E-Commerce Websites Through Containerization, Kubernetes Orchestration, and Istio Service Mesh Monitoring

Shikha Gupta
✉ 09.shikha@gmail.com

Kripa Sarvaiya
✉ kripa.sarvaiya26@gmail.com

Aamir Talat
✉ aamir.talat@gmail.com

Sanya Shrivastava
✉ 23sanya.shrivastava@gmail.com

Atharva Patil
✉ patilatharva876@gmail.com

Department of Computer Engineering
Rajiv Gandhi Institute of Technology
Mumbai, Maharashtra

ABSTRACT

The concept of microservices has emerged as a transformative paradigm, and departure from monolithic architectures. We will showcase how this architecture enables organizations to achieve enhanced resilience, fault isolation, and improved resource utilization. The utilization of Docker containers empowers e-commerce websites with isolated, lightweight, and reproducible environments, streamlining deployment and reducing compatibility issues. Kubernetes orchestration enhances the system by automating the management of these containers, providing dynamic scaling, high availability, and efficient resource allocation. The pivotal aspect of this study lies in the incorporation of the Istio service mesh, which offers a robust framework for traffic management, load balancing, and security, thereby mitigating potential bottlenecks and enhancing fault tolerance. The Kiali dashboard, a powerful visualization tool, grants operators' real-time insights into the traffic flow, service dependencies, and network performance, facilitating proactive decision-making and optimizing the overall user experience. Through a comprehensive analysis of this integrated approach, we demonstrate how it not only addresses the challenges of scalability and reliability in e-commerce but also provides an adaptable architecture. The findings offer practical insights for businesses aiming to deliver seamless and dependable e-commerce solutions. E-Commerce Websites through Containerization, Kubernetes Orchestration, and Istio Service Mesh Monitoring.

KEYWORDS *: Microservice, Containerization, Docker, Kubernetes, Istio, Traffic monitoring, Kiali dashboard, Load balancing, Visualization, Network dependencies.*

INTRODUCTION

In software engineering, the system development life cycle, is used to describe the process of planning, developing, testing and maintenance. Development is when the real work begins when a programmer, network engineer and database developer work on the project. Maintenance is when the system is put in usage and enrichment are added and errors are fixed as they are discovered. According to the reports average of 40% of time and cost of the system is spent in development, the majority of time and cost is spent on maintenance.

In recent years, there has been a shift from monolithic applications to a microservice architecture. By utilizing a microservice architecture, a software application is split into smaller services. Where each independent services communicate with each other through defined APIs.

The rise of microservice architecture coincided with a growing trend in adopting containerization platforms like Docker. Containers provide a streamlined environment for service execution, facilitating rapid system scalability and accelerating time-to-market for

new features. Utilizing container orchestration tools, such as Kubernetes, enables the automated scaling of containers across diverse servers, contributing to the increased availability and widespread adoption of distributed systems.

Recently a technology called service-mesh has gained admiration in handling the problems with the fallacies of distributed computing, where one example of a service mesh is Istio. As applications scale and the number of microservices increases, monitoring service performance becomes increasingly difficult. Manage connections between services, a service mesh provides new features like monitoring, logging, tracing, and traffic control. it also provides observability features for communication between services.

LITERATURE SURVEY

Thesis by Ennio Mara et al. evaluates using a service mesh as a solution for monitoring network performance metrics like bandwidth, latency, and packet loss. The thesis experiments with Istio and examines if its observability features can be used to monitor inter-service communication in a distributed system. Experiments apply known network faults and evaluate if the faults can be detected from the metrics provided by Istio. The performance impact of using Istio is also measured [1].

The paper by Gagan Mittal et al. underscores the importance of DevOps practices, especially containerization, in e-commerce software development. It highlights the need for specific competencies and cultural patterns for effective software delivery. Emphasizing replicating production environments on developers' machines to minimize discrepancies, the paper identifies containerization as a key solution. It allows reproducibility across various environments, irrespective of the host OS, separating runtime from software and hardware. This separation streamlines maintenance efforts and ensures continuous application delivery [2].

The methods used in Devops in E-Commerce Software Development Demand For Containerization by Roman Zakharenkov et al. to collect data on software delivery practices and evaluate solutions included conducting a survey among professionals at Vaimo Group. The survey was designed to gather information on the perceived value of the implemented CI/CD solution, the existing DevOps culture, and the feasibility of potential capabilities for a new software delivery pipeline. The survey data was then analyzed using Rapid miner software, and manual classification and analysis were carried out for survey questions requiring textual feedback [3].

