Adaptation as a Service

Preprint · October 2018

DOI: 10.13140/RG.2.2.20701.26080

Adaptation as a Service

Hamzeh Khazaei
Department of Electrical and Computer Engineering
University of Alberta
Edmonton, Canada
hkhazaei@ualberta.ca

Alireza Ghanbari
Department of Mathematics
Khatam-ol-Anbia University
Tehran, Iran
arghanbari@kau.ac.ir

Marin Litoiu
Department of Electrical Engineering and Computer Science
York University
Toronto, Canada
mlitoiu@yorku.ca

ABSTRACT

Current and emerging complex systems of many types including but not limited to big data systems, web-based systems, data centers and cloud infrastructure, social networks and the Internet of Things (IoT) have increasingly distributed and dynamic architecture that provide unprecedented flexibility in creating and supporting applications. However, such highly distributed architecture also increases the complexity of end-to-end management of such systems. Due to the sheer complexity, uncertainty and at the same time programmability of cloud environments, microservices and finally big data analytics, it is now required, and possible, to enable autonomic management in distributed systems in a dependable manner. In this paper, we argue that building autonomic management systems is a challenging task and requires its own set of expertise and knowledge. Therefore, in the light of current challenges, available enablers and recent successful stories, we propose the idea of moving from self-adaptation to ADaptation-as-a-Service (ADaaS).

CCS CONCEPTS

• **Software and its engineering** → **Cloud computing**; **Software dependability**;

KEYWORDS

adaptation as a service, microservices, containers, self-adaptation, cloud computing, software engineering.

ACM Reference Format:
Hamzeh Khazaei, Alireza Ghanbari, and Marin Litoiu. 2018. Adaptation as a Service. In *Proceedings of 28th Annual International Conference on Computer Science and Software Engineering (CASCON'18).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

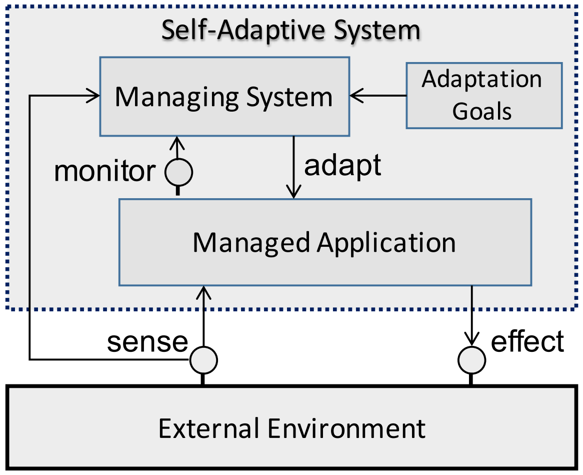

Figure 1: Conceptual model of self-adaptive systems.

1 SELF-ADAPTIVE SYSTEMS

The conceptual model of self-adaptive systems comprises four basic elements: environment, managed system, adaptation goals, and autonomic manager. The managed system comprises the application code that realizes the system's domain functionality. The adaptation goals are concerns of the autonomic manager over the managed system [1]. Four principle types of high-level adaptation goals can be distinguished: self-configuration (i.e., systems that configure themselves automatically), self-optimization (systems that continually seek ways to improve their performance and/or cost), self-healing (systems that detect, diagnose, and repair problems resulting from bugs or failures), and self-protection (systems that defend themselves from malicious attacks or cascading failures) [2]. Figure 1 shows the conceptual model of self-adaptive including both internal and external interactions.

The autonomic management system (AMS) comprises the adaptation logic that deals with one or more adaption goals. The four elements: Monitor, Analyze, Plan, and Execute realize the basic functions of any self-adaptive system. These elements share common Knowledge hence the model is often referred to as the MAPE-K model [2]. Figure 2 illustrates the managing system components which is based on MAPE-K loop model.

One well-known approach to engineer adaptive software systems is architecture-based self-adaptation. The core of architecture-based self-adaptation is the use of a feedback loop accompanied by a runtime analytical/experimental model to monitor the software application behavior during execution, evaluate the model for requirements compliance, and if needed, perform adaptations, either at system, module, or parameter settings [3, 4].

