Cloudera Edge Management 1.5.1

EFM Infrastructure and Performance

Date published: 2019-04-15
Date modified: 2023-04-18

CLOUDΞRA

Legal Notice

Contents

Introduction to EFM infrastructure and performance

This documentation is intended to provide guidelines for creating and sizing infrastructure hosting Edge Flow Manager (EFM). This guide contains general information and guidance, covering several use cases. It is important to note that each and every production deployment is different, so you need to find the ideal configuration for your deployment.

In this guide, it is aimed to create and configure an infrastructure to be able to handle 100000 concurrent agents under the same agent class.

> **Important:** Based on the testing, it is concluded that 100k agents can be controlled through the following configurations:

- Cluster size/nodes: A 3-node cluster
- CPU Cores: 8
- Memory: 16 GB
- Network: 12.5 Gbps
- Storage: 200 GB HDD machines

  This is a minimal configuration, so a bigger setup is recommended to provide a safety buffer.

Infrastructure of the setup

Learn about the infrastructure of the Edge Flow Manager (EFM) setup used for this guide.

In this setup, Kubernetes is used to host the EFM infrastructure. The Kubernetes cluster consists of 4 nodes. Each node hosts only one Kubernetes POD to mitigate the noisy neighbor issue.

Details of the Kubernetes nodes are as follows:

- 3rd Generation Intel Xeon Scalable processors with an all-core turbo frequency of 3.5 GHz
- 8 vCPU
- 16 GB RAM
- 12.5 Gbps network bandwidth
- 200 GB disk storage

> **Note:** Installing EFM onto bare metal or virtual machine instances should give similar or better results in terms of performance.

In this setup, EFM is used in a clustered deployment with 3 instances and a standalone relational database instance for data persistence.

Ingress acts as an entry point to the Kubernetes cluster and forwards requests to the EFM cluster. It is hosted as a scalable service. Sizing and configuring the ingress is not in the scope of this guide.

At last but not least, in this setup, a standalone instance is used to host the agent simulation software. Agents are simulated by a small application written in Scala using the Gatling framework. With a large enough instance, you will be able to simulate 100000 concurrent agents.

The following image shows the architecture of the setup:
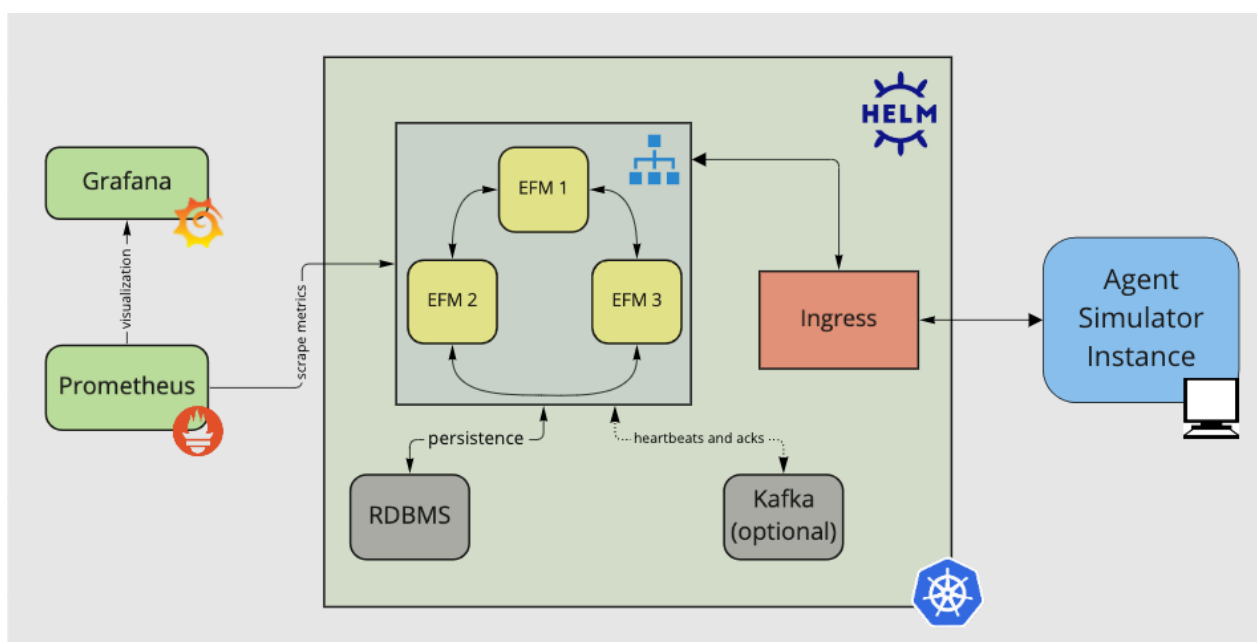
Configurations of the setup

Learn about the configurations of the Edge Flow Manager (EFM) setup used for this guide.

EFM

EFM is configured to run in a clustered mode. There are 3 instances each running in Docker in a Kubernetes POD with 6 cores and 12 GB RAM allocated.

Note that PODs run exclusively on Kubernetes nodes to minimize the noisy neighbor issue.

Java version used in this setup is Adoptium OpenJDK (Temurin) 11.0.15.

Properties	Description
EFM properties	
management.metrics.efm.enabled=true management.metrics.export.prometheus.enabled=true	Metrics collection and metrics export to Prometheus are enabled.
logging.level.com.cloudera.cem.efm=ERROR	Log level is set to ERROR. This is recommended to be set in high-volume production environments as well, as logging has an impact on performance and also logs can fill up the disks easily.
efm.db.maxConnections=150	Database connections' maximum value is increased from 50 to 150 to be able to serve the increased number of database connection requests in some circumstances. Note that this is a per instance property which means that the EFM cluster will have 450 connections in total against the single database.
efm.server.jetty.threads.max=600	The maximum value of Jetty thread number is increased from 200 (default) to 600. This gives EFM some buffer to have enough threads for new connections if there are some straggler requests.
efm.event.maxAgeToKeep.debug=0	Retention period for debug level events is set to 0. In high volume deployments, there will be thousands of events generated in every minute, which will cause the in-memory event store to fill up.
JVM properties	

Properties	Description
-XX:+UseG1GC -XX:+UseStringDeduplication -XX:+ParallelRefProcEnabled	The recommended garbage collection is G1 on Java11. On Java8, CMS garbage collector is recommended. Ensure that you always use the latest Java Update to have all the fixes and backports.
-Xms8g -Xmx8g	Heap memory is configured to 8 GB. It is recommended to set both the initial and the maximum size to the same value. Note that only 8 GB is allocated for the heap while the POD's available memory is 12 GB. The reason behind this is besides the heap, there are other memory areas and entities which consume space like the metaspace, network buffers, threads, and so on.

Database

EFM requires a relational database for storing persistent data. Although it can be run with in-memory H2 database, for clustered setups and in-production environments an external database instance is necessary.

MySQL is a popular relational database and supported by EFM. In this setup, MySQL 8.0.28 docker image is used, which is publicly accessible on Docker Hub.

There is 1 instance running in a Kubernetes POD with 4 cores and 8 GB RAM allocated.

Note: POD runs exclusively on a Kubernetes node to minimize the noisy neighbor issue.

Configurations	Description
--max_connections=451	Maximum number of connections is increased to 451 to be in sync with EFM configuration. 150 connections are configured per instance for the 3 instance cluster.
--innodb-dedicated-server=ON	For production deployments, this option needs to be set for enabling MySQL to use all the available resources on the hosting instance.

EFM infrastructure test cases

Learn about what happens when agents register and then heartbeat to Edge Flow Manager (EFM).

Prewarming EFM

When a new agent class is created with agents having a manifest unknown to EFM, it is highly recommended to register only a small number of agents before letting the traffic of the total agents on EFM.

The reason behind this is when a new manifest is registered in EFM, it involves a heavy database operation. When many agents (for example, hundreds) concurrently try to register the same manifest, it leads to a race condition, which can cause EFM to hang and refuse serving requests.

Agents registering to EFM

Learn about the estimated number of requests, latencies, CPU usage, memory usage, and network usage when agents register to Edge Flow Manager (EFM).

In this scenario, 100000 agents are registering into EFM under the same agent class. 120 new agents are registered every second. Registering the agents takes 833 seconds. Heartbeat interval is 60 seconds. Each agent heartbeats 14 times. This results in nearly 1800 heartbeat TPS maximum. After an agent sends all of the 14 heartbeats, it does not send more so after the heartbeat TPS reaches its maximum. The heartbeat TPS then starts to decrease slowly.

The goal of the test is to demonstrate that all agents can be registered successfully with the given configuration.

