Containerization of a polyglot micro service application using Docker and Kubernetes

Vamsi Krishna Yepuri, Venkata Kalyan Polamarasetty, Shivani Donthi, Ajay Kumar Reddy Gondi
Kent State University

***Abstract*—This project investigates the benefits of containerization technology in modern software development and deployment. The study emphasizes the advantages of using Kubernetes and Docker in the development process, including the easy packaging and deployment of micro services, efficient resource utilization, faster startup times, and greater scalability and flexibility. The project concludes by proposing a study that involves creating a polyglot micro service application using Java, Python, and JavaScript, containerizing it with Docker, and deploying it in Kubernetes. The study aims to evaluate service discovery and auto-scaling in distributed mode and compare the performance metrics with virtual machine and container. The results of this study can inform software development teams about the benefits of containerization in modern software development and deployment.**

***Index Terms*—Docker, Kubernetes, Containers, Microservices, Polyglot**

I. Introduction

Containerization has transformed software development and deployment with Docker being one of the most widely adopted open-source platforms for building, shipping, and running distributed applications. However, as containerized applications become more complex, managing them at scale can be a challenge. Kubernetes, an open-source container orchestration system, provides a platform for managing and deploying containerized applications at scale. Its popularity is attributed to its ability to automate deployment, scaling, and management of containerized applications, with features like automatic load balancing, scaling, and self-healing. Trends in Kubernetes technology include the adoption of serverless computing, which enables developers to run their code in lightweight containers without managing infrastructure or container orchestration. Kubernetes also finds use in hybrid and edge computing, where computing resources are distributed across multiple devices and locations, demonstrating its versatility and adaptability to modern software development and deployment needs.

Public cloud providers have played a significant role in the adoption and advancement of container technology. They offer container services that allow developers to deploy, manage, and scale containerized applications, which have reduced the barrier of entry to container adoption, making them available to businesses of all sizes. These providers have also been instrumental in promoting container standards and enabling interoperability, contributing to the adoption of Kubernetes as the de facto standard for container orchestration. Standardization of container orchestration simplifies the deployment and management of containerized applications across different cloud providers and on-premises environments.

Containers provide a variety of benefits in microservices architecture, with one of the most significant being the ease of packaging and deploying microservices. Each microservice can be packaged as a container image, which abstracts it from the underlying infrastructure and makes it simpler to deploy and run the services across various environments without compatibility issues. For example, consider an e-commerce application based on microservices, with various services such as product catalog, shopping cart, payment processing, and order management. Each of these services can be containerized, making it easy to deploy and update them independently. This enables developers to add new features or modify existing ones without impacting other services.

Additionally, containers provide a high degree of isolation between microservices, minimizing conflicts and dependencies between them. By packaging each service as a container, they can each run their own instance of the required database, without impacting the other services in the system. As a result, containers provide greater agility, scalability, and consistency in a microservices project, allowing developers to focus on developing and updating microservices rather than the infrastructure and deployment processes.

The use of virtual machines (VMs) can present challenges when it comes to running applications, as VMs require their own operating system and consume significant amounts of CPU, memory, and storage resources, which can result in slower application performance and increased complexity. This can be solved through the use of container technology, which allows multiple containers to share the same operating system kernel, leading to better resource utilization and faster startup times for applications, and ultimately greater scalability, flexibility, and cost efficiency.

For instance, consider a data analytics application that needs to process a large amount of data quickly and efficiently. If the application runs on a virtual machine, it may suffer from slower performance due to the overhead and complexity of running a full operating system. By using container technology, however, developers can run multiple lightweight containers on the same host operating system, resulting in faster performance, lower resource usage, and greater scalability for the application, allowing for faster data processing and analysis.

