# Adjusting Deployed Container Number to Reduce Rolling Update Time

Muhammad Akram[1] Miho Tanaka[1] Takayuki Kushida[1]

**Abstract**：Kubernetes has become fundamental for orchestrating containerized applications, automating deployment, scaling, and management across clusters. A key strategy in Kubernetes is the rolling update, which allows updates without downtime by replacing each pod sequentially. The issue faced is the long time taken for rolling updates to be executed. The proposal is to increase or decrease the number of pods deployed simultaneously during a rolling update. The rolling update is automated after inputting the replicas, maxSurge, and maxUnavailable values, by implementing a Python script. The replicas value sets the number of containers, while the maxSurge and maxUnavailable values sets the number of pods deployed simultaneously. The Evaluation experiment indicated that a rolling update of 100 containers was executed, starting with deploying one pod at a time. The number of containers deployed simultaneously was incremented by 10 sequentially (10, 20, 30, 40, 50, 60, 70, 80, 90) up until 90 containers. The number of pods deployed simultaneously (50 pods) which is half of the number of containers(100 pods) demonstrates that the rolling update takes up the shortest time which is 89 seconds. The time taken for the rolling update where 50 pods were deployed simultaneously is 89 seconds compared to where 1 pod was deployed at a time is 133 seconds indicates a decrease of approximately 33% in time taken which is 44 seconds.

## 1. Introduction

### Background

Kubernetes manages applications for developers and businesses. This open-source platform, functioning as a conductor for containers, ensures smooth operation across local servers and the cloud. Management tasks such as scaling applications up or down based on demand, restarting failed components, and distributing workloads are handled by Kubernetes. This platform serves as a crucial tool for modern software development [1].

Updates enhance system security and effectiveness. These updates safeguard applications against security threats, resolve bugs, and add features that improve usability. Compatibility with newer technologies is maintained by these updates, assisting businesses in adapting to the evolving tech landscape. Updates ensure system security, reliability, and relevance.

[1] Tokyo University of Technology, Department of Computer Science
1404-1 Katakuracho, Hachioji City, Tokyo 192-0982

Pods serve as the core unit in Kubernetes management. A pod functions as the smallest unit managed by Kubernetes, often described as a small, self-contained environment where containers coexist. These containers, sharing resources such as storage and networking, make pods suitable for hosting and managing application components [2]. For instance, a pod operates a web server container alongside a helper container that processes logs. Pods enable Kubernetes to maintain application efficiency and readiness for scaling or recovery as necessary.

Rolling updates is a feature for application upgrades in Kubernetes. This strategy updates an application incrementally, replacing components sequentially to prevent downtime. Although this approach keeps the application operational, rolling updates requires time as each pod undergoes updating, health checking, and removal of the older version [3]. The objective is to expedite this process while ensuring stability and smooth operation.

### Main Issue

Rolling updates extends updates completion time. Con-

tainers deploy sequentially in this process, extending the time required to finalize the update fully.

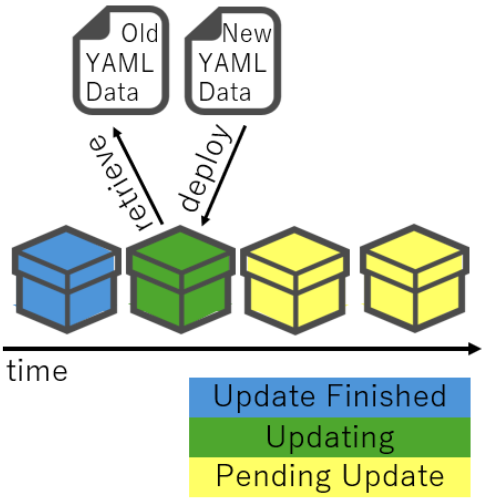


Figure 1: Rolling Update

Figure 1 illustrates the Kubernetes rolling update process. Containers update sequentially, maintaining continuous application availability and replacing the data in each container one by one. Initially, the blue container represents the updated version, followed by deployment of the green container, the updating version. The green container's old YAML data is retrieved and the data is placed into a new container, which will become the updated container with new YAML data. Once the green container is fully updated, the process repeats with the yellow containers, the versions pending update. This gradual replacement of older versions after validation ensures no downtime, highlighting a primary benefit of rolling updates in Kubernetes.