Number of requests

You can see, in the following images, that the number of active agents builds up in 15 minutes until it reaches the maximum. At the top, EFM is under the load of approximately 1900 requests per second - 600 heartbeats and 40 acknowledgements per agent per second.
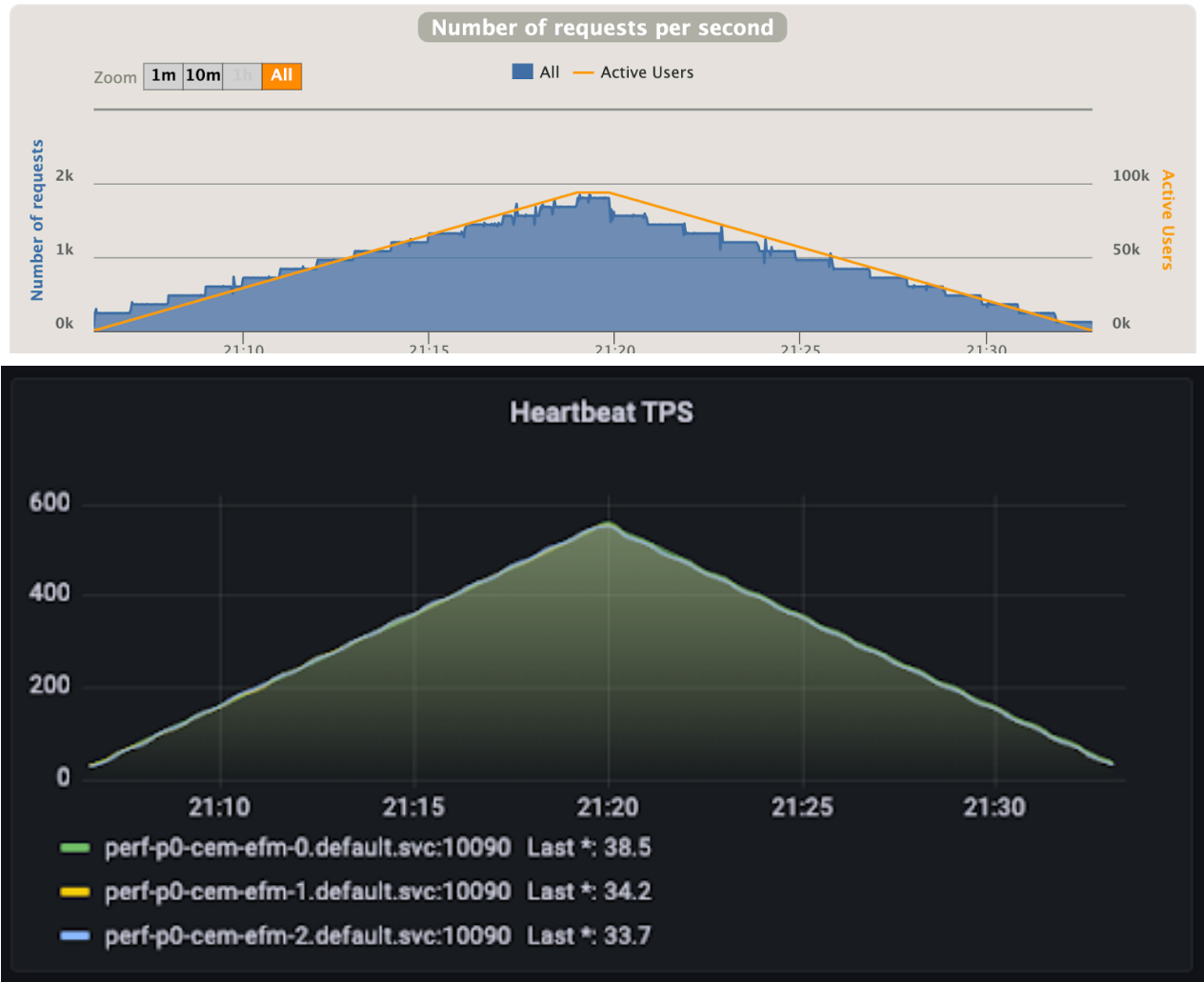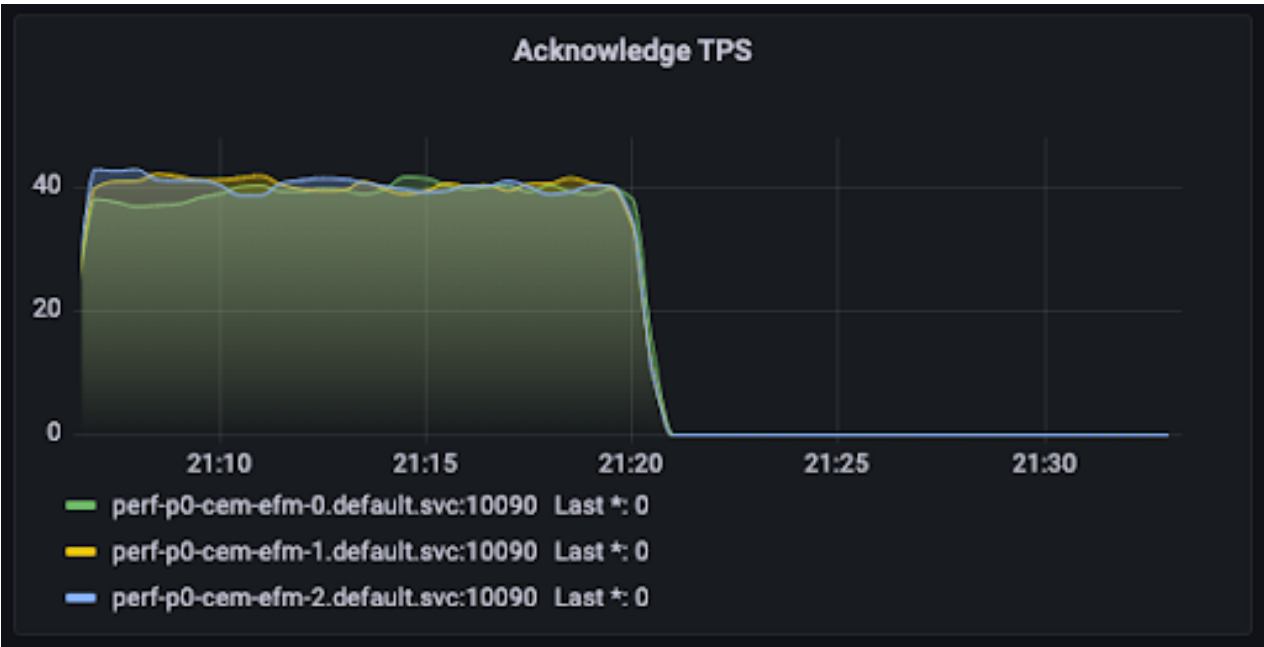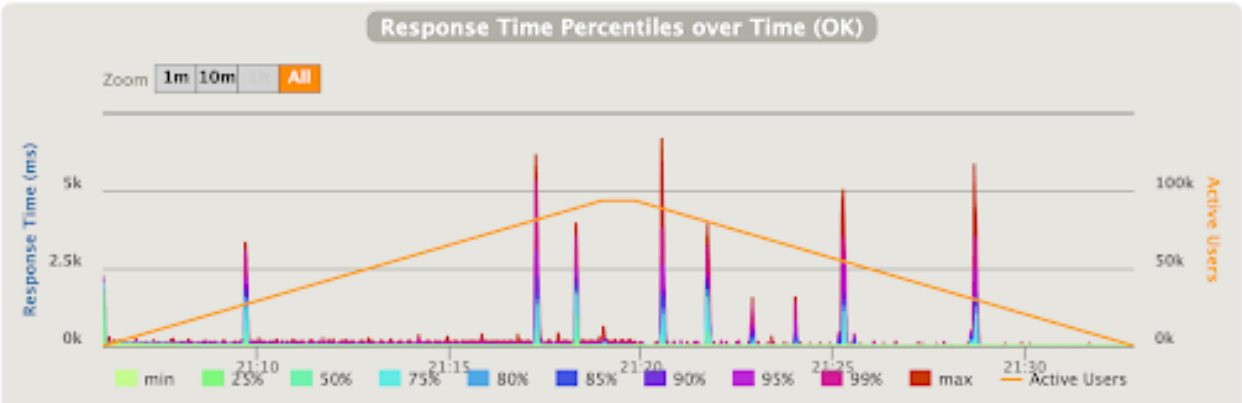
Latencies

Most of the time latencies are below 100 ms with peaking up in 5 seconds.

The first chart shows the full latency including the network latency, while the second chart shows results without the network latency.

CPU usage

CPU usage is measured in two ways. The first chart shows the JVM level metrics, and the second chart shows values measured by Kubernetes. The results are in line with each other. Since you have 6 CPUs allocated, the peak CPU consumption of approximately 35 percent is equal to 2 CPU cores. You can observe higher CPU usage on one specific EFM node. This is because one node is always selected as master node, which has some extra tasks resulting in higher CPU usage.
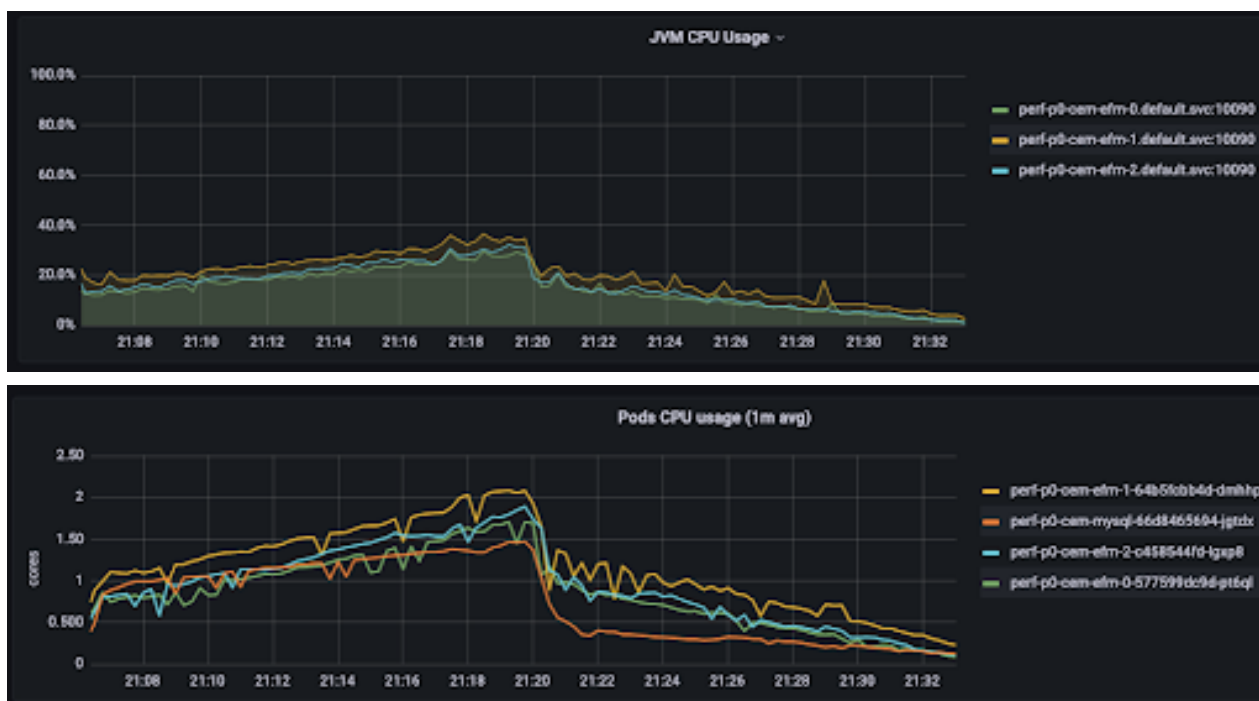
Memory usage

