Journal of Applied Engineering and Technological Science
Vol 6(2) 2025: 829-848

PROVISIONING OF LIVE CONTAINER MIGRATION IN EDGE/CLOUD ENVIRONMENTS: TECHNIQUES AND CHALLENGES

Radhwan B. Al-Bayram[1*], **Rawaa P. Qasha**[2]
College of Information Technology, University of Ninevah, Mosul, Iraq[1]
College of Computer Sciences and Mathematics, Department of Computer Sciences, University of Mosul, Mosul, Iraq[12]
radwanbasher@uoninevah.edu.iq, rawa_qasha@uomosul.edu.iq

Received: 03 December 2024, Revised: 04 May 2025, Accepted: 05 May 2025
**Corresponding Author*

ABSTRACT
Containers have become increasingly popular in the virtualization landscape. Their lightweight nature and fast deployment behavior make them an efficient alternative to traditional hypervisor-based virtual machines. In IoT applications and edge/cloud deployment, the live container migration can substantially reduce computing system overheads by minimizing the migration time and transmitting minimum memory pages from the source host without interrupting the service process. Until today, there has been a lack of comprehensive research discussing live container migration in the IoT domain and investigating the challenges of representing them in the edge/cloud environment. This survey presents cutting-edge articles that involve a live container migration approach. This survey aims to boost current knowledge, identify best practices, and highlight the challenges of live container migration in the IoT and edge/cloud environments, which will contribute to the advancement of container technology, as well as the optimization of deployment practices. The survey results indicate that selecting a suitable container engine relies heavily on the workload characteristics in the edge/cloud environment, particularly given the constraintions of live container migration. The survey highlights the direct and indirect challenges that influence container migration and proposes machine learning and blockchain as potential solutions.
Keywords: *Container, Live Container Migration, CRIU.*

1. Introduction

The cloud-based computing model is primarily based on two types of virtualization paradigms: H/W-level virtualization and OS-level virtualization(Bhardwaj & Rama Krishna, 2022). The first type is a virtual machine that behaves like a real computer with isolated applications running on a separate OS and bare metal components. The virtualization level is based on transparently encapsulating user applications in a high level of abstraction, which is controlled by a virtual supervisor engine that bundles up the virtual machine management operations(Doan et al., 2019)

The second type of virtualization is OS-level encapsulation or containerization. It isolates the instances of the user-space domain in the same kernel and enables multi-user applications to share the underlying hardware spaces without resource conflicts. The container encapsulates the application data with all necessary package libraries and binary files in image file format so that each image can be executed in different and independent environments. Therefore, this technology can be arranged as a lightweight service and achieve faster initialization than a virtual machine (Stephen et al., 2007).

Container features enable the development of application platforms that encourage high performance in deployment, shutdown, upgrade, and migration within just a few milliseconds. These and many other features have prompted the DevOps community to adopt container-based techniques as an alternative to VMs in the edge/cloud era(Felter et al., 2014).

However, the essential feature considered besides virtual machines is the possibility of being wholly isolated applications with solid security support, since virtualization in a virtual machine is done at the hardware level rather than at the kernel level in the container.

Another type of virtualization model is a hybrid model with features of both the container and virtual machine concepts (Dua et al., 2014).This survey focuses on state-of-the-art container migration approaches, presents and discusses live container migration techniques and challenges in edge/cloud environments, and employs IoT deployment as a case study.

Despite the recent adoption and provision of container services in cloud and edge environments, the robustness and isolation features of this technology have led to its extensive use across all cloud service platforms (public or private), edge services, and IoT deployments.

The integration of container-based support has revealed a multitude of scientific challenges and complexities. These include security and authentication hurdles, implementing workflows across diverse environments, adapting to varying network access, the necessity of achieving uniformity in modeling the regulations that define the global structure of container images, and managing the orchestration of multiple containers.

The lightweight and faster start-up behavior of containerized applications has been the most prominent features that enable developers and the DevOps community to propose live container migration through an application checkpointing scheme, which can allow shifting the running application with underlying memory pages, CPU variables, and network status from one host to another without interrupting user services. At the same time, the workload needs to be balanced, or upgrade recovery is triggered in the system, or even during the maintenance planning.

Live container migration in edge/cloud environments faces significant challenges, including high downtime, security vulnerabilities, and network latency, particularly in latency-sensitive IoT applications. Despite advancements, existing approaches lack comprehensive solutions for minimizing migration overhead while ensuring service continuity in heterogeneous edge/cloud settings. The increasing dependence on IoT and edge computing for real-time applications, such as healthcare and smart cities, necessitates a seamless migration to avoid service disruptions.

The rapid adoption of containerized applications in edge and cloud environments has surpassed the development of effective live migration strategies. Existing studies often emphasize cloud-centric deployments, overlooking the distinct constraints of edge computing, such as limited bandwidth and diverse hardware. This survey examines the latest live container migration approaches and consolidates these tools within an edge/cloud environment, addressing essential challenges while offering relevant solutions.

The authors in (Bhardwaj & Rama Krishna, 2022) highlighted the advantages of container-based migration compared to VM migration, illustrating its potential to enhance cloud computing performance. The same previous concept applies here as well, where the study evaluated comparisons in the cloud computing environment but did not assess its experiments on the edge side. The study in (Solayman H.E. & Qasha R. P., 2023)examined the challenges of migrating stateful applications for machine learning-driven services and proposed a Kubernetes-based optimization framework; however, it overlooked the optimization challenges specific to the edge environment for the same case study.

While the paper (Bellavista et al., 2024) highlighted the challenges of current CaaS rental services in the cloud environment, where VM isolation necessitates a separate cluster for each tenant, leading to resource overhead, it proposed a multitenancy approach within the Kubernetes model. For edge computing, a framework is suggested that enables tenants to share a single cluster with a common control plane, thus reducing overhead while maintaining workload isolation. Although this study focuses on general CaaS challenges in cloud-edge environments, it ruled out the possibility of container migration approaches. (Andrijauskas et al., 2024) concludes that while CRIU offers potential benefits for high-performance computing, its current limitations prevent full integration into batch systems like OSPool. Future developments are needed to enhance its usability, particularly in containerized environments and GPU support. (Yang et al., 2024) discusses the challenge of high startup latency in GPU-based serverless computing, particularly for machine learning applications. The research highlights the effectiveness of integrating parallel and on-demand restore strategies to optimize GPU serverless workloads and proposes a multi-checkpoint mechanism using CRIU to increase shared content across checkpoint images. (Yang et al., 2024) presents Wharf, a novel framework for transparent and efficient live migration across heterogeneous hosts, which improves performance, adaptability, and system resilience. (X. Jin et al., 2024) explores container migration in edge computing within the industrial

Internet, focusing on reducing latency and enhancing reliability.(Meliani et al., 2025) focuses on proactive lifecycle management for stateful microservices in multi-cluster containerized environments. The authors introduce a zero-touch management (ZTM) framework that integrates with Kubernetes and enables seamless stateful container migrations across clusters.

In this survey, we will explore the intricacies of the container migration concept, focusing specifically on live container migration. Our discussion will center on its implementation in cloud and edge environments, highlighting advanced tools and addressing the challenges present in these settings.

The main contributions of this survey can be summarized as follows:

1. The study offers a reinforcement approach besides the limited studies proposed in previous literature regarding container live migration.
2. It promotes the idea of identifying the right tools to effectively execute and orchestrate the migration of containers across various locations in edge and cloud environments.
3. Finally, this survey stands on the most critical challenges researchers face while implementing live container migration in IoT applications and edge/cloud domains.

2. Literature Review

2.1. The Virtualization from VMs to Containers:

In cloud computing systems, virtualization technology is an essential foundation for enabling cloud computing centers to transform their physical IT resources into equivalent virtual resources that perform the same functional tasks; this enables cloud provider servers to initiate multi-tenancy of the required resources along with other benefits such as portability and scalability features. One type of virtualization model is known as a Type 1 virtualization system, as in Figure 1. a. In this system, the virtual machine manager is installed directly into the bare metal hardware of the host server; this layered structure gives the VM manager greater access to resources compared with the following type (Type 2). However, this type requires compatibility with tightly coupled hardware.

Another type of virtualization model is known as a Type 2 virtualization system Figure 1. b. This type of virtualization refers to the installation of virtualization software on a pre-existing operating system on a single host server that requires a virtual machine manager or hypervisor that enables the system to install different operating systems on the same host machine, this hypervisor is similar to the network bridge (br0) in the NIC but to convert VM requests to system H/W. This model structure allows guest applications to be device-independent and have a loosely coupled pattern (Doan et al., 2019).

These two types of virtualizations (Types 1 and 2) can be considered as H/W-level virtualization, where each guest operating system has a pre-determined address for a specific number of shared hardware by creating a grouped collection of logical resources such as memory, CPU, network, and storage. The hypervisor in these VMs acts as an abstraction level for the guest demand resources, which are represented by the guest-side application as a real H/W resource.

