Application Profiling for Automated VM to Container Migrations

Stanislav Petkov
Master of Applied IT
Fontys University of Applied Sciences
s.petkov@student.fontys.nl

January 16, 2025

Abstract

The migration of applications from virtual machines (VMs) to container environments offers significant performance and scalability benefits but remains a challenging process. It often requires manual dependency collection—a time-consuming and error-prone process. Existing automation solutions, such as Google Migrate to Containers (M2C), follow a lift-and-shift strategy but often produce bloated containers by including unnecessary system-level dependencies, undermining the core benefits of containers.

This study introduces a novel process-level profiling methodology to automate the migration of applications from Linux-based VMs to containers. The approach combines static and dynamic analysis of application runtime behavior and focuses exclusively on the active Unix processes associated with the target application to identify critical dependencies, ensuring lean and optimized container configurations.

Case studies on NGINX and MySQL demonstrate the efficacy of the proposed approach, achieving container size reductions of up to *98%* compared to Google M2C while maintaining or improving application performance. To validate its scalability and practical applicability, a command-line tool was developed based on this methodology. While effective, the method may face challenges with non-deterministic workloads, where certain runtime dependencies might be missed, so improvements are needed to ensure consistency. Nonetheless, this research represents a significant step toward streamlined, efficient, and accessible container migration.

Keywords: VM to container migration, process-level profiling, dependency analysis, application replatforming, Linux.

1 Introduction

The growing adoption of cloud computing has significantly increased the demand for application containerization due to its efficiency and scalability advantages. Containers provide a lightweight alternative to virtual machines (VMs), offering faster startup times and portability while minimizing resource consumption [1]. However, transitioning applications from VMs to containers remains a challenging process.

The manual migration to containers often faces significant bottlenecks, due to its time-consuming and error-prone nature. According to Meng *et al.* [2], the manual discovery of software and configurations requires significant time and effort, especially when assessing applications for containerization in large and complex file systems. This process becomes even more daunting considering the diverse methods of software installation—such as binaries, package managers, and source code—which complicates the discovery of all relevant components [3].

Further, manual methods often face challenges due to the absence of up-to-date business or implementation documentation, making it difficult to assess application dependencies accurately during migration [4]. Wang *et al.* [5] and Meng et al. *et al.* [3] highlight that manual approaches, which depend on ongoing human expertise and frequent updates, drive up operational costs and hinder scalability across diverse environments. Therefore, automation is essential to reduce reliance on manual labor, lower costs, and enhance scalability in the migration process.

Despite the advancements in tools and methodologies for automating the migration process, significant research gaps remain. Current research efforts, such as *Columbus* [3] and *Janus* [4], focus primarily on application configuration discovery but provide only partial solutions to the broader migration challenge. These rule-based approaches require substantial manual input and ongoing maintenance to keep pace with evolving software landscapes. Machine learning-based methods, such as *MetaConf* [5] and *Praxi* [6], aim to reduce manual effort, but rely heavily on extensive training datasets and may not generalize well to uncommon configurations, which limits their applicability across diverse environments.

Further, enterprise solutions, such as *AWS App2Container* [7] and *Google Migrate to Containers* [8], address specific migration scenarios but are limited in scope, restricting their adaptability to diverse use cases. These tools focus on rehosting strategies, which simplify migration but often produce bloated containers with unnecessary dependencies, limiting the performance and scalability benefits of containerization. In contrast, replatforming, which optimizes applications for leaner and more efficient containers, remains underexplored.

There is a clear need for a scalable and automated solution that can accurately profile applications across diverse environments with minimal reliance on manual effort or large datasets. Garg *et al.* [9] suggest, as future work, the development of an automated framework to evaluate and assist in migrating VM workloads to containers. Such solution should be adaptable to new technologies, capable of handling diverse configurations, and efficient in resource utilization.

This research introduces a novel process-level profiling method to automate application migration from Linux-based VMs to containers. By analyzing application runtime behavior, the proposed approach extracts key dependencies, like libraries and configurations, and packages them into minimal container images. The solution focuses on replatforming, optimizing applications for efficient and lean containerization. The feasibility of this method is demonstrated through case studies on NGINX [10] and MySQL [11] applications, showing significant improvements in containerization efficiency compared to existing technologies.

