ISSN: **3107-6513**

Auto Scaling Web Application Using Docker +Kubernetes

Sai Shrimathi K[1], Thamizharasan.N [2]

Department of Computer Science, Rathinam College of Arts and Science (Autonomous), Coimbatore, Tamil Nadu, India

Corresponding Author: saishrimathikannan2006@gmail.com

Abstract: *Due to the increase in use of web applications at a dramatic rate, developers are looking for solutions to effectively manage variable workloads while delivering high-performance and high-reliability. Therefore, this project will present a method for building an "auto-scaling" web application (built using Docker/Kubernetes) as well as a demonstration of the use of Kubernetes' Horizontal Pod Auto scaler (HPA) to dynamically scale application have the means (like containers), to be deployed consistently across environments. By utilizing Kubernetes to manage the deployment and scaling of containers, we can use the HPA to adjust the number of containers in real-time based on the metrics collected such as how much CPU is being utilized and how much traffic is entering into the application. As a result, we can effectively deploy resources as needed, reduce operational expenses and maintain a high availability factor - especially during high traffic loads. Lastly, using load balancing techniques, we can direct user requests to multiple instances of the application, thus improving both performance and fault-tolerance of your applications overall. Thus, this project is an example of using containerization and orchestrating to create a scalable, efficient, and resilient architecture (in real world instances) to build web-based application.*

Keywords *– Docker, Kubernetes, Auto-scaling, Containerization, Horizontal Pod Auto scaler (HPA), Load Balancing, Cloud Computing, Microservices, Scalability, High Availability*

I. INTRODUCTION

Web-based applications require great scalability and availability to ensure efficient operation when dealing with unexpected and fluctuating workloads. The conventional methods used for deploying software may have some constraints when responding to changes in workload. In order to overcome such problems, new technologies like Docker and Kubernetes have been invented.

The Docker technology allows developers to bundle up applications and dependencies together in portable packages known as containers to ensure seamless deployment from one environment to another. On the other hand, Kubernetes technology provides a platform to automate the process of managing and deploying containers.

The project highlights the use of auto-scaling of a web application through Docker and Kubernetes in terms of maximizing resource usage, providing high availability, and ensuring fault tolerance of the system. With the use of the Horizontal Pod Auto scaler (HPA), the system is capable of scaling the number of instances based on the current condition of the workload. This means that when there is peak load, more instances are added, but when the load is low, instances will be decreased to save resources.

Apart from making a system scalable, another feature it needs to have is adaptability due to the continuous changes of the workload. Automation in resource management makes this possible since

ISSN: 3107-6513

there is no need for continuous manual management in order to make the system adapt to changes In addition, visualization and monitoring are important in the process of gaining an understanding of how the system behaves. Real-time monitoring helps to gain insight into the performance parameters, aiding in decision-making and optimizing the system. With the inclusion of a comprehensive monitoring system, it becomes easier to manage the resources effectively and efficiently.

II. EXISTING AND PROPOSED SYSTEM

2.1 Existing system

In traditional web applications, the underlying architecture is characterized by static infrastructure. This means that the physical resources do not vary based on fluctuations in the volume of traffic to the site. Hence, the number of servers or applications deployed in the site cannot be changed in relation to changing workloads, thereby making for an inefficient system design. As a result, allocation of resources in such a system happens manually through system administrators.

Inefficient scaling poses several challenges. For instance, during high levels of traffic, there is likely to be an overload of resources, resulting in slower response times. Conversely, during low traffic levels, resources would be underused, which translates into inefficiencies. The second major problem faced by traditional computing environments is their lack of effective means of workload balancing. In order for tasks to be equally distributed among the available computing resources, there needs to be some sort of coordination, and without that, some computing devices are likely to be overloaded, while others will be underutilized.

2.2Drawbacks of Existing System

The limitations of current web app deployment/scaling technologies limit the performance/efficiency of web apps today; there is a problem with automation in that it is often done manually leading to much dependency on the HR department for continual supervision/intervention; There are numerous problems with system availability related to resource utilization for both the high-traffic/low-traffic periods – this leads to inadequate resources for production work (poor performance) during low-traffic times and overuse of production resources at times of high-traffic. There is little scalability due to the inability of most old systems to dynamically adjust to workload variations so most systems are not able to handle high amounts of work at a given time. The legacy platforms used for deployment do provide some level of auto-scaling capability; however as with all other legacy systems those systems rely on external monitoring sources for configuring the auto-scale process; thus legacy systems used for deploying web applications will become increasingly difficult for first-time users to configure and navigate; it is impossible to visualize real-time operation of the web application and use of this operation to make rational decisions about how to scale your application. These issues create an illustrative example of the necessity for an automated, scalable, easy to use solution for managing modern web applications. Web applications' deployment and scalability currently have substantial issues that affect their overall efficiency and performance. The

With all these needs taken into consideration, this project aims at coming up with a single platform that will cater to all these needs. Scalability, automation, and real-time monitoring are incorporated into this platform. Through this, the system is able to monitor all the performance parameters and automatically manage resource allocation within a certain threshold.

