Journal of Technological Innovations

Est. 2020

Evaluating High Availability and Performance in Containerized Linux

Arun Pandiyan Perumal
Systems Integration Advisor, NTT DATA
Servicesarun4pap@gmail.com

Abstract:

In the pursuit of a robust and efficient computing infrastructure, system resilience and operational efficiency have become indispensable. Conventionally, virtual machines (VMs) have been the cornerstone of infrastructure deployment but are often hampered by significant resource overhead, scalability constraints, and extended failover durations. With the evolution from VMs to containerized environments, this study aims to evaluate the enhancements in availability and performance that containerized Linux environments provide over their traditional counterparts. As modern IT infrastructures necessitate unprecedented system availability and performance levels, traditional VMs often falter due to excessive resource needs, flexibility issues, and delayed recovery periods. Containerization, with its lightweight nature and microservice-friendly architecture, offers a potential remedy to these challenges. Employing an empirical approach, this study examines container technologies, including Docker and Kubernetes, across various Linux distributions. Key performance indicators (KPIs) such as failover times, CPU and Memory utilization, scalability, network latency, I/O throughput, etc., serve as evaluation criteria, which provide a comparative analysis against VMs. The findings reveal that containerized environments significantly outperform VMs in terms of rapid scalability, improved fault tolerance, and more efficient resource utilization, thereby enhancing overall system performance. The implications of these findings suggest a pivot for both practitioners and researchers in technology infrastructure operations to consider containerized frameworks. This study improves the current understanding by elucidating the benefits of container technologies compared with conventional virtualization methods. This study underscores the need for continued research to refine container technologies, potentially unlocking further advancements in high availability and performance.

Keywords: Containerization, Linux Containers, Virtual Machines (VMs), Container Engine (Docker), Container Orchestration (Kubernetes), High Availability, System Performance.

1. Introduction

The advent of virtualization technologies has revolutionized the domain of enterprise computing by introducing unprecedented flexibility, improved resource management, and enhanced security through isolation. Virtual machines (VMs) are at the forefront of this evolution, enabling multiple operating systems to coexist on a single physical host, each running in an isolated environment. The architecture of traditional VMs, which relies on a hypervisor layer to manage guest operating systems, has served in many use cases ranging from server consolidation to application testing and development environments [1]. Although VMs are prevalent in enterprise environments, a relentless pursuit of high availability and performance remains imperative. In the context of mission-critical applications, the costs associated

with downtime are directly linked to financial losses and damage to reputation, necessitating architectures designed to ensure uninterrupted service and swift recovery from failure [2].

In recent years, containerization technology has emerged as a compelling alternative to VMs, particularly in Linux environments. Containers offer a

lightweight approach to virtualization, packaging applications, and their dependencies on isolated userspace instances [3].

Linux-based containers, epitomized by Docker and orchestrated by Kubernetes, provide a lightweight alternative to heavier VMs. Containers encapsulate applications and their dependencies into a single cohesive unit of deployment, leveraging the host OS kernel and avoiding the need for a separate OS per application, resulting in drastically reduced overhead. Process isolation mechanisms, facilitated by Linux features such as namespaces and cgroups, ensure containers remain lightweight and secure [4].

Despite their widespread adoption, VMs exhibit notable limitations and challenges, particularly in the context of resource overhead, scalability, and failover capabilities. Their inherent heaviness, owing to the duplication of entire operating systems, contributes to suboptimal resource utilization and can impede rapid scaling. Moreover, in high- availability scenarios, longer failover times associated with VMs can be detrimental to service continuity. These deficiencies highlight the need for improved high-availability and high-performance solutions, particularly in environments where rapid elasticity and responsiveness are paramount [5].

The primary objective of this study is to thoroughly evaluate how containerized Linux environments can enhance availability and performance compared to VMs. It aims to elucidate and quantify the benefits of containerization, thereby providing empirical evidence to support the growing consensus regarding its efficacy. Furthermore, this study seeks to identify and analyze how containerized environments can successfully mitigate the shortcomings of VMs, offering practical insights and recommendations for enterprise adoption. To achieve this objective, we employed an experimental design that encompasses the selection of prominent containerization platforms and tools for performance and availability monitoring [6].

