American Academic Scientific Research Journal for Engineering, Technology, and Sciences
ISSN (Print) 2313-4410, ISSN (Online) 2313-4402
https://asrjetsjournal.org/index.php/American_Scientific_Journal/index

Auto-Scaling Techniques for Container Workloads in Kubernetes Clusters

Megha Aggarwal*

Software Development Engineer, Amazon AWS,Seattle, WA, USA
Email:megagarw@amazon.com

Abstract

This paper presents a comparative study of automatic scaling mechanisms in Kubernetes clusters. The objective of this study is to conduct a comparative analysis of various techniques for automating the scaling of containerized applications in Kubernetes clusters. The methodological foundation of the research comprises a systematic review and analytical processing of current scientific publications in the field. The work examines the architectural principles, key configuration parameters, and built-in limitations of traditional tools, including the Horizontal Pod Autoscaler, Vertical Pod Autoscaler, and Cluster Autoscaler. Particular attention is devoted to advanced solutions designed to enhance the adaptability and predictability of scaling. These include event-driven scaling using KEDA, high-efficiency node management with Karpenter, and the implementation of predictive strategies based on machine learning models. The scientific novelty of the study lies in the description of a comparative classification model of autoscaling techniques, which enables the formulation of clear recommendations for selecting the optimal strategy based on the type of workload: microservice web applications, big data processing pipelines, or resource-intensive machine learning tasks. The analysis suggests that to achieve high performance and resilience, it is advisable to combine various approaches — including horizontal, vertical, and cluster scaling — supplemented by heuristic or predictive methods. The findings will be valuable to DevOps engineers, cloud system architects, and researchers focused on optimizing operational performance and resource management in modern distributed environments.

Keywords: Kubernetes; autoscaling; containerization; HPA; VPA; Cluster Autoscaler; KEDA; Karpenter; predictive scaling; resource management.

Received: 7/16/2025
Accepted: 9/16/2025
Published: 9/26/2025

---

* *Corresponding author.*

I. Introduction

In the modern IT ecosystem, containerization serves as a key architectural approach for deploying and managing applications. The Kubernetes platform has firmly established itself as the industry standard for container orchestration, providing an extensive toolkit for automating deployment, scaling, and maintenance processes of distributed systems [1].

The relevance of optimizing computational resource utilization is continually increasing. The variable nature of workloads, especially in microservice architectures where traffic volumes can change by orders of magnitude over very short intervals, necessitates the implementation of intelligent and adaptive scaling algorithms. Automatic scaling is regarded as one of the core functions: it not only contributes to cost reduction by releasing idle capacity but also enhances system reliability by promptly provisioning resources to handle peak loads, thereby ensuring high service availability.

Nevertheless, a methodological gap is observed in the scientific literature: there is no systematized and comprehensive comparison of modern auto-scaling approaches that extend beyond the standard Kubernetes mechanisms. Most studies are limited to the analysis of a single tool or model, whereas a thorough evaluation of native, event-driven, and predictive methods remains unaccomplished.

The objective of this research is to conduct a comparative analysis of various automatic scaling techniques for containerized applications in Kubernetes clusters.

The **scientific novelty** of the study lies in the description of a comparative classification model of auto-scaling techniques, which enables the formulation of clear recommendations for selecting the optimal strategy depending on the workload type: microservice web applications, big data processing pipelines, or resource-intensive machine learning tasks.

The **research hypothesis** posits that, under the diversity of use cases — ranging from web services to computational tasks and machine learning models — maximal efficiency and reliability can be achieved not by applying a single universal mechanism, but through the combined use of multiple scaling methods, each optimized for the specific requirements of the particular workload.

At the same time, the study takes the form of a theoretical review and does not include an empirical assessment of the performance of the systems under consideration in a controlled environment, which defines its principal limitations.

II. Materials and Methods

As a starting point, many authors rely on the results of annual reports and survey studies, which provide an overview of the general state of cloud-native technologies. Thus, the CNCF Annual Survey 2023 report describes current trends in Kubernetes adoption and the primary challenges faced by operators in scaling [1]. This report is important because it quantitatively confirms the relevance of the problem, shifting it from a purely technical

task to a strategic priority for business. This underscores the need to systematize and compare approaches, as inefficient scaling directly leads to increased operating expenditures and decreased service reliability.

