VARYS: An Agnostic Model-Driven Monitoring-as-a-Service Framework for the Cloud

Alessandro Tundo
University of Milano-Bicocca
Milan, Italy
alessandro.tundo@unimib.it

Marco Mobilio
University of Milano-Bicocca
Milan, Italy
marco.mobilio@unimib.it

Matteo Orrù
University of Milano-Bicocca
Milan, Italy
matteo.orru@unimib.it

Oliviero Riganelli
University of Milano-Bicocca
Milan, Italy
oliviero.riganelli@unimib.it

Michell Guzmàn
University of Milano-Bicocca
Milan, Italy
michell.guzman@unimib.it

Leonardo Mariani
University of Milano-Bicocca
Milan, Italy
leonardo.mariani@unimib.it

ABSTRACT

Cloud systems are large scalable distributed systems that must be carefully monitored to timely detect problems and anomalies. While a number of cloud monitoring frameworks are available, only a few solutions address the problem of adaptively and dynamically selecting the monitored indicators, based on the actual needs of the operator. Unfortunately, these solutions are either limited to infrastructure-level indicators or technology-specific, for instance, they are designed to work with OpenStack only.

This paper presents the VARYS monitoring framework, a technology-agnostic Monitoring-as-a-Service solution that can monitor KPIs at all levels of the Cloud stack, including the application-level. Operators use VARYS to indicate their monitoring goals declaratively, letting the framework to perform the operations necessary to achieve a requested monitoring configuration automatically. Interestingly, the VARYS architecture is general and extendable, and can be used to support increasingly more platforms and probing technologies.

CCS CONCEPTS

• **General and reference** → **Metrics**; *Design*; • **Networks** → **Cloud computing**; **Network monitoring**; • **Computer systems organization** → **Cloud computing**; Reliability; Availability; • **Software and its engineering** → *Software design engineering*.

KEYWORDS

Monitoring-as-a-Service, Cloud computing, Monitoring Framework

ACM Reference Format:
Alessandro Tundo, Marco Mobilio, Matteo Orrù, Oliviero Riganelli, Michell Guzmàn, and Leonardo Mariani. 2019. VARYS: An Agnostic Model-Driven Monitoring-as-a-Service Framework for the Cloud. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3338906.3341185

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08.
https://doi.org/10.1145/3338906.3341185

1 INTRODUCTION

Cloud computing adoption has demonstrated its undeniable benefits. It provides savings in IT resources, including a reduction in the operational costs and the flexibility to pay only for what is used [7, 17, 32]. Cloud computing also allows business to be more competitive thanks to computing platforms that provide scalability and high-performance resources, enabling and simplifying the construction of configurable, reliable, and adaptable applications [31].

The flexibility and adaptability requirements that characterize the Cloud environment may threat the health of the running applications [21, 24], which must be constantly monitored to timely reveal misbehaviors and anomalies [2]. Monitoring cloud applications can be however challenging. An effective fine-grained monitoring system should be able to collect data from *all the layers* of a cloud system, including *applicative indicators*; should be able to *dynamically adapt the monitoring system*, opportunistically deploying and undeploying probes based on the current monitoring goals; and it should support the presence of *multiple operators* that interact with the monitoring system according to different objectives.

Existing open source and commercial products implement useful but limited monitoring capabilities. For instance, Elastic Stack [9] and Prometheus [23] are extensively used to monitor cloud applications, but they implement limited forms of adaptation and require significant manual effort to be configured and used.

Monitoring-as-a-Service (MaaS) solutions address both automation and flexibility by defining frameworks that support the declarative configuration of the monitoring system. That is, users specify the key performance indicators (KPI) that must be collected (e.g., cpu and memory usage), and the framework reconfigures the monitoring system to collect the required KPIs. For example, CloudWatch [3] is a MaaS that provides the ability to reconfigure the set of monitored KPIs, even if limitedly to the infrastructure. MonPaaS [8] and Monasca [14] provide more general MaaS solutions that can potentially deal with a range of KPIs collected at various levels and from different components. However, MonPaaS and Monasca are platform-specific solutions (they are both designed to work with OpenStack), they require non-trivial engineering to incorporate additional (e.g., application level) indicators, and can be configured through low-level APIs not providing a high-level interface suitable

for the operator. Finally, there are monitoring frameworks that have been designed to address specific aspects, such as monitoring elastic and adaptive tasks [30] and achieving scalability [18].