Key performance indicators such as failover times, CPU and Memory utilization, scalability, network latency, I/O throughput, etc., were meticulously assessed to provide a quantitative basis for comparison. The significance of this study extends to the domains of system architecture and design, offering valuable perspectives to organizations contemplating a shift from VM-based infrastructure to containerized solutions. By providing a detailed performance and availability analysis, our work seeks to inform decision-makers in tailoring their infrastructure to achieve optimal operational outcomes. In conclusion, this study will enhance our understanding of containerization, VMs, and their evolving roles in cloud computing technologies, paving the way for further studies and innovations.

2. Problem Statement

Virtualization technology has played a pivotal role in the advancement of enterprise computing by enabling more efficient use of physical hardware resources and providing the flexibility needed for the dynamic scaling and management of applications. The VM model, which typically involves the emulation of entire operating systems on top of a physical server using a hypervisor, has enabled businesses to improve resource utilization and achieve better elasticity and isolation [1]. However, as enterprise workloads grow in complexity and scale, traditional VM-based architectures increasingly face limitations that can impede the agility and efficiency of modern applications. These limitations can be categorized into several key areas.

Resource Overhead and Inefficiency:

One of the most significant limitations of VMs is the resource overhead associated with running multiple instances of full-fledged operating systems on a single physical host. Each VM requires a complete set of virtualized hardware, including the CPU, memory, disk, and network interface, which can result in the underutilization of physical resources [5]. This inefficiency is amplified in environments where numerous VMs are deployed, leading to increased power consumption and cooling requirements, affecting operational costs and environmental sustainability.

Scalability Concerns:

Due to the overhead of running a full OS stack, scaling applications horizontally or adding more instances in a VM- based infrastructure can be slower and more resource-intensive. Booting up a new VM instance takes considerably longer because the entire operating system and all services need to be loaded, which can impede rapid scaling and responsiveness to fluctuating workloads [6].

Failover and High Availability:

High-availability scenarios necessitate architectures that swiftly recover from failures to minimize downtime. Migrating VMs for failover purposes involves a significant amount of data transfer due to the size of the VM images, the need to boot entire operating systems, and the restoration of application states, leading to delays that could be detrimental to mission-critical applications [6][8].

Complexity in Management and Operations:

Managing a large fleet of VMs entails dealing with the intricacies of multiple operating systems, patching, and configurations. This can lead to increased operational costs, a higher chance of misconfiguration, and a larger attack surface for security threats [8]. The complex nature of VMs, coupled with the need for a hypervisor and full OS stacks, can lead to infrastructure rigidity.

3. Containerization Technology in Linux Environments

Containerization, particularly in Linux environments, represents a significant evolution in virtualization. It offers a streamlined and efficient approach to deploying and managing applications. The lightweight, virtualized, isolated, and portable nature of containers has accelerated their adoption across industries. The concept of containerization was significantly enhanced by the introduction of cgroups into the Linux kernel in 2008 and further matured with the development of Linux Containers. Containers provide improved resource consumption, faster deployment times, and more consistent operations in different environments [3].

Containerization involves encapsulating an application and its dependencies into a container image that can be executed consistently across different computing environments. Unlike VMs, containers share the host system's kernel yet maintain process and filesystem isolation through Linux features, such as namespaces, cgroups, and SELinux. Linux containers offer an environment as close to a VM as possible without the overhead of running a

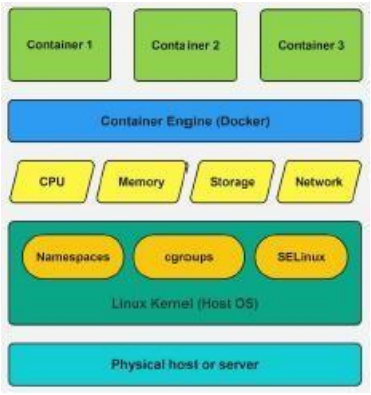
separate kernel and simulating all the hardware [3][4].

