffa-7168-4b78-b767-d851512959ee
Listener results:
Expected received messages: 6000
Received messages: 6000
```

Figure 8: Experiment Automation process

3 Experiment Improvements and Future

This section aims to describe new scenarios for further developments in the MBaaS testing. Therefore, use cases that will be covered to fulfil more requirements of the project will be described.

It is important to refer that the to-be-tested scenarios need to be the most approximated with reality, with the intent of bridging the experiments to real-world scenarios.

The experiments have a baseline which represents the values that are being used in Ubiwhere deployments and reflect Ubiwhere's expertise on the field. They are a standard approach to every system that is deployed. Despite being baselines, these values might change depending on the customer and the scenario.

Traffic and parking use cases follow the following scenario: We have X sensors, and each one communicates with a gateway every time there is an update. The gateway sends the reading to the backend every time it receives an update from sensors or every Y minutes (keep alive mode). For now, one gateway is used for every 50 sensors. With this in mind, we aim to understand how the system handles multiple gateways sending data at the same time. This will allow the system to scale with the IoT infrastructure.

For the environment scenario, each station has its own gateway, which sends the messages to the backend. This use case also tests the multiple gateway scenario and the overall scalability of the backend.

These first tests intend to showcase simple connections and basic load testing on the server. Nevertheless, they need to be improved to provide a realistic enough use case for load testing. To improve the tests, we aim to increase the number of devices and the frequency of the messages. Beyond increasing the number of devices, it is also important to refer that different communication protocols (MQTT, CoAP), other open standards (LwM2M [9], OneM2M [10], NGSI [11]) and other device brokers (Ponte [12], FIWARE IoT Broker [13]) remain to be tested to understand their overall performance and see which would be the best choice given our use case.

It is important to clarify that not every standard is meant to work on every protocol, for example, LwM2M only supports CoAP. Nevertheless, it can be tested against other protocols.

Below the experiment evolution will be briefly documented by providing the next evolutionary step of each one. [11]

3.1 Experiment 1 / Broker Connection

This experiment was only meant to test the device broker connection and is validated by the other two. We haven't conducted any major testing since this was just to validate the connection. Therefore, it does not seem relevant to maintain this experiment in further testing, although the new brokers, protocols and standards will always need to pass this first test.

3.2 Experiment 2 / FIT-IoT LAB Tools

The FIT-IoT-LAB tools are ready for multiprotocol functional validation and performance testing of the Meshblu broker. However, they will likely require some evolutions, if only to integrate well with other tests for EMBERS or to adapt to a new broker such as Ponte. In any case, more MBaaS testing will be carried out using these tools.

The foreseen areas of future testing are threefold:

- 1.) scaling up the load, both in terms of number of devices and events frequency
- 2.) multi-protocol testing (MQTT, CoAP)
- 3.) testing realistic end-to-end IPv6 scenarios

The goal of the load tests would be to help find the tipping point for each broker/protocol using the IoT-LAB A8 nodes. These tests could probably be carried out using the Load Generator, in which case it would make little sense to replicate them on IoT-LAB. If it turns out that the Load Generator cannot "break" the brokers for lack of e.g. throughput, using IoT-LAB would then make sense.

The goal of the multiprotocol testing would be to provide functional validation and some performance comparison for the broker implementations of IoT protocols MQTT and CoAP.

The goal of the end-to-end IPv6 scenarios testing would be to validate the full chain from real devices to the broker and above, asserting the viability of the MBaaS in cases where unreliable, lossy radio-backed communications are involved in the mix. In addition to the planned CoAP scenario, real-life LwM2M implementations could be evaluated using the IoT-LAB M3 nodes.

3.3 Experiment 3 / Load Generator

It is planned to fully integrate the Load Generator with OpenMTC [14] in order to allow interoperability tests with the OneM2M standard. Furthermore, by using the capabilities of OpenMTC it will be possible to support other protocols like CoAP and MQTT.

Since OpenMTC is designed modular, different device brokers can be tested with the use of different OpenMTC Application Entity modules. Therefore, it is planned to provide the necessary modules for different device brokers like Ponte, FIWARE IoT Broker and Meshblu.

Additionally, it is planned to implement the possibility to configure multiple gateways for each experiment.

Finally, it is intended to allow the use of historical data sets in order to allow more realistic experiments.

3.4 Testing Infrastructure improvement

This subsection details the improvements to the overall infrastructure, by providing the next steps in the Developer Dashboard and broker deployment.

Regarding the developer dashboard and future improvements, two major improvements will be done. Firstly, it is intended to integrate with IoT-Lab in order to allow users to interact with the testbeds. This means that it will be possible from the developer dashboard to configure and start experiments in the IoT-Lab testbeds like it is done with the Load Generator. This way the developer dashboard will allow users to launch an experiment with emulated devices using the Load Generator and it will allow starting experiments using real devices and with real use case configurations.

Another improvement that will be done to the developer dashboard is the data catalogue. It aims to be a developer dashboard section where developers can share relevant datasets. The developer dashboard will not be used to host datasets it will only have the location of the dataset. This feature allows users not only to share datasets that they found but also to share their own datasets.

Lastly, it is important to refer that the new device brokers will be deployed in the infrastructure to start being tested. These device brokers are meant to be tested alongside Meshblu, the one which was tested in these experiments.

3.5 Future Experimentation

For future experiments, it is supposed to test the MBaaS against increased load. To achieve that, in the future another set of experiments using the Load Generator and the IoT-Lab injectors to test the MBaaS with increased values of load will be executed. Since the experiments referred in this deliverable were done with variations of the number of sensors (100 and 300 sensors) and an interval between messages (60 and 30 seconds), the future experiments will follow the same structure but with different values.

