EVENT DRIVEN KUBERNETES AUTOSCALER FOR CLOUD NATIVE APPS

Prof. Bhagwati Galande[1] , Aayush Dharmadhikari[2] , Neel Dhotre[3] , Atharv Papat[4] , Arnavi Shelar[5] Professor, Department of Information Technology [1]
Students, Department of Information Technology [2,3,4,5]
Smt. Kashibai Navale Engineering, Pune, Maharashtra, India

ABSTRACT

Kubernetes and Docker have emerged to be a key solution for managing and organizing containers in microservice applications. With autoscaling, one of Kubernetes key features that dynamically modifies resources like CPU and memory to fit changing demands, it gives us a strong and efficient way to manage our application. Yet, the conventional autoscaling methods, such as the Horizontal Pod Autoscaler (HPA), often have difficulties on customary metrics, like CPU and memory utilization, which may not be fully suitable with the constant changes in the modern applications.

This project aims at solving this problem by suggesting a unique solution which is Event Driven Kubernetes Autoscaler for Cloud Native Apps that is specifically made to work with the scaling needs of the application based on the message queue traffic. The custom scaling algorithm is an important part of the project, which unlike the previous traditional autoscalers who needed predefined metrics and did not provide much flexibility, the custom Metric based Autoscaler allows developers to select their own metrics and scaling parameters, thus providing flexibility and great user experience as they can meet the performance needs of all the applications.

Keywords: *Kubernetes, Docker, Microservices, Autoscaling, Cloud-Native, Horizontal Pod Autoscaler (HPA)*

I. INTRODUCTION

This Project mainly focused on developing a Custom Metric Scaler for Kubernetes where a system is used to manage and scale different applications. Instead of using the traditional way, the numbers of services in our applications are automatically customized by Kubernetes. This system will keep an eye on how busy our applications are in real time, especially by monitoring the traffic going through a message queue. Specific things are only monitored, like message traffic and it will also have control to automatically decide when more replicas of application should be created or when they should be reduced.

When servers are busy they will make sure that the application have enough computing power. It will work like cohesive with the traditional way for scaling Kubernetes, it will also support for managing traffic for messages.

This project aims to enhance Kubernetes autoscaling capabilities and provide us with more ability to control over scaling behavior for all various applications scenarios. Likewise, our project is dedicated to user friendliness. It assigns a setup where we can modify how our application can scale to match our exact requirement. We get to decide how the Autoscaler responds to certain metrics and thresholds. It's not just conceptual, it's about solving real problems.

This main objective is to make smarter, use resources more systematically, and be highly responsive. By customizing how we can scale our application based on message traffic in the queue, we're verifying that we make most of Kubernetes in Cloud-Native environment. This brings us to the future of dynamic resources management in the world of modern applications.

II. RELATED WORK

Kubernetes is one of the most suited options for resource allocation in cloud-native applications to improve QoS. Al-Dhuraibi et al., the author proposed a technique for allocating enough resources to satisfy the growing demands of cloud-native applications. Autonomic Cloud Computing resource scaling is the technique, which utilizes a constant value for a collection of resource-level measures including CPU usage. The workload can be categorized as light or heavy depending on whether the resource need exceeds the predetermined value. Only Virtual Machines are supported by this resource scaling architecture. Complex cloud-native apps, on the other hand, cannot be created on tiny servers or workstations with low capabilities. In the Elastic Docker framework was offered by the researchers as the first device to dynamically provide vertical elasticity to docker containers. Their strategy is based on IBM's MAPE-K guidelines (Monitor, Analyze, Planning,Execute,Knowledge). Elastic Docker adjusts the amount of CPU and RAM allotted to each container based on the application workload. As vertical elasticity is restricted by the machine's capacity, Elastic Docker executes live container migration when there are insufficient resources. Their quoted method surpasses Kubernetes Elasticity by 37.63%. Their solution, however, is reliant on particular functions, such as Linux's CRIU (Checkpoint Restore In User space) capability. The proposed method is not adaptable to all operating systems[3]

In auto-scaling, resources are dynamically provisioned based on changes in applications or environments. This is particularly useful during periods of high workload. There are several challenges in autoscaling, and the researcher proposes a taxonomy for web applications. An autoscaling survey has been proposed by Thai et al. , the survey covered scaling approaches, resource estimation, threshold-based metrics, and container-based multi-tier application scaling and autoscaling. For building Bag-Of-Task applications in the public cloud, the Verma and Bala presents a resource optimization taxonomy 5 and survey. Rossi et al. In their research, the author looked at autoscaling approaches, including the building of auto-scaling algorithms. As per their findings, Kubernetes Horizontal Pod autoscaling functions as a threshold-based reactive controller that adapts the necessary resources by an application via a closed-loop. Jindal et al. The possibilities of Kubernetes autoscaling were exhibited, and the researchers found that the autoscaling approach did not produce a higher performance based on CPU and memory use. They did not, however, explore the influence of other indicators on microservices based applications under severe demand[4]