Achieving container capabilities with virtual machines is challenging due to the fundamental differences between the two technologies. Containers are lightweight and efficient, sharing the same operating system kernel as the host machine, while virtual machines emulate an entire hardware environment, including a separate operating system, which can lead to higher resource usage and greater complexity. Additionally, virtual machines lack portability and flexibility as each one requires a specific configuration of hardware and operating system, making it difficult to move or replicate them across different environments. Containers, on the other hand, can be easily packaged and deployed on any host system that supports the same container runtime. Although virtual

machines have their own strengths and use cases, containers are preferred for most modern applications due to their greater efficiency, flexibility, and portability, and because they can start up much more quickly, making them better suited for modern application deployment and scaling.

The traditional monolithic application approach involves building a single, comprehensive app, which can be time-consuming to develop and maintain, making it challenging to achieve agility. To address these issues and achieve greater flexibility and ease of maintenance, the world is moving towards microservices. In the era of microservices, numerous independent modules are set up in traditional virtual machines, which can lead to conflicting libraries and make the process of provisioning, scaling, service discovery, load balancing, and deployment manual and time-consuming. To solve these problems, microservices can be packed in containers instead of virtual machines, which will help to address conflicts and create independence between each microservice. Workloads can be run in a scalable and distributed manner using Kubernetes, which can solve auto-scaling and service discovery issues. This project involves building a polyglot microservice application from the ground up using Java, Python, and JavaScript, containerizing it with Docker, and deploying it in Kubernetes to examine service discovery and auto-scaling in distributed mode, while comparing the performance differences between the containerized technique and the traditional virtual machine strategy.

In this project, we aim to compare the performance of virtual machines and containers using several metrics. Firstly, we will examine the resource utilization of the microservice application between the two technologies, analyzing the CPU, memory, and storage usage to determine which technology is more efficient. Secondly, we will compare the startup time and scaling capabilities of virtual machines and containers to see which is faster and more scalable. Thirdly, we will examine the storage space utilized by both virtual machines and containers to see which technology uses storage more effectively. Finally, we will evaluate the portability of both virtual machines and containers and their ability to be moved across different environments. By comparing these performance metrics, we aim to determine which technology is more suitable for our microservice application and gain insights into the strengths and weaknesses of each technology.

II. Related Work

Micro services, Docker, Kubernetes and polyglots are some of the major collectively used technologies in the field of software development . There are many works can be found on these topics which demonstrates the purpose of shift towards the docker and kubernetes along with the challenges involved the approaches.The importance of adopting a microservice-based architecture to achieve high availability for stateful applications in Kubernetes is being explained in [1]. The challenges of managing state ful micro services in Kubernetes and also proposed a solution to improve their availability using a State Controller but did not mention about the performance of Stateless polyglot micro service.In [2] , presented a case study of a web application that consists of several micro services, each running in a separate container and highlighted the improved portability, scalability, and flexibility. There are some of the situations that demands use of polyglots in order to achieve the desired solutions. [3] discussed the approaches for achieving consistent distributed transactions across polyglot micro services that use multiple databases and addressed the challenges in distributed transactions in a polyglot environment. [4] proposes a performance model to analyze the resource management in Kubernetes clusters. The model uses queuing theory and stochastic processes to represent the resource utilization of nodes and pods. The proposed model is validated through experiments that compare its accuracy against the real system performance. The paper highlights the importance of having an accurate performance model to optimize resource allocation and improve the overall efficiency of Kubernetes clusters. Also in [5] the evaluation of the performance of micro services architectures using containers, specifically Docker. The authors conduct experiments to compare the performance of a monolithic application with a micro services-based application, both deployed on Docker. The experiments involve measuring the response time and throughput of the applications under various workloads. The results of the experiments demonstrate that the micro services architecture outperforms the monolithic architecture in terms of response time and scalability. Overall, the use of micro services, Docker, Kubernetes, and polyglots in software development has become increasingly popular in recent years due to their benefits in terms of portability, scalability, and flexibility. However, there are also challenges involved in using these technologies, such as managing stateful micro services in Kubernetes and achieving consistent distributed transactions across polyglot micro services.To address these challenges, researchers have proposed various solutions and approaches, such as using a State Controller to improve the availability of stateful micro services in Kubernetes and developing a performance model to optimize resource allocation in Kubernetes clusters.In addition, studies have been conducted to evaluate the performance of micro services architectures using containers like Docker, and the results have shown that micro services architectures can outperform monolithic architectures in terms of response time and scalability.Overall, the adoption of micro services, Docker, Kubernetes, and polyglots in software development is likely to continue to grow as more organizations seek to achieve the benefits these technologies provide while also addressing the challenges involved in their use.