The heap consumption fluctuates between 1.5 GB and 7.5 GB with maximum Garbage Collection (GC) pauses of 100 ms; sometimes peaking up to 400 ms.
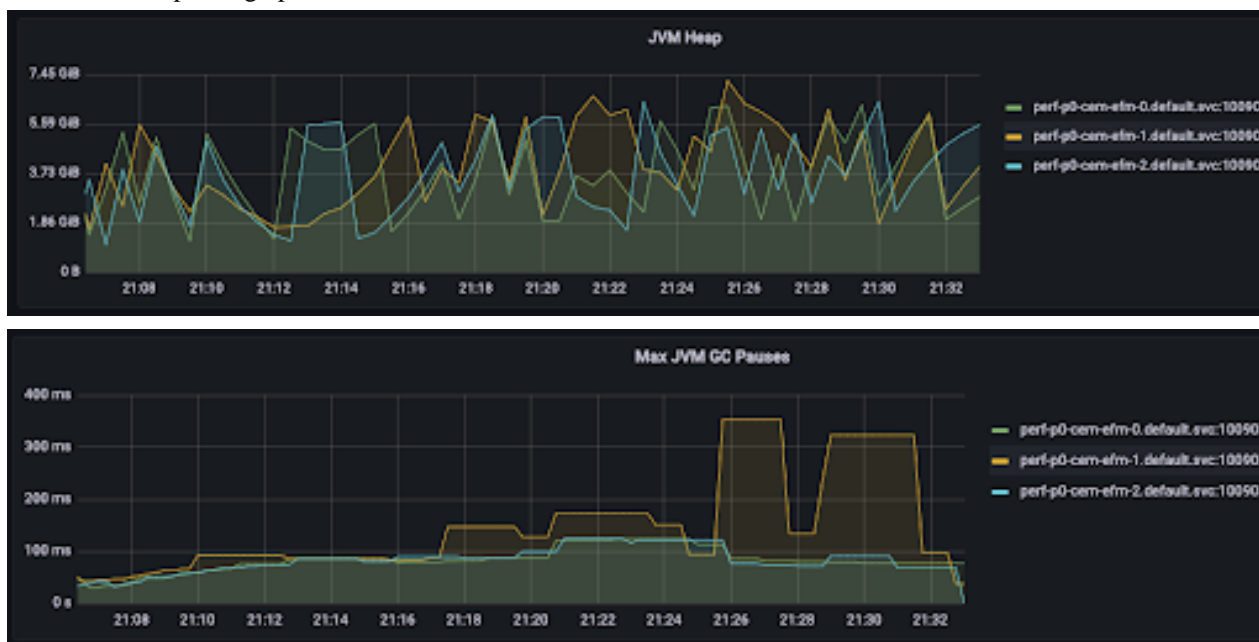
Network usage

The aggregated network in-bound rate tops out at approximately 100 MB/s, 30 MB/s per EFM node, and 5 MB/s for the database instance. Higher network traffic can be observed in the first half of the test. The reason behind this is when the agents are registering to EFM with sending a lightweight heartbeat, EFM requests the agents to send the agent manifest. Agent manifests are an order of magnitude higher in size than lightweight heartbeats.

In the second half of the test all agents are registered, and only lightweight heartbeats are sent. Output traffic rate is lower than 10 MB/s at the peak.
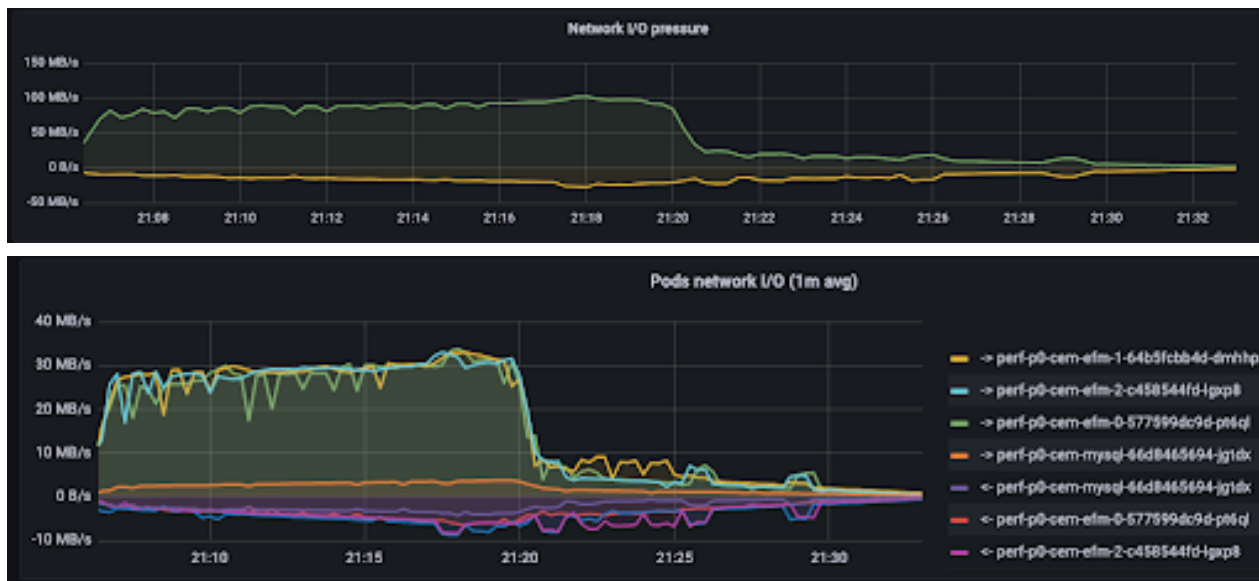

Agents heartbeating to EFM

Learn about the estimated number of requests, latencies, CPU usage, memory usage, and network usage when agents heartbeat to Edge Flow Manager (EFM).

After all agents have successfully registered in the previous test, the same agent class and the agents are reused in this scenario. Agents are brought online in 1 minute with a heartbeat interval of 60 seconds. Each agent heartbeats 25 times and then exits. Because the agents are brought online almost at the same time, you see a steep increase in the heartbeat TPS at the start of the test and a steep decrease in the heartbeat TPS at the end of the test.

The goal of the test is to see that the system is able to serve the agents' heartbeat requests under maximum TPS during a longer period of time.

Number of requests

You can see, in the following images, the number of active agents builds up in 1 minute until it reaches the maximum. At the top, EFM is under the load of approximately 1800 requests per second - 600 heartbeats per agent per second.
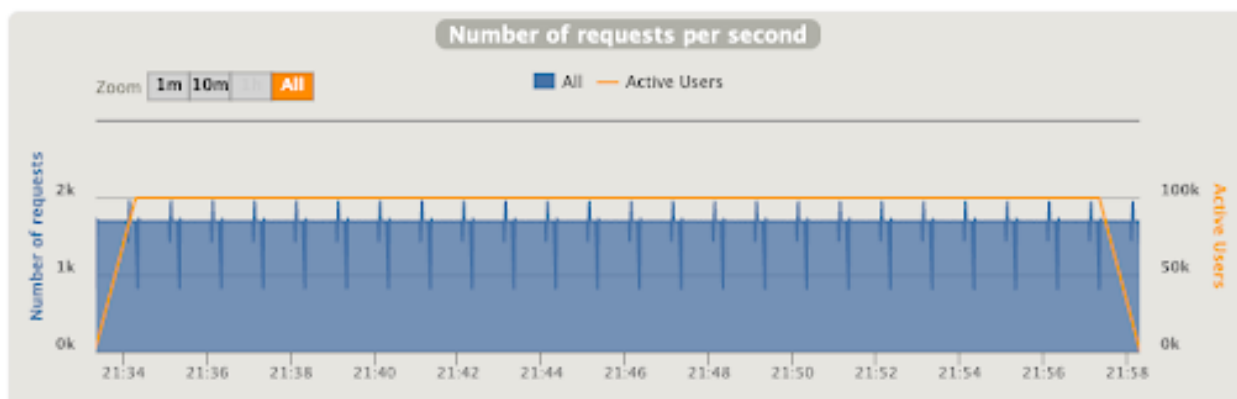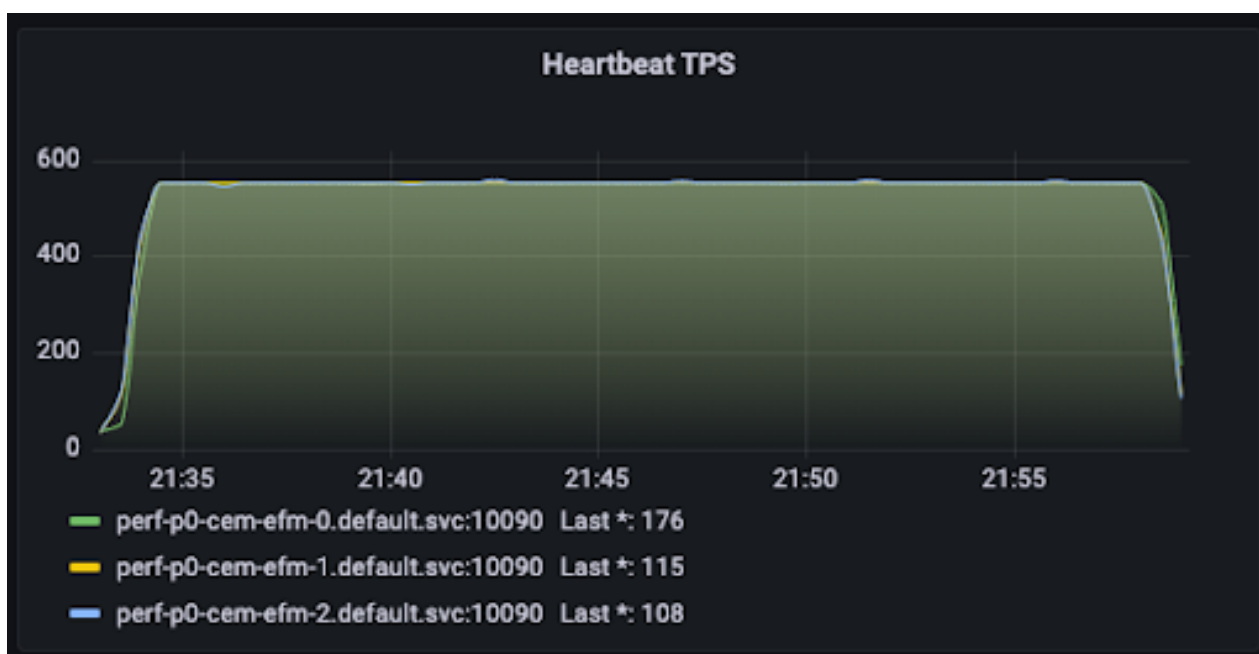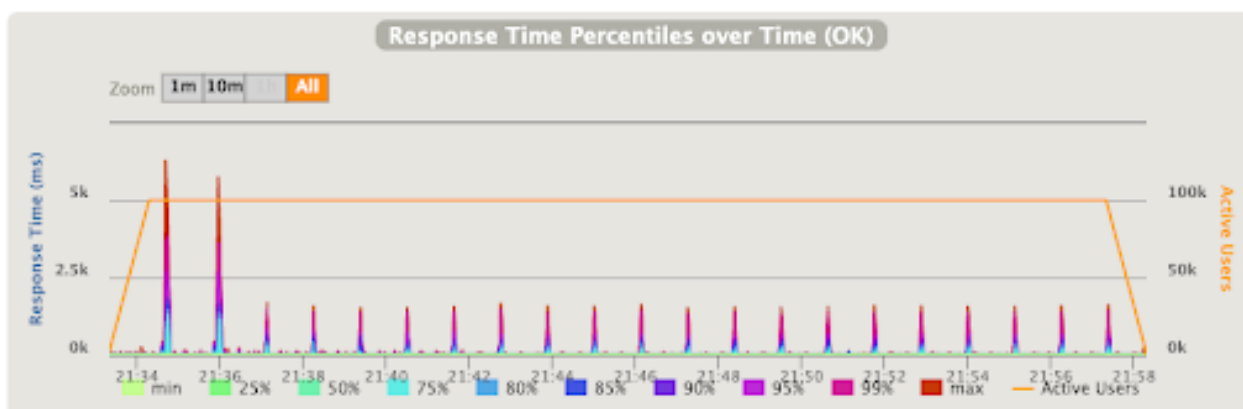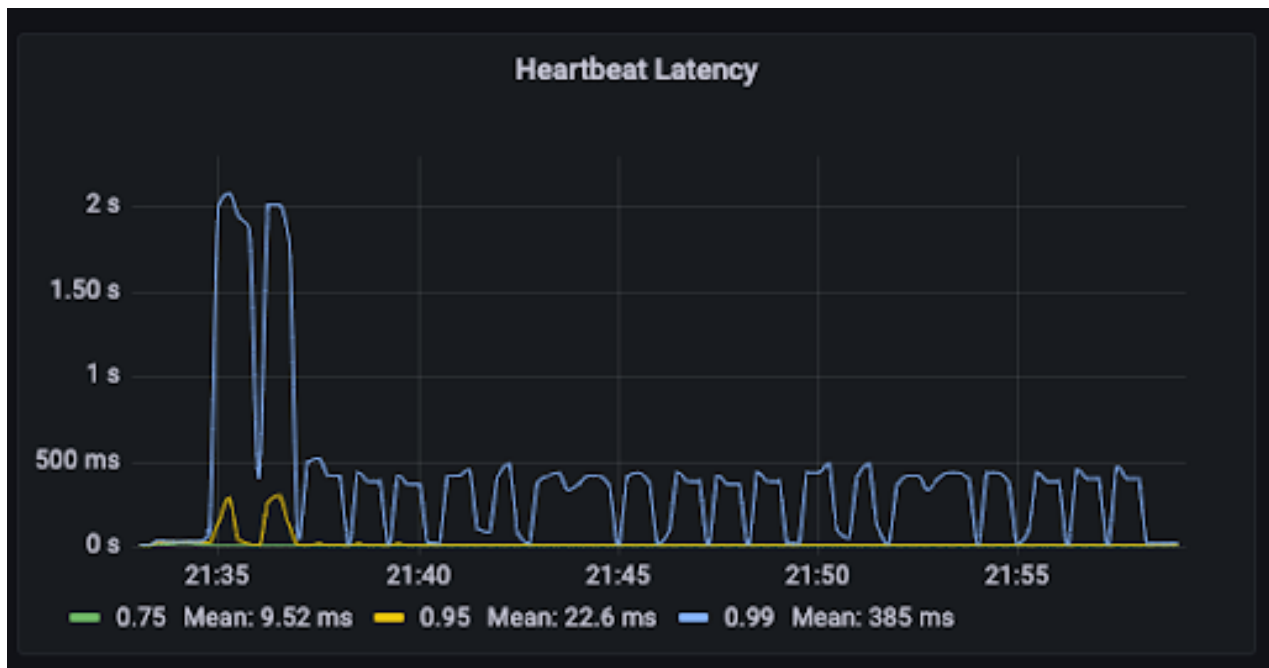
Latencies

