UDC 004.4 https://doi.org/10.33619/2414-2948/108/18

A COMPARATIVE STUDY OF RESOURCE MANAGEMENT APPROACHES IN KUBERNETES AND DOCKER SWARM: EFFICIENCY AND SCALABILITY

©***Tiumentsev D.,*** *ORCID: 0009-0003-5275-3223, SPIN-code: 7874-7803,*
East Siberian State University of Technology and Management,
Ulan-Ude, Russia, tyumencev_dv@rambler.ru

СРАВНИТЕЛЬНОЕ ИССЛЕДОВАНИЕ ПОДХОДОВ К УПРАВЛЕНИЮ РЕСУРСАМИ В KUBERNETES И DOCKER SWARM: ЭФФЕКТИВНОСТЬ И МАСШТАБИРУЕМОСТЬ

©***Тюменцев Д. В.,*** *ORCID: 0009-0003-5275-3223, SPIN-код: 7874-7803,*
Восточно-Сибирский государственный университет технологий и управления,
г. Улан-Удэ, Россия, tyumencev_dv@rambler.ru

Abstract. This article presents a comparative analysis of resource management (RM) approaches in two popular container orchestration platforms, Kubernetes and Docker Swarm. The key differences in RM, scheduling, and scaling are discussed, with a focus on the flexibility and granularity of Kubernetes compared to the simplicity and ease of use of Docker Swarm. The advantages and disadvantages of each tool are also analyzed to provide a more complete understanding of their applicability.

Аннотация. Представлен сравнительный анализ подходов к управлению ресурсами в двух популярных платформах оркестрации контейнеров Kubernetes и Docker Swarm. Рассматриваются ключевые различия в управлении ресурсами, планировании и масштабировании, при этом особое внимание уделяется гибкости и детализации Kubernetes по сравнению с простотой и удобством использования Docker Swarm. Также анализируются преимущества и недостатки каждого инструмента для более полного понимания их применимости.

Keywords: resource management, Kubernetes, Docker Swarm, efficiency, scalability, containers, orchestration.

Ключевые слова: управление ресурсами, Kubernetes, Docker Swarm, эффективность, масштабируемость, контейнеры, оркестрация.

In the realm of modern software development and deployment, container orchestration has emerged as a key solution for managing the complexities of large-scale, distributed systems. Containers, by virtue of their portability and consistency, have become the preferred method for packaging and deploying applications. Managing containers at scale demands a robust orchestration tool, and Kubernetes and Docker Swarm have risen to prominence in this space. Both tools aim to simplify the deployment, scaling, and operation of containerized applications, yet they do so with distinct philosophies and methodologies, particularly in their approaches to RM.

Kubernetes, developed by Google, is renowned for its robust and flexible architecture, which enables efficient scaling and resource utilization across large clusters. In contrast, Docker Swarm, which is developed as part of the Docker ecosystem, emphasizes simplicity and ease of use. The aim of this study — to present a comparative study of RM approaches in Kubernetes and Docker Swarm in terms of efficiency and scalability.

Comparison of Kubernetes and Docker Swarm characteristics and features as containerization tools

The rise of containerization has reshaped software development and deployment by offering an efficient, consistent approach to packaging applications and their dependencies. As organizations continue to adopt containerized environments, managing and orchestrating containers across distributed systems presents new challenges [1]. Resource efficiency and scalability are key concerns at the application level in today's various IT sectors [2]. Tools like Kubernetes and Docker Swarm have emerged to address these problems, providing automated solutions for scaling containerized applications.

Kubernetes, often abbreviated as K8s, become the standard for container orchestration, largely due to its rich feature set and extensive community support. Kubernetes was inspired by Google's internal system, Borg, which was designed to manage and scale containerized workloads efficiently across its vast data centers. According to statistics (https://lyl.su/PNlM), in 2023 it became the second most used containerization tool in the world after Docker (Figure 1).

