International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 10 October 2024
Page No: 2521 - 2528
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2024.0346

Scalable Cloud Execution Engines

G Renugadevi[1], M L Sharvesh[2], S Subhashini[3], V S Vaishaal Krishna[4]
[1]Associate professor, Dept. of CSE, Sri Krishna College of Engg. & Tech., Coimbatore, Tamil Nadu, India.
[2,3,4]UG Scholar, Dept. of CSE, Sri Krishna College of Engg. & Tech., Coimbatore, Tamil Nadu, India.
Emails: *grenugadevi@skcet.ac.in[1], 20eucs130@skcet.ac.in[2], 20eucs145@skcet.ac.in[3], vaishaalinc@gmail.com [4]*

Abstract
Scalability remains a major concern for many organizations, and as technology evolves expeditiously, the number of users utilizing it also increases rapidly. In this paper, we propose a novel approach to address this challenge through the implementation of a scalable cloud execution engine using a microservices architecture. By using this design we can achieve a system with loosely coupled and independently deployable methods. Also through this, we can achieve enhanced flexibility, scalability, and reusability in our application. Through experimenting with various execution engines it is evident that most of their design relies on monolithic architecture. However, this design poses potential challenges especially when the traffic experiences sudden spikes. Our proposed design provides practical insights for architects and developers seeking to design and deploy highly scalable cloud applications.

Keywords: *Microservice Architecture, Container Orchestration, Cloud Computing, Docker, and Kubernetes, Execution Engines.*

1. Introduction

Microservices [1] is an approach to decomposing complex applications into modular components or services. Each module is an independent deployable service and is implemented for a specific purpose. This design helps developers build and deploy applications quickly, enhance fault isolation, adapt to the architecture easily, and scale components independently. The proposed system, "Scalable Cloud Execution Engine using Microservices" aims to implement the principles of microservice architecture to establish a coherent and robust execution engine for cloud-based applications. By cultivating this approach, our proposed project can overcome the limitations of traditional monolithic architecture and experience the advantages of cloud computing. This introduction plays a major role in a comprehensive exploration of the project, highlighting the importance of microservice architecture to address problems like scalability, efficiency, and flexibility in cloud-based architecture. The project uses containerization, orchestration, scalability, fault tolerance, security, and load balancing as its key components.

1.1. Monolithic vs Microservices

Monolithic architecture [2] is a traditional way of building software applications where the entire application is developed as a single, tightly coupled unit. In monolithic architecture, all the components, like user interface, business logic, and data layer, are packed and deployed together as a single unit on servers. Microservices architecture is a modern approach to building software applications where the application consists of small, independent modules where each module communicates with each other through well-defined APIs. Each microservice module is responsible for a specific functionality that can be developed, deployed, and scaled independently. Monolithic architecture involves building a single, tightly integrated application, whereas microservices architecture involves breaking the application into smaller, independent services. Scaling monolithic applications can be tedious as their applications are large and tightly coupled but in microservices architecture, the application can be scaled easily enabling independent development and deployment. Monolithic applications have limited

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 10 October 2024
Page No: 2521 - 2528
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2024.0346

flexibility in choices of technology while microservices allow for diversity in technology stacks.

1.2. Microservices Architecture

Microservices architecture [3] allows a large application to be divided into small manageable functionalities where each has its own roles and responsibilities in the application.Each service has an individual codebase and are managed by small teams. Components like containers, orchestration, and API Gateway play an impressive role in microservices architecture.Containers are an isolated environment where the code of each service is placed on individual containers such that each container can be managed accordingly.Each container holds the code and all its dependencies of a specific functionality such that it can be executed in any computing environment quickly. Containers also provide various benefits such as resource efficiency, portability, rapid deployment, and version control. Container orchestration is used for automatic provisioning, deployment, scaling, and managing the containers without worrying about the application's architecture.This makes it easier to manage the containers and can ensure scalability and security. API Gateway is a server between clients and microservices that enables communication between them. API defines the data formats that the application uses to request and exchange data, enabling interoperability between different systems and services.