The experiments will have the number of sensors increased from 300 sensors (maximum in the experiments of this deliverable) to a bigger number of sensors and the messages interval will be reduced from 30 seconds (minimum in the experiments of this deliverable) to lower amounts of time. This is intended to test the overall platform and numbers are to be defined. The table below represents an example of the future experiments, in the future the values might change.

Iteration	Type	Number of devices	Duration (s)	Messages interval (s)
A	Traffic	1000	1800	60
				30
				15
	Parking	1000	1800	60
				30
				15
	Environment	1000	1800	60
				30
				15

So, with these tests, like the ones referred in this deliverable, data will be collected regarding the MBaaS host: CPU usage, memory RAM usage, Disk I/O, Network output and Network input by container/service.

These metrics will allow the MBaaS evaluation, identify possible points of failures and will be used as performance indicators in multiple use cases.

First of all, these will be used to compare the maximum number of messages per second that the different brokers can handle.

Also, it is to be tested, if the number of gateways (and devices) used in the experiments makes difference to the numbers that we are getting and if the size of the payload matters in the overall performance of the brokers.

Another test will compare the protocols MQTT and COAP with the tested HTTP.

Lastly, the tests made for this deliverable were solely for the southbound interface, in further developments the northbound interface will be included in order to understand the overall performance of the MBaaS as a full Mobility platform.

4 Conclusion

This deliverable summarized the results of the first wave of experiments executed in the scope of EMBERS.

The work described and concluded in the past months shows that the technical foundation of EMBERS is well advanced and the interworking with the two FIRE+ infrastructures provides valuable input for the MBaaS development. This is supported by the success the project had in the Hackathon and this first batch of experiments, which were meant to connect the platform to the FIRE+ testbeds and provide some basic insights on how performance will be measured during the project.

More experimentation is still ahead in order to fulfil all technical objectives of the project. Further deliverables will cover more technical aspects by providing more protocols, brokers and standards which remain to be tested.

At this point, we have seen that the MBaaS can handle small to medium amount of devices communication at low/medium frequencies. In the next tests that will cover by the next deliverables, more tests will be done with an increase in the number of devices and with higher frequencies update (interval between messages).

Given the experiments, it is possible to say that the MBaaS can be ran using a host with low characteristics.

All in all, the next iteration of experiments is expected to further support this foundation by integration of additional device brokers and M2M protocols to test interoperability capabilities.

5 Appendix

5.1 FiT - IoT Lab Experiments

This appendix section aims to show the results obtained for each iteration of the FIT-IoT Lab experiment. These results include screenshots of time series analytics tool.

For the time series analytics tool the colours for each chart represent the following services:



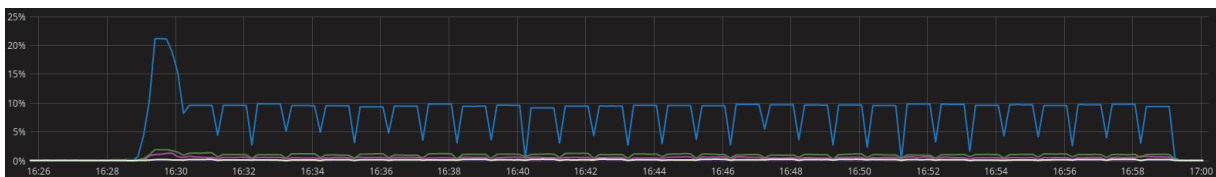
5.1.1 Iteration A - Traffic

This iteration was done in order to test the MBaaS against traffic data. For that purpose, this iteration used 100 sensors to communicate with the MBaaS during 30 minutes with an interval between messages of 60 and 30 seconds.

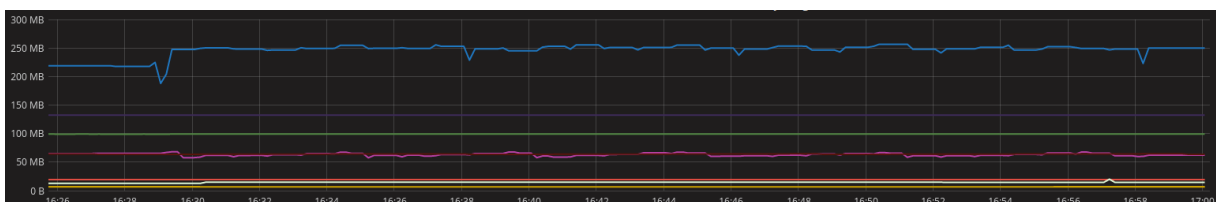
5.1.1.1 Interval between messages of 60 seconds

5.1.1.1.1 Time series analytics results

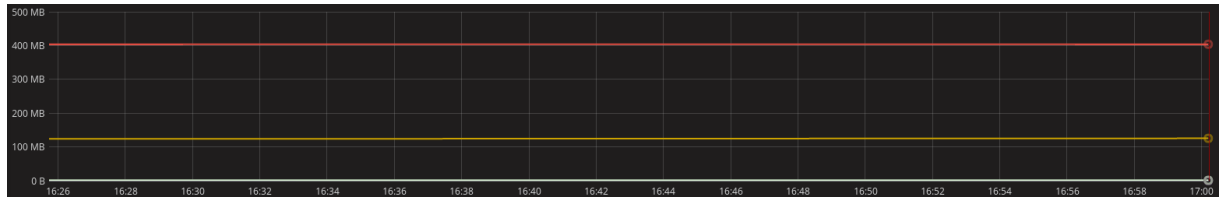
CPU usage by container



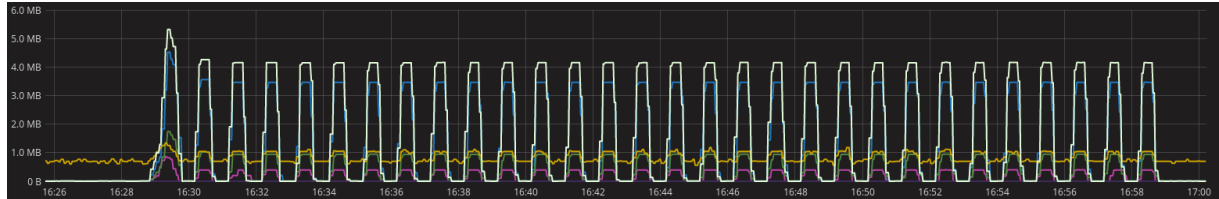
Memory RAM usage by container



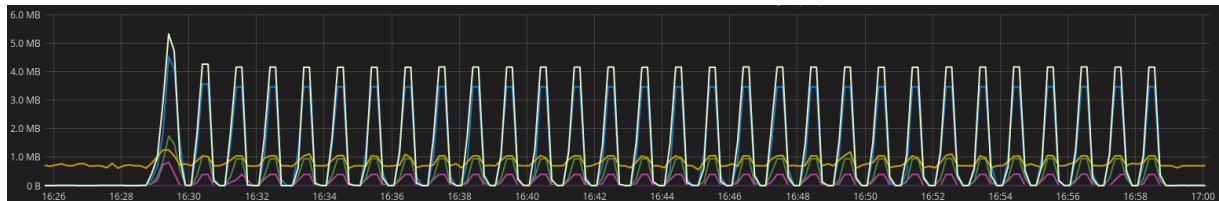
Disk input/output by container



Network output by container



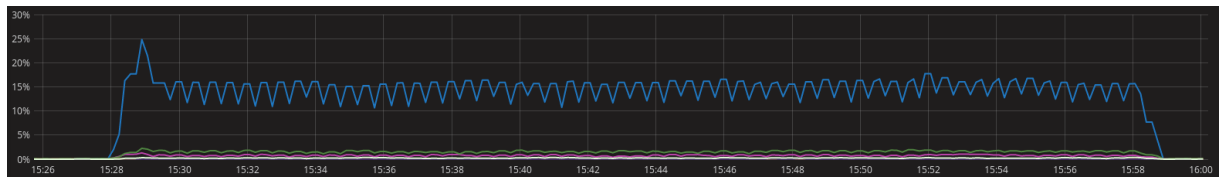
Network input by container



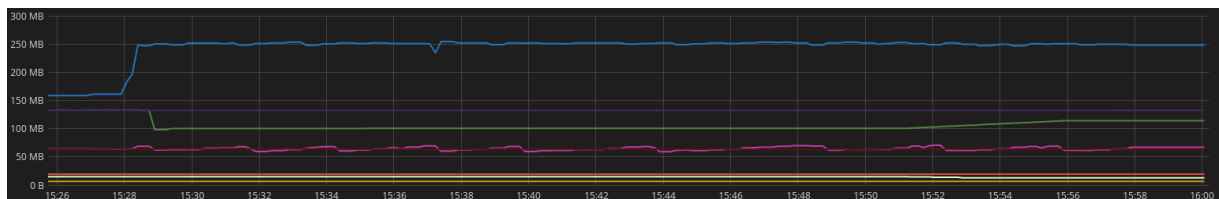
5.1.1.2 Interval between messages of 30 seconds

5.1.1.2.1 Time series analytics results

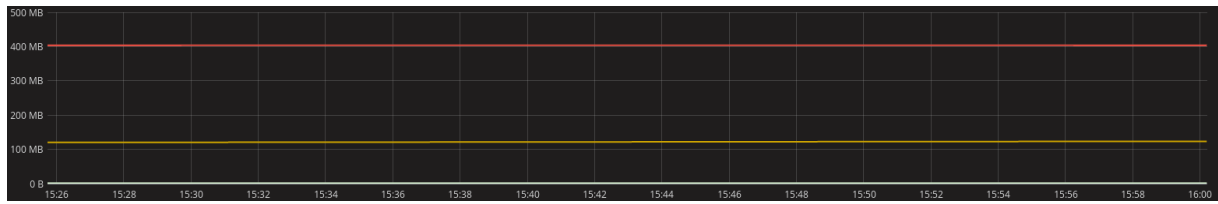
CPU usage by container



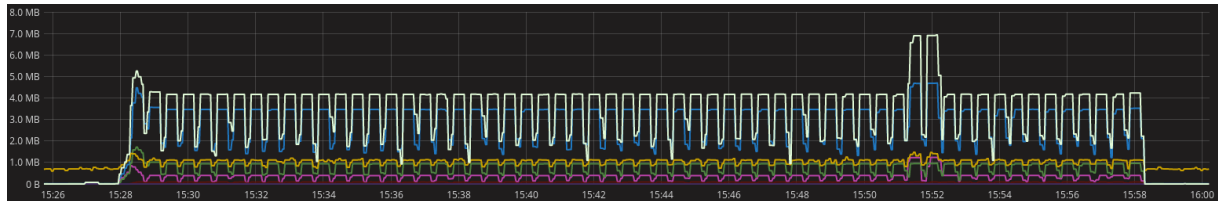
Memory RAM usage by container



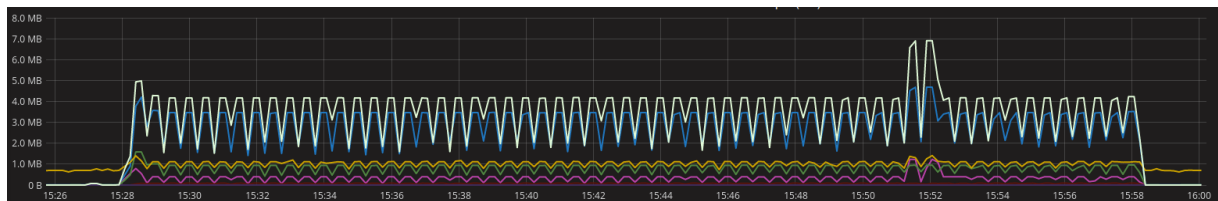
Disk input/output by container



Network output by container



Network input by container



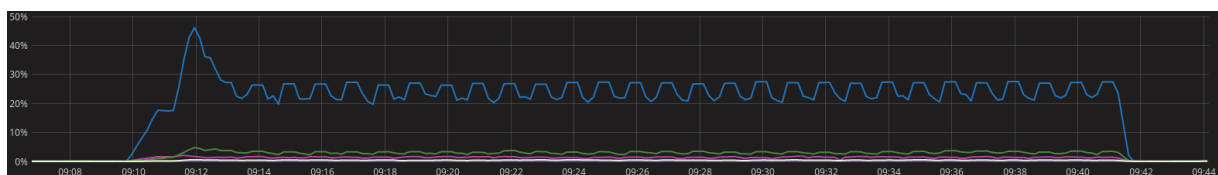
5.1.2 Iteration B - Traffic

This iteration was done in order to test the MBaaS against traffic data. For that purpose, this iteration used 300 sensors to communicate with the MBaaS during 30 minutes with an interval between messages of 60 and 30 seconds.

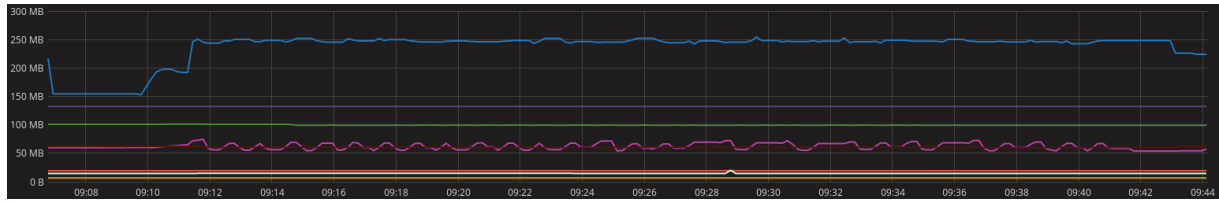
5.1.2.1 Interval between messages of 60 seconds

5.1.2.1.1 Time series analytics results

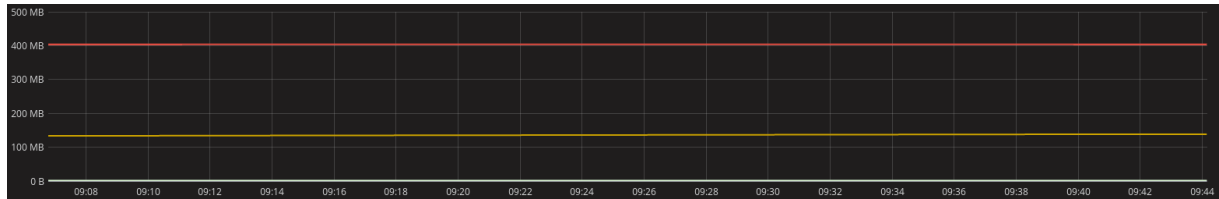
CPU usage by container



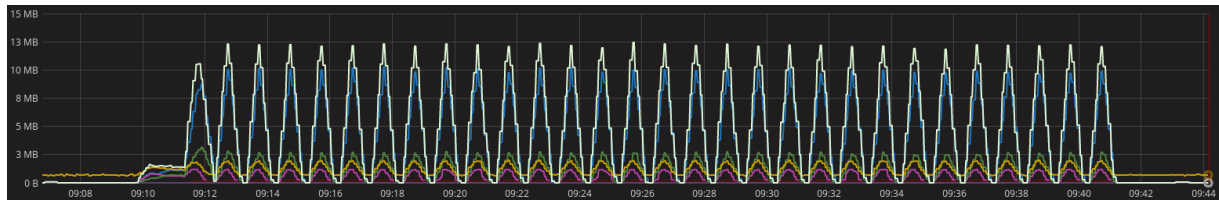
Memory RAM usage by container



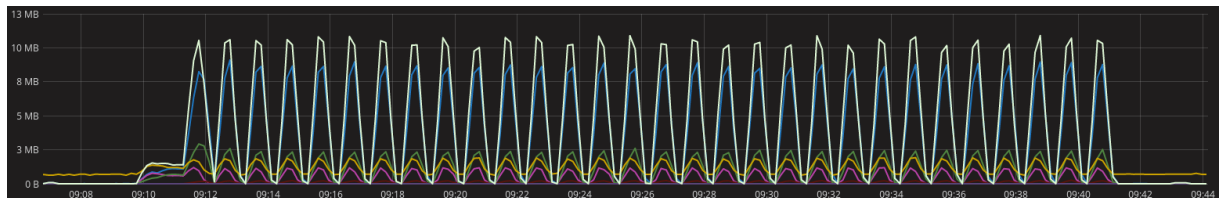
Disk input/output by container



Network output by container



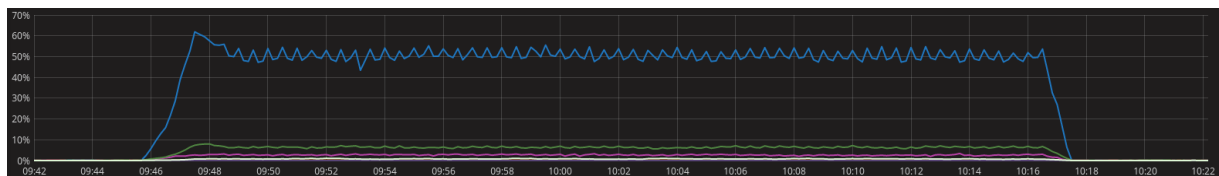
Network input by container



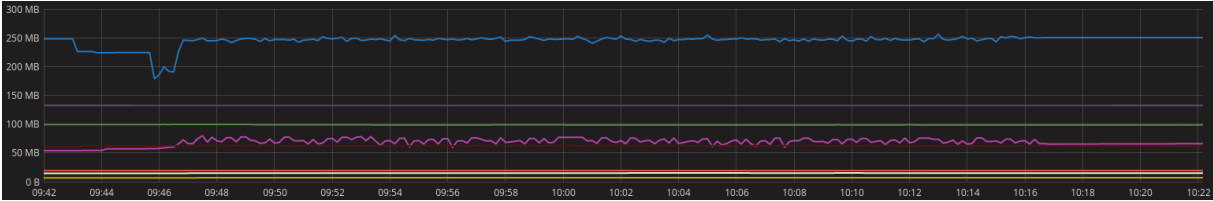
5.1.2.2 Interval between messages of 30 seconds

5.1.2.2.1 Time series analytics results

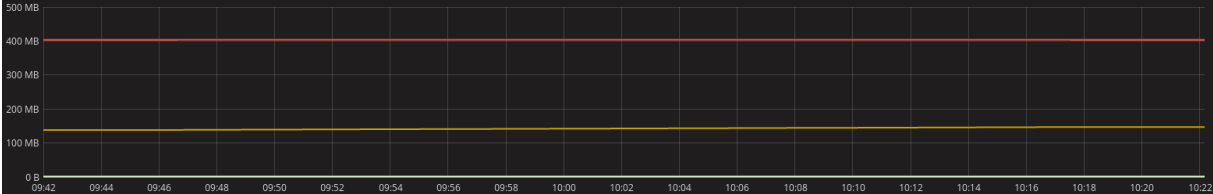
CPU usage by container



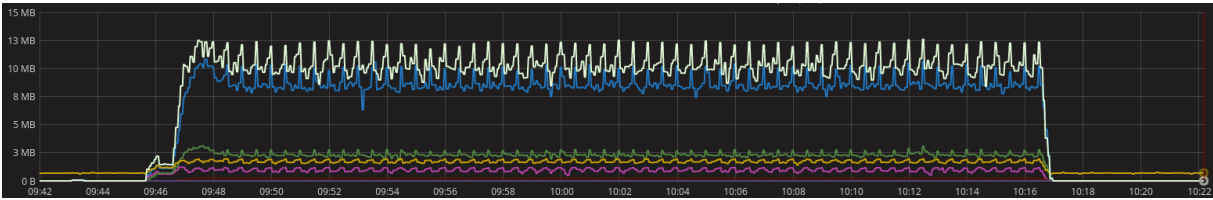
Memory RAM usage by container



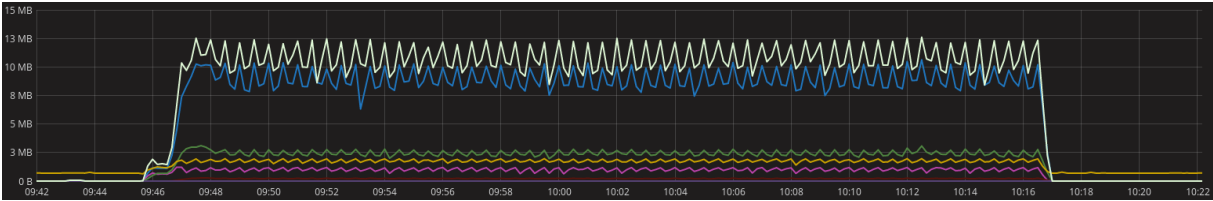
Disk input/output by container



Network output by container



Network input by container






5.2 Load Generator Experiments

This appendix section aims to show the results obtained for each iteration of the load generator experiment. These results include screenshots of the automation tool used to automate the tests and of the time series analytics tool.

For the time series analytics tool the colours for each chart represent the following services:

	Mongo		Postgres		Meshblu Socket.io		Load generator broker
---	-------	---	----------	---	-------------------	---	-----------------------

	Redis		Meshblu HTTP		Meshblu Core		Reverse proxy
---	-------	---	--------------	---	--------------	---	---------------

5.2.1 Iteration A - Traffic

This iteration was done in order to test the MBaaS against traffic data. For that purpose, this iteration used 100 sensors to communicate with.

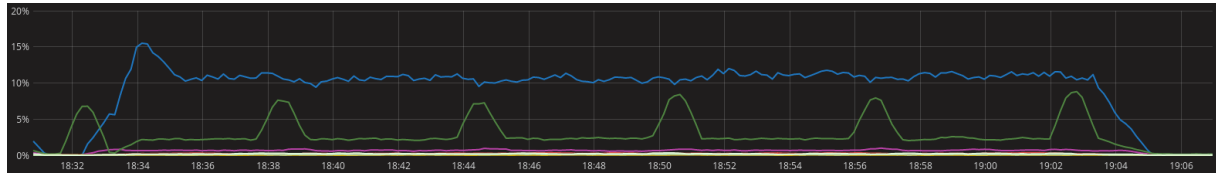
5.2.1.1 Interval between messages of 60 seconds

5.2.1.1.1 Automation tool results

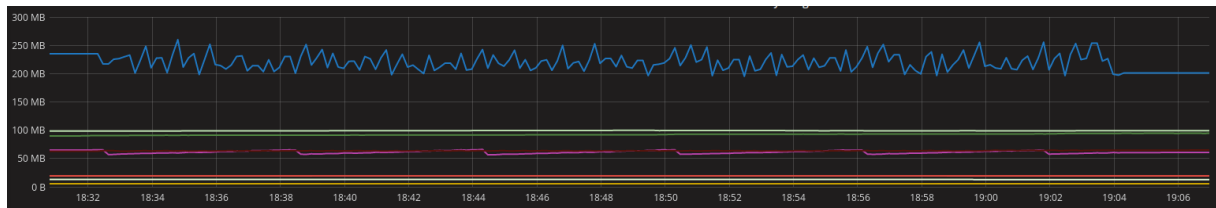
```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-20 18:32:06.674821
[localhost] local: docker-compose up
Creating performance
Creating listener
Attaching to listener, performance
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener | Downloading socketIO-client-0.7.2.tar.gz
performance | Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading websocket-client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener | Running setup.py install for websocket-client: started
listener | Running setup.py install for websocket-client: finished with status 'done'
