Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 2068278, 15 pages
https://doi.org/10.1155/2018/2068278

Research Article

FMonE: A Flexible Monitoring Solution at the Edge

Álvaro Brandón,[1] **María S. Pérez**,[1] **Jesus Montes**,[1] **and Alberto Sanchez**[2,3]

[1]*Universidad Politecnica de Madrid, Madrid, Spain*
[2]*Universidad Rey Juan Carlos Madrid, Spain*
[3]*Center for Computational Simulation (CCS), Madrid, Spain*

Correspondence should be addressed to Álvaro Brandón; abrandon@fi.upm.es

Received 16 May 2018; Revised 27 September 2018; Accepted 15 October 2018; Published 19 November 2018

Guest Editor: Marco D. Santambrogio

Monitoring has always been a key element on ensuring the performance of complex distributed systems, being a first step to control quality of service, detect anomalies, or make decisions about resource allocation and job scheduling, to name a few. Edge computing is a new type of distributed computing, where data processing is performed by a large number of heterogeneous devices close to the place where the data is generated. Some of the differences between this approach and more traditional architectures, like cloud or high performance computing, are that these devices have low computing power, have unstable connectivity, and are geo-distributed or even mobile. All of these aforementioned characteristics establish new requirements for monitoring tools, such as customized monitoring workflows or choosing different back-ends for the metrics, depending on the device hosting them. In this paper, we present a study of the requirements that an edge monitoring tool should meet, based on motivating scenarios drawn from literature. Additionally, we implement these requirements in a monitoring tool named FMonE. This framework allows deploying monitoring workflows that conform to the specific demands of edge computing systems. We evaluate FMonE by simulating a fog environment in the Grid'5000 testbed and we demonstrate that it fulfills the requirements we previously enumerated.

1. Introduction

Technological advancements of the last years are making a connected world a reality [1]. According to the 2017 IHS Markit report, more than 20 billion devices are used worldwide to generate, process and store data [2]. This is causing a ubiquitous phenomenon; we can find sensors everywhere, clusters of processors, and almost unlimited storage. This interconnected network of devices is nowadays known as the Internet of Things (IoT) [3]. These devices are currently generating about 2.5 quintillion bytes of data daily [4]. Applications from different domains, such as Smart Cities, Healthcare, or Connected Vehicles [5], can make use of this data.

So far, traditional cloud architectures have been used to provide the infrastructure for these previously mentioned applications. However, with this approach all the data generated has to be moved from the edge of the network to a data center [6]. This can create an important bottleneck in the network, specially if this data center is located in a distant region. In addition, cloud providers have their computing assets spread throughout few specific locations [7], making it difficult to host the applications close to the edge. Finally, the response time is expected to be high and unpredictable, as the round time trip of the communication impedes real-time responses.

Edge computing has been proposed as an alternative method that could solve these problems, by performing data processing closer to its source [8]. In this line of thought, a new architecture complementary to the cloud, named fog computing [9], has been proposed. In this approach, the central cloud is still present, but some intermediary nodes stand between the edge devices and the cloud in a hierarchical manner [10]. This enables to filter and preprocess the data before sending it to the cloud, reducing the network traffic. It also allows pushing some complex applications closer to the edge, reducing the communication latency of the clients using them.

As with any kind of distributed infrastructure, monitoring the fog and its different elements is a valuable tool to contribute to maintain quality of service (QoS) and the performance of the applications running on top. This presents several challenges due to the unique characteristics of its nodes and architecture [11, 12]:

(i) Heterogeneity: machines comprising the fog can have different architectures and resources, like CPU or memory. Some examples include mobile devices, set-top boxes, or Raspberry Pis.

(ii) Mobility: nodes are no longer static but can reside at mobile stations, such as trains or cars, or fixed locations, like shops or buildings.

(iii) Connectivity: nodes in the edge are able to communicate effectively through different types of wireless connections. This connectivity can be intermittent because of the above mobility reason.

(iv) Resource-constrained nodes: every device near the edge providing computation and/or storage can be part of the fog infrastructure. Instead of having a small number of powerful machines, many low-power devices can answer the requests of nearby clients.

(v) Geo-distributed: the nodes that make up the fog are distributed across different locations. This imposes some latency between machines at distant locations that needs to be considered in the monitoring requirements.

In this paper, we provide new insights into fog monitoring, presenting an innovative, flexible monitoring solution, named FMonE, to address fog requirements. FMonE is a lightweight, installation-free, and user-adjustable monitoring tool, specially designed to monitor a fog system. It relies on a container orchestration system to build monitoring pipelines that adapt to the distinct features of a fog infrastructure. Our contributions are as follows:

(i) We describe two different fog monitoring use cases inspired by the literature (Section 2) from where we can draw a set of requirements that a fog monitoring tool should have.

(ii) These requirements are listed and explained in order to provide a clear mapping with FMonE design choices (Section 3).

(iii) We present the design and architecture of FMonE (Section 4) based on these requirements. We evaluate it in a simulated fog environment, verifying that it is able to meet the above-mentioned requirements (Section 5).

(iv) We review the state of the art in monitoring solutions (Section 6) and provide a comparison with FMonE in terms of functionality.

We close the paper with a set of conclusions on the challenges of fog computing and monitoring, together with future research work in this field (Section 7).

2. Motivating Scenarios

Cloud is a well-known, established technology with a clear business model behind it [13]. Cloud architectures support general-purpose services at different levels, relying on centralized data centers and infrastructures. Fog is expected to provide different services and applications at the edge of the network, reducing the latency between the client and the application to support a specific commercial activity, by means of a decentralized infrastructure.

In this section, we present two motivating scenarios by broadening the ideas presented in the literature [14]: a telecommunication company that needs to keep track of the usage of all their devices at the edge and a provider of a Platform as a Service (PaaS) for multiplayer games that needs to monitor the different elements of the platform. These scenarios will illustrate the challenges that arise when the monitoring workflow has to be changed from a cloud environment (centralized, homogeneous, and stable) to a fog environment (decentralized, heterogeneous, and unstable).

2.1. Monitoring an Edge Infrastructure. Telecommunication companies, also called telcos, are one of the key actors in the fog computing paradigm [15]. The fog allows them to take advantage of their geographically distributed infrastructure, made of small smart devices like gateways, smart phones, or any other kind of device with storage and computing power. This infrastructure can be used as the *bare metal* that hosts services in closely delimited regions, such as cities or districts, placing them closer to where they will actually be used.

In this sense, telcos should be able to properly manage their resources, to enable Infrastructure as a Service (IaaS) at the edge of the network. All of this requires to monitor the different running devices to make decisions that are analogous to those needed in a data center: scheduling, resource allocation, QoS, or SLAs, among others. But in this case, the resources are not traditional nodes running in a data center, but small to medium smart devices, with different hardware specifications. There can be thousands of these smart nodes in every geographical region, which must be monitored despite their differences.

Monitoring such an infrastructure poses several challenges. Firstly, nodes may join and leave the network frequently (e.g., consider a mobile device that can leave the carrier's coverage area, shutdown, or fail). The monitoring services should be able to handle this type of behavior. Secondly, job scheduling, resource provisioning, and allocation mechanisms require monitoring metrics with very low and predictable latency to make fast decisions. However, the round-trip time between the central data center and the edge devices in a geographically distributed infrastructure could be long. This creates the need for keeping the monitoring data at the edge, following the same paradigm as fog applications, something that is not easily achieved with current monitoring solutions. Additionally, and as it will be shown later in our experiments, the constrained bandwidth of devices at the edge is another important factor in the monitoring process. The monitored information sent outside the local network should be kept to a minimum, in order to leave the available bandwidth to processes that need to use it intensively. Finally, if monitoring metrics are going to be kept at the edge, we need to consider how they are going to be ingested and stored by the devices (e.g., key-value stores, time series database, and message queue). This depends, firstly, on the amount of measurements and, secondly, on the computation and storage capacity of the devices hosting these back-ends. Consequently, we should be able to change the metrics back-end in a flexible way.

In summary, fog IaaS needs to be monitored at the same level as its cloud counterpart. However, the particular characteristics of the devices in this infrastructure make it necessary to incorporate new functionalities to fog monitoring tools. These functionalities are not present in previous centralized and more homogeneous environments.

