Dissertation on

"Deploying EOX microservice to Kubernetes cluster"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE20CS390A – Capstone Project Phase - 1

Submitted by:

Yuvaraj D C	**PES1UG20CS521**
Suchit S Kallapur	**PES1UG20CS438**
Adarsh Kumar	**PES2UG20CS016**
Veena Garag	**PES1UG20CS492**

Under the guidance of

Prof. Venkatesh Prasad
Professor
PES University

January - May 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100 Feet Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

“Deploying EOX microservice to Kubernetes cluster”

is a bonafide work carried out by

Yuvaraj D C	**PES1UG20CS521**
Suchit S Kallapur	**PES1UG20CS438**
Adarsh Kumar	**PES2UG20CS016**
Veena Garag	**PES1UG20CS492**

in partial fulfilment for the completion of sixth semester Capstone Project Phase - 1 (UE20CS390A) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2023 – May. 2023. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 6th semester academic requirements in respect of project work.

Signature	Signature	Signature
Prof. Venkatesh Prasad	Dr. Shylaja S S	Dr. B K Keshavan
Professor	Chairperson	Dean of Faculty

External viva

Name of the Examiners	**Signature with Date**
1. ______________________	______________________
2. ______________________	______________________

DECLARATION

We hereby declare that the Capstone Project Phase - 1 entitled **"Deploying EOX microservice to Kubernetes cluster"** has been carried out by us under the guidance of **Prof Venkatesh Prasad** and submitted in partial fulfilment of the completion of sixth semester of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2023. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES1UG20CS521	**Yuvaraj D C**	
PES1UG20CS438	**Suchit S Kallapur**	
PES2UG20CS016	**Adarsh Kumar**	
PES1UG20CS492	**Veena Garag**	

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 UE20CS390A - Capstone Project Phase – 1.

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. Shylaja S S**, 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.

TABLE OF CONTENT

LIST OF FIGURES

1. INTRODUCTION

Microservices architecture is an approach where an application is divided into smaller, loosely coupled services that can be developed, deployed, and scaled independently. This gives us a lot of advantages like scalability, resilience, agility and coping up with growing technology. These microservices handle specific business functionality and communicate with other microservices via API to serve the request better. They are easy to maintain and deploy. They can be containerized with all the necessary runtime environments and could be run on any platform. This feature is very much useful when it comes to distributed architecture.

2. PROBLEM STATEMENT

Deploy EOX microservices to Kubernetes clusters. Develop architecture for deploying finished containers on Kubernetes. This has to reduce the life cycle of deployment and management of the containers and improve scalability and performance. The microservice can be coded in different languages.

3. ABSTRACT & SCOPE

3.1 Abstract:

To deploy EOX microservices to Kubernetes clusters using autoscaling algorithms and implementing the Istio – Service Mesh Architecture which we propose will improve scalability and performance. The architecture will have the inherent properties of Kubernetes such as autoscaling, self-healing and service discovery and communication advantages of Service Mesh Architecture.

3.2 Scope:

- Microservices architecture allows for independent deployment of services which makes it easier to update and maintain.
- Microservices architecture is more scalable, flexible, and resilient than monolithic architecture.

4. LITERATURE SURVEY

4.1 Research Paper - 1

“Autoscaling Cloud-Native Applications using Custom Controller of Kubernetes”

4.1.1 Introduction

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 Algorithm is more costly than Custom algorithm. Using the Custom Controller nearly lowered the maintenance costs in half. In accordance with workload, Custom Controller manages and scales Pods dynamically. Pods will be scheduled in accordance with the volume of requests made to the server. For this investigation, Amazon Web Services' t3.2xlarge instance was employed. The application based on microservice was developed in PHP to do the computationally demanding tasks.
The following software and hardware were employed in this study's research: Kubernetes, Kubeadm, Kubectl, Docker, and PHP-Apache.

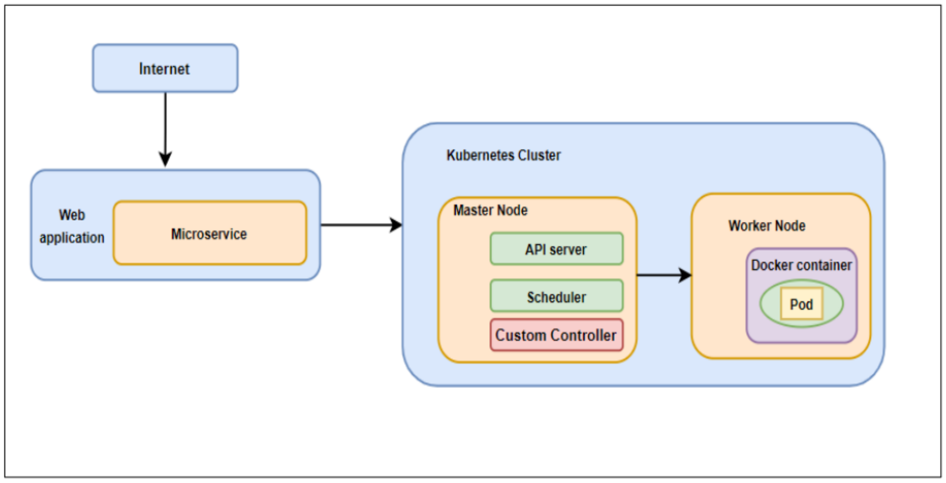
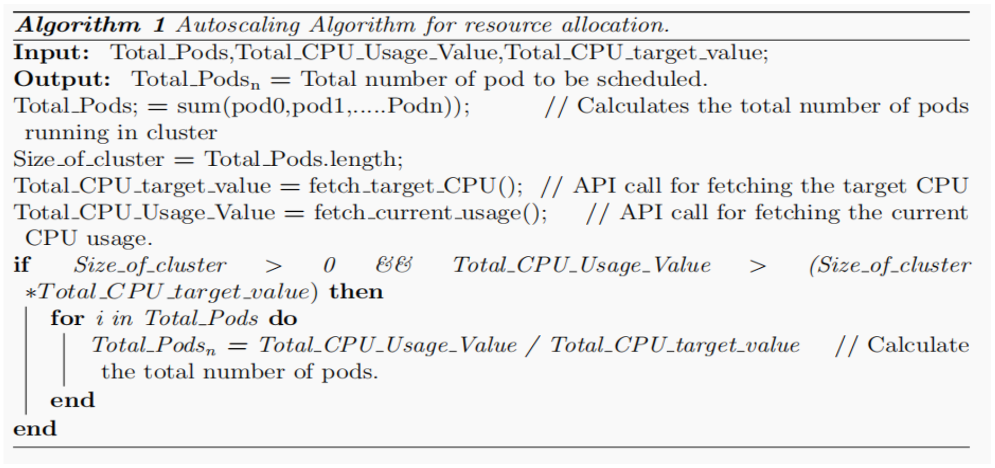


figure 4.2.1: Kubernetes Cluster

4.1.2 Methodology

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.