## 2. Related Research

Kubernetes gains prominence in managing large-scale, containerized applications. Research targets optimization of deployment and auto-scaling strategies to boost availability and minimize downtime [4]. The updates has zero downtime as shown in the study but will cause an elongation in the time taken for said updates. In comparison to this paper, the number of containers deployed simultaneously will decrease the time taken for the updates while ensuring that there is also zero downtime.

The study "A Review of Kubernetes Scheduling and Load Balancing Methods", the authors address Kubernetes scheduling and load balancing challenges, emphasizing resource management to maintain application continuity [5]. Efficient scheduling and resource allocation are essential to rolling updates, preventing performance issues when adding or removing pods. Enhancing these methods improves rolling update stability and responsiveness. In comparison to this paper, The resource allocation for rolling updates are optimised in according to the basic experiment's and evaluation experiment's needs accordingly.

The study "Enhancing Kubernetes Auto-Scaling: Leveraging Metrics for Improved Workload Performance" highlights Kubernetes use of real-time metrics for dynamic workload management, which benefits rolling updates by aligning parameters such as maxSurge and maxUnavailable with demand [6]. In comparison to this paper, the metrics during the rolling update is adjusted accordingly to the number of containers. the number of containers directly affect the maxSurge and maxUnavailable values.

## 3. Proposal

To address the time constraints associated with rolling updates in Kubernetes, an optimized deployment approach is proposed that dynamically adjusts the number of containers (pods) to be deployed simultaneously. The two primary parameters in Kubernetes that govern rolling updates are maxSurge and maxUnavailable, which dictate the number of additional and unavailable pods during the update process, respectively [7]. By tuning these parameters, the speed of the update is controlled while balancing resource availability and application continuity.

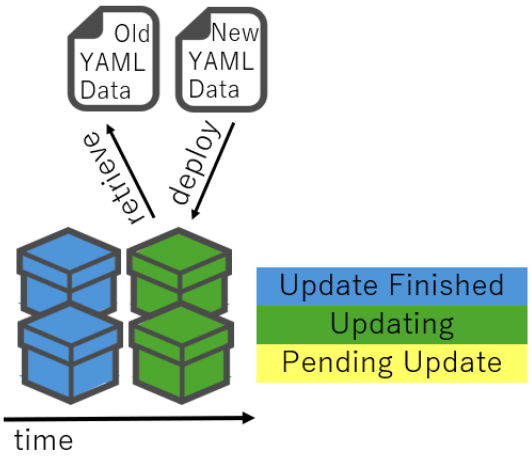


Figure 2: Adjusting number of container deployed in Rolling Update

Figure 2 demonstrates efficiency gains in deployment time by increasing simultaneous pod updates. Figure 1

presents a scenario where rolling updates proceed with only one pod deployed at a time. Figure 2 contrasts this by depicting two pods deployed simultaneously. Specifically, when the total container count is four, the number of containers updated simultaneously is set to half (2). Blue boxes represent updated containers, green ones are updating, and yellow boxes indicate containers pending update. The arrow with the label time shortens in Figure 2 compared to Figure 1, which shows a decrease in time taken for the rolling update. This configuration underscores the time efficiency achieved when the system updates more pods concurrently, handling a larger workload in parallel.

**Basic Experiment**

Adjustment of maxSurge and maxUnavailable parameters optimizes Kubernetes rolling updates. These settings, altered in the deployment YAML file's "spec" section, control simultaneous container (pod) deployment. The test environment includes one master node and two worker nodes. Depending on the test, the deployment configuration's replicas field is set to 15, 20, or 25, aligning the total pod count with these figures.