Most of the time latencies are below 100 ms with peaking up in 2 seconds.

The first chart shows the full latency including the network latency, while the second chart shows results without the network latency.

CPU usage

CPU usage is measured in two ways. The first chart shows the JVM level metrics, and the second chart shows values measured by Kubernetes. The results are in line with each other. Since you have 6 CPUs allocated, the peak CPU consumption of approximately 16-25 percent is equal to 1.0-1.5 CPU cores.

In this case as well, you can observe higher CPU utilization for the master node.

Memory usage

The heap consumption fluctuates between 1.5 GB and 7.5 GB with maximum garbage collection pauses of 50 ms - 100 ms; sometimes peaking up to 400 ms.

You can observe that compared to the previous test case there is less pressure on the Garbage Collection (GC) in this test case. Agent registration is more expensive in terms of resources than just serving heartbeats.
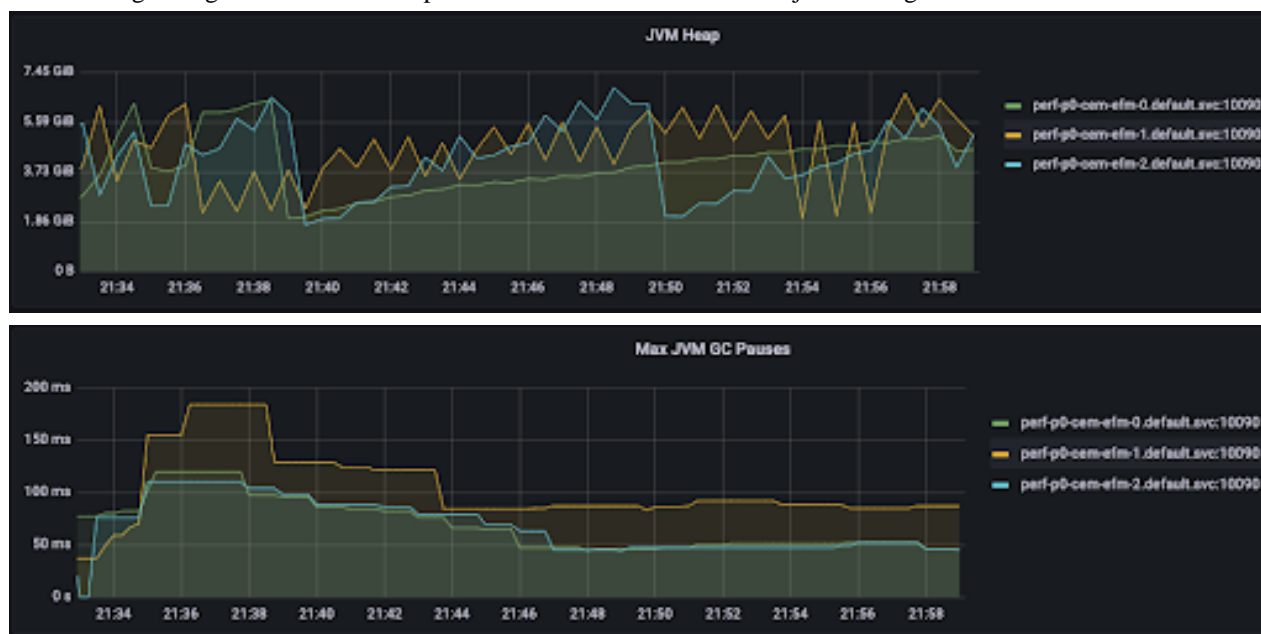
Network usage

The aggregated network inbound rate tops out at approximately 25 MB/s, 8 MB/s per EFM node, and 1 MB/s for the database instance. Output traffic rate is lower than 10 MB/s at the peak.

The lower network usage is expected in this case. Only heartbeat messages without the agent manifest part are sent in this test, whereas in the previous test each agent sends one acknowledgement including the agent manifest, which is approximately 5 times larger.
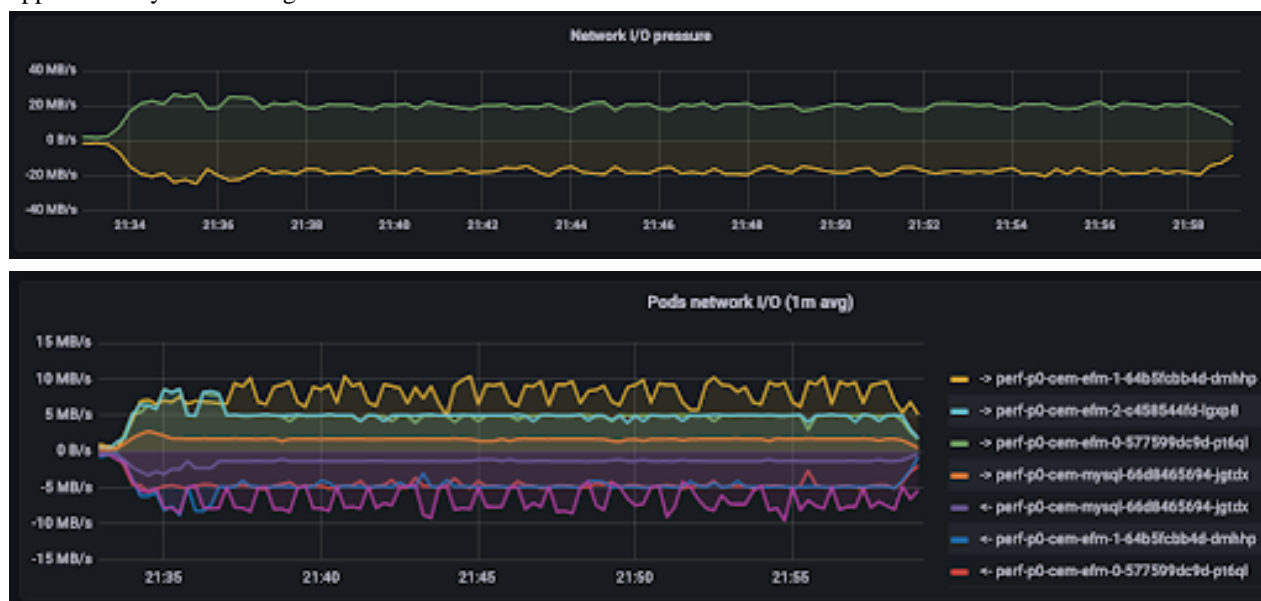
EFM infrastructure test cases with different configurations

Learn about the consequences that happen in case you change the configurations of the earlier test cases in the setup used in this guide.