The next strand of work is devoted to the systematization of knowledge in this domain. Dogani J., Namvar R., Khunjush F. [7] propose a detailed taxonomy of autoscaling techniques in container and edge/fog environments, classifying methods as reactive, predictive, and hybrid, and considering their application in various deployment topologies. Senjab K., and his colleagues [9] review task scheduling algorithms in Kubernetes, emphasizing that the effectiveness of autoscaling largely depends on interaction with the resource scheduler system. Rabiu S., Yong C. H., Mohamad S. M.S. [10] analyze load balancing and autoscaling challenges in microservice architectures, focusing on the integration of load controllers (e.g., NGINX, Istio) with HPA mechanisms to ensure service stability and fault tolerance. These systematic works form the necessary theoretical basis, allowing us to consider auto-scaling not as an isolated function, but as part of a complex resource management system. Their value lies in showing how the effectiveness of scaling depends on the coordinated work of the planner, balancer, and controller itself.

Classic approaches to HPA are primarily oriented toward reactive scaling based on CPU and/or memory consumption. Nguyen T. T., and his colleagues [2] describe the standard mechanism for dynamically adjusting the number of pods using metrics from the Utilization Metrics API and demonstrate its effectiveness for moderately loaded web services. Augustyn D. R., Wyciślik Ł., Sojka M. [3] investigate the possibilities of adaptively tuning HPA parameters (target values, check intervals) using Bayesian optimization methods, thereby simultaneously addressing latency and throughput requirements in cloud deployments. Rolík O., Volkov V. [4] propose their horizontal scaling algorithm, based on a feedback controller and peak-smoothing mechanisms (low–pass filtering), to prevent overregulation of replica counts during sudden load changes. These studies demonstrate attempts to improve the basic reactive approach without relinquishing its simplicity. Work [3] employing Bayesian optimization is of particular interest, as it transforms HPA tuning from a manual process into an automated loop; however, the fundamental problem of delayed response to load in these approaches remains.

For scenarios with highly spiky or seasonal workloads, reactive autoscaling is insufficient; predictive models are employed instead. Dang-Quang N. M. and Yoo M. [6] apply bidirectional LSTM neural networks to forecast future loads and proactively adjust pod counts, demonstrating a reduction in failures during peak loads. Mondal S. K. and his colleagues [8] combine mathematical modeling methods (ARIMA, exponential smoothing) with a configurable autoscaling scheme, allowing flexible threshold adjustments to the characteristics of each workload and ensuring more accurate adherence to specified SLOs. These studies mark a transition from a reactive to a proactive paradigm, which constitutes a qualitative leap in ensuring service reliability. The use of complex models such as LSTM [6] makes it possible to capture nonlinear dependencies in load data, while simultaneously increasing the requirements for the quality of historical data and the computational resources needed to train the model itself.

A line of research has emerged focusing on the peculiarities of autoscaling AI and ML workloads, as well as the portability of solutions across different cloud providers. Emma L. [5] examines multi-cloud AI strategies,

describing containerization patterns for models and orchestration tools that enable dynamic redistribution of workloads among AWS, Azure, and Google Cloud to optimize cost and performance. Nuthalapati A. [11] focuses on scaling AI applications in the cloud with consideration for model efficiency (model pruning, quantization) and workload distribution, proposing an architecture in which HPA integrates with monitoring systems for ML-container-specific metrics (GPU utilization, inference latency) to achieve more precise resource management. This direction emphasizes that for modern AI/ML tasks, simple scaling by CPU/RAM is no longer sufficient, and the focus is shifting to specialized metrics [11]. In addition, work [5] raises an important architectural question of solution portability, since coupling to hardware accelerators of a specific provider can become a serious obstacle in multi-cloud environments.

The literature exhibits a contradiction between the simplicity of reactive HPA mechanisms (low implementation complexity) and the high potential of predictive approaches (more precise adaptation to peak loads). On one hand, Bayesian optimization methods and LSTM forecasts improve scaling quality but require data and complex configuration. On the other hand, classic HPA remains popular due to its integration into Kubernetes and ease of use; however, it often fails to cope with dynamically changing conditions. Moreover, despite the emergence of multi-cloud and AI-oriented studies, there remains a shortage of works dedicated to GPU workload autoscaling and the integration of HPA with the Cluster Autoscaler and Vertical Pod Autoscaler. Energy- and cost-aware scaling, as well as the interaction of scheduling algorithms with autoscaling in heterogeneous clusters, are also underexplored. These directions represent open challenges for further research.