listener | Running setup.py install for socketIO-client: started
listener | Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener |
listener | Listener Process:
listener |
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener |
listener | Listener results:
listener |
listener | Expected received messages: 3000.0
listener |
listener | Received messages: 3000
listener |
listener exited with code 0
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: traffic
performance | Number of sensors: 100
performance | Experiment duration: 1800s
performance | Messages interval: 60s
performance |
performance | Start Time: 2017-03-20 18:33:14.881377
performance | Finish Time: 2017-03-20 19:04:15.356738
performance | Elapsed Time: 0:31:00.475361
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 100
performance | Unregistered sensors: 100
performance | Registration ended: 2017-03-20 18:33:14.877033
performance | Unregistration ended: 2017-03-20 19:04:10.344788
performance | Messages sent: 3000
performance | Messages failed: 0
performance | Experiment start: 2017-03-20 18:32:13.186775
performance | Experiment end: 2017-03-20 19:03:08.489397
performance |
performance exited with code 0
[localhost] local: docker-compose down -v
Removing listener ... done
Removing performance ... done
Stopping performance test at 2017-03-20 19:04:18.941788
Performance test duration: 0:32:12.266967
Done.
```

5.2.1.1.2 Time series analytics results

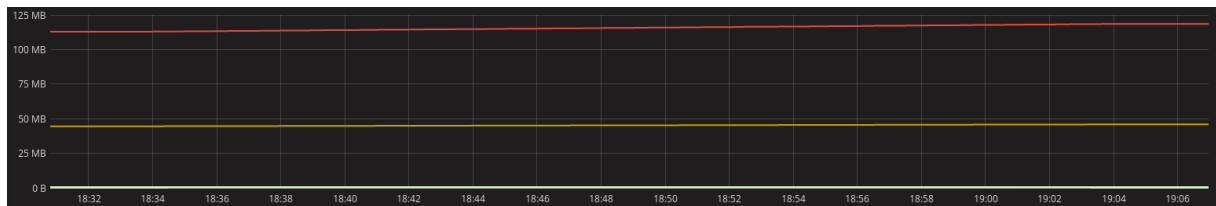
CPU usage by container



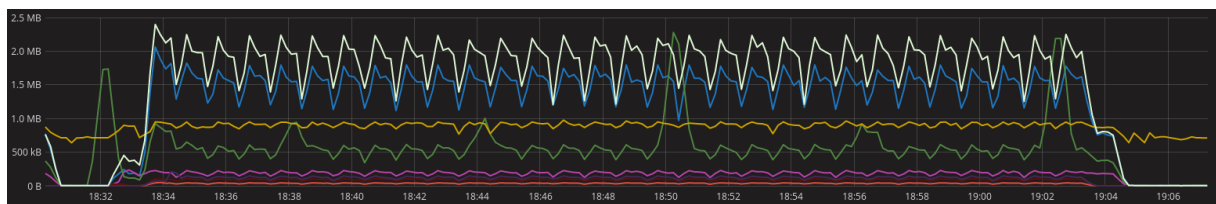
Memory RAM usage by container



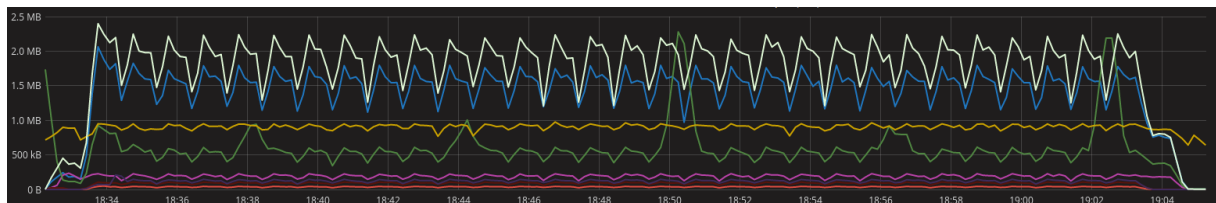
Disk I/O by container



Network output by container



Network input by container



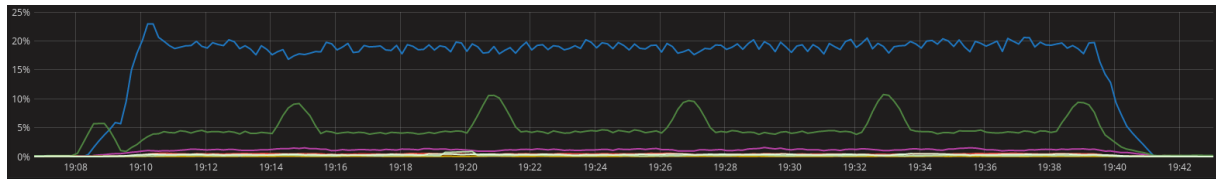
5.2.1.2 Interval between messages of 30 seconds

5.2.1.2.1 Automation tool results

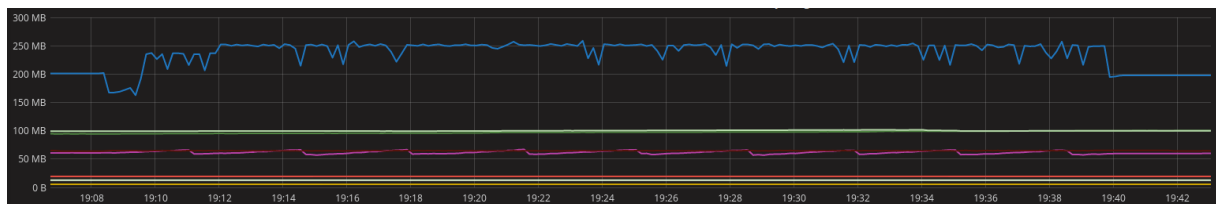
```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-20 19:08:09.953159
[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to listener, performance
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading websocket_client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener | Running setup.py install for websocket-client: started
listener | Running setup.py install for websocket-client: finished with status 'done'
listener | Running setup.py install for socketIO-client: started
listener | Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener |
listener | Listener Process:
listener |
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener |
listener | Listener results:
listener |
listener | Expected received messages: 6000.0
listener |
listener | Received messages: 6000
listener |
listener | listener exited with code 0
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: traffic
performance | Number of sensors: 100
performance | Experiment duration: 1800s
performance | Messages interval: 30s
performance |
performance | Start Time: 2017-03-20 19:09:19.873788
performance | Finish Time: 2017-03-20 19:40:23.501016
performance | Elapsed Time: 0:31:03.627228
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 100
performance | Unregistered sensors: 100
performance | Registration ended: 2017-03-20 19:09:19.869303
performance | Unregistration ended: 2017-03-20 19:40:18.499052
performance | Messages sent: 6000
performance | Messages failed: 0
performance | Experiment start: 2017-03-20 19:08:18.097562
performance | Experiment end: 2017-03-20 19:39:16.745560
performance |
performance | performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done
Stopping performance test at 2017-03-20 19:40:25.591945
Performance test duration: 0:32:15.638786
Done.
```


5.2.1.2.2 Time series analytics results

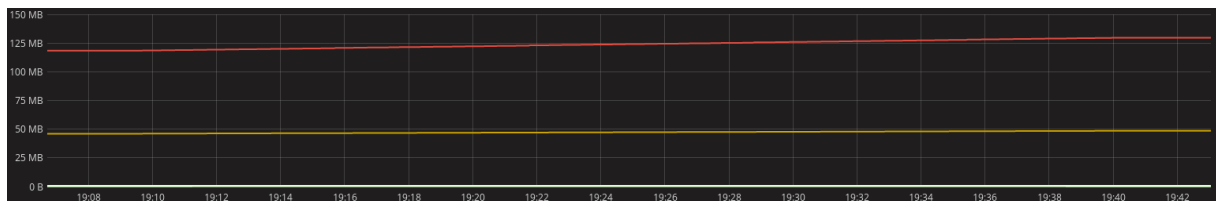
CPU usage by container



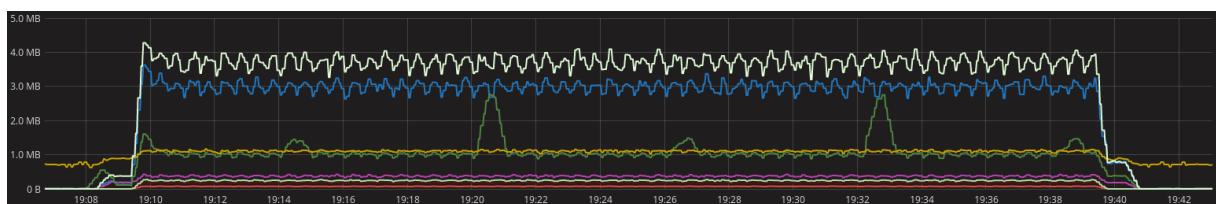
Memory RAM usage by container



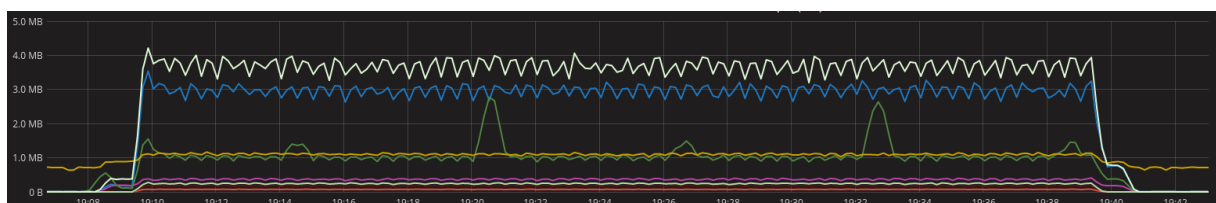
Disk I/O by container



Network output by container



Network input by container



5.2.2 Iteration A - Parking

This iteration was done in order to test the MBaaS against parking data. For that purpose, this iteration used 100 sensors to communicate with the MBaaS during 30 minutes with an interval between messages of 60 and 30 seconds.

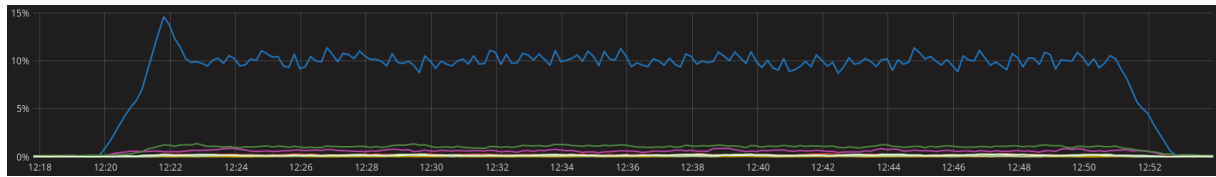
5.2.2.1 Interval between messages of 60 seconds

5.2.2.1.1 Automation tool results

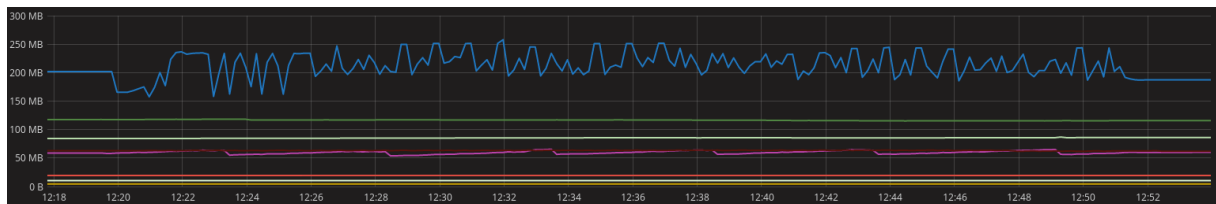
```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-20 12:19:44.602392
[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to performance, listener
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener    | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
listener    |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
listener    | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
performance | Successfully installed requests-2.7.0
listener    | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading six-1.10.0-py2.py3-none-any.whl
listener    | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading websocket_client-0.40.0.tar.gz (196kB)
listener    | Installing collected packages: requests, six, websocket-client, socketIO-client
listener    | Running setup.py install for websocket-client: started
listener    | Running setup.py install for websocket-client: finished with status 'done'
listener    | Running setup.py install for socketIO-client: started
listener    | Running setup.py install for socketIO-client: finished with status 'done'
listener    | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener    |
listener    | Listener Process:
listener    |
listener    | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener    |
listener    | Listener results:
listener    |
listener    | Expected received messages: 3000.0
listener    |
listener    | Received messages: 3000
listener    |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: parking
performance | Number of sensors: 100
performance | Experiment duration: 1800s
performance | Messages interval: 60s
performance |
performance | Start Time: 2017-03-20 12:20:51.157268
performance | Finish Time: 2017-03-20 12:51:51.643546
performance | Elapsed Time: 0:31:00.486278
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 100
performance | Unregistered sensors: 100
performance | Registration ended: 2017-03-20 12:20:51.153902
performance | Unregistration ended: 2017-03-20 12:51:46.641152
performance | Messages sent: 3000
performance | Messages failed: 0
performance | Experiment start: 2017-03-20 12:19:49.463852
performance | Experiment end: 2017-03-20 12:50:44.857478
performance |
performance | performance exited with code 0
performance | listener exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done
Stopping performance test at 2017-03-20 12:51:53.290681
Performance test duration: 0:32:08.688289
Done.
```

5.2.2.1.2 Time series analytics results

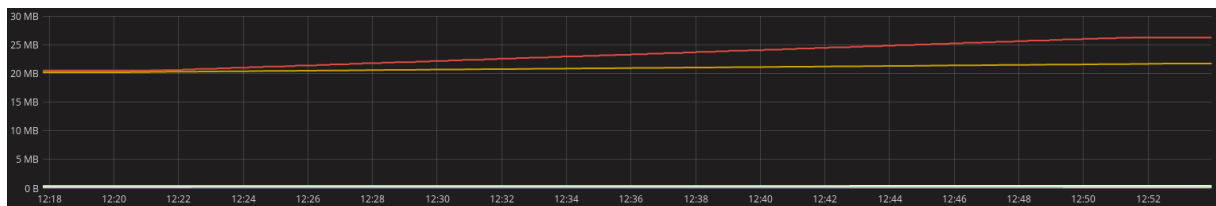
CPU usage by container



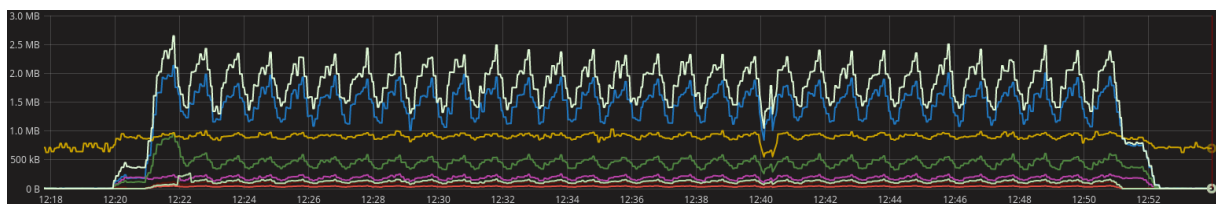
Memory RAM usage by container



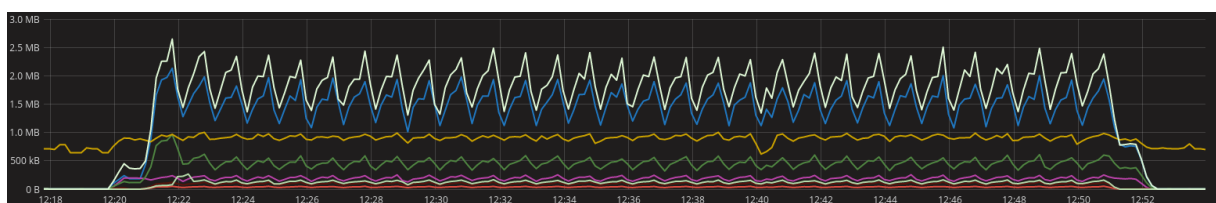
Disk I/O by container



Network output by container



Network input by container



5.2.2.2 Interval between messages of 30 seconds

5.2.2.2.1 Automation tool results