Values for maxSurge and maxUnavailable range from 1 to 19. For each value, 10 deployment time samples are recorded, averaging to represent each data point. This method secures statistically reliable results by capturing variability in deployment times. The application under update is WordPress, a widely-used open-source content management system facilitating content creation and management, such as blogs and multimedia. WordPress has broad application and the critical need for efficient, continuous updates to preserve website availability and user experience.

Program 1's maxSurge parameter enhances update concurrency by allowing extra pods creation. This parameter exceeds the set replica count during updates, facilitating faster updates for smaller images with higher values, such as 10 or above, and minimizing resource consumption for larger images with lower values. The maxUnavailable parameter specifies the allowable number of pods that is offline during updates. Specifically, when the replicas count is even, maxSurge is set at half; for odd counts,the value follows the basic Experiment's results by taking the higher value. The "ImagePullPolicy: Always" setting ensures updates utilize the latest image, preventing outdated versions from affecting timing accuracy [8].

The rolling update monitoring is conducted using the kubectl command get pod -w. This kubectl command displays the update progression, sequentially replacing each pod. The update concludes once all original pods achieve "Terminating" status. Timing begins with the update application and ends when all old pods are replaced [9].

Program 1: YAML File "spec" Cutout

```
spec:
  replicas: 20
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 10
      maxUnavailable: 10
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - name: wordpress
        image: wordpress:6.2.1-apache
        imagePullPolicy: Always
```

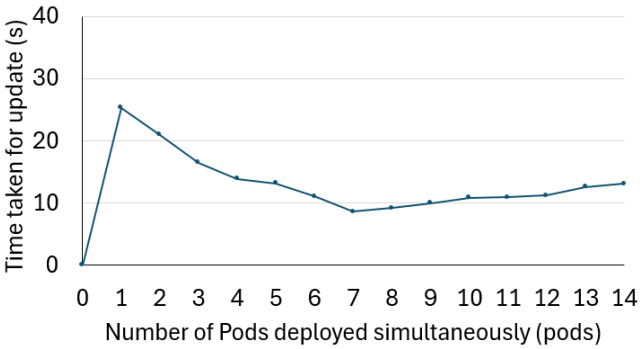


Figure 3: Rolling Update Data when replicas is set to 15

The X-axis on each graph represents the number of pods deployed simultaneously in a rolling update. The Y-axis one ach graph represents the time taken for rolling updates. Figure 3 data reveals that deploying 7 pods simultaneously is faster than deploying a single container, saving approximately 16.4 seconds. This represents an approximately 65% reduction in deployment time. However, deployment times start to rise again when the number of