Given below is the algorithm proposed in the research paper:



```
Algorithm 1 Autoscaling Algorithm for resource allocation.
Input: Total_Pods,Total_CPU_Usage_Value,Total_CPU_target_value;
Output: Total_Pods$_n$ = Total number of pod to be scheduled.
Total_Pods; = sum(pod0,pod1,.....Podn));    // Calculates the total number of pods
  running in cluster
Size_of_cluster = Total_Pods.length;
Total_CPU_target_value = fetch_target_CPU();  // API call for fetching the target CPU
Total_CPU_Usage_Value = fetch_current_usage();  // API call for fetching the current
  CPU usage.
if Size_of_cluster > 0 && Total_CPU_Usage_Value > (Size_of_cluster
  *Total_CPU_target_value) then
    for i in Total_Pods do
        Total_Pods$_n$ = Total_CPU_Usage_Value / Total_CPU_target_value   // Calculate
          the total number of pods.
    end
end
```

figure 4.2.2: custom control algorithm

4.1.3 Experiment

Three trials totalling two minutes, eight minutes, and fifteen minutes were carried out. The performance of the Custom Controller and the default Kubernetes algorithm were compared. The results showed that KHPA allotted 3 pods within the first two minutes of workload while the custom controller allocated 0 pods. In contrast to KHPA, which planned 7 pods after 8 minutes of workload creation, Custom controller only scheduled 2 pods, overprovisioning resources by 4 pods. Performance suffered because KHPA only scheduled 1 pod during the 15 minutes of workload production while Custom Controller scheduled 5.

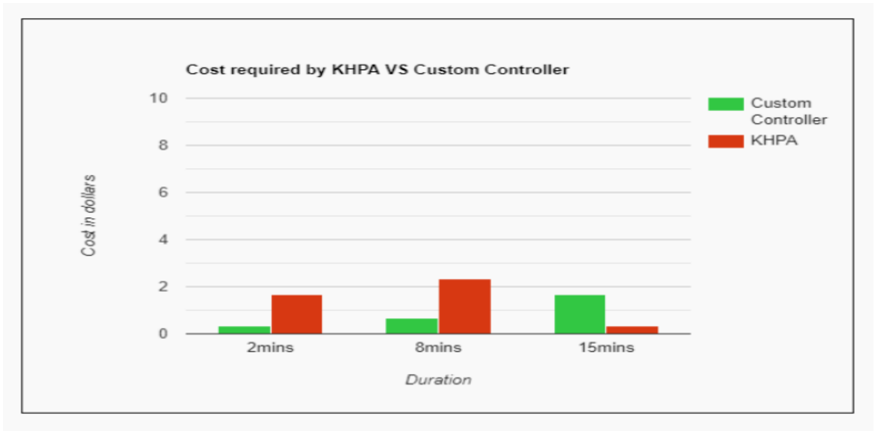


Figure 4.2.3: Cost Comparison between custom controller Vs. KHPA

4.1.4 Conclusion

By implementing a custom controller algorithm which calculates the efficient number of pods, this paper attempts to solve the issue of inefficient pod allocation.

4.1.5 Future Work

The future work suggested by this paper is the above algorithm can be improved for memory and storage intensive microservices. It is found that replicas take time to get stabilized this issue can be considered.

4.2 Research Paper – 2

"Dynamic Load Balancing of Microservices in Kubernetes Clusters using Service Mesh (2022)"

4.2.1 Introduction

This paper suggests a technique that applies service-specific routing across the Istio control plane to inject sidecar proxies onto every micro-service 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.

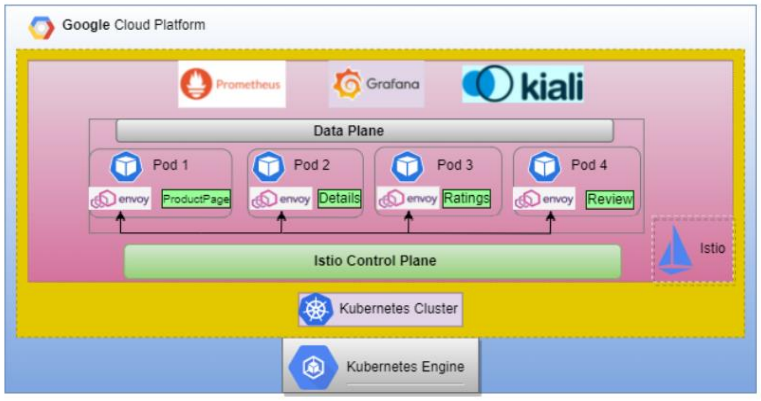


Figure 4.3.1: An example of a High-Level Design of proposed model

4.2.2 Methodology

The accompanying diagram shows the order in which the planned system was implemented. Istio configurations are kept apart using the Istio-system namespace. On the ingress-gateway for this namespace, the network configurations (YAML file) are applied. 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 utilising a control plane. The locust tool causes load on the bookseller application.

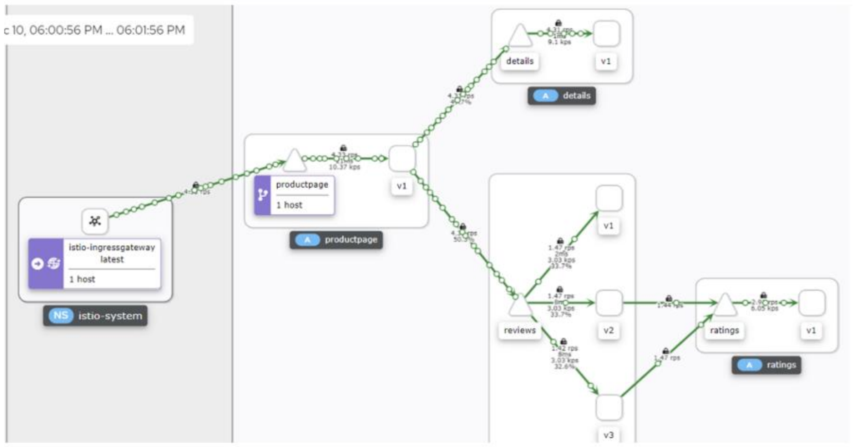
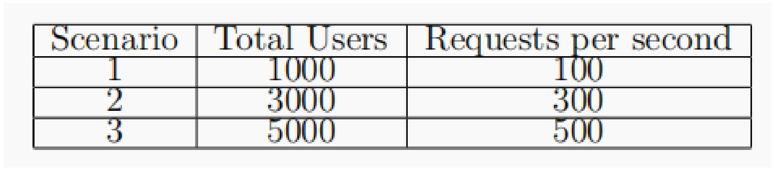


Figure 4.3.2: Traffic Visualization through Kiali Software

4.2.3 Experiment

The studies involved producing load for five minutes under three distinct circumstances (as stated in table below). They simulated user traffic and the number of incoming traffic load per second on the microservices using the load testing tool locust. With the aid of Kiali and Grafana, metrics including response time, CPU usage, and memory usage were monitored and noted.



