FED4FIRE
Federation for FIRE Plus

lavva.io

BELA

Tsipidis Anastasios
Dimitrios Mouratidis

BEnchmarking LAvva's message broker platform performance over extreme IoT applications

14/11/2020

Stage 1 Experiment Review

BELA

- **Experiment Description**
  - Concept and Objectives
  - Background and Motivation
  - Experiment set-up
- **Project Results**
  - Measurements
  - Lessons learned
- **Business Impact**
  - Impact on business
  - Value perceived
- **Feedback**
  - Used resources and tools
  - Added value of Fed4Fire

lavva.io

Experiment Description

lavva.io

BELA

Concept and objectives

IoT is constantly gaining momentum and is now becoming the de-facto of the global tech-manufacturers that are considered drivers of innovation and technology.
Since IoT is gaining more and more traction into our daily lives there is a much higher need for live data transfers.

However, systems that support operations of such scale, are required to perform under uncertain conditions of failure, while maintaining data redundancy support.

Anadyme Ltd has built a message broker platform, in a cloud-native and cloud-agnostic way that allows data transfer, aggregation and delivery over the cloud in a reliable manner.

In the Bela proposal, our primary objective is to conduct multiple performance experiments under realistic conditions. The data produced will allow us to configure the platform and optimize the overall performance as well as the footprint over the underlying infrastructure optimally for less costly operations.

Background and motivation

Live data streams for businesses, usually means data generation of extreme volumes, broadcasted by thousands of producers, streamlined through a message broker application that resides over the cloud and finally delivered to thousands possible end-customers.

Through BELA experiment we will be able to:

- Investigate which configuration set-up will work best for our customers
- Give us the opportunity to discover and test even more extensive features for the platform

- Reduce latency
- Reduce our infrastructure costs
- Improve platform stability & reliability
- Improve Lavva.io's competitive edge

More specifically we need to compile a detailed performance analysis of Lavva.io platform, under realistic conditions for 1/500/1000 producing devices.

Experiment set-up

Infrastructure

Virtual Wall

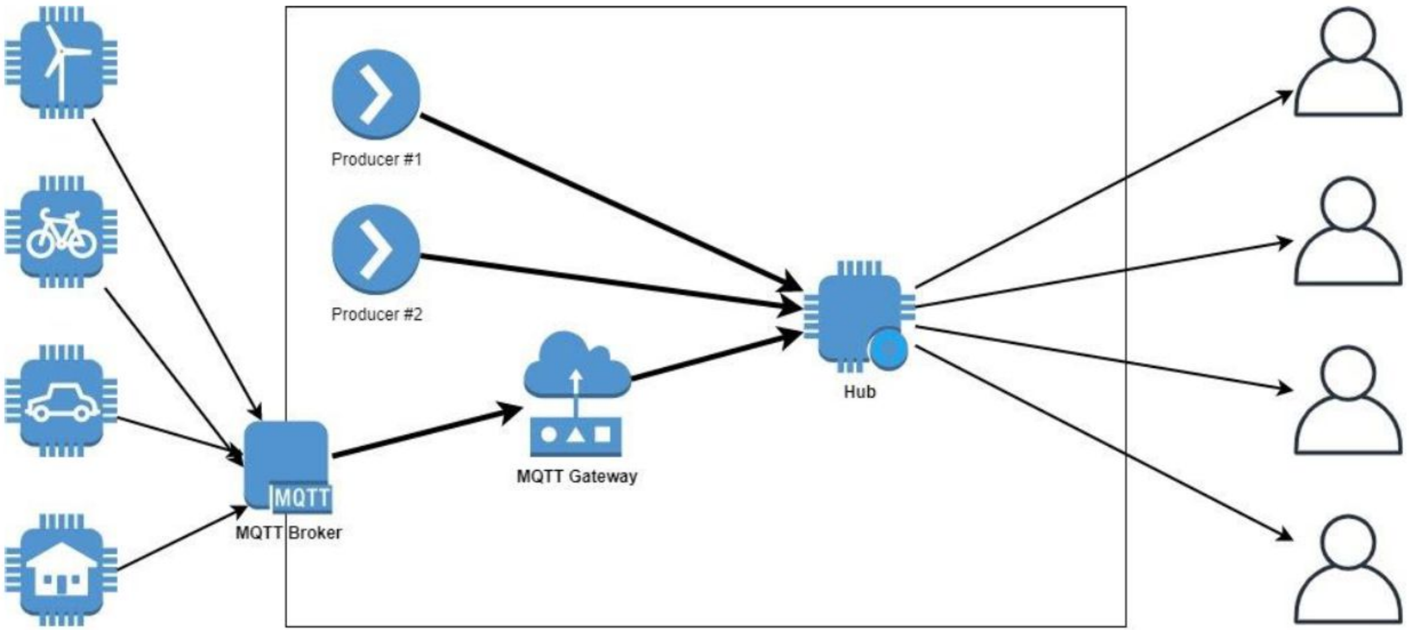
- 1 x Node with 8GB RAM and 8vCPU for Master-node
- 15 x Nodes with 4GB RAM and 4vCPU for our Cluster (Worker-nodes)
- 4 x Nodes with 4GB RAM and 5vCPU for our End-customers
- 1 x Public IPv4 assigned on the Master-node