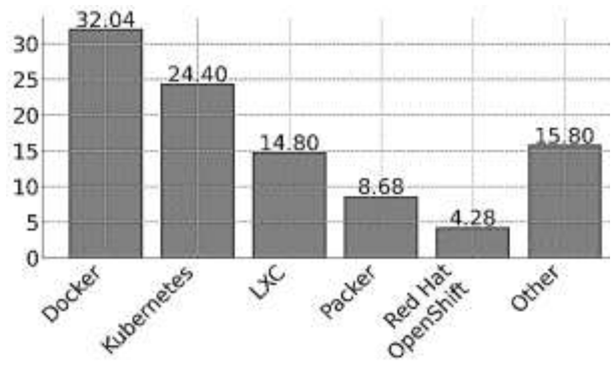

Figure 1. Leading containerization technologies market share worldwide in 2023, %

Docker Swarm is a native clustering and orchestration tool for Docker containers. It was created to provide an easy-to-use, integrated solution for orchestrating containerized applications. Unlike Kubernetes, which was designed as a standalone orchestration platform, Docker Swarm is tightly integrated with the Docker ecosystem, making it a natural choice for users already familiar with Docker's command-line interface (CLI) and tools (https://docs.docker.com/engine/swarm/). Docker Swarm's simplicity and ease of use have made it popular among small to medium-sized teams and projects that require quick setup and straightforward management of containerized environments (Figure 2).

While Kubernetes and Docker Swarm share the common goal of orchestrating containerized applications, they differ significantly in their design philosophies, feature sets, and operational complexity. Kubernetes is designed to support large-scale, multi-tenant environments. Its sophisticated RM and autoscaling mechanisms make it suitable for managing complex workloads across diverse infrastructures. According to IBM specialists (https://www.ibm.com/think/topics/docker-swarm-vs-kubernetes), Docker Swarm, while capable of handling sizable clusters, is generally considered more suitable for smaller-scale deployments or scenarios where quick setup and ease of management are priorities (Table).

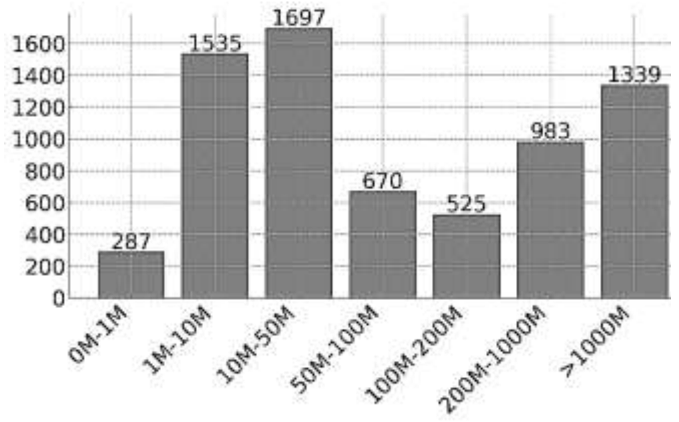

Figure 2. Distribution of worldwide companies that use Docker Swarm based on company size by revenue for the years 2016–2023 (https://lyl.su/rXj6)

Table

ADVANTAGES AND DISADVANTAGES OF KUBERNETES AND DOCKER SWARM

Aspect	*Advantages of Kubernetes*	*Disadvantages of Kubernetes*	*Advantages of Docker Swarm*	*Disadvantages of Docker Swarm*
Scalability	High, with autoscaling.	Requires more resources.	Suitable for smaller clusters.	Manual scaling.
Resource control	Granular control.	Requires detailed configuration.	Simple resource reservation.	Less granular control.
Ecosystem	Extensive tools and plugins.	Complex to integrate.	Seamless Docker integration.	Limited extensibility.
Complexity	Advanced features, flexibility.	Steep learning curve.	Easy to set up.	Limited advanced features.

Kubernetes offers a more complex and feature-rich environment, providing advanced capabilities such as custom resource definitions (CRD), role-based access control (RBAC), and intricate scheduling policies. This complexity allows for fine-grained control over container orchestration but also introduces a steep learning curve. Docker Swarm, in contrast, emphasizes simplicity and ease of use. Its straightforward setup process and Docker-native CLI make it more accessible to teams with limited orchestration experience, albeit at the cost of some advanced features.