From *TABLE I: Related Work*, we can see the different features implemented by the referenced papers and the features that are implemented in this project. Our proposed solution will be dealing with a stateless polyglot micro services, containerized used docker and kubernetes.We deploy our application and analyse the improved performance using kubernetes.

Reference	Microservices	Stateless	Container	Kubernetes	Polyglot
[1]	✓	X	✓	✓	X
[2]	X	✓	✓	X	X
[3]	X	X	✓	X	X
[4]	✓	✓	✓	✓	X
[5]	✓	X	✓	X	✓
Our Project	✓	✓	✓	✓	✓

TABLE I: Related Work

III. Project background

The trend towards microservices architecture has become more and more popular in recent years, as it allows organizations to develop, deploy, and manage large, complex applications in a more agile and efficient manner. However, one of the challenges

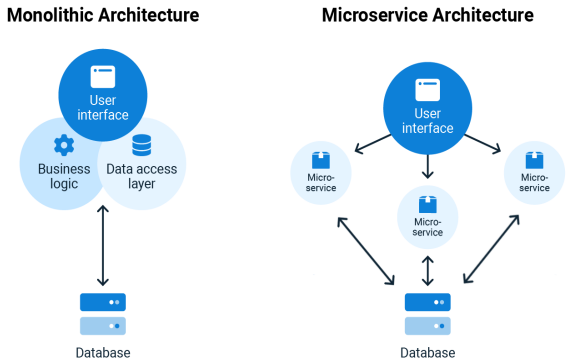

Fig. 1: Monolithic vs Microservices

Fig. 2: Polyglot Microservices

of implementing a microservices architecture is managing the deployment and scaling of individual services in a way that is both effective and efficient. This is where containerization and container orchestration tools like Docker and Kubernetes come into play.In this project we are going to show the analysis of performance improvement with the use of container orchestration tools.

A. *Microservice vs Monilithic architecture*

Monolithic and microservice architectures are two different approaches to designing and building software systems.A monolithic architecture is a traditional approach where an application is built as a single unit, while a microservice architecture is an approach where an application is broken down into smaller independent services. *Fig. 1: Monolithic vs Microservices* shows the difference between monolithic and microservice based architecture. The monolithic approach is easier to develop and deploy, but can become difficult to maintain and scale as the application grows. On the other hand, the microservice approach offers greater flexibility, scalability, and resilience but requires coordination between different teams. The choice of architecture depends on the specific needs and requirements of the application, with monolithic being better suited for smaller applications and microservices for larger, complex applications.

[6]explains the process of transforming monolithic applications into microservices and provides a framework for understanding the migration process. It discusses the benefits and challenges of adopting a microservices architecture.

B. *Polyglot microservices*

Polyglot microservices refer to a software architecture that involves using multiple programming languages and technologies to build independent, small-sized services that work together to accomplish a larger goal. *Fig.2 : Polyglot microservices* represents polyglot microservices. This approach allows developers to choose the best language and tools for each service, rather than being limited to a single technology stack.Polyglot microservices are becoming increasingly popular because they offer several benefits, such as increased flexibility, improved scalability, and better fault tolerance. They can be designed to communicate with each other through APIs, message queues, or other protocols, which enables them to work together as a cohesive system. While there are challenges to building and managing polyglot microservices,