III. EXISTING SYSTEM

The Kubernetes Horizontal Pod Autoscaler (HPA) is an integrated feature that automatically modifies the number of pod replicas in response to CPU or memory utilization changes. This inbuilt functionality allows applications to horizontally scale, ensuring the optimal number of pod replicas according to resource demands. However HPA has limitations, including a lack of flexibility in scaling thresholds, leading to potential inefficient resource utilization. Moreover, it doesn't provide a cluster-wide perspective, limiting its ability to make global scaling decisions. To overcome these constraints, developers may need to implement custom autoscaling solutions to adjust to their application's specific requirements.

IV. PROPOSED SYSTEM

We would therefore develop a customized metric scaler for Kubernetes which autonomously scales workload replica in accordance with user specified metrics drawn from message queue traffic. Unlike the original Kubernetes Horizontal Pod Autoscaler (HPA) that relies more on CPU utilization and memory allocation, this customized solution takes a different approach. On the contrary, custom scalers provide flexibility in tailoring autoscaling approaches that align with distinct message queue patterns, hence observing the traffic we receive on an app and scaling up or down as appropriate. It makes application response optimized, because it presents finer and flexible approach to scaling compared to conventional HPA. The figure below is a flowchart showing how the app can function simply as a whole.

A. *Flowchart*

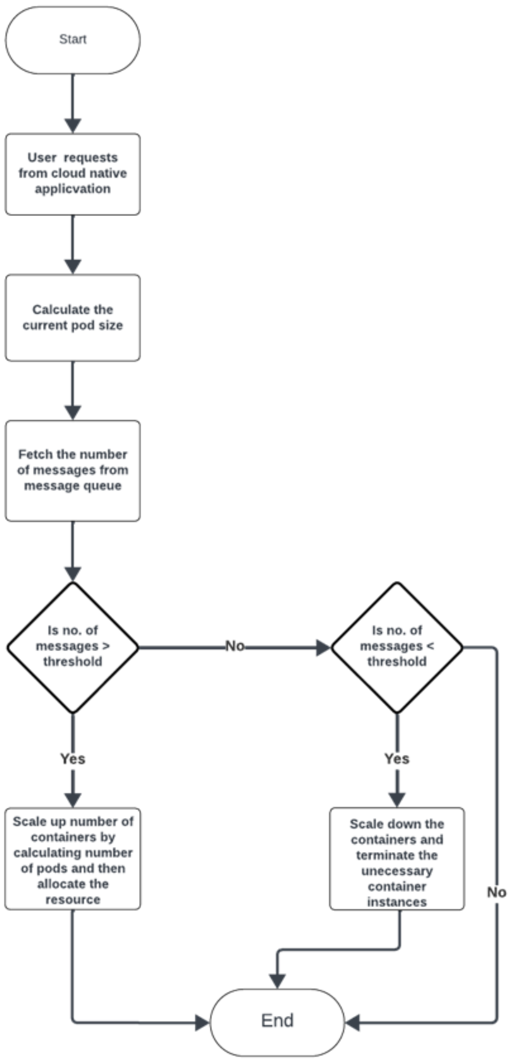
Start
User requests from cloud native applicvation
Calculate the current pod size
Fetch the number of messages from message queue
Is no. of messages > threshold
No
Is no. of messages < threshold
Yes
Yes
Scale up number of containers by calculating number of pods and then allocate the resource
Scale down the containers and terminate the unecessary container instances
No
End

Figure 3. Flowchart of the Custom Metric Scaler

B. *Architecture design*

The system diagram is essentially a system architecture diagram outlining the various modules, parts and their interactions to illustrate what the system contains and how it works. This system uses message- queue to evaluate what the current system/server requirements are for the respective services. Down below is a basic view of the system architecture followed by an explanation for its components:

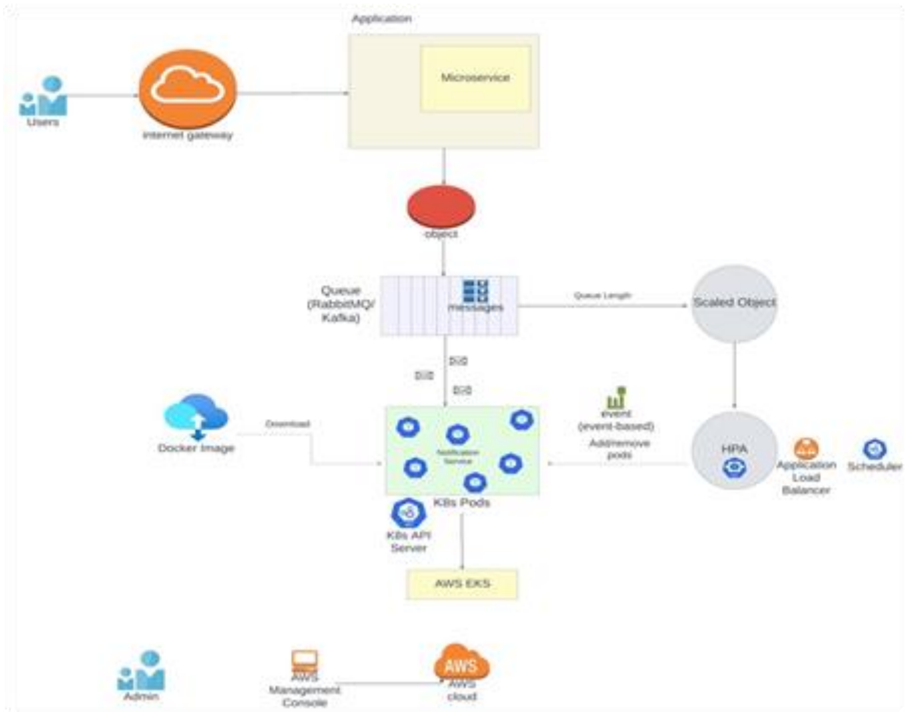

Figure 4. Architecture design

● **User**: This is the end service user, this is not our client, this is the application user. The application/service is provided to several users. Hence, the number of actions occurring on the server through each application/service is tracked along with the type of action performed by each user.

● **Application**: This entity may vary between a standalone application to a webservice which is necessarily a cloud native app or service. This will track the user interactions and send them to the Queue as messages to evaluate further system requirements.

● **Queue**: This is the temporary input-storing unit it works on RabbitMQ which is an open-sourced message broker. It stores messages from the application in a queue and sends it sequentially to the Horizontal Pod Autoscaler (HPA).

● **Horizontal Pod Autoscaler (HPA):** This is where processing and calculations of immediate as well as further requirements of the system (i.e. server-scaling requirements). The HPA takes into account the message queue and issues further orders to the Load-Balancer. The load balancer checks and confirms the requests and assigns synchronizing time to the Kubernetes pods via Scheduler. Scheduler stores a timetable for all the requests and issues time slots for newer requests.

● **Kubernetes Pods:** Contain the copies or images of the services provided by the application. The number of these pods vary as per the commands by the HPA, which is in turn controlled by the current number of users using the app.

● **Amazon AWS:** Cloud service platform provider. Choice of cloud service provider used for hosting the application on.

● **Admin:** Backend user the original owner of the application who can alter the parameters of the service provided by the application

C. *User Interface*

User Interface plays an essential role in making a user-friendly experience for users. This interface is designed so that one can easily communicate on the part of the autoscaling system with users. The solution has an integrated dashboard which provides live status and efficacy reports on how the apps are performing as well as crucial KPIs. Users are able to adjust auto scaling policies through the User Interface, set thresholds and configure tailored scaling behaviors considering the peculiarities of their applications. UI makes it possible to look-up scaling incidents to examine history and use this knowledge to improve future optimizations. With such a focus on simplicity and effectiveness, the UIs are vital for users to properly tune the autoscaling behavior in their cloud-native apps inside the Kubernetes ecosystem.

D. *Proposed Specification of System*

When microservices are containerized, they can be restarted after failing or when they are updated. An orchestration tool named Kubernetes is used in this research to manage containers' operations and deployments. Containers inside the cluster are also managed by this. Three VMs are used to run a Kubernetes cluster on the AWS cloud. In this experiment, Network Time Protocol has been used to achieve asynchronous communication between the nodes in the cluster. The following table 1 and table 2 shows the hardware specifications for Kubernetes cluster and Virtual machine respectively

Kubernetes Cluster	
Instance	AWS / PC
Operating System	Ubuntu
Orchestration Tool	Kubernetes
Container Engine	Docker

Table 1: Required configuration for the cluster

.