2.2. Monitoring Service Applications at the Edge. Mobile online games could be one of the industries that most benefit from fog computing. It would reduce latency problems for users, by bringing the game endpoint closer to them, as well as adding an additional location context that is required by many augmented reality applications [16]. Within this field there are companies that offer a PaaS for game developers to abstract things like scaling, player management, user segmentation, and many complex multiplayer features (https://www.gamesparks.com/ (last accessed Feb 2018)). It also assures game developers that their games are going to run within some QoS and SLAs that PaaS has to comply with. If we imagine this PaaS running in a decentralized fog environment, there are some challenges to be considered in terms of monitoring.

First, there is a disparity of technologies that will form the backbone of this platform, such as databases, messages queues, or data processing frameworks. Ensuring that each of these pieces are working properly means different types of metrics, depending on the technology that is being monitored. Secondly, we expect high computing and networking demands in these kinds of applications. Much of the data has to be moved from the server to the gaming clients in a multiplayer environment; thus the monitoring system should not clog the network connection or stall the computing power of fog nodes, especially when their resources are limited. Finally, we may need to look more closely at the problems that could arise in a particular region. This means that we should be able to deploy monitoring agents on a subset of problematic devices, e.g., all the devices corresponding to a user segment.

From this use case we can draw the conclusion that an already complex system like PaaS can be really difficult to monitor if we add an extra layer of complexity like fog computing. A monitoring tool for such an environment should facilitate this process to the user as much as possible.

3. Requirements of a Fog Monitoring System

From the previous scenarios we can extract the following requirements for a fog monitoring tool:

(i) Installation-free **(R1)**. As previously mentioned, there is a clear need of a dynamic, heterogeneous infrastructure, such as the ones from telcos. The node heterogeneity would hinder an installation process with different dependencies based on their architecture.

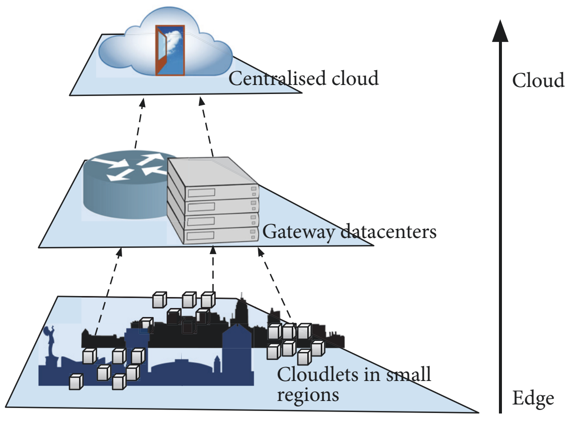


Figure 1: A fog infrastructure starts at the edge of the network, where the data is generated at the IoT layer. Small data centers provide computation and storage close to the applications. If data needs to be sent to the central cloud or intermediate gateways, it is aggregated in a bottom-up fashion.

In addition, nodes can join the network at any time, creating the need for installation-free monitors that need to be up and running as soon as possible.

(ii) Aggregation/filtering of metrics **(R2)**. Fog applications have an implicit aggregation workflow that starts from the edge and ends at the central cloud, as seen in Figure 1. We believe monitoring of such an infrastructure should be done the same way. Fine-grained data can be kept at the edge for regional data analysis, and aggregated information can be sent to the cloud for central reporting or visualization purposes. For example, a telco could have a dashboard that shows the CPU usage by region, overlapped with a map showing the location of its fog computing assets at the time. The telco can navigate up and down the data to have a better control of the infrastructure and its metrics. In addition, users should be able to define a criteria for nonimportant data that can be filtered out to not create any additional traffic or storage burden.

(iii) Flexible back-ends **(R3)**. There are several implications that must be considered when storing monitoring metrics. Some nodes closer to the edge are more unreliable, in the sense that they can disappear from the network at any moment. Therefore, metrics should be stored on more reliable nodes. Also the volume and speed of the monitoring process can change depending on what elements are being monitored, and from how many nodes. For example, for a small amount of monitored components, a MySQL database might be enough. On the other hand, for an intensive log monitoring task on a large geographical area of nodes, we might need a more scalable database, like Cassandra. This requires a flexible back-end in both type and location for the monitored data. Additionally, there may be several back-ends for the same data at more than one of the levels presented in Figure 1.

(iv) Elasticity **(R4)**. Nodes are being continuously added and removed from the network. For example, users can lease their mobile devices as a computing asset [17] for telcos. A fog monitoring solution should be able to detect any node that becomes part of the infrastructure and start monitoring the device and its components as soon as possible. Also, the whole monitoring process should not be hindered by the disappearance of any node in the infrastructure.

(v) Resilience **(R5)**. The nodes in the fog can be faulty, specially considering that they can have limited resources. If a monitoring agent fails, the system should be able to relaunch it. Also, if the host of an important point of the monitoring workflow (e.g., the back-end) disappears, we should be able to reallocate that point to another host nearby.

(vi) Geo-aware **(R6)**. A fog platform has an inherent geo-distributed structure. Nodes are divided into regions depending on their location, which leads to better connectivity between them through local connections and fewer hops across the network. This creates the need of deploying a set of monitoring agents and their back-ends on the same region, in order to leverage these local connections and avoid unnecessary latency in the monitoring workflow. It can also be used to get more detailed data of problematic regions where we need to zoom into a specific technology that is only found on that region. An example of this workflow would be to deploy specialized monitoring agents with a specific plugin in one region hosting a particular set of technologies (e.g., an agent that monitors the number of messages ingested by a Kafka queue in one region).

(vii) Plugin-based monitoring **(R7)**. Different technologies run together in this kind of infrastructure and we might need to monitor them at different levels. For instance, we might want to monitor a Mongo database, a Docker container and a Kafka queue which are part of the chat system of the above-mentioned online gaming PaaS, taking into account the fact that each one will need different kinds of metrics. The deployment agent should have plugins that enable the extraction of these diverse metrics by specifying the types of component that need to be monitored.

(viii) Nonintrusive **(R8)**. Some devices at the edge infrastructure are resource-poor. Consequently, the monitoring process should interfere with the normal functioning of these already constrained devices as less as possible. Especially, the amount of monitored information sent over the network must be kept to a minimum (as bandwidth may be limited). This is specially true when we have demanding applications, like online gaming and running on the nodes.

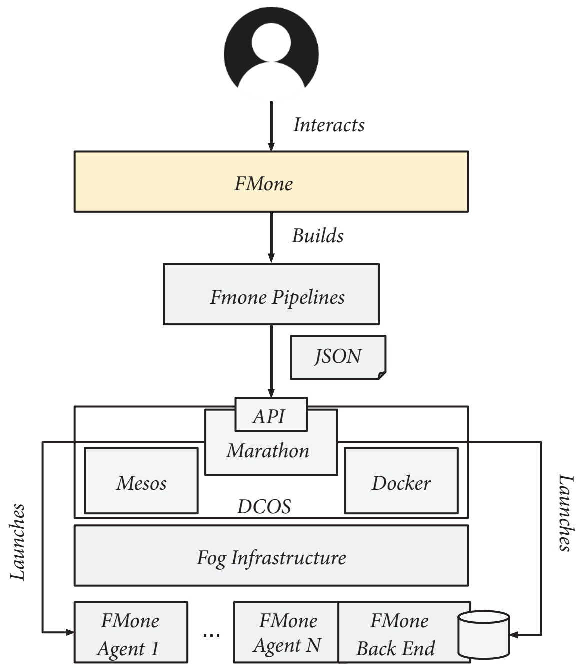


Figure 2: The general architecture that supports FMonE. The user interacts with the FMonE framework (through an UI or command line not yet implemented) which acts as a translator between the user needs and a Marathon deployment plan. When the user has built the monitoring workflow this is translated into an FMonE pipeline, which in turn is passed as a JSON file to the Marathon API. Marathon is then responsible for launching the monitoring agents and the back-ends throughout the infrastructure.

(ix) Hardware and operating system agnostic **(R9)**. A fog infrastructure can consist of normal nodes, mobile phones, gateways or any other kind of devices that can provide some kind of computation power. The monitoring tool should be able to run on these broad spectrum of technologies, independently of the operating system or hardware underneath.