1.3. Containers

Operating system virtualization is achieved through containers [4]. Anything from a little software process or microservice to a more complex program can be run inside a single container. All required libraries, configuration files, binary code, and executables are contained inside a container. However, operating system images are not present in containers as they are in server or machine virtualization methods. As a result, they have substantially reduced overhead and are more lightweight and portable. It is possible to deploy several containers as one or more container clusters in bigger application deployments. Kubernetes, or another container orchestrator, may be in charge of these clusters.

1.4.Container Orchestration

A key technology in contemporary software development, especially in the context of microservices architecture, is container orchestration [5]. It simplifies the deployment, scaling, load balancing, and resource allocation processes by automating the management of containerized applications. Developers may concentrate on writing code by using orchestration technologies like Kubernetes [6] and Docker Swarm [7], which abstract away the complexity of infrastructure management and guarantee dependable and efficient application deployment. Automating deployment procedures is one of container orchestration's main advantages. Orchestration technologies eliminate manual intervention and lower the risk of human error when deploying containerized applications across dispersed environments. Organizations are able to provide software updates more often and dependably thanks to this automation, which also speeds up the release cycle. Furthermore, fault tolerance, resource optimization, and scalability are made easier by container orchestration. Applications stay responsive and performant as orchestration systems dynamically scale container instances to match changing workload demands. By allocating workloads effectively and proactively managing problems, they also maximize resource usage by reducing downtime and improving application reliability. All things considered, container orchestration gives enterprises the ability to create and manage highly accessible, scalable, and resilient software systems in the current dynamic computing environment.

1.5.Advantages of Microservices

Microservice architecture provides various benefits, making it a popular alternative for designing and delivering new systems. Increased scalability is one important benefit. Applications built with microservices are made up of tiny, independently deployable services, each handling a single task. Instead of expanding the entire application, teams may scale specific services independently in response to demand thanks to this modular framework. Consequently, entities are able to distribute resources more effectively and manage workload variations more skillfully. Microservices

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 10 October 2024
Page No: 2521 - 2528
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2024.0346

also encourage better fault isolation. A failure in one service does not always impact the entire application because each one runs separately. Because of this isolation, the impact of errors is reduced and developers can swiftly locate and fix problems without interfering with other system components. Microservices also make continuous integration and deployment procedures easier. Teams can create, test, and implement changes to specific services without impacting the application as a whole since services are decoupled. Because of its agility, enterprises are able to provide updates and new features more quickly by speeding up the development cycle[8]. Furthermore, microservices design promotes flexibility and technological diversity. Teams can take use of the advantages of several technologies inside a single application by having each service built with the best technology stack for its unique needs. This adaptability extends to development procedures as well, since groups can work on several services at once, cutting down on bottlenecks and expediting development. Increased fault tolerance and resilience are two additional benefits of microservices. Organizations can create apps that are more resilient to failures by spreading functionality across several services. Multiple servers or data centers can replicate services, guaranteeing redundancy and high availability. Furthermore, because problems are limited to individual services rather than being entangled with the entire application, microservices facilitate simpler debugging and troubleshooting. Scalability, fault isolation, agility, flexibility, resilience, and fault tolerance are just a few advantages that come with microservice architecture overall. Organizations can create applications that are more resilient, scalable, and flexible to the needs of contemporary computing environments by adopting these principles.