A vast ecosystem of tools, plugins, and integrations is available in Kubernetes, supported by an active open-source software development community. Its extensibility allows users to customize their orchestration environment, integrating tools for logging, monitoring, and security. Docker Swarm, being more opinionated and tightly integrated with the Docker ecosystem, offers fewer customization options but provides a cohesive experience for users who prefer a more straightforward, out-of-the-box solution.

Analysis of various approaches to RM in Kubernetes and Docker Swarm

Resource allocation is an important aspect of container orchestration, as it directly influences the efficiency, performance, and reliability of containerized applications. Both Kubernetes and Docker Swarm provide mechanisms to manage resources, but they differ significantly in their design philosophies and implementation.

Kubernetes adopts a declarative approach to RM, allowing users to define the desired state of the system and letting the platform ensure that the actual state matches the desired one. This is achieved through several key components.

Namespaces are used in Kubernetes to partition the resources of a cluster into separate, virtual sub-clusters. It provides a mechanism to isolate assets within a cluster, making it possible to organize and manage capacity for different teams or projects efficiently. By setting asset quotas and limits at the namespace level, Kubernetes can ensure that no single namespace monopolizes the cluster's resources, thereby promoting fair capacity utilization.

In Kubernetes, requests and limits are the key mechanisms for managing the CPU and memory usage of containers. These requests specify the minimum amount of computing power needed by a container to operate, ensuring that the Kubernetes scheduler assigns the pod to a node with enough available capacity. Limits, on the other hand, define the maximum amount of computing resources a container can consume, preventing it from using more than its allocated share. These controls help maintain the overall health of the cluster by avoiding contention and over-utilization.

Advanced resource scheduling capabilities are provided in Kubernetes through node affinity and taints/tolerations. Node affinity allows users to specify rules for pod placement based on node labels, ensuring that pods are scheduled on nodes that meet specific criteria. Taints and tolerations, conversely, enable nodes to repel certain pods unless they have the appropriate tolerations. This mechanism helps in creating dedicated or specialized nodes within the cluster, optimizing resource usage according to the requirements of different workloads [3].

Kubernetes supports both horizontal and vertical autoscaling to dynamically adjust resource allocation based on current demand. Horizontal Pod Autoscaling (HPA) adjusts the number of pod replicas in response to CPU or memory usage, allowing the system to scale out or in as needed. Vertical Pod Autoscaling (VPA) modifies the resource requests and limits of individual pods, adjusting their CPU and memory allocation to match the observed resource usage patterns. These autoscaling mechanisms enhance the efficiency of resource utilization and ensure that applications remain responsive under varying loads.

A simpler and more streamlined approach to resource management is offered in Docker Swarm, with a focus on ease of use and rapid deployment. While it may not provide the same level of granularity as Kubernetes, Docker Swarm’s RM features are effective for many use cases.

Docker Swarm, like Kubernetes, allows users to set resource limits and reservations for services. Resource reservations define the amount of CPU and memory a service requires, ensuring the scheduler places it on a node with adequate resources. In contrast, resource limits cap the maximum CPU and memory a service can use, preventing it from overloading the node. Together, these settings help distribute workloads evenly across the cluster.

Placement constraints and preferences are used in Docker Swarm to control where services are deployed within the cluster. Placement constraints allow users to specify conditions that a node must meet to run a service, such as node labels or node roles. Placement preferences enable a finer degree of control by specifying rules that influence the placement of services, such as spreading services evenly across different nodes or preferring nodes with specific characteristics. These features facilitate efficient resource utilization by ensuring that services are deployed on the most suitable nodes.

Unlike Kubernetes, Docker Swarm does not have built-in autoscaling features for services. However, it can be achieved through external tools and custom scripts that interact with the Docker Swarm API. For example, users can implement a custom autoscaler that monitors service metrics and adjusts the number of replicas accordingly. While this approach requires additional setup and maintenance, it offers flexibility in how autoscaling is implemented within a Swarm cluster [4].

