

Original Article

Cloud-Native Java Applications on AWS ECS and Kubernetes

Venkatesh Satla

Full Stack Lead Java Developer at Innosoul, Inc, USA.

Received: 07-09-2023

Revised: 07-29-2023

Accepted: 08-06-2023

Published: 08-08-2023

ABSTRACT

The paradigm of cloud-native application development has changed the concept of development of modern software systems, taking advantage of the scalability, flexibility, and resilience capabilities provided by cloud computing platforms for creation, deployment, and management. Thanks to its platform freedom, stable ecosystem, security features and a big collection of frameworks to construct large-scale applications, Java remains one of the most popular programming languages in the enterprise world. Containerization technologies also make it easier to deploy Java programs so they execute safely in many diverse environments. Container orchestration systems like Amazon Elastic Container Service (AWS ECS) and Kubernetes are powerful tools that help manage containerized workloads and automate deployment, scaling, networking, and high availability. The paper discusses the design and implementation of cloud native Java apps on AWS ECS and Kubernetes, including architectural, operational and performance issues. The purpose of this study endeavor is to evaluate several platforms in terms of their deployment performance, scalability, resource utilization, management complexity and applicability for enterprise applications. The methodology employed is based on qualitative and comparative approach by literature review, architectural analysis and real implementation scenarios. The results demonstrate that AWS ECS is user-friendly and provides an easy interface to AWS whereas Kubernetes has more flexibility, portability and ecosystem support. The study provides practical insights for enterprises looking for the best orchestration solutions for their cloud-native Java application deployment and modernization strategy.

KEYWORDS

Cloud-Native Applications, Java Microservices, AWS ECS, Kubernetes, Docker, Container Orchestration, Devops, CI/CD, Scalability, Cloud Computing.

1. INTRODUCTION

1.1. Background

Today, cloud computing is evolving at a rapid pace and transforming the face of modern software development. This has allowed enterprises to build applications that are scalable, resilient and highly available. Historically, enterprise systems have been developed with a monolithic architecture, meaning all functionality is closely coupled together inside a single code base. At first it was easy to design a monolithic system but with the increasing application complexity it was difficult to maintain, scale and update. Enterprises have responded with increasingly cloud native designs that offer agility, scalability and continuous delivery.

This was made possible because of Microservice design. Microservices are small independent services that work together to form an application. Each service is developed, implemented and scaled independently. Improved maintainability, fault isolation and productivity of development. Containerization tools such as Docker have also become an important element of cloud native development, in addition to microservices. You may package apps and their dependencies into containers, so they run consistently in development, testing and production settings.

Java is still a popular choice for enterprise applications. It's platform-independent, has a lot of security features, a mature ecosystem and substantial framework support, making it an excellent choice for cloud-native apps. Spring Boot, Quarkus, and Micronaut are frameworks that enable Java developers to build lightweight, scalable microservices that can be deployed fast in containerized environments.

Large scale containerized workloads are managed via the de-facto means of container orchestration systems. The two most popular orchestration technologies nowadays are Amazon Elastic Container Service (AWS ECS) and Kubernetes. AWS ECS is a fully managed solution that is tightly connected with the AWS ecosystem and easy to design and operate. Kubernetes is a highly automated open source platform that allows multi-cloud and hybrid cloud deployment. Both are popular for constructing cloud native Java programs and supporting the rising DevOps approaches.

1.2. Challenges

Cloud-native architectures have many benefits, but developing and running cloud-native Java apps may be anything but simple. One of the key issues is the administration of microservices. The more systems are decentralised the more complicated it is to co-ordinate communication, fix issues and keep consistency over services.

There are other operational issues like service discovery and networking. Microservices need to discover and talk to one other dynamically and require robust networking, load balancing and traffic management. Those misconfigurations may cause performance bottlenecks and service interruptions.

Security is another topic. Organizations need to protect application workloads, secure container images, limit access and meet laws. The security risks could be against the containers or the orchestration systems and even the apps could be vulnerable. Another worry is the use of resources and control of expenditure. Proper allocation of CPU, memory and storage resources in containerized settings is crucial to minimize underutilization or over-provisioning of cloud resources. Often the trade-off between performance and cost-effectiveness is defined by hard judgments.

In cloud native setups, Monitoring and Observability are just as critical. Distributed applications produce a lot of logs, metrics and traces that need to be gathered and analyzed, to maintain the system's reliability and performance. Inadequate monitoring approaches to diagnose and fix problems.

Organizations are also increasingly choosing multi-cloud and hybrid-cloud strategies, which complicate adoption and maintenance even more. Containerization, memory use and startup latency can significantly impact the performance of cloud native Java applications and require tuning for optimal performance.

1.3. Problem Statement

As cloud-native matures, organizations struggle to pick the best container orchestration platform for Java workloads. Both AWS ECS and Kubernetes are great at orchestrating containerized workloads but are quite different in their architecture, operational complexity, scalability, portability and ecosystem support.