2. Literature Review

An empirical study by Xiang et al.,[9] (2021) explores microservice methods in the industry, revealing 11 real-world difficulties and different levels of maturity. These results provide practitioners with useful recommendations to help align microservice systems with business objectives and capabilities. Furthermore, the study offers scholars a useful resource by emphasizing ways to improve industrial microservice practices. Optimizing container orchestration techniques, removing organizational adoption barriers, and improving microservices architecture's scalability and robustness are among the areas that need more research. By utilizing these discoveries, researchers and practitioners alike can assist in the development and successful implementation of microservices in practical settings, guaranteeing that microservice architectures live up to their claims of autonomy, scalability, and agility in software development. Koleini et al.,[10] (2019) introduces Fractal, an automated application scaling technology that integrates orchestration logic into the program itself. With Fractal, developers can automate scaling operations directly within their applications, in contrast to traditional approaches that divide development from deployment. Fractal, which is based on the Jitsu platform, automates a variety of processes, including network traffic distribution across replicas, replica lifetime management, failure recovery, and scaling up and down. The paper outlines the architecture and salient characteristics of the Fractal implementation and uses a self-scaling website to assess its efficacy. The findings show that Fractal is a novel approach to application scaling in distributed systems, and that it is both possible and useful. This work advances cloud application automation capabilities, expediting the scaling process for increased effectiveness and performance. A modular monolith is used by Faustino et al.,[11] (2022) to study the step-by-step transition of a monolith with a rich domain model to a microservices architecture. For both steps, they assess the effect on migration effort and performance. Although previous research has concentrated on the migration from monoliths to microservices, this study emphasizes the substantial work and performance concerns involved in switching to a modular monolith. Their research highlights the trade-offs between moving to a modular monolith and finishing the full migration process, which offers software architects insightful information. Informed decision-making for architects planning such transitions is facilitated by this research, which adds to our understanding of the

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 10 October 2024
Page No: 2521 - 2528
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2024.0346

difficulties and factors to be taken into account when moving monolithic systems. Pallewatta et al.,[12] (2023) provides MicroFog, a platform for the scalable deployment of microservices based IoT applications in federated fog environments. They emphasize how crucial microservices' scalability and loose coupling are for dispersed deployment among fog and cloud clusters. By supporting MSA application location, dynamic microservice composition, scalability, and heterogeneous application deployment, MicroFog overcomes limitations in current fog computing frameworks. The control engine of the framework provides dynamic composition capabilities, abstracts container orchestration, and runs placement algorithms and applications across federated fog environments. The evaluation's findings show how scalable, extensible, and capable MicroFog is at integrating creative placement strategies and cutting application service response times by as much as 54%. The advancement of fog computing frameworks made possible by this research makes it possible to deploy microservices-based applications in remote IoT environments more effectively. In their 2022 study, Lourenço et al.,[13] examine automated methods for locating potential microservices in monolithic systems. They evaluate 468k decompositions across 28 codebases using five quality indicators, comparing representations based on code structure and development history. The best results are obtained by combining the data from both representations. The significance of taking into account both elements in microservice identification is shown by their discovery that changing authorship representation in codebases with many authors provides results that are comparable to or better than access sequence representation in codebases with few authors. By addressing shortcomings in current methods and offering insights into the efficacy of various representation strategies in determining the best microservice decompositions, this research advances tools for microservice extraction. Wenkai.Lv et al.,[14] (2022) discuss the deployment of microservices in edge computing, taking into account the effects of node loads and fluctuating interaction frequency. They present a multi-objective deployment problem (MMDP) with the goal of minimizing communication overhead while balancing node loads, and they represent microservice invocation as an interaction graph. To tackle MMDP effectively, they present Reward Sharing Deep Q Learning (RSDQL) and suggest Elastic Scaling (ES) for dynamic request management. Their Kubernetes studies show that RSDQL is superior in accomplishing load balancing and faster response times while maintaining elastic service scaling. By guaranteeing scalability and load balancing, while optimizing resource utilization and response times, this work adds insights into microservice deployment strategies in edge computing.

3. Proposed Solution

Traditional monolithic architectures face challenges in scalability, flexibility, and efficient resource utilization. The proposed solution aims to address these issues by implementing a Scalable Cloud Execution Engine using Microservices.

Utilizing microservices architecture involves breaking down the application into small, independent services that communicate through well-defined APIs. Each microservice focuses on a specific function, enabling flexibility, scalability, and easier maintenance.

