International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Index Copernicus Value (2016): 79.57 | Impact Factor (2017): 7.296

Navigating Legacy to Modern: Container Orchestration Strategies, Pitfalls, and Best Practices for Applications

Savitha Raghunathan

Email: *saveetha13[at]gmail.com*

Abstract: ***This white paper delves into the transformative journey from traditional legacy IT architectures to modern containerized systems. It aims to provide a comprehensive guide on adopting container orchestration technologies, highlighting the various strategies, pitfalls, and best practices. By examining the transition to containers and the evolution of orchestration tools, this whitepaper offers insights for organizations to effectively navigate the complexities of modernizing their IT infrastructure for improved agility, efficiency, and scalability.***

Keywords: Cloud transformation, Virtual Machines, Containers, Containerization engine, Modernization

1. Introduction

In the rapidly evolving technological landscape, organizations are facing increasing challenges with traditional legacy architectures. These monolithic systems, typically deployed on physical servers or virtual machines, struggle to meet modern software development and delivery demands. This paper begins by defining traditional IT architecture and its inherent limitations, setting the stage for the necessity of containerization as a means to address these challenges. It then explores the beginning of container technologies and the subsequent development of container orchestration tools, essential for managing containerized applications at scale.

2. Traditional IT Architecture

Traditional IT architecture is characterized by monolithic application design, where all functionalities are tightly integrated into a single, indivisible unit. As depicted in Fig.1, these monolithic applications are deployed on dedicated physical servers or virtual machines [4]. This typical approach led to several issues, including scalability challenges, lengthy deployment cycles, and difficulty updating features.

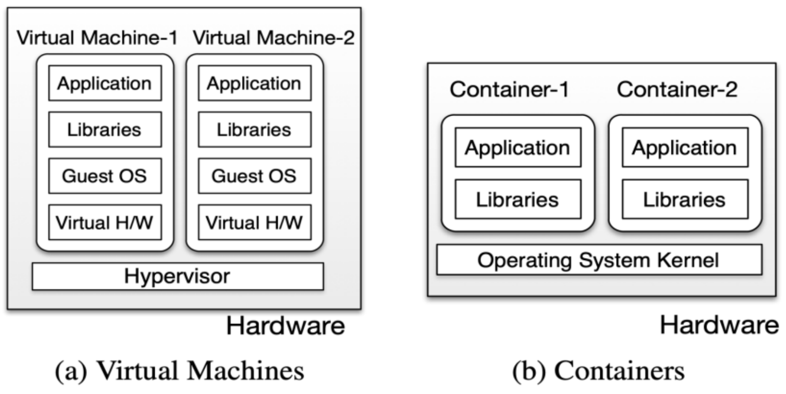

Figure 1: VM vs Container virtualization [1]

This section delves into the specifics of these challenges and their impact on business agility and operational efficiency.

2.1 Core Challenges:

- Tight Integration: Monolithic application's components are interdependent, complicating updates and risking widespread impact from minor changes [5] [6].
- Deployment Complexity: Updates require redeploying the entire application, a time - consuming and error - prone process [5].
- Scalability Limitations: Allocating resources to the entire application rather than specific areas, leading to inefficiency and increased costs [5].

2.2 Business Impact:

- Reduced Agility: The bulky update process slows the release cycle, slowing quick responses to market shifts or customer demands [7].
- Operational Inefficiencies: Challenges in scaling and maintaining monolithic applications increase operational costs and divert resources from innovation.

- Technical Debt Accumulation: Inflexibility and outdated technology stack increase maintenance difficulties and costs, undermining competitive edge and innovation capabilities [7].

The above limitations of traditional IT systems—ranging from scalability issues to technological inflexibility—have driven many organizations toward adopting containerization and microservices architectures. These modern approaches offer enhanced scalability, deployment flexibility, and the ability to innovate rapidly, aligning IT infrastructure with the demands of today's dynamic market environment.

3. Transition to Containerization

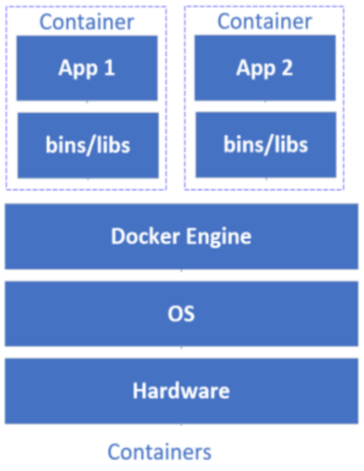

Figure 2: Couple of apps running on containers [2]

Containerization represents a transformation in how applications are packaged, deployed, and managed compared to legacy infrastructure. By packaging applications and their dependencies as a single unit into containers, organizations can achieve higher portability, scalability, and efficiency. This section discusses the emergence of container technologies, their advantages over traditional methods, and their role in facilitating the transition to a modern IT infrastructure.