There are several platforms where organizations may not be aware of the strengths and limits of the various platforms. This uncertainty can lead to expensive deployment decisions, increased operational expenses and difficulty in sustaining systems that are scalable and resilient. The use of Java based microservices for organizations is often a tradeoff between portability, performance, automation and resource utilization. Therefore, organizations must know AWS ECS and Kubernetes inside out to make wise judgments on building cloud-native Java applications.

1.4. Motivation

Cloud based technologies are spreading through all industries and there is an enormous need for scalable, resilient and performant enterprise applications. Organizations are shifting to microservices architectures and replacing outdated systems to become more agile and faster with software development. Java is still a leading technology for enterprise development and knowing how to successfully construct cloud native Java apps is critical.

The necessity for automation, fault tolerance, operational simplicity and cost effectiveness has also lead organizations to start exploring solutions. Both of the major orchestration platforms, AWS ECS and Kubernetes, have achieved these aims but in distinct ways. Understanding capabilities,

deployment patterns, and operational aspects helps firms improve their cloud native approach. The purpose of this work is to provide a comparative overview of different platforms and best practices to deploy and manage cloud native Java applications in modern cloud environments.

1.5. Objective of the Study

The aim of this inquiry is to:

- Learn architecture and functioning of cloud native Java programs.
- Learn about AWS ECS and Kubernetes deployment options.
- Comparison of operational characteristics, managerial capability and ecosystem support of the two systems.
- To test various performance, scalability, stability and maintainability features of cloud native Java applications on AWS ECS and Kubernetes.
- The purpose is to provide ideas about the choice of the correct orchestration platforms for specific business needs and deployment scenarios.

2. LITERATURE REVIEW

2.1. Cloud-Native Computing Foundations

What is Cloud Native Computing? Cloud native computing is the practice of building and running programs that fully use the advantages of the cloud computing delivery model. Native cloud solutions enable companies to build and run scalable, resilient and maintained applications in modern dynamic environments. These are public, private and hybrid cloud environments. At the end of the day, it's all about automation, elasticity, and continuous supply and fault tolerance. The Twelve-Factor App approach is a great fit for cloud-native development because it captures best practices in configuration management, dependency isolation, stateless processes, and environment portability. The usage of these concepts will allow companies to build software applications that can be reliably deployed across a range of cloud infrastructures.

2.2. Java Support for Cloud-Native Application Development

Java is a very developed language for cloud-native requirements. With the usage of Spring Boot, the process of building corporate applications that contain embedded servers, auto-configuration, and production-ready features has gotten easier. Other capabilities included in Spring Cloud include service discovery, distributed configuration, circuit breakers and support for API gateways for microservices. Moreover, Quarkus and Micronaut are two examples of contemporary lightweight frameworks that have been built for container and Kubernetes systems. So the deployment of the cloud has been more efficient, memory consumption has been reduced and starting time has been enhanced. Though Jakarta EE is still maturing, the industry standards and APIs are suitable for today's cloud native apps.

2.3. Containerization Is One Of The Most Crucial Technologies For Cloud Native Development.

Containerization is also called containerization. Docker is the most popular container platform that packages applications and their dependencies into lightweight and portable chunks. Docker is made up of image files, containers, registry entries and the Docker Engine. Containers go through a lifespan which comprises the following stages: production, deployment, execution, scaling, and termination. Containers are faster to start up, more resource efficient, more mobile and more scalable than traditional virtual machines. The resource consumption rate is likewise lower with containers. Such characteristics make containers attractive options for delivering Java microservices in cloud environments.

2.4. Microservices Architectures

Micro services design breaks applications into distinct services that communicate with one another through clearly defined interfaces. Apart from fault isolation and scalability, its main qualities are loose coupling and autonomous deployment. Decomposing services is often a process that uses business skills to allow teams to develop and manage services on their own. Microservices typically connect with each other over REST APIs, gRPC, or event-driven message systems. API Gateways streamline the interaction between clients and dispersed services and provide a single point of access through managing routing, authentication, rate limitation, and traffic management. This is obtained using traffic management control.

2.5. Deploy the Application with Kubernetes

Currently, Kubernetes is the standard container orchestration that is widely utilized. Its design is centered on control plane components and worker nodes that work together to run and deliver applications. Kubernetes supports the smallest deployable units called Pods. Deployments take care of replication and upgrading apps. Various services provide connectivity to the network and load balancing to the Pods. Ingress Controllers are the gatekeepers for traffic arriving from outside. Horizontal Pod Autoscaling is a feature that automatically scales the number of instances of an executing program in response to the requirements of the workload. Java features that simplify deployment and administration of cloud-native apps.

2.6. Amazon Elastic Container Service (Ecs)

Amazon Elastic Container Service (ECS) is a fully managed container orchestration service that allows you to build and manage containerized applications on Amazon Web Services (AWS). ECS architecture is composed of clusters, task definitions, services, and tasks. Task descriptions contain specification of container attributes, resource needs and networking setups. The combination of ECS and AWS Fargate is incredibly powerful. Run containers without thinking about the servers. Additionally, ECS may be auto-tuned to the number of resources used and to the workload requirements, which makes the administration of enterprise application operations much easier.