This efficacy is particularly advantageous for microservices, where each service runs in its container. Namespaces are a key feature of the Linux kernel that provides isolation by partitioning kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources. This isolation can include aspects of the system, such as process IDs, network interfaces, and mount points. Control groups (cgroups) allow the Linux kernel to organize processes into hierarchical groups, allowing the management of resources such as CPU, memory, network bandwidth, or combinations of these resources. Cgroups are essential to prevent containers from consuming excessive resources and affecting other containers. Security-Enhanced Linux (SELinux) is a Linux security module that provides a mechanism for supporting access control security policies. In the context of containers, SELinux can enforce mandatory access controls that restrict the interaction between the containerized applications and the host system, thereby bolstering security [4].

Container engines and orchestration tools are pivotal technologies in containerization, enabling the creation, deployment, and management of containers at scale. These tools and technologies are essential for leveraging the benefits of containerization in software development and operation. A container engine, such as Docker, is a piece of software responsible for managing the lifecycle of containers. Docker has emerged as a leading container engine, simplifying container creation, deployment, and running through its platform. It uses a client-server architecture with the Docker daemon to manage container objects such as images, containers, networks, and volumes [13]. Container orchestration, such as Kubernetes, automates the deployment, management, scaling,

networking, and availability of containerized applications. Kubernetes is an open-source platform that manages and orchestrates the application containers across clusters of hosts in large, dynamic environments, providing a resilient framework for running distributed systems [3][10].

The adoption of containerization in the industry has grown due to its portability, efficiency, and expandability for application deployment, driven by the need for agile development practices, microservices architectures, and DevOps methodologies.



4. Adopting Containerization to address the limitations of Virtual Machines

Containerization, represented predominantly by technologies such as Docker and orchestrated by systems like Kubernetes, has significantly altered the virtualization landscape and has addressed many challenges conventionally associated with Virtual Machines (VMs).

Achieving high availability:

VMs rely on established clustering and replication methods to achieve high availability, but the boot-up times of VMs can be a limitation for rapid failover. Containers, being lightweight and starting quickly, improve high availability strategies by enabling faster restarts and scaling. Container orchestration systems like Kubernetes provide built-in mechanisms for health checking, self-healing (automatic restarts), and rolling updates, contributing to high availability without additional clustering software [7].

Enhancing operational efficiency:

VMs incur performance overhead due to the complete virtualization of hardware and the need to run multiple OS instances. Containers share the host system's kernel, avoiding the need to run a full OS stack for each application instance. This results in a significant reduction in system overhead, leading to better performance when compared to VMs [9]. Containers start faster and have a smaller footprint, which is particularly beneficial for applications that require rapid scaling or frequent redeployments.

Recovery failover:

Failover can be resource-intensive and slow in a VM environment, as it often involves booting up an entire VM on another host. The Kubernetes can automatically detect and replace failed containers much faster than traditional VM failover processes. This quick failover is possible because containerized applications are designed to be stateless and ephemeral [10].

Optimization of resources:

Containers are more efficient resource-utilizers than VMs because they allow for higher density. A single host can run many more containers than VMs since containers require less overhead. This efficient use of server resources can lead to cost savings and reduced environmental impact due to lower power consumption [11].

Scalable instances:

Containers are inherently more scalable than VMs due to their lightweight nature. The Kubernetes can automate the scaling process, spinning up new containers as demand increases and decommissioning them as demand decreases. This automated, horizontal scaling is more fine-grained and faster than scaling VMs, a critical advantage in modern cloud- native application environments [7].

Reduced management overhead:

VMs incur considerable overhead due to the hypervisor and multiple guest operating systems. Containers minimize this overhead by abstracting the application from the underlying infrastructure, which translates into better performance and lower latency. The declarative nature of Kubernetes allows for infrastructure as code (IaC), making it easier to manage, replicate, and deploy containerized environments [14].