In this paper we present VARYS, a MaaS solution, and relative framework, that is designed to be both technology-agnostic and extensible, and that can be used to cost-effectively collect a variety of KPIs, including application-specific indicators. VARYS fully supports the dynamic deployment and redeployment of probes, based on high-level monitoring goals specified by the users of the monitoring system. To obtain a flexible but also powerful monitoring system, the interaction with VARYS is driven by a model that categorizes the KPIs that can be collected. This model is exploited by the operator to control the monitoring system. The designed architecture finally supports a multi-user perspective, decoupling the MaaS client nodes, which can be present in a number of instance, from the MaaS server, which orchestrates the monitoring process. VARYS is part of a more general framework that we are developing to cost-effectively control the health of Cloud applications [26].

The paper is structured as follows. Section 2 presents the VARYS solution. Section 3 discusses its implementation, representative scenarios, and time figures. Section 4 provides final remarks.

2 VARYS

In this section, we briefly illustrate VARYS and its architecture.

2.1 The VARYS Approach

The purpose of VARYS is to enable users to declaratively manage their monitoring goals, while avoiding the burden of delving into technicalities and freeing them from dealing with different underlying Cloud technologies. VARYS provides a same monitoring management environment to stakeholders with different professional background (e.g., managers, technicians, etc.) and different monitoring goals, and is characterized by the following capabilities.

Model-driven: the operators can specify their monitoring goals exploiting a simple tree-like model derived from the ISO 25011 standard about service quality [15, 16]. The model defines multiple quality attributes that are decomposed into finer grained concepts, until reaching measurable properties. Operators can actually select concepts at any level of the tree. The selection is automatically mapped into the collection of the measurable properties associated with the leafs of the selected subtree. This request is then turned into a set of probes that are deployed in the target system.

The model-driven nature of the interaction has several benefits, including the possibility to use VARYS without having specific technical skills, that is, it is suitable even for managers. Moreover, technical users can modify the model, so that the KPIs that can be collected and the specific organization of the quality attributes reflect the actual needs of the organization that uses the technology.

Technology agnostic: VARYS is a general MaaS framework that is designed to support multiple, potentially any, target platform, including VM-based solution [20] and containers-based solutions [5], and multiple probing technologies, such as Elastic Stack Beats [10], Prometheus Exporters [22] or custom probes. Since all the decisions that depend on technologies are taken by the framework itself, operators can use the same MaaS solution to target largely different cloud environments.

Reconfigurable: VARYS offers full MaaS capabilities, that is, operators can change their monitoring goals at any time. As a consequence VARYS automatically deploys and undeploys probes to match the new set of goals. This is done in the context of a multi-user environment, while optimizing the number of probes deployed and the resources consumed.

Multi-layers: VARYS supports collecting KPIs from all the layers of a Cloud stack, including the collection of application-specific indicators. This is possible because VARYS can incorporate any probe once annotated with metadata, and is not limited to probing mechanisms available at the infrastructure level, as other solutions do. Metadata are used to specify information such as the KPIs that a probe can collect and how to deploy the probe.

Extensible: VARYS can be extended according to multiple dimensions, including: the underlying data storage engine, for instance Elasticsearch [11] and Prometheus [23] can be used interchangeably; the set of probes, which can be extended with new probes once properly annotated; the model, which can be modified to reflect the quality attributes used by an organization; and the target platforms, which can be supported by implementing plug-ins responsible for actuating the required changes on the target system.

2.2 Architecture

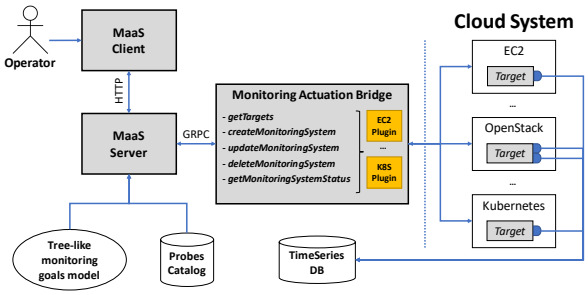

Figure 1: VARYS architecture

Figure 1 shows the architecture of VARYS, which is composed of three main components: the *MaaS Client*(s), which implements the front-end; a *MaaS Server*, which implements the logic of the VARYS approach; and the *Monitoring Actuation Bridge (MAB)*, which is responsible of traducing the technology-independent requests originated by the MaaS Server into concrete operations that must be performed on the cloud platform to monitor the services.