CONTAINERIZATION IN E-COMMERCE

Daunting task of managing software dependencies where one service requires one version of library where another service “another”. Different service is not compatible with the Operating System. Every time something changes in the application; we have to check compatibility between various components and underlying infrastructure. This compatibility metric issue is referred to as “THE MATRIX FROM HELL”. Fortunately, containerization emerges as a solution to simplify this intricate challenge.

The concept of containerization, which has significantly transformed IT, may appear to have emerged unexpectedly. However, the roots of containerization trace back to the 1970s when the original idea was implemented on a Unix system to enhance the isolation of application code. Over the years, there have been numerous attempts to address virtualization, isolation, and resource management in applications.

In 2008, a significant milestone was reached with the introduction of LXC (LinuX Containers), representing the first and most comprehensive implementation of a Linux container manager. Utilizing cgroups and Linux namespaces, LXC achieved effective isolation and resource control within a single Linux kernel, eliminating the need for any additional patches.

Upon its establishment in 2013, Docker initially relied on LXC as its default containerization technology, a choice that was subsequently supplanted by the development and implementation of Docker's proprietary libraries. In the Docker ecosystem, each component operates within its distinct container, encompassing its own libraries and dependencies. Notably, these containers operate within the confines of the same operating system, yet they maintain separate environments, ensuring isolation and encapsulation of each component. Definition

of container as being “a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.” [3].

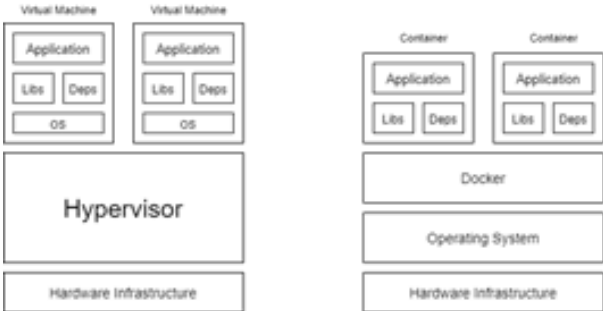

Figure 1. Difference between Virtual Machine and Container

The above figure describes the difference between virtualization and containerization. Each Virtual Machine (VM) has its own its OS inside it, then dependencies, library and application. There is higher utilization of resources as there are multiple virtual OS and Kernel are running.VM make use of hypervisor such as VMware ESXi, is a virtualization platform that enables the creation and management of multiple virtual machines (VMs) on a single physical server. VM have complete isolation from each other as they don’t rely on underlying OS or Kernel. Different type of application built in different OS such as Linux based or Windows based on same hypervisor.

Docker containerization make use of which allows for the following capabilities:

- Replicating a production environment on the developer’s machine, so as to decrease the delta between the two, ensuring that the product being further developed runs on the production environment as expected from being tested on local machine.
- Reproduction of the application in a given state irrespective of the host operating system, through the deployment of container to the target environment.

KUBERNETES

After developing an e-commerce web-store within a container on the developer's workstation, the subsequent phase involves delivering the software update to the customer. Employing containers in software development proves cost-efficient for companies, expediting and streamlining the process, as attested by industry professionals. This efficiency not only accelerates development but also diminishes associated costs. Moreover, containerization enhances the the added reliability, scalability and ease of maintenance.

One of the core features behind Docker containers is that they run ignorant of the environment where Docker is installed in and, therefore, they are designed to run on any environment that Docker supports. Even though it is possible with a larger overhead to achieve the same result using a virtual machine, it does not come out of the box and requires more complicated configuration, setup and maintenance efforts later in the exploitation [8].

Following the containerization of software, Kubernetes comes into play, enabling precise control over container deployment. Kubernetes serves as a Container Orchestration Technology, facilitating the orchestration of deployment and management for numerous containers within a cluster environment. This capability empowers organizations to dictate the specific cluster where containers operate and determine the number of instances, referred to as "pods." Consequently, Kubernetes serves as a sophisticated tool for managing containerized applications efficiently.

Kubernetes does not deploy container on worker node directly. Container are encapsulated into Kubernetes object called as “pods”. When faced with increased load, the solution doesn't involve adding new container instances within the same pod. Rather, a new pod is created for the same application. It's important to note that pods exhibit a one-to-one relationship with containers.