ISSN: 3107-6513

first problem lies in the limited level of automation because most processes are done manually. Therefore, a lot of effort from users is needed to monitor the web application continuously and ensure its smooth operation.

The second challenge relates to resource usage since it is inefficient when not managed well. When traffic is high, the system may not have enough resources, thereby decreasing its efficiency and availability. On the other hand, when the traffic is low, there might be excess resources that are underutilized.

2.3Proposed System

The proposed system is a smart web app that will automatically adapt to user demand based on the underlying data type needed to be processed. This web app will utilize a cloud-native architecture and will leverage Docker, Kubernetes and Streamlit to provide a flexible cloud solution to all end users.

The entire solution will be containerized (via Docker), thus allowing for portability, repeatability, and ease of deployment across different environments. The system will utilize Kubernetes (K8s) as the primary orchestrator to deploy containers to create highly available instances, to perform load balancing, and to automatically scale-in/out the system.

The proposed system will automatically adjust the number of active pods based on the defined workload parameters in order to determine the number of pods that should be executed within a pre-determined range, thereby scaling-up when experiencing high volume of traffic, and scaling-down when experiencing low volume of traffic, this will ensure efficient use of the system's resources and that the same level of application performance will be maintained without any management or human intervention. By using Kubernetes as an orchestration tool, we can schedule deployments (launching) of containers at scale and perform application functionality in containers on several different hosts. The Kubernetes platform provides essential tools such as load balancing, fault-tolerant/disaster-recovery features, automatic scheduling of resources, pod distribution across multiple nodes, and so on.

When using Kubernetes as an orchestration tool, it uses Kubernetes' Deployments and Services to create a specific number of applications and create pods and distribute any incoming requests amongst those pods.

Another key feature of our Proposed System is the Horizontal Pod Auto scaler (HPA) feature of Kubernetes, which allows us to dynamically create additional pods based on system performance metrics such as CPU usage and the number of incoming requests. With this feature enabled, our Proposed System will automatically create additional pods when the system detects periods of high demand and delete/add pods when the system detects low demand (thus optimizing performance and resource utilization).

In addition to using Kubernetes as an orchestration tool and using the HPA feature of Kubernetes for scaling our system in real time based on load and demand, the Proposed System includes a monitoring and visual component via Streamlit. This dashboard provides real time insight into system performance metrics such as CPU usage, Memory Use, and Number of Active Pods. Using the information from this real time dashboard, users will be able to visually monitor the behaviour of the scaling of our Proposed System and make better decisions on how to optimize their system.

2.4Advantages of Proposed System

Several benefits can be derived from the suggested approach in contrast to existing methods used to deploy web applications. One such advantage lies in the dynamic nature of the auto-scaling offered by the

ISSN: 3107-6513

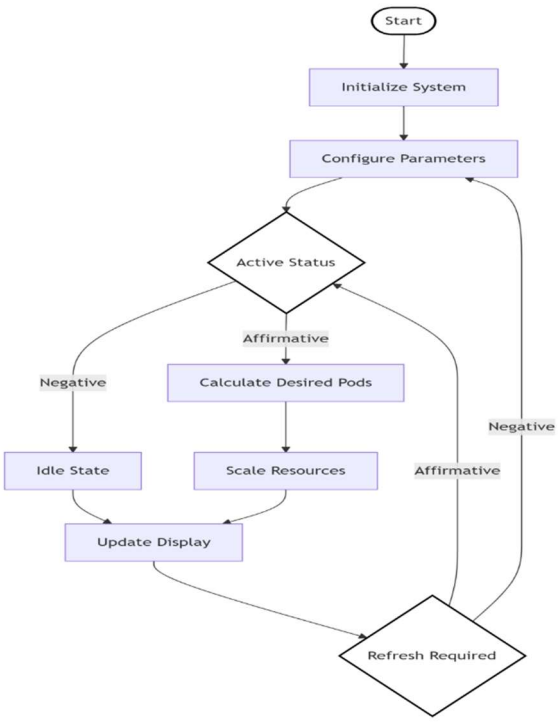
system, which obviates the need for any human intervention. Through this method, it becomes possible to achieve optimal allocation of resources depending on demand levels, resulting in efficient use of resources and reduced costs.

Moreover, the system allows for improved application performance as well as fault tolerance through the maintenance of an optimal number of instances, thus guaranteeing that the system handles incoming requests with minimal downtime. Moreover, the inclusion of a real-time monitoring and visual dashboard gives detailed information about the performance metrics of the system, which include CPU usage, pod state, and scaling. These insights help the user gain better understanding of how the system operates and aid in decision-making regarding its optimization. In summary, the suggested system is more scalable, reliable, and uses resources effectively, thus being a good solution for web applications.