2.7. Evaluates ECS And Kubernetes.

There have also been several comparisons between Kubernetes and Amazon Elastic Container Service (ECS). ECS is often thought to be simpler when it comes to deployment and administration especially for companies who are deeply associated with AWS services. Kubernetes also has more ecosystem support, in addition to being more portable and versatile. Kubernetes is more multi-cloud, ECS has less operational complexity, research says. Resource utilization, cost-effectiveness, scalability and deployment management are also considered in comparisons. On the other hand, the observations usually vary depending on the circumstances of the job and the requirements of the firm.

Table 1: Comparison of AWS ECS and Kubernetes

Feature	AWS ECS	Kubernetes
Deployment Complexity	Simple	Moderate to High
AWS Integration	Native Integration	Requires Configuration
Multi-Cloud Support	Limited	Excellent
Operational Overhead	Low	Higher
Scalability	Good	Excellent
Portability	AWS-Centric	Platform Independent

2.8. Research Deficits

Container orchestration technologies are well-researched however there is a lack of studies on cloud native Java workloads run on AWS ECS and Kubernetes. Much of what you see here is just generic container management, it has nothing whatsoever to do with Java speed. Moreover, there are limited actual deployment case studies and comprehensive reviews with a focus on performance. Because of these constraints, there is a need to perform focused research on the deployment efficiency, scalability, resource utilization, and operational challenges of Java-based cloud-native applications on both the orchestration systems.

3. PROPOSED METHODOLOGY

3.1. Research Framework

The aim of this research is to compare experimental methods to evaluate the deployment and operation features of cloud-native Java applications written using Amazon Elastic Container Service (ECS) and Kubernetes. This study effort is based on the design science technique. More concretely, a realistic cloud-native application is built, developed, deployed and analyzed on both of the required orchestration platforms. The strategy mainly focuses on differentiating the changes in complexity of deployment, performance, scalability, dependability and cost effectiveness. To make a fair comparison, the same application architecture and workload should be used in both environments at the same time. Systematic testing and monitoring make it possible to get the results of the experiments, and provide an objective evaluation of the capabilities and limitations of each platform for the execution of corporate Java tasks.

3.2. System Architecture Design

The created cloud-native solution is based on microservice architecture. The architecture is composed of a set of independent services running in containerized environments. The architecture is built upon several fundamental layers. A client layer is: The Client Layer is the point of connection for users and programs outside of the system to interact with the system. Access via web and mobile devices. An API Gateway is responsible for routing traffic, authentication, authorization, and load balancing of requests. This infrastructure will be used to route requests.

Users can also take advantage of the Java Microservices Layer which has services like User Service, Product Service, Order Service, etc. It is possible to implement these services in a separate location. So, we use Spring Boot to construct these services and use REST APIs to communicate between them inside.

A Service Registry is used by microservices to find services dynamically and communicate with one another. This will allow services to discover one another and discover one another in distributed environments. The Database Layer handles the storage of application data into PostgreSQL, providing reliable and persistent data handling. It might be summarized that Prometheus and Grafana are the parts of the Monitoring Layer. These two technologies are responsible for the gathering, processing and visualization of application and infrastructure metrics. These are used for monitoring of performance and operational visibility in most circumstances.

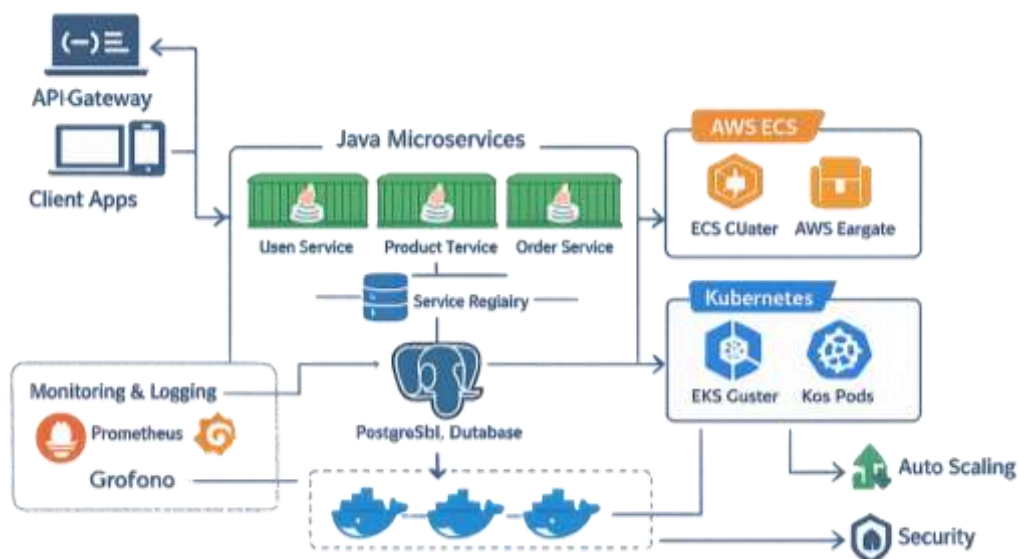


Fig 1- Cloud-Native Java Application Architecture on AWS ECS and Kubernetes

3.3. The Third Stack of Tech

The technique that has been suggested employs a contemporary cloud-native technology stack that is tailored toward the construction and deployment of Java applications as well as other similar programs. The programming language used is Java 21. It was chosen for its long term support features

and enhanced performance. When it comes to building microservices, Spring Boot is by far the most widely utilized framework. Docker has containerization, so applications may be delivered in a consistent manner across environments.