A set of deployments with differences in the configurations are tested in the following scenarios, which affects the performance. The same tests are executed against these deployments as for the default setup.

Only the differences in the results are discussed here.

Using PostgreSQL instead of MySQL for relational database

Learn what happens if you use PostgreSQL instead of MySQL as the relational database.

The goal of this test is to validate whether you can achieve the same performance with the other frequently used relational database. PostgreSQL is used with default parameters; the only parameter changed is the maximum number of connections, like in the case of MySQL.
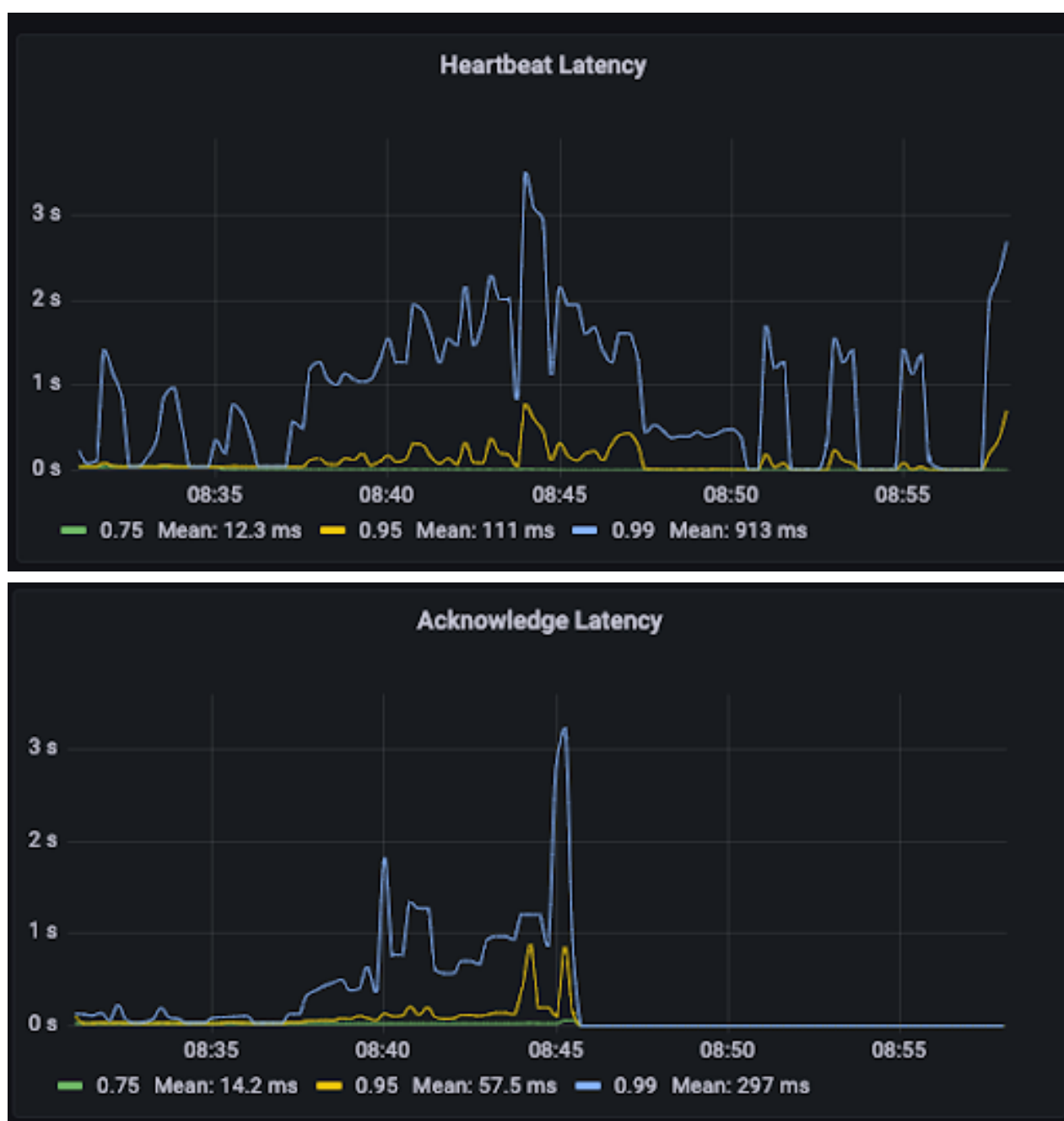
Agents registering to EFM

Learn about the estimated latencies and CPU usage when agents register to Edge Flow Manager (EFM) with PostgreSQL used as the relational database.
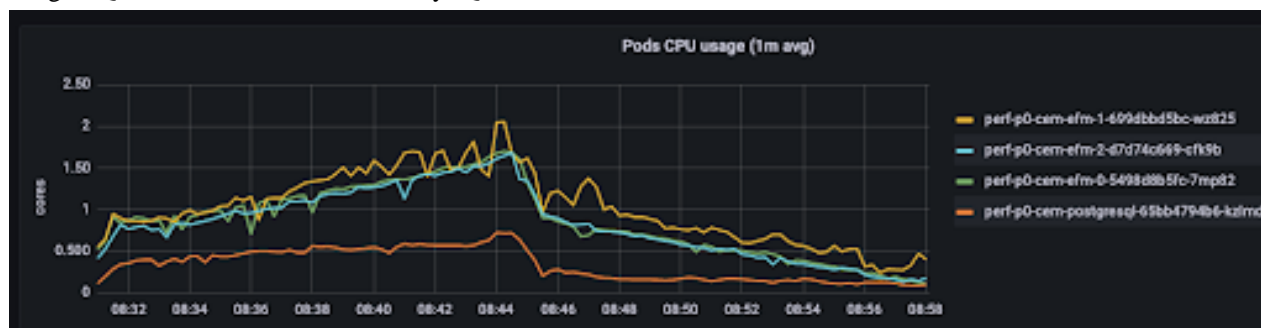
Latencies

The latencies are slightly higher compared to MySQL, but are in the same range.
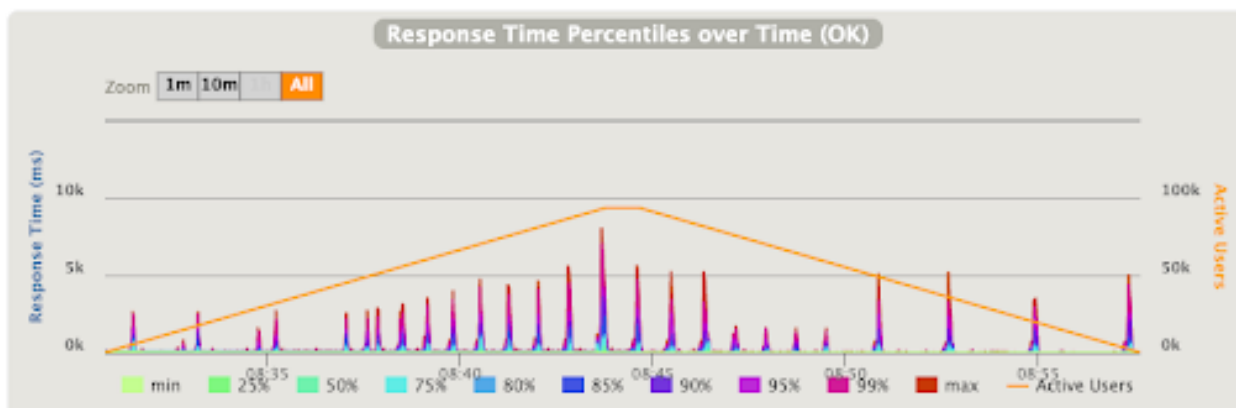
CPU usage

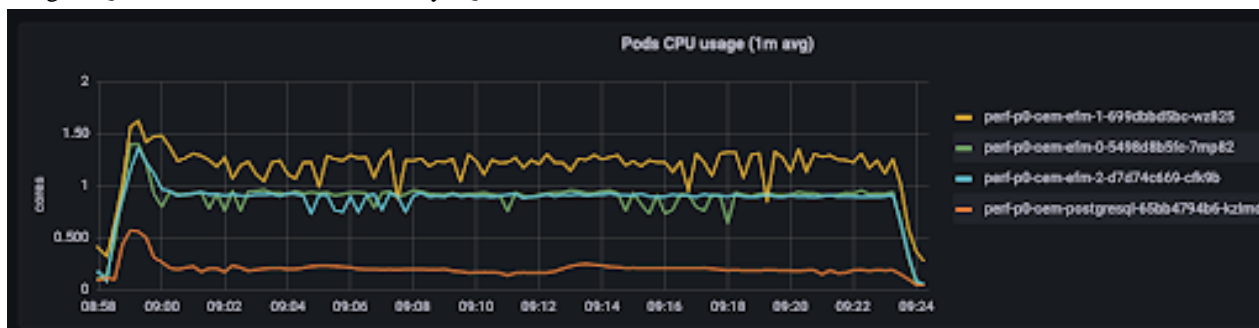PosgreSQL consumes less CPU than MySQL.

Agents heartbeating to EFM

Learn about the estimated CPU usage when agents heartbeat to Edge Flow Manager (EFM) with PostgreSQL used as the relational database.

CPU usage

PosgreSQL consumes less CPU than MySQL.

Takeaways

As per the result, you can achieve the same performance with MySQL or PostgreSQL, with slight differences.

> **Note:** Tuning of the database was not amongst the goals of the test. Both databases were used with default configurations.