```
[localhost] Executing task 'set_vars'

Setting environment variables

[localhost] Executing task 'run'

Starting performance test at 2017-03-20 13:00:47.536354

[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to performance, listener
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
listener | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
performance | Successfully installed requests-2.7.0
listener |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading websocket-client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener |   Running setup.py install for websocket-client: started
listener |   Running setup.py install for websocket-client: finished with status 'done'
listener |   Running setup.py install for socketIO-client: started
listener |   Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener |
listener | Listener Process:
listener |
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener |
listener | Listener results:
listener |
listener | Expected received messages: 6000.0
listener |
listener | Received messages: 6000
listener |
listener exited with code 0
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: parking
performance | Number of sensors: 100
performance | Experiment duration: 1800s
performance | Messages interval: 30s
performance |
performance | Start Time: 2017-03-20 13:01:56.233578
performance | Finish Time: 2017-03-20 13:33:00.141800
performance | Elapsed Time: 0:31:03.908222
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 100
performance | Unregistered sensors: 100
performance | Registration ended: 2017-03-20 13:01:56.229218
performance | Unregistration ended: 2017-03-20 13:32:55.139042
performance | Messages sent: 6000
performance | Messages failed: 0
performance | Experiment start: 2017-03-20 13:00:54.498202
performance | Experiment end: 2017-03-20 13:31:53.412101
performance |
performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done

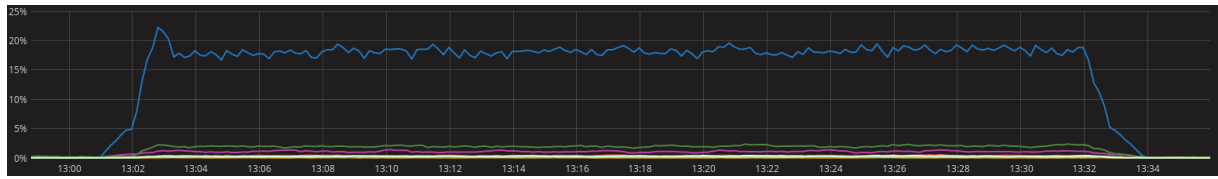
Stopping performance test at 2017-03-20 13:33:02.154584