In the orchestration process we choose to use Amazon Elastic Container Service Fargate as the managed container deployment platform. On the other hand, the Kubernetes deployments are done through the usage of Amazon Elastic Kubernetes (EKS). Currently the relational database management system is PostgreSQL. We automate the processes of continuous integration and continuous deployment (CI/CD) with the help of GitHub Actions. Monitoring is done with Grafana and observability is done with Prometheus.

3.4. Application Development Process Section

When it comes to development, there are four main stages. In the first phase of the application development process, the application will be planned for development in a microservices architecture. This design has three components: User Service, Product Service and Order Service. Each service is developed and tested in isolation with the Spring Boot framework.

The second phase is talking through the process of containerization. Image optimization techniques are used to reduce the size of the images and to increase the performance of the starting process. Dockerfiles are produced for every microservice and optimization techniques are used to the images. The created Docker images are kept in a container registry for future deployment.

In the third stage, we use Amazon Elastic Container Service (ECS). AWS Fargate is used for creating ECS clusters, defining job descriptions, and running services. "Auto-scaling rules dynamically scale resources based on a change in workload requirements.

The fourth stage is deploying Kubernetes using Amazon Elastic Container Service (ECS). Deployment manifests are also meant to offer information about the resources used by an application. Networking and external access is handled by the Ingress Controllers and Kubernetes Services. ConfigMaps and Secrets are used to manage configuration parameters and store credentials securely. Horizontal Pod Autoscalers are meant to support automatic scaling of application instances based on resource consumption and traffic patterns.

3.5. The Evaluation Metrics

There are many evaluation metrics used in comparing Amazon Elastic Container Service (ECS) versus Kubernetes. Performance can be measured in many ways. Reaction time, throughput, latency, etc. Scalability measures quantify the level of activity and resource consumption that can be scaled up or down according to different workloads. Metrics of dependability include the availability (length of time that the system is available) and mean time to recovery from problems. In the process, costs measurements are used to evaluate infrastructure costs and operational overheads connected with the

administration of each platform. Some measurements are available to quantify the platform efficacy and to give insights into its performance.

3.6. Experiment Settings

To ensure consistency across all installations, we run the testing in a particular AWS region. To guarantee an appropriate comparison, we configure the same cluster parameters for both the ECS and EKS options. The workload generation has been carried out by using the industry standard testing tools, such as Apache JMeter and Gatling. These apps can replicate the user traffic patterns with the maximum feasible accuracy. The objective of Grafana dashboards is for data visualization while the purpose of Prometheus is for gathering the monitoring and performance data. This experimental setup allows for a detailed examination of the deployment efficiency, application performance, scaling behavior, reliability attributes and overall operational cost of competing orchestration platforms.

4. CASE STUDY

4.1. Case Study Overview

The case study is a microservice based application for e-commerce and it tries to analyse the performance of Amazon Elastic Container Service (ECS) and Kubernetes for cloud native Java applications. Business typically has to deal with the difficulty of scalable, robust and highly available e-commerce platforms, to satisfy the ever changing wants of clients. The program allows the consumer to register, find things, order, and pay and do many other vital company functions. The most crucial aspects for a business include fast response times, fault tolerance, and secure transactions, limitless scalability during high traffic periods and constant availability of the service. The application satisfies these requirements and is therefore a good quality candidate to test container orchestration solutions in a genuinely cloud native environment.

The system consists of several functional modules, including customer registration and authentication, product catalog management, order processing, and payment processing. Each module is designed to be a separate microservice, allowing for modular development and deployment.

4.2. Application Architecture and Layout

The application presented is based on microservices architecture, created in Java 21 with Spring Boot. Our entire service architecture is built on four cores.

The User Management Service manages the registration, authentication, profile generation and authorization of the customers. The Product Catalogue Service is responsible for storing product information, searching for products, inventory information, administration and category management. The Order Processing Service takes charge of the customer's orders from generation and validation to tracking features. It also talks to other services to do transactions. The Payment Service handles payment requests, validates payment data, and establishes secure financial transactions by communicating with third-party payment gateways if necessary.

All services are using Rest API. All services are containerized. The backend database used is PostgreSQL database. We are utilizing Prometheus and Grafana for monitoring and observability in our org. This design allows you to release, grow, and operate each service independently.

4.3. Version of ECS

The program is first hosted on AWS Fargate, a container service from Amazon Web Services (ECS). To start the deployment process, we need to build various Docker images of each microservice. The optimized Docker images are the ones we build and push to the Amazon Elastic Container Registry (ECR). Next we create an ECS cluster to host the app workloads.

Task Definitions allow you to specify container images, CPU and RAM allocations, networking configurations and environment variables. This service schedules the tasks and makes sure that the right number of instances of the application are executing. AWS Fargate is a computation engine that eliminates the need to manage virtual machines or server architecture. ALB (Application Load Balancers) are those that give availability by dispersing incoming traffic over a large number of service instances.