III.Results and Discussion

In Kubernetes, the autoscaling mechanisms are structured according to a three-tier model, allowing for both the dynamic adjustment of the number of pods and the adaptation of the cluster configuration itself to actual operating conditions. At the first level is the Horizontal Pod Autoscaler (HPA) — a controller implementing a control loop: at each step it queries the Metrics Server, collecting up-to-date metrics (CPU load, RAM usage or any custom metrics integrated via Prometheus adapters and similar systems), after which it compares them with predefined target values. The HPA configuration specifies threshold levels and scaling policies that determine the rate and steps of pod count changes, smoothing abrupt fluctuations, and preventing spikes in load. This approach is ideally suited to stateless applications with a predictable correlation between traffic and resource consumption; however, the reactive nature of HPA means that scaling is triggered only after a threshold is exceeded, which can lead to brief performance degradation during a sudden increase in load [2]. A schematic representation of the HPA control loop is shown in Figure 1.

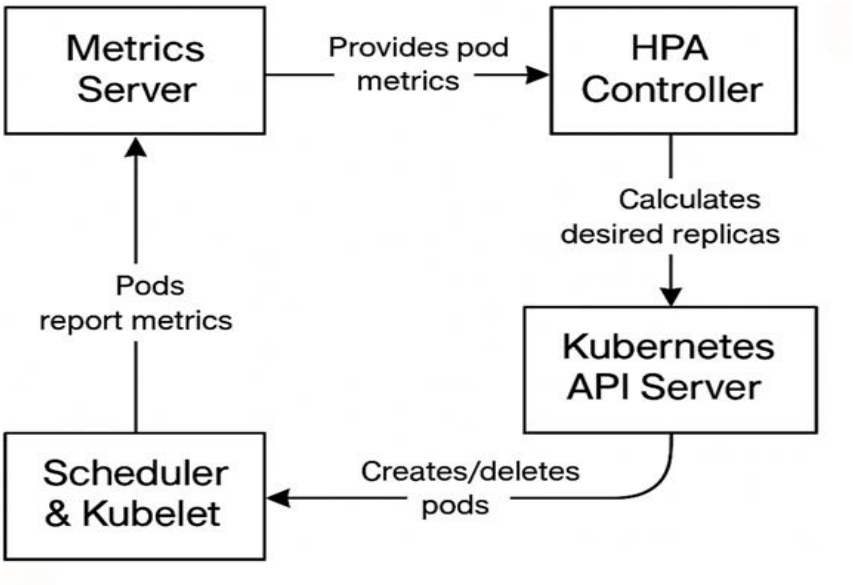

Figure1: Architectural scheme of the Horizontal Pod Autoscaler (HPA) [2, 11]

The second component of the system, Vertical Pod Autoscaler (VPA), does not aim to change the number of pod instances as HPA does, but is designed for dynamic adjustment of CPU and memory requests and limits of already deployed containers. The collection and analysis of historical metrics enables VPA to generate well-founded recommendations for resource values. It supports three modes of operation:
• Off: only generates recommendations without making changes to the configuration;
• Initial: applies optimal resource parameters only when a new pod is created;
• Auto: automatically adjusts requests and limits in real time, which requires restarting the corresponding pod and may lead to brief service interruptions.

The main advantage of VPA lies in precisely fitting resources to the actual needs of applications with variable load, thereby increasing pod density on nodes. At the same time, the Auto mode carries the risk of brief downtime during container restart, which is unacceptable for mission-critical services [3].

At a higher level, Cluster Autoscaler (CA) – the component responsible for scaling the entire cluster by adding or removing worker nodes (virtual or physical machines)- operates. When pods in Pending state appear in the cluster due to insufficient resources (CPU, RAM or GPU), CA uses cloud provider APIs (AWS, GCP, Azure, etc.) to initiate the creation of additional nodes, and under low load removes idle machines if all their pods can be safely redistributed onto the remaining nodes. Effective operation of CA requires coordinated interaction with HPA and VPA, as well as a thoughtful load-balancing strategy across availability zones. A summary comparison of these mechanisms is presented in Table 1.

