ISSN: 2754-6705

Journal of Mathematical & Computer Applications

SCIENTIFIC Research and Community

Review Article

Open Access

Kubernetes Advanced Auto Scaling Techniques

Ramasankar Molleti

Independent Researcher, USA

ABSTRACT

This paper discusses some sophisticated autoscaling strategies in Kubernetes, and they include horizontal and vertical pod autoscaling, cluster levels autoscaling, and metrics based autoscaling. In this regard, it covers the topics of both predictive and event-triggered auto scaling approaches and their applications, the advantages and the potential issues relating to them, and more. Some of the questions answered by the study include how to improve application throughput and resource efficiency and how to achieve cost-efficient K8s clusters for practitioners while providing directions for the research on CA autoscaling in container orchestration.

***Corresponding author**

Ramasankar Molleti, Independent Researcher, USA.

Received: December 01, 2022; **Accepted:** December 13, 2022, **Published:** December 19, 2022

Keywords: Autoscaling, Metrics, Pod, Node, API, Optimization, HPA, VPA, Prometheus

Introduction

Overview of Kubernetes

Kubernetes is an open source container orchestration solution that tries to fully automate practically every aspect about the containers that consist of an application. From the above conclusions the following can be argued-It provides a good basis for guaranteeing availability of applications, and their modularity in addition to their resource utilization. Kubernetes supports something that is called a 'platform,' which abstracts the essence of layers and, at the same time, handles a significant number of chores for the developer [1]. Kubernetes provides a declarative and an extremely versatile configuration model to manage the adaptive structure and processes of today's cloud-native environments and it is a standard choice when it comes to choosing an orchestration manager for a container.

Importance of Autoscaling in Container Orchestration

Autoscaling is a mandatory attribute of container orchestration since applications can only increase the consumed resources when workload increases. This capability helps in best resource utilisation and also can balance the cost based on the loads that are likely to be available or generated. Autoscaling enables an application to request additional resources during high traffic and to release these during low traffic, and decreases the need for a human to constantly monitor an application's resource consumption. Among the key functions used in the context of Kubernetes, it should be noted that autoscaling mechanisms are critical for fulfilling the main rock-solid mission and ensuring the effective overall self-healing mechanism of the platform.

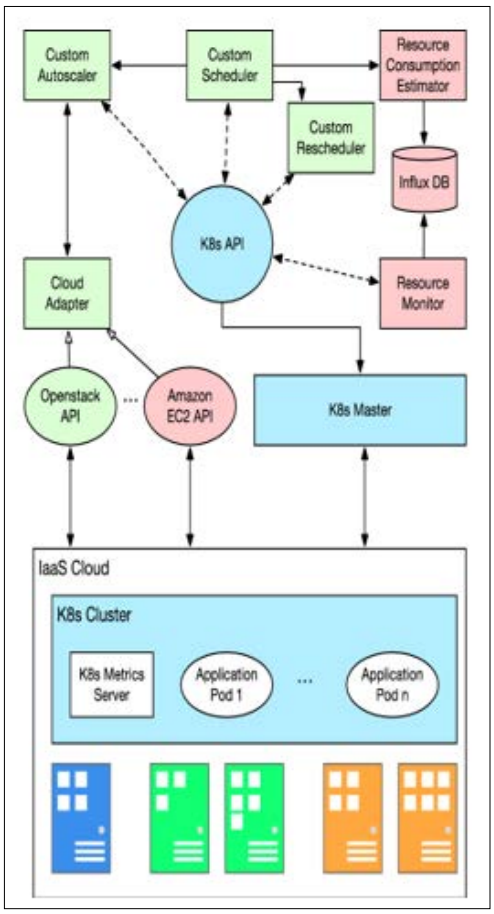

Figure1: Containers Orchestration (Source: https://d3i71xaburhd42.cloudfront.net)

Basic Kubernetes Autoscaling Concepts

Kubernetes offers several native autoscaling opportunities which can be used to address various kinds of scaling issues. The Horizontal Pod Autoscaler (HPA), is used for the scaling up as well as the scaling down of pods depending on CPU usage or metrics. The Vertical Pod Autoscaler – VPA influences higher and lower request and limit of some pods in regard to the identified resources. The node pool size task has been handled manually by the Cluster Autoscaler in the current times. All of these autoscalers are active simultaneously to allow the applications to acquire the required resources and to control the full use rate in the cluster [2]. Here are concepts that get to the fundamentals of Kubernetes and are probably vital to anyone who wishes to autoscale in Kubernetes landscapes.