There are several advantages to the deployment. ECS provides a simple deployment experience that is tightly integrated with Amazon Web Services (AWS) services. With Fargate you don't have to handle the underlying compute resources so you can spend less time maintaining infrastructure. The development teams can deploy swiftly without the overhead of running a platform.

4.4. Kubernetes Deployment

Next, the same application previously launched is deployed using Amazon Elastic Kubernetes Service (EKS). The first step in the deployment process is to construct an EKS cluster and set up worker nodes. We describe the Kubernetes deployment manifests for each micro-service. These manifests describe the Pods, Deployments, Services, ConfigMaps and Secrets;

Services of Kubernetes, including internal network and service discovery Ingress controllers manage routing traffic from outside. There is a service mesh solution to help you boost traffic management, observability and security between micro services. Horizontal Pod Autoscalers can be designed to automatically increase/decrease the number of instances of applications based on cpu utilization and the demand for the task.

Kubernetes Deployment is more flexible and versatile in comparison to Amazon Elastic Container Service (ECS). Organizations can gain by enhancing application management, networking, security policies and scalability performance. However, operational complexity of this flexibility requires a greater level of competence in cluster management, configuration management and maintenance tasks.

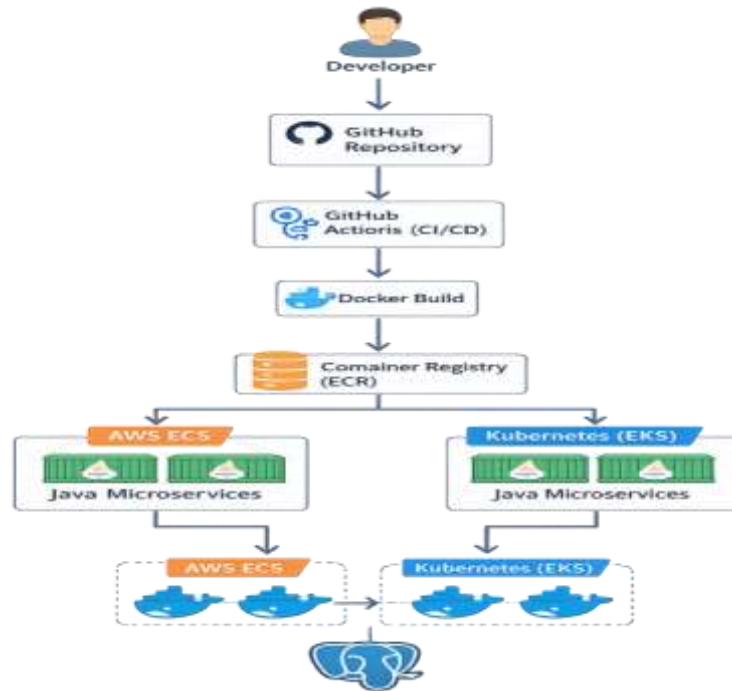


Fig 2 - CI/CD Deployment Workflow for Cloud-Native Java Microservices on AWS ECS and Kubernetes

4.5. Contrast

The two strategies are implemented very differently. ECS is a managed infrastructure design and integrated Amazon Web Services model with less setup labor. Kubernetes does take a bit more work to build and maintain but it gives you a lot more flexibility and control of the platform.

In Kubernetes you have to observe the cluster, you have to manage the cluster. ECS is a little more hands off on monitoring your cluster. Both have dynamic scalability, but Kubernetes has more robust scaling and workload management capabilities. Both platforms provide scalability and maintainability. All systems have great monitoring possibilities with Prometheus and Grafana in the mix but Kubernetes is more flexible on the observability modification. In a nutshell, although Elastic Container Service (ECS) provides a quick and simple way to use, Kubernetes gives additional flexibilities, mobilities and deep orchestrations for cloud native Java applications.

5. RESULTS AND DISCUSSION

5.1. Performance Evaluation

We tested the cloud-native Java e-commerce application performance by executing the same workloads on AWS ECS Fargate and Amazon EKS. Performance evaluations were taken considering response time, throughput and application delay under different traffic scenarios. Results indicate that both platforms performed satisfactorily for enterprise workloads. AWS ECS has somewhat lower response times during the initial deployment due to its reduced networking configuration and managed infrastructure. Kubernetes adds a little orchestration overhead, but never showed

performance degradation under sustained workloads and allowed us more flexibility over resource allocation.

The response time research showed that ECS was able to set up and deploy faster but Kubernetes was more stable in the case of high-volume traffic. However, the variations were rather slight, and mostly depended on cluster setup and workload factors.

5.2. Scalability Analysis

Load testing and simulation of a growing number of concurrent users was accomplished using JMeter and Gatling. Both ECS and Kubernetes grew application instances to manage growing traffic demands with no notable service disruption.

Table 2: Scalability Performance Comparison of AWS ECS and Kubernetes

Concurrent Users	ECS Response Time	Kubernetes Response Time
100	120 ms	125 ms
500	210 ms	195 ms
1000	380 ms	340 ms

Based on the data, it is decided that ECS is an efficient option for managing low levels of traffic, while Kubernetes is more scalable for managing higher levels of traffic. Kubernetes has strong scheduling algorithms and Horizontal Pod Autoscaling features that allow it to distribute workloads across the resources available in the cluster more efficiently. ECS's auto-scaling capabilities are as stable as Kubernetes', but they have less customization choices.