Besides the H/W-based visualization, there exists another type, OS-level virtualization or container-based virtualization. In this type, the visualization is done at OS-level typic, a really in Linux kernel; this container application has lightweight features and the raped boot-up in comparison to VMs system because it has direct plugging and patching interfaces within the existing OS in the system, and the user application needs only the library and binary files binding to be running (Dua et al., 2014). Table 1. represents the general features of container versus VM.

Table 1 - Containers VS VMs Features Comparison

Feature	Container	VM
Weight and size	Light, small	Heavy, big
Resources utilization	economical	exhaustion
Isolated visibility	kernel namespace level	OS space level
Virtualization	OS level	H/W level
Basis OS	single	multiple, OS for each VM
Startup time	millisecond	minutes

Hypervisor	not included	included, types 1 and 2
Provisioning and scalability	slow	real-time provisioning and scalability
Resources allocation	Dynamic allocation	Fixed allocation
Versatility support	No, (only one kernel)	Yes, (many OSs on top of one HW machine)

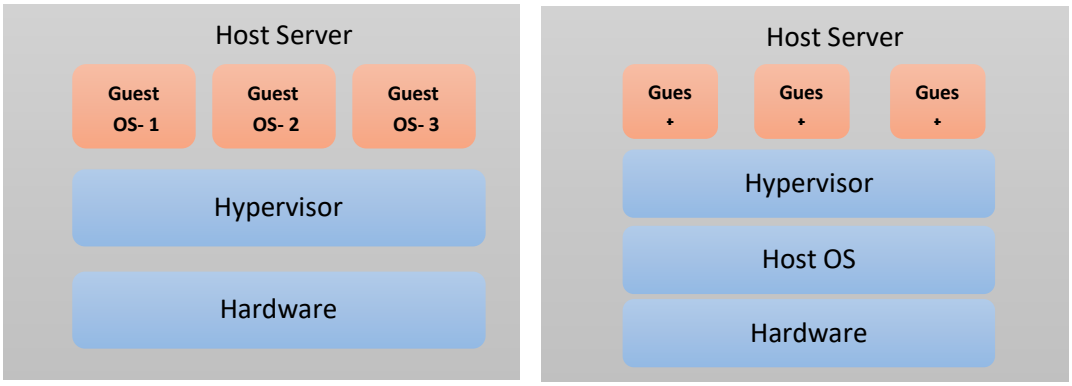

Fig. 1. Hypervisor virtualization models: a. Type 1 virtualization. b. Type 2 virtualization.

The container can also be divided into two types depending on how the container engine is installed in the host machine. A container engine is software that can accept the user request, pull down the container image, and run the container instance from the user's namespace concept. One model structure of the container system is installing the container engine as a wrapper layer for the underlying host's base OS (bar-metal), Figure 2.a. Another type of container system model combines the VM approaches and container features by installing the container engine as an abstraction layer for a Hypervisor-based container. Figure 2. b.

Each of the previous two deployment models has its own advantages and drawbacks, which are outside the scope of this survey. However, it's important to mention that the hybrid model, which consists of running containers on top of VMs, has advanced terminal security isolation but with performance overhead. It is worth noting that the lightweight container features, as well as the advantage of fast deployment and the economical consumption of resources, make container migration feasible and reliable compared to the case of VM migration, especially in edge/fog computing infrastructure where the computing power, device resources, and network throughput are lower than those in cloud data centers (Qasha, 2023).

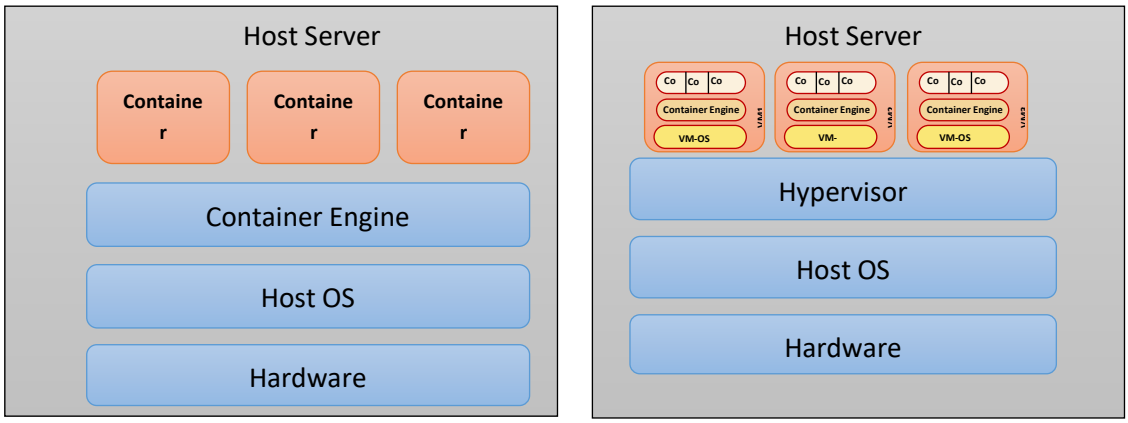

Fig. 2. Container Type Models, a. bar-metal Container Engine, b. Type 2 Hypervisor Virtualization Container

2.2. Container Migration

Most of the features of container migration techniques are inherited from the previous virtualization technique, VM migration. The migration technique is defined essentially as the ability to migrate the running virtualization layers (VM or container) with their dependencies from one physical host to another, with or without losing their states (Doan et al., 2019). The first appearance of OS virtualization implementation was proposed in 2000 by a FreeBSD Jail project through patching the Linux kernel (Kamp, n.d.). In 2005, a company named Virtuozzo released OpenVZ as a container virtualization for resource management based on the Linux checkpoint technique (*Open Source Container-Based Virtualization for Linux.*, n.d.). IBM, in 2008, released LXC as a complete OS virtualization based on Linux, which depends on Cgroups and namespace features(*Linux Containers (LXC) Is an Operating-System-Level Virtualization*, n.d.). This point was the key to further progress and development in containers SW, which contributed to the development of containers integrated platforms like Docker, Podman, and IBM LXC, and after that, with container management and orchestration platforms like Docker Swarm, Google Kubernetes, Apache Mesos, and Red Hat OpenShift.

In the case of VM/container migration, when an application running in a virtualization package is shut down, the underlying system does not preserve its current state, and the user service stops during the migration. This form of migration can be considered as a stateless/cold migration. In cold migration, the downtime is equal to the total migration time, which represents the total time of the container before it is ready to run. On the other hand, stateful/hot migration preserves the current execution statuses (CPU logs, memory pages, file tree system, and network configurations) of running applications, which are necessary to complete the migration and resume the container on the destination host. The ability to migrate running VMs/containers while preserving their state can significantly enhance the parameters factors of reliability, availability, and fault tolerance of edge/cloud computing (Muhammad Waseem & Aakash Ahmad, 2024).

Many works of literature have classified container migration types in various ways depending on the research focus and outcomes. The survey in (He & Buyya, 2021) has categorized container migration into two main types: cold and live migration. The last type has been divided into three types: pre-copy, post-copy, and hybrid-copy of container migration. Also, the survey focuses on container migration management and scheduling aspects in edge/cloud environments.

In (Puliafito et al., 2020b) the authors classified four types of container migration, which are cold migration, pre-copy migration, post-copy migration, and hybrid container migration, and they evaluated the impact of container migration on QoS using an augmented reality application based on the MQTT protocol in fog IoT services. The study in (Singh & Singh, 2022) is done with the same container migration classification as (Puliafito et al., 2020b). Also, they proposed a method to minimize network overhead by reusing memory states during container migration and reducing data transmission by only transferring updated memory pages by employing memory prediction based on both PSO and ANN schemas. The survey in (Kaur et al., 2022) followed an identical approach to the previous classification of container migration and further added the stateless and stateful types as primary root types depending on the live services of the container during migration. Also, the survey classified the container server distribution and how its performance is reflected by the container's host placement over geographical network deployment (edge, fog, core, and cloud).

Previous articles addressed various types of container migration, but they did not claim to cover the most advanced live container migration deployment tools and solutions. This survey focuses on the latest live container migration approaches and consolidates these tools in an edge/cloud environment, addressing critical challenges and providing relevant solutions. The accumulation of these classifications is shown in Figure 3, which can be a reference for future scientific citations.

In Figure 3, the top layer on the container, in general, can be categorized based on the placement of the container, which is if the migration of the container will be done in an edge/fog environment or the container deployment will be in a cloud environment which often in this

case the migration will be done through pods or a cluster of containers. The granularity type represents whether the application depends on a single container to be migrated or on multiple containers performing as cluster models that integrate the container orchestration tools to monitor the group of related containers as one pod. Also, the container migration can be for stateless and stateful containers, as explained later. The final type of container migration depends on the overall loading size of images and packaging needed during the migration process. When the cold migration is enabled, the processes need to migrate the container and the base image as a single step, which poses a heavy burden on the network, while in the hot migration, only the application statuses be migrated with network synchronization between the source host and target and depending one of the migration schemas (pre-copy, post-copy, or hybrid).