Purpose and Scope of the Paper

This paper intends to elaborate more on autoscaling in Kubernetes and not just the basic theory and recognizing the different aspects of autoscaling. It will explain the specifics of horizontal and vertical pod autoscaling, cluster-level autoscaling, and custom metrics. The paper will also look at new trends in auto scaling including; Predictive and event-based autoscaling. Thus, the topical coverage of the paper aims to help practitioners understand and improve the performance and resource efficiency, as well as the cost-effectiveness of their applications and services deployed in Kubernetes environments.

Horizontal Pod Autoscaler (HPA)

Core Concepts and Functionality

HPA or Horizontal Pod Autoscaler is used to scale pod replicas in a deployment or replication controller based on CPU usage or metric specified. It from time to time compares these metrics to a target value and computes the desired number of replicas to sustain the target [3]. This process is helpful to make sure the resource of the application is well utilised and the control available for the application is good for a number of workloads.

Configuration

Configuring HPA is done by setting the desired CPU utilisation or any other metrics, setting minimum and maximum replica count, and lastly the scaling policy. Usually, implementation involves the creation of an HPA resource by using 'kubectl' or YAML manifests. This enables the HPA controller to then watch out for the said metrics and make necessary changes to the replica count that match the desired scaling behaviour of the application.

Table 1: Comparison of HPA Metrics and Their Use Cases

Metric Type	Description	Use Case	Advantages	Limitations
CPU Utilisation	Percentage of requested CPU	General-purpose applications	Easy to configure, built-in	May not reflect application performance
Memory Usage	Percentage of requested memory	Memory-intensive applications	Prevents OOM errors	Can lead to over-provisioning
Custom Metrics	Application-specific indicators	Specialised workloads	Highly tailored scaling	Requires additional setup
External Metrics	Metrics from external sources	Cloud services, queues	Scales based on external factors	Dependency on external systems

Limitations and Challenges

Despite the strength of HPA, there are some known issues; for instance, metric changes lead to intensive scaling, latency-related issues by pod startup times, and multi-dimensional scaling requirements. However, CPU metrics tend to be unsuitable for specific applications, when solely used by themselves [4]. They are the set of features and restrictions that can only be mitigated by fine-tuning the HPA and possibly integrating it with other forms of auto-scaling.

Vertical Pod Autoscaler (VPA)

Operating Principles

The Vertical Pod Autoscaler (VPA) scales the containers in pods and changes the CPU and memory resource requests. Based on the historical data of resource consumption and current requests, it may suggest or automatically assign the most suitable resource utilisation [5]. The concept of VPA is to increase the efficiency of the use of cluster resources and the speed of the application by adjusting the amount of resources available to pods.

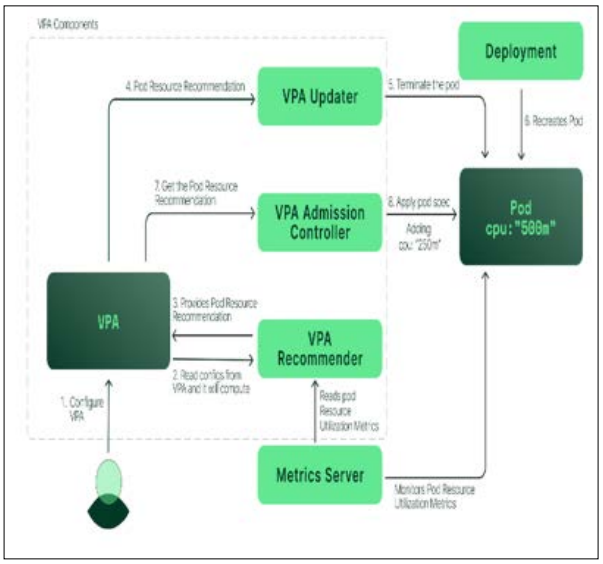

Figure 2: VPA (Source: https://www.kubecost.com)

Uses and Benefits

VPA is most beneficial for workflows with non-static resource requirements or for those that must be set up manually. This aspect assists in avoiding the cases of under-utilising or over-allocation of resources, thus enhancing cost efficiency of the clusters. VPA is advantageous in the lack of interaction of big applications with a few users in response time and resource allocation and appeals to applications that are not frequently required to process an excessive amount of data all at once while their demands rise incrementally.