5.3. Resource Utilization Analysis

The use of resources was evaluated according to the amount of central processing unit (CPU) and memory capacity used by the application services. AWS Fargate supplied Elastic Compute Cloud (ECS) with the automatic management of infrastructure resources, which allowed for optimal distribution of resources without requiring administrative input. (ECS) is an Elastic Compute Cloud. This made the utilization of resources predictable, but also reduced the fees involved in administrative activities.

Because of the control plane and cluster management components, the original resource overhead of Kubernetes was a little higher than other containers. Kubernetes, however, provides a more polished method around managing resource demands, capacity limitations, and task scheduling. Kubernetes gave a better optimization of CPU use among all the instances of an application shown in response to an increase in the traffic volume of an application. The memory use was similar on both the systems but Kubernetes had more granularity in resource allocation. The metrics findings for pod and task efficiency showed that they were equal, but that Kubernetes was more adaptive because of the optimization features it possessed.

5.4. Reliability and Fault Tolerance Evaluation

The main areas of concern in reliability testing were service availability, failover performance, and recovery mechanisms. Recovery procedures also were considered. Both platforms are able to provide high application availability in normal operation and in simulated failure scenarios. Both platforms have this functionality in common. With the ECS system, jobs that were failing were immediately replaced and demand was dispersed among the available resources in a different way. Kubernetes has a lot of self-healing features, including the ability to automatically replace pods and analyze how healthy they are.

Based on the results of the failover testing, Kubernetes showed that it was able to recover from a service outage in a little more expedient way. It was because it has more advanced orchestration and scheduling features. Both systems had availability rates that were above 99.9% which meant that they were suitable for use in applications developed for enterprise-level circumstances. Elastic Container Service (ECS) was easier to use and deploy, while Kubernetes had more flexibility in its technique of recovery configuration.

5.5. Cost and Operational Efficiency Analysis

In the cost study, all expenses connected to the infrastructure, operating costs and managerial overhead were considered. Amazon Web Services Infrastructure Cloud (ECS) has fewer operational costs as Amazon Web Services (AWS) was in charge of managing most of the infrastructure requirements. Also, it would allow for rapid application deployment without the need for extensive experience in platform administration.

As a result of Kubernetes, there was an increase in the operating expenses associated with cluster management, monitoring, upgrades, and maintenance. The infrastructure costs, however, were almost the same for workloads similar to other apps, despite the fact that Kubernetes required more admin labor and specialized knowledge than other programs. This could minimize these costs for companies operating in hybrid or multi-cloud settings through increased portability and less dependency on a single provider. However, enterprises operating in such contexts may be able to minimize these expenses.

5.6. Comparative Findings and Discussion

The findings of the studies suggest that both of the orchestration systems seem to have significant advantages over each other. Benefits of Amazon Web Services Elastic Container Service (ECS) include simplified management, faster deployments, and less operational overhead. If your firm has already made investments into the Amazon Web Services (AWS) ecosystem, then the direct connectivity with AWS services is a feature you should consider being very tempting. With the managed infrastructure model, the teams may spend more time developing the application than managing the platform, which would be their main priority without this model.

Kubernetes offers a wide array of features, including support for a broad ecosystem, platform independence, and strong orchestration. It is a great solution for complex corporate systems. It has extensive scheduling capabilities, flexible networking, integration with service mesh and improved autoscaling features.

This comparison points to a big compromise. The focus of ECS is operational efficiency and simplicity, while Kubernetes is focused on flexibility and adaptability. Here we are comparing these two concepts. The mobility of Kubernetes is better across cloud providers yet Elastic Container Service (ECS) has stronger integration with Amazon Web Services (AWS). Running ECS is often less complicated in a similar way. Kubernetes on the other hand can have a lot more admin costs but the control and extensibility of Kubernetes is greater. Therefore, the firm needs to choose the platform that matches the technical requirements, operational capabilities, scalability and long-term cloud strategy of the organization.

6. CONCLUSION AND FUTURE SCOPE

6.1. Conclusion

The work explored the deployment of cloud native Java programs on two important container orchestration technologies, AWS Elastic Container Service (ECS) and Kubernetes. The study explored the cloud-native design concepts, the Java microservice development, the containerization technologies, the deployment methodologies and the operational features of the two platforms. The results show that AWS ECS offers a more straightforward deployment process, less infrastructure overhead to maintain, and a smooth integration with other AWS services. Kubernetes nonetheless offers more flexibility, portability, highly powerful orchestration features and a vibrant community environment. The research objectives have been met by designing, installing and assessing a cloud native e-commerce application to enable broad comparison of performance, scalability, reliability and cost issues. The results suggest that ECS is a good fit for enterprises that need operational simplicity, whereas Kubernetes is preferable for companies that need deep customization and multi-cloud mobility. The insights offer practical advice for companies building corporate Java applications for today's cloud environments.