The *Background* section covers key concepts like containerization, migration strategies, and existing tools to set the study's context. The *Approach* section explains the profiling methodology, including static and dynamic analysis, and how dependencies are mapped into optimized container configurations. The *Results* section presents the method's implementation, case studies, and performance comparisons with existing solutions. The *Discussion* section examines the study's impact, limitations, and areas for improvement. Finally, the *Conclusion* summarizes the research contributions and potential directions for future work.

2 Background

2.1 Virtualization vs. Containers

Understanding the technical differences between VMs and containers is essential for successful migration. VMs provide hardware-level virtualization by running full operating systems on emulated hardware managed by a hypervisor, which offers strong isolation but with considerable overhead in resource usage and performance [1]. Containers, in contrast, utilize operating-system-level virtualization, sharing the host OS kernel while isolating applications in user space, leading to lightweight and resource-efficient environments [1].

This fundamental difference means that containers offer near-native performance with minimal overhead, as they eliminate the need to virtualize hardware [12]. They also provide faster startup times and better scalability compared to VMs. However, the shared kernel model requires that applications be compatible with the host OS, and the level of isolation is less robust than that provided by VMs [1]. These technical distinctions impact how applications behave in containers and must be carefully considered during the migration process.

2.2 Containerization Compatibility

Not all applications are a good fit for containerization due to various technical and operational challenges, which fall outside the scope of this research. Applications with platform dependencies tied to specific or outdated operating systems are often incompatible with container environments [13]. Others require direct hardware interaction, making them unsuitable for containers that abstract hardware access [14]. Security concerns also arise for applications requiring advanced security measures not readily available or differently implemented in container environments [9]. Licensing restrictions can further complicate migration efforts, as certain applications have legal limitations tied to specific environments, prohibiting their operation within containers [13].

Imran *et al.* [12] discuss the complexities involved in migrating stateful applications like databases, which rely on persistent storage and may not conform well to the stateless nature of containers. Additionally, legacy applications built on obsolete platforms or with outdated dependencies pose significant hurdles, as containers favor modern, lightweight environments [12]. These limitations necessitate careful pre-migration assessments to determine an application's suitability for containerization.

2.3 Cloud Migration Strategies

Organizations migrating applications from virtual machines (VMs) to containers typically choose from several strategies, varying in complexity and optimization. Kumar *et al.* [15] highlight the most common approaches, which include *rehosting*, *replatforming*, and *refactoring*. These strategies vary in terms of the effort required and the degree of cloud-native optimization achieved.

Rehosting is a lift-and-shift approach, which migrates applications without modifying their original components. This method requires minimal effort, however it often results in bloated containers with unnecessary dependencies, limiting the efficiency gains of containerization.

Refactoring (also called *Re-architecting*) involves redesigning an application's source code or architecture to take full advantage of cloud-native features. While this method offers the best optimization in terms of scalability and efficiency, it requires significant development effort, making it the most time-consuming and costly migration strategy.

This research study focuses on a refined *replatforming* approach, which balances effort and optimization. It aims to improve scalability by identifying and incorporating only the essential application components into the final container configuration. This reduces redundant dependencies and minimizes overhead.

2.4 Related Work

Several research efforts provide insights into automated software discovery and migration, though not all are directly focused on containerization. Nadgowda *et al.* [3] introduced *Columbus*, which identifies software by analyzing filesystem tree structures. While useful for discovering applications, its inability to read file contents limits its accuracy in detecting all dependencies necessary for migration to containers. Similarly, *Janus* [4] automates the detection of application configurations and dependencies but still requires manual input to refine incomplete configurations, which reduces the level of automation needed for a streamlined container migration.

Machine learning approaches, such as *Praxi* [6], classify software changesets based on filesystem activity. Though accurate, it relies on training data, which may not generalize well to all environments, especially in diverse enterprise systems. This poses a challenge when profiling applications with unique or non-standard configurations. *MetaConf* [5] addresses configuration file discovery using metadata, but its reliance on file metadata means it may miss crucial configuration details that aren't well-represented in metadata fields, reducing its reliability for comprehensive application profiling.