Integration with HPA

VPA can cooperate with the Horizontal Pod Autoscaler or HPA which gives encompassing scaling options. While HPA controls the total amount of pod replicas, VPA controls the distribution of pods' resources [6]. Integration achieved at this level offers a precise degree of control over the extent to which applications are scaled both, laterally and vertically, while at the same time optimising the use of resources in different scenarios, thus enhancing application performance.

Cluster Autoscaler

Architecture and Components

The Cluster Autoscaler is an additional element of the Kubernetes which is aimed at a dynamic process of the addition or removal of nodes in the cluster. It is made up of a main loop, which runs every interval, and looks for pods that are unable to be scheduled because of resource availability [7]. The autoscaler is communicating with the API of the chosen cloud provider to operate node groups, scale the cluster.

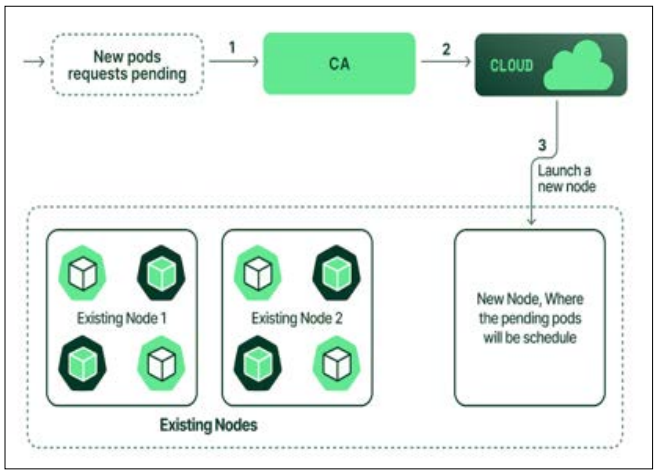

Figure 3: Kubernetes Cluster Autoscaler (Source: https://www.kubecost.com)

Scaling Policies and Strategies

The Cluster Autoscaler uses the following techniques in managing the extent of the cluster. It can increase when pods are unschedulable because of the resource limitations and decrease when the nodes have low CPU usage. It also aims at providing scale recommendations based on various things such as resource quantity and quality requested by pods, nodes' resource usage, priority level of various pods and the like. This also adheres to user-provided constraints like minimum and maximum number of nodes to create.

Cloud Provider-Specific Considerations

Several cloud providers have special characteristics and restrictions, which influence the behaviour of Cluster Autoscaler. These are instance types, and the pricing of such instances, and the Node worthy APIs for Node Group management [8]. There might be a necessity to tune autoscaler settings to take advantage of such advanced features of cloud providers as ASGs or MIGs, respectively, to achieve the best results in terms of cost and efficiency in AWS or GCP.

Custom Metrics Autoscaling

Prometheus Adapter

Prometheus Adapter converts Prometheus metrics to the infrastructure of Kubernetes auto-scaling. It converts Prometheus queries into a form digestible by the Custom Metrics API so that HPA can leverage application level metrics for scale calculations. This adapter allows autoscaling for various parameters, from request rates to business ones, improving autoscaling policies' flexibility and accuracy.

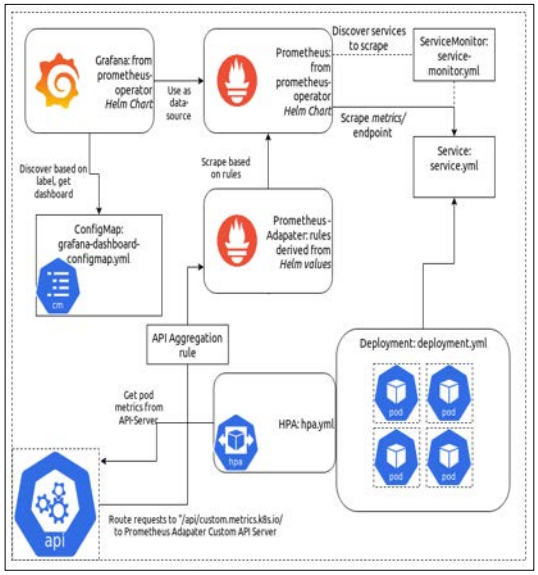

Figure 4: Prometheus Adapter (Source: https://d2908q01vomqb2.cloudfront.net)