Scenario	Total Users	Requests per second
1	1000	100
2	3000	300
3	5000	500

igure 4.3.3: Scenario of Users and Requests per second

According to the aforementioned experiments, the response time between the default Kubernetes-based system and the proposed Istio-based system is almost identical. when there are fewer users and incoming requests per second on the application (case 1) for both models. When the traffic load gradually increases (case 2 3), the default Kubernetes system struggles to perform well. But the proposed Istio-based system performs greatly by maintaining stability compared to the Kubernetes default system.

4.2.4 Conclusion

This experiment suggests a service-mesh Istio-based system that effectively manages the various dynamic load balancing on a web application built with a microservices architecture and deployed on a Kubernetes cluster by speeding up response times and using fewer resources than a default Kubernetes system.

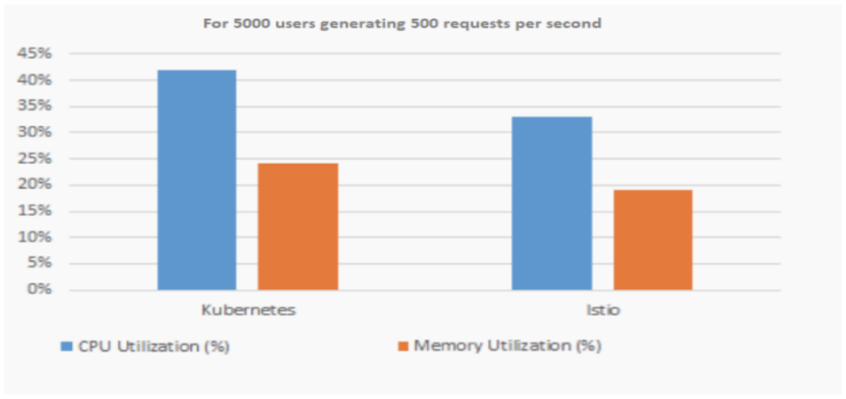


Figure 4.3.4: CPU and Memory Utilization in Kubernetes vs Istio

4.2.5 <u>Future Work</u>

Further research to reduce the latency and complexity can be done

4.3 <u>Research Paper-3</u>

“An Efficient and Scalable Traffic Load Balancing Based on Web Server Container Resource Utilization using Kubernetes Cluster (2022)”

4.3.1 <u>Introduction</u>

Because the microservices that are running on the containers don't communicate with one another, managing the containers is a constant challenge for any business. They operate separately and independently. Here, Kubernetes enters the picture. All that Kubernetes is a platform for managing containers.

4.3.2 <u>Methodology</u>

Horizontal Container Scaling Algorithm:

Desired App Replicas = ceil [current App Replicas* (current Scaling Metric Value / desired Scaling Metric Value)]

4.3.3 <u>Experiment</u>

In this experiment was conducted with a given target CPU usage value of 70%, and a minimum pod count of 4 with a maximum pod count of 20. Current status of deployment is six pods averaging 95% usage.

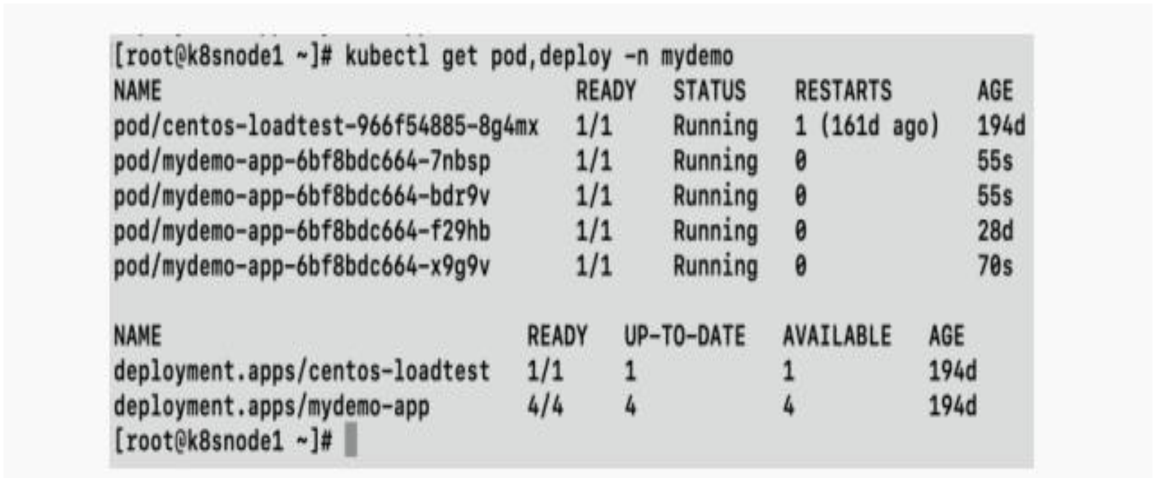
Desired Replicas calculated = ceil [6*(85/70)]

= ceil (8.14) = 9

Scaling down example

The next scenario is with a given target CPU usage value of about 70%, minimum pod count was given as 4, and maximum pod count was given of value 20, current status of application pod deployment is that here are 10 total pods. Given that 10 pods averaging 45% usages. When we use the same algorithm to scale down the pods we get:

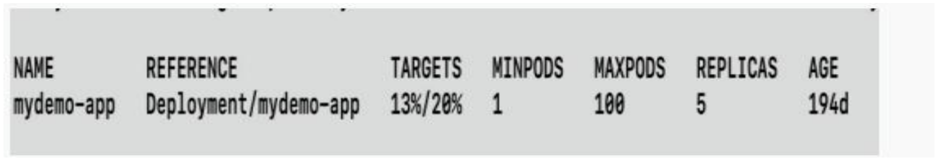
Desired Replicas count = ceil [10*(45/70)] = ceil (6.42) = 7



```
[root@k8snode1 ~]# kubectl get pod,deploy -n mydemo
NAME                                   READY   STATUS    RESTARTS       AGE
pod/centos-loadtest-966f54885-8g4mx    1/1     Running   1 (161d ago)   194d
pod/mydemo-app-6bf8bdc664-7nbsp        1/1     Running   0              55s
pod/mydemo-app-6bf8bdc664-bdr9v        1/1     Running   0              55s
pod/mydemo-app-6bf8bdc664-f29hb        1/1     Running   0              28d
pod/mydemo-app-6bf8bdc664-x9g9v        1/1     Running   0              70s

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/centos-loadtest   1/1     1            1           194d
deployment.apps/mydemo-app        4/4     4            4           194d
[root@k8snode1 ~]#
```

Figure 4.4.3.1: Terminal view for Number of pod scale after web traffic load increased



```
NAME         REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
mydemo-app   Deployment/mydemo-app   13%/20%   1         100       5          194d
```

Figure 4.4.3.2: Terminal view for Number of pods that get auto deployed after load increased

4.3.4 <u>Conclusion</u>

According to user demand and traffic demand, the article proposed a dynamic scaling technique that takes backend delay into account for platforms and apps with high traffic levels.

