State of the Art on Microservices Autoscaling: An Overview

João Paulo K. S. Nunes[1], Thiago Bianchi[2], Anderson Y. Iwazaki[3], Elisa Yumi Nakagawa[3]

[1] IBM, São Paulo, Brazil

[2]Itaú Unibanco S.A., São Paulo, Brazil

[3]University of São Paulo, São Carlos, Brazil

karolsn@br.ibm.com, thiago-bianchi@itau-unibanco.com.br

iwazaki.anderson@usp.br, elisa@icmc.usp.br

Abstract. *The adoption of microservices architecture has taken on great proportions due to its benefits and popularization of containers driven tools, such as Kubernetes and Docker. Besides, the development of microservice-based applications is a complex task, specially because they can be composed of multiple heterogeneous parts. In particular, one of the main challenges is how to conduct the microservices autoscaling (i.e., adding or removing resources on demand), while still avoiding resource waste, such as CPU and memory. This paper presents the state of the art of approaches to solve the problem of microservices autoscaling, the main characteristics to be considered as well as the important future directions that need to be still investigated.*

1. Introduction

Microservices have gained considerable popularity over the last years due to their inherent nature of decoupling a given application in multiple components, allowing flexible management, including individualized scaling operations. Besides, microservices perfectly fit with native cloud-based applications due to their capacity of using containers. In turn, a microservice can be defined as a "small" application that can be deployed independently, scaled and tested independently, and has a single responsibility [Thönes 2015].

A big challenge for developing microservice-based applications is the selection of effective autoscaling approaches for microservices [Abdel Khaleq and Ra 2019]. Such autoscaling refers to the capacity of automatically increase or decrease resources used by cloud-based applications, thereby adapting resource usage to the applications' requirements [Calcavecchia et al. 2012]. There is a variety of autoscaling approaches with diverse characteristics and elements, such as reactive and predictive operations, adoption or not of machine learning techniques, orchestration frameworks (e.g., Kubernetes and Docker Swarm), cluster deployment, and others. They differ from any other autoscaling approaches applied in other areas, mainly regarding the hardware virtualization. In this perspective, the real-world problem addressed in this paper is how architects can manage the variety of issues involving microservices autoscaling. To the best of our knowledge, there is a lack of studies that put together and analyze the existing autoscaling approaches.

The main goal of this paper is to present the state of the art on microservices autoscaling as well as its types, strategies and main characteristics. To do this, we systematically looked for and selected possibly all existing autoscaling approaches and analyzed

them. From the results of our analysis, we observe the topic of microservices autoscaling is relatively new and a prolific area to be exploited yet. In particular, there is still an urgent need for establishing a clear definition of what to expect from a successful autoscaling approach.

This work is organized as follows. Section 2 describes the research method used in our study. Section 3 presents the results of our analysis, while Section 4 is a discussion of the results and points out important next steps. Section 5 concludes this paper.

2. Research Method

We conducted a systematic mapping study (SMS) to obtain the state of the art on microservices autoscaling and, for that, we followed the process proposed by [Kitchenham et al. 2015][Felizardo et al. 2017] that presents three main phases (planning, conduction, and reporting). During the planning, we defined the protocol, whose main elements are the research questions (RQ) and the search strategy. We defined three questions:

- RQ1: Which are the existing autoscaling approaches available for containerized microservice-based applications?
- RQ2: How are implemented the approaches available for containerized microservice-based applications?
- RQ3: How are the autoscaling approaches being evaluated and validated in terms of tools and metrics?

For the search strategy, we defined the search string and selected the source of studies. After several calibrations, the final search string used in our SMS was *((“scaling”) AND (“container”) AND (microservice OR micro-service OR “micro service” OR kubernetes OR “docker swarm”)).* Besides, four publication databases were selected as source of studies: *ACM Digital Library*[1], *IEEE Xplorer*[2], *Scopus*[3], and *Google Scholar*[4]. We also defined one inclusion criteria (IC) and six exclusion criteria (EC) to select studies:

- IC1: Study presents microservice scaling;
- EC1: Study does not present microservice scaling;
- EC2: Study is not accessible;
- EC3: Study is just a summary of a conference;
- EC4: Study is not written in English;
- EC5: Study is published as an abstract; and
- EC6: Study only mention microservice scaling with no details.