When the processes of container migration start by halting all processes in the guest container, and the user loses the connection to the services offered, this situation of migration refers to the cold, non-live, or non-interactive migration. On the other hand, in the hot, live, or interactive migration, the guest container remains running while executing the migration algorithm, and the user services are not affected by migration. The latter condition of container migration has been divided into three types (pre-copy migration, post-copy migration, and hybrid migration). In this survey, we focus on the live migration type.

All mentioned features of container migration, especially the lightweight size of containers and the speed of their provisioning time, motivated interested researchers and the DevOps community in commercial and enterprise businesses to carry out container services for IoT-integrated solutions and with cloud center implementations. In this research, we focus on reviewing the scientific studies that have utilized the container migration models in IoT and edge/cloud environments and presenting the state-of-the-art techniques and the tools that have been implemented for live container migration. Then, we discuss the main challenges facing the developers in this field.

2.3. Container Deployment and Live Migration:

In this section, we explain how containers are deployed in the placement of a cloud computing center or the placement of an edge/fog environment, and we demonstrate the portability of containers in both cases.

2.3.1. Container Deployment in Cloud-native Computing Environment:

The National Institute of Standards and Technology (NIST) (Mell & Grance, 2011)defines cloud computing as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources”. Cloud computing is a type of distributed computing built on a high-performance central data center and designed essentially as privileged resource sharing and dynamic demand of multi-tenancy requests. Virtualization techniques, such as VMs and containers, have been adopted in the cloud data center to efficiently manage cloud resources, great elasticity, and auto-scalability, fulfill demands on large storage and computing resources, and achieve an isolated environment alongside users' applications.

In Platform-as-a-Service (PaaS) for a cloud environment, the deployment of lightweight services like containers for packaging and orchestration implementation is a key issue (Pahl et al., 2017)Recently, many cloud service providers, such as Amazon AWS, Microsoft Azure, Google Cloud, and RedHat OpenShift, have built cluster-structured layering systems for incorporating and orchestrating container services for commercial and enterprise productions.

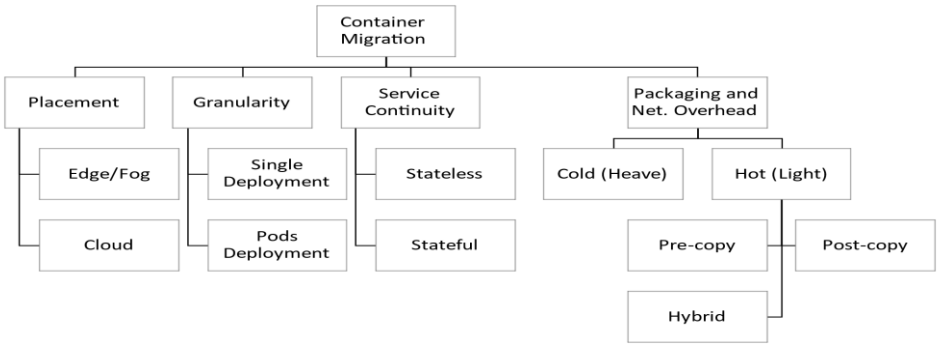

Fig. 3. Container Migration Types

In cloud-native container orchestration services, it relies, in almost all cases, upon stateless containerization virtualization services. To conduct migration processes for these services, cloud elasticity proposed some sorting of storage volume replications and redundant virtual network functions, like Cloud-native Network Function CNF. The mentioned situations make the possibility of handling the live container migrations practically challenging, and facing issues of difficulty and complexity.(Lee et al., 2024)

Despite the unique features of the cloud environment, such as central storage for big data, high computing power, and container orchestration administrations, as well as resource availability adaptation, however, there are issues of limitations of long-time latency, especially in the case of IoT applications, and also, the security risk issues for the users in the public cloud model (Abdullah & Hasan, 2023). These issues with other events led to the emergence of a new direction of distributed computing systems called Edge/Fog computing (Bonomi et al., 2012).

2.3.2. Container Deployment in Edge/Fog Computing Environments:

In 2012, Cisco first released the name “Fog Computing” as a collection of computing nodes with storage and routing capabilities, including gateways and computers, all functioning as a middleware layer between IoT devices and cloud computing.(Bonomi et al., 2012). The fog computing definition, according to (Yi et al., 2016) is: “a geographically distributed computing architecture with a resource pool consisting of one or more ubiquitously connected heterogeneous devices (including edge devices) at the edge of the network and not exclusively seamlessly backed by cloud services”. It is evident that the current trend in cloud-native deployment, especially within the edge/fog ecosystem, is shifting from VM-based virtualization to container-based virtualization solutions due to their offered advantages and properties. The fog computing environment offers a promising solution for geo-distributed node deployments where the user nodes are placed near the clusters of heterogeneous fog servers. In practice, this approach contributes to migrating microservices and container packages for IoT applications, where the speed of communication response and the latency-sensitive services are key challenges. Edge computing aims to force applications, data processing, and other services away from the central cloud data center to the edge network belonging to the user's device network in order to save network bandwidth or manage delay sensitivity in IoT applications (Abdullah & Mohammed, 2022).

Many research studies in the literature have presented promising prospects as well as challenges encountered when deploying and evaluating container solutions in Edge/Fog computing environments. In this context, it is considered that live container migration is a promising mechanism for networks of Edge/Fog environments. This ability can help overcome numerous Edge/Fog computing challenges, such as offloading computing to other hosts locally situated at the network's edge and near the user's location, expanding computing resources in fog computing, or handling the mobility issues on mobile edge computing (Hadeed & Abdullah, 2022).

The IoT environment architecture has been built essentially from small devices (sensors and actuators) connected wirelessly throughout PAN or LAN topology networks. Later, this environment was integrated into the cloud platform (public or private) for further processing and cleaning of the uploaded IoT big data, and achieved cloud resource utilization and storage management solutions (Solayman H.E. & Qasha R. P., 2023). Although this traditional

approach of provisioning IoT objects in cloud computing has achieved many gains, the need for short latency, as well as the processing of real-time data near their source, forces the necessity of pre-processing of uploading data in edge/fog sites. Figure 4 shows the edge computing paradigm at the bottom level in the three tiers: Cloud-Fog-Edge hierarchy as presented in (Kaur et al., 2022) as in Figure 4.

The emergence of SW virtualization, especially the lightweight containers, prompted the developers of IoT solutions to integrate their lightweight single-board technologies like Raspberry Pi that are placed at the edge of the network with lightweight container solutions to execute sensor responses and actuators processing with the real-time application in IoT fields, while at the same time maintaining interconnectivity with fog and cloud computing by migrating containers when needing to utilize data orchestration platforms and also employing big-data storage and clean/filtered data management (Pallewatta et al., 2023; Puliafito et al., 2020a).