Virtual Machine	
vCPU	8
Memory	32 GiB
Network Performance	Upto 3 Mbps

Table 2: Required configuration for the Virtual machine

V. LIMITATIONS

- Kubernetes HPA Limitations:
- HPA cannot be co-used with the VPA based on the CPU and Memory metrics' VPA can just expand based upon CPU and memory measures thus in case VPA is switched on, HPA needs to utilize one or more custom metrics with regards to avoiding scalability problems arising between VPA. Cloud providers have custom metrics adapters that HPA can utilize with the help of custom metrics.
- HPA is only viable for stateless applications, which are able to execute several instances simultaneously. HPA can also be applied along with stateful sets dependent on replica pods. However, it is not applicable for those applications where the use of Kubernetes HPA is impossible.
- The application becomes at risk for slow downs or outages because HPA (and VPA) doesn't take into consideration the I/Os, networks, and storage in their calculations.
- However, HPA does not remove the administrative load from its users when it comes to recognizing waste in a Kubernetes cluster made because of wasted but demanded resources as per the container level. Third party tools powered using machine learning are required for detecting container usage inefficiency, which is ignored by Kubernetes.

VI. CONCLUSION

This project has successfully completed its goal to make cloud-native applications, especially those using message queues, work more efficiently. It has achieved this by giving users more control and flexibility in how they modify their applications. Resource efficiency has always been a challenge in cloud-native systems. With traditional autoscaling frequently causing wastages. This project aims to solve this by making decisions in real-time based on data. This guarantees significant cost savings and a more efficient infrastructure, which is a great success for the project. The core of this project is about flexibility and responsiveness. It departs from the rigid ways of traditional autoscaling and empowers users to define their own rules and metrics , allowing them customize autoscaling to their specific needs. This flexibility is a significant improvement, putting users in charge of how their applications scale. The project has also developed the efficient use of resources, saving costs, and enhanced the performance of real-time systems. This project is a boost, not only in technology but also in making cloud-native applications more user-friendly and flexible to the changing digital world. It sets a new standard and opens the door to a future where applications can modify naturally to changing needs, ensuring that the cloud-native atmosphere remains at the forefront of technology.

VII. REFERENCES

[1] LászlóToka , Member, IEEE, GergelyDobreff, Balázs Fodor, and BalázsSonkoly "Machine Learning-Based Scaling Management for Kubernetes Edge Clusters", IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 18, NO. 1, MARCH 2021.

[2] TengfeiHu;Yannian Wang; (May 2021). "A Kubernetes Autoscaler Based on Pod Replicas Prediction" , 2021 Asia-Pacific Conference on Communications Technology and Computer Science.

[3] Muhammad Abdullah, WaheedIqbal, Josep Ll. Berral, Jorda Polo, David Carrera "Burst-Aware Predictive Autoscaling for Containerized Microservices", 20 May 2020 IEEE.

[4] Muddinagiri, Ruchika; Ambavane, Shubham; Bayas, Simran "SELF-HOSTED KUBERNETES: DEPLOYING DOCKER CONTAINERS LOCALLY WITH MINIKUBE ", 18 August 2020 [IEEE 2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET) - SHEGAON, India.

[5] Chan-Yi Lin, Ting-An Yeh and Jerry Chou Computer Science Department, National TsingHua University, Computer Science Department, Hsinchu Taiwan (R.O.C), Taiwan "DRAGON: A Dynamic Scheduling and Scaling Controller for Managing Distributed Deep Learning Jobs in Kubernetes Cluster", January 2019.

[6] Jay Shah ,DushyantDubaria ,Telecommunication and Network Engineering Southern Methodist University "Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform" 14 March 2019.

[7] Duc-Hung LUONG, Huu-Trung THIEU, Yacine GHAMRI-DOUDANE, Abdelkader OUTTAGARTS Nokia Bell Labs, France , "Predictive Autoscaling Orchestration for Cloud-native Telecom Microservices", 1 November 2018.

[8] EmilianoCasalicchio , Vanessa Perciballi , "Auto-scaling of Containers: the Impact of Relative and Absolute Metrics" - Knowledge Foundation grant n. 20140032, Sweden, 12 October 2017.

[9] Anshul Jindal∗ , Vladimir Podolskiy† and Michael Gerndt Chair of Computer Architecture, Technical University of Munich Garching , Germany, "Multilayered Cloud Applications Autoscaling Performance Estimation" ,November 2017.

[10] Ziad A. Al-Sharif,YaserJararweh, Ahmad Al-Dahou, Luay M. Alawneh "ACCRS: autonomic based cloud computing resource scaling" November 2016 © Springer Science+Business Media New York.