3.1 Emergence of Container Technologies

Containers have drastically simplified and enhanced application deployment. Containers encapsulates an application's code, libraries, and dependencies in a portable executable package. This encapsulation ensures that the application runs consistently across different computing environments, from a developer's laptop to a test environment and ultimately to production, irrespective of the base infrastructure.

3.1.1 Core features

- Isolation: Containers provide independent environments for applications, preventing conflicts and ensuring consistent performance across different settings [10].
- Portability: With all necessary components included, containers ensure applications run reliably in various environments, eliminating compatibility issues [8].
- Microservices Compatibility: Containers support microservices by allowing individual services to be encapsulated and managed independently, promoting scalable and agile development [8].

3.1.2 Advantages Over Traditional Approaches:

- Scalability: Containers enable quick and efficient application scaling in response to demand changes without scaling the entire system.
- Deployment Speed: Container consistency accelerates deployment, supported by CI/CD pipelines for faster, automated rollout processes [9].
- Resource Efficiency: Containers use resources more efficiently than traditional virtual machines by sharing the host's kernel and avoiding full OS virtualization.
- DevOps Integration: Containerization enhances DevOps practices by simplifying application lifecycle management and fostering team collaboration through integrated CI/CD workflows [9].
- These core features and benefits of containerization signify a transformative approach to application development and deployment, offering solutions to legacy system limitations.

4. Container Orchestration: The Need and Solutions

As containerization became increasingly popular for deploying and managing applications, the complexity of orchestrating containerized workloads across different environments expanded. This complexity brought to light the need for advanced management tools capable of automating container lifecycle—leading to the development of container orchestration platforms. Fig.3 depicts the high level view of a container orchestration engine. Container orchestration solutions like Kubernetes, Docker Swarm, Hashicorp Nomad, etc are developed based on this design. These solutions are designed to automate and simplify container operations across host groups, such as deployment, scaling, networking, and management.

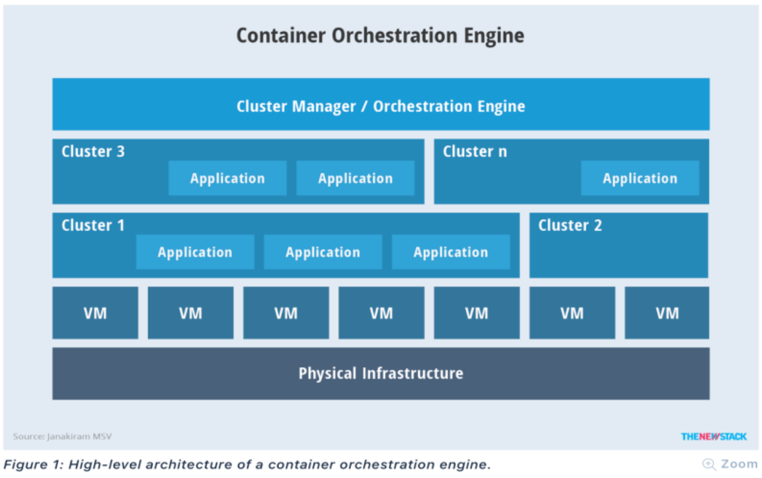

Figure 3: Container orchestrator - high level view [3]

4.1 Docker Swarm

Docker Swarm [11] seamlessly integrates with Docker, presenting an easy solution for those already familiar with Docker's ecosystem. Emphasizing ease of use, Docker Swarm facilitates a straightforward setup process, significantly reducing the learning curve for new adopters. Additionally, its automatic load - balancing feature efficiently allocates requests across containers, optimizing resource use and enhancing response efficiency.

However, Docker Swarm is often viewed as less feature - rich compared to its counterparts, particularly Kubernetes. Its simplicity and ease of use come at the cost of advanced features and scalability options, which makes Docker Swarm ideal for smaller - scale projects or those prioritizing simplicity over extensive configurability.

4.2 Kubernetes

Kubernetes [11] has emerged as the leading orchestration tool due to its scalability, flexibility, and comprehensive feature set. Capable of managing large clusters, Kubernetes supports a wide array of containerized applications with features like automated rollouts, service discovery, and secret management, making it a robust solution for complex deployments. Kubernetes' design to abstract the underlying infrastructure allows for remarkable portability, enabling applications to run seamlessly across different environments, be it public cloud, private cloud, or on - premises setups. This level of flexibility, however, introduces a steep learning curve and complexity in setup and management, potentially making it a huge barrier to entry for newcomers to container orchestration.