4.4 Research Paper-4

"Managing Multi-Cloud Deployments on Kubernetes with Istio, Prometheus and Grafana (2022)"

4.4.1 Introduction

Deploying and managing clusters across multi public clouds has a great advantage of harnessing the prominent features provided by each cloud service provider. It gives us the negotiating power and reduces single vendor dependency. When the project requirement is not met by the features provided by a single vendor, in such cases we go for a multi-vendor system or multi public cloud platform. but implementing such architecture becomes difficult.

4.4.2 Implementation

Anthos which is implemented using Istio open-source project implemented by google is a milestone that has made this possible. We can have many clusters on various public cloud providers be it an AWS, google cloud platform and Istio control plane using which we can inject envoy proxies and form a service mesh of interconnected clusters. These clusters can communicate with each other using a VPN tunnel. here Kubernetes plays a major role in scaling, pod allocation and deployment.

Istio is going to take care of traffic management and observability and security issues the implementation makes use of various technologies such as Kubernetes, Istio, Grafana, Prometheus Grafana is a metric analysis and visualization tool along with-it Prometheus is also a very useful tool to monitor and alerting

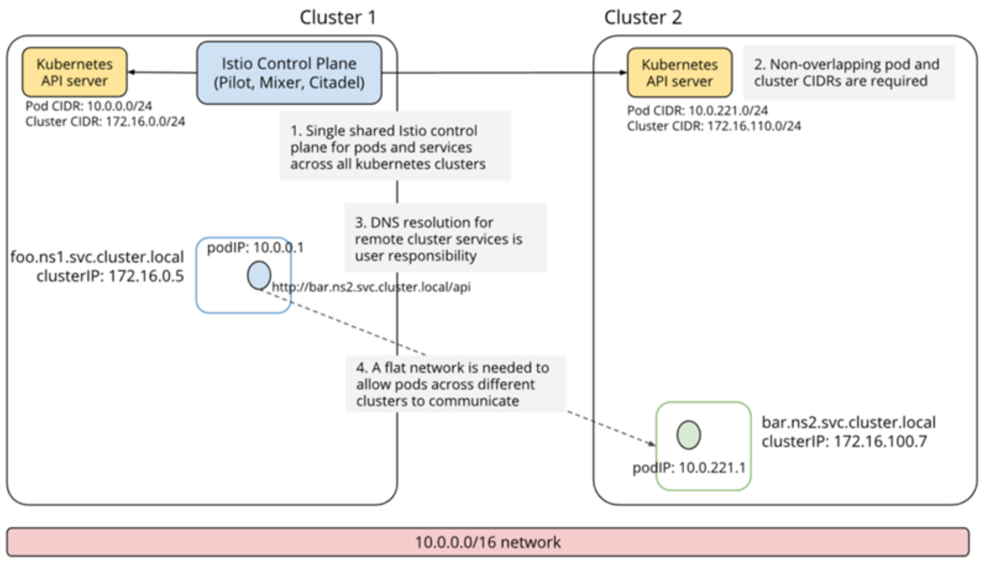

Cluster 1
Cluster 2
Kubernetes API server
Istio Control Plane (Pilot, Mixer, Citadel)
Pod CIDR: 10.0.0.0/24
Cluster CIDR: 172.16.0.0/24
1. Single shared Istio control plane for pods and services across all kubernetes clusters
Kubernetes API server
2. Non-overlapping pod and cluster CIDRs are required
Pod CIDR: 10.0.221.0/24
Cluster CIDR: 172.16.110.0/24
3. DNS resolution for remote cluster services is user responsibility
foo.ns1.svc.cluster.local
clusterIP: 172.16.0.5
podIP: 10.0.0.1
http://bar.ns2.svc.cluster.local/api
4. A flat network is needed to allow pods across different clusters to communicate
bar.ns2.svc.cluster.local
clusterIP: 172.16.100.7
podIP: 10.0.221.1
10.0.0.0/16 network

Figure 4.5.2: Istio mesh spanning multiple Kubernetes clusters with direct network access to remote pods over VPN

4.5 Summary of Literature Survey

After analysing all these papers, we get some insights on how to deploy and manage microservices in multiple public cloud platforms, by making use of various tools like Istio, Grafana, Prometheus etc. We came across various algorithms that talk about efficient load balancing techniques - horizontal container scaling algorithm, scaling up and down algorithms, and dynamic scaling algorithms. Also, we could see that the default Kubernetes algorithm can lead to inefficient usage of resources when the number of requests is high, hence the paper proposes a custom controlled algorithm which actually gave 50% cost reduction.

4.6 Conclusion

Default Kubernetes algorithm can lead to inefficient usage of resources when the number of requests is high, hence the paper proposes a custom controlled algorithm which actually gave 50% cost reduction. Hence, this algorithm can be used for deploying microservice in our project.

5. DESIGN APPROACH

1. Containerize the microservices that are provided by EOX Vantage.
2. Define the Kubernetes deployment and service and configure the Kubernetes ingress to map traffic to different backends based on rules defined via K8 API.
3. Deploying the microservices through Istio Service Mesh Architecture on top of Kubernetes. We propose this will have the Inherent properties of Kubernetes like autoscaling, self-healing, service discovery, load balancing and communication advantages from service mesh architecture.
4. Deploying to a localhost through different methods of deployments such as rolling, blue-green and canary deployment. Monitoring and testing which of the methods provides better results