3.1.Design Patterns

To use microservices, first we need to decide upon the design pattern to be used in our application, this plays a vital role in the efficiency of our application. While applying design patterns we have to ensure that shared access and dependencies have to be managed correctly, data has to be consistent, smooth communication between services, and secure services. There are several design patterns like backend for frontend, keystore value, document value, API Gateway [15] , chain of responsibility, strangler pattern, etc. Our proposed design for a “scalable execution cloud engine” uses an API gateway as a design pattern to facilitate communication between various services. This approach is advantageous for connecting user interfaces to various microservices, this model would be great for enhancing communication between different microservices. Also, using this, we can aggregate the results and send it back to the client easily. Furthermore, the utilization of an API

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 10 October 2024
Page No: 2521 - 2528
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2024.0346

gateway ensures high security by providing a single point of contact, thereby following the security protocols on every aspect of the system. By utilizing an API gateway our system optimizes communication, and data aggregation and also strengthens the security framework, making it an ideal design that can be easily scaled, flexible, and secure.

3.2.Deployment Strategies

Usually, microservice architecture consists of many services, and dependencies that are hard to manage. Also after deployment updates and monitoring needs to be done in any application. So, choosing the correct deployment strategy plays a vital role in the efficiency of the application. There are deployment strategies like single service per host and multiple service per host. Our model uses multiple services per host [16], in this pattern, multiple service instances will share the same server and operating system. This pattern not only provides efficient resource utilization but also scalability and performance can be improved by using this pattern. Furthermore, leveraging this deployment strategy facilitates dynamic scaling and load balancing.

3.3.Communication Mechanism

Microservices are made up of numerous tiny modules, and in order to communicate between services or between processes, they require an appropriate and intelligent channel. Synchronous communication [17] is used in our suggested solution. The request and response method is used in this kind of communication. After submitting a request, the client waits for a response and occasionally even blocks other clients for a predetermined amount of time. All processes must respond promptly and on schedule in order for communication to run smoothly. REST [18] (Representational State Transfer) or HTTP [19] (Hypertext Transfer Protocol) protocols can be utilized for this.

3.4.Proposed Architecture

Our system aims to build a scalable executable cloud engine that uses containerization technologies, Kubernetes orchestration, and Nginx [20] as a reverse proxy and load balancer. This project tackles multiple difficulties with its diverse components, which include front-end services, orchestration services, Kubernetes clusters, and links to an S3 bucket [21] for data storage. First and foremost, the system makes it easier to create an online Integrated Development Environment (IDE) [22] that can run frontend and backend apps together smoothly for longer periods of time. Because users can start a new environment using whatever programming language they choose, adaptability and diversity are guaranteed. In addition, the technology allows servers to automatically scale in response to incoming requests, which maximizes resource usage and performance. Code execution in isolated contexts is essential to this architecture because it increases security and stability and prevents interaction between separate operations. Our project uses WebSockets [23] to allow real-time communication between the client and server. By synchronizing metadata with an S3 bucket, this bidirectional communication mechanism makes it possible to transmit live changes from the frontend to the backend instantly. WebSocket allows for the asynchronous exchange of messages in contrast to regular HTTP, which is unidirectional. This allows for real-time updates without the need for frequent polling or refreshing.