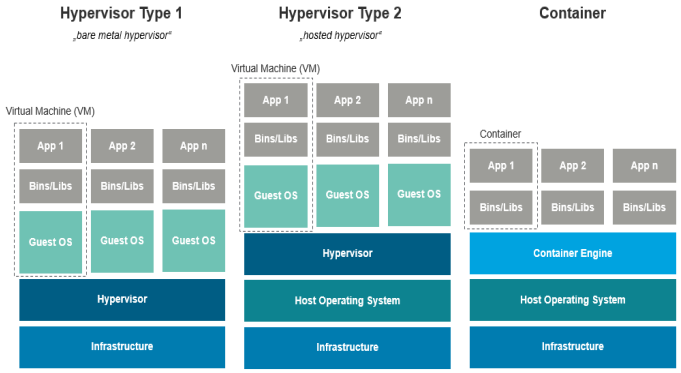

Fig. 3: Virtual Machine vs Container

such as the need to manage multiple programming languages and tools, these can be overcome with the right approach and tools. As a result, many organizations are adopting this approach to build more resilient and adaptable software systems.

The methods for achieving consistent distributed transactions across polyglot microservices that use several databases were covered in [3], that also addressed the challenges of distributed transactions in a polyglot environment.

C. *Container*

A container is a lightweight, standalone and executable package of software that includes everything needed to run an application, such as code, libraries, and system tools. Containers provide a consistent runtime environment, ensuring that the application can run reliably regardless of the underlying system's configuration. *Fig. 3: Virtual Machine vs Container* shows the difference between a virtual machine and a container. [7] is a comprehensive report on the use of container technology in modern software development. It covers various aspects of container technology, including its history, benefits, challenges, and future trends.The goal of containerization is to package each service and its dependencies into a lightweight, portable container that can run on any machine or cloud environment.

Docker is widely used as a containerization platform because it offers a reliable and reproducible setting for running applications. Docker makes it simple to create, deploy, and manage images, which makes it easier to develop and test applications across

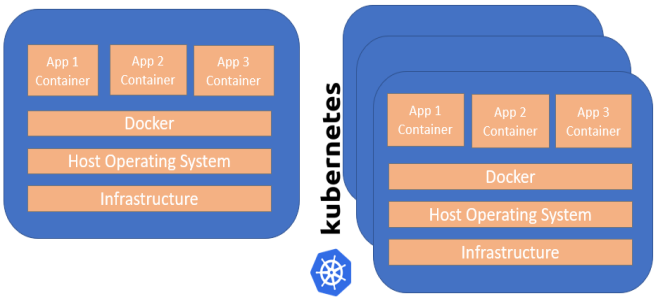

Fig. 4: Docker vs Kubernetes

different environments. A Docker container is a self-contained software package that operates independently of other containers and the host system. It has its own network, file system, and resources, making it perfect for deploying applications with specific requirements. Docker images serve as snapshots of an application's file system, and they can be created using a Dockerfile script. Images can be utilized to start one or more containers.Docker containers have advantages such as portability, consistency, isolation, resource efficiency, and scalability, while their disadvantages include complexity, a learning curve, security concerns, dependency management challenges, and limited access to hardware resources.

D. Kubernetes and Docker

Kubernetes is an open-source platform for container orchestration, which automates the deployment, scaling, and management of containerized applications.*Fig. 4: Docker vs Kubernetes* shows the difference between docker and kubernetes and how kubernetes is used in distributed environment and orchestrating the docker containers. It provides a framework for deploying, scaling, and managing multiple containers across a cluster of servers, allowing organizations to run and manage their microservices at scale. [8] explains how Docker and Kubernetes can be used to automate the scaling of a defence application in a cloud environment. It highlights the benefits of these tools, such as improved security and reliability, and provides a detailed deployment process.

In the case of a polyglot microservices application, where multiple programming languages and frameworks are used to develop individual services, containerization and container orchestration can be particularly valuable. Docker and Kubernetes enable organizations to easily package and deploy services written in different languages or frameworks, while ensuring that each service runs in a consistent and isolated environment. This makes it easier to manage dependencies, test and deploy services, and scale applications as needed.