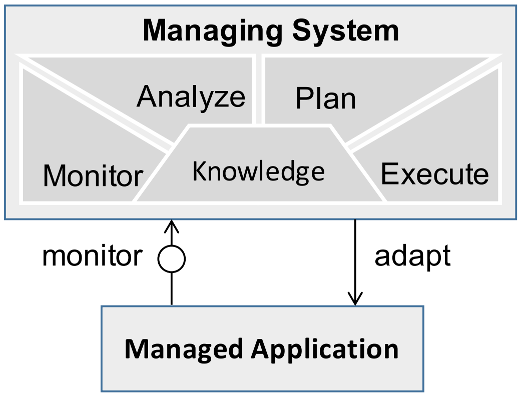

Figure 2: The MAPE-K loop.

2 RECENT DEVELOPMENTS

2.1 Microservices and Containers

Modern cloud applications moved from a monolithic architecture, where the application is built as a single unit, to a distributed architecture, where the application is composed of small and loosely coupled elements, each providing a single functionality. Microservices architectures embody this distributed approach. Microservices are highly decoupled software components that take care of a specific application functionality in a complete way and communicate with each other through lightweight APIs or asynchronous message queues. They are designed to be easily replicated to cope with high workload as needed and, typically, execute within the context of a container that isolates them from any other component and simplifies their scaling on different computation nodes [5].

Microservices are best realized through kernel-level virtualization such as Docker. Docker containers can run an application as an isolated process on a host machine, including only the application and all its dependencies (the kernel of the Operating System is shared among other containers) and providing to it only the resources it requires [6, 7]. Docker allows developers to implement their application and their services using the technology or language that is most suitable to them. Docker containers can be deployed in very different settings, from servers in a cloud computing infrastructure to ARM-based IoT devices [8].

Breaking an application into a web of basic communicating microservices does not solve the challenge of runtime management. What is needed, in part, is a type of container orchestration management to manage a large number of microservices that constitute a distributed application. Container orchestration platforms can be generally defined as a system that provides an enterprise-level framework for integrating and managing containers at scale. Core functionalities of orchestration platforms are cluster state management and scheduling, providing high availability, enabling service discovery, simplifying networking, fault tolerance and security, making continuous deployment possible and finally providing monitoring and governance [9].

2.2 Modern Cloud Computing

Infrastructure as Code, Software-defined infrastructure and programmable infrastructure are describing the same concept in modern cloud data centers. Nowadays, computing, networking, storage and other services in cloud can be provisioned, modified or deprovisioned on the fly by either human or the software systems themselves in a programmable fashion. In other words, applications reside on the cloud may leverage the full elasticity and programmability to autonomously adapt themselves to the new conditions. Current cloud data centers provide well-defined and easy-to-use APIs for collecting monitoring data as well as executing corrective actions at all layers in the cloud stack.

2.3 Big Data Analytics and Machine Learning

As can be seen from Figure 1, self-adaptation is a continues and realtime data-intensive process. Monitoring data, including performance metrics and health check information, are required to be constantly collected, analyzed and processed; then corrective plan(s) may be built and executed on the managed system. For large-scale cloud software systems, this process has become a big data challenge due to speed, diversity and the amount of monitoring data that need to be collected. Scalable data analytic platforms are required to analyze this data in realtime and at rest.

During the last five years big data analytic platforms have been mature enough to be efficiently leveraged for analyzing monitor data. One inborn problem with big data systems though, is the complexity attributed to their configuration and deployment which directly affect the applicability of such systems. This has been a motivation for many to move toward data Analytic-as-a-Service paradigm that can be leveraged to support analysis and plan steps in MAPE-K model.

Machine learning (ML) and statistical techniques are key to transforming big data into actionable knowledge. Existing scalable systems that support machine learning (eg, Spark MLlib[1] and Mahout[2] are typically not accessible to software engineers without a strong background in distributed systems and low-level primitives [10]. Proper configuration of such systems has been proved to be a challenging task [11]. Amazon's Amazon ML [12], Microsoft's Azure ML [13, 14], IBM's Watson [15], Google Cloud ML [16] and BigML [17] are some of the leading providers of MLaaS and are now providing machine learning as-a-service in a scalable and reliable manner. Such services could be leveraged and tuned to build the knowledge base in the MAPE-K model.

3 PROBLEM STATEMENT AND POTENTIAL SOLUTIONS

As described in section 2, leveraging microservices brings about many advantages including maintainability, scalability, DevOps support, fault tolerance and separation of software responsibilities; however, we still need to provision a cluster of resources to run the services, then manage and scale these resources at runtime.