The primary distinction between VMs and containers lies in their architectural approach to virtualization. VMs utilize hypervisors to create fully isolated environments with dedicated resources. At the same time, containers share the host OS kernel, allowing for a more lightweight and efficient system that reduces redundancy and overhead. The streamlined architecture of containers translates to reduced latency, improved performance, and increased application availability.

5. Comparative Analysis for performance evaluation

The comparative analysis leverages experimental data to evaluate containerized Linux environments against VMs across the identified Key performance indicators. The below approach was used to perform the analysis using a combination of synthetic and real-world workload scenarios.

Environment Setup:

Created a containerization environment using Docker and Kubernetes to manage and orchestrate the Linux containers. Deployed a set of identical services in VMs and containerized environments. Maintained the hardware resources and software versions were consistent across both setups to minimize variables affecting performance.

Benchmarking Tools:

Utilized industry-standard tools such as Apache JMeter for network performance, Sysbench for CPU, memory, and I/O benchmarks, and Prometheus with Grafana for real-time monitoring and visualization of resource utilization.

Performance Metrics:

The following key metrics were used for performance evaluation,

Failover times - Uptime percentages, Mean Time Between Failures (MTBF), and Mean Time to Recovery (MTTR) for service availability. Recovery Point Objectives (RPO) and failover times to assess redundancy strategies.

CPU Utilization - Percentage of CPU resources utilized during idle and peak loads.

Memory Utilization - Memory usage efficiency, including total memory used and available under various loads.

Scalability - The time to scale out and scale in and the performance impact of scaling the container instances.

I/O Throughput - Disk throughput, IOPS, and latency measurements.

Network Latency - Network throughput and ping Round-Trip Time (RTT) for network performance.

Data Collection Process:

Performance data was collected using the selected benchmark tools, where both containerized Linux environments and VMs were induced with similar workloads that reflect typical usage patterns and stress conditions to evaluate how each environment handled high demand.

The statistical analysis underpins the comparative results, with containerized environments showing statistically significant improvements in resource utilization, startup times, and scalability. The high availability metrics favored containerized environments due to Kubernetes' orchestration capabilities, which streamline replication and recovery.

Failover times for service availability:

Containerized environments demonstrated significantly shorter failover times and MTTR due to their lightweight nature and rapid provisioning capabilities. MTBF was also improved, indicating

higher reliability. VMs, while robust, showed longer recovery times, affecting the RPO for redundancy strategies.

CPU and Memory Utilization:

Containers showed a more efficient CPU utilization pattern, particularly under peak load conditions, due to minimal overhead. VMs exhibited higher CPU overhead under similar conditions, primarily due to the hypervisor layer. Containers were more efficient in memory usage, with better memory allocation and deallocation strategies, leading to a higher availability of memory resources under various load conditions. VMs consumed more memory, partly due to the overhead of the guest operating systems.

Scalability:

Containerized environments scaled out and scaled in more swiftly than VMs, directly impacting performance positively during dynamic load adjustments.

I/O Throughput:

I/O operations were generally more efficient in containerized environments, with higher IOPS and lower latency, attributable to direct access to the host OS's I/O capabilities. VMs faced additional overhead, leading to reduced I/O throughput.

Network Latency:

Containers provided better network performance, with lower RTT measurements and higher throughput, benefiting from optimized networking stacks. VMs experienced higher latency due to additional virtualization layers. The experimental results underscored containerized environments' superior availability and performance compared to traditional VMs. The implications of these findings are profound, suggesting that containerized environments can significantly reduce operational costs by optimizing resource utilization and minimizing downtime [12]. This transition boosts performance and enhances the resilience of applications against failures, thereby ensuring business continuity. The shift to containerized solutions requires re-evaluating traditional system architecture and design principles. Organizations can leverage containerization by adopting microservices architectures to improve the modularity and scalability of applications. This study provides decision-makers with a detailed understanding of the performance benefits and highavailability features of containerized environments [15]. With the comparative analysis between containerization and VMs, the study informs infrastructure choices that align with organizational objectives and technical requirements. Ultimately, the study emphasizes the importance of containerization for organizations looking to modernize their IT infrastructure and improve operational competence. It highlights the need for organizations to adapt to evolving technologies to maintain competitive advantage and meet the growing demands for reliability and efficiency in digital services.