4. FMonE Design: Tackling the Challenges

In this section we explain the design principles of FMonE, a monitoring tool created to satisfy the requirements of Section 3. By leveraging existing technologies, FMonE facilitates to the user the deployment of the monitoring agents across the fog infrastructure. The main advantage of its plugin-based design is to enable the creation of monitoring pipelines that are adapted to the above-mentioned fog particularities.

4.1. Architecture of FMonE

4.1. Architecture of FMonE. FMonE is designed as a framework that includes the monitoring agents together with the management of those agents and their back-ends. The general architecture of the system is depicted in Figure 2. We explain here each of the parts.

(i) FMonE: this is the general framework that coordinates the monitoring process across the whole fog infrastructure. It is a Python framework that communicates with Marathon (https://mesosphere.github.io/marathon/ (last accessed Feb 2018)), a container orchestrator able to coordinate and maintain across the fog devices the monitoring agents and the back-ends for the metrics. Both the monitoring agents and the back-ends are implemented as Docker containers, which can be easily deployed in any kind of node without any installation process and independently of the platform, fulfilling requirements **R1** and **R9**. The Marathon container orchestrator with which FMonE communicates, forms part of DCOS (https://dcos.io/docs/latest/overview/what-is-dcos/ (last accessed Feb 2018)), a software package that can be easily installed on all nodes. Besides Marathon, it also deploys the Docker engine and Apache Mesos [18], which acts as a robust cluster manager that can scale to thousands of nodes. This combination of technologies will facilitate to FMonE the deployment of its different agents and back-ends. Note that any other container orchestrator could be used instead of Marathon. Ideally, this orchestrator should be lightweight in order to work with guarantees on a fog infrastructure. In this paper we have chosen DCOS, because of its ease of deployment and several features that allow us to meet the following requirements:

(a) High availability: Marathon achieves high availability through Zookeeper [19]. This means that there are several Marathon instances available and, in case that the current leading instance fails, a new leader will be elected, achieving 100% uptime. Additionally, it takes care of relaunching any Docker container that fails or relocating it if its current machine fails. This enables requirement **R5**.

(b) Constraints: it allows us to specify a criteria to control in which nodes the FMonE containers should be executed. An example would be to assign a label to the fog nodes depending on the region they belong to and then constraining the deployment of a set of containers to a specific region. As long as the different resources in the fog infrastructure are correctly labeled, it will allow us to have a geo-aware control of the fog infrastructure, by using FMonE to launch monitoring agents in specific regions only. This fulfills requirement **R6**.

(c) Adding nodes: through the above-mentioned constraints, we can also define that in each host an FMonE agent container must be active. As soon as a node joins the system, the framework will start monitoring it without the operator intervention. Therefore, we can meet requirement **R4**. Remember that constraints are part of the Marathon API (https://mesosphere.github.io/marathon/docs/constraints.html (last accessed Feb 2018)) and that FMonE uses them to achieve these features.

(ii) Pipeline: it represents a workflow of FMonE agents that can optionally communicate with each other and/or dump their metrics to one or more back-ends. Users can define their own pipelines. For the purpose of this paper, we have defined three pipelines, as we will see in the experiments section (Section 5). Nonetheless, an interactive layer such as a dashboard or a console could be added to FMonE, allowing the user to examine the different regions, the devices assigned to them and facilitating the definition of new pipelines. Once the pipeline has been defined, it will be translated by FMonE into a Marathon application group (https://mesosphere.github.io/marathon/docs/application-groups.html), a JSON file that establishes an order for the containers to be deployed. Figure 3 includes a description of the conceptual parts that constitute the pipeline concept. Each pipeline takes care of one or more regions of the fog, as the geo-distribution of its devices is one of its distinctive characteristics. In this way users can build different pipelines with different configurations for its monitoring agents, depending on the physical location of the devices. It also facilitates placing them in problematic regions or near their back-ends. A pipeline can have one or more back-ends, which are hosted in one of the regions. Monitoring agents in a pipeline can store their metrics in one of these back-ends, which can be an existing one (previously launched in another pipeline) or a new one, which is created. In each of the regions covered by the pipeline, one or more FMonE agents are launched. For the FMonE agents a set of InPlugin, MidPlugin, and OutPlugin has to be chosen. Details about the plugins will be provided later in this same section. Figure 4 shows an example of a pipeline. As explained in Figure 3, a pipeline takes care of monitoring several regions, in this case regions A, B, and C. Each region has a different number of FMonE agents, whose plugins can be chosen by the user. In region A all the metrics are dumped into an existing back-end 2 in Region D. Region B has 4 agents, three of which perform monitoring tasks and another one aggregates the metrics extracted by the other three. The aggregation is stored in back-end 2. The same pattern is used in region C, but this time we store the full detail of the metrics in back-end 1, hosted in the same region, while another agent aggregates the metrics and store them in back-end 2. This means that metrics can be stored in several back-ends at different granularities. All these data can be queried from the intelligence in the central cloud or by actors at the edge. As previously mentioned, these pipelines enable to change the monitoring workflow depending on the user needs, aggregating metrics, and storing them at different levels (e.g., gateways or IoT devices with storage), fulfilling requirement **R2**.

(iii) Flexible back-ends: users should be able to choose a back-end for their metrics depending on several

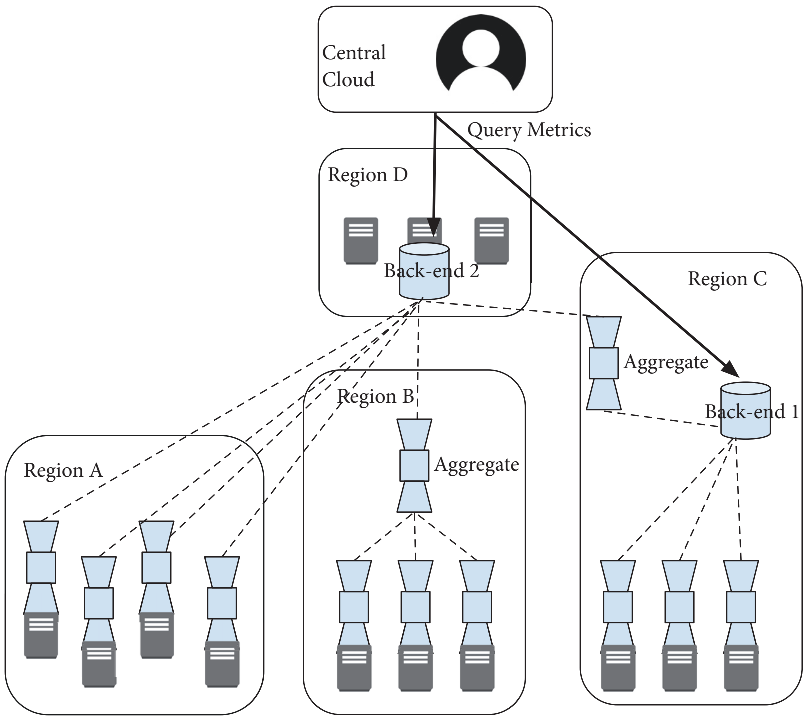


Figure 3: An example of an FMonE pipeline that takes care of three different regions using two back-ends. Users can query the metrics stored in any of the different back-ends.

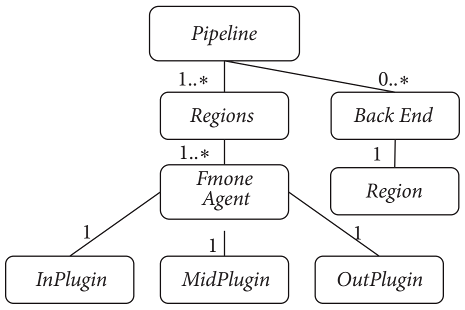


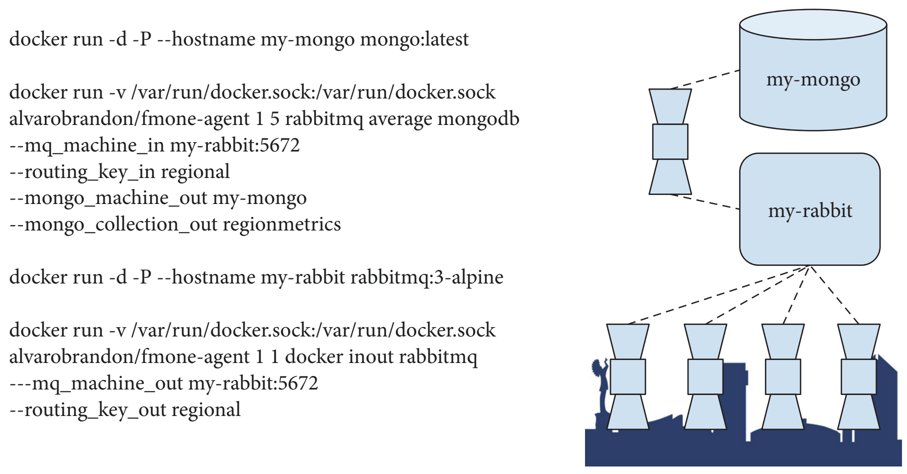
Figure 4: The different concepts that form a monitoring pipeline. A pipeline takes care of one or more regions of the fog infrastructure. Additionally it may have several back-ends. A set of one or more FMonE agents is launched in each region. These agents have three components to gather the metrics: InPlugin to collect data, Midplugin to process it, and OutPlugin to publish it.