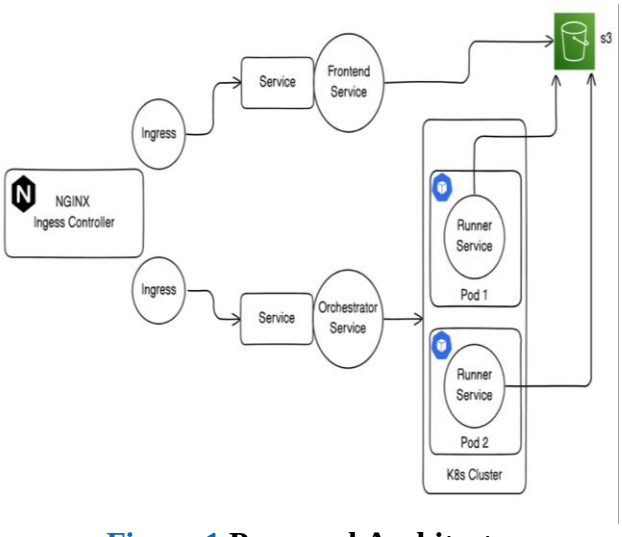

Figure 1 Proposed Architecture

Pseudo-terminals [24] controlled by the Node.js "node-pty" module are used to run commands from the frontend on distant servers. Pseudo-terminals facilitate command execution and provide smooth

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 10 October 2024
Page No: 2521 - 2528
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2024.0346

interaction between the UI and distant services. The architecture is made up of orchestration services that oversee task execution within Kubernetes clusters, reverse proxy and load balancer Nginx, front-end services that function as the user interface, and access to an S3 bucket for data storage. For the overall system design to be scalable, reliable, and efficient, each component is essential. (Refer Figure 1)

4. Implementation

In order to implement our system, we strategically used containerization technology, Kubernetes orchestration, and Nginx for reverse proxy and load balancing. Our cloud execution engine is scalable and efficient because of the well-integrated integration of many technologies.

4.1.Workflow

In our system architecture, a web interface is used by users to communicate with the front-end service. Nginx handles HTTP requests and uses predefined Ingress rules to route them to the correct backend service. By interacting with the orchestration service, the front-end service handles user requests and starts orchestration activities. The orchestration service schedules the task for execution on the Kubernetes cluster after determining the task type (node.js or Java) based on the requests it receives. Pods that run Java and node.js code are deployed and scaled by Kubernetes, which takes workload demands and resource availability into account. These execution pods run the designated code, obtain the required data from the S3 bucket, and produce output. Ultimately, these findings are obtained by the front-end service, which uses the web interface to display them to users.

4.2.Frontend Service

As the primary user interface that makes application interactions possible, the frontend service is essential. It acts as a portal for users to interact fluidly with a range of features and functionalities. Users interact with this interface using a wide range of carefully crafted elements that are intended to improve user experience. These parts consist of necessary components like input fields and buttons that are carefully placed to allow for simple operation and navigation. Uploading code, triggering runs, and visualizing the results are some of the main features provided by the frontend service. Users may now interact with the program with ease thanks to this streamlined procedure, which promotes an easy-to-use and productive workflow. Additionally, the frontend service creates a vital link with the S3 object store, which is an essential part of getting access to the base code needed to run code. This connection makes it possible for the frontend service to easily interact with the S3 object store and retrieve the files required for code execution. The frontend service also makes sure that users have simple access to the resources they require for modification or execution, which improves the application's general usability and functionality by handling the process of fetching files from S3. (Refer Figure 2)

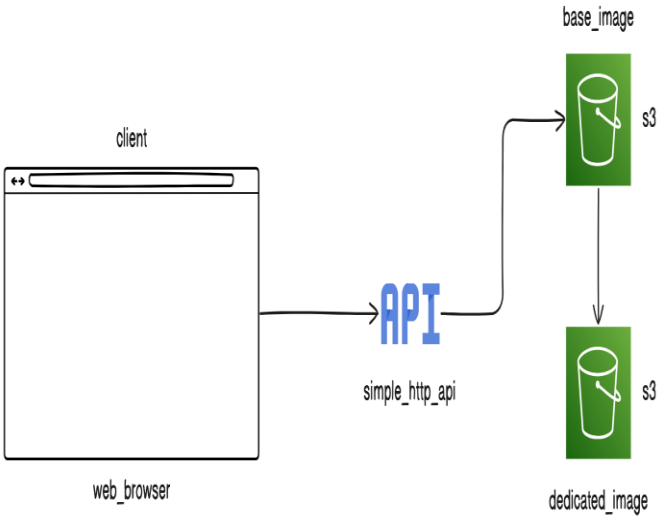

Figure 2 Frontend Service

4.3.Execution Service

The orchestrator service carefully oversees the execution service's deployment procedure to guarantee best use of available resources. With a cluster environment that is both scalable and resilient, the execution service is delivered as pods, utilizing Kubernetes as the underlying architecture. The deployment technique that is orchestrated guarantees optimal resource allocation and containerized instance management. Fundamentally, the execution service takes on the vital duty of supervising the execution of user-submitted code. It coordinates the execution of particular scripts or code snippets in response to requests from the