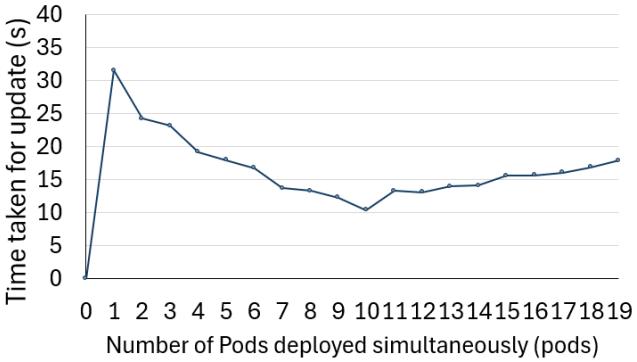


Figure 4: Rolling Update Data when replicas is set to 20

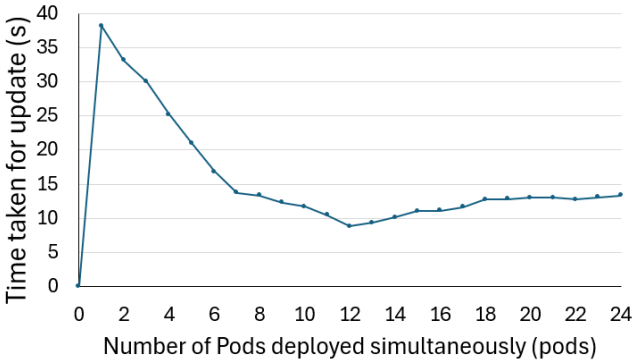


Figure 5: Rolling Update Data when replicas is set to 25

pods increases to 14. Figure 4 shows a similar pattern. Deploying 10 pods simultaneously results in a time saving of approximately 21.3 seconds, an approximately 67% reduction compared to deploying a single container. As with Figure 3, deployment times begin to increase when the number of pods reaches 19. In Figure 5, the trend continues with the most optimal deployment occurring at 12 pods, reducing the deployment time by approximately 28.7 seconds, an approximately 77% reduction. Deployment times start to rise again at 24 pods, aligning with the other figures. Despite these variations, a consistent pattern emerges across all figures: the fastest deployment times occur when the number of simultaneously deployed pods is approximately half the total replicas. Specifically, the optimal pod counts are 7-8 for 15 replicas (Figure 3), 10 for 20 replicas (Figure 4), and 12-13 for 25 replicas (Figure 5). These findings underscore the importance of configuring maxSurge and maxUnavailable values close to half the replicas count to optimize rolling update durations.

### Use Case Scenario

A major update to a critical payment processing service in a Kubernetes-powered e-commerce platform requires deployment to address security vulnerabilities and enhance fraud detection. The platform experiences heavy traffic, causing updates without service interruptions become essential. To achieve this, the update strategy involves tuning parameters such as maxSurge to allow faster updates and maxUnavailable to maintain service availability during the process. Ensuring the shortest possible update duration ensures users maintain easy access to the e-commerce site.

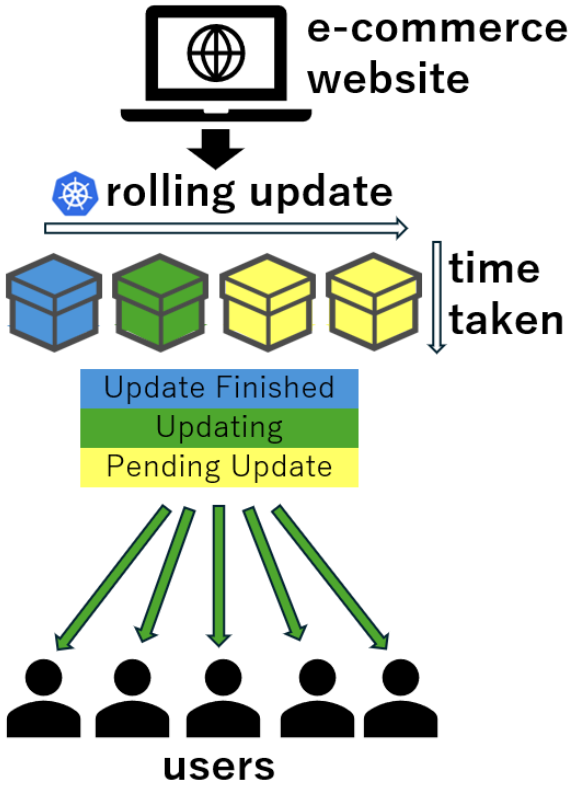


Figure 6: Rolling Update Scenario

Figure 6 demonstrates uninterrupted operation of e-commerce sites during rolling updates. Updates are incrementally applied to a subset of containers, enabling the remaining containers to manage user requests. The blue containers represent updated pods, the green containers represent updating pods, and the yellow containers represent pending pods. Adjusting the number of sequentially deployed pods reduces the time required for completing the rolling update.

## 4. Implementation

### Python Code Implementation

The Python code automates WordPress application updates on Kubernetes clusters. This implementation updates the application from version 6.2.1 to version 6.3.1 using rolling updates. The process optimizes updates by

adjusting key parameters, including maxSurge and maxUnavailable, to control the number of pods deployed simultaneously. The script modifies the deployment configuration in the Kubernetes YAML file, applying adjusted values for maxSurge, maxUnavailable, and replicas to evaluate the impact on deployment time and resource consumption.