Table 1: Comparative analysis of embedded Kubernetes autoscaling mechanisms [2, 3, 5]

Characteristic	Horizontal Pod Autoscaler (HPA)	Vertical Pod Autoscaler (VPA)	Cluster Autoscaler (CA)
Object of scaling	Number of pods (replicas)	Pod resources (CPU/RAM requests & limits)	Number of cluster nodes
Trigger	Exceeding target metrics (CPU, RAM, custom)	Analysis of historical resource consumption	Insufficient resources for pods (Pending state) or low node utilization
Main scenario	Stateless applications with variable load	Stateful applications, applications with unpredictable resource consumption	Any scenarios where the load may exceed the capacity of the current cluster
Advantages	Ease of configuration, broad applicability, and industry standard	Resource utilization optimization, increased placement density	Infrastructure management automation, cost savings
Disadvantages	Reactive nature, response latency, inefficient for event-driven workloads	Requires pod restarts (in Auto mode), which affects availability	Latency in provisioning new nodes (minutes), dependency on cloud provider

Standard scaling mechanisms, despite their functionality, remain predominantly reactive and operate only on typical metrics, which has stimulated the development of more intelligent solutions. One such solution is the KEDA project (Kubernetes Event-Driven Autoscaling), incubated by CNCF. Unlike HPA, which reacts only to CPU and memory parameters, KEDA is capable of initiating changes in the number of pods based on events arriving from a wide range of sources, including queue length in Kafka or RabbitMQ, message count in Azure Service Bus, custom metrics from Prometheus, and many other systems. Instead of periodically polling node load metrics, it triggers scaling (up to a complete zeroing of replicas) in response to external triggers [4].

The architecture of KEDA is built around a set of scalers — adapters that extract metrics from various sources and forward them to a ScaledObject. It is the ScaledObject that manages the target application deployment, adjusting the number of replicas according to incoming events (figure 2). This approach is particularly effective for asynchronous workloads and data-streaming scenarios, where event intensity does not necessarily correlate with CPU or memory consumption.

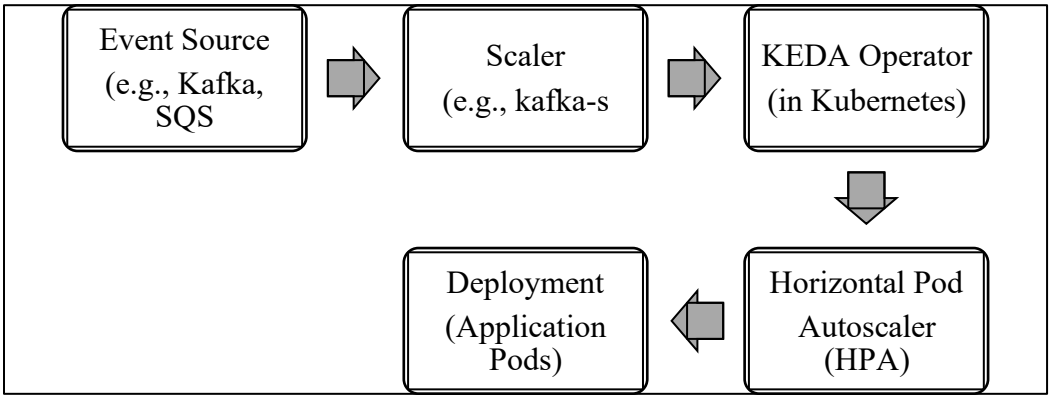

Figure2: Event-oriented scaling architecture with KEDA [4]

One of the most notable innovations in the Kubernetes ecosystem is Karpenter — AWS's response to the classic Cluster Autoscaler. Unlike the latter, which operates on autoscaling groups, Karpenter initiates the provisioning of nodes directly through the cloud API, bypassing the intermediate abstraction layer. This direct resource allocation mechanism reduces the interval from a pod entering the Pending state to the actual application launch to mere tens of seconds. In contrast, the traditional CA often requires several minutes [5]. An additional advantage is the more flexible instance selection, including the dynamic utilization of spot instances, which offers considerable scope for optimizing operational costs.