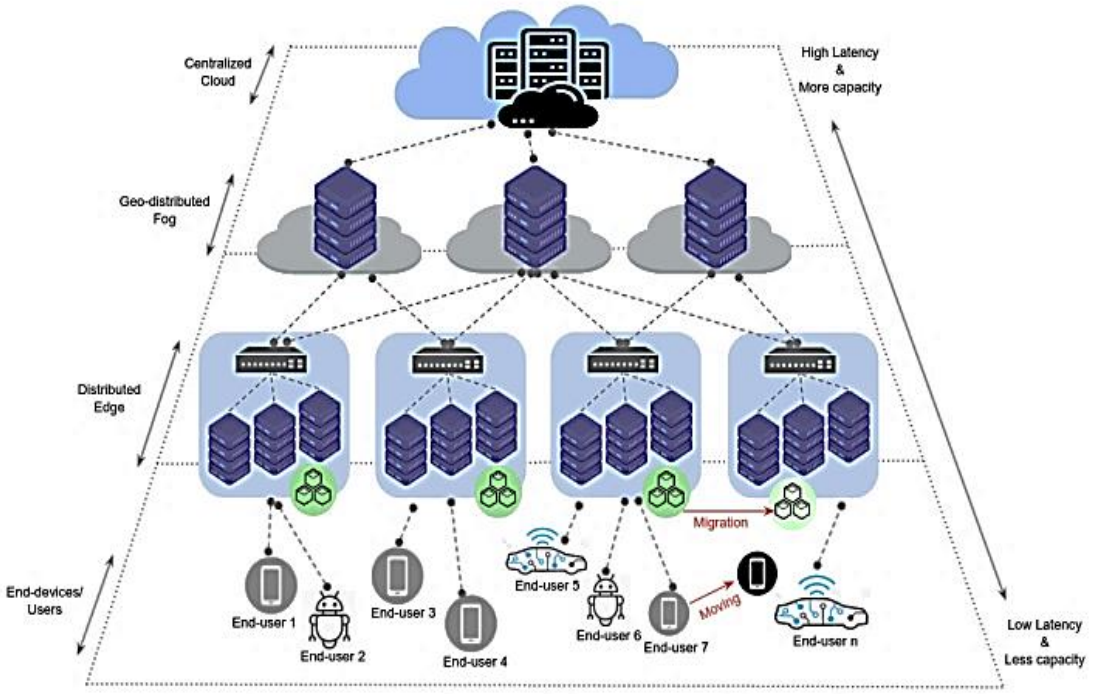

Fig. 4. 3-Tiers Structure of Cloud-Fog-Edge

3. Containers Live Migration Schemes

Currently, live container migration is a valuable tool in edge and cloud environments. In edge computing, containerization, and microservices migration are employed for real-time sensitive applications such as healthcare systems, video streaming, online gaming, traffic light management, and smart vehicle services (Pahl & Lee, 2015). On the other hand, commercial cloud infrastructure and cloud service providers such as AWS, Google, Amazon, and Red Hat have been offering containerization management and orchestration tools through a new approach called Container as a Service CaaS model (Senel et al., 2023). The live container migration is concerned with moving the running container from one host to another, with its status, and without needing to restart the process on the destination site. Keeping source memory status and paging resources is a vital thing that must be preserved and synchronized during the live migration. This happens with the condition that the user is unaware of the change of server site or the switch of the location of a service provider.

The evaluation of the migration processes is affected by two critical factors: the first one is the downtime, which represents the period between the stop-freeze stage of the container in the source host and the point of reactivation in the target host. During this period, the application services are unavailable to the user. The second factor is the total migration time, which involves the start of triggering the migration processes up until they are finished.

Live migration of VMs and containers is classified essentially into two main types, with a third type that combines the features of them. These algorithms are categorized depending on how many times (at minimum) they perform the process of checkpointing and synchronization for the memory status during the migration processes. The pre-copy migration algorithm performs two checkpoints (memory snapshots) and two synchronization processes (Clark et al., 2005) while the post-copy approach performs a single checkpoint and synchronization process (Hines et al., 2009). The hybrid algorithm has been designed to overcome the weaknesses of the two traditional approaches and to utilize their strengths(Sahni et al., 2012).

3.1. Pre-Copy Migration

In the pre-copy live container migration, or so-called iterative migration, the memory pages are migrated iteratively. During the pre-copy phases, the container remains in the running mode at the source host until the last pre-dumping memory, which follows the container freezing stage. After the final dumping of memory pages has been migrated, the container can be started on the target host from the freezing point. The pre-copy migration technique depends basically on two checkpoints and two synchronizing stages. The algorithm works with six stages as shown.

Stage 1: 1st checkpoint, the memory pages of the container are checked out with the checkpoint tool, and the container is left running.

Stage 2: 1st synchronization, the images of the 1st checkpoint are synchronized to the destination.

Stage 3: Comparison, counting the dirty memory pages (updated memory pages compared to the 1st stage).

Stage 4: 2nd checkpoint, checking out the dirty pages with reset container status.

Stage 5: 2nd synchronization, the images of the 2nd checkpoint are synchronized to the destination and halt the container.

Stage 6: Restore, restoring the images from the previous step in the destination host, and the container resumed running as before the 2nd checkpoint.

With pre-copy migration, the service provided by the container is live until stage 4 (2nd checkpoint), and the procedure has the option to repeat iterations 1 to 4 until it reaches the smallest interrupted time. The total migration time and the downtime are two metrics used to measure the performance of pre-copy migration. The performance of this approach depends on the application running in the container and the network connection. If the application creates more memory-dirty pages, that will influence the performance of the migration process, so choosing the application or job task to implement pre-copy container migration is a key issue.

The main advantages of this approach basically form two features: the high availability, which drew from the ability to keep the container alive for a more extended period before the freezing stage without affecting the services provided, and the ability to reduce the total cost of migration overhead by migrated a small chunk of data iteratively, however, the pre-copy procedure is not deterministic since we can’t guess how much the size of memory pages will be needed to migrate.

3.2. Post-Copy Migration

The post-copy live container migration, or so-called lazy migration, has been proposed as an alternative to the pre-copy algorithm. The essential idea of this approach is initially forced to stop the container in the source host, then migrate only the CPU states from the source so that the container can be directly started running in the destination site. Then the destination can pull down the memory pages from the source as needed. The post-copy algorithm works with one checkpoint and synchronization method as shown below.

The pos-copy algorithm needs eight stages to complete its job:

Stage 1: When deciding to migrate the running container, first, the container will be suspended and proceed to the checkpoint. However, post-copy is different from the pre-copy technique, where the images of the post checkpoint contain minimal information about

the execution container, which leads to running the container directly on the destination host.

Stage 2: Synchronize the previous images to the destination host.

Stage 3: The container is switched immediately to resume at the destination, starting from the minimal status images.

Stage 4: When the running container in the destination has access to nonresident memory, the algorithm throws a fault image.

Stage 5: The daemon running in the destination container handles the image faults and sends a request to pull down the lost pages from the source host.

Stage 6: The suspended original container receives the request, retrieves the lost images, and sends them to the destination.

Stage 7: The destination consumes the faulty images and continues running the application until it finds another lost image.

Stage 8: The post-copy technique repeats steps (4 to 7) until finished.

Post-copy, thus, ensures that each memory page is migrated only once. This avoids the duplicate migrated overhead of pre-copy. Also, despite the complexity of the post-copy algorithm, the total migration time can be practically reduced if the resuming container in the target host does not frequently access the non-synchronized memory pages, where the performance of the migration container could decrease due to the frequent memory fault recall procedure. On the other hand, the biggest drawback of the post-migration approach is represented by its unavailability since the application services initially terminate in the source host, so when the target host crashes during the post-migration processes, this also leads to the system crashing.

In comparing the total migration time and downtime for the previous two container migration schemes, it is observed that the pre-copy approach has a shorter total migration time than the post-copy approach. This is because, in the pre-copy approach, migration depends on the migrated dirty pages during the downtime phase, while the post-copy approach attempts to migrate all non-pageable states. On the other hand, the post-copy approach has the lowest downtime overhead as it prepares the container's CPU status for migration and defers the memory status to the pull-down stage. In contrast, the pre-copy approach involves many iterative migrations of memory pages, which take more time to complete. Choosing a suitable application in container migration is an essential key for both migration techniques, where write-intensive applications very much influence the performance of the pre-copy approach, and read-intensive applications influence the post-copy(Ma et al., 2019).

3.3. Hybrid Migration

The hybrid migration is a live container migration built to combine the advantages of both pre-/post-copy migrations. Actually, this technique is developed by combining a post-copy algorithm with preceding pre-copy stages. The scientific features that have been achieved by the combination done in the hybrid approach consist of both availability and reliability, where the availability feature, which is obtained from the pre-copy model, generates much more information about running container status, which assists post-copy processes after resuming the container in the destination host. Meanwhile, the post-copy model achieves reliability by reducing the number of pages that must be downloaded on demand.

Table 2 summarizes the strengths and weaknesses of the previous three methods for container live migration, assessed based on availability, efficiency, migration overhead, complexity, service determinism, and applicability.

Table 2 - Comparison between pre-copy, post-copy, and hybrid container migration

Criteria	Pre-Copy	Post-Copy	Hybrid
Availability	Guaranteed	Unguaranteed	Moderate, workload dependent
Efficiency	Good for stable workloads, poor for dynamic ones.	Good for dynamic workloads, slow if many page faults.	Balances both, less optimal for extremes.
Migration Overhead	Low for stable memory, high for dynamic memory.	Low for dynamic workloads, high with faults.	Moderate, varies by workload.
Complexity	Simple, overhead with dirty pages.	Simple transfer, complex in fault handling.	Most complex
Service Determinism	Predictable for stable loads	Predictable and variable performance.	Fairly predictable
Applicability	Heavy workloads	Latency-sensitive workload	Multi workload

4. Container Live Migration Techniques

For container migration, many applications and tools have been released to conduct the integrity on running container migration management for edge/cloud environments, especially for IoT virtualization models that assist services workload for many critical issues like load balance, storage redundancy, fast recovery on disaster and farther for improving the reliability and availability. In this context, we're focusing on the most commonly used tools and the popular SWs used in live container migration, where the migration of services from source host to destination while the container is remained running. All containers' running statuses are reserved in the target machine, keeping the migration time and downtime as short as possible without impacting user services or underlying applications.

One of the popular and powerful integrated tools for live container migration is called Linux Checkpoint and Restore in Userspace CRIU (*Linux Checkpoint/Restore In Userspace*, n.d.). In the next section, we briefly present the CRIU design and then discuss up-to-date container live migration applications and container management packages integrated with CRIU.