In scenarios where a single pod is running an application and an unfortunate event, such as a crash, occurs, users lose access to the application. To mitigate this risk, it becomes essential to have more than one instance or pod concurrently operational. This is where the Replication Controller becomes crucial, enabling the simultaneous running of multiple instances of the same pod. It plays a pivotal role in automatically initiating a new pod when an existing one fails.

The Replication Controller diligently monitors pods, allowing for the potential existence of hundreds of pods. Labels are employed as a filter for the replication set, contributing to the efficient and organized management of multiple pod instances.

ISTIO SERVICE MESH

In recent years, the software architecture landscape has undergone significant transformations, marked by a notable shift away from large monolithic applications to more fine-grained deployment units known as microservices.

In the past, monolithic architectures were commonplace, with applications developed as a single unit. A typical monolith comprised a client-side for user interaction, an application layer housing business logic, and a storage layer, often a database. However, as monolithic applications are packaged as a unified entity, issues like a burgeoning codebase, challenges in maintaining modular structure, and scaling problems emerged over time. Scaling difficulties arose because the entire unit had to be scaled up, rather than just the specific component requiring additional resources.

The adoption of a microservices architecture addresses these challenges by breaking down the application into distinct services, each being small and autonomous. Microservices are designed to be small, focused on performing a single task effectively, and operate as independent entities.

While microservices offer advantages, they also introduce unique challenges. The decentralized nature of services, their fluidity, and elasticity make tracking instances, versions, and dependencies a complex task. The management of a growing service landscape further complicates this issue. Services may fail independently, aggravated by unreliable networks. In a large system, some parts may experience minor outages at any given time, impacting a subset of users, often without the operator's immediate awareness.

To navigate these challenges and ensure smooth system operation without adversely affecting customers or overwhelming developers, effective strategies must be employed. Proactive monitoring, robust version control, and comprehensive error handling mechanisms become crucial. Additionally, maintaining clear communication channels, implementing automated testing, and continuously refining system architecture can contribute to a resilient microservices ecosystem.

A service mesh constitutes a dedicated infrastructure layer that can be integrated into applications, offering the ability to seamlessly incorporate features like observability, traffic management, and security without necessitating modifications to the underlying code. The term "service mesh" encompasses both the software utilized to implement this pattern and the resultant security or network domain it establishes.

As the scale and intricacy of distributed services, particularly within Kubernetes-based systems, expand, managing and comprehending them can pose significant challenges. The evolving demands of such deployments encompass facets like service discovery, load balancing, fault recovery, metric tracking, and monitoring. Moreover, a service mesh typically addresses more intricate operational requirements such as A/B testing, canary deployments, rate limiting, access control, encryption, and end-to-end authentication.

SECURITY CONSIDERATION

Jaeger

In contrast to a monolithic architecture, identifying the root cause of issues becomes more challenging in a distributed system. Resolving these challenges requires insights into which service transmitted specific parameters to another service or component.

Traces serve as a valuable visual tool, offering a comprehensive view of our system and facilitating a better understanding of the interrelationships between services. This visualization proves instrumental in investigating and precisely locating issues within the architecture.

Jaeger, an open-source distributed tracing platform developed by Uber in 2015, encompasses instrumentation SDKs, a backend for data collection and storage, a user interface for visualizing data, and a Spark/Flink framework for aggregate trace analysis. Following the Open Tracing specification, Jaeger, like many other distributed tracing systems, operates with spans and traces.

A trace, essentially a list of spans interconnected in parent/child relationships, can be conceptualized as a directed acyclic graph of spans. Traces provide a

clear depiction of how requests propagate through our services and various components.

Grafana

Grafana, a solution for open-source data analytics, plays a crucial role in the field of data analytics making it an essential tool for ensuring the smooth operation of your e-commerce platform. Users can efficiently visualize data by creating integrated dashboards, charts, and graphs that span multiple dashboards, thereby simplifying the interpretation of complex datasets.

Grafana excels in time series analysis, user behavior tracking, application performance monitoring, and assessing error frequency and types in different environments (production, pre-production, etc.), proving invaluable for understanding errors in various scenarios. These insights are crucial for enhancing operational performance and making well-informed decisions.

The platform also simplifies alert management by allowing users to visually define thresholds and receive notifications through platforms such as Slack and PagerDuty for prompt issue resolution. Being completely open source, Grafana provides users with the flexibility to deploy it on their preferred platforms.