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 10 October 2024
Page No: 2521 - 2528
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2024.0346

frontend service, configuring the execution environment to meet the demands of the current job. The service guarantees the smooth processing of code given by users by effectively managing the execution environment, which in turn promotes dependable and efficient execution processes. In addition, to facilitate data transfer between the S3 object storage and the frontend service, the execution service uses a pseudo-terminal method. This novel technique enables smooth communication and data interchange, allowing the execution service to transmit important signals and data with the least amount of latency between the object store and the user interface. Through the use of this mechanism, the service maximizes the effectiveness of data transfer procedures, improving the application's overall responsiveness and performance. (Refer Figure 3)

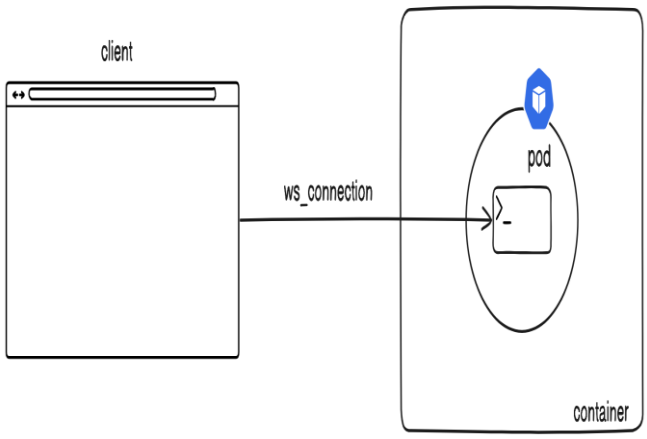

Figure 3 Execution Service

4.4.Orchestrator Service

In the Kubernetes cluster, the orchestrator service is essential to the deployment and administration of the execution service pods. It is in charge of managing the scaling, provisioning, and general upkeep of the execution environment in addition to coordinating different parts of the deployment process. The orchestrator service makes sure that the execution service in the Kubernetes ecosystem runs well by orchestrating these vital tasks. The orchestrator service's primary function is to serve as a mediator between the execution environment and the frontend service. When the frontend service makes requests to begin code execution, it coordinates the launch of the execution service pods, carefully distributing resources and setting up the execution environment to meet the demands of the current task. The orchestrator service makes sure that requests for code execution are processed quickly and efficiently within the Kubernetes cluster by means of good coordination. Additionally, the orchestrator service strengthens the performance and durability of the execution environment by implementing strong methods for fault tolerance and scalability. It maximizes resource consumption and improves the system's overall scalability by dynamically adjusting the number of execution service pods in response to shifting workload needs. The orchestrator service also keeps a close eye on the availability and health of the pods, quickly identifying and resolving any possible issues or disturbances to guarantee the execution service runs consistently and dependably. The orchestrator service protects the integrity of code execution workflows inside the Kubernetes cluster and strengthens the system against future failures by taking these preventative actions. (Refer Figure 4)