6.2. Future Work

In this paper, we offer a practical deployment architecture for Java microservices on cloud and systematic way of evaluating container orchestration solutions. It gives an insight to the decision making process on how to choose the best deployment solutions depending on the demands of the organization. In future we will further enhance the cloud native operations by including serverless containers, AI based autoscaling, adopting service mesh and multi-cluster deployment. More research is needed on edge-native Java apps, hybrid-cloud orchestration models, sustainability and green cloud computing techniques and security-oriented cloud-native architectures. These new areas can provide improvements in scalability, efficiency, resilience and security and would enable continued advancement of cloud native Java application development and deployment.

REFERENCES

- [1] Mahajan, A., & Gupta, M. K. (2018). *Cloud-Native Applications in Java: Build microservice-based cloud-native applications that dynamically scale*. Packt Publishing Ltd.
- [2] Kratzke, N., & Peinl, R. (2016, September). Clouds-a cloud-native application reference model for enterprise architects. In *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)* (pp. 1-10). IEEE.
- [3] Parakala, A. (2021). Building Analytics-Driven Bots: RPA Meets Business Intelligence. *International Journal of Emerging Research in Engineering and Technology*, 2(1), 77-87. <https://doi.org/10.63282/3050-922X.IJERET-V2I1P109>
- [4] Marchioni, F. (2019). *Hands-on Cloud-native Applications with Java and Quarkus: Build High Performance, Kubernetes-native Java Serverless Applications*. Packt Publishing Ltd.
- [5] Srigadde, B. R. (2023). The Hidden Gem: Lightning Headless Component. *International Journal of Emerging Trends in Computer Science and Information Technology*, 4(1), 244-254. <https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P125>
- [6] Kertész, D. R., Farkas, K., & Szabó, G. (2021). Best Practices of Cloud Native Application Development. *Bachelor of profession's thesis, Budapest University of Technology and Economics, Budapest*.
- [7] Vppalapati, M., & Talasila, P. K. (2021). Failure without Faults: How Enterprise Storage Systems Degrade Long Before They Break. *International Journal of Emerging Trends in Computer Science and Information Technology*, 2(1), 115-123. <https://doi.org/10.63282/3050-9246.IJETCSIT-V2I1P113>
- [8] Kumar Doodala, A. N. (2023). Offline-First Android Architecture for waste management in low connectivity zones. *International Journal of Emerging Trends in Computer Science and Information Technology*, 4(1), 201-209. <https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P121>
- [9] Domingus, J., & Arundel, J. (2022). *Cloud Native DevOps with Kubernetes*. "O'Reilly Media, Inc."
- [10] Gaddam, R. R. (2022). Cost-Aware Autoscaling for Batch vs. Online Inference. *International Journal of Emerging Trends in Computer Science and Information Technology*, 3(4), 134-143. <https://doi.org/10.63282/3050-9246.IJETCSIT-V3I4P113>
- [11] Arundel, J., & Domingus, J. (2019). *Cloud Native DevOps with Kubernetes: building, deploying, and scaling modern applications in the Cloud*. O'Reilly Media.
- [12] Muppaneni, K. (2022). Comparative Analysis of Client-Side Storage Mechanisms. *International Journal of AI, BigData, Computational and Management Studies*, 3(1), 171-182. <https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I1P119>
- [13] Vitale, T. (2022). *Cloud Native Spring in Action: With Spring Boot and Kubernetes*. Simon and Schuster.
- [14] Allenki, S. S., & Korutla, R. (2021). Agile Development in Practice: From Intern to Contributor. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, 2(3), 91-103. <https://doi.org/10.63282/3050-9262.IJAIDSML-V2I3P110>
- [15] Vppalapati, M., & Talasila, P. K. (2021). Failure without Faults: How Enterprise Storage Systems Degrade Long Before They Break. *International Journal of Emerging Trends in Computer Science and Information Technology*, 2(1), 115-123. <https://doi.org/10.63282/3050-9246.IJETCSIT-V2I1P113>
- [16] Fugaro, L., & Vocale, M. (2019). *Hands-On Cloud-Native Microservices with Jakarta EE: Build Scalable and Reactive Microservices with Docker, Kubernetes, and OpenShift*. Packt Publishing Ltd.
- [17] Muppaneni, R. K. (2022). Data Privacy in the Age of AI: How Dynamics 365 Handles Regulatory Challenges. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, 3(4), 159-170. <https://doi.org/10.63282/3050-9262.IJAIDSML-V3I4P117>
- [18] Vanapalli, Satyendra Kumar. "Bridging Business and Technology: The Role of CRM in Digital Transformation." *International Journal of Artificial Intelligence & Digital Transformation* 5.2 (2022): 01-17.
- [19] Sartoni, M. (2022). *AWS Services for Cloud Robotics Applications* (Doctoral dissertation, Politecnico di Torino).
- [20] Gaddam, R. R. (2021). Vertex AI as a Unified Control Plane for MLOps. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, 2(2), 92-102. <https://doi.org/10.63282/3050-9262.IJAIDSML-V2I2P110>
- [21] Shiramalla, R. (2022). Design of a Unified API Interface Using Workato for Cross-Platform Data Orchestration Between Salesforce and Oracle ERP. *International Journal of Emerging Trends in Computer Science and Information Technology*, 3(1), 157-168. <https://doi.org/10.63282/3050-9246.IJETCSIT-V3I1P118>
- [22] Kommuru, Madhurima. "Concurrency is hard: Java vs Python pitfalls you can't ignore." *International Journal of Modern Research in Science & Engineering* 5.2 (2022): 01-18.