The next evolutionary step has been predictive scaling based on machine learning. Instead of a reactive approach, where automation catches up with increased load, algorithms analyze historical metrics (for example, stored in Prometheus) and generate forecasts using time-series models such as ARIMA as well as recurrent LSTM architectures [6, 7]. The forecasted value is then supplied to the HPA as a custom metric, enabling preemptive scaling up or down of pod counts before the onset of peak windows. This approach is particularly beneficial for workloads with pronounced diurnal or weekly patterns — a classic example being marketplaces during sale periods or streaming services during major broadcasts.

A critical element of implementation is the selection of metrics and policies. Relying solely on CPU utilization and memory consumption is convenient, but it does not always accurately reflect the actual bottleneck. In I/O-sensitive applications or when interacting with external services, business metrics become critical: RPS, median response latency, and the number of active sessions. Thanks to the Prometheus ecosystem and HPA adapters, such multidimensional scaling schemes can be implemented with minimal effort [10]. Practice shows that the best results are achieved by combining static guarantees (a minimum number of replicas to maintain fault tolerance) with dynamic rules that use multiple metrics simultaneously.

Additional challenges arise when working with stateful components and tasks requiring hardware accelerators. Horizontal scaling of databases or message queues involves creating new pods and attaching persistent volumes, as well as data replication and synchronization [8]. For ML workloads utilizing GPUs or TPUs, not only are the available accelerators important, but also the network topology between them is important. Kubernetes tools — including taints/tolerations, node affinity, and anti-affinity — enable the isolation of GPU pools and control of

pod placement, thereby optimizing throughput and latency [9, 11].

The orchestration process and ensuring fault tolerance are no less significant. The effect of instantaneous pod scaling is nullified if the layer responsible for cluster expansion (whether CA or Karpenter) fails to provision nodes promptly. During scale-down, the Pod Disruption Budget ensures that the minimum required number of instances remains operational. For stateful services, a graceful shutdown with state preservation is also required.

Ultimately, the choice of an autoscaling strategy becomes a multifaceted engineering task that requires a synthetic view of application architecture, traffic patterns, and business economic requirements. A judicious combination of tools and methods — from Karpenter's direct node provisioning to machine learning-based load forecasts — enables the building of a system that reliably withstands peaks while remaining cost-effective in everyday operation.

Thus, one can observe a clear paradigm shift: from the use of isolated, reactive tools to the construction of comprehensive, multi-layered autoscaling systems. The modern approach consists not in selecting a single best tool but in their judicious composition. For example, HPA can control basic horizontal scaling by RPS, KEDA handles asynchronous event-driven workloads within the same system, while Karpenter at the cluster level ensures the fastest and most cost-effective provisioning of nodes for both types of tasks. This shift from configuring individual components to designing a holistic adaptive system changes the role of the engineer, requiring not only knowledge of specific technologies but also a deep understanding of application architecture, traffic models, and the economic realities of the cloud platform. Ultimately, the maturity of an autoscaling system is determined not so much by the sophistication of an individual component as by the harmony of their joint operation to achieve specific business goals such as adherence to SLO and cost optimization.

IV.Conclusion

The analysis of existing automatic scaling mechanisms within the Kubernetes ecosystem confirms that no universal strategy applicable to all categories of workloads currently exists. A rational selection of a specific method or their synergistic combination must be based on a scrupulous analysis of both application characteristics and infrastructure operational constraints.

Classic built-in Kubernetes components (HPA, VPA, and Cluster Autoscaler) form a reliable foundation; however, their practical value declines as the environment becomes more complex and dynamic. HPA remains the de facto standard for stateless services, with a well-predictable request profile, due to its response to pod-level metrics (CPU, memory, RPS) exhibiting acceptable latency. VPA, by contrast, serves as a fine-grained calibration tool for resources. By adjusting requests and limits, it increases utilization efficiency but simultaneously elevates the risk of brief downtimes during container redistribution. Cluster Autoscaler is essential in public clouds, where cost savings are achieved through elastic management of node groups; nevertheless, the inertia of scaling operations (minutes until actual VM provisioning) can become a bottleneck for bursty workloads.

Contemporary next-gen solutions address precisely those scenarios in which basic tools demonstrate limitations.