Performance test duration: 0:32:14.618230

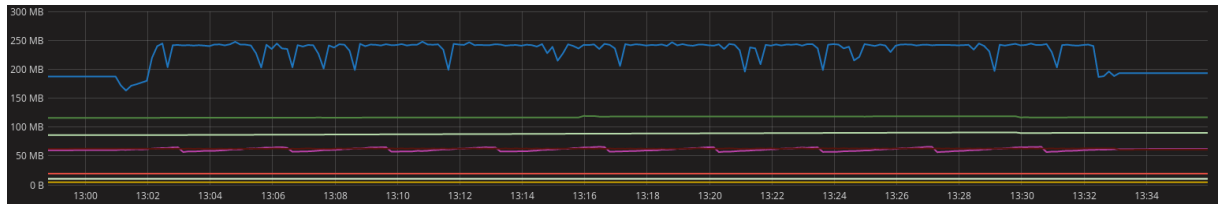
Done.
```

5.2.2.2 Time series analytics results

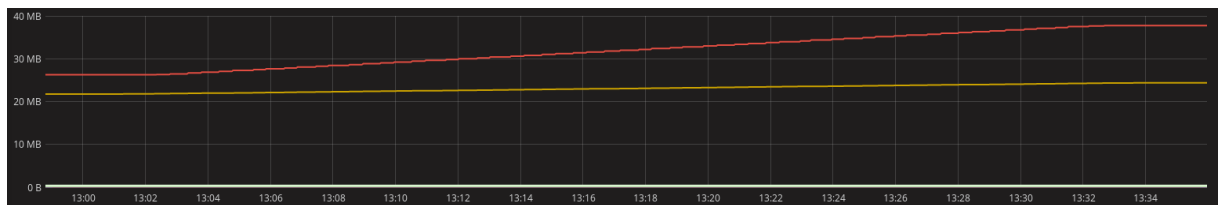
CPU usage by container



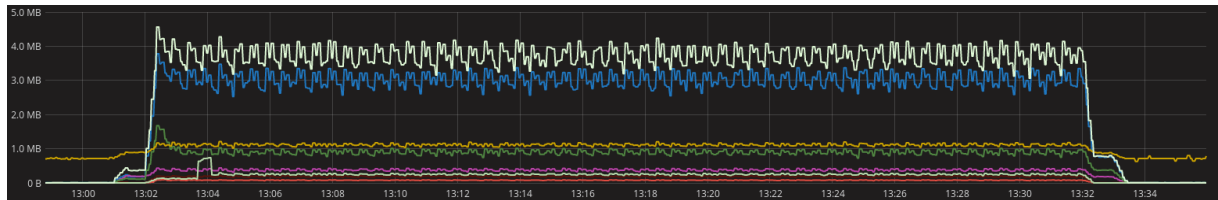
Memory RAM usage by container



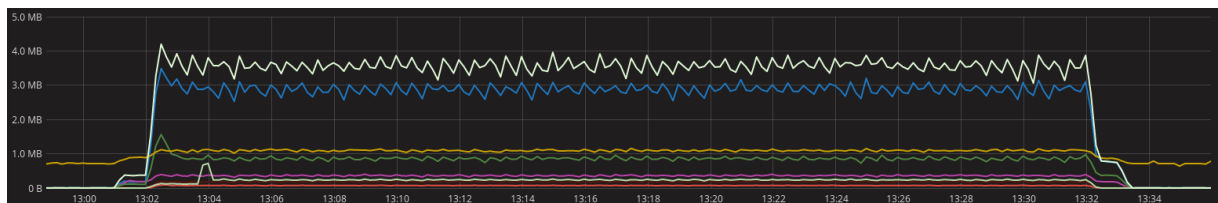
Disk I/O by container



Network output by container



Network input by container



5.2.3 Iteration A - Environment

This iteration was done in order to test the MBaaS against environment data. For that purpose, this iteration used 100 sensors to communicate with the MBaaS during 30 minutes with an interval between messages of 60 and 30 seconds.

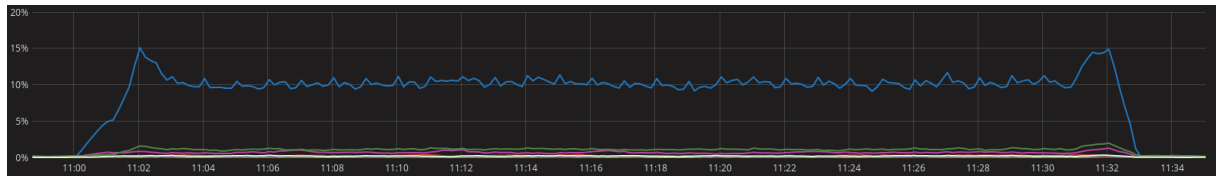
5.2.3.1 Interval between messages of 60 seconds

5.2.3.1.1 Automation tool results

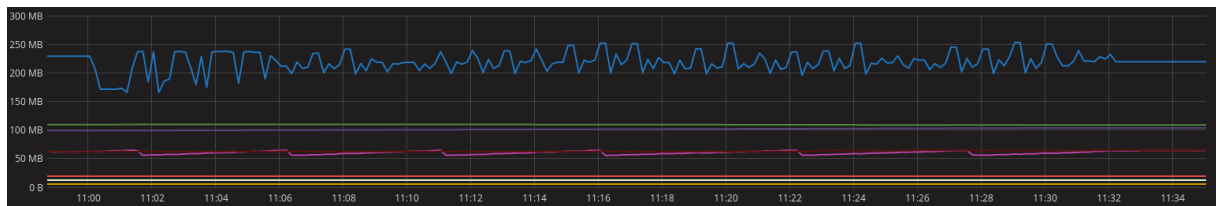
```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-21 10:59:58.413979
[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to listener, performance
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener | Downloading socketIO-client-0.7.2.tar.gz
performance | Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading websocket-client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener | Running setup.py install for websocket-client: started
listener | Running setup.py install for websocket-client: finished with status 'done'
listener | Running setup.py install for socketIO-client: started
listener | Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener |
listener | Listener Process:
listener |
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener |
listener | Listener results:
listener |
listener | Expected received messages: 3000.0
listener |
listener | Received messages: 3000
listener |
listener exited with code 0
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: environment
performance | Number of sensors: 100
performance | Experiment duration: 1800s
performance | Messages interval: 60s
performance |
performance | Start Time: 2017-03-21 11:01:05.905658
performance | Finish Time: 2017-03-21 11:32:06.919167
performance | Elapsed Time: 0:31:01.013509
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 100
performance | Unregistered sensors: 100
performance | Registration ended: 2017-03-21 11:01:05.902945
performance | Unregistration ended: 2017-03-21 11:32:01.914533
performance | Messages sent: 3000
performance | Messages failed: 0
performance | Experiment start: 2017-03-21 11:00:03.752698
performance | Experiment end: 2017-03-21 11:30:59.999133
performance |
performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done
Stopping performance test at 2017-03-21 11:32:08.557466
Performance test duration: 0:32:10.143487
Done.
```

5.2.3.1.2 Time series analytics results

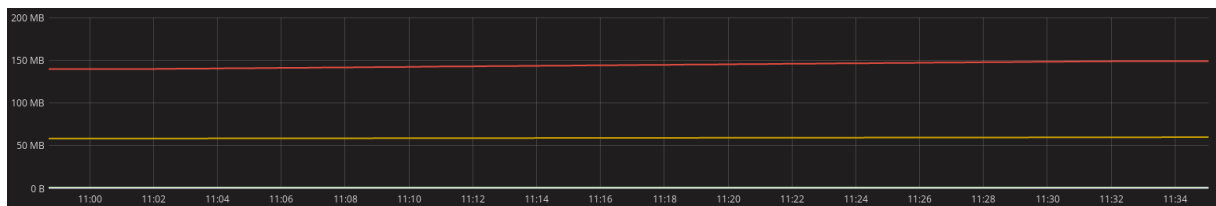
CPU usage by container



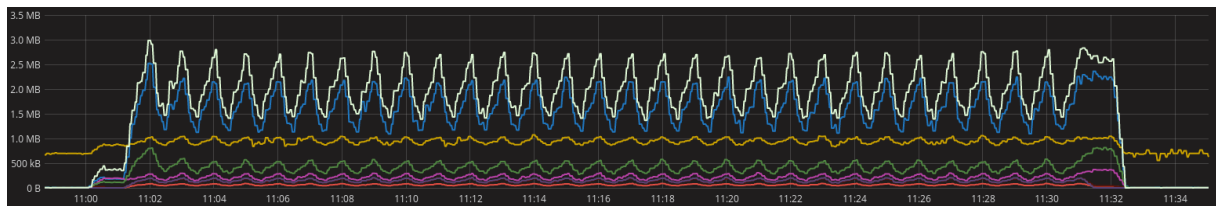
Memory RAM usage by container



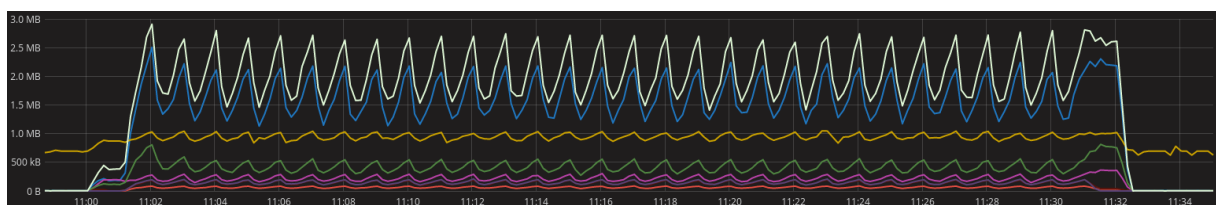
Disk I/O by container



Network output by container



Network input by container



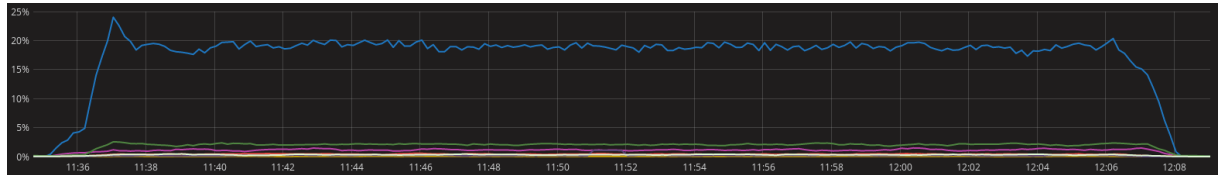
5.2.3.2 Interval between messages of 30 seconds

5.2.3.2.1 Automation tool results

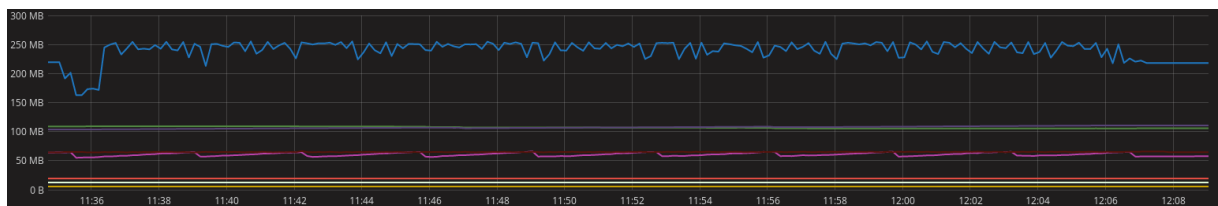
```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-21 11:35:00.647143
[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to performance, listener
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener    | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
listener    |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener    | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener    | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading six-1.10.0-py2.py3-none-any.whl
listener    | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading websocket-client-0.40.0.tar.gz (196kB)
listener    | Installing collected packages: requests, six, websocket-client, socketIO-client
listener    |   Running setup.py install for websocket-client: started
listener    |   Running setup.py install for websocket-client: finished with status 'done'
listener    |   Running setup.py install for socketIO-client: started
listener    |   Running setup.py install for socketIO-client: finished with status 'done'
listener    | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener    | Listener Process:
listener    | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30
c7351d9
listener    | Listener results:
listener    | Expected received messages: 6000.0
listener    | Received messages: 6000
listener    | listener exited with code 0
performance | Performance Process:
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance | Performance Results:
performance | Experiment type: environment
performance | Number of sensors: 100
performance | Experiment duration: 1800s
performance | Messages interval: 30s
performance | Start Time: 2017-03-21 11:36:11.246352
performance | Finish Time: 2017-03-21 12:07:15.256638
performance | Elapsed Time: 0:31:04.010286
performance | Load Generator Results:
performance | Registered sensors: 100
performance | Unregistered sensors: 100
performance | Registration ended: 2017-03-21 11:36:11.101826
performance | Unregistration ended: 2017-03-21 12:07:10.243732
performance | Messages sent: 6000
performance | Messages failed: 0
performance | Experiment start: 2017-03-21 11:35:09.407688
performance | Experiment end: 2017-03-21 12:06:08.309451
performance | performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done
Stopping performance test at 2017-03-21 12:07:17.988441
Performance test duration: 0:32:17.341298
Done.
```


5.2.3.2 Time series analytics results

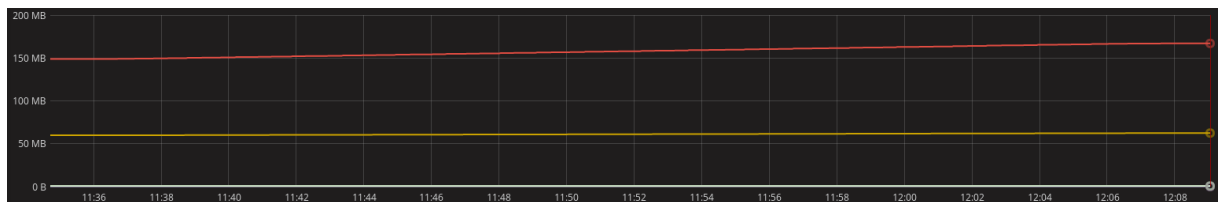
CPU usage by container



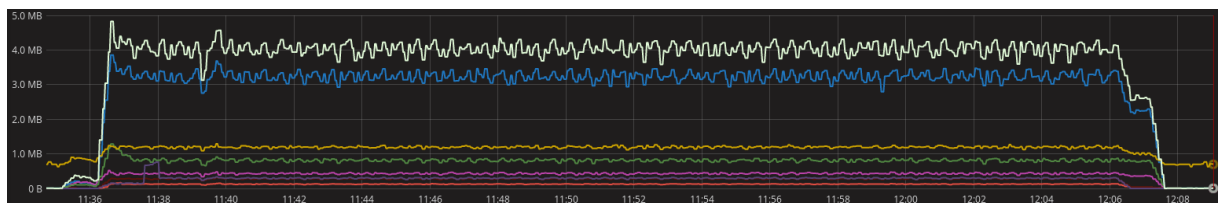
Memory RAM usage by container



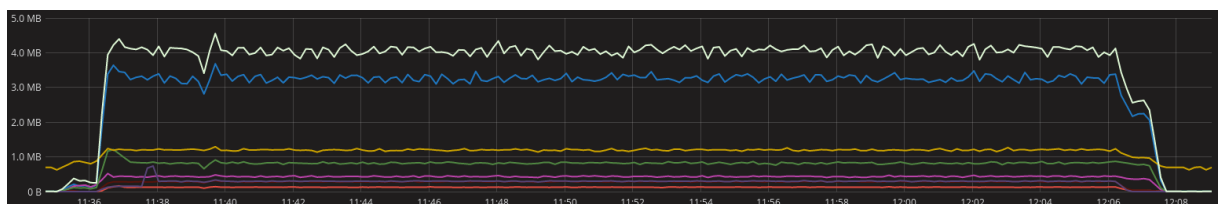
Disk I/O by container



Network output by container



Network input by container



5.2.4 Iteration B - Traffic

This iteration was done in order to test the MBaaS against traffic data. For that purpose, this iteration used 300 sensors to communicate with the MBaaS during 30 minutes with an interval between messages of 60 and 30 seconds.

5.2.4.1 Interval between messages of 60 seconds

5.2.4.1.1 Automation tool results