- [23] Carpenter, J., & McFadin, P. (2022). *Managing cloud native data on Kubernetes: architecting cloud native data services using open source technology*. "O'Reilly Media, Inc."
- [24] Muppaneni, R. K. (2020). Retail Reimagined: How Dynamics 365 Commerce Is Driving Omnichannel Experiences. *International Journal of AI, BigData, Computational and Management Studies*, 1(1), 49-59. <https://doi.org/10.63282/3050-9416.IJAIBDCMS-V1I1P106>
- [25] Muppaneni, R. K. (2022). Data Privacy in the Age of AI: How Dynamics 365 Handles Regulatory Challenges. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, 3(4), 159-170. <https://doi.org/10.63282/3050-9262.IJAIDSML-V3I4P117>
- [26] Vanapalli, Satyendra Kumar. "AI-Driven Fraud Detection in Financial Systems: A CRM-Centric Approach." *International Journal of Machine Learning and Predictive Analytics* 5.2 (2022): 01-14.
- [27] Sethi, M. (2017). *Cloud Native Python*. Packt Publishing Ltd.
- [28] Katangoori, S., & Katangoori, A. (2021). AI-Augmented Data Governance: Enabling Intelligent Access, Lineage, and Compliance across Hybrid Clouds. *American International Journal of Computer Science and Technology*, 3(6), 36-45. <https://doi.org/10.63282/3117-5481/AIJCSST-V3I6P104>
- [29] Vppalapati, M. (2022). The Storage Stack Nobody Draws: Cabling, Panels, and the Illusion of Isolation. *International Journal of Emerging Research in Engineering and Technology*, 3(2), 211-220. <https://doi.org/10.63282/3050-922X.IJERET-V3I2P121>
- [30] Suryadevara, S. S. K. (2022). Knowledge-Graph-Enabled Tagging and Taxonomy Automation Framework. *American International Journal of Computer Science and Technology*, 4(1), 77-89. <https://doi.org/10.63282/3117-5481/AIJCSST-V4I1P108>
- [31] Saleh, A., & Karslioglu, M. (2021). *Kubernetes in Production Best Practices: Build and manage highly available production-ready Kubernetes clusters*. Packt Publishing Ltd.
- [32] Allenki, S. S. (2022). Securing Databases in the Cloud with RBAC and Encryption Best Practices. *International Journal of Emerging Research in Engineering and Technology*, 3(3), 173-182. <https://doi.org/10.63282/3050-922X.IJERET-V3I3P117>
- [33] Kumar Doodala, A. N., & Thatraju, S. (2022). NLP-Driven Benefits Interpretation Engine for Personalized Member Communication. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, 3(1), 173-183. <https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P118>
- [34] Saeed, L., & Abdallah, G. (2022). *Pro Cloud Native Java EE Apps*.
- [35] Muppaneni, K. (2021). HTTP/3 & REST Latency Improvement. *International Journal of Emerging Research in Engineering and Technology*, 2(1), 122-132. <https://doi.org/10.63282/3050-922X.IJERET-V2I1P113>
- [36] Suryadevara, S. S. K., & Polinati, A. K. (2022). Cross-Cloud Governance Engine Using Policy-as-Code for CMS Platforms. *International Journal of Emerging Research in Engineering and Technology*, 3(4), 165-175. <https://doi.org/10.63282/3050-922X.IJERET-V3I4P118>
- [37] Srigadde, B. R. (2020). When Force Is With You but Not Lightning Component. *American International Journal of Computer Science and Technology*, 2(1), 23-33. <https://doi.org/10.63282/3117-5481/AIJCSST-V2I1P103>
- [38] Khatri, A., & Khatri, V. (2020). *Mastering Service Mesh: Enhance, secure, and observe cloud-native applications with Istio, Linkerd, and Consul*. Packt Publishing Ltd.
- [39] Katangoori, S., & Deore, S. (2022). Edge-Cloud Hybrid Data Pipelines: Architectures for Federated Analytics and Learning. *American International Journal of Computer Science and Technology*, 4(3), 20-34. <https://doi.org/10.63282/3117-5481/AIJCSST-V4I3P103>
- [40] Parakala, A. (2022). Integrating Salesforce and UiPath: Cross-System Intelligent Automation. *International Journal of Emerging Trends in Computer Science and Information Technology*, 3(4), 88-99. <https://doi.org/10.63282/3050-9246.IJETCSIT-V3I4P109>
- [41] Leymann, F., Breitenbücher, U., Wagner, S., & Wettinger, J. (2016, April). Native cloud applications: why monolithic virtualization is not their foundation. In *International Conference on Cloud Computing and Services Science* (pp. 16-40). Cham: Springer International Publishing.