KEDA integrates optimally into event-driven topologies and asynchronous data-processing pipelines, providing the ability to scale to zero and thereby minimize idle expenditure. Karpenter, by leveraging direct interaction with the cloud API, significantly outperforms the traditional Cluster Autoscaler in node provisioning latency and instance selection flexibility, rendering it the preferred choice for environments characterized by sudden and hard-to-predict peaks as well as for cost-optimization tasks. Predictive scaling algorithms trained on historical telemetry series currently represent the cutting edge of the technology: they are particularly effective for services with pronounced diurnal or weekly seasonality (e-commerce platforms, streaming media), allowing compute capacity to be reserved in advance and thus preventing degradation of QoS during peak load periods.

Based on the above, the following practical recommendations can be formulated:
• Standard web services and APIs: a combination of HPA targeting RPS and p99 latency with Karpenter is optimal as it ensures both rapid response to high-level metrics and swift addition/removal of nodes
• Asynchronous tasks and data-processing pipelines: KEDA is recommended, as it provides native triggers for queues, brokers, and streaming services and can reduce the cluster to zero between waves of tasks
• ML workloads with GPU dependency: application of Karpenter with correctly configured taints and tolerations is advisable, augmented by a predictive autoscaling model (for example, based on time-series forecasting) if request patterns exhibit repeatable behavior

Probable vectors for the further evolution of automatic scaling lie, first, in the deeper integration of predictive approaches directly into the Kubernetes core and its control plane, and second, in the development of unified mechanisms for hybrid and edge-computing scenarios, where resource management is complicated by variable network connectivity and the heterogeneity of hardware profiles. Research and engineering efforts of the community will focus on these domains in the coming years.

References

[1]. Cloud Native Computing Foundation. (2024). CNCF annual survey 2023: The state of cloud native development, from: https://www.cncf.io/reports/cncf-annual-survey-2023/ (date accessed: 17.05.2025).

[2]. Nguyen, T. T., et al. (2020). Horizontal pod autoscaling in Kubernetes for elastic container orchestration. *Sensors, 20*(16), 4621. https://doi.org/10.3390/s20164621.

[3]. Augustyn, D. R., Wyciślik, Ł., & Sojka, M. (2024). Tuning a Kubernetes horizontal pod autoscaler for meeting performance and load demands in cloud deployments. *Applied Sciences, 14*(2), Article 646. https://doi.org/10.3390/app14020646.

[4]. Rolík, O., & Volkov, V. (2024). Method of horizontal pod scaling in Kubernetes to omit overregulation. *Information, Computing and Intelligent Systems,* (5), 55–67.

[5]. Emma, L. (2025). *Multi-cloud AI strategies: Deploying portable machine learning solutions across AWS, Azure, and Google Cloud*, 1-12.

[6]. Dang-Quang, N. M., & Yoo, M. (2021). Deep learning–based autoscaling using bidirectional long short-term memory for Kubernetes. *Applied Sciences, 11*(9), Article 3835. https://doi.org/10.3390/app11093835

[7]. Dogani, J., Namvar, R., & Khunjush, F. (2023). Auto-scaling techniques in container-based cloud and

edge/fog computing: Taxonomy and survey. *Computer Communications, 209*, 120–150. https://doi.org/10.1016/j.comcom.2023.06.010

[8]. Mondal, S. K., et al. (2023). Toward optimal load prediction and customizable autoscaling scheme for Kubernetes. *Mathematics, 11*(12), Article 2675. https://doi.org/10.3390/math11122675

[9]. Senjab, K., et al. (2023). A survey of Kubernetes scheduling algorithms. *Journal of Cloud Computing, 12 (*1).

[10]. Rabiu, S., Yong, C. H., & Mohamad, S. M. S. (2022). A cloud-based container microservices: A review on load-balancing and auto-scaling issues. *International Journal of Data Science, 3*(2), 80–92. https://doi.org/10.18517/ijods.3.2.80-92.2022

[11]. Nuthalapati, A. (2025). Scaling AI applications on the cloud toward optimized cloud-native architectures, model efficiency, and workload distribution. *International Journal of Latest Technology in Engineering, Management & Applied Science, 14*(2), 200–206. https://doi.org/10.51583/ijltemas.2025.14020022