Deploying Microservices on Kubernetes Clusters and Improving Scalability Using a Custom Scheduler

Adarsh Kumar
Department of computer science and engineering
PES university
Bengaluru, India
pesug20cs016@pesu.pes.edu

Yuvaraj D C
Department of computer science and engineering
PES university
Bengaluru, India
dcyuvaraj@gmail.com

Veena Garag
Department of computer science and engineering
PES university
Bengaluru, India
veenagarag929@gmail.com

Suchit S Kallapur
Department of computer science and engineering
PES university
Bengaluru, India
such4869@gmail.com

***Abstract*— Project deals with deploying microservices on cloud and reducing the resource consumption like CPU memory, and bringing down the cost of creation of pods. The project is aimed to improve and optimize resource utilization, improve security features and be able to monitor the performance and visualize the metrics in order to study the behavior of the microservices to increase load.**

Keywords — Kubernetes, Custom Controller, Microservices, Load testing, Monitoring, External access, Custom Controller (key words)

I. INTRODUCTION

In the field of software development, the need to effectively manage and scale applications has become further more important. In the ever-growing and increasing requirements of the modern applications, the traditional architectures struggle and when facing fluctuating workloads and fail to adapt to the fluctuations which in turn leads to performance bottlenecks and potential downtime. Because of such issues and facing such challenges, the combination of Kubernetes and microservices architecture has emerged as a useful and efficient solution by providing a robust and scalable method to building the required modern applications.

In this paper, we work on the implementation of the custom controller auto-scaler in a Kubernetes Cluster environment to scale the microservices based application dynamically. In this approach we are benefitting from the advantages of both Kubernetes and microservices to scale the application seamlessly and ensure optimal performance under dynamically varying load conditions.

A. Background

Kubernetes is an open-source container orchestration platform that has changed the way applications are deployed, managed, and scaled in cloud-native environments. One of the main features is its ability to automate some of the steps involved in managing containerized applications manually which has made it a very valuable tool for many developers and operations teams.

On the other hand, microservices architecture promotes the increase of smaller and independently deployable services in place of complex applications. This type of modular approach increases development efficiency, enabling of finer-grained scalability, and simplifying maintenance making it ideally suited for dynamic workloads in modern application.

Scaling applications has traditionally been a manually and often a reactive process, requiring operators to manually adjust the number of deployable instances while monitoring the resource utilization. This is a very time-consuming method and is prone to making errors. It also fails to adapt to the demands of modern applications which is rapidly changing.

B. Proposed Solution

As a solution to the issues and challenges, a custom controller auto-scaler being implemented in the Kubernetes environment is proposed. The custom controller dynamically adjusts the number of microservice instances based on the current workload by monitoring the application metrics like CPU usage and request latency continuously. This ensures that the application will be able to handle the fluctuating traffic without failing on the user-experience or the performance.

II. RELATED WORK

The survey on several research papers related to Kubernetes and microservices lead to insights on the improvements possible in resource utilization and autoscaling.

[1] suggests the use of custom controllers to optimize the pod usage and bring down the total pods as compared to default Kubernetes HPA. The default Kubernetes method results in ineffective resource allocation, declining performance, and increased maintenance costs. As a result, they created the new algorithm, known as Custom Controller.

Default KHPA is more expensive than Custom scheduler algorithms. Using the Custom Controller nearly lowered the maintenance costs in half. In accordance with workload, Custom Controller manages and scales Pods dynamically. The system consists of one master node and two worker nodes. In this study, an effort is made to create a custom controller to calculate the necessary number of pods to prevent resource waste and website crashes. The Kube-controller manager component is present. The Kubernetes cluster uses YAML. The parameters in the YAML file are sent to the Kube-API server and then to the Kube-controller management to control replicas. The custom controller is then informed to start or create new pods by the Kube-API server.

[2] suggests a technique that applies service-specific routing across the Istio control plane to inject sidecar proxies onto every microservice using service mesh Istio and

dynamically balancing the load among services. Due to its static nature, the default Kubernetes load balancing strategy performs poorly as the workload on the application grows and is unable to handle the fluctuating traffic. Encrypting the communications between services with mTLS ensures the security of inter-service communication.