In summary, Kubernetes' use of namespaces, resource requests, and limits, along with advanced scheduling features like node affinity and taints/tolerations, enables fine-grained control over resource allocation and ensures optimal utilization of cluster resources. Kubernetes' built-in autoscaling capabilities allow for dynamic adjustment of computing resources in response to changing demands, enhancing both efficiency and scalability, which ultimately improves user satisfaction [5].

In contrast, Docker Swarm's RM is more straightforward, catering to users who prioritize simplicity and ease of use. Its reservation and limits system, combined with placement constraints and preferences, provide essential control over resource distribution. The lack of native autoscaling in Docker Swarm means that achieving dynamic resource adjustment requires additional effort and external tools.

Conclusion

The choice of a container orchestration platform is a significant decision for organizations aiming to optimize the deployment, management, and scaling of their applications. Kubernetes and Docker Swarm, as two of the most prominent solutions, represent different philosophies in RM and operational complexity. The selection between these two platforms hinges on an organization's specific needs regarding scalability and RM. As containerization continues to shape the future of application development and deployment, understanding the fundamental strengths and trade-offs of Kubernetes and Docker Swarm will remain essential for making informed decisions about infrastructure design.

References:

1. Mozharovskii, E. (2024). Performance Analysis of Flutter Applications vs. Native iOS and Android Apps. *Mezhdunarodnyi zhurnal gumanitarnykh i estestvennykh nauk*, (8-2 (95)), 150-155.

2. Pshychenko, D. (2024). Evaluation of the effectiveness of implementing AI-based CRM systems. *Innovacionnaja nauka*, (7-2), 40-45.

3. Balla, D., Simon, C., & Maliosz, M. (2020, April). Adaptive scaling of Kubernetes pods. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium* (pp. 1-5). IEEE. https://doi.org/10.1109/NOMS47738.2020.9110428

4. Singh, N., Hamid, Y., Juneja, S., Srivastava, G., Dhiman, G., Gadekallu, T. R., & Shah, M. A. (2023). Load balancing and service discovery using Docker Swarm for microservice based big data applications. *Journal of Cloud Computing*, *12*(1), 4. https://doi.org/10.1186/s13677-022-00358-7

5. Ogarkov, A. (2024). Application of big data analytics to improve business customer service. *Innovacionnaya nauka*, (7-1), 61-65.

Список литературы:

1. Mozharovskii E. Performance Analysis of Flutter Applications vs. Native iOS and Android Apps // Международный журнал гуманитарных и естественных наук. 2024. №8-2 (95). С. 150-155.

2. Pshychenko D. Evaluation of the effectiveness of implementing AI-based CRM systems // Innovacionnaja nauka. 2024. №7-2. P. 40-45.

3. Balla D., Simon C., Maliosz M. Adaptive scaling of Kubernetes pods // NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2020. P. 1-5. https://doi.org/10.1109/NOMS47738.2020.9110428

4. Singh N., Hamid Y., Juneja S., Srivastava G., Dhiman G., Gadekallu T. R., Shah M. A. Load balancing and service discovery using Docker Swarm for microservice based big data applications // Journal of Cloud Computing. 2023. V. 12. №1. P. 4. https://doi.org/10.1186/s13677-022-00358-7

5. Ogarkov A. Application of big data analytics to improve business customer service // Innovacionnaya nauka. 2024. №7-1. P. 61-65.

Работа поступила в редакцию 16.10.2024 г.

Принята к публикации 22.10.2024 г.

Ссылка для цитирования:

Tiumentsev D. A Comparative Study of Resource Management Approaches in Kubernetes and Docker Swarm: Efficiency and Scalability // Бюллетень науки и практики. 2024. Т. 10. №11. С. 140-145. https://doi.org/10.33619/2414-2948/108/18

Cite as (APA):

Tiumentsev, D. (2024). A Comparative Study of Resource Management Approaches in Kubernetes and Docker Swarm: Efficiency and Scalability. *Bulletin of Science and Practice, 10*(11), 140-145. https://doi.org/10.33619/2414-2948/108/18