One solution would be to provision computing resources manually and then follow the classic principals of self-adaptation and autonomic computing to handle the management of resource at

[1]spark.apache.orgmllib
[2]mahout.apache.org

runtime. In this approach, software engineers are responsible to develop the autonomic management system (AMS) in addition to the real software system, i.e., the managed application, that is going to fulfill the requirements of end-users. Design and implementation of AMSs gets complicated particularly when the managed application is highly distributed and heterogeneous in nature such as big data and IoT systems. In other words, efficient AMSs can be built for each software system only if we assign a right set of software engineers with proper skills to realize the MAPE-K loop efficiently [18].

Another solution is to leverage new concepts such as serverless computing or fully-manged services to build, in part or in full, the intended adaptive cloud applications [19]. Serverless computing is a style of cloud computing where we write the code and define the events that should cause the code to execute and leave it to the cloud to take care of the rest. AWS's approach to this is called *Lambda*. For Azure it is called *Functions* for Google it is called *Cloud Functions* and IBM has built it's solution based on Apache *OpenWhisk*. The so called "fully-managed services" take the idea of serverless computing one step further because these type of services manages all of the infrastructure resourcing, management, and scaling, along with the workflow needed to carry out the computation [6]. Here, there is no need for the user to allocate or manage resources. Currently, there are fully-managed services for some NoSQL datastores, machine learning algorithms, stream analytics, and messaging systems in popular cloud service providers such AWS, Azure and Google cloud. The drawback here though, is the unavailability of all possible software components as fully-managed services at the moment; also if the software providers or users are not interested in using serverless computing or fully-managed services for any possible reason, then the software stack needs to be deployed and manage by software providers.

Our proposed solution that addresses the shortcomings of the two above is to take the middle ground and let the software (i.e., managed application) to be developed in-house by the software team while delegating the runtime managements (i.e., managing system) to the cloud environment. The following section elaborates separation of concerns between the software owners and the cloud service providers through ADaptation-as-a-Service (ADaaS).

4 ADAPTATION AS A SERVICE

Building upon serverless computing, fully-manged services and cloud-native applications along with further iterations of as-a-service paradigm it is possible to allow cloud service providers to design and deliver services that can take care of non-functional properties for hosted cloud applications. In other words, if cloud service providers adapt hosted software systems according to the owner objectives, there is no need for the application owners to develop their own adaptation solutions. Currently, we have various technologies, methodologies, concepts and best practices available that allow us to imagine providing ADaptation-as-a-Service (ADaaS) for all cloud applications regardless of their internal logic. These enablers include:

(1) Shared-nothing architecture and horizontal scalability
(2) Cloud elasticity and availability
(3) Infrastructure as Code and Software-defined Infrastructure (SDI)
(4) Microservice architecture
(5) Application isolation provided by containers
(6) Software-defined sensors or probes
(7) Advancement in machine learning techniques
(8) Big data analytics capabilities

By leveraging above capabilities, it will be possible to design AMSs as-a-service that provide various adaptations for cloud applications. Table 1 shows the impact of above enablers on the MAPE-K model building blocks in a spectrum starting from nothing to strong.

Elasticity in cloud enables us to plan for any change in computing, storage and networking on the fly at runtime [20]. The main impact of software defined infrastructures is the capability of executing plans in a programmable manner which makes the changes autonomous, instant and durable [21]. Microservices architecture has made the monitoring and planning highly feasible for each component independently [22]. Realizing microservices through containers have strong impact on the whole cycle of MAPE loop as we now dealing with a well-defined segment of the software that can be monitored and changed specifically and without affecting any other part of the software system [23, 24]. Instrumentation of software components has become a straight forward task thanks to software defined sensors such as Elastic Beats [25, 26]. The availability of data, big data management capabilities [27] and the recent advancement in AI and machine learning [28] all promises a bright future to build general-purpose and solid knowledge bases for MAPE-k loop model.

4.1 Security-as-a-Service (SecaaS)

SecaaS usually is provided as SaaS; they charge a monthly subscription fee to reduce cost burden for outsourced services. But instead of providing access to a tool or platform, they provide protection for applications, data, and operations that run in the cloud. By 2020, 85% of large enterprises will use a cloud access security broker solution for their cloud services according to MarketsandMarkets. Currently there exist many SecaaS providers such as Pallera, Proofpoint, White Hat Security, and Juniper, to name a few [3]. Current SecaaS providers offer a portfolio of prevention, detection and resolution services including email encryption, security information and event management, identity and access management, endpoint protection, intrusion detection systems, data loss prevention, network security and disaster recovery as a service [4].