```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-21 12:09:56.782357

[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to performance, listener
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading websocket-client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener | Running setup.py install for websocket-client: started
listener | Running setup.py install for websocket-client: finished with status 'done'
listener | Running setup.py install for socketIO-client: started
listener | Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener |
listener | Listener Process:
listener |
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener |
listener | Listener results:
listener |
listener | Expected received messages: 9000.0
listener |
listener | Received messages: 9000
listener |
listener exited with code 0
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: traffic
performance | Number of sensors: 300
performance | Experiment duration: 1800s
performance | Messages interval: 60s
performance |
performance | Start Time: 2017-03-21 12:13:06.757426
performance | Finish Time: 2017-03-21 12:46:07.844475
performance | Elapsed Time: 0:33:01.087049
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 300
performance | Unregistered sensors: 300
performance | Registration ended: 2017-03-21 12:13:06.744324
performance | Unregistration ended: 2017-03-21 12:46:02.841070
performance | Messages sent: 9000
performance | Messages failed: 0
performance | Experiment start: 2017-03-21 12:10:04.872518
performance | Experiment end: 2017-03-21 12:43:00.803871
performance |
performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done

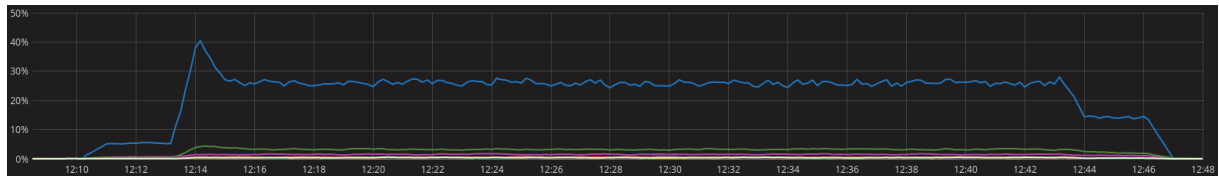
Stopping performance test at 2017-03-21 12:46:09.184187

Performance test duration: 0:36:12.401830

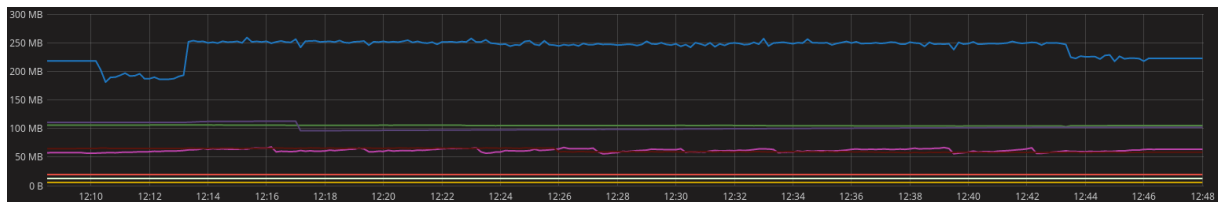
Done,
```

5.2.4.1.2 Time series analytics tools

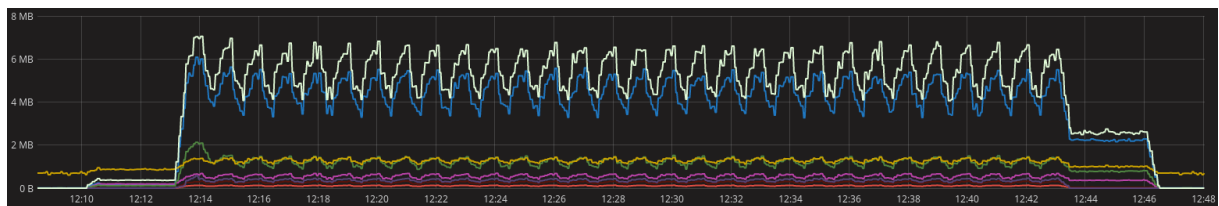
CPU usage by container



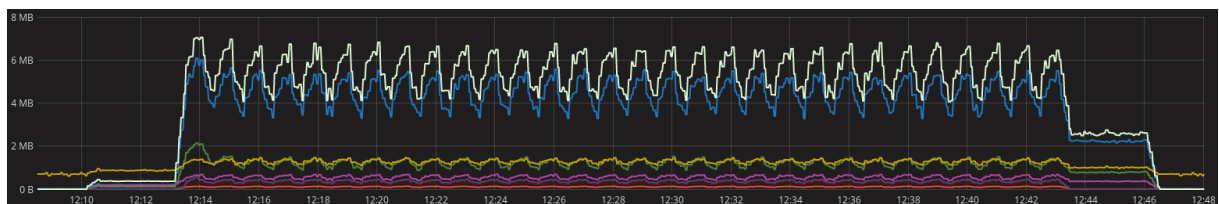
Memory RAM usage by container



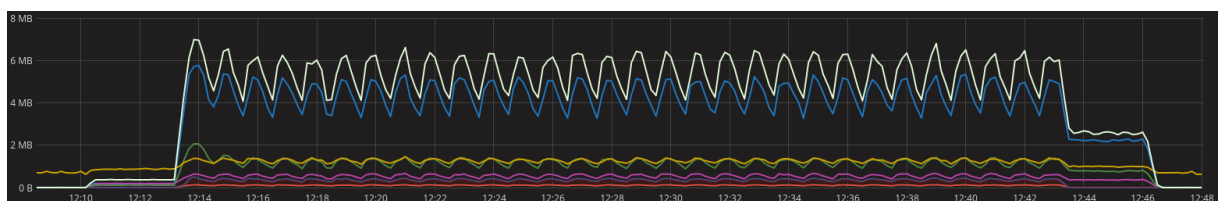
Disk I/O by container



Network output by container



Network input by container



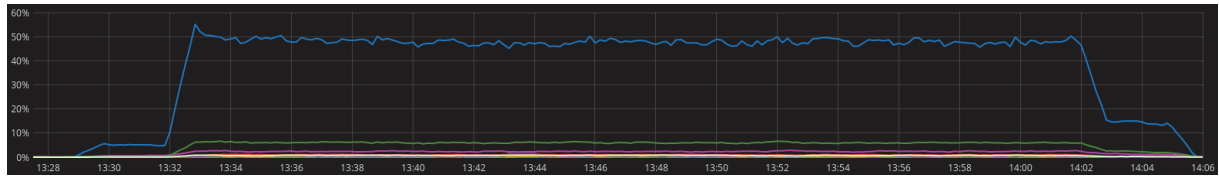
5.2.4.2 Interval between messages of 30 seconds

5.2.4.2.1 Automation tool results

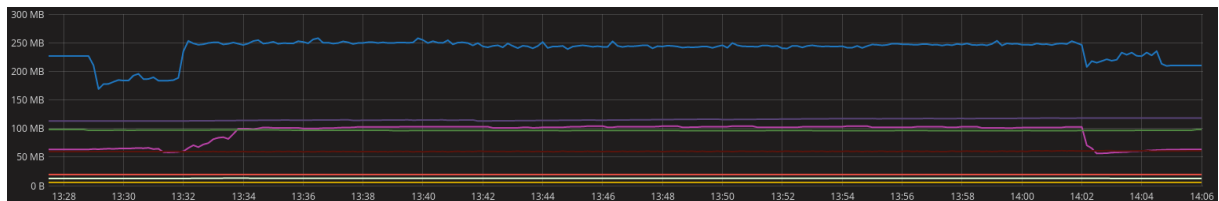
```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-21 13:28:38.189591
[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to listener, performance
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading websocket-client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener |   Running setup.py install for websocket-client: started
listener |     Running setup.py install for websocket-client: finished with status 'done'
listener |   Running setup.py install for socketIO-client: started
listener |     Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener |
listener | Listener Process:
listener |
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener |
listener | Listener results:
listener |
listener | Expected received messages: 18000.0
listener |
listener | Received messages: 18000
listener |
listener | listener exited with code 0
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: traffic
performance | Number of sensors: 300
performance | Experiment duration: 1800s
performance | Messages interval: 30s
performance |
performance | Start Time: 2017-03-21 13:31:49.527022
performance | Finish Time: 2017-03-21 14:04:54.761505
performance | Elapsed Time: 0:33:05.234483
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 300
performance | Unregistered sensors: 300
performance | Registration ended: 2017-03-21 13:31:49.518794
performance | Unregistration ended: 2017-03-21 14:04:49.759028
performance | Messages sent: 18000
performance | Messages failed: 0
performance | Experiment start: 2017-03-21 13:28:47.510907
performance | Experiment end: 2017-03-21 14:01:47.657324
performance |
performance | performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done
Stopping performance test at 2017-03-21 14:04:56.044555
Performance test duration: 0:36:17.854964
Done.
```

5.2.4.2.2 Time series analytics results

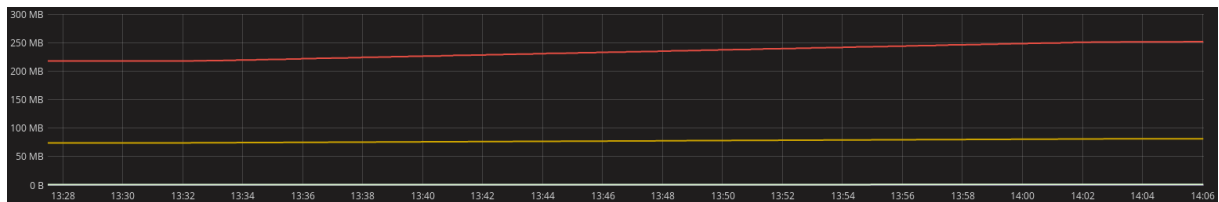
CPU usage by container



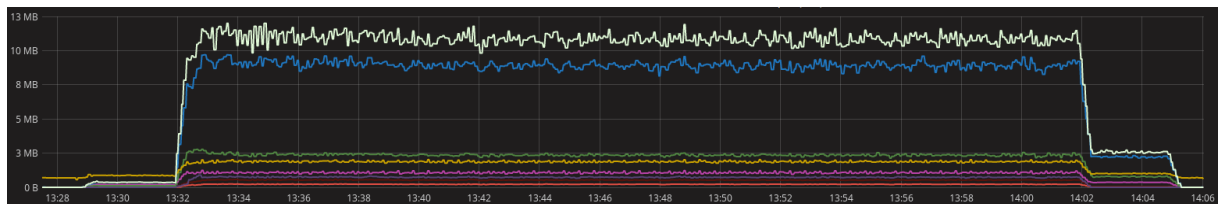
Memory RAM usage by container



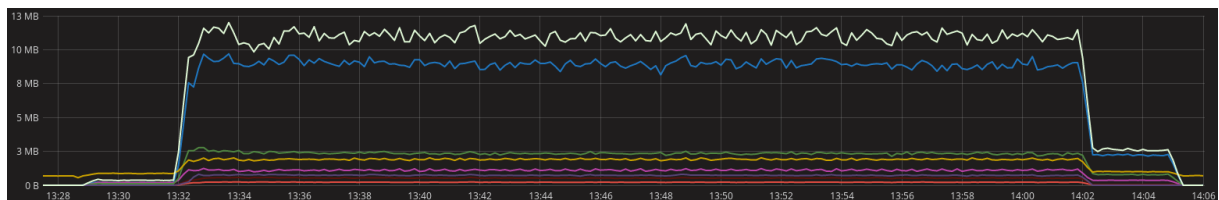
Disk I/O by container



Network output by container



Network input by container



5.2.5 Iteration B - Parking

This iteration was done in order to test the MBaaS against parking data. For that purpose, this iteration used 300 sensors to communicate with the MBaaS during 30 minutes with an interval between messages of 60 and 30 seconds.

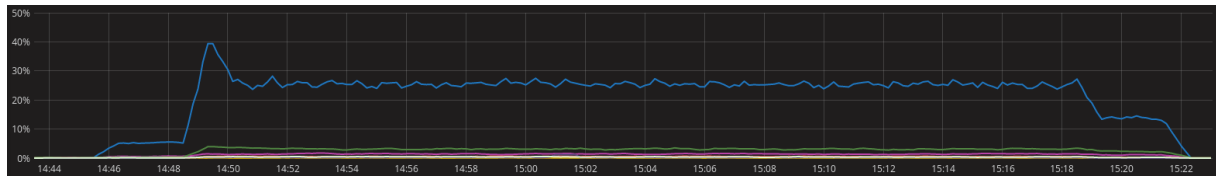
5.2.5.1 Interval between messages of 60 seconds

5.2.5.1.1 Automation tool results

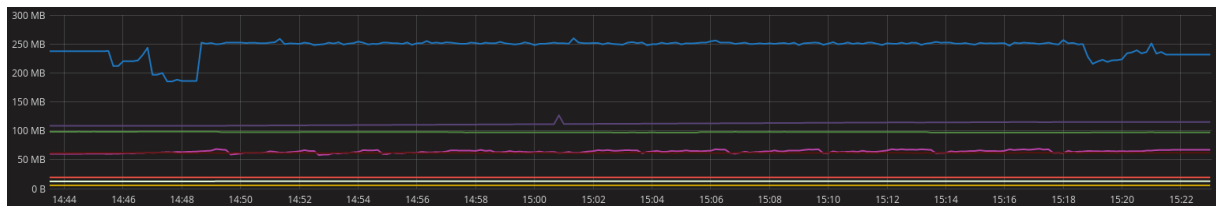
```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-21 14:45:19.622732
[localhost] local: docker-compose up
Creating performance
Creating listener
Attaching to performance, listener
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener | Collecting requests==2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading websocket_client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener | Running setup.py install for websocket-client: started
listener | Running setup.py install for websocket-client: finished with status 'done'
listener | Running setup.py install for socketIO-client: started
listener | Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener | Listener Process:
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener | Listener results:
listener | Expected received messages: 9000.0
listener | Received messages: 9000
listener exited with code 0
performance | Performance Process:
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance | Performance Results:
performance | Experiment type: parking
performance | Number of sensors: 300
performance | Experiment duration: 1800s
performance | Messages interval: 60s
performance | Start Time: 2017-03-21 14:48:25.084591
performance | Finish Time: 2017-03-21 15:21:25.692256
performance | Elapsed Time: 0:33:00.607665
performance | Load Generator Results:
performance | Registered sensors: 300
performance | Unregistered sensors: 300
performance | Registration ended: 2017-03-21 14:48:25.074940
performance | Unregistration ended: 2017-03-21 15:21:20.688929
performance | Messages sent: 9000
performance | Messages failed: 0
performance | Experiment start: 2017-03-21 14:45:23.235362
performance | Experiment end: 2017-03-21 15:18:18.604849
performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done
Stopping performance test at 2017-03-21 15:21:26.935218
Performance test duration: 0:36:07.312486
Done.
```

5.2.5.1.2 Time series analytics results

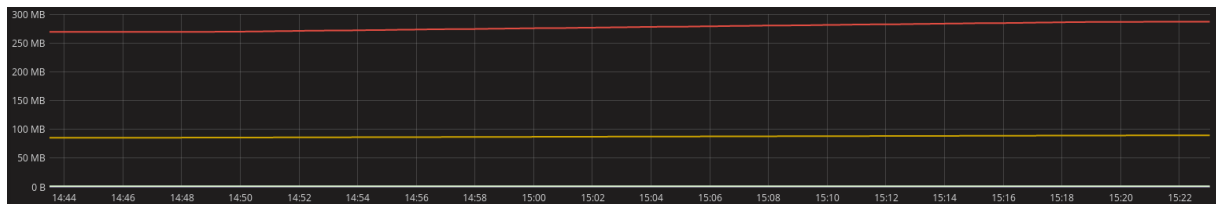
CPU usage by container



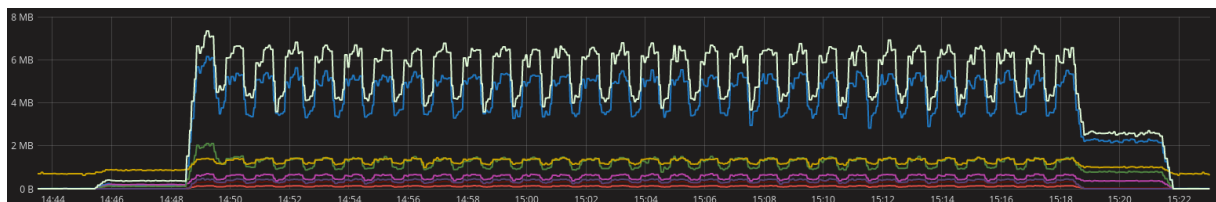
Memory RAM usage by container



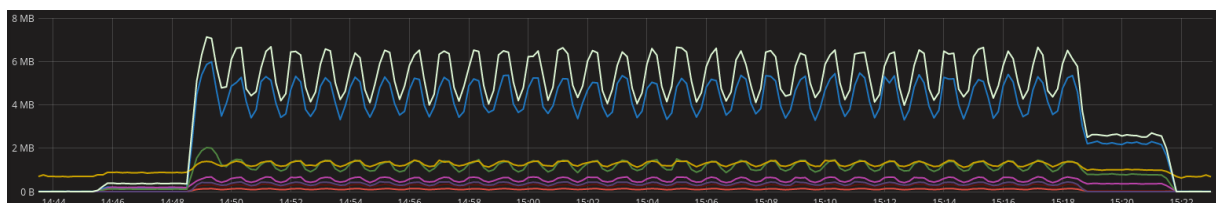
Disk I/O by container



Network output by container



Network input by container



5.2.5.2 Interval between messages of 30 seconds

5.2.5.2.1 Automation tool results