Forwarding heartbeats and acknowledge requests to Kafka

EFM integrates with Kafka to store heartbeats and acknowledgements, so those can be used for further analysis.
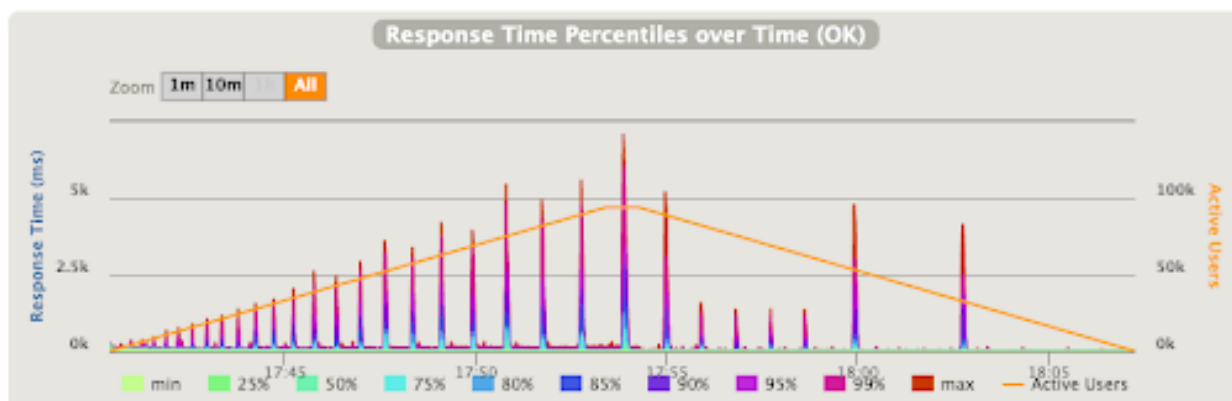
The goal of the test is to see the level of performance degradation due to the extra operation overhead done by EFM.
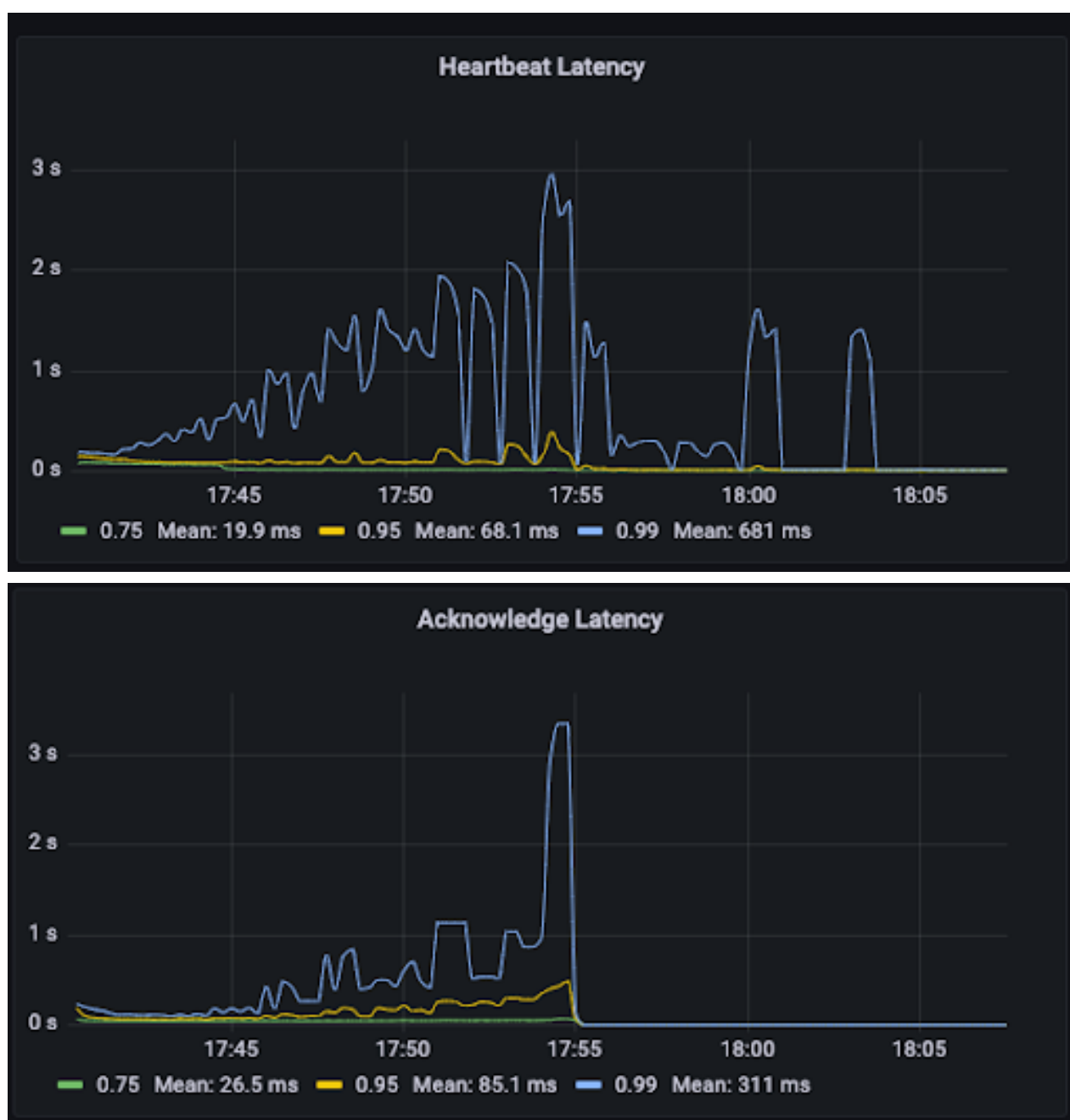
Agents registering to EFM

Learn about the estimated latencies and CPU usage when agents register to Edge Flow Manager (EFM) which is integrated with Kafka to store heartbeats and acknowledge requests.

Latencies

Latencies are a bit higher compared to when Kafka is not present, but still within the range of 100 ms and 5 seconds.

Heartbeat Latency

— 0.75 Mean: 19.9 ms — 0.95 Mean: 68.1 ms — 0.99 Mean: 681 ms

Acknowledge Latency

— 0.75 Mean: 26.5 ms — 0.95 Mean: 85.1 ms — 0.99 Mean: 311 ms

CPU usage

CPU usage is higher as expected. As per the JVM, at the maximum level, approximately 50% of the available CPU is used, which is 3 cores. Compared to the configuration without Kafka that means 50% more CPU usage.
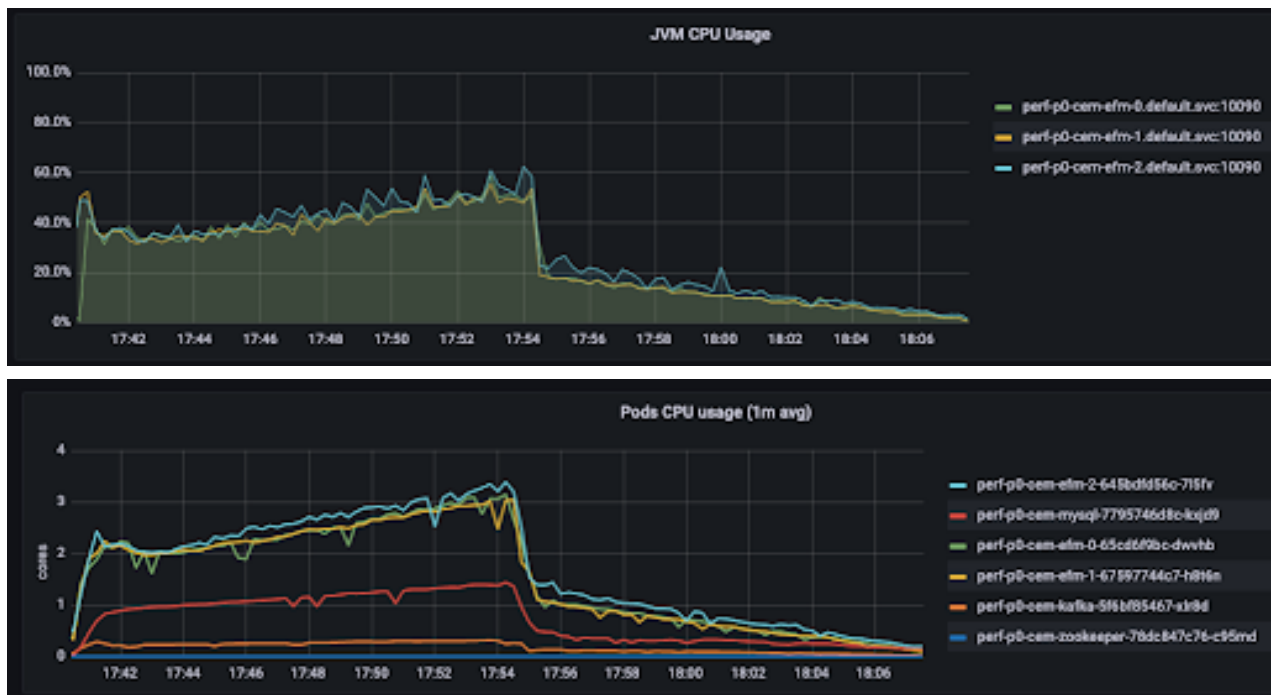
Agents heartbeating to EFM

Learn about the estimated CPU usage when agents heartbeat to Edge Flow Manager (EFM) which is integrated with Kafka to store heartbeats and acknowledge requests.

CPU usage

CPU usage is slightly higher, but the difference is negligible.

Takeaways

According to the test, you should expect increased CPU consumption for EFM when Kafka integration is enabled.

Compressed traffic through C2 protocol between EFM and agents

MiNiFi C++ agents can be configured to compress the requests sent to EFM. With this option the network traffic can be reduced.

