

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

Project Team: Veena Garag PES1UG20CS492

Adarsh Kumar PES2UG20CS016

Suchit S Kallapur PES1UG20CS438

Yuvaraj D C PES1UG20CS521

Project Guide: Prof Venkatesh Prasad, Mr. Vijay R (EOX Vantage)

Department of Computer Science and Engineering,

PES University

**Problem Statement**

**a) The** problem **addressed in the project**.

**b) Provide** a **basic introduction** of the

**project and also an overview of scope** it **entails**.

Project deals with deploying microservices on Kubernetes cluster and reduce the resource consumption such as CPU and bring down the cost of creation of pods.

The project is aimed to improve and optimise resource utilization, improve security feature and be able to monitor the performance and visualize the metrics in order to study the behaviour of the microservices to increased load.

Design Approach / Methods

**a)** Describe the **methodology** / **approach**

**b) Constraints**, **Dependencies and assumptions**

**c) Figure of selected** design

**d) Provide the** model **summary**

The project employed a comprehensive approach to implement the custom controller auto-scaler and achieve seamless scalability for the microservices-based application. The methodology involved the following key steps:

1. Development and Integration: Develop the custom controller auto-scaler using appropriate programming languages and frameworks. Integrate the auto-scaler with the Kubernetes environment and ensure seamless communication with application components.
2. Testing and Validation: Conduct rigorous testing to validate the auto-scaler's functionality and performance. Employ unit tests, and load tests to ensure the auto-scaler can effectively scale the application under various load conditions.
3. Deployment and Monitoring: Deploy the auto-scaler to the production environment and monitor its performance closely. Continuously evaluate the auto-scaler's effectiveness in handling workload fluctuations and maintaining application performance.

Constraints:

Resource Availability: Availability of adequate hardware resources, such as CPU, memory, and storage, can limit the application's scalability.

Dependencies

* Kubernetes: The project relies on Kubernetes as the underlying container orchestration platform for managing microservices deployments.
* Istio Mesh Networks: Istio provides service-to-service communication and visibility, enhancing application observability and control.
* Kailali and Prometheus: Kailali and Prometheus are employed for comprehensive monitoring of microservices, collecting metrics on resource utilization, request latency, and other performance indicators.
* Grafana: Grafana serves as a centralized dashboard for visualizing and analysing metrics collected by Kailali and Prometheus, providing insights into application behaviour and performance.

Summary of Project Outcome

**Write a** summary **of the project outcome whether the initial estimates are the same as that of the results achieved.**

The project successfully implemented a custom controller auto-scaler within a Kubernetes environment to dynamically scale a microservices-based application. This approach will leverage the strengths of both Kubernetes and microservices to achieve seamless scalability and ensure optimal performance under varying load conditions.

The project utilizes Istio mesh networks to enhance service-to-service communication and visibility. Kailali and Prometheus are employed for comprehensive monitoring of services, while Grafana provides a centralized dashboard for visualizing and analysing key metrics.

Background

**Describe the inferences drawn from the Literature Review.**

Traditionally, scaling applications has been a manual and often reactive process, requiring operators to monitor resource utilization and manually adjust the number of application instances. This approach is not only time-consuming and error-prone but also fails to adapt to the rapidly changing demands of modern applications.

**Dataset and Features /** Project Requirements **/** Product Features

**a) Describe the dataset along** with **its features and processing**.

**b) List of** requirements

**c) Product** features.

Here are some of the requirements:

1. Define the microservices architecture and design the application.
2. Choose the right containerization technology and create Docker images for each microservice.
3. Create a Kubernetes cluster and configure it.
4. Define the Kubernetes deployment and service manifests for each microservice.
5. Configure the Kubernetes ingress controller to expose the microservices to the outside world.
6. Implement Kubernetes resource quotas and limits to manage resource consumption.
7. Implement Kubernetes security features such as network policies, secrets, and role-based access control (RBAC).
8. Implement Kubernetes monitoring and logging using tools such as Prometheus, Grafana.

Results **and Discussion**

**a) Brief explanation on testing and evaluation activities with images**.

b) **Results obtained** can be displayed in **graphs**/**charts/tabular**.

The project successfully implemented a custom controller auto-scaler, enabling dynamic scaling of the microservices-based application. Through a comparative analysis of the Horizontal Pod Auto-scaler (HPA) algorithm and our custom controller, our algorithm demonstrated superior performance compared to the default algorithm. This enhanced efficiency resulted in improved resource utilization and application responsiveness.

**Conclusions and Future Work**

**a)** Summarize the **key points**.

**b) Provide** a **glimpse of** Future **work**.

Future work that can be done on our custom scheduler for auto-scaling based on CPU usage data from a matrix server includes:

* Support for other metrics. In addition to CPU usage, you could also consider supporting other metrics, such as memory usage, network traffic, or response time. This would give you more granular control over how your scheduler scales your microservices.
* Predictive scaling. Instead of simply reacting to current CPU usage, you could use predictive scaling to anticipate future load and scale your microservices accordingly. This could help to avoid performance bottlenecks and improve overall scalability. This could be done using machine learning to train a model on historical CPU usage data.
* Integrate with CI/CD pipelines. This would allow you to automatically deploy your custom scheduler to your Kubernetes cluster. You could also use CI/CD pipelines to collect feedback on the performance of your scheduler and to make improvements over time.

Integrate with Kubernetes monitoring tools. This would allow you to get real-time information about the CPU usage of your microservices. You could use this information to trigger scaling actions, or to provide feedback to your predictive scaling algorithm.

**References**

* Deshpande, Neha (2021) Autoscaling Cloud-Native Applications using Custom Controller of Kubernetes. https://norma.ncirl.ie/5089/

Masters thesis, Dublin, National College of Ireland.

* Shitole, Abhishek 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.

* 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>

* 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>

**Photographs of team members (students) and Guide with names**.