The Istio-ingress gateway is configured using the ingress host, ingress port, and secure ingress port. By dispersing the incoming load among the many application services, this gateway serves as a crucial load balancer. In Google Cloud Platform, firewall rules are created to permit the ingress port and secure ingress port that the Istio-ingress gateway will use. Envoy proxies are installed as sidecars to each pod that runs a service. The entire set of security policies and dynamic routing must be applied to each service is included in a YAML file with the titles Virtual Service and Enable mTLS, respectively. These policies are then applied to each sidecar found in each pod utilizing a control plane

In monolithic systems, scaling often requires replicating the entire application, leading to potential resource inefficiencies. Microservices, with their independent scalability, enable more efficient resource utilization by allocating resources to specific services based on demand. This flexibility is particularly advantageous in dynamic and evolving environments.

[3] Because the microservices operating on the containers are isolated from one another, managing the containers is a constant responsibility for any organization. As a distinct entity, they operate on their own. Here's where Kubernetes comes into play. All Kubernetes is a container management platform. Install and configure the Kubernetes cluster on each of these three nodes using a master and two worker/slave architectures. This will provide enhanced container failover in the event that a virtual machine fails or reboots. They distribute the load evenly among all application containers by using the Round Robin algorithm.

The suggested system providing a complete solution can be broken down into six parts. The setup phase, the initial module is responsible for starting the system configuration. An infrastructure is created by the system for Kubernetes Cloud and Containers in the second module, which then sets the stage for an adaptable and scalable deployment. The third module mainly concentrates on application deployment, which guarantees a smooth software integration. Deployment of backend scripts is introduced in the fourth module which carries out the suggested algorithm. To maintain traffic balance, these scripts dynamically scale the web server containers depending on predetermined limitations while simultaneously inserting them behind a load balancer. This is achieved by real time monitoring of traffic trends. The fifth module is introduced in order to maximize resource efficiency and system stability by placing a strong emphasis on dynamic scaling and high availability. Finally, the sixth module produces multiple comprehensive analysis reports that offer insightful information regarding the functionality and the traffic patterns of the system. When these modules are combined, there is a stable and flexible architecture that can effectively handle the containerized applications in a demanding and changing environment.

For large traffic platforms and apps, the study developed a dynamic scaling algorithm that takes backend latency into account. The system will dynamically scale up and down in response to user requests and traffic demand.

[4] A multi-cloud environment is used by Cloud-Architects to eliminate vendor dependence. Utilizing numerous cloud service providers and providers within a single cloud network infrastructure is known as multi-clouding. Utilizing multi-cloud settings allows for the distribution of computing resources, low downtime, and high data availability—all advantages offered by various suppliers.

This multi-cloud will be a hybrid environment that leverages various infrastructure settings, including public and private clouds. Selecting a cloud solution from a single vendor has benefits and drawbacks. Positively, dealing with a single party streamlines negotiations and agreements by simplifying the process. When there are only one vendor handling all of the work, it makes it less likely for people to point the finger at one another when something goes wrong. Dealing with a single source makes support and services easier to understand and allows for seamless service integration. There is less of a requirement for in-depth user training, and compatibility problems are less common. This strategy, nevertheless, is not without its problems. Price talks might not go as well if you have less negotiating power. Reliance on the policies of a single vendor may restrict options and freedom of choice, thus resulting in vendor lock-in problems.

Cloud with Multiple Vendors: Anthos:- Google's take on the robust Istio open-source project is called Anthos Service Mesh, which lets you monitor, control, and secure your services without requiring any changes to your application code. Multi-public cloud architecture offers many benefits, including low latency, negotiation power, risk reduction, and the ability to leverage the distinctive qualities of many vendors' platforms to meet specific business needs and promote innovation. A variety of technologies, including Prometheus, Istio, Grafana, and Kubernetes, are employed to make this concept a reality. Two clusters running on the infrastructure of cloud suppliers are connected via a VPN connection. Here, Kubernetes is crucial to the management of workloads across cloud platforms and to the automation of several manual processes, including rollouts, rollbacks, and horizontal scaling.

III. TOOLS AND TECHNOLOGY

Kubernetes (Minikube): kubernetes cluster with minimum of two nodes master and slave

Prometheus, Grafana(monitoring tools) and Kubernetes metric server.