One limitation with this version of SecaaS is that they treat the managed software system as a black box and doesn't change (i.e., adapt) the topology or components of the underlaying application. By leveraging enablers 2, 3, 5, 6 and 7 in table 1, we can promote SecaaS to next level in which it can change the underneath application dynamically to protect/adapt it according to the application owners' needs [29].

4.2 Configuration as a Service (CaaS)

As cloud software systems have been more and more distributed and heterogeneous, the proper configuration (ie, acceptable performance, security, reliability, availability, fault-tolerance, etc) is

[3]https://goo.gl/B3T3in
[4]https://goo.gl/sWpc6k

Table 1: Enablers and their impacts on MAPE-K loop building blocks.

Enablers	Monitor	Analyze	Plan	Execute	Knowledge
1) Shared-nothing arch and horizontal scaling	weak	strong	strong	strong	nothing
2) Elasticity in cloud	nothing	nothing	strong	medium	nothing
3) Infrastructure as code (aka SDI)	medium	weak	nothing	strong	weak
4) Microservice architecture	strong	medium	strong	nothing	nothing
5) Container isolation	strong	strong	strong	strong	nothing
6) Software-defined sensors	strong	weak	nothing	nothing	weak
7) Advancement in machine learning	nothing	strong	nothing	nothing	strong
8) Big data management (analysis and storage)	medium	strong	strong	nothing	strong

challenging to achieve. The Default Configurations of many distributed systems including but not limited to big data platforms and NoSQL datastores are only good enough for "hello-world" applications [30]. For any production or even experimental scenarios, configuration setting is a challenging task and mostly sub optimum. Providing CaaS is in its infancy and needs more attention. The first step for CaaS could be designing and implementation of smart deployers [11]. The primary function of such smart deployers is to find the best possible orchestration of resources for the managed application. For example, for a containerized application, platform or datastore, the smart deployer may identify the proper type and number of containers to be deployed on the infrastructure. One step further is to let the smart deployer choose the right infrastructure, i.e., flavor, size and number of VMs, for the managed application. Reinforcement learning might be a good candidate to build the brain of such smart deployers. All enablers in Table 1 will play important roles in the realization of CaaS in nowadays cloud computing data centers.

4.3 Healing as a Service (HaaS)

In software systems, the self-healing term describes any system or application that can discover that it is not working correctly and, without any human intervention, make the necessary changes to restore the normal or desired state. The goal of self-healing is to make fault-tolerant and responsive system capable of responding to changes on demand and recuperation from failures. Self-healing systems can be divided into three levels, depending on size and type of resources we are monitoring, and acting upon. Those levels include application level, platform level and infrastructure level.

Application level healing is the ability of an application, or a service, to heal itself internally. These type of self-healing capabilities will be implemented by programmers and software engineers during software development life cycle. Software developers may use new tools and libraries such as Akka (https://akka.io) to develop fault-tolerant applications capable of recovery from potentially disastrous circumstances. Self-healing in this context refers to internal processes and cloud service providers cannot help on providing this type of healing as-a-service.

Unlike the application level healing, platform level self-healing can be generalized and be applied to all services and applications, regardless of their internals. This is the type of self-healing that can be provided by cloud service providers as-a-service. To do so, the cloud service provider requires to perform fault detection, location, containment and recovery. For examples Docker, Consul Watches, and Jenkins automatically monitor both underlying infrastructure and deployed services in order to provide self-healing at system level oblivious to the internal code and logic of the application [31].

The real meaning of healing at infrastructure level is redeployment of services from an unhealthy node to a healthy one. As with the platform level, we need to periodically check the status of different hardware components, and act accordingly. In fact, most of infrastructure failures are addressed at the platform level. If a physical node is not working correctly, chances are that the service will fail, and thus will be fixed by platform-level healing such as VM migration or container redeployment. As can be seen, enablers number 2, 3, 4 and 5 in Table 1 empower cloud service providers to provide HaaS at infrastructure and platform levels.