Checkpoint and Restore in Userspace CRIU

The CRIU (pronounced kree-oo) is a project started with initial release (v0.1) in 2012 as an implementation to checkpoint and restore running applications in the Linux namespace, and in 2013, it was released with the Linux kernel (v3.11)(*Linux Checkpoint/Restore In Userspace*, n.d.). Pavel Emelyanov proposed the 1st integration with the container platform with the OpenVZ project (*Checkpoint-Restore_p.Haul_ Live Migration Using CRIU*, n.d.). CRIU is a Linux tool that can check (freeze) any running process or vitalization-based application (VM or container) and dump their running status (memory page maps, open sockets, open files, etc.) as a collection of files in local host storage, which can later restore and resume these applications from the point of previous checking. Today, CRIU is the de facto and most successful tool for live container migration. This tool can retrieve the kernel file system from the process path, which contains the necessary information about the memory page map, child processes, and file descriptions. Then CRIU injects a parasite code into appropriate spaces in the process addresses to run the CRUI subroutine as a daemon process and to dump the memory contained as page maps files using the process's address space and tracing mechanism using the system-call ptrace. Unfortunately, this tool doesn't offer any facility for synchronizing files to another host in real time, and we need an external tool to do so.

From the viewpoint of the process that has been checked, it seems like normal behavior, and no additional operations are needed to support the CRIU procedure. This feature enables CRIU to check any Linux process. Figure 5 represents the main processes in the CRIU checking procedure. The container can be restored in the target host by calling the system function *fork()*. CRIU can create a new child process as a copy of the parent process, restore (unfreeze) the dump file, and re-run the application or container to the previous status as before checking.

Among the core features in the Linux kernel that the container has utilized in migration processes are the user namespace and cgroups. Namespace is a technique in the Linux kernel developed by E. Biederman in 2002 using C lang. (*Linux Namespaces*, n.d.). This feature provides a resource isolation layer for the container engine for each newly created container. By implementing a namespace, each process in the container has its own instance access to the available resources, and the object instances inside the container have visibility only in the space of the container instantiation and not outside. Currently, there are eight kinds of isolation in the namespace (Cgroup, IPC, Net, Mount, PID, Time, User, and UTS). When the container migrated, its namespace also migrated to the other side, and the restore procedure employed the namespace features to perform the container restoration.

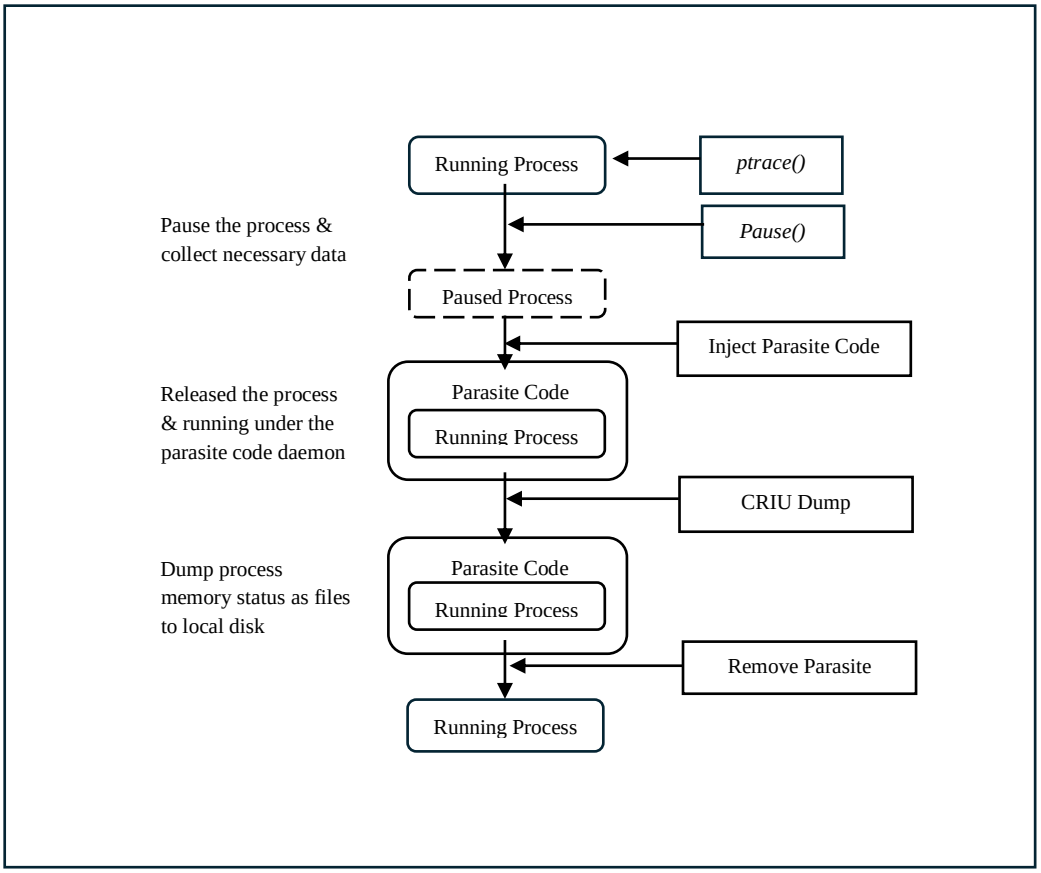

Fig. 5. CRIU Checkpoint Procedure

ON the other side, Linux Control Group Cgroup is a feature that limits resources like CPU cores, memory preservation, and I/O access for the hierarchical structure of process groups. Cgroup can nominate the type and quality of resources individually for each group of processes, so that it can effectively assist resource management in the container. CRIU utilizes this feature in the container freezing phase by individually freezing all resources in each Cgroup and then unfreezing them to the original state after container migration (*Cgroup-Freezer*, n.d.).

Many container applications and container management tools have integrated CRIU facilities for container live migration. In the next sections, we briefly demonstrate four examples of open-source container engine tools, two as OS-level or system-level container virtualization (OpenVZ and LXD) and two as application-level container virtualization (Docker and Podman), and we focus on how they implement CRIU for container live migration.

4.1. OS-Level Container

The OS-level container is similar to the virtual machine in the concept of versatility usage, where, in user space isolation, the system can install and run multiple applications inside the OS container. OpenVZ and LXD are two examples of this category of container platforms. On the other side, at the application level of the container, the system can run only a single

application or service. This is done even when the system is running multiple processes in the container, but they still belong to a single process or application. Docker and Podman are two examples of this type.

4.1.1. OpenVZ

Open Virtuozzo (OpenVZ) is an open-source OS-level container virtualization for Linux. It was produced by Virtuozzo in 2005. OpenVZ has a command-line interface CLI and web-based container management WebVS. In OpenVZ, the virtualization system (guest OS) that implements the running container must be the same as the core OS, which is the Linux kernel. OpenVZ started with a checkpoint tool for container migration in the Virtuozzo kernel and then implemented CRIU in the Linux kernel. CRIU features are fully integrated with OpenVZ version 7.0, released in 2016, with features of user namespace and Cgroup, where the previous migration processes had been done in kernel space, which produces a lot of restrictions (Mirkin OpenVZ et al., 2008). OpenVZ is perfect for isolated workload environments, but it demands a custom kernel, which limits flexibility. In OpenVZ, the command *vzmigrate* with the option –*online* can perform the live container migration.

4.1.2. LXD

LXD is another type of OS-level container virtualization that wraps the Linux container runtime LXC (LXC was released in Linux kernel 2.6.24 in 2008). LXD also supports building an instance of a virtual machine. LXD has daemon facilities with a REST API to develop and manage the container(*Run System Containers with LXD*, n.d.). LXD also has a command line interface lxc that provides the user with flexibility and compatibility, and a web-based GUI called LXD UI, but it's still under experimental features when writing this article. To do a live (stateful) container migration in LXD using the integrated CRIU features, the CRIU feature must be enabled first. The current documentation of LXD mentions that only basic containers (non- systemd containers and without attachment of network devices) can be live migrated smoothly. Otherwise, the system needs to stop the container before LXD can checkpoint it(*How to Move Existing LXD Instances between Servers*, n.d.).

4.2. Application-Level Container

At the application level of the container, the system can run only a single application or service. This is done even where the system is running multiple processes in the container, but they still belong to a single process or application. Docker and Podman are two examples of this type.

4.2.1. Docker

Docker started in 2015 as a project named Open Container Innovation OCI carried out by the Linux Foundation. Docker is a client-server application-level container engine with three main parts: the Docker server, the Docker API REST, and the Docker CLI. The Docker server or Docker Daemon, is responsible for managing and maintaining the container's life cycle from start to end and responding to API requests. The API REST is the interface used to communicate with the Docker daemon. The Docker CLI is the user interface environment. The new version of Docker has a GUI version called Docker Desktop. Docker also has a cloud-based distributed-registered Hup platform for public and private container image repositories. Docker run time engine started by integrating with a lightweight Linux container runtime program called runc. Because of the complexity and the autoconfiguration lacking in the runc functions, Docker got its own runtime API independent from the runc environment called libcontainer. For integrating with CRIU, Docker has produced checkpoint and restore features in experimental mode for migrating containers since Docker 1.13 in 2017, but with limited features. To migrate a stateful container after restating the Docker daemon in experimental mode, we need to create a checkpoint ID, and then we can start the migration with the command *start*.