factors, like the number of elements to be monitored or the machine in which this back-end is going to be installed. It is evident that if a large amount of metrics needs to be stored, more scalable technologies like Kafka should be used for the ingestion. On the other hand, if we are going to store fewer metrics on a low resource device, we could use lightweight databases like sqlite, which has already been used in some IoT prototypes [20]. In any case and since a fog environment is very diverse, this should be flexible and not limited to one technology. It is not mandatory for a pipeline to have a back-end, since an FMonE agent can always take advantage of an existing one. There is also the possibility of creating one back-end or more for that pipeline. These back-ends are also implemented as Docker containers and the most popular database vendors already provide them. The users can choose the one that best fits their use cases, satisfying requirement **R3**.

(iv) FMonE agent: this is the Docker container responsible for monitoring each of the devices. We divide a monitoring job into three fundamental tasks: collect, filter, and publish metrics. Their responsibilities are divided into three plugins:

(a) Inplugin: the plugin extracts the monitoring information from a component of the system every $x_{collect}$ seconds. This set of metrics is kept in memory before publishing it to the back-end. Examples of these metrics are the CPU load, the stats of a Docker container, or the number of messages received per second in an MQ queue. We have defined a series of plugins that the users can use, allowing them to create new ones on their own by implementing a simple interface. This versatility to monitor different technologies answers to requirement **R7**.

(b) Midplugin: here the user can define a custom function that filters or aggregates the set of metrics that the agent keeps in memory. This function is applied before publishing the values. Aggregating and filtering metrics can reduce


```
docker run -d -P --hostname my-mongo mongo:latest

docker run -v /var/run/docker.sock:/var/run/docker.sock
alvarobrandon/fmone-agent 1 5 rabbitmq average mongodb
--mq_machine_in my-rabbit:5672
--routing_key_in regional
--mongo_machine_out my-mongo
--mongo_collection_out regionmetrics

docker run -d -P --hostname my-rabbit rabbitmq:3-alpine

docker run -v /var/run/docker.sock:/var/run/docker.sock
alvarobrandon/fmone-agent 1 1 docker inout rabbitmq
---mq_machine_out my-rabbit:5672
--routing_key_out regional
```

Figure 5: Several Docker containers are involved in an FMonE pipeline. To extract the metrics, the Docker image for FMonE is started in each host, with `docker` as the Inplugin parameter, `inout` for the MidPlugin, and `rabbitmq` as the Outplugin. The RabbitMQ container will also be created as part of the pipeline and its hostname used as a parameter for the previous `outplugin`. The aggregation will be performed by a different FMonE agent that will pull the metrics with its corresponding `rabbitmq` Inplugin, average them with the `average` Midplugin, and store them in a database with the `mongodb` Outplugin. The pipeline will be completed with the creation of a containerized MongoDB back-end. Marathon and Mesos are used to start these containers in the desired regions.

traffic and fulfills requirement **R8**, for example, publish CPU load values that are above a certain threshold. A more complex logic can be used, from simple aggregates, such as averaging metrics for a time window, to a machine learning classifier that decides whether or not a metric should be published.

(c) Outplugin: it dumps all the filtered data by the Midplugin to a back-end every $x_{publish}$ seconds. This back-end can be any kind of storage as long as it is supported by an outplugin implementation. A file, a document database such as MongoDB, and a key-value store like Cassandra are some examples of back-ends. Another target can be a message queue, from which another FMonE agent can pull, aggregate/filter the metrics, and push them to a different location. This is part of the requirement **R3**.

4.2. Using FMonE. When users want to launch an FMonE pipeline, they have to specify the monitoring workflow throughout the fog infrastructure. First, they choose a set of regions that the pipeline is going to take care of. Afterwards, they have to specify for each of the regions the $x_{collect}$ and $x_{publish}$ intervals together with the Inplugin, Midplugin, and Outplugin that they want to use for the FMonE monitors. Moreover, additional parameters can be passed to the plugins if it is needed. For example, if we want to push metrics to a Kafka queue we need to give as a parameter the name of the Kafka broker and the topic we want to publish to. This process enables the extraction of metrics at different levels of the system, their processing, and their storage in the chosen back-ends in a flexible way. Lastly, the user can optionally provide one or more back-end types (e.g., MongoDB, Cassandra, and InfluxDB) that will be created as part of the pipeline. The back-ends do not have to be in the same region as the agents, although it is recommended because of the previously mentioned latency between locations. This user specification will be translated into a container deployment plan inside the fog infrastructure, in our case, powered by Marathon. We have included the different plugins that we have developed so far and the different parameters that they require as an appendix in Tables 3, 4, and 5. As a final summary of how these pipelines are translated into Docker commands, we detail an example in which the user wants to monitor all of the container metrics in a single region, send them through a RabbitMQ message queue to another FMonE agent that averages the metrics for the last 5 seconds, and finally store them into a MongoDB. The example is depicted in Figure 5. Note that all these Docker commands are automatically executed by Marathon, the container orchestrator tool, while FMonE takes responsibility for building the deployment plan needed by Marathon for each monitoring pipeline.

5. Evaluation

This section evaluates FMonE, demonstrating that the prototype we have built meets the above-mentioned requirements (see Section 3). Keep in mind that these experiments are part of a quantitative evaluation of FMonE performance. A complete qualitative evaluation of the whole set of requirements is included in the related work section, together with a comparison between the different state-of-the-art monitoring tools (see Section 6). A prototype of the agent can be found on Docker Hub (https://hub.docker.com/r/alvarobrandon/fmone-agent/ (last accessed Feb 2018)). We plan to upload in the future the DCOS framework able to coordinate the deployment of the FMonE pipelines.

To set up the evaluation environment, we have considered the reference scenarios shown in Section 2. We envision a fog architecture where an organization has resources in different geographical regions. These resources can be of any nature: devices, servers, or any software component that resides on the edge and fuels the applications from users on different locations. Since the regions are distant from each other, there are bandwidth and latency restrictions between them. As previously shown in Figure 1, a fog architecture assembles a tree as a hierarchical structure. A central region is located at the highest level and all the other regions communicate with it, either to pull data that applications at the edge may need, or to push data needed at the central location. From the root of this tree, the organization needs to monitor and makes decisions about its fog infrastructure. For example, thanks to the edge monitoring agents, we can detect overloaded edge devices in a given region or a node that went down. There are several issues that arise when monitoring this type of environment. The first one is whether sending all this monitoring information to a central location affects application performance (Section 5.2). We hypothesize that it does, since at the edge the connectivity resources such as available bandwidth are limited. The flexible monitoring pipelines that FMonE provides are expected to alleviate that effect. Overhead and resource usage are also important characteristics, since we do not want the monitoring process to interfere with the host performance (Section 5.3). We also need to know how fast we can deploy a monitoring agent in an ever-changing infrastructure where elements are constantly joining and leaving the network (Section 5.4). Finally, we need an autonomous and resilient monitoring system in such an unstable scenario (Section 5.5).

5.1. Experiment Setup. We simulate a fog scenario with virtual machines hosted in the Grid5000 testbed [21] to evaluate FMonE at large scale. Grid5000 is a testbed that provides access to a large amount of computing resources. It is highly customizable and offers a wide range of tools for reproducible experiments. In fact the repositories with the scripted experiments are publicly available (https://github.com/Brandonage/execo-utilities-g5k, https://github.com/Brandonage/execo-g5k-benchmarks). We use vagrant together with the vagrant-g5k (https://github.com/msimonin/vagrant-g5k) plugin specific to this testbed to provision the VMs. Grid5000 has nine sites available with several machines each. We launched 78 VMs in the Rennes site, with 4 cores and 10 GB of RAM and we configured the testbed infrastructure as follows.