Four reviewers performed the conduction phase of our SMS. In Step 1, we identified primary studies from the databases and obtained 969 studies. During Step 2, we selected the primary studies by reading titles and abstracts and applying inclusion and exclusion criteria, resulting in 80 studies. In Step 3, these studies were read in full and inclusion and exclusion criteria were applied again and, 53 studies[5] were considered relevant for answering our RQs. We used a data extraction form to obtain all relevant data

[1] https://dl.acm.org
[2] https://ieeexplore.ieee.org/Xplore/home.jsp
[3] https://www.scopus.com
[4] https://scholar.google.com/
[5] The list of studies is available in https://github.com/joaopauloksn/autoscaling/blob/main/README.md

from the identified studies that supported us to classify and analyze the studies and answer our RQs, as reported in the next section.

3. Results

3.1. Microservice Autoscaling Approaches

This section presents the answer for RQ1, which addresses approaches for autoscaling microservices, their types, strategies, and main characteristics. Figure 1 presents the distribution of studies in regards to the autoscaling types (vertical and/or horizontal) and strategies (reactive and/or proactive). Most studies are related to the type "horizontal" and strategy "reactive", while "vertical" and "proactive" have received to some extent less attention. The type "vertical" is more frequent in combination with "horizontal" scaling. Below, each type and strategy is discussed, followed by a discussion on the adoption of machine learning in such approaches. In the context of this work, we highlight the most representative studies; the other studies are listed in the external material.

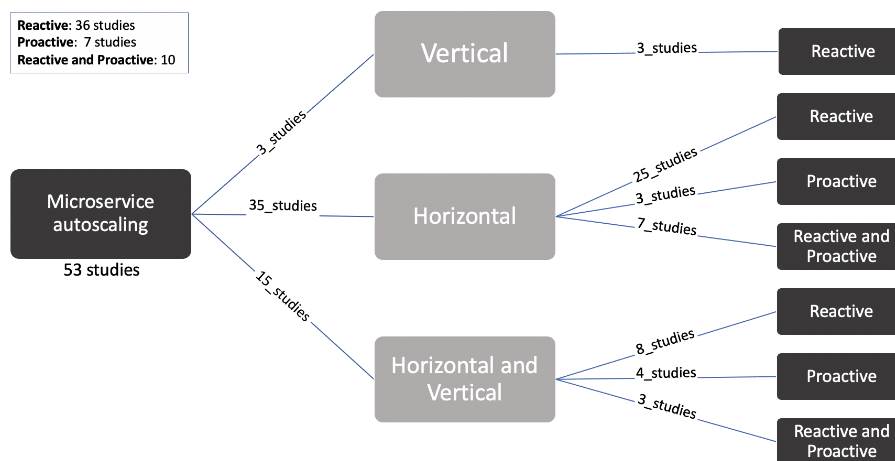

Figure 1. Distribution of studies by types and strategies of microservice autoscaling

3.1.1. Types of microservice autoscaling

Scaling type defines how the autoscaler decides the method to provision resources and what combination of resources is provisioned to the application, depending on the particular environment, scaling can be performed vertically, horizontally, or both [Imdoukh et al. 2020]. This is a well-known concept already used to define scaling types in other areas, such as Cloud and Computer Network.

Horizontal scaling, increases or decreases the number of replicas or instances of the same microservice. In a Kubernetes-based environment, it is referenced by increasing or decreasing the number of replicas of the same POD. When an application requires more computational resources, instead of having to adjust the specifications of the existing pods, users can simply create another identical pod to share the load [Nguyen et al. 2020]. In an environment purely using Docker containers (without any orchestration framework), it would be analogous to refer only to the addition or reduction of one or more containers

of the same service. In the case of Vertical scaling, it is about the increase or decrease in resources used by a given microservice within its replication or instance. For example, in vertical scaling, we could increase or decrease the amount of CPU and available memory for a given instance of microservices.

The main characteristics of each type of microservice autoscaling are presented following:

