Data-driven Adaptation in Microservice-based IoT Architectures

Martina De Sanctis
Gran Sasso Science Institute
L'Aquila, Italy
martina.desanctis@gssi.it

Henry Muccini
University of L'Aquila
L'Aquila, Italy
henry.muccini@univaq.it

Karthik Vaidhyanathan
Gran Sasso Science Institute
L'Aquila, Italy
karthik.vaidhyanathan@gssi.it

Abstract—**Architecting self-adaptive Internet of Things (IoT) systems pose a lot of challenges due to heterogeneity, resource constraints, interoperability, etc. Although microservice architectures (MSA) emerged as a popular solution for developing next-generation IoT systems, they further increase these challenges. This can be attributed to the complexity involved in managing adaptation concerns arising at different levels: i) IoT devices level, due to open and changing contexts, resource constraints, etc; ii) microservices level, due to dynamic resource demands; iii) application level itself, due to the changing user goals. In fact, recent studies have shown that traditional self-adaptation techniques are not flexible enough to be applied to MSA based systems. Moreover, what proposed in the literature handles adaptation either at the architectural level or at the application level. Towards this direction, we propose a self-adaptive architecture for microservice-based IoT systems. In particular, the architecture supports data-driven adaptations, by also leveraging machine learning techniques, and handles adaptations at different levels in a different manner: i) at device level, through a fog layer; ii) at microservice level, by leveraging the use of service mesh; iii) at application level, by means of dynamic QoS-aware service composition.**

Index Terms—**Self-adaptation, microservices, IoT architectures, machine learning, adaptation levels, data-driven adaptation, software architecture**

I. Introduction

The Next Generation Internet (NGI) initiative [1] envisioned the need to re-engineer the Internet of the future as well as modern systems. In fact, modern systems must have a rightful place at the crossroads of the Internet of Services (IoS), the Internet of Things (IoT), and the Internet of People (IoP), thus making them more complex. In particular, dealing with IoT-based technologies brings new challenges, such as *heterogeneity*, *interoperability* and *scalability* that typically identify the IoT domain [2].

In this context, microservice-based architecture (MSA) is considered as one of the best possible solutions for architecting IoT systems [3]. However, new challenges arise when microservices-based solutions are applied to IoT systems. These refer to the evaluation and maintenance of the Quality-of-Service (QoS) characteristics of systems (e.g., performance and reliability) due to the uncertainties faced by IoT devices because of resource constraints (e.g., battery level, network traffic), and by the microservices themselves due to challenges of resource management (e.g., monitoring VMs and containers which can fail/have resource constraints) [4].

Self-adaptation techniques is known to be one of the promising solutions for managing run-time uncertainties [5]. The existing self-adaptation techniques can be classified into two categories, such as proactive and reactive. More specifically, *proactive adaptation* techniques enable the system to foresee any possible QoS issues that may arise at the service or the device levels [5]. Increasingly demanding users with dynamic behaviors, and the *open* nature of the IoT context, call for *reactive adaptation* techniques. Concerning the challenges mentioned above, an ideal microservices-based IoT system should have self-adaptive capabilities to handle the uncertainties in reactive as well as proactive manner. However, a recent study has shown that traditional self-adaptation techniques that make use of MAPE feedback loops or three layers models, in general, may not work in MSA [6]. This is due to a *fundamental mismatch between the adaptation needs of microservice-based systems and the support offered by traditional self-adaptive frameworks and models*. Towards this direction, the approach in [6] proposes a service-mesh based approach for handling adaptation. That approach works at the microservice level, only, while microservice-based IoT systems expose adaptation needs that can arise even from the device level as well as the application level, and they might further impact the adaptation needs of the microservices. This calls for an adaptation framework that encompasses the adaptation concerns arising from these different levels.

In this direction, we propose a novel *self-adaptive architecture for microservice-based IoT systems*. In particular, the architecture handles proactive adaptation by using *machine learning* (ML) techniques, and reactive adaptation by exploiting *dynamic microservices composition*. Furthermore, it manages the adaptation at different levels and in a different manner: i) at device level, through the monitoring of QoS data of IoT devices, through a fog layer, ii) at service level, by continuously monitoring the QoS data of services and effectively leveraging the use of service mesh; iii) at application level, employing dynamic QoS-aware service composition, driven by users goals.