(i) We set up a DC/OS cluster with 1 bootstrap node, 3 master nodes, 1 public node, and 73 private nodes, which are going to be monitored. Remember that DC/OS is the container orchestrator that is going to deploy the FMonE pipelines. Further information about its parts can be found in their website (https://docsmesosphere.com/1.10/overview/concepts/ (last accessed Feb 2018)).

(ii) We divide the 73 private nodes in 4 regions to emulate a geo-distributed fog environment:

(a) One region called "central region" which represents the upper cloud layer. This region has a Cassandra cluster installed, which will be used to measure performance with Yahoo Cloud Serving Benchmark (YCSB) [22]. In addition, this region also hosts a Kafka cluster, which we will later use to collect the monitored metrics, acting as a traditional centralized monitoring system. Both technologies will be hosted in 4 different nodes each so as to avoid any interference or overhead.

(b) In the remaining three regions, named "edge regions", we install the YCSB clients that are going to query the Cassandra database in the central region. This represents the communication between components at the edge and the cloud in terms of operations/sec. We will use this metric later to assess how a centralized monitoring approach can degrade the performance of the other components. The clients are configured to perform as many operations as they can per second until they complete 1000 operations.

(iii) Between regions the communication bandwidth is restricted to 4 Mbps and to emulate communication latency, we introduce a delay of 50 ms in all sent packets through the traffic control utility (https://linux.die.net/man/8/tc (last accessed Feb 2018)). Bandwidth was chosen based on the statistics of speed testing in the European Union (http://testmy.net/rank/countrycode.up/ (last accessed Feb 2018)) and taking into account the fact that many of the fog devices use technologies with variable speeds such as 4G/3G.

A diagram of the setup is depicted in Figure 6 for a better comprehension. Note that this is a simulated scenario and there are conditions not included, like routing the traffic through gateways, which we plan to evaluate in future work. Hardware heterogeneity also comes to mind, but it is important to note that FMonE can allocate its agents in any device with a Docker daemon, which has been proved to work even in small single-board devices [23].

5.2. Benefits on Performance. In our first experiment, we want to evaluate the impact on performance of an FMonE pipeline compared to a centralized approach. We configure the following scenario to do this:

(i) To simulate the applications and components that run inside fog devices and that will be monitored by our agents, we launch 5 dummy Docker containers per VM that just sleep for a long period of time.

(ii) Two FMonE agents are deployed in each node. One is going to collect 30 different types of metrics from the disk, memory, network, and CPU usage at the host level. The second one will collect 24 different metrics about the resource usage of each of the five containers running in that node.

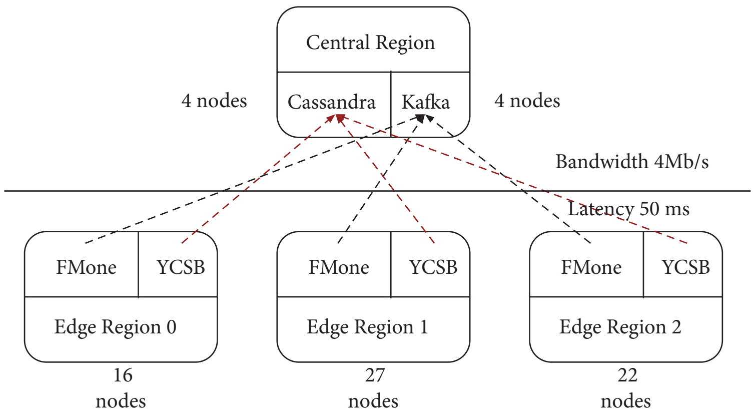


Figure 6: The configuration of our experiments. A central region hosts both a cluster of Cassandra and Kafka. On the edge, three regions of different size host the YCSB clients communicating with the Cassandra database together with the FMonE agents sending their metrics to Kafka. Note that Kafka and Cassandra are running on different nodes to avoid overhead. Bandwidth and latency between regions are limited to 4Mb/s and 50 ms, respectively.

(iii) Metrics are sent every second, since we want to consider the settings of two possible scenarios that can occur in a fog monitoring context. The first one is latency sensitive applications, like the previously explained online gaming use case, where alerts have to be raised almost in real time to act accordingly. The second is the postmortem diagnosis, where users need to determine the exact source of what caused a component failure in their system. Both need a fine-grained level of detail and a continuous record of what is happening on the different components, which justifies the choice of one second as an interval.

(iv) We measure the performance of the YCSB clients with the following FMonE pipelines:

(1) Centralized: we sent all the metrics to the Kafka queue in the central region. This could be considered as the baseline and the paradigm followed by most monitoring solutions, where everything is stored in a single back-end. Note that the nodes that host the Cassandra cluster used for the evaluation of YCSB and the ones that host the Kafka cluster are different. Otherwise we will create an obvious overhead in the database operations.

(2) Aggregated: we have an FMonE agent in each region that is going to aggregate each type of metric for all the hosts of the same region, before sending them to the central location.

(3) RegionalDB: all the agents of one region are going to store their metrics in a MongoDB back-end, which resides in that same region.

(v) In addition, to compare the FMonE pipeline workflow with existing centralized solutions, we measure the performance of a Prometheus [24] server in the central region, in the same vein as the centralized FMonE pipeline. Prometheus is an extensively used open-source cloud monitoring toolkit that is based on a pull model, where metrics are extracted from monitoring agents that expose them through HTTP endpoints. Although the monitoring agents are not part of the Prometheus framework per se, there is a great variety of them developed by the open-source community. In our case, we launch a cAdvisor (https://github.com/google/cadvisor (Last Accessed May 2018)) agent in each node which exposes 52 metrics, a number close to the amount of metrics generated with the FMonE scenarios (30 for host and 24 for containers). The system is configured to pull these metrics every second. We have chosen Prometheus because it is open source, it is easy to configure, and it offers a high degree of customization for its agents. It is also widely adopted in current infrastructures and it is used in many cases together with popular container orchestrators like Kubernetes (https://techcrunch.com/2018/08/09/prometheus-monitoring-tool-joins-kubernetes-as-cncfs-latest-graduated-project/?guccounter=1).

We expect a drop in YCSB client performance for the centralized FMonE pipeline and the Prometheus scenario, since the limited bandwidth available to the device when communicating with the central region will have to be shared between the YCSB operations and the metrics sent by the monitors. We execute all of the workloads that are available by default. The characteristics of the different workloads are depicted in Table 1. Note that since there are connection constraints between the YCSB clients and the central region where the Cassandra database resides, we expect a low number of operations per second. The clients execute each workload until they complete 1000 operations and this is repeated three times. The results can be seen in Figure 7. The operations per second throughput are averaged for the 65 nodes that host the YCBS clients and the three executions. Notice how workload E has a low number of ops/sec since read short range is an expensive operation. As expected, there

Table 1: The different YCSB workloads.

Workload	Description	App Example
A	50/50 reads/writes	Session Store
B	95/5 reads/writes	Photo Tagging
C	100% read	Read User Profile
D	Read latest inserted	Status Update
E	Read short range	Threaded Conversation
F	Read-modify-write	Record user activity

is an improvement in the performance on all workloads with respect to the centralized Prometheus approach, reaching as much as 8% for some cases. It is important to note that these are the results for an emulated scenario in which we use the Linux traffic control utility, while, in a real scenario, this effect would be further exacerbated by the metrics generated by thousands of nodes and the additional stress on the backbone of the network [25]. The aggregated and regionalDB approaches have similar performance as the traffic going out of each region is reduced to the minimum, thanks to the aggregation and flexible back-ends features of FMonE. In the former, only one aggregated metric is sent to the central region and, in the latter, everything is stored within the monitored region. The flexible monitoring model that FMonE provides, through the aggregation, filtering, and geo-aware storage of the metrics, enables an optimal placement of the monitoring workflow, eliminating any impact on the performance of the applications.