4.3 Apache Mesos with Marathon

Apache Mesos/Marathon [11] is a solution for managing resources across data centers and cloud environments. Mesos offers granular control over resources like CPU and memory across all cluster nodes, optimizing the utilization and efficiency of the infrastructure. Marathon extends Mesos's capabilities into container orchestration, supporting both containerized and traditional applications at scale. This combination excels in scalability and fault tolerance, handling tens of thousands of nodes without compromising performance. However, Mesos and Marathon's advanced capabilities come with increased complexity in setup and ongoing management.

5. Strategies for Migration

Migrating to a containerized environment is challenging and demands careful planning and execution. Here, we explore strategic approaches to transition from legacy systems to modernized containerization solutions.

5.1 Assessment and Planning

The first step in any migration process is thoroughly assessing the existing landscape [12]. It involves:

- Inventory of Applications: Cataloging all applications and services to understand their architecture, dependencies, and interactions.
- Priority Setting: Identifying which applications would benefit most from containerization, such as those needing frequent updates or those that can improve scalability.
- Resource Allocation: Estimating the resources (time, budget, personnel) required for the migration project.
- A comprehensive plan should outline the migration phases, set realistic timelines, and prepare for potential challenges.

5.2 Incremental Migration

- A phased, incremental approach to migration reduces risk and allows for adjustments based on lessons learned. It can be approached in several ways:
- Pilot Program: Start with less complex, non - critical applications as proof of concept to gain experience and build confidence.
- Horizontal vs. Vertical: Decide whether to migrate services one at a time (horizontally) or to move an entire stack or segment of the business (vertically).
- Parallel Operation: Run containerized services parallel to legacy systems to ensure functionality and performance before complete cutover.

- This strategy minimizes disruption to operations and allows for gradual adjustment of processes and team skills.

5.3 Leveraging Microservices

- Adopting a microservices architecture can greatly enhance the benefits of containerization by breaking down applications into smaller, independent services [13]. It involves:
- Decoupling Services: Identify natural boundaries within applications to separate into services that can be developed, deployed, and scaled independently.
- Domain - Driven Design: Organize microservices around business capabilities to ensure they are self - contained and aligned with business functions.
- Managing Inter - service Communication: Implement patterns for inter - service communication, such as APIs or messaging queues, to maintain loose coupling.

Microservices facilitate scalability and flexibility and encourage more agile development practices.

6. Pitfalls to Avoid

The migration journey has pitfalls that can delay progress and impact success. This section addresses the common challenges organizations face, such as underestimating the complexity of the transition, neglecting security considerations, and overlooking the importance of monitoring and logging. By identifying these pitfalls, organizations can better prepare for and navigate the complexities of migration.

6.1 Underestimating Complexity

- Cultural and Organizational Change: Acknowledge that migration affects not just technology but also people and processes [14]. Preparing the organization for change is as important as the technical aspects.
- Technical Debt: Addressing legacy issues and technical debt before migration can reduce complications and ensure a cleaner transition.

6.2 Neglecting Security Considerations

- Container Security [15]: Implementing robust security practices for container management, including securing the container runtime environment and using trusted container images.
- Network Security [15]: Ensuring network security policies are updated to accommodate the dynamic nature of containerized environments.

6.3 Overlooking Monitoring and Logging

- Visibility: Adopting tools and practices that offer deep visibility into containerized environments is essential for troubleshooting and performance tuning.
- Centralized Logging: Implementing a centralized logging solution to aggregate logs from all containers, facilitating easier analysis and monitoring.

7. Best Practices

Organizations must adhere to a set of best practices to ensure a smooth and effective transition. These include comprehensive team training, adopting continuous integration and deployment (CI/CD) pipelines, and implementing DevOps principles to foster collaboration and efficiency. This section provides actionable recommendations for organizations to follow, ensuring a successful transition to a modern containerized environment.

7.1 Comprehensive Training

- Skills Development: Investing in training and development programs for teams to acquire necessary skills in container management, orchestration tools, and microservices architecture [14].
- Knowledge Sharing: Encouraging knowledge sharing and collaboration across teams to disseminate learnings and best practices.

7.2 Embrace CI/CD

- Automation: Leveraging CI/CD pipelines for automated testing, building, and deployment processes. This ensures a consistent and error - free deployment process, which is crucial for managing containerized applications [16].
- DevOps Culture: Fostering a DevOps culture where development and operations teams collaborate closely, enabling faster iterations and more resilient systems [16].