The *MaaS Client* is the entry point to VARYS. The operator interacts with the MaaS client to produce monitoring requests that are processed by the MaaS Server. These requests are formulated as a number of monitoring goals that must be achieved, selected from a tree-based representation of the quality attributes that can be monitored. Of course, multiple MaaS clients with different operators can connect to the same MaaS server.

The *MaaS Server* receives requests that specify the monitoring goals that must be satisfied from the MaaS Client, and transforms these requests into a technology agnostic probe deployment strategy that can be actuated with the Monitoring Actuation Bridge. The strategy includes the list of probes that must be deployed, the

list of services that must be monitored with these probes, and the monitoring pattern that must be used to deploy the probes.

The MaaS Server supports multiple monitoring patterns that can be actuated depending on the targets that are monitored. In particular, the probes can be installed in the same computing units that host the monitored targets, if the computing units are accessible, or in a separated sidecar computing unit [6]. The former pattern implies accessing the virtual machine or the container that already runs the monitored target to deploy the probes. The latter pattern implies creating new virtual machines or new containers running the selected probes to collect data from another virtual machine or container that runs the monitored target. The sidecar pattern is regularly used with the container technology, since sidecar containers can share resources with the target containers and monitoring can be performed seamlessly [6]. Note that the MaaS server identifies the probes and the pattern to be used for their deployment, but it does not take any technology-specific choice (e.g., the MaaS server does not decide if the sidecar must be a container or a virtual machine, but only that a sidecar must be used for monitoring).

The *probes* that can be deployed are stored in a *probe catalog* and associated with metadata that allow the MaaS server to take deployment decisions. The metadata include information such as the patterns supported by the probe (e.g., deployment in a sidecar versus deployment in the same computing unit that hosts the monitored service), the KPIs that the probe can collect, and the reference to the deployment artifacts associated with the probe.

The *Monitoring Actuation Bridge (MAB)* is the component responsible for mapping the pattern identified by the MaaS Server into a number of operations, which are performed atomically, to actually deploy the necessary set of probes. In particular, the Monitoring Actuation Bridge implements five operations: operations for creating, updating, and deleting the set of probes present in the cloud platform, an operation for getting the list of targets that can be monitored, and an operation for getting the status of the deployed probes. The capability to address specific cloud environments, such as applications running on Kubernetes [28] or OpenStack [29], is achieved through a plug-in architecture whose plug-in components implement the five interface operations for a specific cloud environment. This design allows VARYS to simultaneously cope with targets deployed over different cloud providers. For instance, the operation that returns the list of the available targets iterates over all the active MAB plug-ins to finally return a single list with the targets returned by each of the plug-ins. Each target is annotated with metadata that allow VARYS to access the cloud platform that runs the monitoring target.

The deployed probes push data in a *time series database (TSDB)* that stores the collected data. VARYS can be integrated with any TSDB, as long as probes are properly configured. VARYS also supports the possibility of using a publish-subscribe channel that decouples the probes from the TSDB.

Note that VARYS cannot only deploy probes, but can dynamically undeploy existing probes, deploy additional probes, and so on, achieving total flexibility in the dynamic (re)definition of the monitored KPIs. When running multiple clients, the MaaS server can optimize the number of probes deployed accordingly to the identified pattern, collecting the data useful to multiple clients once.

3 FRAMEWORK IMPLEMENTATION

This section illustrates the details of the implementation, discusses representative usage scenarios for our framework, and reports time figures about deployment operations.

3.1 Implementation Details

```
{
    'human_readable_name': 'Metricbeat CPU Consumption',
    'name': 'metricbeat_cpu_consumption',
    'supported_target_envs': ['ACCESSIBLE_VM','CONTAINER'],
    'supported_cloud_properties': ['cpu_consumption'],
    ...
```

Figure 2: Excerpt of probe metadata

Our framework implementation is available at the following Git repository: https://gitlab.com/learnERC/esec-fse-tool-demo. We implemented the components of the architecture as follows.

MaaS Client It is a web application built with Vue.js, a progressive framework for building user interfaces [12].