Overall, the containerization of a polyglot microservices application using Docker and Kubernetes can help organizations achieve greater agility, scalability, and reliability in their software development and deployment processes. It can also help reduce the complexity of managing and scaling multiple services, which can be particularly challenging in a polyglot environment.

IV. Infrastructure

The infrastructure required for containerization using Docker and Kubernetes typically includes :

a. physical or virtual machines,
b. storage systems, and
c. networking components.

Physical or virtual machines are needed to host the containerized applications. These machines should have sufficient CPU, memory, and storage to run the application and the containers. Storage systems are necessary to store the container images, as well as any data or files that the application needs to access. The storage should be accessible to all the machines in the infrastructure. Networking components such as load balancers and firewalls are also required to ensure that traffic can be routed to the appropriate containers and that the network is secure. We also need a cloud engine , for that we are using GCP. Google Cloud Platform (GCP) is a cloud provider that offers a range of services for hosting and managing applications, data, and infrastructure in the cloud. GCP provides tools and services such as Google Kubernetes Engine (GKE), Google Container Registry (GCR), and Cloud Storage that are well-suited for containerization using Docker and Kubernetes. Additionally, GCP provides tools for monitoring and managing containerized applications, such as Stackdriver Monitoring and Logging, and Cloud Trace for application tracing.

There are examples provided in [9] describes the steps to deploy a web application using Google Cloud Platform, which involves creating a new project, enabling necessary APIs, choosing a deployment method, creating an instance or app, configuring it, and deploying the application. Once deployed, the application can be managed and monitored through Google Cloud Platform.

Also the GCP services used were VPC networks (Virtual Private Cloud Network) and Compute Engine. VPC networks allow for the creation of private virtual networks within GCP, while Compute Engine provides virtual machines with configurable amounts of CPU, memory, and storage for running kubernetes and deploying applications and services.

References

[1] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. Microservice based architecture: Towards high-availability for stateful applications with kubernetes. In *2019 IEEE 19th international conference on software quality, reliability and security (QRS)*, pages 176–185. IEEE, 2019.

[2] Vivek Sharma, Harsh Kumar Saxena, and Akhilesh Kumar Singh. Docker for multi-containers web application. In *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pages 589–592. IEEE, 2020.

[3] Guogen Zhang, Kun Ren, Jung-Sang Ahn, and Sami Ben-Romdhane. Grit: consistent distributed transactions across polyglot microservices with multiple databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2024–2027. IEEE, 2019.

[4] Víctor Medel, Omer Rana, José Ángel Bañares, and Unai Arronategui. Modelling performance & resource management in kubernetes. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 257–262, 2016.

[5] Marcelo Amaral, Jorda Polo, David Carrera, Iqbal Mohomed, Merve Unuvar, and Malgorzata Steinder. Performance evaluation of microservices architectures using containers. In *2015 ieee 14th international symposium on network computing and applications*, pages 27–34. IEEE, 2015.

[6] Daniel Escobar, Diana Cárdenas, Rolando Amarillo, Eddie Castro, Kelly Garcés, Carlos Parra, and Rubby Casallas. Towards the understanding and evolution of monolithic applications as microservices. In *2016 XLII Latin American Computing Conference (CLEI)*, pages 1–11, 2016.

[7] Tamanna Siddiqui, Shadab Alam Siddiqui, and Najeeb Ahmad Khan. Comprehensive analysis of container technology. In *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, pages 218–223, 2019.

[8] San Kho Lin, Umer Altaf, Glenn Jayaputera, Jiajie Li, Davis Marques, David Meggyesy, Sulman Sarwar, Shivank Sharma, William Voorsluys, Richard Sinnott, Ana Novak, Vivian Nguyen, and Kristan Pash. Auto-scaling a defence application across the cloud using docker and kubernetes. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 327–334, 2018.

[9] Ambika Gupta, Pragati Goswami, Nishi Chaudhary, and Rashi Bansal. Deploying an application using google cloud platform. In *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pages 236–239, 2020.