Custom Metrics API

The Custom Metrics API is new from Kubernetes and broadens the autoscaling options to not just CPU and memory. It offers a common way of exposing application-specific metrics to the HPA [9]. This API is used for multiple metric sources that can be included into the autoscaling process and therefore allows to scale on more than server load, queue length, latency or even business metrics which make the autoscaling more application-aware.

Advanced Strategies for Custom Metrics-Based Autoscaling

The process of using advanced methods accompanied by custom metrics to scale an application. This is made up of metrics that involve the use of more than one counter; metrics that adopt the use of rate of change to scale; and the use of percentile metrics to establish the true capacity needed. Some of these strategies may need fine-tuning, and the use of custom rules to analyse application behaviours to improve on the auto scaling efficiency.

Predictive Auto Scaling Techniques

Machine Learning Approaches

Predictive auto scaling as used in the technique of machine learning then gathers information on the resource consumption patterns so that the requirement in the future can be estimated. This is made with the support of such activities as regression analysis, time series forecasting and use of neural networks to

predict workload tendencies [10]. These models analyse the past behaviour of resource usage and other applications, the actual parameter of utilisation and other external factors in order to predict the need for scaling in the future and, therefore, minimise reaction time to such scales.

Table 2: Comparison of Predictive Auto Scaling Models

Model Type	Technique	Strengths	Weaknesses	Complexity
Linear Regression	Time series forecasting	Simple, interpretable	Limited to linear relationships	Low
ARIMA	Time series analysis	Handles trends and seasonality	Assumes stationarity	Medium
Neural Networks	Deep learning	Can capture complex patterns	Requires large datasets, black box	High
Random Forest	Ensemble learning	Robust to outliers, handles non-linearity	Can overfit, computationally intensive	Medium-High
Prophet	Additive model	Handles holidays, missing data	May oversimplify complex patterns	Medium

Time-series Analysis Models

Different kinds of time series models are also quite handy for predictive auto scaling in the Kubernetes system. Methods such as ARIMA, exponential smoothing, and Prophet assist in identification of trends, seasonality, as well as cyclicality of the resources. Such models can be used in detecting temporal dependencies within the workload behaviour and thus, based on the resources required that will influence autoscaling, the short run and long run behaviour can be predicted.

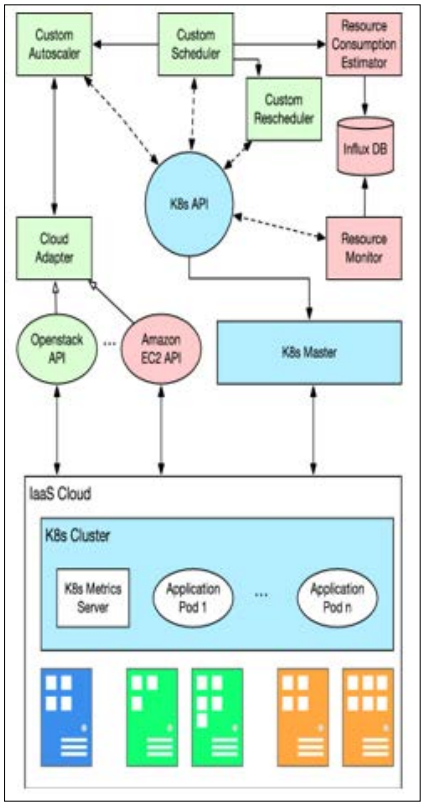

Figure 5: Time Series-Based Approach to Elastic Kubernetes (Source: https://www.mdpi.com)

Challenges in Predictive Auto Scaling

There are some problems with which predictive auto scaling becomes difficult and these are some issues related to autoscaling. The ability to manage accuracy over the models in terms of handling different workloads and even a shift in their usage is something which proves to be quite complex. The issues concerning the trade off of different parameters like the accuracy at the different prediction levels and the time taken in the computations should also be well handled [11]. However, reversing possible anomalies, the possibility of feedback into the solution, and the way of integrating the predictions with other solutions for Kubernetes autoscaling are the issues of constant further discussion in this sphere.