II. Related Work

Different approaches for *self-adaptation in IoT* have been proposed. An approach based on the MAPE-K loop for performing adaptations to automatically manage IoT architectures was proposed by Weyns *et al.* [7]. Model-Driven Engineering (MDE) based approaches were presented in [8] and [9] where the adaptation is carried out using the concept of models@run.time and MAPE-K loop. An agent-based framework for performing self-adaptation for IoT applications was proposed in [10]. Works providing *adaptation in microservice-based* environments have also been presented. Florio *et al.* [11] provided an agent-based approach for handling adaptation through a decentralized MAPE loop. A reference architecture for self-adaptation in microservices using the concept of an adaptation registry was proposed by Baylov *et al.* [12]. Khazaei *et al.* [13], instead, introduced the idea of using self-adaptation as a service for managing adaptations concerns in microservice-based architectures. A self-adaptation service for microservice-based architecture using Kubernetes based on the Rainbow framework was proposed by Aderaldo *et al.* [14]. Yet, a self-healing microservice-based architecture that identifies anomalous behaviors in docker containers using ML was proposed by Magableh *et al.* [15]. To the best of our knowledge, a full-fledged approach, framework or reference architecture for self-adaptation that combines MSA and IoT does not exist.

Differently from the works reported above, our architecture takes into account adaptation concerns that may arise from different levels, such as IoT devices, microservices, and users. It also considers the adaptation challenges that emerge when IoT devices and microservices are used in tandem. Furthermore, besides reactive adaptation, the architecture leverages the use of ML techniques to perform proactive adaptation in scenarios where such adaptation guarantees higher effectiveness.

III. Motivating Example

The NdR Street science fair [16] is an event organized by the University of L'Aquila. It takes place at multiple venues in the city center and witnesses around 25,000 participants every year. Our research group has been invited to develop an application for improving the quality of visiting experience by providing information on events, venues, parking lot, localization, etc. by using various sensors, i.e., people counter, parking mats, beacons, cameras, QR code reader, etc. These sensors are deployed mostly in outdoor environments. Based on this scenario, our plan is to develop a microservice-based IoT application consisting of microservices for handling venue booking, venue management, localization, payment, etc.

In this context, considering the heterogeneity and multitude of involved actors, such as IoT devices, microservices, and users, different kinds of uncertainties have to be managed. They can be classified into three levels, corresponding to the involved actors. For example:

i) **application level:** Suppose users want to use online booking for both the venue and desired transport mode. Embedding in the system behavior all the possible combinations is cumbersome. Possible adaptation than can be to *dynamically* combine the venue booking and selected transport mode microservices (e.g., exhibition event and taxi) to accomplish the user goal.

ii) **microservice level:** Consider that the booking microservice suddenly gets lots of requests due to a popular event and we gather context information from the camera about the flow of people to venues. This information, along with expected response time, can be used to perform proactive adaptation by adding new instance(s) of the booking microservice.

iii) **device level:** Consider that hand-held QR code reader has limited battery capacity. ML can be used to predict the battery level and dynamically adapt the data transfer frequency.

IV. Proposed Architecture

The proposed architecture as depicted in Figure 1 consists of three layers, namely *Edge*, *Fog* and *Cloud*. While the fog layer handles the device level adaptations specific to devices in the edge layer, the cloud layer handles adaptations at the microservice and application levels.

Edge and Fog Layer. The *Edge* layer represents the set of IoT devices (sensors and actuators) in the system. Sensors send the data sensed to the *Fog layer* based on the frequency of data transfer. They also periodically send their QoS data including information such as battery level, memory consumption, etc. to *Fog Layer*.