Kiali, graphical visualizer of the microservice application

Locust : load generating tool for testing the application

WinSCP and PuTTY : tools to remote login to live server instance and deploy application

IV. CUSTOM SCHEDULER

Kubernetes, a container orchestration platform, provides various auto-scaling mechanisms to efficiently manage resource utilization and application performance. The Horizontal Pod Auto-scaler (HPA), Vertical Pod Auto-scaler (VPA), and Cluster Auto-scaler are the primary built-in auto-scalers, each addressing specific scaling needs.

The HPA dynamically adjusts the number of replicas (pod instances) based on resource metrics, primarily CPU or memory utilization, ensuring optimal resource allocation and performance for applications with predictable workloads.

The Custom Autoscaler scales the number of nodes depending on the resource demands of pods within the cluster. This methodology is specifically useful when the load is dynamic or scenarios which requiring high availability.

Default controllers of Kubernetes are standard which handles only common task. Whereas custom controllers are designed in a special way to handle specific requirements and particular jobs. They are found to be very precise which addresses unique requirements. Custom Autoscaler in Kubernetes allows users to create custom rules and policies for handling resources. These Autoscalers manage resources to satisfy the user requirements.

A. Advantages of a Custom Auto Scaling Controller over Default Controller:

Custom controller allows users to create specific rules and policies for scaling the applications based on specific needs. It permits to use advanced algorithms and integration with external systems for metrics. This leads to improved resource utilization and allocation along with reduced cost consumption and improved performance.

B. Implementation of Custom Controller Algorithm

During the Initial phase of the project, The application used i.e Nginx application, was deployed as a microservice onto the kubernetes cluster. This deployment process resulted in successfully launching the Nginx application pods. Further, a service was configured to facilitate the access of the application pods via web browser. Locust was utilized to evaluate the system's performance under varying loads such as scalability and responsiveness of the application. The aim was to examine how the Nginx application pods were handling different demands. This deployment and testing served as framework to integrate a unique controller algorithm. The custom autoscaler is expected to enhance the availability of the system and optimize resource utilization by contributing to the overall efficiency and reliability.

The main focus is implementation of custom scheduler, meticulously crafted in python. The aim is to enhance the orchestration of Nginx pods. The algorithm is encapsulated within the docker image. To integrate a dockerfile was meticulously made to execute the python code. The intricacies of the scheduling process was then managed by creating a dedicated pod within the kubernetes environment.

Having a clear understanding of how kubernetes works and its various components is vital. To design a custom scheduler algorithm it is of utmost importance to realize the needs of performance metrics and latency.Conducting a thorough testing to ensure that the custom scheduler works as expected and come up with a meaningful benchmarks and metrics by comparing the custom scheduler with Kubernetes Horizontal Pod Autoscaler (HPA). Further integrate the kubernetes cluster with monitoring tools to capture performance metrics for analysis.

During the testing phase, locust was used to generate the required dynamic load to adjust the number of Nginx pod deployments. The adjustment was computed by the custom algorithm. This methodology enabled the successful integration of the custom scheduler within kubernetes environment. It is crucial to have a smart custom controller algorithm for adjusting pods for dynamic workloads.

When the current CPU usage value is greater than target CPU usage value the code snippet in figure 1 calculates the number of pods to be scheduled. However when the target CPU usage value is greater than current CPU usage value, the code snippet in the figure 2 calculates the number of pods to be reduced or removed when the load drastically decreases.

```
scaling_factor = (cpu_usage_m - target_cpu_value) / 100.0
predicted_pods = round(max_pods - (max_pods - min_pods) * scaling_factor)
predicted_pods = max(min_pods, min(predicted_pods,max_pods))

//cpu_usage_m - current cpu usage value
//target_cpu_value - target cpu usage value
//max_pods - maximum number of pods
//min_pods - minimum number of pods
```

Fig 1. Algorithm to calculate the pods to be scheduled

```
scaling_factor = (cpu_usage_m - target_cpu_value) / 100.0
predicted_pods = round(max_pods - (max_pods - min_pods) * scaling_factor)
predicted_pods = min(min_pods, min(predicted_pods,max_pods))

//cpu_usage_m - current cpu usage value
//target_cpu_value - target cpu usage value
//max_pods - maximum number of pods
//min_pods - minimum number of pods
```

Fig 2. Logic to calculate the pods to be discarded

In order to gather the live CPU usage values form the Nginx pods, the custom controller makes use of Kubernetes Metric Server. The metrics are vital for the calculations of custom algorithm which allows it to dynamically schedule pods on varying workload. This approach when integrated with the kubernetes orchestration environment leads to efficient resource allocation and better performance.