City of Things

- 5 x nodes with 4GB ram and 4vCPU for the virtual IoT devices
- All nodes include a public IPv4 by default allowing them to be accessed directly

Experiment set-up

Overview scenario we want to cover



Typical end-to-end user scenario

Project Results

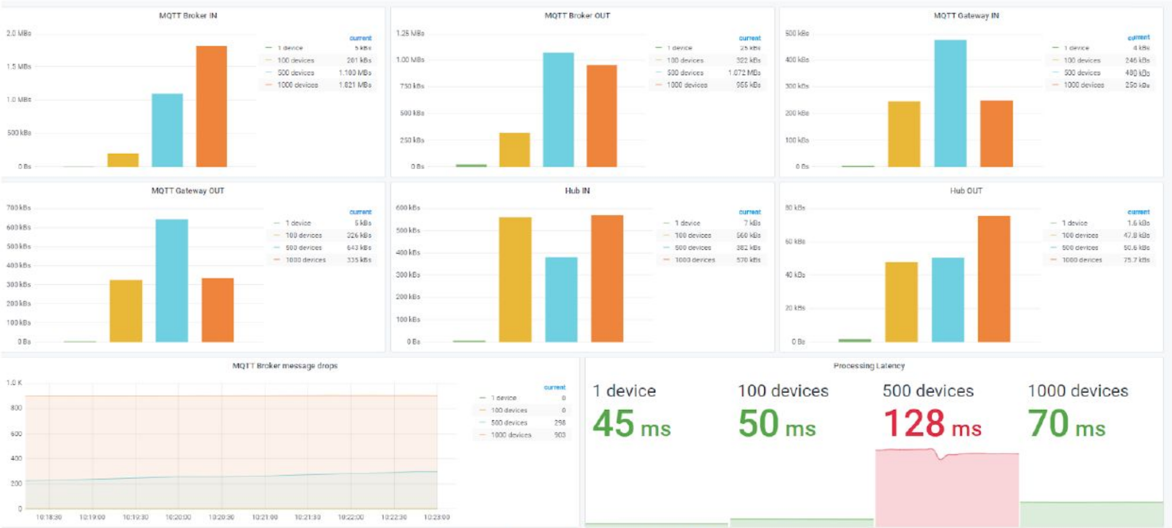
lavva.io

BELA

Measurements

We used Grafana to visualize the gathered KPIs. Selected KPIs are:

- Processing Latency
- Broker Throughput IN
- Broker Throughput OUT
- Gateway IN
- Gateway OUT
- Hub IN
- Message drops



Dashboard for the KPIs

Measurements

Experiment 1

Component	Devices	CPU milliCPU	Memory
		Test Run #1	
Hub	1	10mCPU	34 MiB
Nats	1	10mCPU	32 MiB
Stan	1	10mCPU	30 MiB
MQTT Broker	1	10mCPU	105 MiB
MQTT Gateway	1	10mCPU	35 MiB
		Test Run #2	
Hub	500	75mCPU	42MiB
Nats	500	88mCPU	120MiB
Stan	500	222mCPU	200MiB
MQTT Broker	500	231mCPU	357MiB
MQTT Gateway	500	80mCPU	102MiB
		Test Run #3	
Hub	1000	144mCPU	88MiB
Nats	1000	193mCPU	276M
Stan	1000	450mCPU	366MiB
MQTT Broker	1000	355mCPU	755MiB
MQTT Gateway	1000	177mCPU	132MiB