The *Fog Layer* is responsible for performing the lightweight computations on the sensed data and further perform architectural adaptations/re-configurations on the IoT devices if required based on the QoS data. It consists of multiple *Compute Node* consisting of a *Compute* component and an *Adapter* component. The former is responsible for performing preliminary computations on the sensed data such as data aggregation, cleaning, etc. The later is responsible for leveraging the QoS data obtained from the devices, by using ML models to perform *proactive* adaptations of the IoT devices. The adapter applies some pre-processing such as feature scaling, normalization, etc. on the real-time QoS data of the IoT devices. It then uses ML models to predict the expected QoS for a given time interval. These models are periodically received from the cloud layer. The Adapter then selects an adaptation plan, to be used if any QoS issue is forecasted, and communicates it to the compute component. The selected plan is then used to perform device-level adaptation, such as reduce the sensor data transfer frequency, modify the communication protocol, etc (eg. device level in Section III). The *Fog Layer* further communicates the data received, which includes the QoS data as well as the sensed data, to the *Cloud Layer* through the *Message Broker*.

Cloud Layer. This layer performs heavyweight computations. It consists of four main layers. (Despite the ordering depicted in Figure 1, we describe, in the order, microservice, management, adaptation and application infrastructure layers, for comprehensiveness).

1) Microservice Layer: It consists of the set of microservices implementing the functionalities of a given IoT system. For monitoring the individual microservices and further use this for performing adaptation, we use the concept of *service-mesh/sidecar*, as suggested by Mendonça *et al.* [6], [17]. The service-mesh provides ways to monitor various QoS parameters of each microservice, such as traffic, response time,

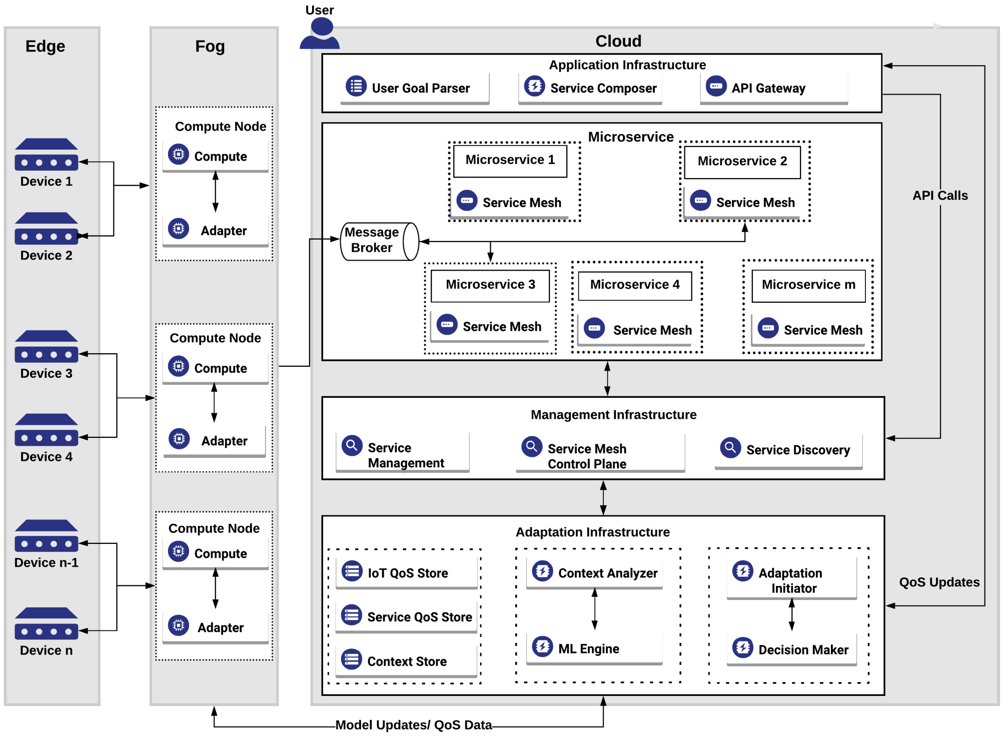

Fig. 1. Self-Adaptive architecture proposed for microservice based IoT Systems

etc., that can be obtained from the *Service Mesh Control Plane* in the *Management Infrastructure layer*.