MaaS Server It is a Python web application based on Flask [4] which exposes a REST API to the MaaS Clients and relies on Redis [25] to fulfill monitoring requests using a job-oriented approach.

Monitoring Actuation Bridge (MAB) It is a Python component that exposes a *gRPC* [13] interface, which is a Remote Procedure Call (RPC) framework used to connect services in and across data centers. We currently implemented a MAB plug-in for targets running in Kubernetes. It is able to operate with multiple clusters and different contexts exploiting the standard *kubeconfig* file.

Probes Catalog Probe metadata is represented as a JSON document. Figure 2 shows an excerpt of the metadata for a probe. The reported fields include a human readable description of the probe, the identifier of the probe, the supported targets environments, and the identifier of the collected KPIs (the identifier of the KPI must match the identifier associated with the elements of the tree-model presented to the operators). Our implementation uses MongoDB [19] to store the metadata. The probes can be implemented in any language and their deployment approach is designed within the plug-in. Probes can either push data directly to the TSDB or publish data in a channel (our implementation uses Apache Kafka [27]) to decouple communication between the probes and the TSDB.

Time Series Database Our implementation uses Elasticsearch [11] as TSDB because it offers the highest flexibility in terms of the type of data that can be stored (e.g., it can also store logs and not only numeric KPIs). We already experimented a version of our framework with Prometheus [23] to check the flexibility of the design, but we use Elasticsearch as default option.

3.2 Usage Scenario

In this section, we present two representative usage scenarios of the presented framework: the canonical case of an operator who exploits VARYS to monitor some services, and an engineer who customizes and extends our implementation. A video demonstrating the usage of the framework is available at https://drive.google.com/file/d/1lqkzkI-T2FqHtQilValhQ0B_IRrCS-Ta/view.

Operator who monitors services Let us assume we have Kubernetes running a Content Management System (CMS), composed of a front end, an authentication system, a database, and a session

management service. Let us assume the operator would like to collect KPIs about the amount of resources consumed by the database service and the number of users who are interacting with the CMS. To this end, the operator can login into our VARYS service and thus access the tree with the monitorable KPIs. The operator simply selects the subtree with KPIs about resource consumption, which include CPU, memory, and network consumption, and associates these KPIs with the database service. In addition, the operator selects the KPI that measures the number of users logged into the CMS and applies the KPI to the CMS service. Finally, it submits the request. Note that the operator does not have to perform additional operations, VARYS processes the request and deploys exactly the probes necessary to collect the required KPIs.

In particular, the MaaS Server exploits the metadata associated with the probes to determine the probes that both can collect the selected KPIs and can be deployed on Kubernetes. Moreover, the MaaS server selects the Sidecar monitoring pattern to deploy the probes. Finally, once the full set of probes to be deployed has been determined, it asks the MAB to proceed with the actual deployment. The MAB interacts with the target platform, in this case Kubernetes, to put the probes in place.

Finally, the operator can iterate this scenario several times. For instance, the operator may want to stop collecting KPIs about the CMS and start collecting resource consumption KPIs from the front-end component. The MaaS Server turns this request into the un-deployment and deployment of probes, based on the status of the current deployment, and passes the set of changes to be actuated to the Monitoring Actuation Bridge.

If additional operators use the monitoring system, the MaaS Server exploits information about the deployment to avoid redundancy in the deployment with the respect to the selected pattern.

Engineer who extends the framework VARYS offers useful opportunities also in terms of the extensions that can be operated by engineers who want to target unsupported situations.

If the set of KPIs that can be collected must be arranged differently, the tree-model can be conveniently changed (e.g., adding, moving and dropping elements and subtrees), as long as the correspondence between KPI names and probes metadata is preserved.

If a new KPI must be collected, the engineer has to perform four main steps: (i) updating the tree-model to add the new KPI in the proper subtree; (ii) adding a new probe to the catalog, (iii) annotating the probes with metadata so that the MaaS Server can recognize it; and (iv) providing the deployment scripts required by the available MAB plug-ins.

The actual effort required to complete these steps is variable. If the probe implementation is already available (for instance as Elastic Stack Beats, or Prometheus Exporters), the engineer has only to provide the metadata and the deployment scripts, which can be often reused across scripts. If a new probe must be implemented, the effort depends on the specific probing technology and the observability of the metric that must be collected.