Metrics that show resource usage per component/device batch

Measurements

Experiment 1

Component	Instances	CPU		Memory	
		Request	Limit	Request	Limit
Hub	2	100 mCPU	200 mCPU	50 MiB	100 MiB
Nats	3	100 mCPU	200 mCPU	100 MiB	300 MiB
Stan	3	100 mCPU	300 mCPU	150 MiB	400 MiB
MQTT Broker - Vernemq	2	200 mCPU	350 mCPU	400 MiB	700 MiB
MQTT Gateway	2	150 mCPU	200 mCPU	100 MiB	150 MiB
Total		1.5vCPU / 1500 mCPU	3vCPU / 3000 mCPU	1.85 Gi / 1850 MiB	4 Gi / 4000 MiB

Resource metrics & Instances for requests and limits (CPU & Memory)

Lessons learned from experiment #1

After understanding initial resource consumption metrics with default configuration for the test runs (per 1/500/1000 devices) and after setting up the resource requests and limits both for CPU and Memory per component, we can infer the following:

- The MQTT Broker wastes much more resources than expected and requires fine-tuning with an end-target that it will release resources faster than it does at this state
- Being restricted by the default CPU/RAM we have applied initial resource limits. This is expected to have major impact on throttling as well as message drops.

Measurements

Experiment 2

QoS 0

Devices	Processing Latency	Broker Throughput IN	Broker Throughput OUT	Gateway IN	Gateway OUT	Hub IN	Message Drops (avg/5mins)
1	45ms	2kBs	13kbs	2.1kbs	2kBs	4.4kBs	0
100	50ms	100kBs	161kbs	126.6kBs	168kBs	322kBs	0
500	128ms	629kBs	643kBs	325.4kBs	438kBs	922kBs	232
1000	70ms	1.233MBs	821kbs	150.3kBs	202kBs	413kBs	882

QoS 1

Devices	Processing Latency	Broker Throuput IN	Broker Throughput OUT	Gateway IN	Gateway OUT	Hub IN	Message Drops (avg/5mins)
1	51ms	2kBs	11kBs	2kbs	2kbs	4.1kBs	0
100	63ms	101kBs	157kBs	147kbs	170kbs	355kBs	0
500	146ms	645kBs	726kbs	410kbs	539kbs	902kBs	172
1000	79ms	833kBs	350kbs	81kbs	109kBs	222kBs	579

Lessons learned

- As expected, Resource Limits have high impact under 500 and 1000 devices. The processing latency increases because the gateway cannot process all the messages on time and as a result, messages are dropped from timeouts.
- QoS 1 has less message drops due to the reliability level of the mode.
- Under both QoS 0 and QoS 1 modes, for 1000 devices the messages are being queued in the broker ready to process but being dropped due to timeouts. Thus, we can see that the processing is much lower because of the message drops

Measurements

Experiment 3

QoS 0

Devices	Processing latency	Broker Throughput IN	Broker Throughput OUT	Gateway IN	Gateway OUT	Hub IN	Message Drops
1	49ms	3kbs	21kb	2kbs	3kb	3kbs	0
100	60ms	130kbs	148kbs	127kbs	166kbs	354kbs	0
500	169ms	601kbs	614kbs	615kbs	807kbs	417kbs	0
1000	96ms	1.217MBs	544kbs	512kbs	702kbs	601kbs	303

QoS 1

Devices	Processing latency	Broker Throuput IN	Broker Throughput out	Gateway IN	Gateway Out	Hub IN	Message Drops
1	44ms	3kbs	21kbs	2kbs	3kbs	3kbs	0
100	61ms	138kbs	150kbs	131kbs	173kbs	288kbs	0
500	77ms	651kbs	645kbs	624kbs	821kbs	412kbs	0
1000	353 ms	1.521kbs	859kbs	476kbs	625kbs	498kbs	135

Lessons learned

- It is noticed that there is less jamming of data on the Gateway IN level, thus processing speed is clearly faster which leads to less message drops. This is achieved due to the traffic being load balanced between the multiple gateways.
- Increasing number of gateways seems to be a great way to unjam the flow of information and thus reduce message drops even further. Unfortunately, we have now capped the resource limits based on the initial assumption of minimal virtual machine available on the market.
- We have managed to drop message drops from 500 devices to zero. Regarding the 1K devices test runs, we have managed to reduce the drop messages by almost 75%, yet there is still room for improvement.