Multi-dimensional Autoscaling

Combining HPA and VPA

The utilisation of pods along with CPU is called multi-dimensional autoscaling, and working with both HPA and VPA concurrently. This allows the users to scale up the number of pods and scale up or scale down the resource that is needed in a pod [12]. Particular emphasis should be placed on the creation of While one should not interfere with each other, HPA working in harmony with VPA should improve the application's execution and resource utilisation.

Multi-metric Auto Scaling Strategies

Regarding the multi-metric auto scaling strategies, this is the procedure to make decisions on scaling taking into account several metrics. This can be done by adopting the use of the measures of CPU, memory, and specific to the particular application that is desired, measures like request latency or queue length. These strategies can be viewed as the more general conception of the application, and therefore, such autoscaling processes may be more effective and precise.

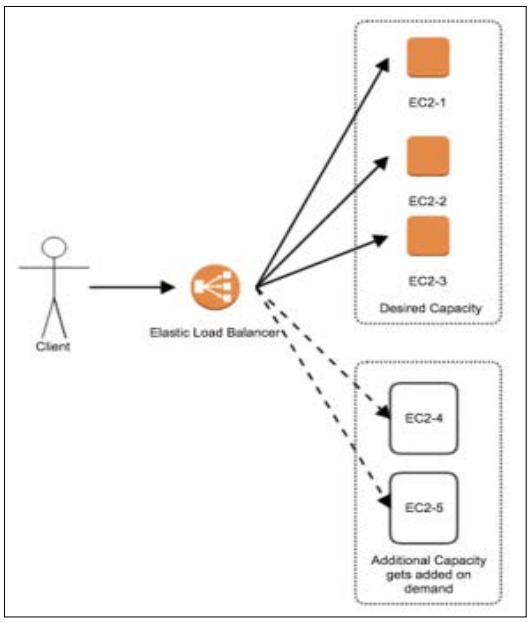

Figure 6: AWS Auto-Scaling (Source: https://k21academy.com)

Trade-offs and Optimizations

Multi-dimensional auto scaling means that a rather significant amount of trade-offs must be worked through in order to achieve the desired outcomes, if they are desirable at all. The above are; dealing with more than one scaling dimension, competition issues in scaling up, and improved operating expense [13]. Optimisations can include setting focus on some of these metrics at the expense of others, incorporating checks so that the system's scaling actions go up and down, and on tuning the thresholds to get the right level of reaction from the autoscaling system to achieve stability.

Event-driven Autoscaling

KEDA (Kubernetes Event-driven Autoscaling)

KEDA stands for Kubernetes Events Driven Autoscaler and is an autoscaler categorised as open-source. It is suitable for making micro-overscale with respect to the number of events that make the application alive. Currently KEDA supports the following event sources: There are message queues, databases, and own metric [14]. Thus, when there is no event, the scale is almost zero; and it is the same when talking about Kubeless and KEDA that assists in the optimal usage of the application resources and, therefore, a low impact on costs in event-driven jobs.

Table 3: KEDA Scalers and Their Applications

Scalar Type	Event Source	Use Case	Scaling Metric
Apache Kafka	Kafka topics	Stream processing	Message lag
RabbitMQ	RabbitMQ queues	Asynchronous job processing	Queue length
Prometheus	Custom metrics	Application-specific scaling	Query result
Azure Blob	Azure Storage	File processing	Blob count
AWS SQS	SQS queues	Serverless workflows	Queue length
Cron	Time-based events	Scheduled scaling	Schedule

Server Less Auto Scaling Patterns

The patterns correlated to server less auto scaling in Kubernetes originate from access to the application scaling concerning customers' requests or events. They are referred to as zero-to-zero and zero-to-many patterns, similar to Function-as-a-service. There are toolchains like Native which are used to deploy and run server less applications on Kubernetes; it self-scales for event-based deployment with a minimal amount of configuration. This approach enables one to easily reach high levels of scalability where the load is burst or unpredictable.