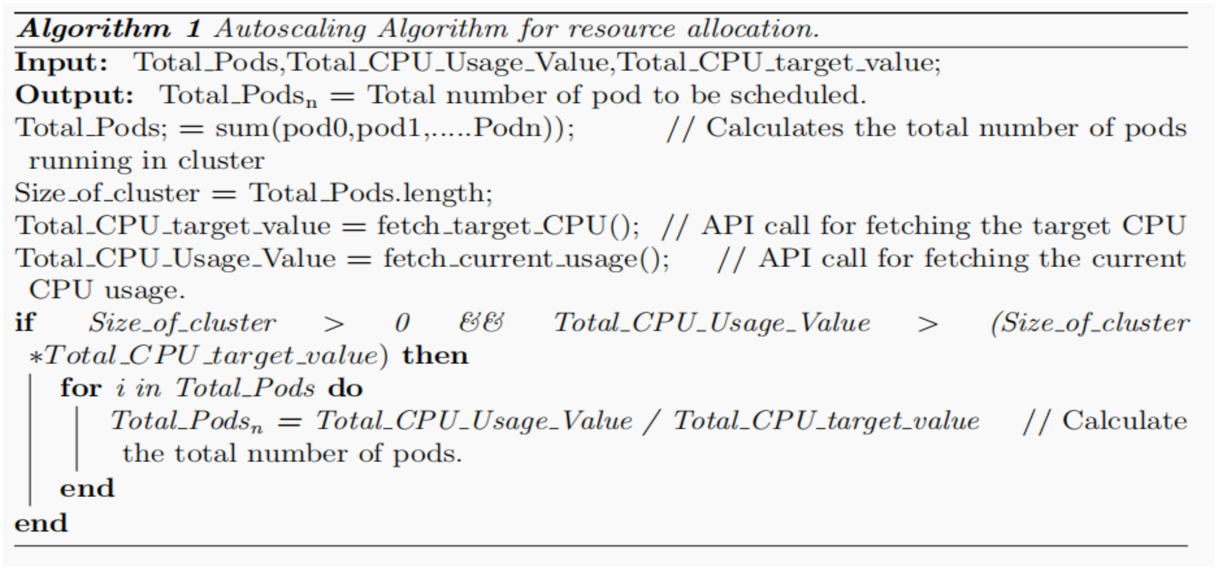
6. PROPOSED METHODOLOGY

6.1 Methodology for microservice deployment on Kubernetes:

To address the challenges of microservice deployment on Kubernetes, we propose a methodology that includes the following steps: design, build, test, deploy, and monitor. During the design phase, the microservices are identified and their dependencies are mapped out. In the build phase, the services are developed and containerized. Testing ensures that each service works correctly and communicates with other services. Deployment involves setting up the Kubernetes cluster and deploying the services. Finally, monitoring ensures that the services are running correctly, and alerts are raised if any issues arise.

6.2 The methodology for load balancing and scalability:

Custom controller algorithm**:**



```
Algorithm 1 Autoscaling Algorithm for resource allocation.
Input: Total_Pods,Total_CPU_Usage_Value,Total_CPU_target_value;
Output: $Total\_Pods_n$ = Total number of pod to be scheduled.
Total_Pods; = sum(pod0,pod1,.....Podn));    // Calculates the total number of pods
  running in cluster
Size_of_cluster = Total_Pods.length;
Total_CPU_target_value = fetch_target_CPU();  // API call for fetching the target CPU
Total_CPU_Usage_Value = fetch_current_usage();    // API call for fetching the current
  CPU usage.
if Size_of_cluster > 0 && Total_CPU_Usage_Value > (Size_of_cluster
  *Total_CPU_target_value) then
    for i in Total_Pods do
        $Total\_Pods_n$ = Total_CPU_Usage_Value / Total_CPU_target_value    // Calculate
          the total number of pods.
    end
end
```

Deshpande, Neha (2021) Autoscaling Cloud-Native Applications using Custom Controller of Kubernetes.

https://norma.ncirl.ie/5089/

Master’s thesis, Dublin, National College of Ireland.

6.2.1 Methodology

In order to prevent resource waste and website crashes, the architecture consists of one master node and two worker nodes. In this paper, an attempt was made to create a custom algorithm to calculate the required number of pods. Kube-controller manager is one of the components. Kubernetes cluster using YAML. The YAML file's parameters are sent to the Kube-API server and then to the Kube-controller manager, where they are used to control replicas. The custom controller is then instructed to start existing pods or generate new ones via the Kube-API server.

6.2.2 Conclusion

This methodology schedules the efficient number of pods and preserves the resources along with maintaining stability.

6.3 Benefits and Drawbacks of the proposed methodology

- Decrease in cost for maintaining Quality of Service.
- Provides supports only CPU intensive microservices.

7. ARCHITECTURE

7.1 Design pattern used:

Consider an enterprise operating system, the application might consist of so many microservices which need to communicate with each other for better serving of requests, it could be business logic, or consistency, scalability, security, handling partial failures etc communication is really important.

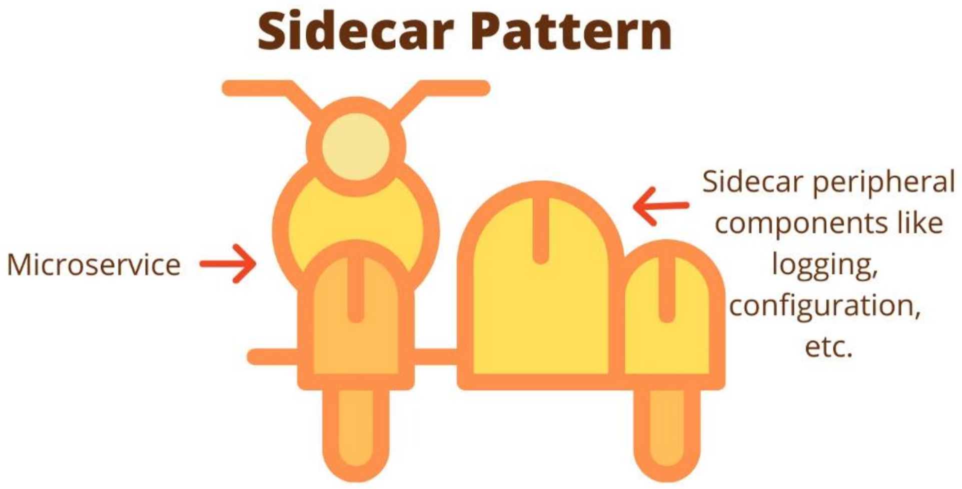

Sidecar Pattern
Microservice
Sidecar peripheral components like logging, configuration, etc.

figure 7.1.1 Sidecar Pattern

We can use a sidecar design pattern, the sidecar which is also called a sidecar/envoy proxy which facilitates the various functionalities like service discovery, traffic management, load balancing, circuit breaking, etc. It is a container that will be deployed along with microservice. The microservice knows nothing about the outside world. Any interaction with the outside will happen through the sidecar proxy. These sidecars are independent of the microservices hence their manipulation and configuring becomes easy and they can be controlled by the service mesh control plane. So, each of the microservice will be containerized and deployed along with a sidecar/envoy proxy.

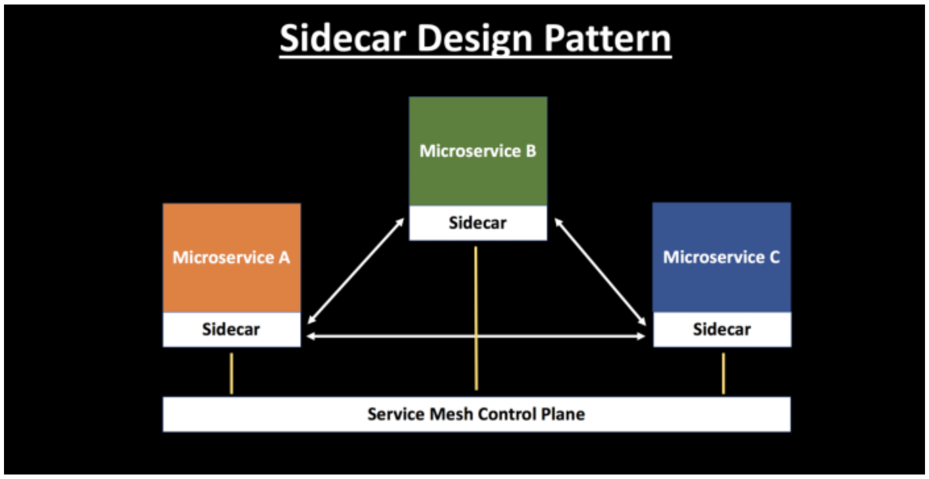


figure 7.1.2 Sidecar Pattern with Microservices

7.2 <u>Istio service mesh architecture:</u>

The design describes the usage of Istio service mesh to inject the sidecar and load balance the requests between the containers from the control plane. Deploying Istio on Kubernetes will involve running a cluster on a dedicated namespace and will depend on Kubernetes for networking and service discovery. Kubernetes does not implement service to service communication therefore Istio service mesh will take care of this including load balancing, traffic management, security and fault tolerance.