These approaches, while providing partial solutions, are either limited by their scope (e.g., missing critical details during software discovery) or require significant manual intervention, which does not align with the goal of fully automating the VM to container migration process.

2.5 Available Tools

A few enterprise solutions facilitate the migration of VM-based workloads to containers, each offering distinct capabilities tailored to specific scenarios. However, their limitations in scope and approach highlight the need for more comprehensive automation solutions.

For instance, *AWS App2Container (A2C)* [7] is a command-line tool that converts Java and .NET applications running on bare metal, VMs, or cloud instances into container format. While applicable to these application types, its restricted scope limits its generalizability to a wider range of technology stacks.

By comparison, *Google Migrate to Containers (M2C)* [8] supports a broader range of applications, including Linux and Windows-based web servers, business middleware, and small-to-medium databases. While M2C facilitates migration to Google Kubernetes Engine (GKE), it employs a lift-and-shift strategy, often resulting in large containers and diminishing the performance advantages of containerization.

Finally, *KubeVirt* [16] enables VMs to run alongside containers within a Kubernetes environment, providing a unique bridge for mixed infrastructure. However, it does not directly convert VMs to containers, making it more suitable for hybrid deployments rather than true migration.

3 Approach

The focus of this research is on the replatforming of applications from Linux-based virtual machines (VMs) to container environments. While the ultimate goal is broad compatibility across major Linux distributions, Ubuntu was selected for this study as a widely used and well-supported baseline to enable a controlled and replicable testing environment.

The proposed methodology adopts a process-level profiling strategy to analyze application runtime behavior. This novel approach treats applications as black boxes, it relies solely on their active processes to identify critical dependencies. By examining runtime behavior, the method aims to create accurate profiles of essential application components required for containerization without relying heavily on prior domain knowledge or detailed application documentation.

To support this, the approach combines static and dynamic analysis of application processes to comprehensively capture dependency data. Static analysis examines predefined system information, such as executable paths and environment variables, while dynamic tracing monitors application activity to detect dependencies that may only appear during operation. This dual-phase approach aims for improved accuracy of the dependency discovery process.

The collected dependency data is then further refined to focus on essential components for containerization. Filtering mechanisms are applied to exclude redundant or irrelevant information, resulting in a minimal set of critical dependencies. These are finally organized into a format suitable for containerization. The method focuses on adaptability, it aims to streamline the migration process while producing optimized containers.

The designed framework is validated through case studies with NGINX, a stateless web server, and MySQL, a stateful database. These examples represent a broad range of dependency requirements, which helps assess the method's applicability across diverse workloads. The resulting containers undergo basic functionality tests, and key performance metrics, such as image size and request handling rates, are compared with existing solutions to assess the efficiency of the proposed method. Finally, a proof-of-concept command-line tool is developed to test the practical implementation of the process-level approach and verify its applicability in real-world scenario.

4 Results

4.1 Available Solutions

Google Migrate to Containers (M2C) is currently the only actively maintained solution that automates container migrations for a broad set of application types. To evaluate its capabilities, NGINX and MySQL applications were deployed on a Linux VM and migrated using M2C. The resulting containers passed basic functionality tests—NGINX successfully served web content, and MySQL processed SQL queries.

Performance metrics, including request handling rate and container size, are summarized in Table 1. While the performance degradation is notable (-18.7% for NGINX and -2.3% for MySQL), the primary issue lies in container bloat. M2C produced containers up to **30$\times$ larger** for NGINX and **10$\times$ larger** for MySQL compared to official Docker images.

This bloat stems from the tool's filesystem-based analysis, which captures all system-level dependencies instead of focusing on application-specific ones. These limitations highlight the need for a more selective migration method to preserve the benefits of containerization, as discussed in the following section.

Application	VM Perf.	Cont. Perf.	Change (%)	Off. Size	M2C Size
NGINX	20,811 req/s	16,926 req/s	−18.7	192 MB	5.97 GB
MySQL	3,266 evt/s	3,190 evt/s	−2.3	603 MB	6.5 GB

Table 1: Performance and container size comparison post-M2C migration.