```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-21 15:30:32.835885

[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to listener, performance
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener    | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
listener    |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener    | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener    | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading six-1.10.0-py2.py3-none-any.whl
listener    | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener    |   Downloading websocket-client-0.40.0.tar.gz (196kB)
listener    | Installing collected packages: requests, six, websocket-client, socketIO-client
listener    |   Running setup.py install for websocket-client: started
listener    |     Running setup.py install for websocket-client: finished with status 'done'
listener    |   Running setup.py install for socketIO-client: started
listener    |     Running setup.py install for socketIO-client: finished with status 'done'
listener    | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener    |
listener    | Listener Process:
listener    |
listener    | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener    |
listener    | Listener results:
listener    |
listener    | Expected received messages: 18000.0
listener    |
listener    | Received messages: 18000
listener    |
listener exited with code 0
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: parking
performance | Number of sensors: 300
performance | Experiment duration: 1800s
performance | Messages interval: 30s
performance |
performance | Start Time: 2017-03-21 15:33:38.799776
performance | Finish Time: 2017-03-21 16:06:45.067443
performance | Elapsed Time: 0:33:06.267667
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 300
performance | Unregistered sensors: 300
performance | Registration ended: 2017-03-21 15:33:38.792110
performance | Unregistration ended: 2017-03-21 16:06:40.044578
performance | Messages sent: 18000
performance | Messages failed: 0
performance | Experiment start: 2017-03-21 15:30:36.884135
performance | Experiment end: 2017-03-21 16:03:37.708515
performance |
performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done

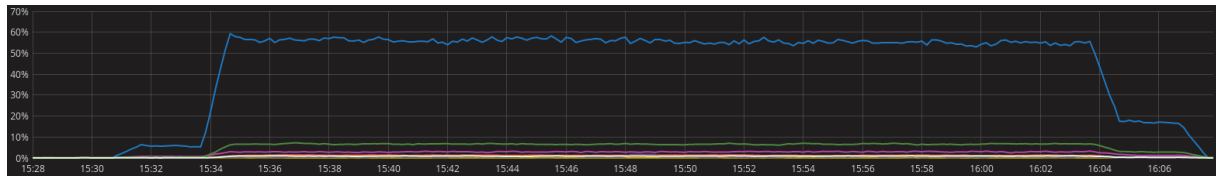
Stopping performance test at 2017-03-21 16:06:49.047945

Performance test duration: 0:36:16.212060

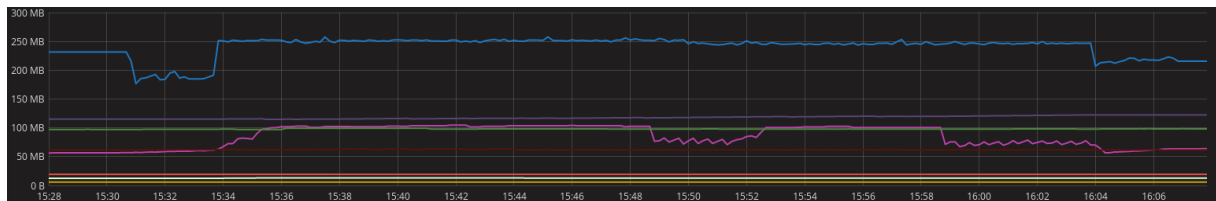
Done.
```

5.2.5.2.2 Time series analytics tools

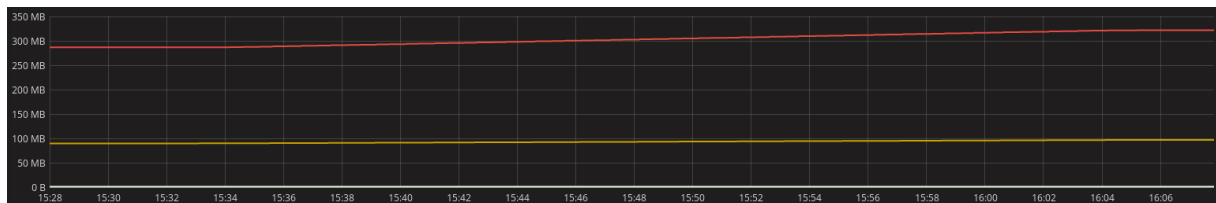
CPU usage by container



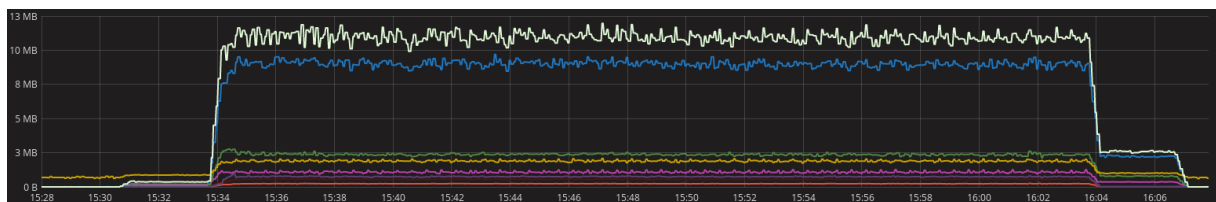
Memory RAM usage by container



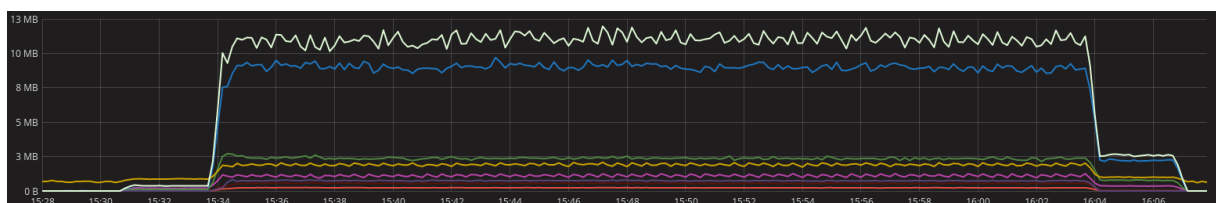
Disk I/O by container



Network output by container



Network input by container



5.2.6 Iteration B - Environment

This iteration was done in order to test the MBaaS against environment data. For that purpose, this iteration used 300 sensors to communicate with the MBaaS during 30 minutes with an interval between messages of 60 and 30 seconds.

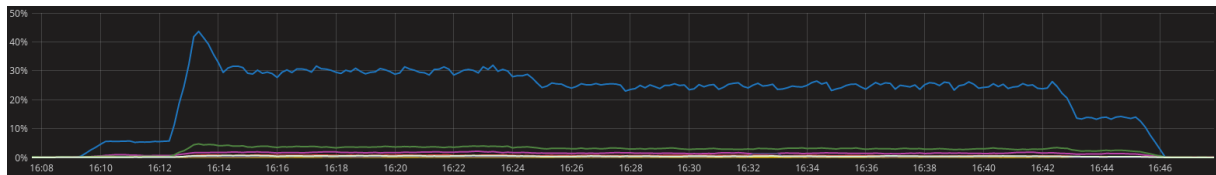
5.2.6.1 Interval between messages of 60 seconds

5.2.6.1.1 Automation tool results

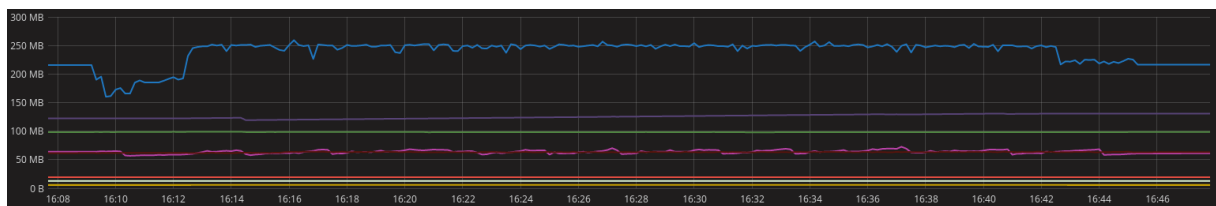
```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-21 16:08:54.313673
[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to listener, performance
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener | Downloading socketIO-client-0.7.2.tar.gz
performance | Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener | Downloading websocket-client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener | Running setup.py install for websocket-client: started
listener | Running setup.py install for websocket-client: finished with status 'done'
listener | Running setup.py install for socketIO-client: started
listener | Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener | Listener Process:
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener | Listener results:
listener | Expected received messages: 9000.0
listener | Received messages: 9000
listener exited with code 0
performance | Performance Process:
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance | Performance Results:
performance | Experiment type: environment
performance | Number of sensors: 300
performance | Experiment duration: 1800s
performance | Messages interval: 60s
performance | Start Time: 2017-03-21 16:12:16.153736
performance | Finish Time: 2017-03-21 16:45:17.309220
performance | Elapsed Time: 0:33:01.155484
performance | Load Generator Results:
performance | Registered sensors: 300
performance | Unregistered sensors: 300
performance | Registration ended: 2017-03-21 16:12:16.133682
performance | Unregistration ended: 2017-03-21 16:45:12.302216
performance | Messages sent: 9000
performance | Messages failed: 0
performance | Experiment start: 2017-03-21 16:09:14.167940
performance | Experiment end: 2017-03-21 16:42:10.122063
performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done
Stopping performance test at 2017-03-21 16:45:18.870607
Performance test duration: 0:36:24.556934
Done.
```

5.2.6.1.2 Time series analytics results

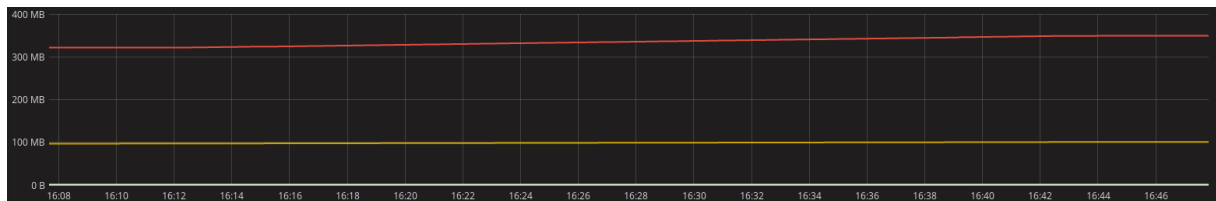
CPU usage by container



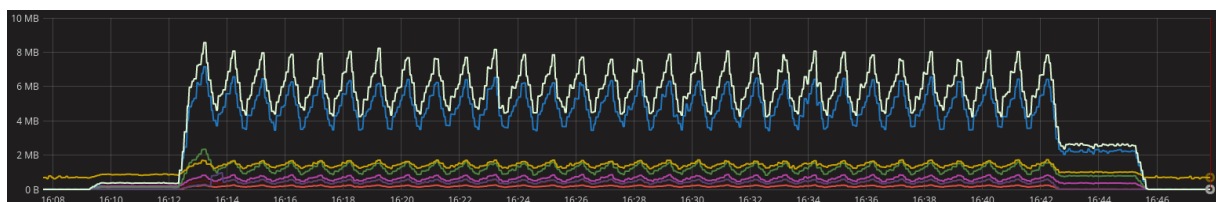
Memory RAM usage by container



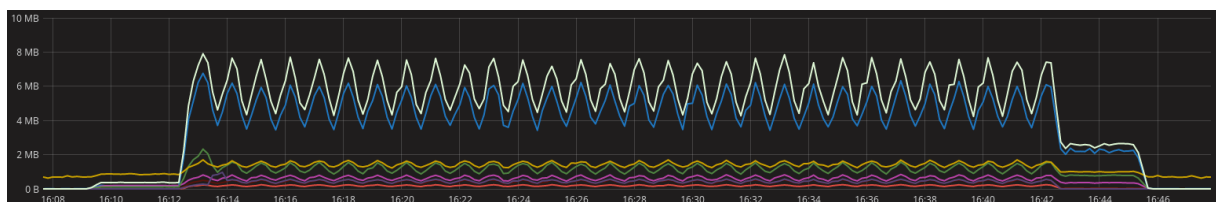
Disk I/O by container



Network output by container



Network input by container



5.2.6.2 Interval between messages of 30 seconds

5.2.6.2.1 Automation tool results