If a new target platform must be supported, engineers have to provide a plug-in implementation that covers the five operations defined in the Monitoring Actuation Bridge.

Overall, this demonstrates that VARYS is a general purpose framework for Monitoring-as-a-Service in Cloud environments.

3.3 Timing Figures

We designed and used VARYS in the context of the NGPaaS EU project [1]. In particular, we assessed the flexibility of our solution by using our technology with several components of telecommunication systems, implemented with different technologies and running on different cloud environments.

Since one of the core capabilities of VARYS is its ability to dynamically change the set of collected KPIs, and consequently change the set of probes running on the target system, we exemplify the cost of these operations for a RESTful Python application that uses a MongoDB database, both deployed in a Kubernetes cluster. We used a simple application on purpose, since the cost of probe deployment is independent on the complexity and size of the monitored target. The used cluster includes our VARYS solution, which runs Apache Kafka, as publish-subscribe communication channel, Logstash to consume data produced by the probes, and Elasticsearch as time series database to store the collected values.

We measured the cost of creating, updating and deleting the full monitoring system when a single replica (1 probe deployed) and three replicas (3 probes deployed) are available. We repeated these measurements considering both the case the Docker probe image is already available (default case in our implementation) and the case the image must be downloaded from the Web (necessary the first time a probe is deployed). We considered two metrics, the time between the request is submitted and the probe is deployed, and the time between probe deployment and the first data entry appears in the time series database. Note that the latter measure is non-deterministically affected by the sampling frequency of the used probes and the metadata refresh period of the Kafka consumer, which is 60sec in our setup. Table 1 shows the results.

Table 1: Timing measurements

Request type	Number of replicas	Cached image	Request → deployment (s)	Deployment → data entry (s)
CREATE	1	NO	13	33
	1	YES	5	
	3	NO	26	33
	3	YES	14	
UPDATE	1	NO	10	17
	1	YES	5	
	3	NO	18	40
	3	YES	16	
DELETE	1	-	6	-
	3	-	23	-

We can observe that VARYS is quite efficient. Probes are deployed, updated, and undeployed in about 5-6 seconds (up to 13 if the probe is deployed for the first time). Values may require some time to appear in the database, depending on sampling frequency, network latency, and channel refresh rate. This may fluctuate between 10 and 40 seconds in average in our setup.

4 CONCLUSIONS

In this paper we presented VARYS, a technology-agnostic MaaS solution for cloud applications. VARYS can be used to address a number of diverse cloud platforms and technologies and to collect virtually any KPI. Moreover, VARYS can be easily extended by engineers, thus providing a single general-purpose monitoring platform that can be used every time monitoring capabilities are required.

Acknowledgements. This work has been supported by the H2020 5G-PPP Phase2 NGPaaS project (Grant Agreement No. 761557) and the H2020 ERC CoG Learn project (Grant Agreement No. 646867).

REFERENCES

[1] 5gPPP. 2014. NGPaaS: Next Generation Platform as a Service. https://5g-ppp.eu/ngpaas/. [Online; accessed 15-May-2019].

[2] G. Aceto, A. Botta, W. de Donato, and A. Pescapè. 2013. Cloud monitoring: A survey. *Computer Networks* 57, 9 (2013), 2093–2115.

[3] Amazon Web Services, Inc. 2019. CloudWatch. https://aws.amazon.com/it/cloudwatch/. [Online; accessed 15-May-2019].

[4] Armin Ronacher. 2019. Flask (A Python Microframework). http://flask.pocoo.org/. [Online; accessed 15-May-2019].

[5] D. Bernstein. 2014. Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing* 1, 3 (2014), 81–84.

[6] B. Burns and D. Oppenheimer. 2016. Design Patterns for Container-based Distributed Systems. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*.

[7] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems* 25, 6 (2009), 599–616.

[8] Jose Alcaraz Calero and Juan Gutierrez Aguado. 2015. MonPaaS: an adaptive monitoring platform as a service for cloud computing infrastructures and services. *IEEE Transactions on Services Computing* 8, 1 (2015), 65–78. https://doi.org/10.1109/TSC.2014.2302810

[9] Elasticsearch BV. 2018. The Elastic Stack. https://www.elastic.co/products. [Online; accessed 15-May-2019].

[10] Elasticsearch B.V. 2019. Elasticsearch Beats - Lightweight Data Shippers. https://www.elastic.co/products/beats/. [Online; accessed 15-May-2019].