The goal of the test is to check the performance characteristics of EFM when it needs to process compressed traffic.
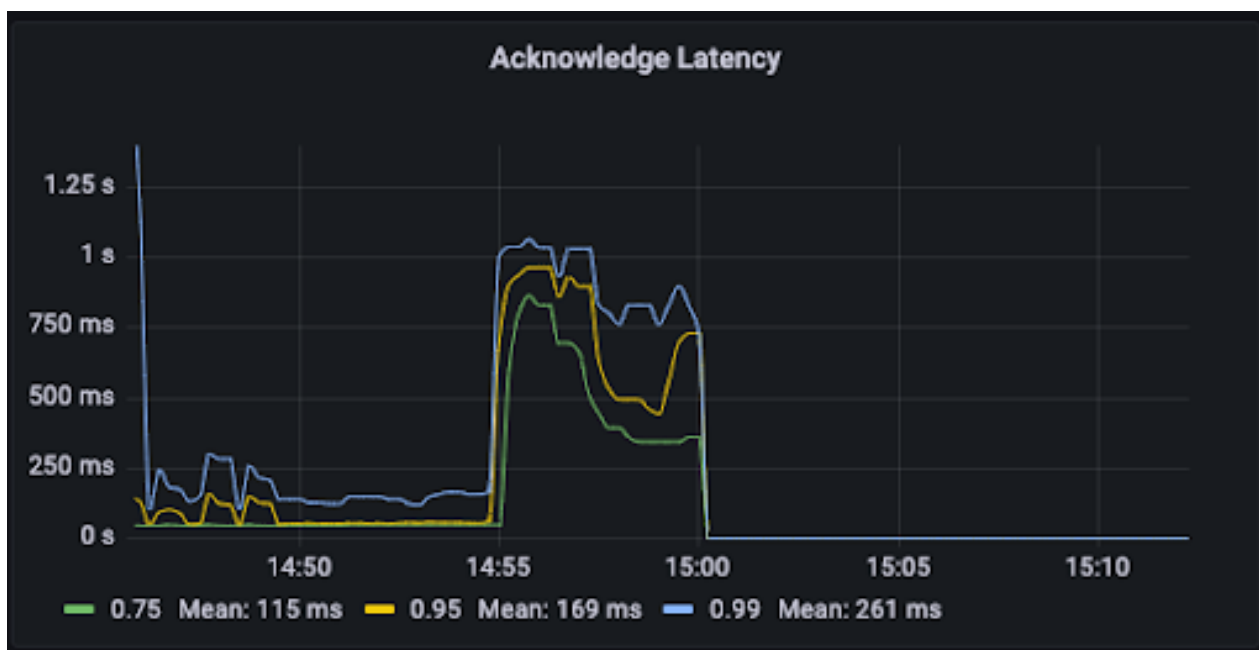
Agents registering to EFM

Learn about the estimated latencies, CPU usage, memory usage, and network usage when agents register to Edge Flow Manager (EFM) which is configured to process compressed traffic.
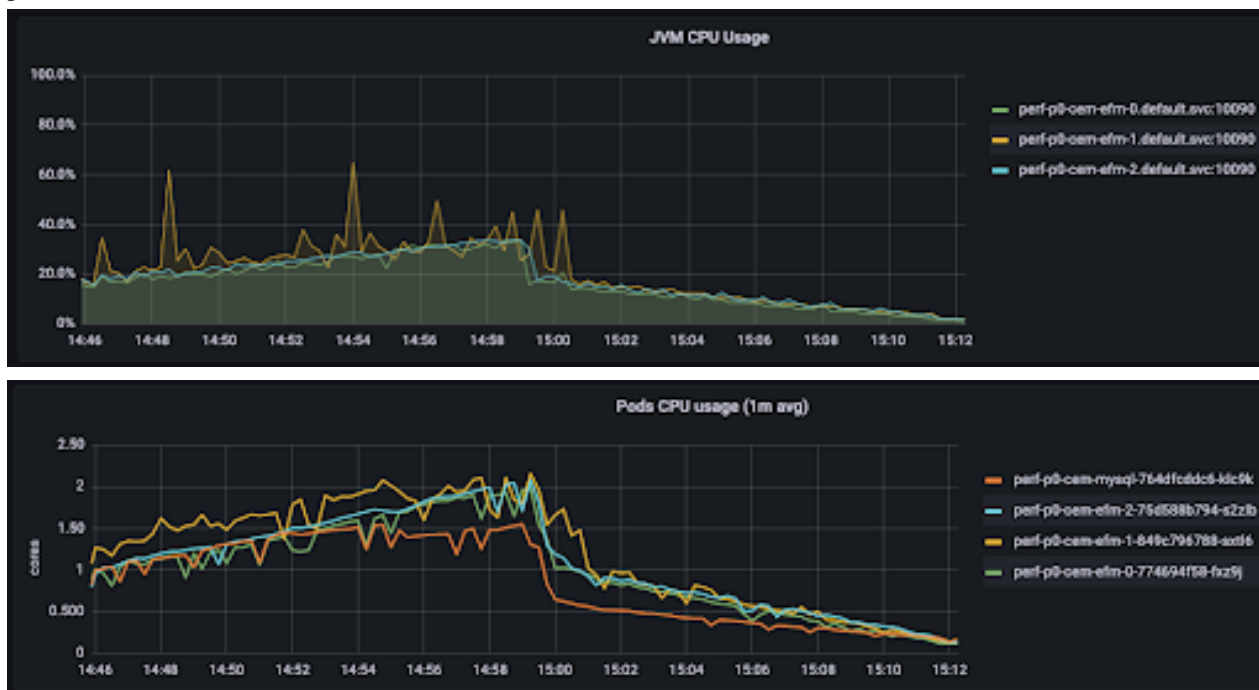
Latencies

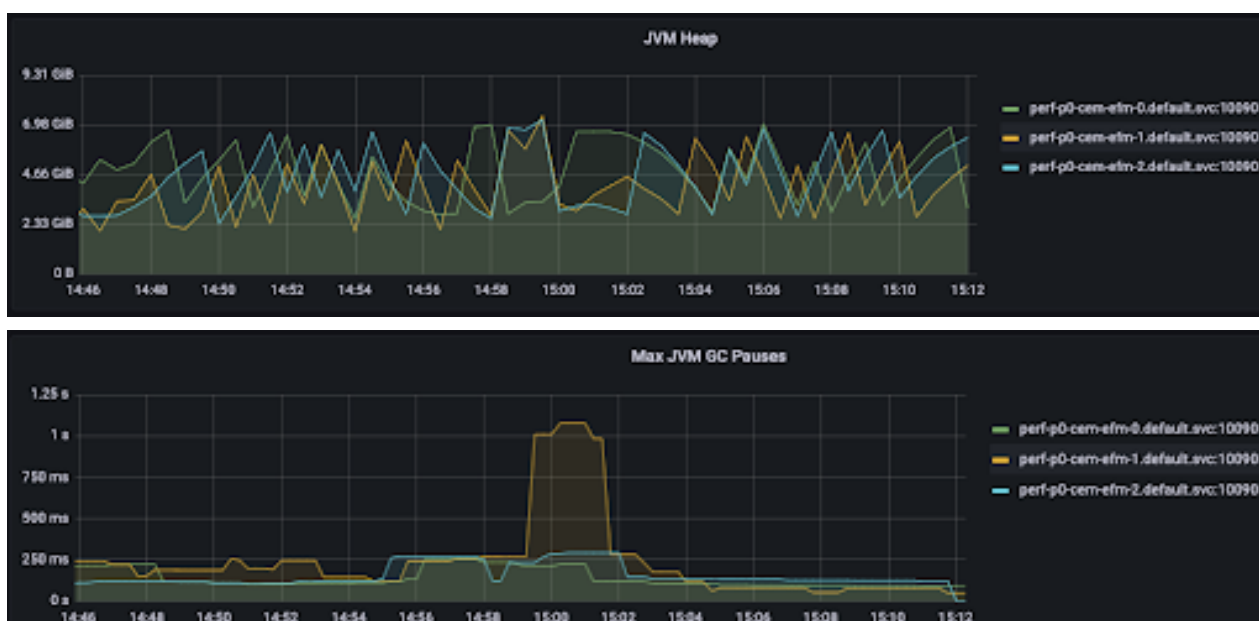The latencies are slightly higher, but are in the same range.
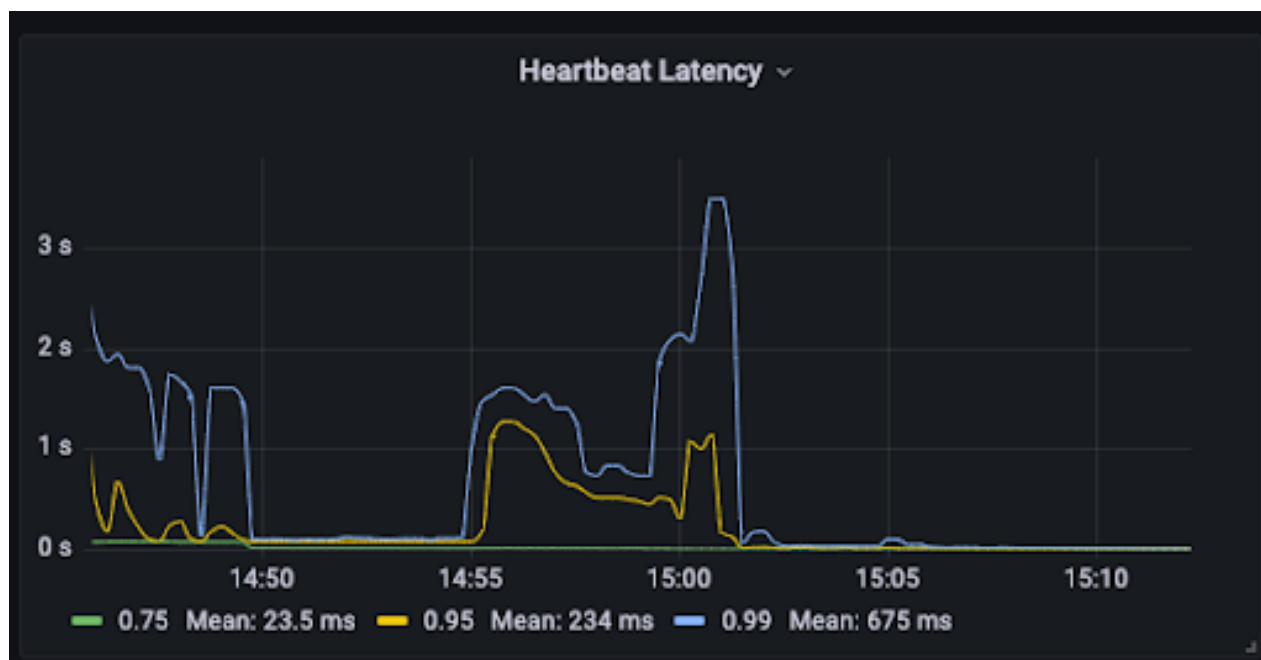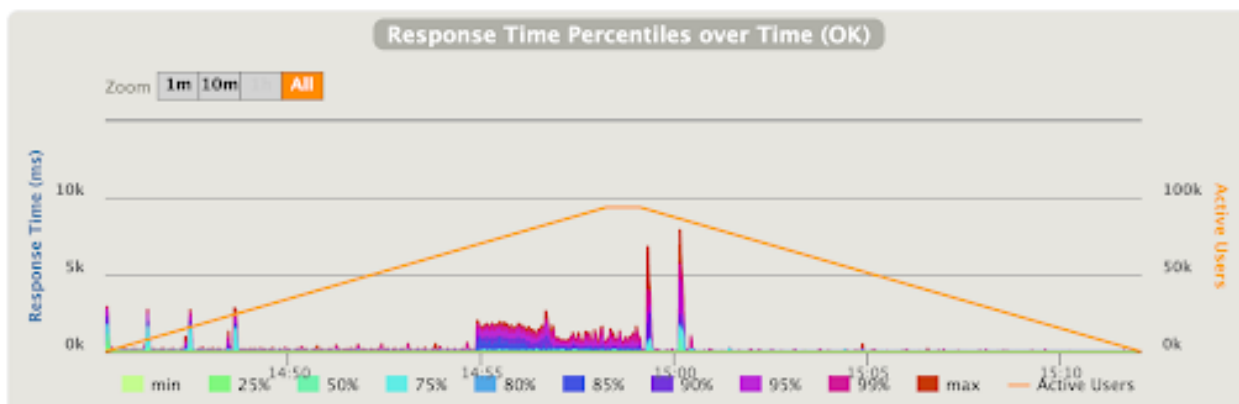
CPU usage

There is a slight increase in the CPU usage compared to the baseline test case. CPU usage is 5-15% more, with higher peaks.