```
[localhost] Executing task 'set_vars'
Setting environment variables
[localhost] Executing task 'run'
Starting performance test at 2017-03-21 16:51:03.330630

[localhost] local: docker-compose up
Creating listener
Creating performance
Attaching to listener, performance
listener | Collecting socketIO-client==0.7.2 (from -r requirements.txt (line 1))
performance | Collecting requests==2.7.0 (from -r requirements.txt (line 1))
listener |   Downloading socketIO-client-0.7.2.tar.gz
performance |   Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
performance | Installing collected packages: requests
performance | Successfully installed requests-2.7.0
listener | Collecting requests>=2.7.0 (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
listener | Collecting six (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading six-1.10.0-py2.py3-none-any.whl
listener | Collecting websocket-client (from socketIO-client==0.7.2->-r requirements.txt (line 1))
listener |   Downloading websocket-client-0.40.0.tar.gz (196kB)
listener | Installing collected packages: requests, six, websocket-client, socketIO-client
listener | Running setup.py install for websocket-client: started
listener | Running setup.py install for websocket-client: finished with status 'done'
listener | Running setup.py install for socketIO-client: started
listener | Running setup.py install for socketIO-client: finished with status 'done'
listener | Successfully installed requests-2.13.0 six-1.10.0 socketIO-client-0.7.2 websocket-client-0.40.0
listener |
listener | Listener Process:
listener |
listener | Listening to Gateway ef5cd734-b565-4fe9-85d2-14f30c7351d9
listener |
listener | Listener results:
listener |
listener | Expected received messages: 18000.0
listener |
listener | Received messages: 18000
listener |
listener | listener exited with code 0
performance |
performance | Performance Process:
performance |
performance | [PASS] Create User
performance | [PASS] Create Experiment
performance | [PASS] Configure Experiment
performance | [PASS] Start Experiment
performance | [PASS] Get Experiment Log
performance | [PASS] Delete Experiment
performance | [PASS] Delete User
performance |
performance | Performance Results:
performance |
performance | Experiment type: environment
performance | Number of sensors: 300
performance | Experiment duration: 1800s
performance | Messages interval: 30s
performance |
performance | Start Time: 2017-03-21 16:54:28.099396
performance | Finish Time: 2017-03-21 17:27:34.232465
performance | Elapsed Time: 0:33:06.133069
performance |
performance | Load Generator Results:
performance |
performance | Registered sensors: 300
performance | Unregistered sensors: 300
performance | Registration ended: 2017-03-21 16:54:28.088825
performance | Unregistration ended: 2017-03-21 17:27:29.226123
performance | Messages sent: 18000
performance | Messages failed: 0
performance | Experiment start: 2017-03-21 16:51:25.775420
performance | Experiment end: 2017-03-21 17:24:27.085984
performance |
performance | performance exited with code 0
[localhost] local: docker-compose down -v
Removing performance ... done
Removing listener ... done

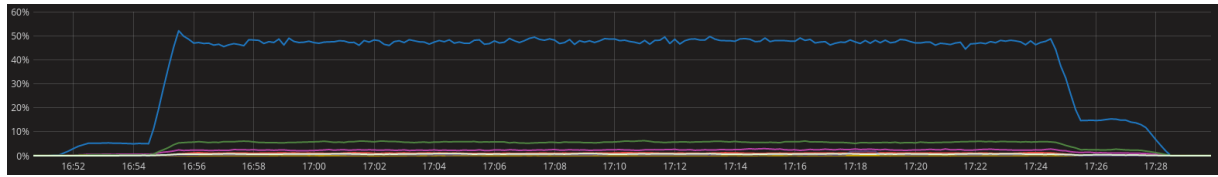
Stopping performance test at 2017-03-21 17:27:37.150517

Performance test duration: 0:36:33.819887

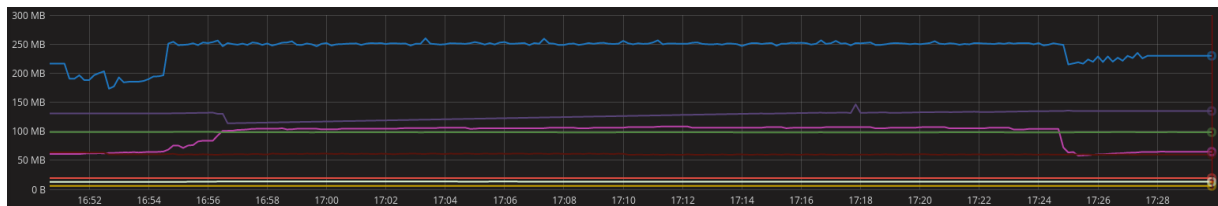
Done.
```

5.2.6.2.2 Time series analytics results

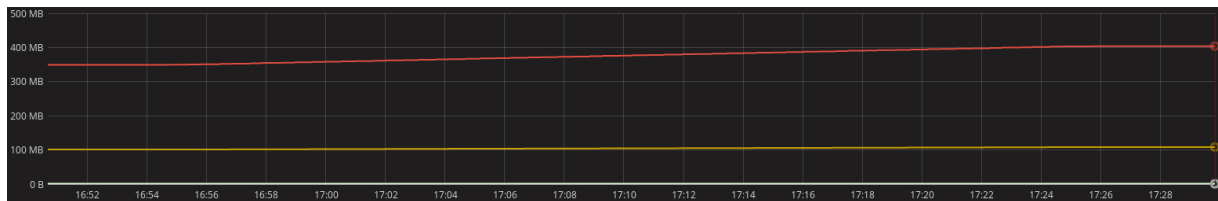
CPU usage by container



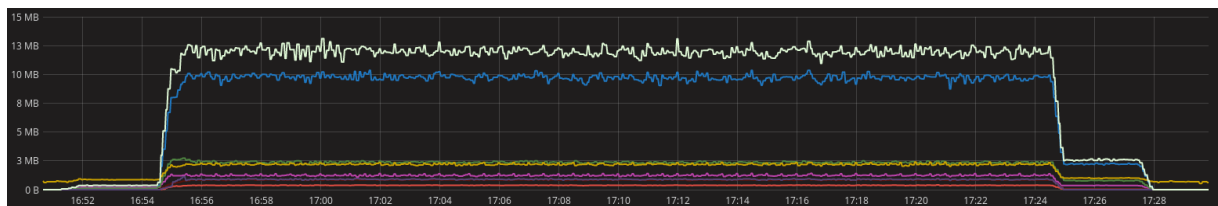
Memory RAM usage by container



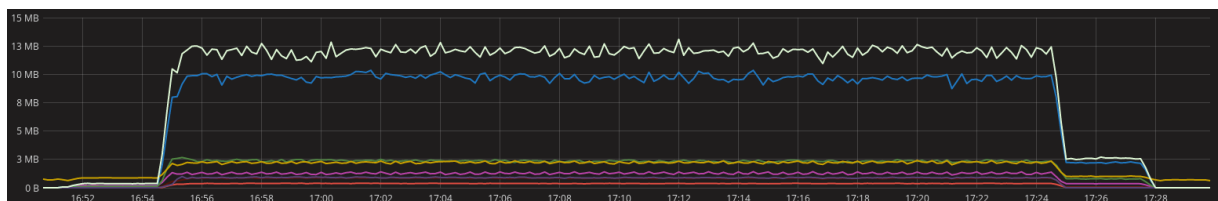
Disk I/O by container



Network output by container



Network input by container



6 References

- [1] European Commission, “FIRE+,” [Online]. Available: <http://www.ict-fire.eu/home.html>.
- [2] City Pulse, “City Pulse,” [Online]. Available: <http://www.ict-citypulse.eu/>.
- [3] Octoblu, “Meshblu,” [Online]. Available: <https://meshblu.readme.io/>.
- [4] FIT-IoT Lab, “A8 Open Node,” 31 3 2017. [Online]. Available: <https://www.iot-lab.info/hardware/a8/>.
- [5] FIT-IoT Lab, “Deployment,” 31 3 2017. [Online]. Available: <https://www.iot-lab.info/deployment/>.
- [6] FIT-IoT Lab, “iotlab-injectors,” 31 3 2017. [Online]. Available: github.com/iot-lab/iotlab-injectors/.
- [7] MQTT, “MQTT,” [Online]. Available: <http://mqtt.org/>.
- [8] TZI, “CoAP,” [Online]. Available: <http://coap.technology/>.
- [9] OMA, “OMA LwM2M,” [Online]. Available: <http://openmobilealliance.org/specifications/lightweight-m2m-specification-form-oma/>.
- [10] oneM2M, “oneM2M,” [Online]. Available: <http://www.onem2m.org>.
- [11] Telefonica, “FI-WARE NGSI Open RESTful API Specification,” [Online]. Available: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_NGSI_Open_RESTful_API_Specification.
- [12] Eclipse Foundation, “Eclipse Ponte,” [Online]. Available: <http://www.eclipse.org/ponte/>.
- [13] FIWARE, “IoT Broker,” [Online]. Available: <https://catalogue.fiware.org/enablers/iot-broker>.
- [14] FRAUNHOFER FOKUS, “openMTC,” [Online]. Available: <http://www.openmtc.org/>.