The capability to explore large logs efficiently using label filters is a key feature, facilitating rapid data retrieval and analysis. Grafana further allows users to present data via templates or custom reports for easy visualization and sharing of results with internal teams and stakeholders. Report creation and sharing are possible, though this feature is not available in the open-source version and requires an upgrade to the platform.

IMPLEMENTATION

Online Boutique stands as a cloud-first microservices demonstration application, designed as a web-based e-commerce platform. Users engage with the application by browsing available items, adding selected products to their carts, and completing purchases seamlessly. The architecture is composed of independent microservices, each catering to specific functionalities, thereby enhancing the overall user experience. This cloud-native application showcases modern development practices and serves as a valuable example of microservices architecture in the e-commerce domain.

Recommendation Service

The Recommendation service provides personalized product recommendations based on user preferences and historical data.

Product Catalog Service

The Product Catalog service manages the product inventory, providing information and updates about available products.

Cart Service

The Cart service handles user shopping carts, allowing users to add, remove, and manage items before completing the checkout process.

Redis Cache

The Redis Cache service is a high-performance, in-memory data store used for caching frequently accessed data, enhancing system responsiveness.

Frontend

Figure 2 describes the Frontend service represents the user interface, interacting with users and serving as the entry point for various functionalities.

Checkout Service

The Checkout service is in charge of processing and managing the checkout process, handling user orders, and coordinating with payment services.

Shipping Service

The Shipping service manages the logistics and coordination of shipping processes, ensuring timely delivery of user orders.

Currency Service:

The Currency service handles currency conversion, allowing users to view and transact in different currencies based on their preferences.

User Service

The User service manages user-related functionalities, user authentication, registration, and profile management.

Load Generator

The Load Generator service is responsible for simulating

user traffic to assess and optimize the performance and scalability of the entire system.

Ad Service

The Ad service handles the display and management of advertisements, enhancing the user experience by incorporating targeted advertising.

Payment Service

Figure 3 describes the Payment service is responsible for processing user payments securely, integrating with payment gateways and ensuring transactional integrity.

Email Service

The Email service manages the sending of transactional emails, including order confirmations, shipping notifications, and other relevant communication.

In Figure 4, a Docker file outlines instructions and arguments to create a Docker image, serving as an executable package with all necessary dependencies (Figure 6). Figure 5 introduces the Docker build command to construct the image, while Figure 7 highlights the Docker run command for launching isolated containers. Figure 8 shifts focus to efficiently deploying the online Boutique microservice app using Kubernetes for seamless management and scalability. In Figure 9, Kiali visualizes traffic flow between microservices in a Kubernetes cluster. Figure 10 features a Grafana Dashboard for Telemetry, offering insights into various metrics, and Figure 11 details the Client Request Duration chart, aiding in performance optimization. These figures collectively underscore the importance of monitoring and visualization tools for the health and efficiency of microservices in Kubernetes environments.

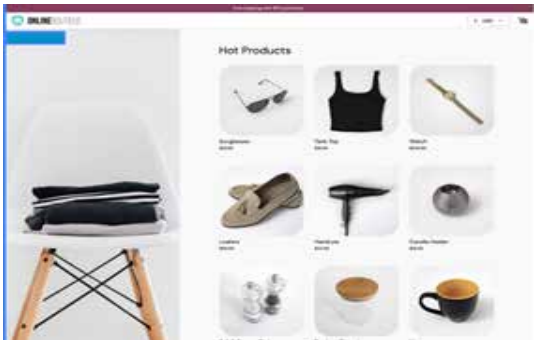

Fig 2. Online Boutique System Frontend-1

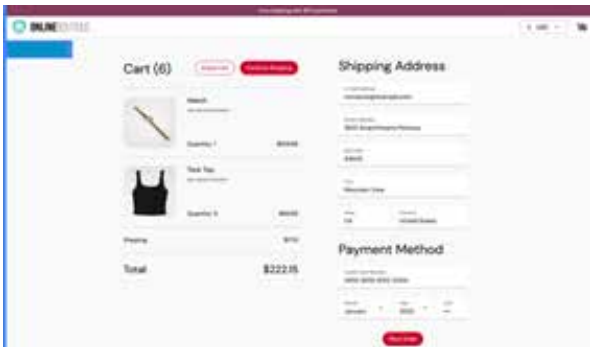

Fig 3. Online Boutique System Frontend-2

```
root@ip-172-31-36-170:/home/ubuntu/new/docker# cat Dockerfile
FROM python:3.6-alpine

RUN pip install flask

COPY . /opt/

EXPOSE 8080

WORKDIR /opt

ENV APP_COLOR=red

ENTRYPOINT ["python", "app.py"]
```