7.3 Implement DevOps Principles

- Collaboration and Communication: Enhancing collaboration between teams to break down silos and improve efficiency.
- Continuous Improvement: Adopting a continuous improvement mindset to enhance processes, tools, and skills iteratively.

By following these strategies, avoiding common pitfalls, and adhering to best practices, organizations can navigate the complexities of migrating to a containerized environment, setting the stage for future innovation and growth.

8. Conclusion

The transition from traditional IT architectures to container platforms marks a critical evolution for organizations aiming to remain competitive in the digital era. Organizations can successfully navigate this transition by understanding the types of container orchestration solutions available, adopting strategic migration approaches, avoiding common pitfalls, and adhering to best practices. Container orchestration, despite its complexities, offers substantial benefits in terms of scalability, efficiency, and agility, signifying a significant advancement in the way software is deployed and managed. This white paper serves as a comprehensive guide for organizations embarking on this transformative journey.

References

[1] P. Sharma, L. Chaufournier, P. Shenoy, and T. Y. C, "Containers and virtual machines at scale: A comparative study, " in *Middleware '16: Proceedings of the 17th International Middleware Conference*,

Association for Computing Machinery, 2016, pp.1–13. doi: https: //doi. org/10.1145/298 8336.2988337.

[2] N. Wardle, “An Exploration of VMs, Containers, K8s & Microservices, ” Oct.22, 2018. https: //blogs. ultima. com/an - exploration - of - vms - containers - k8s - microservicess

[3] J. MSV, “Kubernetes: An Overview, ” *The New Stack*, Nov.07, 2016. https: //thenewstack. io/kubernetes - an - overview/

[4] P. Rubens, “What are containers and why do you need them?, ” *www.cio. com*, Jun.27, 2017. https: //www.cio. com/article/247005/what - are - containers - and - why - do - you - need - them. html

[5] G. Georgovassilis, “Climbing the monolith, ” *George’s Techblog*, May 20, 2018. https: //blog. georgovassilis. com/2018/05/20/climbing - the - monolith/

[6] J. Lumetta, “Should You Start With A Monolith or Microservices? | Nordic APIs |, ” *Nordic APIs*, Jan.25, 2018. https: //nordicapis. com/should - you - start - with - a - monolith - or - microservices/

[7] J. Nordström, “Architecting for speed: How agile innovators accelerate growth through microservices, ” *LinkedIn*, Sep.02, 2016. https: //www.linkedin. com/pulse/architecting - speed - how - agile - innovators - accelerate - growth - nordstr%C3%B6m/

[8] M. Pare, “Docker and Kubernetes: What is the Value of Containerization?, ” *CloudOps*, Jul.28, 2017. https: //www.cloudops. com/blog/docker - and - kubernetes - what - is - the - value - of - containerization/

[9] G. Haff, “5 advantages of containers for writing applications | The Enterprisers Project, ” *The Enterprisers Project*, Sep.06, 2017. https: //enterprisersproject. com/article/2017/8/5 - advantages - containers - writing - applications

[10] P. Iorio, “Container Based Architectures I/III: Technical Advantages, ” *Medium*, Jul.13, 2017. https: //pablo - iorio. medium. com/container - based - architecture - i - iii - technical - advantages - 7176195456c5

[11] J. MSV, “From Containers to Container Orchestration, ” *The New Stack*, May 11, 2016. https: //thenewstack. io/containers - container - orchestration/

[12] S. Orban, “6 Strategies for Migrating Applications to the Cloud | Amazon Web Services, ” *Amazon Web Services*, Nov.01, 2016. https: //aws. amazon. com/blogs/enterprise - strategy/6 - strategies - for - migrating - applications - to - the - cloud/

[13] Z. Dehghani, “How to Break a Monolith into Microservices, ” *Thoughtworks*, 2018. https: //martinfowler. com/articles/break - monolith - into - microservices. html

[14] S. Kuenzli, “Success Strikes Back: Containerizing Legacy Applications, ” *Qualimente*, Dec.30, 2016. https: //www.qualimente. com/2016/12/30/containerizing - legacy - applications/

[15] G. Kosaka, “You Can’t Secure What You Can’t See - Docker Network Security, ” *LinkedIn*, Nov.28, 2016. https: //www.linkedin. com/pulse/you - cant - secure - what - see - docker - network - security - glen - kosaka/

[16] R. Scott, “7 Features That Make Kubernetes Ideal for CI/CD, ” *The New Stack*, Jun.28, 2018. https: //thenewstack. io/7 - features - that - make - kubernetes - ideal - for - ci - cd/