4.4 Optimization as a Service (OaaS)

Self-optimized applications aim to adapt to changes that may occur in their operational contexts, environments and system requirements in an optimized manner. Primary object functions for optimization may include performance, operation cost, reliability, availability and maintainability. Cloud service providers have already begun to provide some basic services to help their customers to optimize cost, performance or balance them; an example would be Amazon auto-scalability features for EC2 instances [32]. Also, we have been witnessing new optimization as services in the platform layer such as IBM Bluemix auto-scalability for microservices among others [33]. One fundamental limitation with above approaches is that they stay at platform level and have no sense of the inside applications. Current auto-scalability feature in cloud, basically monitor the resource utilizations (CPU, Memory, I/O or Network) and scale in/out services if thresholds have been crossed. In many scenarios, high order performance measures such as response time, throughput, blocking probabilities or acceptance rates are far better indicators for planning and executing corrective scalability actions. To this end, software-defined sensors/probes such as Elastic beats may be leveraged to instrument cloud applications and measures such high order performance indicators in a transparent manner. Following this approach, a cloud service provider now can monitor the whole stack including infrastructure, platform and application continuously and perform corrective actions to optimize the application according to the end-user goals. This way, software engineers will be completely liberated from design and implementation of autonomic management systems which enable them to be totally focused on developing the managed application.

5 PROOF OF CONCEPT

Recently, a pattern has been adopted by many software-as-a-service providers in which both virtual machines (VMs) and containers are leveraged to offer their solutions as microservices. This way, strong isolation, preferred for having higher orders of security, inherited from hypervisor-based virtualization will be coupled with flexibility and portability of containers to offer reliable and easy-to-manage services. Scaling algorithms and solutions in such scenarios are required to provide scalability for both VMs and containers.

We designed and developed Elascale [34] to show that auto-scalability for the whole application stack can be provided as-a-service as the proof-of-concept for the idea of adaptation-as-a-service. Elascale is general-purpose, cloud and application agnostic monitoring and auto-scaling solution for modern cloud software systems, which embodies both micro and macro services. In the following we briefly describe Elascale.

5.1 Managed Application: a Generic IoT Application

We developed and deployed a generic distributed IoT application, Fig. 3, on SAVI Cloud (savinetwork.ca) based on the layered architecture presented in [34]. SAVI is a an OpenStack-based two-tier academic-cloud with the core located at the University of Toronto and smart edges located in seven Canadian universities. Our application leveraged the SAVI Core-Cloud and one of the SAVI smart edges located at the University of Victoria, British Columbia. We used SAVI-IoT platform [35], to provision required VMs on the Core and the Edge cloud; Then, a Swarm cluster is created out of provisioned VMs in which the Swarm master is located at the Core-Cloud. All VMs have been labeled and tagged with their roles and locations. Next, the microservices are deployed on top of macroservices, i.e., Infrastructure-as-a-Service (IaaS), at all layers and then were linked together to constitute the target application. We consider this generic IoT application as the "managed application" in self-adaptive systems' terminology.

In the generic IoT application, as sensors, we deploy containerized virtual sensors that collect performance metrics including, CPU utilization, network load and memory consumption of themselves. We refer to these containers as "*virtual-sensor-container*". In other words, each virtual-sensor-container embodies three probe sensors that report resource utilizations every 15 seconds. Every virtual-sensor-container is attached to a co-located aggregator, i.e., a microservice that is running Kafka (kafka.apache.org), automatically. The Kafka service on that aggregator takes the responsibility to forward the aggregated data from virtual sensors to the upper service. Here we set the Kafka service to aggregate sensor data every 60 seconds and then forward them up to a service called IoT-Edge-Processor. This streaming service at the Edge-Cloud aggregates all the received data streams from it's aggregators and simply ingest them into the Cassandra datastore (cassandra.apache.org) located at the Core-Cloud.

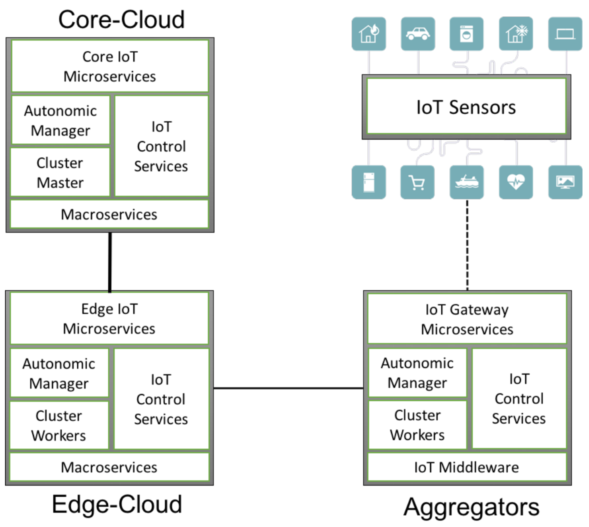

Figure 3: Reference architecture for IoT applications. This layered architecture provides various functionalities at right locations in order to preserve both realtimeness and flexibility [34].