2) Management Infrastructure Layer: It handles the discovery of microservices, provides information on their status, manages them and executes adaptation if needed. It consists of three main components: i) *Service Management*, it is responsible for executing the architectural adaptation of the microservices, e.g., increasing the memory of the microservice instance, automatically add an instance, etc. based on the adaptation decision provided by the lower layer. It is also responsible for providing information on the status of the microservices upon request to the ii) *Service Mesh Control Plane*, it regularly monitors the QoS level of every microservice. These QoS data are further sent to the *Adaptation Infrastructure layer* for processing; iii) *Service Discovery*, it routes the requests from the *API Gateway* to the respective instances of microservices.

3) Adaptation Infrastructure Layer: This is a dedicated layer providing mechanisms for effectively supporting adaptations at different levels. It is responsible for collecting the context and QoS data from the *Fog* and *Management Infrastructure layers*. It leverages these data to generate learning models for QoS prediction and further decide on the best adaptation strategy based on the context. It consists of:

i) *IoT QoS Store*, ii) *Service QoS Store* and iii) *Context Store* stores information such as device-level QoS, service level QoS, and sensor data from Fog Layer, respectively.

iv) *ML Engine* is the key component as it is responsible for leveraging the data obtained to create forecast models that can predict the expected QoS of IoT devices and microservices. It mainly uses two data sources, IoT and Service QoS Stores. These data form time-series datasets consisting of the QoS values for different intervals of time. ML Engine applies pre-processing steps on these datasets, such as feature scaling, data aggregation, etc. These pre-processed data are then used for creating forecast models, by using LSTM networks [18] as defined in [19]. This process is repeated periodically to ensure continuous update of the models thereby avoiding the possible issue of concept drift [20]. It results in the creation of two types of models: 1) for forecasting the QoS of IoT devices, which are communicated periodically to the Fog layer, and 2) for forecasting the QoS of microservices, which are sent to the *Context Analyzer*;

v) *Context Analyzer* is responsible for identifying the need for adaptation of microservices based on the QoS forecast models from the ML engine and the data from the context Data Store. It follows a two-step process. First, at every instant of time, it obtains the latest Service QoS data to forecast the expected Service QoS for a given interval of time, using the forecast model. This is then used to identify any possible QoS issues in any of the microservices. Second, it uses the context data to gather specific information (refer Section III (microservice level). It combines these data to identify the need for adaptation and triggers the decision-maker if an adaptation need arises.

vi) *Decision Maker* is responsible for identifying the best adaptation technique based on the information from the context analyzer. It uses a set of adaptation techniques that consists of adaptation options, such as dynamically scaling microservices, auto-rollback and restarting microservices, etc.

These techniques can also be combined to form more complex *strategies*. In particular, to this end, we can exploit the Q-Learning technique as defined by Muccini *et al.* [19] or AI planning as defined by Bucchiarone *et al.* [21]. For instance, in the (microservice level) scenario in Section III, the *Context Analyzer* can trigger the adaptation based on the forecasted response time of the booking service and context data. Based on this, a new instance can be added. This decision is then communicated to the *Adaptation Initiator*.

vii) *Adaptation Initiator* acts as a bridge between the decision-maker and the *Management Infrastructure layer*, by forwarding the adaptation request to the higher layer.

4) Application Infrastructure Layer: Its purpose is to execute application-level adaptation based on their goals. In fact, some of the functionalities provided by the application are specified only as abstract goals (e.g., an application-level scenario in Section III) that can be dynamically refined at runtime, through a composition of microservices whose execution allows users to achieve the goals. It consists of three main components: i) *User Goal Parser:* it parses the abstract goals and translates them to the format as required by the *Service Composer*. Goals can be specified through various goal models available in the literature, for instance, the one used in [22]. ii) *Service Composer:* it uses the QoS data/forecasts available from the lower layer to decide on the best composition of microservices, driven by the abstract goals from the user goal parser. The composition can then be performed, for instance, by using an AI planning method as suggested by De Sanctis *et al.* [22]. The identified composition of microservices is sent back to the user application, which then uses the *API gateway* to invoke the respective microservices. For those scenarios where the service composition is not required, e.g., *login* operation, the user request is directly routed to the *API Gateway*. iii) *API Gateway:* it performs the routing of the requests from the user to the corresponding instances of the microservices through the service discovery component.