4.2.2. Podman

Podman (Pod MANager) is another type of application-level container management produced by Red Hat, and it launched in 2018. Podman is open-source with compatibility with the Open Container Invention OCI which supports other types of containers and image generation engines like Docker and CRI-O, and it relies on an OCI-compliant container runtime (runc, crun, runy, etc) to interface with the host operating system and create the running containers. Podman is a daemonless run-time engine that enables the user to run Podman commands in a rootful or rootless privilege permissions (*What Is Podman*, n.d.)The CLI commands in Podman are very familiar with the Docker CLI commands. Podman can manage containers, images, volumes, and pods, which are groups of similar containers.

In 2019, Adrian Reber conducted a project for integrating Podman with CRIU to enable stateful container migration between different systems by implementing the checkpoint and restore features in the running containers (Adrian Reber, n.d.)To migrate the status in the running containers from one host to another, the subcommand *checkpoint* of the command *podman container* is used. Table 3 provides a summary comparing the four previous types of container engines.

Table 3 - Comparing of four container engines

Feature	OpenVZ	LXD	Docker	Podman
Type	System container	System container	Application container	Application container
Running Type	Running as a full OS	Lightweight VMs and containers	Runs as a daemon process	Runs as a single process
Performance	High	Lightweight	Lightweight	Lightweight
Security	Restricted, good isolation	Strong isolation	Risk with the root daemon	Rootless, enhanced security
Scalability	High	Supports clustering up to 50 servers	integrates with Kubernetes and Swarm	Kubernetes-ready, integrated
Storage	Snapshots and NFS	Flexible storage backends	External volumes	External volumes
OS Support	Linux only, requires OpenVZ kernel	Linux only	Linux, Windows, macOS	Linux, Windows, macOS
Proprietor	Virtuozzo	Canonical	Docker Inc.	Red Hat

Live container migration is a cornerstone of modern edge/cloud computing, enabling flexibility, reliability, and efficiency in dynamic environments. Advancements such as CRIU optimizations, hardware acceleration, and edge-specific frameworks have made migration faster and more robust, addressing industry demands for scalability and low latency. The migration of running processes and live containers offers numerous advantages, particularly for edge and cloud applications, considering the work environment in container live migration. Among these features are: resource optimization and workload balance, fault tolerance and reliability, node mobility in edge computing, hardware accelerations, and automation and policy management in orchestration cloud platforms.

5. Container Live Migration Challenges and Open Issues Directions

It is evident that the current trend in cloud-native deployment, especially within the edge/fog implementations, is shifting from VM-based virtualization to container-based virtualization solutions due to their offered advantages and properties, such as light load, scalability, and fast start-up. However, the road is still not paved enough to do so, and many challenges and obstacles have to be addressed. Some of these challenges - we focus on the live migration of the containers - are inherited from VM-based approaches, and others come from the containers ' build structure themselves. In this context, the previous literature studies in this field have reviewed and discussed these challenges, proposed methods to overcome them, and left the door open for further research questions.

The overview of these challenges mainly depends on where exactly the containers are set up, i.e., whether they are deployed in distributed edge/fog environments or hosted in the central cloud computing structures. This overview follows the same challenges as the live container migration. For example, in challenging situations where the container's live migration needs to be triggered in edge/fog applications, like handling the expected failure in edge servers site

hosting the single type of container's virtualization deployment, or on the other side, the example of challenges facing the needing to auto-provisioning the stateless or stateful containers that are installed in multi-pods clustered structure that requires the containers to be migrated in order to conduct auto-scaling and self-management in K8s-like cloud services.

From a general perspective, the challenges and obstacles that face the execution of live container migration, regardless of where the containers are running, can be generally classified into two main directions. The challenges that arise directly from the algorithm type and the software tool implemented to support container live migration (CRIU, pre-copy, post-copy, and hybrid approaches). Additionally, there are other challenges indirectly related to container live migration, such as security issues, network type and its capacity, the heterogeneous environment of the source and target of live migration, and the need for automated and dynamic container scalability.

5.1. Direct Challenges in Live Container Migration

When performing a live migration of a stateful container, it's crucial to consider the size of the memory data snapshots configured to be migrated, which includes the running status of the CPU, memory data, and dynamic configurations of the file system. The size of the memory snapshot significantly impacts the overall performance of container live migration since it is influenced by the total migration time. Larger memory sizes take more time to migrate, leading to increased latency and potentially affecting user utilization consistency.

For these challenges, many literary studies have proposed solutions for reducing the memory migration size. The authors in (Wu et al., 2017) proposed genetic-based or machine learning-based schemes for memory dirty page prediction in order to reduce the total amount of memory pages throughout the iterative process of memory dumping and reduce the download time of container migration. The study in(Lu & Jiang, 2023) addressed the problem of paging duplication in the pre-copy container migration algorithm. It proposed a prediction schema based on the locality principle of the random forest model. Also, they implemented an incremental compression model to reduce the size of data transmission. In (Nie et al., 2017), the researchers proposed an optimization model for the pre-copy container algorithm by reducing the number of migrated memory pages. The optimization method follows the Gray-Markov model, which minimizes the total migration time.

The authors in (Junior et al., 2020)utilize the layering facility in the container structure by exploiting the OverlayFS in the Docker layered structure. The enhancement in this approach comes from the ability to reduce the total migration time by migrating just the updated writable container layer and then pending the base readable layers by synchronizing the file system between the source and target host. The same approach is followed in (Machen et al., 2017). The authors also addressed the user mobility challenge when the user shifted to new location services and proposed an algorithm for synchronizing encapsulated base image layers to the potential additional closer server.

Other factors that directly impact the overall performance of container live migration are detailed in (Feitosa et al., 2025). The paper discusses several challenges associated with container migration strategies in edge and cloud environments, including resource consumption, network bandwidth, and the complexity of stateful containers during live migration. It also suggests an optimized migration strategy to balance efficiency, downtime, and resource consumption.

5.2. Indirect Challenges in Live Container Migration

The indirect challenges involved in stateful live container migration do not arise from factors that affect the internal structures of the tools or algorithms implementing the live migration functions. Instead, they are influenced by outside factors or foreign circumstances, such as security vulnerabilities, client movement out of service, connection bandwidth constraints, multi-cloud deployments, and resource limitations in IoT edge devices, which must be taken into account by the admin manager to distribute container load balancing between those operating in the central cloud and those operating in the decentralized edge center.

All the previous factors significantly impact the performance of live container migration. In this context, we are specifically focusing on two key aspects that we consider to be the most influential in ensuring optimal performance for live container migration tasks: the security challenges and the challenges of implementing live container migration in multi-cloud systems.

Quite a few studies have addressed the security challenges in the live migration of virtualization-based applications. Most of them included those related to the migration of VM applications, which is outside the interest of this survey. At the same time, only a few of them have addressed the security challenges in live stateful container migration and, precisely, in edge/fog environments. In contrast, some studies have suggested implementing application-level encryption and authentication schemes throughout container migration, while others propose integrating with third-party solutions, like the integration with a Blockchain environment, which grants a high level of container security guarantees.

In (Hosseinzadeh et al., 2016) an improved protocol is proposed for live migration of vTPM-VM, which includes a TPM-based integrity check policy and a specific cryptographic scheme to protect data during migration of container-based virtualization. In (Azab et al., 2016) the ESCAPE framework is proposed as an MTD mechanism for monitoring container migration against malicious behavior. ESCAPE leverages CRIU to seamlessly capture snapshots of the container's state during migration, empowering it to swiftly restore the container to a secure state upon detecting any attack. The study referenced in (H. Jin et al., 2021) introduces the DSEOM framework as an enhancement to the MTD protocol aimed at safeguarding the target system from potential attacks. To assess its effectiveness, the proposed method involves implementing live migration of Docker containers. The authors in (Ma et al., 2019) proposed utilizing the layered structure in the storage system of the container structure to minimize the overhead of file system synchronization. To reduce the security risks of offloading services in edge servers, the authors implemented an isolation layered structure by isolating different services in different Docker containers.