5.2 Elascale

Now we will show how Elascale provides auto-scalability for this generic IoT application without any prior knowledge about the application. Elascale has two important components: (i) an ELKB (Elasticsearch, Logstash, Kibana, Beats) based monitoring stack, and (ii) an auto-scaling engine. Together, the two components provide auto-scalability for any cloud application that has been deployed through microservices. Currently, Elascale purely supports Docker technology family and the plan is to support non-docker solution in next versions, Kubernetes for clustering in particular. Therefore, Elascale V0.9 (current version) assumes and leverages the followings regarding the manged application:

- Docker as the container engine
- Deployment by Docker 'service' or 'stack' commands
- Docker Swarm for clustering
- Docker Machine to manage the backend cloud(s)

Elascale itself is deployed as a microservice on the Swarm Master node or another machine that has access to the Docker engine on the Swarm Master node. The following command deploys Elascale as a microservice on Swarm Master node which is set to scale both microservices and macroservices for the current application:

Listing 1: A sample deployment of Elascale

```
$ docker service create --name elascale
  --env MICRO=TRUE --env MACRO=TRUE
  --env CLUSTER=SWARM --env
  DOCKER_MACHINE=TRUE --constraint
  'node.labels.role==manager' henaras/elascale:0.9
```

The above command triggers scalability for both microservices and macroservices, could be false for each, and assumes Swarm as the cluster management system. Also it assumes that Docker Machine has already been installed and configured. Now we dive down under the hood to see what will happen after issuing above command:

(1) First, it will create a virtual machine and deploys Elasticsearch, Logstash and Kibana as "*replicated*" docker microservices.
(2) Elascale deploys Metricbeat and Dockbeat on all VMs using a "*global*" docker service; this way any newly added node to the Swarm cluster will get these two beats automatically.
(3) Elascale then discovers all microservices and create "*microservice.ini*" that contains the list of application microservices along with default parameters to be used by the Elascale's scaling engine.
(4) In the same fashion, Elascale discovers macroservices, i.e., virtual machines, and create "*macroservice.ini*" file to be customized by the user in the Elascale web UI, like microservices.
(5) After customizing the configuration files by the application owner, the Elascale auto-scaling engine starts monitoring the application's microservices and macroservices based on the active scaling algorithm.

Figure 4 shows the data flow and command flow between target application and Elascale.

The Elascale's scalability engine has been detailed in [34]. Figure 5 shows the orchestration of Elascale and the generic IoT application. As can be seen, Elascale seamlessly integrated into the IoT application without violating any design principles pertaining to microservice architecture.

In this experiment Elascale just deployed two beats (Metricbeat and Dockbeat) to monitor the resource utilization in VMs and containers. However, there exist many other ready-to-use beats from Elasticsearch and the community that can be leveraged by Elascale. These beats (eg. packetbeat, heartbeat, kafkabeat, nginxbeat and many others) can be leveraged to collect application related metrics for many components that we use to build the target application. Elascale may suggest various beats for a particular micro/macro service during the initial setup to be confirmed/filtered by the application owner.

6 CONCLUSIONS

In this paper, we propose and support the idea of providing adaptation for cloud applications as-a-service. Having key enablers such as elasticity in cloud, infrastructure as code, microservice architecture, containers, software-defined sensors, advancement in ML and big data management capabilities at the same time support the feasibility and suitability of ADaptation-as-a-Service (ADaaS) in near future. This is particularly related to the movement toward