Memory usage

The heap consumption fluctuates between 2.3 GB and 7.5 GB with maximum Garbage Collection (GC) pauses of 200 ms sometimes peaking up to 1 second.

Network usage

Accumulated in-bound network traffic is between 20 and 30 MB/s. Compared to the baseline test case, there is 60-75% less pressure on the network I/O.
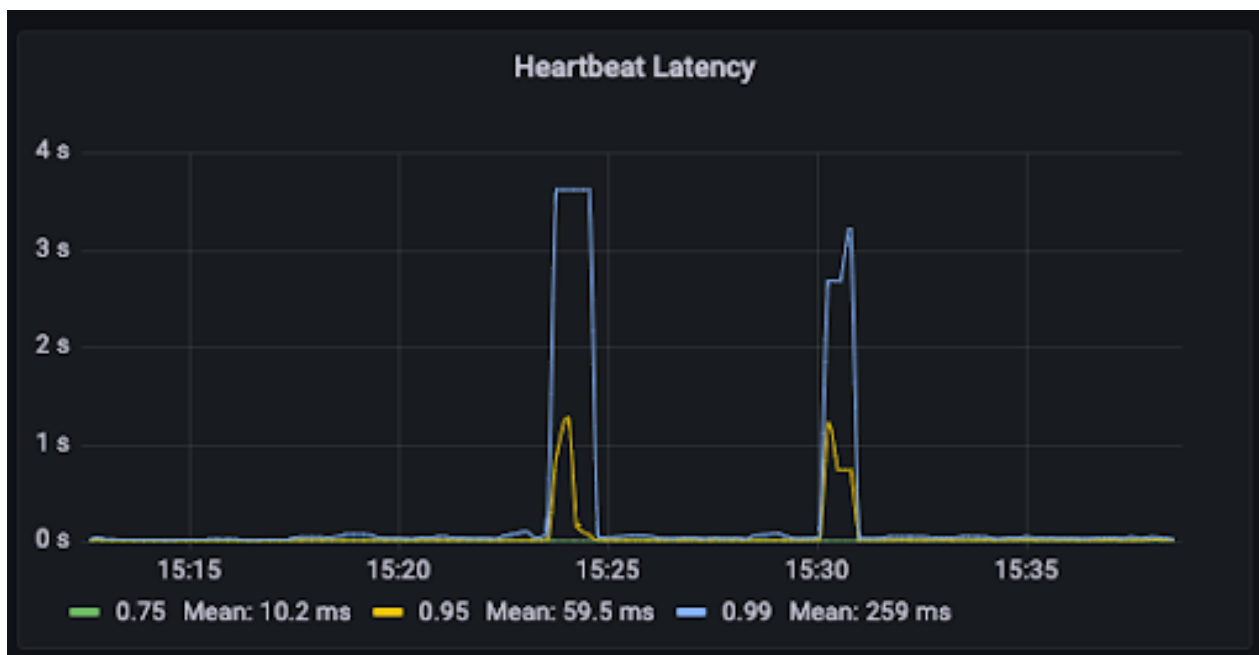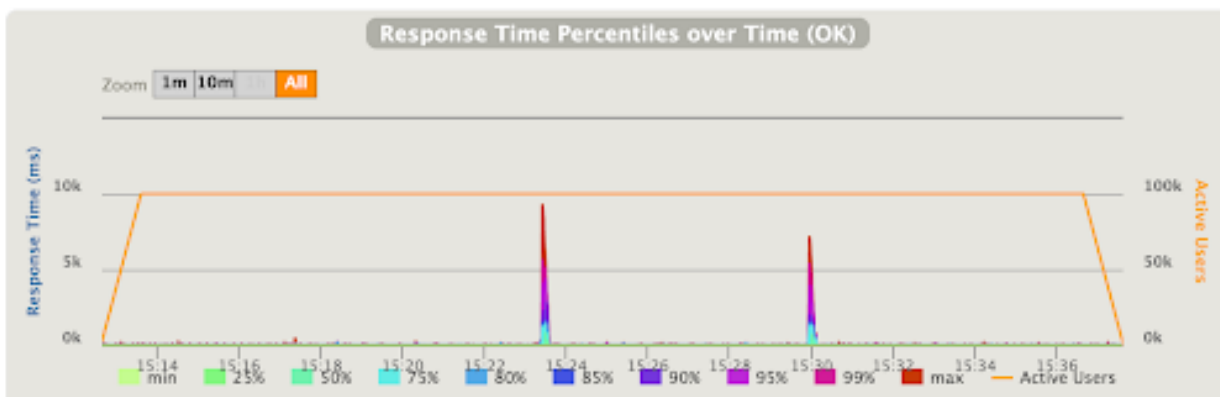

Agents heartbeating to EFM

Learn about the estimated latencies, CPU usage, memory usage, and network usage when agents heartbeat to Edge Flow Manager (EFM) which is configured to process compressed traffic.
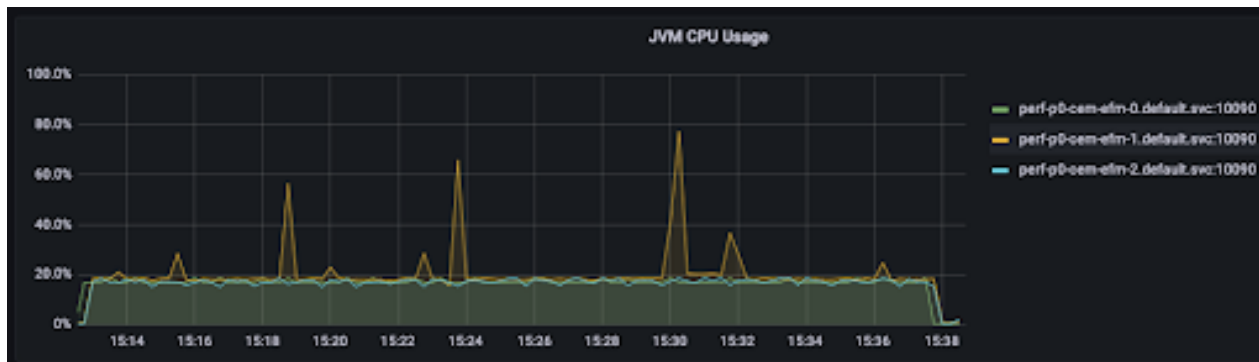
Latencies

The latencies are slightly lower, but with higher peaks.

CPU usage

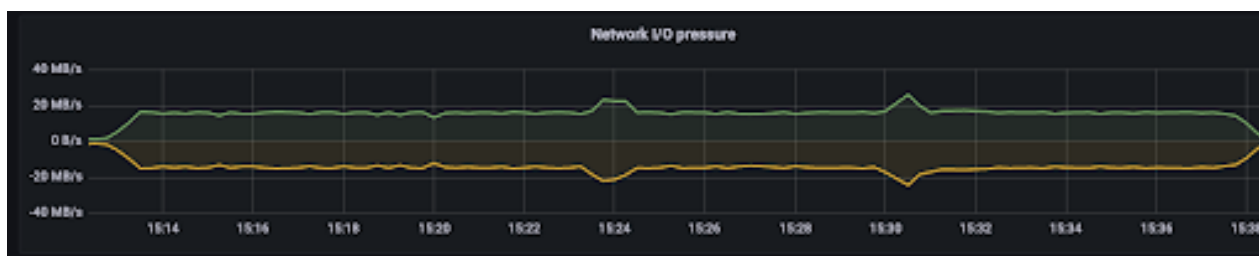CPU usage is in the same range, but with higher peaks.

Memory usage

Memory usage is slightly higher compared to the baseline test, the Garbage Collection (GC) latency is also higher with higher peaks.

Network usage

Network usage is slightly lower; between 18-20 MB/s compared to the baseline 20-25 MB/s, that is a 10-20 percent decrease in the traffic.

Takeaways

Enabling compressed traffic between EFM and the agents slightly increases CPU utilization and Garbage Collection (GC) pressure, but decreases the network traffic.

Also, it is worth considering to increase the amount of heap to decrease the GC pressure.

Hazelcast memory considerations

Learn about what you need to consider for Hazelcast's memory consumption when you allocate memory for EFM.

Hazelcast is an in-memory data grid, used as cache manager for EFM. Hazelcast runs in an embedded mode, in the same JVM as EFM.

Hazelcast's memory consumption has to be considered as well when you allocate memory for EFM.

To demonstrate Hazelcast memory consumption scales linearly with the number of agents, the baseline test is run with 100k and then with 20k agents. The expectation is that 5 times less cache is required in the second case.

Results

	100000 Agents		20000 Agents		Ratio	
Test Case	Total Items	Total Size MB	Total Items	Total Size MB	Total Items	Total Size
Agents registering to EFM	234,000.00	215.00	46,500.00	42.80	19.87%	19.91%
Agents heartbeating to EFM	1,096,000.00	808.00	202,900.00	160.90	18.51%	19.91%

Takeaways

The expectation was to see linear scaling in cache items/sizes which is in line with the results. Cache consumption changes proportionally with the number of agents, for example, 80% less agents result in 80% less cache consumption.