Fig 4. Docker File

```
root@ip-172-31-36-170:/home/ubuntu/new/docker# docker build -t sripaservaiya/hello:v1 .
[+] Building 0.3s (9/9) FINISHED
```

Fig 5. Docker build

```
root@ip-172-31-36-170:/home/ubuntu/new/docker# docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
sripaservaiya/hello       v1        c24c3e184cb4   28 seconds ago   51.9MB
```

Fig 6. Docker Image

Fig 7. Docker Run

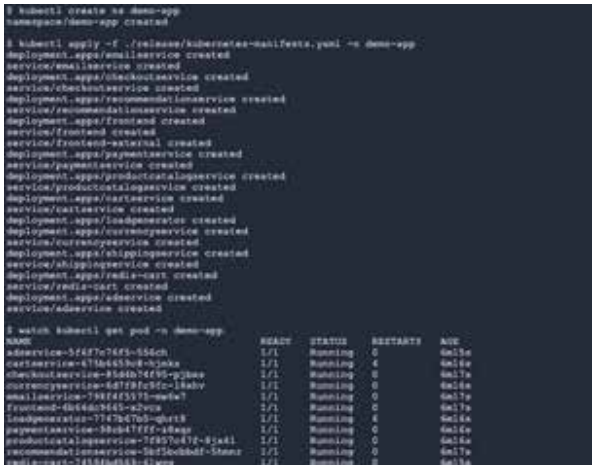

Fig 8. Deploying Online Boutique App

Fig 9. Istio Kiali Dashboard

Fig 10. Grafana Dashboard for Telemetry

Fig 11. Client Request Duration Telemetry.

CONCLUSION

In this project, we have successfully containerized an e-commerce web application and deployed it on a Kubernetes cluster for orchestration and load balancing. By leveraging Docker containers, we can achieve portability, consistency, and isolation for the application across environments.

Kubernetes allows us to easily scale, update, and monitor the application through features like horizontal pod autoscaling, rolling updates, and its metrics server.

Integrating Istio service mesh further enhances traffic management, security, and observability. Its sidecar proxies and rich telemetry provide detailed insights into service communication and performance.

Grafana enable gathering and visualizing metrics from the infrastructure, containers, and Istio data plane. This helps identify issues and optimize services. Jaeger gives request tracing to pinpoint latency problems. Overall, we have an end-to-end observable system.

The use of containers, orchestrators, service mesh and monitoring deliver operational efficiency, scalability, reliability, and visibility. It is a modern cloud-native approach suitable for today's dynamic microservices-based applications.

There are several areas where this system can be enhanced further:

Looking forward, the trajectory of e-commerce websites built on containerization, Kubernetes orchestration, and Istio service mesh monitoring reveals a dynamic future. Emphasis will be placed on optimizing scalability, ensuring these platforms can adeptly respond to varying user demands. The integration of edge computing is poised to enhance user experiences by minimizing latency through the strategic placement of computing resources. AI and machine learning are anticipated to become integral, driving personalized recommendations and bolstering fraud detection capabilities. Security enhancements will persist as a crucial focus, with ongoing developments in container security and service mesh features. Build CI/CD pipelines supporting multiple environments.

REFERENCES

1. Mara, E., & Frykholm, E.2021. Evaluating Service Mesh as a Network Monitoring Solution.

2. Mittal, G., Dhingra, G., & Vardan, H.2021 . Automation and Containerization For Smart E-Marketplace. IJMTST0712086, Volume(Issue 12)

3. Roman Zakharenkov DEVOPS IN E-COMMERCE SOFTWARE DEVELOPMENT: DEMAND FOR CONTAINERIZATION.

4. Jösch, R. M. 2020 . Managing Microservices with a Service Mesh: An Implementation of a Service Mesh with Kubernetes and Istio.

[5] Sharma, K. 2020. A Surge in E-commerce Market in India After COVID-19 Pandemic.

[6] Sedghpour, M. R. S., Klein, C., & Tordsson, J. 2022. An Empirical Study of Service Mesh Traffic Management Policies for Microservices.

[7] Ashok, S., Godfrey, P. B., & Mittal, R. 2021.Leveraging Service Meshes as a New Network Layer.

[8] What's a Linux Container. https://www.redhat.com/en/topics/containers/whats-a-linux-container