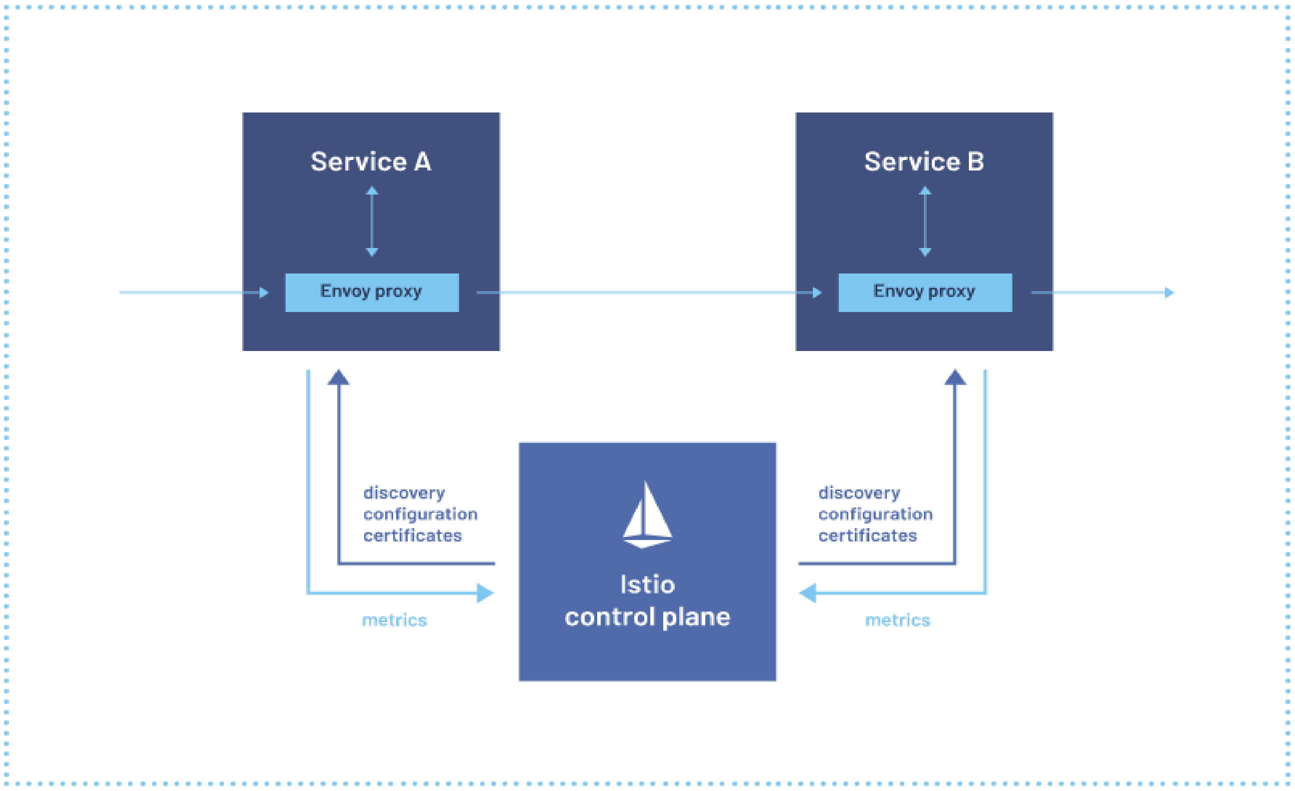


figure 7.2 Istio service mesh

7.2.1 <u>**Features of Istio service mesh:**</u>

Istio is an open-source service mesh platform. It helps manage and secure microservices in a Kubernetes cluster

1. Envoy proxy: Istio deploys a sidecar proxy, the Envoy proxy, alongside each service instance in the cluster. The proxy intercepts all inbound and outbound traffic to the service instance and sends it through the Istio data plane.

2. Istio data plane: The Istio data plane consists of a set of Envoy proxies and a control plane. The Envoy proxies forward requests to other services and implements the various Istio features such as traffic management, security, and observability.

3. Istio control plane: The Istio control plane manages the Envoy proxies and their configuration. It provides a central place to configure and manage the various Istio features, such as routing rules, security policies, and telemetry.

4. Traffic management: Istio can route traffic between services based on various criteria, such as HTTP headers, source IP address, or user identity. It can also implement features like load balancing, circuit breaking, and retries.

5. Security: Istio can enforce mutual TLS authentication between services, implement access control policies, and provide encryption for service-to-service communication.

6. Observability: Istio provides various telemetry features such as request tracing, metrics, and logging. This allows operators to understand how traffic is flowing through the system and to diagnose issues

There is scope for implementing custom controller algorithms to reduce the maintenance cost as the default Kubernetes algorithm leads to improper resource allocation, performance degradation and increase the maintenance cost.

Implementing this at the master node which is responsible for scaling up and down will impact by reducing maintenance cost by 50% as per the literature survey.

The service mesh architecture also provides a path for observability and trace the origin of the problem