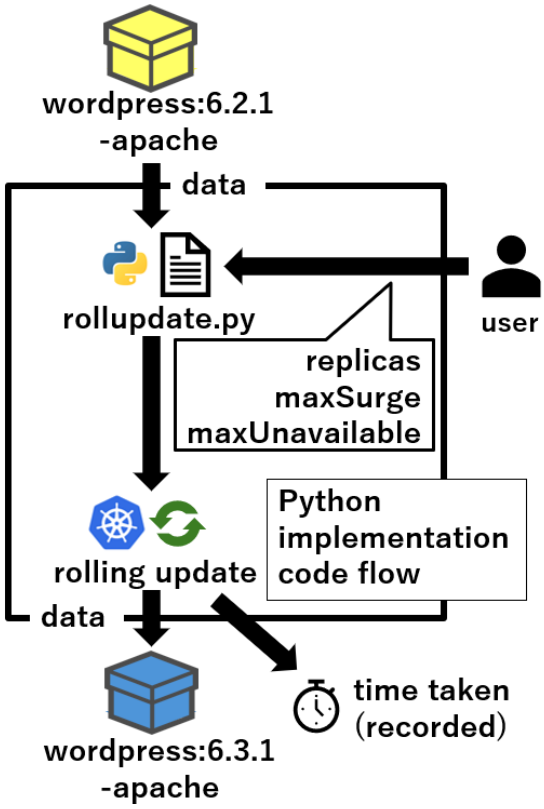


Figure 7: Python Code Flow

Figure 7 outlines the steps for updating the deployment YAML file. The process begins with editing the YAML file to specify the number of replicas, along with maxSurge and maxUnavailable values, as provided by the user. The rollupdate.py Python script reads the YAML file, updates these parameters, and saves the modifications. Once the configuration is updated, the script applies the new deployment using the kubectl apply command. Following this, the script monitors the deployment to ensure all pods reach the "Running" state, confirming the rolling update progresses as expected. The script updates the YAML file from version 6.2.1 to version 6.3.1. The yellow container represents an outdated YAML file, which is version 6.2.1 and the blue container represents an updated YAML file, which is version 6.2.1.

Additionally, the script records the time taken for the rolling update to complete, providing essential data for this research. The recorded duration serves as the primary metric for analyzing the efficiency of different configurations. This analysis evaluates how various maxSurge and maxUnavailable settings impact update speed while balancing resource utilization. To ensure consistent updates, the deployment YAML file includes the imagePullPolicy: Always setting, preventing the use of cached images, as detailed in Program 1.

**Experiment Environment**

- Nodes : One master node and two worker nodes
- Application : wordpress:6.2.1-apache to wordpress:6.3.1-apache
- Software : k3s version v1.25.13+k3s1
- OS : Ubuntu Server 24.04.1 LTS
- vCPU : 1 Core
- RAM : 5 GB
- SSD : 30 GB

The details written above are the environment used for the Evaluation Experiment. By using one master node and two worker nodes, the workload of the Virtual Machine during Rolling updates is distributed to ensure that the updates are executed smoothly. Apart from that, the other settings are all in according to the Laboratory's usual settings for Virtual Machines, ensuring that the virtual machine runs smoothly without problems.

## 5. Evaluation

**Evaluation Experiment**

To assess the impact of varying the number of pods deployed simultaneously during Kubernetes rolling updates, experiments were conducted with the replicas field set to 100. The number of pods deployed simultaneously was varied from 1 to 100, with an increment of 10 pods for each data, and for each configuration, the time taken to complete the rolling update was recorded. One master node and two worker nodes were used in the setup, and updates were performed fully without utilizing cache to ensure consistency. All recorded values represent the average of 10 experiments conducted for each configuration. Additionally, system resource metrics, including CPU load, disk read speed, and disk write speed, were monitored to ensure system stability and analyze the effects of increasing concurrency on resource utilization.