IV. RESULT

The Auto-Scaling Web Application using Docker and Kubernetes was successfully developed and tested under varying workload conditions. The system demonstrated stable and efficient scaling behavior by dynamically adjusting the number of pods based on real-time demand. During peak load periods, the number of pods increased automatically to maintain performance, while during low load periods, the system reduced the number of pods to optimize resource usage. The application maintained continuous operation without downtime throughout the testing phase. It effectively optimized resource utilization while ensuring consistent performance across different workload scenarios. The Streamlit dashboard provided real-time monitoring of key metrics such as CPU usage and active pod count,

DATA FLOW DIAGRAM



enabling effective observation of system health. The auto-scaling process was performed promptly without any significant delay between making the decision and taking the necessary actions to improve performance. The metrics for resources used confirmed proper dynamic resource management since CPU and memory usage were kept within optimal parameters.

Furthermore, the system exhibited high reliability with minimal response delay and no service interruptions. Kubernetes efficiently handled container orchestration, including load balancing and automatic recovery from failures. The monitoring interface clearly visualized scaling activities, confirming the effectiveness of the auto-scaling mechanism.

ISSN: **3107-6513**

V. CONCLUSION

Demonstrating that cloud-native technology can be leveraged for creating scalable and reliable web systems. Providing a dynamic method for scaling applications based on real-time workloads, the proposed solution also provides consistent levels of performance with high availability under different volume scenarios.

The use of containerization and orchestration technologies will allow the deployment process to be more flexible, enabling a greater degree of portability and better resource utilization. Kubernetes will automate much of the scaling, load balancing, and fault recovery operations, thus reducing the amount of manual intervention required and increasing the overall reliability of the system. The use of the Horizontal Pod Autoscaler to allocate resources will further ensure that applications are not underutilizing or over-utilizing available resources.

In addition, the real-time visualization and monitoring dashboard will enhance transparency into how the system is performing by providing metrics regarding CPU usage, pod status, and scaling activity. This will facilitate improved decision making and proactive management of the system as it becomes evident from those metrics.

Experimental evaluation of the system indicates that the proposed system provides improved scalability, fault tolerance, and resource management compared with traditional deployment methods, with the system providing stable performance at low latency and maintaining service availability despite fluctuations in workload volume.

Using Docker containers creates an environment that is consistent, portable (extending to multiple platforms) and allows for seamless application deployment. And building on this foundational architecture is Kubernetes's orchestration capabilities: automated scaling out of your application, load balancing across your current application, self-healing of containers that fail or are terminated unexpectedly, and optimal resource allocation via scheduling.

Adding the Horizontal Pod Autoscaler to the mix makes a tremendous contribution to improving the performance of the overall system. The Horizontal Pod Autoscaler enables real-time scaling decisions by consuming performance metrics (e.g. CPU usage).

Furthermore, a well-designed, real-time monitoring and visualization dashboard will greatly enhance the transparency and visibility of the system, because it provides users with highly detailed performance-related information: (e.g. CPU usage information for each container (pod), current status of all containers (pods) and overall system status as well as the conditions required to perform scaling actions in real-time). This allows each user to fully appreciate the application's performance in order to proactively make adjustments to optimise overall system performance. Thus increasing the overall ability to manage and operate the system more effectively.

REFERENCES:

1. Docker Inc., *Docker Documentation*. [Online]. Available: https://docs.docker.com/ (Accessed: April 2026).
2. Cloud Native Computing Foundation, *Kubernetes Documentation*. [Online]. Available: https://kubernetes.io/docs/ (Accessed: April 2026).
3. Kubernetes Authors, *Minikube Documentation*. [Online]. Available: https://minikube.sigs.k8s.io/docs/ (Accessed: April 2026).

ISSN: **3107-6513**

4. Kubernetes Authors, *kubectl Command Reference*. [Online]. Available: https://kubernetes.io/docs/reference/kubectl/ (Accessed: April 2026).
5. Streamlit Inc., *Streamlit Documentation*. [Online]. Available: https://docs.streamlit.io/ (Accessed: April 2026).
6. K. Hightower, B. Burns, and J. Beda, *Kubernetes Up & Running*, O'Reilly Media, 2017.
7. N. Poulton, *Docker Deep Dive*, Independently Published, 2020.
8. M. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, pp. 2–7, 2014.
9. B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.
10. A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," *European Conference on Computer Systems*, 2015.
11. P. Sharma, S. Chandra, and A. Singh, "Performance Analysis of Container-based Virtualization for Cloud Computing," *International Journal of Computer Applications*, vol. 135, no. 7, pp. 15–20, 2016.
12. D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.
13. P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.
14. A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, 2016.
15. S. Newman, *Building Microservices: Designing Fine-Grained Systems*, O'Reilly Media, 2015.
16. J. Turnbull, *The Docker Book: Containerization is the New Virtualization*, 2014.
17. K. Zhang, M. Chen, and L. Zhang, "Performance Analysis of Docker Containers and Virtual Machines," *Journal of Cloud Computing*, 2018.
18. T. Hightower, *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications*, O'Reilly Media, 2019.