- **Horizontal scaling characteristics:** While applications can be scaled vertically by allocating more resources to a running instance of a microservice, the dominant mechanism found in production cloud computing environments remains the mentioned horizontal scaling in which replicas of the same service are added and removed on demand [López and Spillner 2017]. It can also be seen in the number of related studies, see Figure 1.
  Horizontal autoscaling is generally used when there is an increase or decrease in the number of service requests, usually triggered by either a higher or lower demand of a given service. An IoT application for example can have its demand increased after the entry of one or more devices connected simultaneously. In this case, without horizontal autoscaling, the service can be overloaded with the number of requests, reducing performance or even causing service instability. One way to solve this problem without horizontal autoscaling would be to estimate the number of replicas needed to satisfy the maximum load peak, always adding enough resources to satisfy this condition. In this case, resource waste will occur, because the service will not be using the resource assigned all the time. In a cloud environment, it can be expensive, since it charges for the amount of resources being used.
  The most popular tool available on the market used is the HPA[6] (Horizontal Pod Autoscaler), a default and ready to use autoscaling feature. It depends on manually setting up some threshold values such as the target CPU utilization, minimum and maximum number of pods [Abdel Khaleq and Ra 2019].
- **Vertical scaling characteristics:** Vertical autoscaling is generally used when we have either a large or short use of resources in the same instance to satisfy the service requests. As microservices will perform from very simple to more complex tasks, they might require more or fewer resources in a given period. For example, a microservice performing a machine learning task may behave quite differently along its journey. While some requests can be simple, others can be more complex, requiring more resources from the same replica, such as CPU and memory.
  As horizontal scaling, many times the solution without using an appropriate autoscaling approach, is to estimate the maximum needed resources that the given microservice will use and make it available. This again can lead to a waste of resources and consequently high costs. Kubernetes also provides a default mechanism for vertical autoscaling, VPA[7] (Vertical Pod Autoscaler). When configured, it will set the requests automatically based on usage and thus allow proper scheduling onto nodes so that the appropriate resource amount is available for each pod.
  Despite not being the object of study by the majority of the reviewed studies (see Figure 1), vertical scaling does have high relevance in the area.

[6] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
[7] https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler

- **Using vertical and horizontal scaling in the same method:** Hybrid scaling techniques reap the benefits of both the fine-grained resource control of vertical scaling and the high availability of horizontal scaling. This makes hybrid scaling a promising solution for effective autoscaling [Kwan et al. 2019]. When proposing a generic method where there is no previous knowledge about the application, it is important to analyze which of the types of autoscaling would be most suitable for resource optimization. Before choosing between one of the scheduling types, an initial phase of understanding the characteristics of the autoscaling demands is needed. Preferably, a complete method should be able to identify the best time to autoscale resources either horizontally or vertically. By fully exploiting elasticity, an application can more quickly react to small workload variations, through fine-grained vertical scaling, as well as to sudden workload peaks, through horizontal scaling [Nguyen et al. 2020].

3.1.2. Strategies for microservice autoscaling

Another important factor in microservices autoscaling is how to scale, either reactive or proactive. This definition was proposed by [Lorido-Botran et al. 2014] from the cloud IaaS autoscaling perspective. This is also commonly used to represent microservices autoscaling types.

In reactive autoscaling, the system automatically adapts to the requested demand. When necessary it will scale up or down, in general, due to the increase and decrease in requests. For example, a threshold-based system is prepared to react if a service exceeds the defined threshold max or min values. In case of high demand, the system will either increase the resources of an instance (vertical scaling) or create new replicas of the service (horizontal scaling). The same is true in low demand where the system will reduce resources or decrease the number of replicas.

Proactive autoscaling uses sophisticated techniques to predict future demands to arrange resource provisioning with enough anticipation [Lorido-Botran et al. 2014]. It helps to decide to scale up or down according to a predetermined forecast usually obtained through statistical models or machine learning.

Following, we characterize the strategies for microservice autoscaling:

- **Reactive autoscaling characteristics:** Because of its nature of reacting to an existent service request demand, reactive autoscaling presents some challenges to be solved, such as cold initiation of replicas (horizontal scaling), delay in the application of additional resources (vertical scaling), and mainly a prior knowledge of the characteristics of services being monitored. Prior knowledge is required when setting thresholds that will support the autoscaling decision. Another important aspect mentioned in most studies is the stabilization time to avoid excessive scaling fluctuation. For example, a service that exceeds 80 of the established CPU usage should be replicated only if that excess exceeds 3 minutes in duration. This example avoids scaling the service up and down simultaneously because of a sporadic increase in demand, causing cluster overhead.
- **Proactive autoscaling characteristics:** As in reactive autoscaling, predictive scheduling also has its particular challenges. Among them is the difficulty of

establishing reliable initial criteria due to the lack or little data for training; the demand for better monitoring metrics; and the need for a historical data structure, such as the maintenance of a database. In general, predictive autoscaling has a more complex architecture than reactive autoscaling, especially because of the demand for statistical models and machine learning algorithms. Well-applied predictive scheduling is shown to be more efficient than reactive scheduling as shown by [Zhao et al. 2019], mainly due to the capacity of providing additional resources before the actual demand, avoiding the delay during resource allocation. Another important advantage is the knowledge acquired during the use of micro-services and its learning capacity.