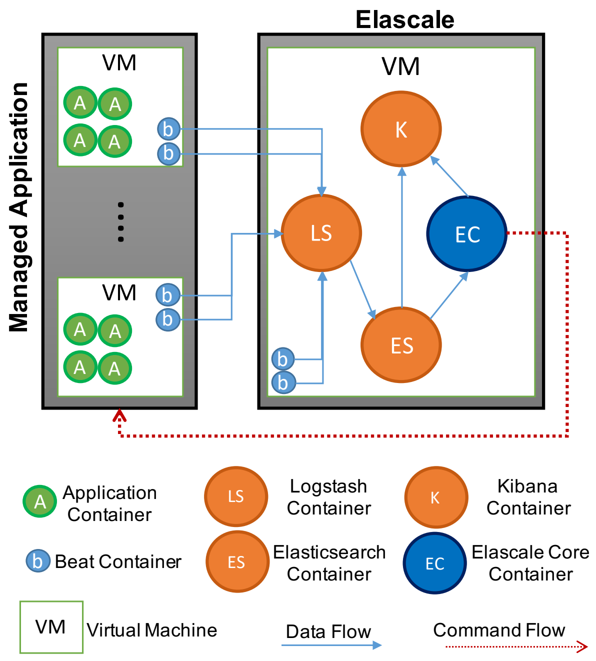

Figure 4: Elascale is agnostic to the underlying cloud and the managed application. All steps from the software instrumentation, data collection, analysis, planning and execution of corrective action are done autonomically [34].

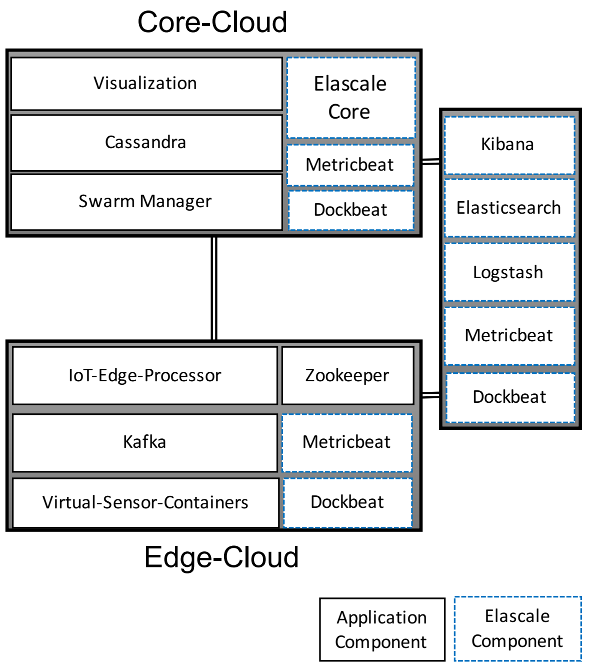

Figure 5: Orchestration of a generic IoT application and the Elascale components [34].

DevOps or even NoOps in which we prefer to remove the runtime operations from the administrators' task list. Another aspect of the proposed idea is to move from self-adaptation, in which the software itself adapts to the new conditions toward the ADaaS, in which the environment (ie, the cloud environment) is smart enough to protect, optimize, configure and heal the software system instead. This way, software developers will be liberated from developing complex autonomic management systems (AMSs) for their applications. From distributed system design point of view, one can see adaptation-as-a-service as a middleware offered by cloud service providers to prevent error-prone and inefficient implementations of AMSs by software engineers who may not be specialized in autonomic computing.

REFERENCES

[1] D. Weyns and T. Ahmad, "Claims and evidence for architecture-based self-adaptation: a systematic literature review," in *Springer ECSA*. Springer, 2013, pp. 249–265.

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[3] S.-W. Cheng, A.-C. Huang, D. Garlan, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," in *International Conference on Autonomic Computing, 2004. Proceedings.*, May 2004, pp. 276–277.

[4] S. Mahdavi-Hezavehi, V. H. Durelli, D. Weyns, and P. Avgeriou, "A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems," *Information and Software Technology*, vol. 90, pp. 1–26, 2017.

[5] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, September 2017.

[6] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, May 2016.

[7] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[8] L. Florio and E. Di Nitto, "Gru: An approach to introduce decentralized autonomic behavior in microservices architectures," in *IEEE International Conference on Autonomic Computing*. IEEE, 2016, pp. 357–362.

[9] A. Khan, "Key characteristics of a container orchestration platform to enable a modern application," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 42–48, September 2017.

[10] T. Condie, P. Mineiro, N. Polyzotis, and M. Weimer, "Machine learning on big data," in *IEEE 29th International Conference on Data Engineering (ICDE)*, April 2013, pp. 1242–1244.