The approach can be realized by using a combination of various technologies, such as *Apache Kafka* (Message broker); *Istio* (service mesh); *Kubernetes* and *Docker* (deploying and managing microservices); *Apache Zookeeper* (service discovery); *Elasticsearch* (Data stores); *Grafana* (visualization), and *Keras with Python* (machine learning models).

V. Conclusion and Future Work

We proposed a self-adaptive architecture for microservice-based IoT systems. Future work includes a concrete implementation of the architecture for the NdR application to show its effectiveness and efficiency. This shall be performed by realizing the architecture using the technologies mentioned above. The effectiveness shall be obtained by measuring the accuracy of predictions made, energy saved for IoT devices, the degree of user goals achieved, the average response time of microservices, etc. The efficiency shall be measured based on the quality of the adaptations performed. The implementation shall also be extended to consider different challenges such as DevOps integration, testing, etc. as mentioned in [17].

References

[1] E. Commission, "Next Generation Internet initiative." 2019, https://ec.europa.eu/digital-single-market/en/policies/next-generation-internet.

[2] A. Taivalsaari and T. Mikkonen, "A roadmap to the programmable world: software challenges in the iot era," *IEEE Software*, vol. 34, no. 1, pp. 72–80, 2017.

[3] Microsoft, "Azure iot reference architecture, ver. 2.1," 2018, https://aka.ms/iotrefarchitecture.

[4] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[5] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184–206, 2015.

[6] N. C. Mendonça, D. Garlan, B. R. Schmerl, and J. Cámara, "Generality vs. reusability in architecture-based self-adaptation: the case for self-adaptive microservices," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA 2018*, 2018, pp. 18:1–18:6.

[7] D. Weyns, M. U. Iftikhar, D. Hughes, and N. Matthys, "Applying architecture-based adaptation to automate the management of internet-of-things," in *European Conference on Software Architecture*. Springer, 2018, pp. 49–67.

[8] F. J. A. Padilla, "Self-adaptation for internet of things applications," Ph.D. dissertation, 2016.

[9] F. Ciccozzi and R. Spalazzese, "Mde4iot: supporting the internet of things with model-driven engineering," in *International Symposium on Intelligent and Distributed Computing*. Springer, 2016, pp. 67–76.

[10] N. M. do Nascimento and C. J. P. de Lucena, "Fiot: An agent-based framework for self-adaptive and self-organizing applications based on the internet of things," *Information Sciences*, vol. 378, pp. 161–176, 2017.

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

[12] K. Baylov and A. Dimov, *Reference Architecture for Self-adaptive Microservice Systems*. Cham: Springer International Publishing, 2018, pp. 297–303.

[13] H. Khazaei, A. Ghanbari, and M. Litoiu, "Adaptation as a service." 2018.

[14] C. M. Aderaldo, N. C. Mendonça, B. Schmerl, and D. Garlan, "Kubow: An architecture-based self-adaptation service for cloud native applications," in *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ser. ECSA '19, 2019, p. 42–45.

[15] B. Magableh and M. Almiani, "A self healing microservices architecture: A case study in docker swarm cluster," in *Advanced Information Networking and Applications*. Springer International Publishing, 2020, pp. 846–858.

[16] "Notte dei ricercatori aq," 2019, https://nottedeiricercatoriaq.it/.

[17] N. C. Mendonça, P. Jamshidi, D. Garlan, and C. Pahl, "Developing self-adaptive microservice systems: Challenges and directions," *IEEE Software*, 2019.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] H. Muccini and K. Vaidhyanathan, "A machine learning-driven approach for proactive decision making in adaptive architectures," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2019, pp. 242–245.

[20] A. Tsymbal, "The problem of concept drift: definitions and related work," *Computer Science Department, Trinity College Dublin*, vol. 106, no. 2, p. 58, 2004.

[21] A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, and P. Traverso, "Incremental composition for adaptive by-design service based systems," in *IEEE International Conference on Web Services, ICWS*, 2016, pp. 236–243.

[22] M. De Sanctis, R. Spalazzese, and C. Trubiani, "Qos-based formation of software architectures in the internet of things," in *Software Architecture - 13th European Conference, ECSA 2019*, 2019, pp. 178–194.