Figure 8 illustrates the relationship between simultaneous pod deployment and update duration in Kubernetes rolling updates. The highest recorded update time is approximately 133 seconds when deploying only 1 pod at

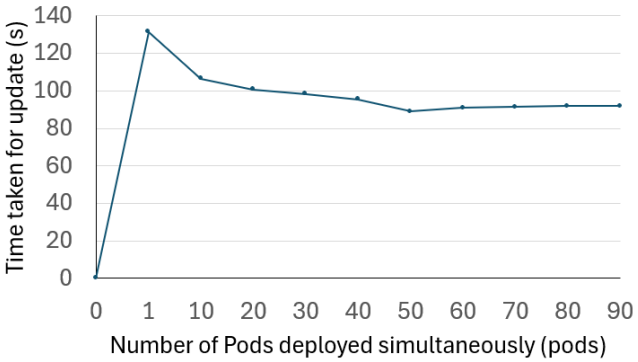


Figure 8: Evaluation Experiment Rolling Update Graph

a time. Increasing the number of pods deployed simultaneously reduces the update duration, reaching the lowest value of approximately 89 seconds when 50 pods are deployed concurrently. This result demonstrates that update duration decreases as concurrency increases.

The graph reveals a turning point: after deploying 50 pods simultaneously, update duration begins to rise. For example, deploying 70 pods takes approximately 91.43 seconds, and deploying 90 pods takes approximately 93 seconds. Although the cause of this increase remains unexplored, potential factors include metrics such as CPU usage and disk read speed.

The graph exhibits a U-shaped pattern, with a decrease in update time as concurrency grows, followed by an increase beyond the optimal configuration. This finding emphasizes the importance of balancing deployment speed and resource utilization to optimize rolling update performance.

**Analysis of Evaluation Experiment Results**

The results indicate that adjusting maxSurge and maxUnavailable values impacts rolling update times in Kubernetes. Increasing these values reduces update durations by deploying more containers simultaneously. The data demonstrates that optimal performance occurs when both maxSurge and maxUnavailable are set to approximately half the total number of replicas.

The evaluation revealed that setting maxSurge and maxUnavailable to approximately 50 resulted in the shortest update time of approximately 89 seconds. Compared to deploying one pod at a time, which took approximately 133 seconds, this configuration reduced the update time by approximately 33%. This outcome highlights the efficiency of updating more pods concurrently [10].

This configuration balances deployment speed and resource management. Deploying fewer pods simultaneously ensures the system handles updates without overloading resources [11]. This balanced approach avoids resource strain, enabling a smoother and faster rolling update process, which minimizes update times while maintaining system stability.

## 6. Discussion

The findings highlight the impact of concurrency on rolling updates and suggest avenues for future exploration. One potential research direction involves understanding why the shortest rolling update times occur when the number of pods deployed simultaneously approximates half the container count. This relates to metrics such as CPU usage and disk read/write speed, potentially influenced by the number of nodes, as each node exhibits distinct metrics.

The nodes in the Evaluation Experiment (one master node and two worker nodes) were distributed across three separate physical machines. This placement results in differing metrics and varying efficiency levels among the machines, which may affect the rolling update process, either prolonging or shortening the duration. Conducting experiments with all three nodes on the same physical machine, recording rolling update times, and comparing these results with those from nodes distributed across different machines reveal the significance of disk read/write speeds in determining update times.

Extending the research to larger clusters, particularly in cloud environments, offers another area for investigation. Medium-sized clusters were used in the experiments. Therefore, exploring larger clusters provides insights into Kubernetes' behavior under varying resource conditions. The elastic scalability of cloud resources introduces dynamic challenges and opportunities in high-concurrency rolling updates, offering a more comprehensive understanding of the system's performance.