Measurements

Experiment 4

	Instances	CPU		Memory	
		Request	Limit	Request	Limit
MQTT Broker - VerneMQ	2	200 mCPU => 300 mCPU	350 mCPU => 450 mCPU	400 MiB => 450 MiB	700 MiB => 750 MiB
MQTT Gateway	2	150 mCPU => 50 mCPU	200 mCPU = > 100mCPU	100 MiB => 50 MiB	150 MiB => 100 MiB

Recalibrating resource limits

Measurements

Experiment 4

QoS 0

Users	Processing latency	Broker Throughput IN	Broker Throughput OUT	Gateway IN	Gateway OUT	Hub IN	Message Drops
1	41	3kbs	21	3	4	4	0
100	48	217	234kb/s	127	168	290	0
500	105ms	630	629	654	872	438	0
1000	184ms	1.280	842kb/s	825	1.096MB/s	412	178

QoS 1

Users	Processing latency	Broker Throughput IN	Broker Throughput OUT	Gateway IN	Gateway OUT	Hub IN	Message Drops
1	48	3	21	3	4	3	0
100	58	125	138	126	165	291	0
500	80	1.350mb/s	1.251	489	637	508	0
1000	161	1.524	964kb/s	605	795	395	75

Lessons learned

FED4FIRE

- We can see even less message drops because we switched the internal message serialization between our components from JSON to msgpack. MessagePack has proven to be an efficient binary serialization format. It lets you exchange data among multiple languages like JSON but it is a faster process and produced data smaller in size. Small integers are encoded into a single byte, and typical short strings require only one extra byte in addition to the strings.
- By utilizing msgpack we managed to use less resources thus, allowing us to recalibrate them amongst other components of importance. More specifically we managed to assign more resources to the broker which is still heavily resource-hungry and seems to require further optimizations
- Unfortunately, we did not manage to reach 0 message drops for both scenarios but the Bela experiment in general has given us a lot of insights on what to focus and optimize furthermore, apart from the advanced and innovative features that we intend to build in the future.