In some related literature, the decentralized networks Blockchain has been proposed as an integration framework with container allocations to enhance the security challenges in container-based implementation. The main contribution in (Antonio Marques et al., 2023) is introducing a framework called Clustered Event2ledger, which monitors Docker container environments using a consortium Blockchain to ensure data integrity, reliability, and availability. The framework collects the container and its service events, sends them to a Hyperledger Fabric Blockchain through signed transactions, and provides a distributed, tamper-proof repository for auditing. In (Farahmandian et al., 2024) the authors delved into the critical issue of fault tolerance in distributed systems, focusing on Byzantine faults and how security vulnerabilities influence the overall system. They proposed an integration model of container-based applications with the Blockchain principle to achieve reliability and availability, reduce resources, and increase fault tolerance in the system. The article (Sun et al., 2020) proposed a system that utilizes Blockchain technology to improve container cloud security. It aims to prevent the upload of malicious container images and to verify container image integrity using Blockchain, ensuring that the images are not tampered with. For Blockchain integration, the proposed approach uses Ethereum smart contracts. The system maintains a decentralized and tamper-proof record of image security profiles, which enhances reliability and transparency.

The article (Nawar A. Sultan & Rawaa Putros Qasha, 2023) presents a blockchain-based framework for securely monitoring vehicle traffic flow systems using containers in Docker.

The previous solutions are intended to work with individual independent containers or small groups of containers working together as a service provider in an IoT edge environment. Container management and orchestration platforms such as Docker Swarm, Kubernetes, Portainer, and OpenShift are valuable open tools for automating container orchestration and managing enterprise pod clusters of containers across inter-cloud system federation. Considering the intricate design structure of these tools, most container service solutions are tailored for stateless container provisioning. Consequently, numerous challenges emerge when attempting to implement live migration of stateful container types. The article (Bellavista et al., 2024) discusses the challenges of migrating stateful services in Industry 4.0 scenarios,

particularly for ML-driven applications, and presents a Kubernetes-based framework to optimize this process. The authors developed an enhanced Kubernetes stateful service migration mechanism that minimizes downtime by separating the application state into hot and cold states and preparing the target node in advance.

The framework CloudHopper proposed in (Benjaponpitak et al., 2020) empowers live migration of stateful containerized applications across AWS, Google Cloud, and Microsoft Azure, featuring advanced pre-copy optimization. This framework is engineered to deliver multi-cloud support, interdependent live container migration, rapid migration times, secure data transfer, uninterrupted client connections, and automated migration processes. In this context, the Ansible platform /www.ansible.com(*Red Hat Ansible Automation Platform*, n.d.) also supports the automated configuration of active container migration in a multi-cloud deployment. IPsec VPN tunnels under TCP/HTTP with a load balancer are used to ensure consistency and redirect the network traffic between the source cloud and the destination during container migration. The paper in (Swetha et al., 2025) highlighted the lack of existing research that discussed how resource utilization can influence overall performance in existing cloud container orchestration solutions and emphasized the importance of optimizing resource allocation in containerized cloud environments.

6. Conclusion

Live migration based on the container virtualization technique is a powerful and reliable strategy in distributed computing systems, whether provisioning in a cloud data center or on near-edge devices. The rapid and lightweight deployment of these containers significantly enhances application efficiency and performance, particularly when load-balancing is active on servers or when immediate responses to system failures are required. Implementing the pre-copy algorithm for container migration, as part of the CRIU approach, stands out as the most significant and practical method for conducting container migration in an edge/cloud environment. This study has sought to enhance understanding of the advanced techniques and tools utilized in this field. Avoiding security breaches and performing live container migration between multi-device architecture or multi-cloud combinations are among the most critical challenges facing researchers in this domain. Collaborating to establish a unified vision, consensus on a standard container design structure, and effective consultation during the migration process are essential keys for the future success of this promising technique.

References

Abdullah, D. B., & Hasan, B. T. (2023). HRRMLQ: Container scheduling algorithm on edge nodes cluster. *AIP Conference Proceedings*, *2834*(1). https://doi.org/10.1063/5.0171070

Abdullah, D. B., & Mohammed, H. H. (2022). DHFogSim: Smart Real-Time Traffic Management Framework for Fog Computing Systems. *ICOASE 2022 - 4th International Conference on Advanced Science and Engineering*, 60–65. https://doi.org/10.1109/ICOASE56293.2022.10075605

Adrian Reber. (n.d.). *Container migration with Podman on RHEL*. Retrieved June 22, 2024, from https://www.redhat.com/en/blog/container-migration-podman-rhel

Andrijauskas, F., Sfiligoi, I., Davila, D., Arora, A., Guiang, J., Bockelman, B., Thain, G., & Wurthwein, F. (2024). *CRIU -- Checkpoint Restore in Userspace for computational simulations and scientific applications*. http://arxiv.org/abs/2402.05244

Antonio Marques, M., Christian Miers, C., Rodrigues Obelheiro, R., Antonio SImplico Jr, M., Marques, M. A., Miers, C. C., Obelheiro, R. R., & Simplício Jr, M. A. (2023). *Clustered event2ledger: Docker event traceability using consortium Hyperledger blockchains*. https://doi.org/10.21203/rs.3.rs-2761768/v1

Azab, M., Mokhtar, B., Abed, A. S., & Eltoweissy, M. (2016, November 9). Toward Smart Moving Target Defense for Linux Container Resiliency. *IEEE 41st Conference on Local Computer Networks (LCN), Pp. 619-622. IEEE*. https://doi.org/10.1109/LCN.2016.106

Bellavista, P., Dahdal, S., Foschini, L., Tazzioli, D., Tortonesi, M., & Venanzi, R. (2024). Kubernetes Enhanced Stateful Service Migration for ML-Driven Applications in Industry

4.0 Scenarios. *2024 IEEE Annual Congress on Artificial Intelligence of Things (AIoT)*, 25–31. https://doi.org/10.1109/AIoT63253.2024.00015

Benjaponpitak, T., Karakate, M., & Sripanidkulchai, K. (2020). Enabling Live Migration of Containerized Applications Across Clouds. *In IEEE INFOCOM 2020-IEEE Conference on Computer Communications, Pp. 2529-2538. IEEE*.

Bhardwaj, A., & Rama Krishna, C. (2022). A Container-Based Technique to Improve Virtual Machine Migration in Cloud Computing. *IETE Journal of Research*, *68*(1), 401–416. https://doi.org/10.1080/03772063.2019.1605848

Bonomi, Flavio, Rodolfo Milito, Jiang Zhu, & Sateesh Addepalli. (2012). Fog Computing and Its Role in the Internet of Things. *In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Pp. 13-16*, 66.

Cgroup-freezer. (n.d.). Retrieved May 3, 2024, from https://www.kernel.org/doc/Documentation/cgroup-v1/freezer-subsystem.txt

checkpoint-restore_p.haul_ Live migration using CRIU. (n.d.). Retrieved May 3, 2024, from https://github.com/checkpoint-restore/p.haul

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., & Warfield, A. (2005). Live Migration of Virtual Machines. *In Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2, Pp. 273-286*.

Doan, Tung V., Giang T. Nguyen, Hani Salah, Sreekrishna Pandi, Michael Jarschel, Rastin Pries, & and Frank HP Fitzek. (2019). *Containers vs Virtual Machines: Choosing the Right Virtualization Technology for Mobile Edge Cloud*. *In 2019 IEEE 2nd 5G World Forum (5GWF), pp. 46-52. IEEE,*.

Dua, R., Raja, A. R., & Kakadia, D. (2014). Virtualization vs containerization to support PaaS. *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014*, 610–614. https://doi.org/10.1109/IC2E.2014.41

Farahmandian, M., Foumani, M. F., & Bayat, P. (2024). Improving fault tolerance in LinuX container-based distributed systems using blockchain. *Cluster Computing*. https://doi.org/10.1007/s10586-024-04279-9

Feitosa, L., Barbosa, V., Sabino, A., Lima, L. N., Fé, I., Silva, L. G., Callou, G., Carvalho, J., Leão, E., Nguyen, T. A., Rego, P., & Silva, F. A. (2025). A comprehensive performance evaluation of container migration strategies. *Computing*, *107*(2). https://doi.org/10.1007/s00607-025-01423-0

Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2014). An Updated Performance Comparison of Virtual Machines and Linux Containers. In *Computer Science*. http://domino.watson.ibm.com/library/CyberDig.nsf/home.

Hadeed, W., & Abdullah, D. B. (2022). Load Balancing Mechanism for Edge-CloudBased Priorities Containers. *International Journal of Wireless and Microwave Technologies*, *12*(5), 1–9. https://doi.org/10.5815/ijwmt.2022.05.01

He, T., & Buyya, R. (2021). *A Taxonomy of Live Migration Management in Cloud Computing*. http://arxiv.org/abs/2112.02593

Hines, M. R., Deshpande, U., & Gopalan, K. (2009). Post-Copy Live Migration of Virtual Machines. *Hines, Michael R., Umesh Deshpande, and Kartik Gopalan. "Post-Copy Live Migration of Virtual Machines." ACM SIGOPS Operating Systems Review 43, No. 3: 14-26*.