[11] R. Sandel, M. Fokaefs, M. Shtern, H. Khazaei, and M. Litoiu, "To default or not to default: Exposing limitations to hbase cluster deployers," in *International Conference on Computer Science and Software Engineering*. IBM Corp., 2015, pp. 227–230.

[12] A. Inc. (2018, July) Machine Learning on AWS, https://aws.amazon.com/machine-learning. Cloud Services.

[13] Microsoft. (2018, July) Azure Machine Learning Studio, https://azure.microsoft.com/en-ca/services/machine-learning-studio. Cloud Services.

[14] S. Klein, "Azure machine learning," in *IoT Solutions in Microsoft's Azure IoT Suite*. Springer, 2017, pp. 227–252.

[15] IBM. (2018, July) Watson Machine Learning, https://www.ibm.com/cloud/machine-learning. Cloud Services.

[16] Google. (2018, July) Cloud Machine Learning Engine, https://cloud.google.com/ml-engine. Cloud Services.

[17] BigML. (2018, July) Cloud-born Machine Learning, https://bigml.com. Cloud Services.

[18] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke, *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32.

[19] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-native applications," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 16–21, 2017.

[20] N. R. Herbst, S. Kounev, and R. H. Reussner, "Elasticity in cloud computing: What it is, and what it is not." in *ICAC*, vol. 13, 2013, pp. 23–27.

[21] F. Irmert, T. Fischer, and K. Meyer-Wegener, "Runtime adaptation in a service-oriented component model," in *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*. ACM, 2008, pp. 97–104.

[22] K. Baylov and A. Dimov, "Reference architecture for self-adaptive microservice systems," in *International Symposium on Intelligent and Distributed Computing*. Springer, 2017, pp. 297–303.

[23] 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," *Journal of Systems and Software*, vol. 136, pp. 19–38, 2018.

[24] A. G. Keller, "Monitoring containerized applications," Nov. 2 2017, uS Patent App. 15/142,918.

[25] D. Fiedler and S. Dabirsiaghi, "Method and system for runtime instrumentation of software methods," Mar. 29 2018, uS Patent App. 15/818,460.

[26] G. Iuhasz, D. Pop, and I. Dragan, "Architecture of a scalable platform for monitoring multiple big data frameworks," *Scalable Computing: Practice and Experience*, vol. 17, no. 4, pp. 313–321, 2016.

[27] S. Schmid, I. Gerostathopoulos, C. Prehofer, and T. Bures, "Self-adaptation based on big data analytics: a model problem and tool," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017 IEEE/ACM 12th International Symposium on*. IEEE, 2017, pp. 102–108.

[28] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Generation Computer Systems*, vol. 78, pp. 191–210, 2018.

[29] N. Beigi-Mohammadi, C. Barna, M. Shtern, H. Khazaei, and M. Litoiu, "CAAMP: Completely automated DDoS attack mitigation platform in hybrid clouds," in *2016 12th International Conference on Network and Service Management*, Oct 2016, pp. 136–143.

[30] H. Khazaei, M. Fokaefs, S. Zareian, N. Beigi-Mohammadi, B. Ramprasad, M. Shtern, P. Gaikwad, and M. Litoiu, "How do i choose the right nosql solution? a comprehensive theoretical and experimental survey," *Big Data and Information Analytics (BDIA)*, vol. 2, p. 1, 2016.

[31] V. Farcic, *The DevOps 2.0 Toolkit*. Packt Publishing Ltd, 2016.

[32] Y. Guo, A. Stolyar, and A. Walid, "Online vm auto-scaling algorithms for application hosting in a cloud," *IEEE Transactions on Cloud Computing*, 2018.

[33] L. Herger, C. Fonseca, B. Rippon, and M. Bodarky, "Ibm research cloud evolution & efficiencies," in *Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), 2017 IEEE 8th Annual*. IEEE, 2017, pp. 352–355.

[34] H. Khazaei, R. Ravichandiran, B. Park, H. Bannazadeh, A. Tizghadam, and A. Leon-Garcia, "Elascale: Autoscaling and monitoring as a service," in *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2017, pp. 234–240.

[35] H. Khazaei, H. Bannazadeh, and A. Leon-Garcia, "SAVI-IoT: A Self-Managing Containerized IoT Platform," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud*, Aug 2017, pp. 227–234.