C. Experiment and Testing

The dynamic pod allocation was assessed through the experiment, using locust tool under two different workloads. In the beginning the test was conducted using 3000 users generating 600 requests per second. Similarly, the second test was performed with 200 users and 50 RPS. During the first test observations were recorded on the creation and deletion of pods during the load. The data is systematically organized into tabular format, recording the number of pods created during three time intervals: 2 minutes, 8 minutes and 15 minutes.

The same kind of experiment was conducted in the next workload scenario with 200 users and 50 RPS. The data of number of pods scheduled during three time intervals: 2 minutes, 8 minutes and 15 minutes was recorded. These observations provide the complete understanding of the resource management and allocation, to optimize the microservice deployment.

Under identical loads, the performance of the custom autoscaler was compared with that of the Kubernetes Horizontal Pod Autoscaler. The test was conducted with locust tool with 200 users generating 50 requests per second (RPS) and 3000 users generating 600 RPS. The load was then reduced to 200 from 3000 and the behavioral result was documented. The number of pods created and deleted were examined carefully under different load condition. This comparative analysis of the pods gives insights on relative performance and dynamic scheduling compared to default Kubernetes HPA.

TABLE 1. NUMBER OF PODS SCHEDULED WHEN LOAD IS 3000 USERS AND 600 RPS.

Algorithm Used(for 3000 users, 600 rps)	***KHPA***	***Custom Scheduler***
2 minutes	9	2
8 minutes	10	3
15 minutes	10	5

TABLE 2. NUMBER OF PODS SCHEDULED WHEN LOAD IS 200 USERS AND 50 RPS

Algorithm Used(for 200 users, 50 rps)	***KHPA***	***Custom Scheduler***
2 minutes	10	2
8 minutes	5	1
15 minutes	1	1

V. RESULTS AND DISCUSSION

Analysis of the data of pod scheduling between custom scheduler and KHPA reveals the distinct patterns. The experimental observation is that custom scheduler scheduled less number of pods compared to KHPA, as shown in the Table 1 and Table 2. This is more prominent during high system loads which shows aggressive scaling of KHPA.

After analyzing the response times of KHPA and custom scheduler under different workload conditions, It is clearly observed that custom scheduler exhibited a faster pod adjustment when there was drastic change in the workload. It efficiently reduced the number of pods in comparison with KHPA. This difference in performance is significant which showcases the ability of custom scheduler to effectively manage and allocate resources for optimized resource consumption and cost management.

The inquiry provided insights on how efficiently resource utilization is performed by both the algorithms. KHPA allocated more pods during the experiment conducted, leading to increased resource consumption. Whereas, custom scheduler allocated less number of pods which showcases effective technique for resource management. This observation emphasizes the intricate balance between custom scheduler algorithm and KHPA concerning resource consumption, responsiveness and scalability of the microservice deployment.

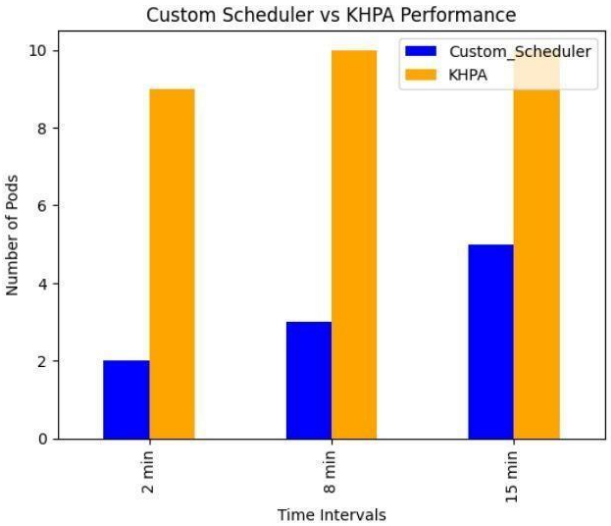

Fig 3. Comparison between Custom Scheduler and KHPA during 3000 users and 600 rps