[11] Elasticsearch BV. 2019. Elasticsearch: RESTful, Distributed Search & Analytics. https://www.elastic.co/products/elasticsearch. [Online; accessed 15-May-2019].

[12] Evan You and contributors. 2019. Introduction - Vue.js. https://vuejs.org/v2/guide/. [Online; accessed 15-May-2019].

[13] gRPC. 2019. gRPC. https://grpc.io/. [Online; accessed 10-May-2019].

[14] Hewlett-Packard Enterprise Development LP. 2017. Monasca - an OpenStack Community project. http://http://monasca.io/. [Online; accessed 15-May-2019].

[15] ISO. 2011. *ISO/IEC TS 25010:2011(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.* ISO. International Organization for Standardization, Geneva, Switzerland.

[16] ISO. 2017. *ISO/IEC TS 25011:2017(en) Information technology — Systems and software Quality Requirements and Evaluation (SQuaRE) — Service quality models.* ISO. International Organization for Standardization, Geneva, Switzerland.

[17] Anthony D JoSEP, RAnDy KAtz, AnDy KonWinSKi, LEE Gunho, DAViD PAttERSon, and ARiEL RABKin. 2010. A view of cloud computing. *Commun. ACM* 53, 4 (2010).

[18] Hamzeh Khazaei, Rajsimman Ravichandiran, Byungchul Park, Hadi Bannazadeh, Ali Tizghadam, and Alberto Leon-Garcia. 2017. Elascale: Autoscaling and Monitoring As a Service. In *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering (CASCON '17)*. IBM Corp., Riverton, NJ, USA, 234–240. http://dl.acm.org/citation.cfm?id=3172795.3172823 event-place: Markham, Ontario, Canada.

[19] MongoDB, Inc. 2019. MongoDB. https://www.mongodb.com/. [Online; accessed 15-May-2019].

[20] R. Nasim and A. J. Kassler. 2014. Deploying OpenStack: Virtual Infrastructure or Dedicated Hardware. In *Proceedings of the IEEE International Computer Software and Applications Conference Workshops*.

[21] M. Orrú, M. Mobilio, A. Shatnawi, O. Riganelli, A. Tundo, and L. Mariani. 2018. Model-Based Monitoring for IoTs Smart Cities Applications. In *4th Italian Conference on ICT for Smart Cities And Communities i-CiTies 2018*.

[22] Prometheus Authors. 2019. Exporters and Integrations. https://prometheus.io/docs/instrumenting/exporters/. [Online; accessed 15-May-2019].

[23] Prometheus Authors. 2019. Prometheus. https://prometheus.io/. [Online; accessed 15-May-2019].

[24] F. Sabahi. 2011. Cloud computing security threats and responses. In *2011 IEEE 3rd International Conference on Communication Software and Networks*. 245–249. https://doi.org/10.1109/ICCSN.2011.6014715

[25] Salvatore Sanfilippo and contributors. 2019. Redis.io. https://redis.io/. [Online; accessed 15-May-2019].

[26] A. Shatnawi, M. Orrú, M. Mobilio, O. Riganelli, and L. Mariani. 2018. CloudHealth: A Model-Driven Approach to Watch the Health of Cloud Services. In *Proceedings of the 1st International Workshop on Software Health (SoHeal 2018)*. ACM/IEEE, 40–47.

[27] The Apache Software Foundation. 2019. Apache Kafka. https://kafka.apache.org/. [Online; accessed 15-May-2019].

[28] The Kubernetes Authors. 2019. Kuernetes. https://kubernetes.io/it/ [Online; accessed 15-May-2019].

[29] The OpenStack Foundation. 2019. OpenStack. https://www.openstack.org/. [Online; accessed 15-May-2019].

[30] D. Trihinas, G. Pallis, and M. D. Dikaiakos. 2014. JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 226–235. https://doi.org/10.1109/CCGrid.2014.41

[31] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. 2011. Dynamically Scaling Applications in the Cloud. *SIGCOMM Comput. Commun. Rev.* 41, 1 (Jan. 2011), 45–52. https://doi.org/10.1145/1925861.1925869

[32] Y. O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady. 2010. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In *2010 IEEE 3rd International Conference on Cloud Computing*. 91–98. https://doi.org/10.1109/CLOUD.2010.66