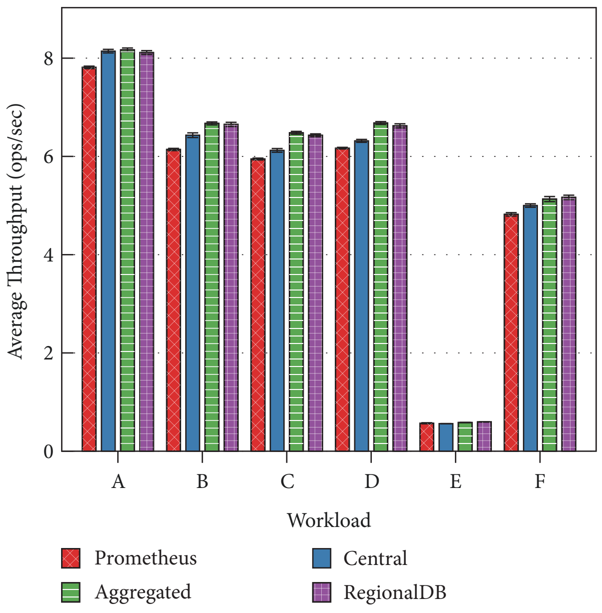


Figure 7: Impact on the performance of edge applications with different monitoring methods. A typical centralized approach, where all the metrics from the edge are sent to a central location, is compared to two FMonE pipelines that aggregate and store metrics in their own region, respectively. The traffic caused by the monitoring information consumes much of the limited bandwidth at the edge, affecting the number of ops/sec.

5.3. Overhead and Resource Usage. The monitoring process should be lightweight and nonintrusive, specially in resource-poor IoT devices. The aim of this experiment is to prove the requirement of nonintrusiveness of FMonE. We monitor the performance of the YCSB clients in each region in two situations: without any monitoring at all and with the previously introduced FMonE regionalDB pipeline, where the metrics from the host and from 15 additional containers per host are dumped to a MongoDB in their same region. Again, metrics are gathered and published every second and the YCSB clients perform requests until they complete 1000 operations. The results are depicted in Figure 8, averaged for the 65 nodes that host the YCBS clients. The overhead is minimal with a maximum drop of 1.2%. The fact that the agent can keep the metrics in memory before publishing them enables us to transfer the data in bulk, using resources in a more effective way. This, together with the fact that the FMonE agent is a lightweight container, means low impact on performance. We also show in Figure 9 the resource usage of the nodes hosting the different back-ends for the metrics. We mentioned that a fog environment has heterogeneous hardware. One of our requirements was deploying different back-ends depending on the computational power of the device that is going to host it. In this experiment we use different back-ends and show their different resource usage profiles. The objective is to motivate why back-ends should be chosen depending on the computational capabilities of the hosting device. We compare Prometheus with two of the different back-ends that are available in FMonE: MongoDB and Kafka. The latter was deployed as a cluster in 4 different hosts and the resource usage was averaged across them. This can be effectively used to divide the storage burden across different devices if needed. The Prometheus back-end has a higher CPU usage than other options as MongoDB or Kafka, indicating that Prometheus might not be suitable for a device with low CPU capacity. The percentage of memory used remains very similar for the three solutions with values of around 22 and 25%. The disk usage graphic reveals peaks of up to 25 MB written to disk for Prometheus while Kafka and MongoDB show a more stable write pattern of a few MB per second. As it can be seen, different back-ends show different resource usage patterns. Detaching the back-end required by the monitoring solution and offering different options for the metrics storage enable a better planning of the monitoring

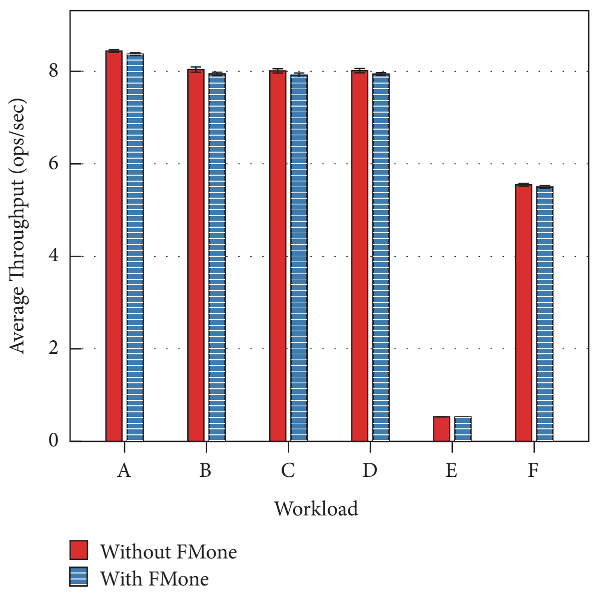


Figure 8: Overhead of an FMonE deployment. Even when monitoring 15 containers per host, the overhead is negligible for an FMonE pipeline.

workflow. Light back-ends can be used for fog devices with limited resources and more powerful and complex ones for upper layers near the cloud.

5.4. Elasticity. Fog devices can frequently join and leave the infrastructure, as stated in our requirements section. For instance, a mobile node that joins the network, or a device that is switched on, has to be included again in the monitoring process. We need to assess how soon agents can be up and running on a device. We measure the time in seconds that it takes to deploy FMonE agents in the following configurations: 1 in one node, 2 in one node, 10 in 5 nodes, and 30 in 15 nodes. When using a Docker container, the node needs to pull down the image from a repository if it has not been done yet. We perform the test in both situations, where the node pulls and does not pull the image. The results are shown in Figure 10 with the average deployment time and standard deviation for 5 rounds of experiments. As expected, a node that runs FMonE for the first time introduces some overhead when downloading the image. This overhead will be higher the more agents we deploy at the same time, due to the time needed to download the image. The maximum is 34 seconds for 30 agents and 15 hosts. Bear in mind that this situation, where the node has to pull the Docker image, only happens once and it takes only 2.5 seconds to deploy them once the nodes already have the Docker image.

5.5. Resilience. Lastly, we test the resilience of all the FMonE parts. The following three scenarios are laid out where we introduce a series of failures that can happen in an unstable scenario such as fog:

Table 2: Resilience of the system.

	Mean recover time	Standard deviation
Agent	6.637 s	2.454 s
Backend	27.602 s	17.889 s
Marathon + Pipeline	96.844 s	34.627 s

(i) We shut down the FMonE agents running on a host and measure the time that passes between the shutdown and the agent recovery.

(ii) We then shut down the MongoDB instance. Since the agents do not have a back-end to publish to, they will also exit with an error code. We measure the time interval for the whole pipeline to go up again.

(iii) Finally we shut down all the containers of the pipeline and the node containing the Marathon engine. Marathon is the foundation in which the FMonE operation is based on. However, it is also resilient by means of a Zookeeper instance [19] and is able to relaunch itself again by choosing a new leader. The pipeline is automatically restored afterwards. Again, this time interval is measured.

We repeat the experiment 10 times for each scenario. The mean time and its standard deviation are shown in Table 2. We can see how an agent can recover quickly and lose only about 7 seconds of monitoring information. There is more overhead for a failure in a back-end, since these containers have more complex deployment times than the lightweight FMonE agent. Additionally, a failure in a back-end also affects all the other agents, as previously explained. Nonetheless, the maximum delay we observed was 58 seconds with a mean of 27 seconds for the 10 experiments. The most costly situation to recover is when Marathon fails. In this case the system has to elect a new Marathon instance and relaunch the FMonE pipeline. The system recovered in 96 seconds on average. Remember that this is an extreme situation, since the Marathon instance does not have to be running in any of the fog nodes which are unstable and constrained devices. The idea is to have all the core components like the Mesos master, Marathon, and the FMonE framework in a stable location from where the whole fog infrastructure can be coordinated.

6. Related Work

From traditional clusters to modern cloud and fog systems, monitoring has always been an important aspect of distributed system analysis and management. Different approaches and tools have been applied at different levels of the distributed environment, often combining their efforts. This creates a vast ecosystem of monitoring tools that we try to summarize in this section by detailing the most recent work.