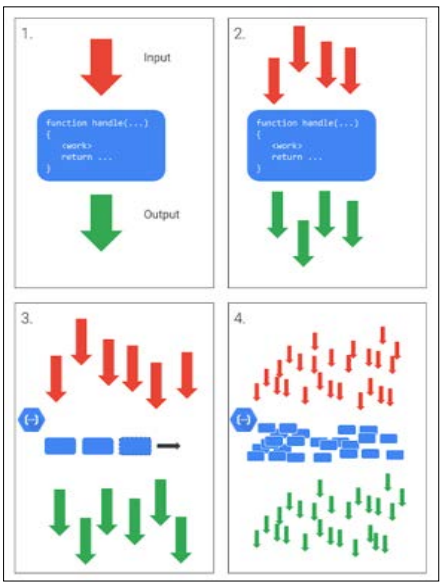

Figure 7: Scaling Server less Applications (Source: https://storage.googleapis.com)

aReal-world Applications

Specifically, the technique of auto scaling based on the events is invaluable for a plethora of real-life applications. It is most valuable in the micro services architecture, Iota data processing, and for the work that is performed on a batch basis. For instance, the payment processing system has to grow in tandem with the growing transaction queue, similarly images that are uploaded would depend on how many such uploads are there for an image processing service [15]. These applications can take full advantage of the outward characteristics of event-driven scaling; this is because such applications most of the time are characterized by unpredictable demands.

Future Directions and Conclusion

Emerging Auto scaling Technologies

Newer trends in auto scaling in Kubernetes relate to the improvement of precise predictions and self-decision processes. The approaches that are obtainable with the help of machine learning are growing more complex, including the real-time data analysis and machine learning algorithms that learn on the go. Integration of edge computing is also starting to happen as it allows for auto scaling of various units to be distributed [16]. Furthermore, while improving the container technology or other approaches to resource isolation provided a basis for the new scaling mechanisms, it is now possible to scale elements in the application down to being much finer-grained when it comes to truly isolating the things that need to scale separately.

Integration with Cloud-Native Ecosystems

Subsequent generation auto scaling options nonetheless likely to become much more tightly integrated with other aims and features of the cloud-native panorama. This is because the new version offers a better compatibility with service meshes and server less computing platforms; has integration with multiple clusters' management tools. Accuracy in regard to specific observability tools means that the context of scaling decisions will be less ambiguous [17]. But regarding the growth of the new cloud-native technologies, micro-service auto scaling mechanisms will transform, to fit new opportunities and coexist with different and more assorted environments.

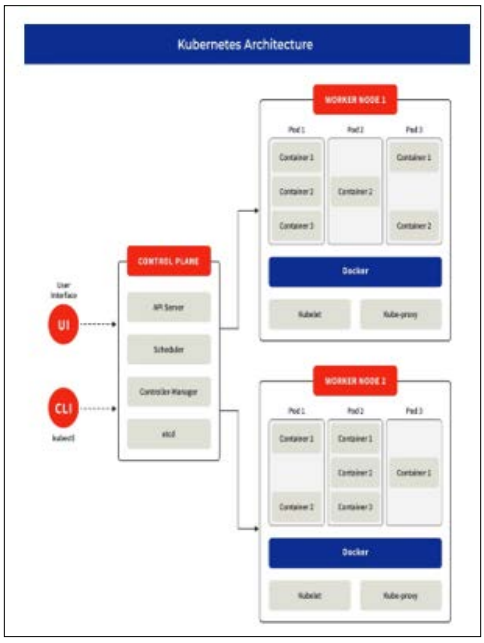

Figure 8: Cloud Native Development Services (Source: https://cdn.azilen.com)

Summary of Key Findings

This paper has therefore looked at some of the more complex Kubernetes auto-scaling mechanisms that include auto-scaling based on pods, consumption metrics, vertical auto scaling, and even anti-patterns. Several studies' major insights include MSD being critical to auto scaling and custom metrics gradually becoming ingrained to auto scaling processes. The ML and TS combination introduced the improvement of prediction, while auto scaling based on events introduced the new paradigms for resource adaptation.