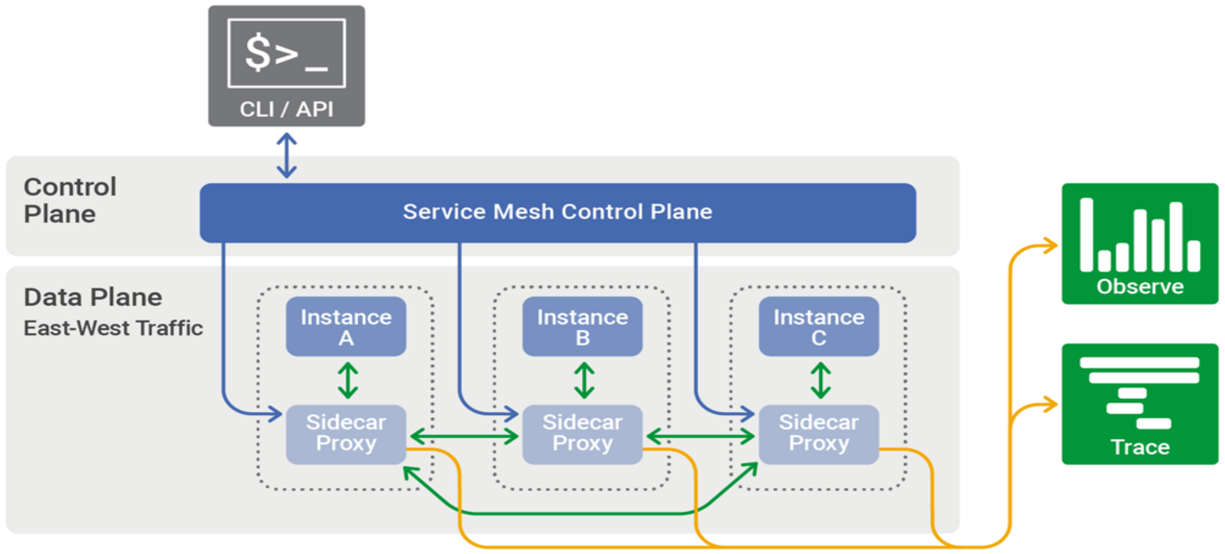


figure 7.2.1: Example of Istio Service Mesh Architecture

7.3 <u>Novelty</u>

Using service-mesh architecture on top of Kubernetes. This will result in addon benefits from both the aspects, Inherent properties of Kubernetes like autoscaling, self-healing, service discovery and load balancing etc and communication advantages from service mesh architecture. We are basically thinking of implementing a service mesh architecture of deployment on Kubernetes that will allow better communication and scalability and performance. We can also have an abstraction layer to implement custom control algorithms for autoscaling of pods because in literature we saw that the default Kubernetes algorithm plays bad when a lot of requests are directed. Autoscaling will simply deploy too many nodes and result in wastage of resources.

7.4 <u>Deployment Strategies</u>

There are various strategies of deployment such as rolling, blue-green, canary deployment

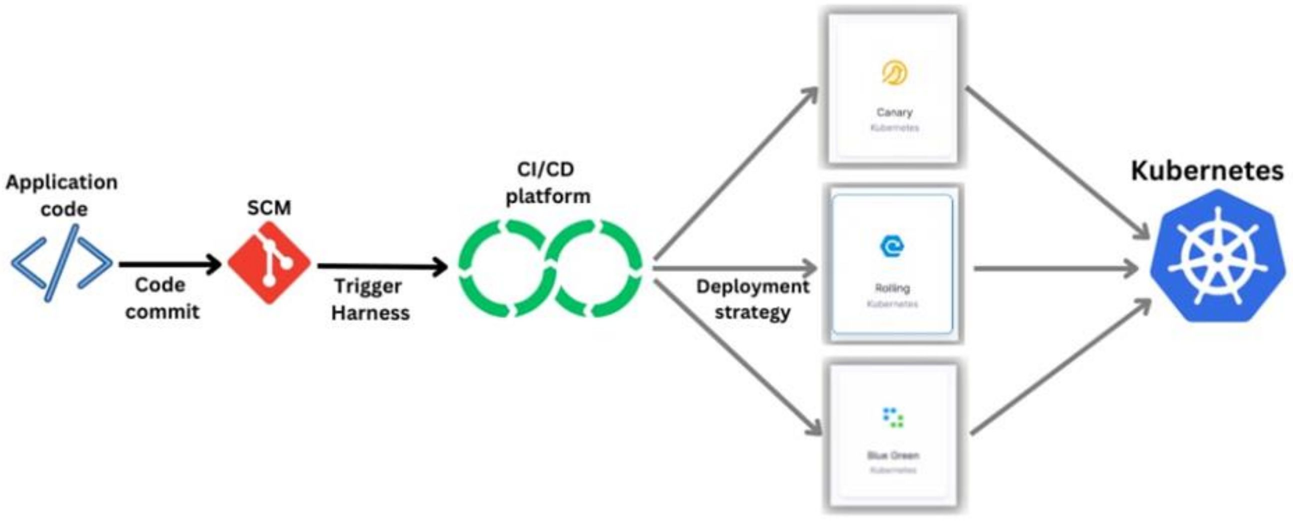


figure 7.4: deployment strategies

Canary deployment

If we have any new feature to be added, we do so in such a way that the feature does not span to the entire system rather we will deploy in small percentage and see if it is working fine. After confirming that it will work fine the feature can be deployed to the entire system

Rolling deployment

Rolling deployment includes creating a new replica sets with the updated version of the software along with gradually scaling down the old replica set.

Blue-green deployment

It's like having two instances of the software they are called blue version which is the actual production environment and the green version which is the version with new feature. This ensures thorough testing, before switching the production environment. This will reduce the downtime of the application and no error when updates are released.

7.5 CI/CD Pipeline to Deploy Microservices on Kubernetes

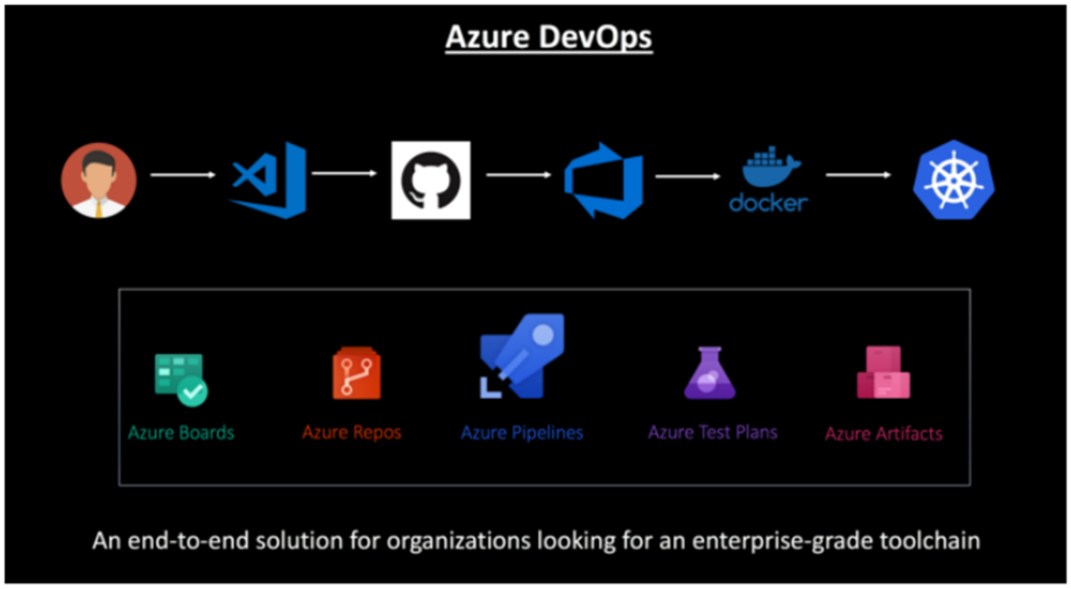


figure 7.5: azure devops

The developer writes code and push it to the GitHub, using continuous integration the code will be used to build the container images and unit tested and pushed into the azure container registry.