6. Conclusion

This study contributes to the evaluation of the high availability and performance of containerized Linux environments, aiming to address the shortcomings inherent in traditional VMs. The comparative analysis involving container engines and orchestration tools, such as Docker and Kubernetes, with Key performance indicators revealed that containerized environments excel in rapid scalability, enhanced fault tolerance, and efficient resource utilization, thereby contributing

to superior system performance. Specifically, the study highlighted the lightweight nature of containers, enabling faster startup times and reducing the overhead of running multiple instances of full-fledged operating systems. By leveraging the inherent advantages of containers, organizations can achieve unprecedented levels of agility, performance, and reliability, which are essential for meeting the demands of modern applications and workloads. Future studies can delve into advanced topics such as container security, multi-cloud container deployments, and the integration of containerization with emerging technologies like serverless computing and edge computing, driving forward innovation in the realm of container technologies.

References

[1] J.E. Smith, & R. Nair, The Architecture of Virtual Machines, IEEE Computer, 38(5), 32-38, May 2005.

[2] M. Arnold, S.J. Fink, D. Grove, M. Hind, and P.F. Sweeney, A Survey of Adaptive Optimization in Virtual Machines, Proceedings of the IEEE, vol. 93, no. 2, pp. 449-466, February 2005.

[3] D. Bernstein, Containers and Cloud: From LXC to Docker to Kubernetes, IEEE Cloud Computing, vol. 1, no. 3, pp. 81-84, September 2014.

[4] D. Merkel, Docker: lightweight Linux containers for consistent development and deployment, Linux Journal, May 2014.

[5] P. Sharma, L. Chaufournier, P. Shenoy, and Y.C. Tay, Containers and Virtual Machines at Scale: A Comparative Study, Proceedings of the 17th International Middleware Conference, November 2016.

[6] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, An updated performance comparison of virtual machines and Linux containers, IEEE International Symposium on Performance Analysis of Systems and Software, March 2015.

[7] E. Casalicchio and V. Perciballi, Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics, IEEE 2nd International Workshops on Foundations and Applications of Self* Systems, September 2017.

[8] N. Regola and J-C. Ducom, Recommendations for Virtualization Technologies in High-Performance Computing, 2010 IEEE Second International Conference on Cloud Computing Technology and Science, November 2010.

[9] M.G. Xavier, M.V. Neves, F.D. Rossi, T.C. Ferreto, T. Lange, and C.A.F. De Rose, Performance Evaluation of Container-Based Virtualization for High-Performance Computing Environments, Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, February 2013.

[10] K. Hightower, B. Burns, J. Beda, Kubernetes: Up and Running: Dive into the future of Infrastructure, O'reilly, October 2017.

[11] Z. Li, M. Kihl, Q. Lu, and J.A. Andersson, Performance Overhead Comparison between Hypervisor and Container Based Virtualization, IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), March 2017.

[12] S. Fernandes, J. Barreto, P.T. Endo, D. Beserra, E.D. Moreno, and D. Sadok, Performance analysis of Linux containers for high-performance computing applications, International Journal of Grid and Utility Computing, December 2017.

[13] P. Saha, A. Beltre, P. Uminski, and M. Govindaraju, Evaluation of Docker Containers for Scientific Workloads in the Cloud, Proceedings of the Practice and Experience on Advanced Research Computing, July 2018.

[14] W. Li, A. Kanso, and A. Gherbi, Leveraging Linux Containers to Achieve High Availability for Cloud Services, IEEE Xplore, March 2015.

[15] J. S. Hale, L. Li, C.N. Richardson, and G.N. Wells, Containers for Portable, Productive, and Performant Scientific Computing, Computing in Science & Engineering, November 2017.