**ABSTRACT**

The software engineering industry is rapidly embracing containers to deploy microservice-based cloud-native services. When an application has massive workload, Kubernetes can automatically scale the microservices using the default Kubernetes Horizontal Pod Autoscaler. The default Kubernetes approach results in inefficient resource allocation, which affects cloud-native application performance and increases maintenance expenses. This project uses a custom controller algorithm and estimates the appropriate number of instances for containers. The suggested approach maintains cloud-native applications' Quality of Service (QoS) while cutting down on maintenance expenses. This project found that on comparison the default Kubernetes Horizontal Pod Autoscaler is more expensive and scales up and down very inefficiently and is more expensive than the custom controller. Project deals with deploying microservices on cloud and reduce the resource consumption like CPUs, memory, and bring down the cost of creation of pods.

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

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Title** | **Page No.** |
|  | **INTRODUCTION** | **1** |
|  | **PROBLEM STATEMENT** | **2** |
|  | **ABSTRACT AND SCOPE** | **2** |
|  | **LITERATURE SURVEY**   * 1. **Introduction**   2. **Research Paper - 1**   “Autoscaling Cloud-Native Applications using Custom Controller of Kubernetes”   * 1. **Research paper-2**   “Dynamic Load Balancing of Microservices in Kubernetes Clusters using Service Mesh (2022)”   * 1. **Research Paper-3**   “An Efficient and Scalable Traffic Load Balancing Based on Webserver Container Resource Utilization using Kubernetes Cluster (2022)”   * 1. **Research Paper-4**   “Managing Multi-Cloud Deployments on Kubernetes with Istio, Prometheus and Grafana (2022)”   * 1. **Summary of Literature Survey**   **Conclusion** | **2**  **2-4**  **4-7**  **7-9**  **9-10**  **10**  **10** |
|  | **DESIGN APPROACH** | **11** |
|  | **PROPOSED METHODOLOGY**   * 1. **Methodology for microservice deployment on Kubernetes**   2. **The methodology for load balancing and scalability**   **Benefits and Drawbacks of the proposed methodology** | **11-13**  **11**  **12**  **13** |
|  | **ARCHITECTURE**   * 1. **Design pattern used**   2. **Istio service mesh architecture**   3. **Novelty**   4. **Deployment strategies**   5. **CI/CD Pipeline to Deploy Microservices on Kubernetes**   6. **EOX vantage logical architecture of application and services**   **Conclusion** | **13-19**  **13**  **14**  **17**  **17**  **18**  **19**  **19** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| 1 | Kubernetes Cluster | 3 |
| 2 | custom control algorithm | 4 |
| 3 | Cost Comparison between custom controller Vs. KHPA | 4 |
| 4 | An example of a High-Level Design of proposed model | 6 |
| 5 | Traffic Visualization through Kiali Software | 7 |
| 6 | Scenario of Users and Requests per second | 7 |
| 7 | CPU and Memory Utilization in Kubernetes vs Istio | 8 |
| 8 | Terminal view for Number of pod scale after web traffic load increased | 10 |
| 9 | Terminal view for Number of pods that get auto deployed after load increased | 10 |
| 10 | Istio mesh spanning multiple Kubernetes clusters with direct network access to remote pods over VPN | 12 |
| 11 | Structure of the Kubernetes Master Node and the Kubernetes slave node | 16 |
| 12 | Kubernetes Control Plane with introduction of custom scheduler | 17 |
| 13 | Sidecar Pattern | 18 |
| 14 | Sidecar Pattern with Microservices | 18 |
| 15 | Istio service mesh | 19 |
| 16 | Istio Service Mesh Architecture | 21 |
| 17 | prototype/ product-based approach | 22 |
| 18 | final deployment architecture | 23 |
| 19 | Flowchart working of the algorithm | 24 |
| 20 | Fetching Metrics Values | 27 |
| 21 | Scaling Factor of Algorithm | 28 |
| 22 | Creation of pods when scaling up | 28 |
| 23 | Deletion of pods when scaling down | 29 |
| 24 | Demo Application running on live server | 30 |
| 25 | Load Generation using Locust | 31 |
| 26 | Pods at 2nd min frame | 32 |
| 27 | Pods at 8th min frame | 32 |
| 28 | Pods at 15th min frame | 32 |
| 29 | Decreasing Load | 33 |
| 30 | Pods after decreasing load | 33 |
| 31 | Load generation using Locust | 34 |
| 32 | Pods at 2nd min frame | 34 |
| 33 | Pods at 8th min frame | 35 |
| 34 | Pods at 15 min frame | 35 |
| 35 | Load Fluctuation in the Locust Testing | 35 |
| 36 | Pods after decreasing load | 36 |