Hosseinzadeh, S., Laurén, S., & Leppänen, V. (2016). Security in container-based virtualization through vTPM. *Proceedings - 9th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2016*, 214–219. https://doi.org/10.1145/2996890.3009903

How to move existing LXD instances between servers. (n.d.). Retrieved April 15, 2025, from https://documentation.ubuntu.com/lxd/en/stable-5.21/howto/move_instances/

Jin, H., Li, Z., Zou, D., & Yuan, B. (2021). DSEOM: A Framework for Dynamic Security Evaluation and Optimization of MTD in Container-Based Cloud. *IEEE Transactions on Dependable and Secure Computing*, *18*(3), 1125–1136. https://doi.org/10.1109/TDSC.2019.2916666

Jin, X., He, S., & Chen, Y. (2024). Container migration for edge computing in industrial Internet with joint latency reduction and reliability enhancement. *Scientific Reports*, *14*(1). https://doi.org/10.1038/s41598-024-77086-2

Junior, P. S., Miorandi, D., & Pierre, G. (2020). Stateful Container Migration in Geo-Distributed Environments. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, *2020-December*, 49–56. https://doi.org/10.1109/CloudCom49646.2020.00005

Kamp, P.-H. (n.d.). *Jails: Confining the omnipotent root*.

Kaur, K., Guillemin, F., & Sailhan, F. (2022). Container placement and migration strategies for Cloud, Fog and Edge data centers: A survey. *International Journal of Network Management 32, No. 6*.

Lee, J., Kang, H., Yu, H. J., Na, J. H., Kim, J., Shin, J. H., & Noh, S. Y. (2024). MDB-KCP: persistence framework of in-memory database with CRIU-based container checkpoint in Kubernetes. *Journal of Cloud Computing*, *13*(1). https://doi.org/10.1186/s13677-024-00687-9

Linux Checkpoint/Restore In Userspace. (n.d.). Retrieved June 4, 2024, from https://criu.org/Main_Page

Linux Containers (LXC) is an operating-system-level virtualization. (n.d.). Retrieved May 25, 2024, from https://en.wikipedia.org/wiki/LXC

Linux namespaces. (n.d.). Retrieved June 8, 2024, from https://en.wikipedia.org/wiki/Linux_namespaces

Lu, Y., & Jiang, Y. (2023). A Container Pre-copy Migration Method Based on Dirty Page Prediction and Compression. *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, *2023-January*, 704–711. https://doi.org/10.1109/ICPADS56603.2022.00097

Ma, L., Yi, S., Carter, N., & Li, Q. (2019). *Efficient Live Migration of Edge Services Leveraging Container Layered Storage*.

Machen, A., Wang, S., Leung, K. K., Ko, B. J., & Salonidis, T. (2017). Live Service Migration in Mobile Edge Clouds. *IEEE Wireless Communications 25, No. 1 (2017): 140-147*. https://doi.org/10.1109/MWC.2017.1700011

Meliani, A. E., Mekki, M., & Ksentini, A. (2025). Resiliency focused proactive lifecycle management for stateful microservices in multi-cluster containerized environments. *Computer Communications*, *236*. https://doi.org/10.1016/j.comcom.2025.108111

Mell, P. M., & Grance, T. (2011). *The NIST definition of cloud computing*. https://doi.org/10.6028/NIST.SP.800-145

Mirkin OpenVZ, A., Kuznetsov OpenVZ, A., & Kolyshkin OpenVZ, K. (2008). Containers checkpointing and live migration. *In Proceedings of the Linux Symposium, Vol. 2, Pp. 85-90*.

Muhammad Waseem, & Aakash Ahmad. (2024). Containerization In Multi-Cloud Environment: Roles, Strategies, Challenges, and Solutions for Effective Implementation. *IEEE International Conference on Program Comprehension*, *2022-March*, 36–47.

Nawar A. Sultan, & Rawaa Putros Qasha. (2023). *Blockchain-Based Framework for Secure Monitoring of Vehicles Traffic Flow System*.

Nie, H., Li, P., Xu, H., Dong, L., Song, J., & Wang, R. (2017). Research on optimized pre-copy algorithm of live container migration in cloud environment. *Communications in Computer and Information Science*, *729*, 554–565. https://doi.org/10.1007/978-981-10-6442-5_53

Open source container-based virtualization for Linux. (n.d.). Retrieved May 25, 2024, from https://openvz.org/

Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2017). Cloud Container Technologies: a State-of-the-Art Review. *IEEE Transactions on Cloud Computing 7, No. 3 (2017): 677-692*.

Pahl, C., & Lee, B. (2015). Containers and clusters for edge cloud architectures-A technology review. *Proceedings - International Conference on Future Internet of Things and Cloud, FiCloud*, 379–386. https://doi.org/10.1109/FiCloud.2015.35

Pallewatta, S., Kostakos, V., & Buyya, R. (2023). Placement of Microservices-based IoT Applications in Fog Computing: A Taxonomy and Future Directions. *ACM Computing Surveys 55, No. 14s (2023): 1-43*.

Puliafito, C., Virdis, A., & Mingozzi, E. (2020a). Migration of Multi-container Services in the Fog to Support Things Mobility. *Proceedings - 2020 IEEE International Conference on Smart Computing, SMARTCOMP 2020*, 259–261. https://doi.org/10.1109/SMARTCOMP50058.2020.00058

Puliafito, C., Virdis, A., & Mingozzi, E. (2020b). The Impact of Container Migration on Fog Services as Perceived by Mobile Things. *Proceedings - 2020 IEEE International Conference on Smart Computing, SMARTCOMP 2020*, 9–16. https://doi.org/10.1109/SMARTCOMP50058.2020.00022

Qasha, H. E. (2023). On the use of container-based virtualisation for IoT provisioning and orchestration: a survey. In *Int. J. Computing Science and Mathematics* (Vol. 18, Issue 4).

Red Hat Ansible Automation Platform. (n.d.). Retrieved September 28, 2024, from https://www.redhat.com/en/technologies/management/ansible

Run system containers with LXD. (n.d.). Retrieved June 11, 2024, from https://canonical.com/lxd

Sahni, Shashank, & Vasudeva Varma. (2012). A Hybrid Approach To Live Migration Of Virtual Machines. *International Conference on Cloud Computing in Emerging Markets (CCEM), Pp. 1-5. IEEE*.

Senel, B. C., Mouchet, M., Cappos, J., Friedman, T., Fourmaux, O., & Mcgeer, R. (2023). Multitenant Containers as a Service (CaaS) for Clouds and Edge Clouds. *IEEE Access*, *11*, 144574–144601. https://doi.org/10.1109/ACCESS.2023.3344486

Singh, G., & Singh, P. (2022). A Container Migration Technique to Minimize the Network Overhead with Reusable Memory State. *International Journal of Computer Networks and Applications*, *9*(3), 350–360. https://doi.org/10.22247/ijcna/2022/212560

Solayman H.E., & Qasha R. P. (2023). *On the use of container-based virtualisation for IoT provisioning and orchestration: a survey*. *18*(Int. J. Computing Science and Mathematics), 299–311.

Stephen S, Herbert P, & Marc E. (2007). Container-based Operating System Virtualization: AScalable, High-performance Alternative to Hypervisors. *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 412.

Sun, J., Wu, C., & Ye, J. (2020). Blockchain-based Automated Container Cloud Security Enhancement System. *Proceedings - 2020 IEEE International Conference on Smart Cloud, SmartCloud 2020*, 1–6. https://doi.org/10.1109/SmartCloud49737.2020.00010

Swetha, R., Thriveni, J., & Venugopal, K. R. (2025). Resource Utilization-Based Container Orchestration: Closing the Gap for Enhanced Cloud Application Performance. *SN Computer Science*, *6*(3). https://doi.org/10.1007/s42979-024-03624-4

What is Podman. (n.d.). Retrieved June 22, 2024, from https://docs.podman.io/en/latest/

Wu, T. Y., Guizani, N., & Huang, J. S. (2017). Related Dirty Memory Prediction Mechanism for Live Migration Enhancement in Cloud Computing Environments. *Journal of Network and Computer Applications*, *90*, 83–89. https://doi.org/10.1016/j.jnca.2017.03.011

Yang, Y., Du, D., Song, H., & Xia, Y. (2024). On-demand and Parallel Checkpoint/Restore for GPU Applications. *SoCC 2024 - Proceedings of the 2024 ACM Symposium on Cloud Computing*, 415–433. https://doi.org/10.1145/3698038.3698510

Yang, Y., Hu, A., Zheng, Y., Zhao, B., Zhang, X., & Quinn, A. (2024). *Transparent and Efficient Live Migration across Heterogeneous Hosts with Wharf*. http://arxiv.org/abs/2410.15894

Yi, S., Hao, Z., Qin, Z., & Li, Q. (2016). Fog computing: Platform and applications. *Proceedings - 3rd Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2015*, 73–78. https://doi.org/10.1109/HotWeb.2015.22