Recommendations for Practitioners

It is recommended that practitioners pay cognizance to the fact that different workloads would require different types of auto scaling and thus, orchestrating more than one technique for auto scaling should be the approach taken by most practitioners. There is a lot of emphasis in making sure that auto scaling has adequate monitoring and observability systems that can be implemented for auto scaling. In fact, it is suggested to begin with the Kubernetes auto scaling and extend it with the use of the other sophisticated methods, such as the custom metrics and predictive auto scaling [18]. Another point that needs to be further discussed is frequency of the auto-scaling settings checks and their further tuning according to the application needs.

References

1. Taherizadeh S, Grobelnik M (2020) Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications. Advances in Engineering Software 140: 102734.
2. Zhao A, Huang Q, Huang Y, Zou L, Chen Z, et al. (2019) Research on resource prediction model based on kubernetes container auto-scaling technology. In IOP Conference Series: Materials Science and Engineering 569: 052092.
3. Altaf U, Jayaputera G, Li J, Marques D, Meggyesy D, et al. (2018) Auto-scaling a defence application across the cloud using docker and kubernetes. 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion) 327-334.
4. Rattihalli G, Govindaraju M, Lu H, Tiwari D (2019) Exploring potential for non-disruptive vertical auto scaling and resource estimation in kubernetes. 2019 IEEE 12th International Conference on Cloud Computing (CLOUD) 33-40.
5. Nguyen TT, Yeom YJ, Kim T, Park DH, Kim S (2020) Horizontal pod autoscaling in kubernetes for elastic container orchestration. Sensors 20: 4621.
6. Baresi L, Hu DYX, Quattrocchi G, Terracciano L (2021) KOSMOS: Vertical and horizontal resource autoscaling for kubernetes. International Conference on Service-Oriented Computing Cham: Springer International Publishing 821-829.
7. Thurgood B, Lennon RG (2019) Cloud computing with kubernetes cluster elastic scaling. Proceedings of the 3rd International Conference on Future Networks and Distributed Systems 1-7.
8. Perera HCS, De Silva TSD, Wasala WMDC, Rajapakshe RMPRL, Kodagoda N, et al. (2021) Database scaling on Kubernetes. 3rd International Conference on Advancements in Computing (ICAC) 258-263.
9. Podolskiy V, Jindal A, Gerndt M (2018) Iaas reactive autoscaling performance challenges. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD) 954-957.
10. Poniszewska-Marańda A, Czechowska E (2021) Kubernetes cluster for automating software production environment.

Sensors 21: 1910.

11. Ju L, Singh P, Toor S (2021) Proactive autoscaling for edge computing systems with kubernetes. Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion 1-8.
12. Turin G, Borgarelli A, Donetti S, Johnsen EB, Tapia Tarifa SL, et al. (2020) A formal model of the kubernetes container framework. International Symposium on Leveraging Applications of Formal Methods. Cham: Springer International Publishing 558-577.
13. Sayfan G (2018) Mastering Kubernetes: Master the art of container management by using the power of Kubernetes. Packt Publishing Ltd https://www.amazon.in/Mastering-Kubernetes-Master-container-management/dp/1788999789.
14. Dang-Quang NM, Yoo M (2021) Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes. Applied Sciences 11: 3835.
15. Delnat W, Truyen E, Rafique A, Van Landuyt D, Joosen W (2018) K8-scalar: a workbench to compare autoscalers for container-orchestrated database clusters. Proceedings of the 13th International Conference on software engineering for adaptive and self-managing systems 33-39.
16. Deshpande N (2021) Autoscaling Cloud-Native Applications using Custom Controller of Kubernetes. (Doctoral dissertation, Dublin, National College of Ireland) https://norma.ncirl.ie/5089/1/nehanarendradeshpande.pdf.
17. Xing S, Qian S, Cheng B, Cao J, Xue G, et al. (2019) A QoS-oriented Scheduling and Autoscaling Framework for Deep Learning. 2019 International Joint Conference on Neural Networks (IJCNN) 1-8.
18. Beni EH, Truyen E, Lagaisse B, Joosen W, Dieltjens J (2021) Reducing cold starts during elastic scaling of containers in kubernetes. Proceedings of the 36th Annual ACM Symposium on Applied Computing 60-68.