- **Using reactive and predictive autoscaling in the same method:** In the simplest cases or with little complexity, the reactive autoscaling itself already has decent results and will cover most of the scenarios. As it is more complex, predictive scheduling can have an additional development cost due to its complexity. It worth noting that in most of the papers presented for predictive scheduling, reactive scheduling is also used as a fallback mechanism at the beginning of the service provisioning, since we do not have enough data to make accurate decisions. Therefore, it is understood that a complete and generic method makes use of both types of scheduling.

3.1.3. Adoption of machine learning

The recent developments in artificial intelligence and machine learning contribute to building general-purpose autoscalers [Imdoukh et al. 2020]. Machine learning techniques are most commonly related to proactive scaling strategy, but they are also employed in the reactive strategy scope as highlighted in [Fourati et al. 2019] that propose the adoption of an anomaly detection system to support scaling decisions for both vertical and horizontal scenarios.

During our review, the most common machine learning methods identified for microservices autoscaling were based on regression [Ye et al. 2017], genetic algorithms [Guerrero et al. 2017], and neural networks [Imdoukh et al. 2020]. According to [Imdoukh et al. 2020], a neural network can learn from past scaling decisions and workload behavior to generate scaling decisions ahead of time.

For non-machine learning techniques, most proactive methods use statistical models based on Gaussian functions and correlation, there are also mathematical models based on differential calculus referred to in the searched literature [Cerqueira De Abranches and Solis 2016].

It is noteworthy that most of the works presented in the machine learning area are in their early stages of development and experimentation; however, there are few already deployed in the production environment, for example, Autopilot stands out: workload autoscaling on Google [Rzadca and et al. 2020].

3.2. Implementation of Microservice Autoscaling Approaches

This section presents the answer for RQ2, which addresses the main tools and infrastructure used in the deployment of the microservice autoscaling approaches. Container

deployment is usually associated with an orchestration tool that plays an important role in microservice autoscaling, especially because it provides up front the architecture and infrastructure required to implement an autoscaling solution. Most studies of our SMS point out the two platforms for deploying container autoscaling solutions: Kubernetes[8] and Docker Swarm[9].

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. It is by far the most used technology in the proposed autoscaling microservices approaches, and its high adoption by the industry explains this preference of most papers we analyzed. Accordingly to the Datadog research[10], today, half of the organizations running containers use Kubernetes.

In turn, Docker Swarm is the cluster management and orchestration features embedded in the Docker Engine. Despite being way behind Kubernetes in popularity, it still presents a relevant amount of work in the microservice autoscaling area.

In addition to Kubernetes and Docker Swarm, some researchers have based their work on other platforms and provided methods where autoscaling was agnostic concerning the host platform. For instance, [Kampars and Pinka 2017] proposed an autoscaling architecture for cloud platforms in general, not limited to a specific tool or platform.

3.3. Tools and Metrics for Microservice Autocaling

This section presents the answer for RQ3, which addresses the way researchers have evaluated and validated microservice autoscaling approaches. The most common validation metrics are related to the SLO (Service Level Objectives), whose goal is to establish the metrics values the application is required to meet. An autoscaling method will not always seek for perfect performance or 100% uptime, what will dictate the use of resources is what is defined in the SLO. The autoscaler must be aware of the economical costs of its decisions, which depend on the pricing scheme used by the provider, to reduce the total expenditure [Lorido-Botran et al. 2014].

An autoscaling system requires the support of a monitoring system providing measurements about user demands, system (application) status and compliance with the expected SLA [Lorido-Botran et al. 2014]. It is important to well define the type of metrics collected, as they directly impact the accuracy of the autoscaling decisions. Table1 lists the commonly used metrics for microservice autoscaling.

Metric	Usage
CPU and RAM	Common metric usually provided by container
Response Time	Metric type largely used. It is part of Service Level Agreement
Number of Requests	Metric used in conjunction with the deployment of a load balancer. Mostly seen in horizontal scaling.
Custom Metrics	Adds application knowledge to the autoscaling model and can possibly improve accuracy

Table 1. Commonly used microservice autoscaling metrics

[8]https://docs.docker.com/engine/swarm/key-concepts/
[9]https://kubernetes.io/
[10]https://www.datadoghq.com/container-report/

4. Discussion

Microservice autoscaling can be considered a relatively new research topic with a little more than 50 studies. There is still no consensus of the types, strategies, and machine learning techniques and their combination that could work better.