The continuous deployment ensures deployment of the containers build using the images into the Kubernetes cluster/azure container service.

7.6 <u>EOX vantage logical architecture of application and services</u>

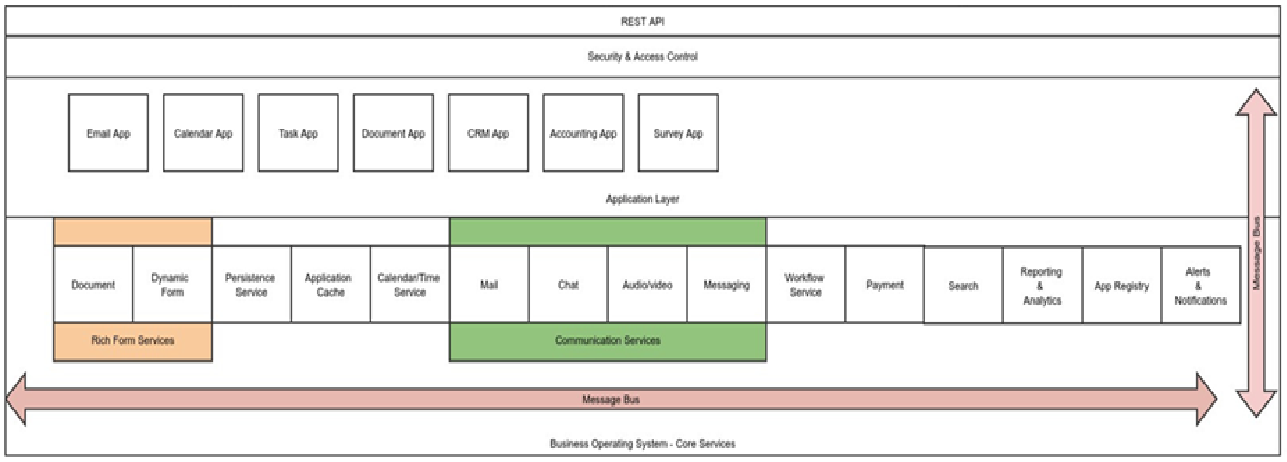


figure 7.6: logical architecture

7.7 <u>Conclusion:</u>

The microservices will be containerized and deployed on Istio service mesh on Kubernetes platform. Each microservice is associated with an envoy proxy, and the control plane is responsible for injecting the side cars and configuring them. And the master node will do the work of scaling the pods and deploying and managing the containerized workloads. For all these we have to build docker images and write yaml files to build and configure the containers and then deploy locally on Kubernetes cluster and test it.

8. PROJECT PROGRESS

We installed Docker and set it up for the project. This involved configuring the necessary settings and installing the required dependencies. Then we installed MySQL 5.7 and configured the database. We created the necessary database and set up the database rules. This included creating user accounts, assigning access rights, and configuring other security settings. We used to docker-compose to build the API once we set up the database. We tested the API on our local machine and on different operating systems. We are now working on the database migration.

9. TECHNOLOGIES USED

- **Docker:** Docker is an open-source platform for app development, delivery, and running smoothly. Docker allows us to decouple our apps from the infrastructure we have in place, allowing us to release software more quickly. The Docker software platform enables programmers to run their applications in containers. These self-contained environments provide all necessary dependencies and settings. Docker's useful tools and services, which include Docker Engine, Docker Compose, Docker Hub and Docker Swarm, enable developers to create, manage, and share containers across a number of platforms and settings.
  Docker, in essence, simplifies the process for developers to ensure that their programmes work reliably and consistently wherever they are deployed. Docker's extensive features and functions simplify the process of developing and managing applications in containerized environments.

- **Kubernetes:** Kubernetes is a piece of software that assists in the administration of containers such as Docker over a network of machines. This allows you to easily deploy, scale, and manage your containerized apps while keeping them portable and adaptable. Kubernetes can also help you ensure your applications' availability and health, as well as their network and storage requirements. In brief, Kubernetes is a fantastic tool for simplifying container-based application administration.

- **Minikube:** Minikube is a piece of software that allows you to run Kubernetes on your PC without the requirement for a full-scale cluster. Minikube allows you to construct a single-node or multi-node cluster for testing and developing Kubernetes applications. Minikube's default container runtime is Docker, however you can change it to another runtime, such as containers, if you choose. Minikube gives you quick access to Kubernetes capabilities like Services, Ingress, and Dashboard. It's a wonderful tool for folks who wish to experiment with and learn about Kubernetes without having to put up a full cluster.

10. CONCLUSION OF CAPSTONE PROJECT PHASE-1

We began Phase-1 of the Capstone Project by comprehending the problem's description, difficulties, and intended solution. We undertook extensive study to acquire a greater understanding of the problem and how to tackle it. We also highlighted the requirement for a scalable, high-performance system capable of handling the anticipated load.

To resolve the issue, we did a literature review on the problem description and investigated several solutions. We weighed the benefits and drawbacks of numerous architectures and design patterns before settling on the best one for our challenge.

We focused on the microservices architecture as a viable option for scalability and performance. We investigated how this design may assist us solve our challenge and achieve our goal. We have outlined several potential issues that may arise during the deployment of the microservices architecture.

Finally, we sketched out a rough design for the architecture and solution to the problem.

11. PLAN OF WORK FOR CAPSTONE PROJECT PHASE-2

We will be implementing the suggested architecture and design that we established in Phase-1 for Phase-2 of the Capstone Project. This involves building the required components and modules using the specified technologies and programming languages. We will also check that the components can interact with one another and that they meet the project requirements. It is critical to ensure that the architecture used is scalable, stable, and capable of handling the anticipated load.

We will deploy the completed containers on a Kubernetes cluster after the architecture has been established. We will set up the essential infrastructure, such as networking, storage, and security. It is critical to set up the containers such that they function as a cohesive system and that the deployment is fault-tolerant, scalable, and secure.

We will put the implemented architecture to the test using technologies such as Kiali, Prometheus, and Grafana. Kiali will aid in the visualisation of the service mesh and the monitoring of service-to-service communication. Prometheus and Grafana will assist us in gathering and analysing system performance indicators. End-to-end testing will be performed to confirm that the system can manage the predicted traffic and is fault tolerant.

We will use an e-commerce application to show the implemented architecture. The application will be configured to leverage the service mesh and the microservices architecture. We will ensure that the application is working properly and can manage the projected load.

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, 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.

- 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

phase1_report_merged

ORIGINALITY REPORT

7%	5%	1%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	ijisrt.com Internet Source	2%
2	Submitted to National College of Ireland Student Paper	2%
3	dzone.com Internet Source	1%
4	slides.com Internet Source	1%
5	Submitted to PES University Student Paper	1%
6	dspace.lboro.ac.uk Internet Source	<1%
7	Submitted to University of Teesside Student Paper	<1%

Exclude quotes	On	Exclude matches	< 5 words
Exclude bibliography	On		