Business Impact

lavva.io

BELA

Business Impact

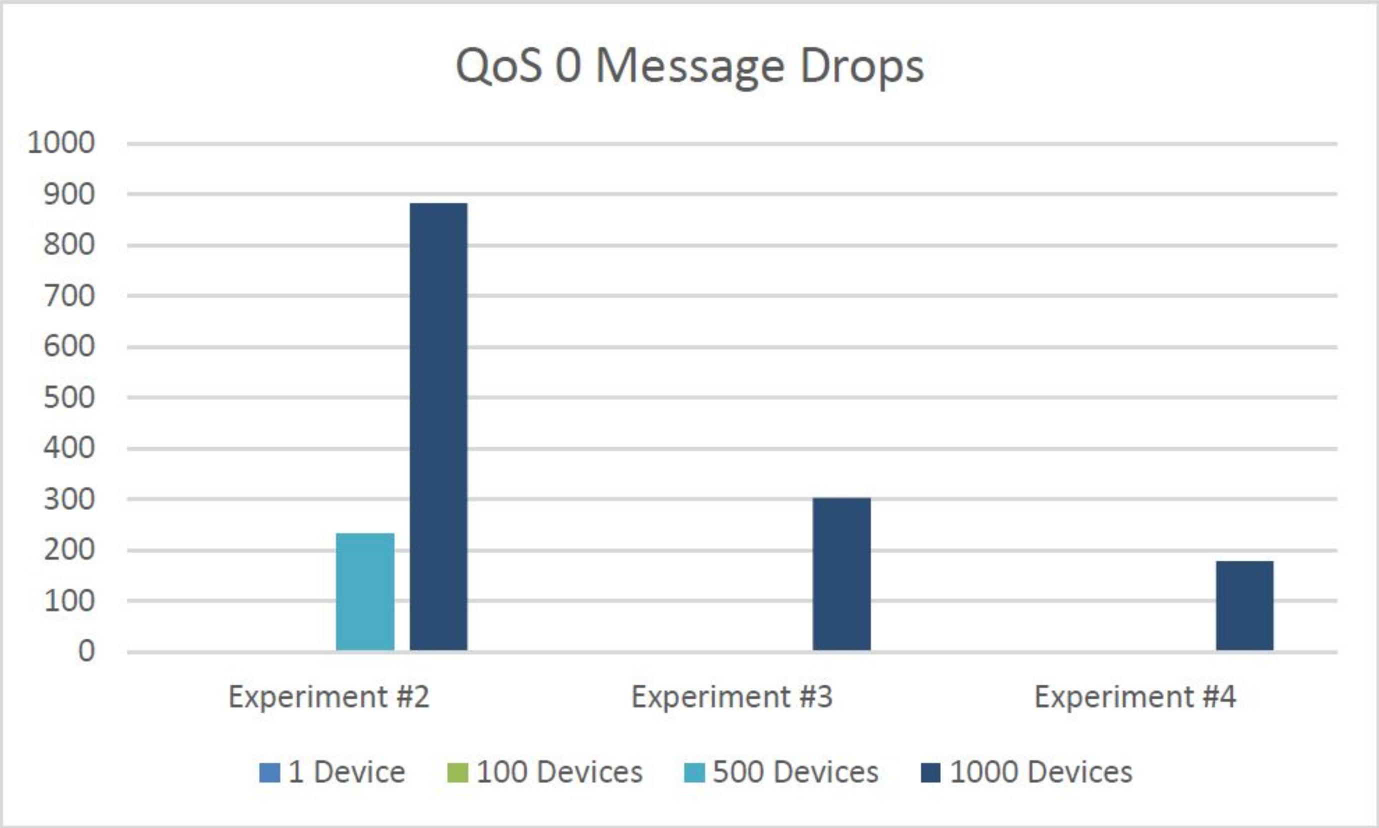
- The first set of experiments was a remarkable learning experience for Anadyme members, where we have managed to:
    - Improve platforms stability
    - Enriched Lavva.io with multiple new features
    - Pivoted our business model to a pay-per-use model
    - Managed to successfully stress-test our platform
    - Through extensive product discovery we have managed to create a clear roadmap plan
    - Improved platform performance and mission critical KPIs

Further experiments

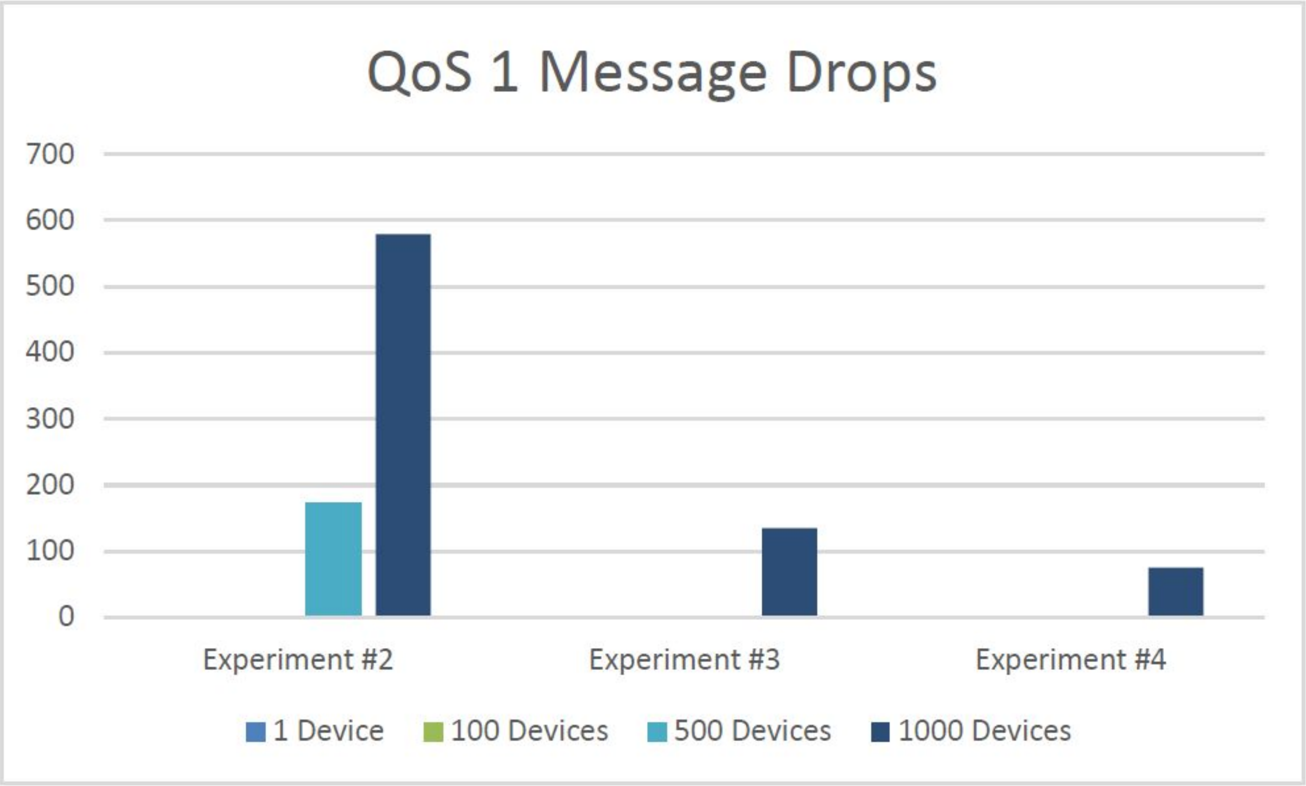
Need to experiment and execute the following:

- Stress test Lavva.io platform under chaotic conditions,
- Add support for new cluster types such as MQTT/RabbitMQ
- Implement Federation support with multi-clustering capabilities
- Implement auto-scaling as well as auto-healing capabilities
- Scale-out and optimize the capabilities of the platform on more VMachine types

Business Impact



Business Impact



Value perceived

There was a huge return comparing the time invested on Stage 1 of the experiment. Some of the things we have acquired are the following:

- **Optimized** the footprint of most of our internal applications
- **Realized** the power of Lavva.io and the performance capabilities of our platform with real, measurable data
- **Gained** understanding of Kubernetes behaviour in realistic conditions as well as utilized Kubernetes features with improved potentials
- **Configured and deployed Lavva.io** as near-perfect application with very high availability and zero downtimes
- **Expanded** Lavva.io with new capabilities and ideas with a clear vision to test them through an innovative Stage-2 experiment (Multi-cloud, Federation/Multi-cluster, Chaos monkey engineering, Usage profiles)
- **Reduced** the costs of operations of Lavva.io which will lead to even lower prices for our customers
- **Tracked** multiple bugs which we fixed
- **Saved mission critical money** that will be invested in key strategic initiatives to serve overall business development

Feedback

lavva.io

BELA

Added value of Fed4FIRE+

Usefulness

1. Saved us a lot of money comparing testing to public cloud
2. IoT resource availability/Realistic scenarios
3. Freedom of experimentation & resource utilization

Key offerings

1. Combined infrastructure types (IoT, Cloud servers etc)
2. Software interfaces to manage infrastructure (JFed)
3. Availability of resources
4. Resources power (CPU, RAM etc)
5. Freedom of customization of resource
6. Ease of experimentation setup
7. Availability of documentation & other resources (highly qualified testbed experts)
8. Realistic experimentation conditions

Directions for improvement

- A web-based JFed version would have been a very nice addition.
- Although email has worked perfectly in our occasion, it is highly possible that a communication platform like slack for instant messages shared with all testers would have been very valuable.
- The wiki page could include more common problems and troubleshooting.
- Infrastructure monitoring tools per node and as a whole. Live updates directly in the platform. Would also be nice to see resource RAM/CPU usage for any of the nodes. And any errors that occur.
- Fed4FIRE+ access to private cloud providers e.g. Google Cloud, Azure or AWS.

BELA

BEnchmarking LAvva's message broker platform performance over extreme IoT applications

WWW.FED4FIRE.EU

Co-funded by the European Union

Co-funded by the Swiss Confederation

This project has received funding from the European Union's Horizon 2020 research and innovation programme, which is co-funded by the European Commission and the Swiss State Secretariat for Education, Research and Innovation, under grant agreement No 732638.