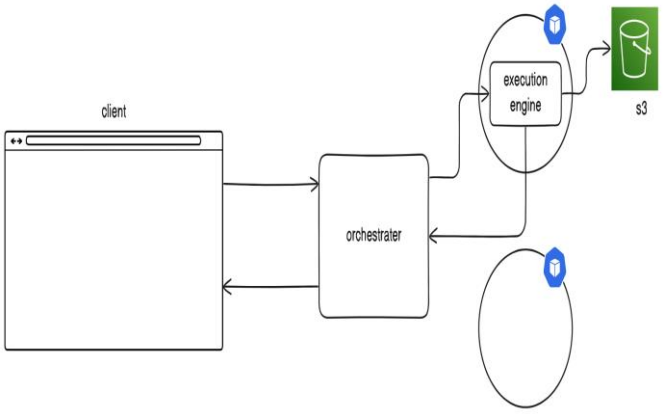

Figure 4 Orchestrator Service

Conclusion

In conclusion, our study highlights how microservices design can be a game-changer when it comes to creating a scalable execution engine. Through breaking down the engine into discrete services, we have explained how this method overcomes the drawbacks of monolithic systems. Numerous advantages come with this paradigm

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 10 October 2024
Page No: 2521 - 2528
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2024.0346

shift, such as improved resilience, scalability, and modularity, which promote effective resource use and easy workload adaptation. Furthermore, the incorporation of containerization technologies and orchestration platforms such as Kubernetes guarantees improved system availability while also expediting deployment processes, providing a solid basis for flexible and adaptable software ecosystems. Because microservices architecture offers unmatched flexibility, scalability, and efficiency in tackling the issues of modern computing environments, our study thus promotes the adoption of microservices architecture as a cornerstone of contemporary software development.

Acknowledgements

The authors wish to thank the Department of Computer Science and Engineering of Sri Krishna College of Engineering and Technology for providing us with continuous support during the development of the project.

References

[1]. https://microservices.io/
[2]. https://www.techtarget.com/whatis/definition/monolithic-architecture
[3]. https://microservices.io/patterns/microservices.html
[4]. https://www.docker.com/resources/what-container/
[5]. https://www.vmware.com/topics/glossary/content/container-orchestration.html
[6]. https://kubernetes.io/
[7]. https://docs.docker.com/engine/swarm/
[8]. https://www.javatpoint.com/software-engineering-software-development-life-cycle
[9]. Xiang, Q., Peng, X., He, C., Wang, H., Xie, T., Liu, D., Zhang, G., & Cai, Y. (2021). No Free Lunch: Microservice Practices Reconsidered in Industry. arXiv:2106.07321 [cs.SE].
[10]. Fractal: Automated Application Scaling, M. Koleini, C. Oviedo, D. McAuley, C. Rotsos, A. Madhavapeddy, T. Gazagnaire, M. Skejgstad, and R. Mortier, 2019, arXiv:1902.09636 [cs.DC]
[11]. Faustino, D., Gonçalves, N., Portela, M., Rito Silva, A. (2022). Stepwise Migration of a Monolith to a Microservices Architecture: Performance and Migration Effort Evaluation. arXiv:2201.07226 [cs.SE].
[12]. Pallewatta, S., Kostakos, V., Buyya, R. (2023). MicroFog: A Framework for Scalable Placement of Microservices-based IoT Applications in Federated Fog Environments. arXiv:2302.06971 [cs.DC].
[13]. Lourenço, J., & Silva, A. R. (2022). Monolith Development History for Microservices Identification: a Comparative Analysis. arXiv:2212.11656 [cs.SE].
[14]. W. Lv et al., "Microservice Deployment in Edge Computing Based on Deep Q Learning," in IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 11, pp. 2968-2978, 1 Nov. 2022, doi: 10.1109/TPDS.2022.3150311.
[15]. https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do
[16]. https://microservices.io/patterns/deployment/multiple-services-per-host.html
[17]. https://www.ringcentral.com/gb/en/blog/definitions/synchronous-communication/
[18]. https://www.codecademy.com/article/what-is-rest
[19]. https://developer.mozilla.org/en-US/docs/Web/HTTP
[20]. https://www.nginx.com/
[21]. https://aws.amazon.com/s3/
[22]. https://www.codecademy.com/article/what-is-an-ide
[23]. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
[24]. https://unix.stackexchange.com/questions/21147/what-are-pseudo-terminals-pty-tty