4.2 Process-Level Analysis

To mitigate excessive container bloat, a *process-level analysis* method is proposed that treats applications as black boxes, represented by their underlying Unix process(es). In Linux, processes encapsulate an application's key characteristics, including runtime dependencies (e.g., libraries, configuration files) and operational parameters (e.g., open ports, environment variables). These details provide a practical foundation for containerization.

By focusing on active processes, the aim is to capture **only** the dependencies actually used at runtime by an application and avoid redundant system-level components. This transition from system-wide to *process-specific* analysis aims for a more efficient, targeted replatforming strategy.

4.3 Dependency Collection

4.3.1 Static Analysis

The proposed method first captures a static snapshot of each process by examining the `/proc` filesystem, which contains detailed information about running processes in Linux. This snapshot includes user/group IDs, executable, working directory, environment variables, and other critical process metadata for containerization, as presented in Table 2.

Static analysis establishes a baseline of resources specified at startup but may overlook dependencies that are loaded conditionally or at runtime. Identifying these potential gaps motivates the subsequent dynamic analysis step to ensure a more reliable dependency discovery.

Dependency	Description	Source
User	Process owner. Determines permissions.	`/proc/[pid]/status`
Group	Group of the process. Affects group access.	`/proc/[pid]/status`
Executable	Path to the binary or script executed.	`/proc/[pid]/exe`
Command-line	Startup arguments for the process.	`/proc/[pid]/cmdline`
Working Directory	Current directory of the process.	`/proc/[pid]/cwd`
Environment Vars	Runtime configuration settings.	`/proc/[pid]/environ`
Unix Sockets	Unix domain sockets used by the process.	`/proc/net/unix`
TCP Ports	TCP ports open for connections.	`/proc/net/tcp`
UDP Ports	UDP ports open for datagrams.	`/proc/net/udp`
CPU Usage	CPU resources consumed.	`/proc/[pid]/stat`
Memory Usage	Allocated memory.	`/proc/[pid]/statm`
Disk I/O	Disk read and write activity.	`/proc/[pid]/io`

Table 2: Static metadata for Linux processes.

4.3.2 Dynamic Analysis

Runtime tracing of the process's behavior is introduced to capture transient dependencies loaded during execution (e.g., libraries, plugins, or HTML assets).

For example, using a tracing tool like *strace* [17] exposed on-demand library loading in MySQL upon accessing its administration console. In a separate case, HTML files required by NGINX were identified only after receiving specific HTTP requests to serve them. Monitoring file access in real time provides insight into dependencies that are not evident from static analysis alone.

Combining the static baseline with dynamically observed dependencies offers a more holistic and complete overview of the application's runtime footprint.

4.4 Dependency Mapping

Following dependency collection, redundant or non-essential elements are filtered out, and the remaining dependencies are mapped into a corresponding Docker configuration.

Essential components (e.g., libraries, configuration files) are compiled into a minimal filesystem tree, as illustrated in Figure 1. Process metadata (as shown in Table 2) is then applied to generate a streamlined Dockerfile.

Packaging only validated runtime elements ensures the container image includes only what the application actively uses. The resulting containers were then tested to ensure they meet functional and performance requirements.

```
application-profile
├── etc
│   ├── nginx
│   └── ssl
├── lib
│   └── x86_64-linux-gnu
│       ├── libc.so.6
│       ├── libssl.so.3
│       └── [...]
├── usr
│   ├── lib
│   │   └── ssl
│   ├── sbin
│   │   └── nginx
│   └── share
│       └── ca-certificates
└── var
    ├── lib
    │   └── nginx
    ├── log
    │   └── nginx
    └── www
        └── html
```

Figure 1: Minimal filesystem for NGINX.

4.5 Testing and Validation

NGINX and MySQL were used to evaluate the proposed approach, each application was deployed on an Ubuntu VM and subjected to both static and dynamic data collection. The resulting docker containers underwent performance and functionality tests.

Both containerized applications correctly performed their primary functions: NGINX served web pages, while MySQL executed SQL queries. Table 3 compares performance metrics and resulting images were significantly smaller in size relative to the ones created by Google's Migrate to Containers.