In relation to the work presented in this paper, we describe a first group of monitoring approaches designed for the cloud, sharing some properties with a fog/edge environment. Monitoring in the cloud has experienced many important advancements in later years. Nagios Core [26] is

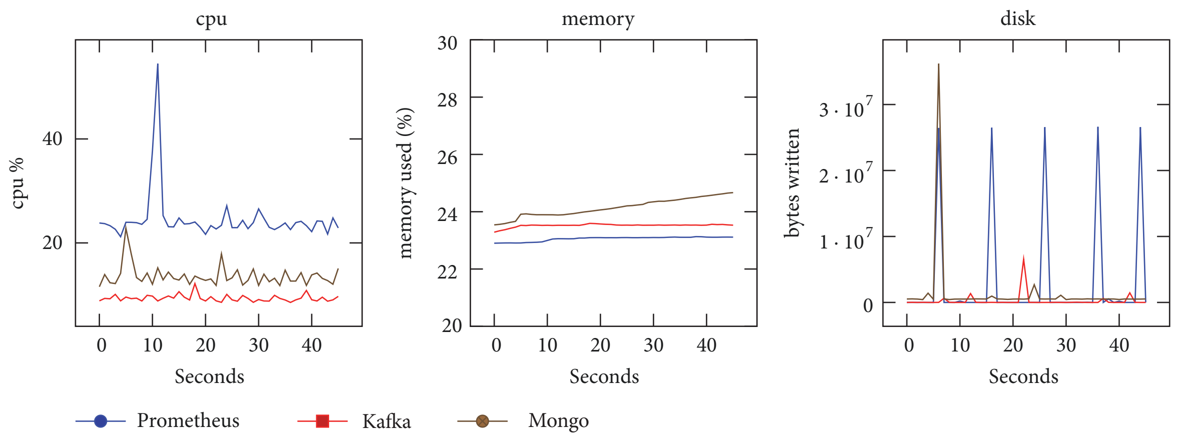


Figure 9: Time series of the resource usage for Prometheus and two of the available FMonE back-ends: Kafka and MongoDB. Prometheus is a more resource intensive back-end compared to the other two. Note that the resource usage is important when placing the back-end for metrics on devices with limited resources such as fog.

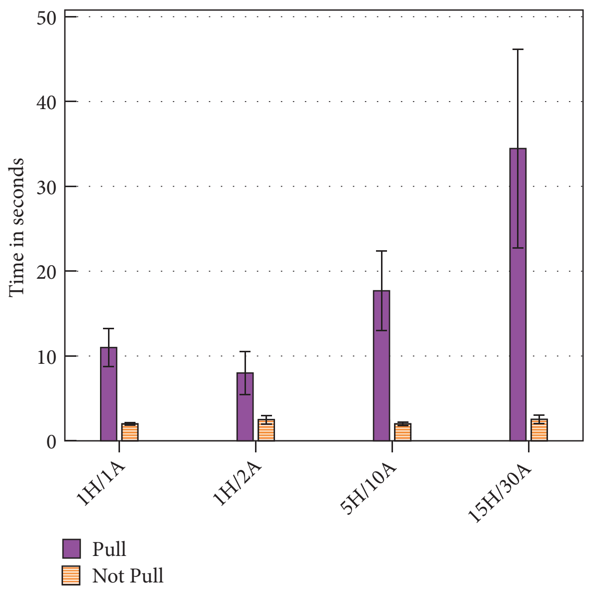


Figure 10: Time that it takes to start up the monitoring agents. We measure it when the container image needs to and does not need to be pulled from the Docker repository. We also vary the number of host and containers from 1 host and 1 agent to 15 hosts and 30 agents. Pulling creates an obvious traffic overhead compared to the fast deployment of a container that has already been pulled.

an integral monitoring tool, capable of monitoring systems, network, and infrastructure. Although Nagios was originally conceived outside the cloud, its most current versions can be easily deployed in a cloud environment, being capable of monitoring both physical and virtual resources. GMonE [27] is a general-purpose monitoring approach that covers the different aspects of the cloud, like the main cloud service models (IaaS, PaaS and SaaS), client-side or service provider-side, and virtual or physical side monitoring. DARGOS [28] is a decentralized cloud monitoring tool for multitenant clouds, designed to efficiently disseminate monitoring information providing aggregate and filter capabilities. DARGOS uses a hybrid pull/push strategy to disseminate data and relies on standard technologies, such as REST and JSON. Datadog [29] is a commercial cloud monitoring solution that incorporates a SaaS-based analytics platform. Datadog combines information from servers, platform tools, and applications, providing a unified view of the cloud system. New Relic [30] is also a popular SaaS solution that can be easily deployed to monitor data in a cloud. The New Relic Platform provides real-time analytics and full-stack visibility at cloud scale. Similarly, Dynatrace [31] commercializes a solution for application performance management, focusing on application monitoring inside a PaaS cloud. Dynatrace manages the availability and performance of software applications and the impact on user experience in the form of deep transaction tracing, synthetic monitoring, real user monitoring, and network monitoring. Finally, Prometheus [24] is an open-source monitoring and alerting tool that has been widely adopted as a complement for container orchestrators. It follows a pull model, where metrics are collected from endpoints that expose them.

All these approaches incorporate interesting ideas that can be used in the fog, like multilevel monitoring, decentralized coordination, and efficient data dissemination. None of them, however, fulfills the complete list of necessary requirements presented in Section 3 to efficiently monitor a fog system, since they are designed for a cloud scenario, where monitoring is usually centralized in one place. In this sense, covering all aspects of fog monitoring could involve, among other things, undesired redundancy and system overhead. This paper has presented a general-purpose fog monitoring framework that covers all fog requirements. FMonE provides flexible back-ends and user-tailored monitoring pipelines across the system. This is important in a fog environment, where there is a hierarchical structure with diverse hardware in place. FMonE allows aggregating monitored data from the edge up to the cloud, making it possible to choose between different back-ends.

In very recent years there have been some publications that address the challenges and specific characteristics of fog

Table 3: Available InPlugins. Their responsibility is to extract metrics from different components.

Type	Plugin parameters	Description
Host	None	CPU, memory, disk and network metrics of the machine hosting the container. In the case of UNIX, they are extracted from the /proc directory
Docker	None	CPU, memory, disk and network metrics of the container itself. These are extracted from the /var/run/docker.sock that streams, among other things, stats about the containers.
RabbitMQ	mq_machine_in: The RabbitMQ server to connect to routing_key_in: The routing key from which we want to read the messages	It extracts metrics that have been previously published by other FMonE agents to a RabbitMQ server with a routing key
Kafka	kafka_bootstrap_in: The Kafka bootstrap server to connect to kafka_topic_in: The topic from which we want to read the messages	It extracts metrics that have been previously published by other Fmone agents to a Kafka topic. The user can choose this messaging service over RabbitMQ when the amount of metrics needs a more scalable solution

Table 4: Available MidPlugins. Their responsibility is to filter and aggregate the metrics collected by the InPlugin.

Type	Plugin parameters	Description
Inout	None	It just passes the metrics without any preprocessing from the InPlugin to the OutPlugin
Average	None	It averages all the metrics that have been collected by the agent between the publish intervals defined by the $x_{publish}$ parameter

Table 5: Available OutPlugins. Their responsibility is to push the metrics to an available back-end.

Type	Plugin parameters	Description
File	outfilepath: The path inside the filesystem where the metrics are going to be dumped into	It stores all the metrics in a file. Useful if the output of the agent is unstructured data like text
Console	None	It prints all the metrics to the stdout of the process
RabbitMQ	mq_machine_out: The RabbitMQ server to connect to routing_key_out: The routing key to which we want to push the messages	It pushes the metrics of the agent to a RabbitMQ server with a routing key.
Kafka	kafka_bootstrap_out: The Kafka bootstrap server to connect to kafka_topic_out: The topic to which we want to push the messages	It pushes the metrics of the agent to a Kafka topic. The user can choose this messaging service over RabbitMQ when the amount of metrics requires a more scalable solution
MongoDB	mongo_machine_out: The MongoDB server to connect to mongo_collection_out: The MongoDB collection where we want to store the metrics	It stores the metrics in a MongoDB backend.

monitoring. Abderrahim et al. identify a collection of properties that a monitoring service should have from a fog/edge point of view [32], but they do not offer an implementation neither an evaluation. Taherizadeh et al. [33] lay out some of the same observations, presenting a state-of-the-art review on monitoring edge computing applications. In their conclusions they state the need for a fog solution that meets all the monitoring requirements of the fog/edge computing paradigm, which, as a matter of fact, are fulfilled by FMonE. Regarding specifically implemented tools, FAST [34] is a fog-based monitoring tool focused on health applications instead of infrastructure monitoring. In FAST, real-time event detection is distributed throughout the network by splitting the detection task between the edge devices (e.g., smart phones attached to the user) and the server (e.g., servers in the cloud). Wu et al. propose a monitoring approach for fog systems in a cyber-manufacturing scenario [35]. This proposal presents a fully developed framework that includes wireless sensor networks, communication protocols, and predictive analytics. Both approaches are focused on particular application scenarios and lack the necessary generality to be considered as complete fog computing monitoring solutions. PyMon [36] is the solution most similar to our work, where the author implements and evaluates a lightweight framework designed to monitor elements at the edge. In comparison, FMonE has more features at its core, providing flexible back-ends and allowing the aggregation and filtering of metrics. In addition, compared to all alternatives, FMonE includes the pipeline concept, a flexible system to adapt the monitoring workflow to the heterogeneity, and hierarchical organization of the fog.