## 7. Conclusion

A detailed analysis identifies the benefits of adjusting container updates for faster and smoother system operations. Rolling updates provide updates without downtime and longer durations pose challenges for certain scenarios. Experiments demonstrate that setting maxSurge and maxUnavailable values to approximately half the replicas count optimizes update times. The Evaluation Experiment involves a rolling update of approximately 100 containers, beginning with deploying one pod at a time. Sequential increments of approximately 10 containers (10,

20, 30, 40, 50, 60, 70, 80, 90) increase the number of containers updated simultaneously. Deploying approximately 50 pods, half the total container count, results in the shortest update time of approximately 89 seconds. Deploying one pod at a time, taking approximately 133 seconds, shows a reduction of approximately 33% which is 44 seconds compared to deploying 50 pods simultaneously. The Evaluation Experiment findings indicates that setting maxSurge and maxUnavailable values to 50 when replicas are set to 100. This configuration balances speed and resource usage, achieving the most efficient update times for the specific deployment setup.

## References

[1] Méndez, S.: *Edge Computing Systems with Kubernetes: A use case guide for building edge systems using K3s, k3OS, and open source cloud native technologies* (2022).

[2] Mondal, S. K., Zheng, Z. and Cheng, Y.: On the Optimization of Kubernetes toward the Enhancement of Cloud Computing, *Mathematics*, Vol. 12, No. 16 (online), DOI: 10.3390/math12162476 (2024).

[3] Laukka, L., Fransson, C. and Pappas, N.: Load Balancing Traffic Among Kubernetes Replicas by Utilizing Workload Estimation, *2023 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 353–356 (online), DOI: 10.1109/CSCN60443.2023.10453145 (2023).

[4] Saleh, A. and Karslioglu, M.: *Kubernetes in Production Best Practices: Build and manage highly available production-ready Kubernetes clusters* (2021).

[5] Sun, Y., Xiang, H., Ye, Q., Yang, J., Xian, M. and Wang, H.: A Review of Kubernetes Scheduling and Load Balancing Methods, *2023 4th International Conference on Information Science, Parallel and Distributed Systems (ISPDS)*, pp. 284–290 (online), DOI: 10.1109/ISPDS58840.2023.10235497 (2023).

[6] Joyce, J. E. and Sebastian, S.: Enhancing Kubernetes Auto-Scaling: Leveraging Metrics for Improved Workload Performance, *2023 Global Conference on Information Technologies and Communications (GCITC)*, pp. 1–7 (online), DOI: 10.1109/GCITC60406.2023.10426170 (2023).

[7] Malhotra, A., Elsayed, A., Torres, R. and Venkatraman, S.: Evaluate Canary Deployment Techniques Using Kubernetes, Istio, and Liquibase for Cloud Native Enterprise Applications to Achieve Zero Downtime for Continuous Deployments, *IEEE Access*, Vol. 12, pp. 87883–87899 (online), DOI: 10.1109/ACCESS.2024.3416087 (2024).

[8] Sayfan, G. and Ibryam, B.: *Mastering Kubernetes: Dive into Kubernetes and learn how to create and operate world-class cloud-native systems* (2023).

[9] Hasan, B. T. and Abdullah, D. B.: Real-Time Resource Monitoring Framework in a Heterogeneous Kubernetes Cluster, *2022 Muthanna International Conference on Engineering Science and Technology (MICEST)*, pp. 184–189 (online), DOI: 10.1109/MICEST54286.2022.9790264 (2022).

[10] Abirami, T., Vasuki, C., Jayadharshini, P. and Vigneshwaran, R. R.: Monitoring and Alerting for Horizontal Auto-Scaling Pods in Kubernetes Using Prome Theus, *2023 International Conference on Computer Science and Emerging Technologies (CSET)*, pp. 1–8 (online), DOI: 10.1109/CSET58993.2023.10346811 (2023).

[11] Reddy, Y. S. D., Reddy, P. S., Ganesan, N. and Thangaraju, B.: Performance Study of Kubernetes Cluster Deployed on Openstack,VMs and BareMetal, *2022 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pp. 1–5 (online), DOI: 10.1109/CONECCT55679.2022.9865718 (2022).