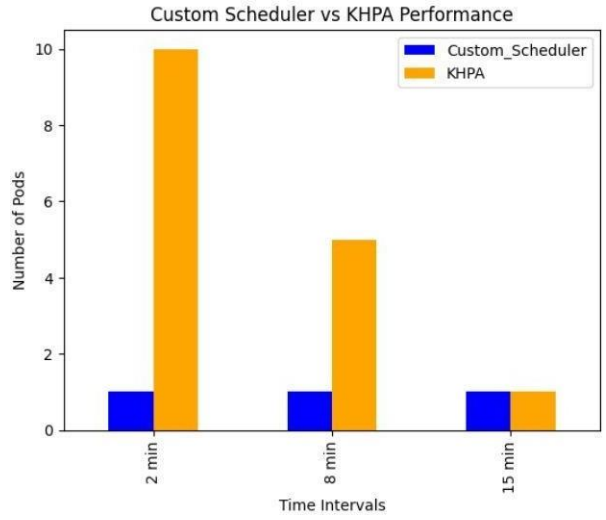

Fig 4. Comparison between Custom Scheduler and KHPA during 200 users and 50 rps

VI. CONCLUSION AND FUTURE WORK

The custom controller auto-scaler when it came to dynamically scaling scenarios outperformed against the Kubernetes Horizontal Pod Autoscaler (KHPA). The custom controller was very sensitive and quickly modified the total number of container instances in response to the varying system loads. The custom controller demonstrated a more responsive behavior during varying loads, especially when there was decrease in the loads effectively reducing pod counts in contrast to a very sluggish adaptation seen in KHPA.

The custom controller demonstrated a clever approach in addition to approach the resource management under high workloads. There were fewer pods in the custom controller when compared with KHPA as it takes a more conservative approach. This in turn results in increased resource use, thus demonstrating its ability to strike a balance between performance and efficiency, which makes it a reliable option for microservice deployment in Kubernetes environments.

There are multiple directions that can be investigated when looking into future work to improve the system further more. First off, adding extra metrics like memory utilization or response time beyond CPU usage can offer more sophisticated management over microservices scaling. Proactive resource allocation can also be improved by adding predictive scaling, which uses machine learning on past data to predict future loads. Finally, deployment procedures can be automated and continuous performance feedback can be obtained for continual enhancements by linking the custom scheduler with CI/CD pipelines. The aforementioned future directions are intended to enhance the custom controller auto-scaler's scalability, efficiency, and adaptability in dynamic microservices contexts.

ACKNOWLEDGEMENT

We would like to express our gratitude to Prof. Venkatesh Prasad, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE20CS390B - Capstone Project.

We are grateful to the project coordinator, Dr. Priyanka H., all the panel members & the supporting staff for organizing, managing, and helping the entire process.

We take this opportunity to thank Dr. Mamatha H R Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support we have received from her.

We are grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, Dr. B.K. Keshavan, Dean of Faculty, PES University for providing us various opportunities and enlightenment during every step of the way.

Finally, this project could not have been completed without the continual support and encouragement we have received from our family and friends.

REFERENCES

1. Deshpande, Neha (2021) Autoscaling Cloud-Native Applications using Custom Controller of Kubernetes. https://norma.ncirl.ie/5089/ Masters thesis, Dublin, National College of Ireland.
2. Shitole, Abishek Sanjay (2022) Dynamic Load Balancing of Microservices in Kubernetes Clusters using Service Mesh. https://norma.ncirl.ie/5943/ Masters thesis, Dublin, National College of Ireland.
3. An Efficient and Scalable Traffic Load Balancing Based on Web Server Container Resource Utilization using Kubernetes Cluster. (May – 2022) Ashok L Pomnar, AVCOE Sangamner 422 605, Maharashtra, India. https://ijisrt.com/assets/upload/files/IJISRT22MAY1644_(1)_(1).pdf
4. V. Sharma, "Managing Multi-Cloud Deployments on Kubernetes with Istio, Prometheus and Grafana," 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, https://doi.org/10.1109/ICACCS54159.2022.9785124