7. Conclusions and Future Work

Fog computing is powered by a complex and heterogeneous infrastructure that is constantly evolving. This presents several challenges when ensuring the performance and quality of service of the applications they host. In this paper, we have listed those challenges and proposed FMonE, a monitoring framework that coordinates highly customizable monitoring agents and their back-ends in a flexible way. We have demonstrated that this tool can satisfy the different requirements through a series of experiments that demonstrate its resilience, elasticity, and nonintrusiveness. We have also showed that the centralized monitoring approach used by the cloud is not the best fit for fog computing. The devices that comprise the fog have features such as location, connectivity, or hardware resources that vary across the infrastructure and, therefore, the monitoring process should adapt accordingly.

Our experiments also show that container technologies can enable the fog computing paradigm. Same as with FMonE, other fog applications can be deployed on disparate devices, such as smart phones or any other system as long as they support the deployment of containers. This will completely detach the application from the hardware and operating system on which it runs. This also opens up the possibility of a lightweight container orchestrator for fog that coordinates thousands of devices with limited resources. We have used DCOS to ease the execution of the experiments but a more lightweight container orchestrator like Hypriot Cluster Lab [37] might be needed when using resource-constrained devices at the edge.

As future work, we want to evaluate the traffic bottlenecks that this kind of system can experience in an environment with real gateways that route traffic between regions. We also want to add new plugins to the tool and explore the possibility of using complex algorithms to raise alarms and detect anomalies directly inside the agent. To facilitate the management of the FMonE pipelines by the user, we would like to add a web UI, allowing the user to easily select the placement of the different back-ends and monitoring agents of the pipeline. As a parallel line of work, we want to explore methods that can define regions within a fog infrastructure, depending on the connectivity between its components.

Appendix

A. The Different Types of Plugins Available in FMonE and Their Parameters

See Tables 3, 4, and 5.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities as well as other organizations (see https://www.grid5000.fr). This work is part of the BigStorage project, supported by the European Commission under the Marie Sklodowska-Curie Actions (H2020-MSCA-ITN-2014-642963) and it has been partly funded by the Spanish Ministry of Economy (TIN2015-70799-R and TIN2016-78011-C4-4-R).

References

[1] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[2] IHS Markit, *IoT Trend Watch 2017*, 2017.

[3] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1101-1102, 2012.

[4] IBM Marketing Cloud, *10 Key Marketing Trends for 2017*, 2017.

[5] E. Balandina, S. Balandin, Y. Koucheryavy, and D. Mouromtsev, "IoT Use Cases in Healthcare and Tourism," in *Proceedings of the 17th IEEE Conference on Business Informatics, CBI 2015*, pp. 37–44, Portugal, July 2015.

[6] E. Qin, Y. Long, C. Zhang, and L. Huang, "Cloud computing and the internet of things: Technology innovation in automobile service," in *Proceedings of the International Conference on Human Interface and the Management of Information*, pp. 173–180, Springer, 2013.

[7] Aws global infrastructure, 2017, https://aws.amazon.com/about-aws/global-infrastructure.

[8] M. Satyanarayanan, P. Simoens, Y. Xiao et al., "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the 1st ACM Mobile Cloud Computing Workshop, MCC 2012*, pp. 13–16, Finland, August 2012.

[10] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the 3rd ACM Workshop on Mobile Cloud Computing and Services, MCS'12*, pp. 29–36, UK, June 2012.

[11] P. Varshney and Y. Simmhan, "Demystifying Fog Computing: Characterizing Architectures, Applications and Abstractions," in *Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 115–124, Madrid, Spain, May 2017.

[12] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and Opportunities in Edge Computing," in *Proceedings of the IEEE International Conference on Smart Cloud, SmartCloud '16*, pp. 20–26, USA, November 2016.

[13] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing—the business perspective," *Decision Support Systems*, vol. 51, no. 1, pp. 176–189, 2011.

[14] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, vol. 546 of *Studies in Computational Intelligence*, pp. 169–186, 2014.

[15] R. Vilalta, V. Lopez, A. Giorgetti et al., "TelcoFog: A Unified Flexible Fog and Cloud Computing Architecture for 5G Networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 36–43, 2017.

[16] A. V. Dastjerdi and R. Buyya, "Fog Computing: Helping the Internet of Things Realize Its Potential," *The Computer Journal*, vol. 49, no. 8, pp. 112–116, 2016.

[17] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[18] B. Hindman, A. Konwinski, M. Zaharia et al., "Mesos: A platform for fine-grained resource sharing in the data center," *NSDI*, vol. 11, pp. 22-22, 2011.

[19] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *Proceedings of the USENIX annual technical conference*, vol. 8, p. 9, Boston, MA, USA, 2010.

[20] S. A. H. Z. Abidin and S. Noorjannah Ibrahim, "Web-based monitoring of an automated fertigation system: An IoT application," in *Proceedings of the 12th IEEE Malaysia International Conference on Communications, MICC 2015*, pp. 1–5, Malaysia, November 2015.

[21] D. Balouek, A. C. Amarie, G. Charrier et al., "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *Cloud Computing and Services Science*, I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, Eds., vol. 367 of *Communications in Computer and Information Science*, pp. 3–20, Springer International Publishing, 2013.

[22] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10*, pp. 143–154, USA, June 2010.

[23] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, "The Glasgow raspberry Pi cloud: A scale model for cloud computing infrastructures," in *Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems Workshops, ICDCSW 2013*, pp. 108–112, USA, July 2013.

[24] Prometheus, 2018, https://prometheus.io/.

[25] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2009.

[26] W. Barth, *Nagios: System and Network Monitoring*, No Starch Press, San Francisco, CA, USA, 2nd edition, 2008.

[27] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, "GMonE: a complete approach to cloud monitoring," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2026–2040, 2013.

[28] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler et al., "Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds, Future Generation Computer System," in *Proceedings of the The fourth IEEE International Conference on e-Science 2011 e-Science Applications and Tools & Cluster, Grid, and Cloud Computing*, vol. 29 (8), pp. 2041–2056, including Special sections: Advanced Cloud Monitoring Systems, 2013.

[29] Datadog - real-time performance monitoring, 2017, https://www.datadoghq.com.

[30] New relic, 2018, https://newrelic.com/.

[31] Dynatrace, 2017, https://www.dynatrace.com/.

[32] M. Abderrahim, M. Ouzzif, K. Guillouard, J. Francois, and A. Lebre, "A Holistic Monitoring Service for Fog/Edge Infrastructures: A Foresight Study," in *Proceedings of the 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 337–344, Prague, August 2017.

[33] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski, "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review," *The Journal of Systems and Software*, vol. 136, pp. 19–38, 2018.

[34] Y. Cao, S. Chen, P. Hou, and D. Brown, "FAST: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation," in *Proceedings of the 10th IEEE International Conference on Networking, Architecture and Storage, NAS 2015*, pp. 2–11, USA, August 2015.

[35] D. Wu, S. Liu, L. Zhang et al., "A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing," *Journal of Manufacturing Systems*, vol. 43, pp. 25–34, 2017.

[36] M. Grobmann and C. Klug, "Monitoring Container Services at the Network Edge," in *Proceedings of the 29th International Teletraffic Congress, ITC 2017*, pp. 130–133, Italy, September 2017.

[37] M. Großmann, A. Eiermann, and M. Renner, "Hypriot cluster lab: an arm-powered cloud solution utilizing docker," Tech. Rep., Hypriot, 2016.