Concerning the threats to the validity of this work, some potential threats could have affected its validity. We addressed all threats (according to [Zhou et al. 2016]):

- *Construction validity*: It consists of identifying the correct conduction of an SMS. To mitigate threats to this validity, we systematically developed and followed the protocol to ensure its completeness.
- *Internal validity*: It consists in the rigorous conduction of an SMS, e.g., the use of a well-defined search strategy and correct conduction of data analysis and synthesis. For that, all recommendations for the SMS process were followed, including the assurance of a broader number of relevant studies; however, some studies may have been neglected.
- *External validity*: It is related to the findings' generalizability over the primary studies and accessibility of these studies and databases. Multiple databases were used to reduce subjective errors during the conduction phase.
- *Conclusion validity*: It consists of interpreting results that could be subjective. To mitigate a threat to this validity, all authors of this SMS completely participated in all SMS steps, including in the synthesis of results.

5. Conclusions

Based on the state of the art presented in this paper, it is worth highlighting horizontal autoscaling using reactive and proactive strategies has drawn more attention from academia and industry. There are also benefits of using hybrid approaches (i.e., vertical and horizontal; reactive and proactive) and combining them with machine learning techniques to mainly predict workload and define threshold values. In the nutshell, no unique solution efficiently serves all types of microservices. From the perspective of target deployment platform or container orchestration tools, although few authors refers to platform agnostic approaches, Kubernetes and Docker Swarm are the most common used by the state-of-the-art autoscaling methods.

As the main future work, it is necessary to investigate and evaluate different implementations towards reliable and efficient solutions for microservice autoscaling. It is also necessary to perform evaluations of various combinations together with new metrics to verify quality attributes, including efficiency, scalability, portability, and sustainability, also considering different real-world scenarios. Additionally, the list of studies presented in this paper could be complemented by increasing the scope of this SMS and providing additional information, such as categorization, metrics used, and target deployment platforms.

References

Abdel Khaleq, A. and Ra, I. (2019). Agnostic approach for microservices autoscaling in cloud applications. In *CSCI*, pages 1411–1415.

Calcavecchia, N., Caprarescu, B., Di Nitto, E., Dubois, D., and Petcu, D. (2012). Depas: A decentralized probabilistic algorithm for auto-scaling. *Computing*, 94.

Cerqueira De Abranches, M. and Solis, P. (2016). An algorithm based on response time and traffic demands to scale containers on a cloud computing system. In *NCA*, pages 343–350.

Felizardo, K., Nakagawa, E., Fabbri, S., and Ferrari, F. (2017). *Systematic Literature Review in Software Engineering: Theory and Practice*. Elsevier Brazil (in Portuguese).

Fourati, M. H., Marzouk, S., Drira, K., and Jmaiel, M. (2019). Dockeranalyzer: Towards fine grained resource elasticity for microservices-based applications deployed with Docker. In *PDCAT*, pages 220–225.

Guerrero, C., Lera, I., and Juiz, C. (2017). Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. *Journal of Grid Computing*, 16(1):113–135.

Imdoukh, M., Ahmad, I., and Alfailakawi, M. G. (2020). Machine learning-based auto-scaling for containerized applications. *Neural Computing and Applications*, 32(13):9745–9760.

Kampars, J. and Pinka, K. (2017). Auto-scaling and adjustment platform for cloud-based systems. In *ISPC*, pages 52–57.

Kitchenham, B., Budgen, D., and Brereton, O. (2015). *Evidence-Based Software Engineering and Systematic Reviews*. CRC Press.

Kwan, A., Wong, J., Jacobsen, H., and Muthusamy, V. (2019). Hyscale: Hybrid and network scaling of dockerized microservices in cloud data centres. In *ICDCS*, pages 80–90.

Lorido-Botran, T., Miguel-Alonso, J., and Lozano, J. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592.

López, M. and Spillner, J. (2017). Towards quantifiable boundaries for elastic horizontal scaling of microservices. In *UCC*, pages 35–40.

Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., and Kim, S. (2020). Horizontal pod autoscaling in kubernetes for elastic container orchestration. *Sensors (Switzerland)*, 20(16):1–18.

Rzadca, K. and et al. (2020). Autopilot: Workload autoscaling at Google. In *EuroSys*, pages 1–16.

Thönes, J. (2015). Microservices. *IEEE Software*, 32(1):116–116.

Ye, T., Guangtao, X., Shiyou, Q., and Minglu, L. (2017). An auto-scaling framework for containerized elastic applications. In *BigCom*, pages 422–430.

Zhao, H., Lim, H., Hanif, M., and Lee, C. (2019). Predictive container auto-scaling for cloud-native applications. In *ICTC*, pages 1280–1282.

Zhou, X., Jin, Y., Zhang, H., Li, S., and Huang, X. (2016). A map of threats to validity of systematic literature reviews in software engineering. In *APSEC*, pages 153–160.