After confirming feasibility through these case studies, a CLI application was developed as a proof-of-concept to automate the designed workflow.

Application	VM Perf.	Cont. Perf.	Change (%)	Off. Size	Gen. Size
NGINX	180,708 req/s	181,094 req/s	$+0.21$	192 MB	89 MB
MySQL	3,954 evt/s	4,070 evt/s	$+2.93$	603 MB	449 MB

Table 3: Performance and container size comparison after process-based migration.

4.6 Automation

A command-line tool was prototyped to demonstrate how the process-level approach can be automated for real-world environments. The tool provides two main actions: `profile`, which collects both static and runtime data from the target process, and `dockerize`, which uses these discoveries to generate a Docker configuration and minimal filesystem layout.

By leveraging the designed data collection approach, the tool minimizes manual input requirements, it relies primarily on the process ID of the application to be migrated. This successful implementation further validates the practicality of process-level profiling to streamline container migrations. The prototype, including implementation details and usage instructions, is available on GitHub [18].

5 Discussion

This study demonstrates the feasibility of process-level profiling as a means for automating the migration of Linux-based VM workloads to containers. By focusing on application runtime behavior rather than static filesystem analysis, the method addresses excessive container size commonly seen in existing lift-and-shift solutions.

Compared to Google M2C, which produced container image of **5.97 GB** for NGINX, this novel approach yielded an **89 MB** image while maintaining performance comparable to the original

VM (+0.21%). These results indicate that minimal, application-specific dependency sets can be generated by focusing on a processes-level analysis, thereby addressing the need for a more selective and efficient migration strategy.

By requiring only the main process ID of the target application, this black-box approach has the potential to reduce prerequisite knowledge, therefore it can enable organizations with limited expertise in containerization to efficiently migrate applications.

The successful application of this method to both stateless (NGINX) and stateful (MySQL) workloads highlights its versatility and potential for broader adoption across diverse scenarios.

Moreover, the development of a command-line tool further validates the scalability of the framework in real-world settings, offering a pathway to more streamlined and accessible containerization processes.

While the proposed methodology demonstrates significant potential, certain limitations warrant further exploration. This research focuses exclusively on Linux-based VMs and assumes compatibility with Ubuntu. While this approach simplifies testing and validation, its applicability to other Linux distributions or non-Linux environments remains untested and requires further investigation to improve generalizability.

Further, dynamic analysis relies on runtime scenarios and may miss edge-case dependencies if specific behaviors are not triggered during profiling. Non-deterministic applications, such as web servers with dynamic user interactions, are particularly susceptible, while deterministic ones like databases are less affected. Incorporating machine-learning techniques could enhance the profiling process and improve coverage.

Additionally, the current method was validated using single-container applications (NGINX and MySQL), which may not fully represent the complexities of distributed or multi-process systems. Future efforts should focus on validating the framework in these more demanding use cases to ensure scalability and reliability.

Applications, dependent on hardware-components, or those with licensing constraints remain outside the scope of this method, as they are not suitable candidates for containerization. Developing a pre-migration assessment framework to identify incompatible workloads could broaden the applicability of the proposed solution.

Lastly, the security implications of the generated containers have not been thoroughly examined. Future research should include comprehensive security assessments to ensure the containers maintain or exceed the security measures of their source VMs.

6 Conclusion

This research presents a novel process-level profiling approach to automate the migration of applications from Linux-based VMs to containers. By focusing on runtime behavior, this method captures only the essential dependencies required for efficient and minimal container configurations. Case studies with NGINX and MySQL demonstrated that this approach can reduce container size by up to **98%** compared to Google Migrate to Containers, while maintaining or improving performance over the original application setup.

Despite these promising results, the study focused on Ubuntu-based environments and tested only two representative workloads (NGINX and MySQL). Future work should validate the framework in larger, more diverse settings, including distributed and multi-process systems. Further, non-deterministic applications, which rely on specific runtime behaviors, present a challenge as some dependencies may not surface during profiling. Machine learning techniques could be explored to address these gaps and improve profiling coverage. Finally, the security implications of selectively packaging dependencies should be thoroughly investigated to ensure safe containerization.

The proposed approach has significant implications for the domain of cloud migration. It has the potential to enable organizations to efficiently replatform applications without requiring extensive containerization expertise. Future research should focus on extending compatibility across Linux distributions, refining methods for complex workloads, and incorporating premigration assessments to identify unsuitable applications.

Ultimately, this study lays the foundation for advancing automated, efficient, and accessible container migration solutions.

References

[1] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172, 2015. doi:10.1109/ISPASS.2015.7095802.

[2] F. J. Meng, J. M. Xu, M. Guo, C. S. Li, H. Wang, and X. Zhuo. A generic framework for application configuration discovery with pluggable knowledge. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 236–243, 2013. doi:10.1109/CLOUD.2013.83.

[3] S. Nadgowda, S. Duri, C. Isci, and V. Mann. Columbus: Filesystem tree introspection for software discovery. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pages 67–74, 2017. doi:10.1109/IC2E.2017.24.

[4] H. Ho, D. Gordon, A. Kalia, J. Xiao, and M. Vukovic. Janus: A tool to modernize legacy applications to containers. In *Service-Oriented Computing – ICSOC 2019 Workshops*, pages 304–307, 2019. doi:10.1007/978-3-030-45989-5_35.

[5] H. Wang, F. J. Meng, X. Zhuo, L. Yang, C. S. Li, and J. M. Xu. Learning from metadata: A fuzzy token matching based configuration file discovery approach. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 405–412, 2015. doi:10.1109/CLOUD.2015.61.

[6] A. Byrne, S. L. Allen, H. Ho, S. Duri, and A. K. Coskun. Praxi: Cloud software discovery that learns from practice. In *Proceedings of the 21st International Middleware Conference Demos and Posters*, pages 5–6, 2020. doi:10.1145/3429357.3430525.

[7] Amazon Web Services. AWS App2Container, 2024. Accessed Oct. 25, 2024. URL: https://aws.amazon.com/app2container/.

[8] Google. Google Migrate to Containers, 2024. Accessed Oct. 25, 2024. URL: https://cloud.google.com/products/cloud-migration/containers.

[9] S. K. Garg, J. Lakshmi, and J. Johny. Migrating vm workloads to containers: Issues and challenges. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 778–785, 2018. doi:10.1109/CLOUD.2018.00104.

[10] NGINX. NGINX: High-Performance Load Balancer, Web Server, and Reverse Proxy, 2024. Accessed Dec. 16, 2024. URL: https://nginx.org/.

[11] MySQL. MySQL: The World's Most Popular Open-Source Database, 2024. Accessed Dec. 16, 2024. URL: `https://www.mysql.com/`.

[12] M. Imran, M. El-Sayed, U. M. Khan, A. Bocci, M. Tosi, and D. Giordano. Migration of cmsweb cluster at cern to kubernetes: A comprehensive study. *Cluster Computing*, 24(4):3085–3099, 2021. doi:10.1007/s10586-021-03295-3.

[13] Equinix Blog. How to Solve for Migrating Virtual Machines to Containers, Aug. 30 2021. Accessed Oct. 19, 2024. URL: `https://blog.equinix.com/blog/2021/08/30/how-to-solve-for-migrating-virtual-machines-to-containers/`.

[14] Spiceworks. Steps to Migrate from VM to Container. Accessed Oct. 19, 2024. URL: `https://www.spiceworks.com/tech/devops/articles/steps-to-migrate-from-vm-to-container/`.

[15] Y. Kumar, S. N, and H. K. S. Review of cloud migration strategies: Exploring advantages, challenges and cost analysis. *International Journal of Scientific Research in Engineering and Management (IJSREM)*, 8(6):1–7, 2024. doi:10.55041/IJSREM35549.

[16] KubeVirt. Building a virtualization API for Kubernetes, 2024. Accessed Oct. 25, 2024. URL: `https://kubevirt.io/`.

[17] strace. strace - diagnostic, debugging and instructional userspace tracer for Linux, 2024. Accessed Dec. 26, 2024. URL: `https://strace.io/`.

[18] S. Petkov. Command-Line Tool for Application Profiling and Containerization, 2025. Accessed Jan. 4, 2025. URL: `https://github.com/stassig/application-profiling`.