Container in Depth: A Comprehensive Approach to Resolving Linux Packaging and Dependency Issues

Simon Zam, *24256816, Author, Auckland University of Technology,*
Dr. Jian Yu, *Supervisor, Associate Professor, Auckland University of Technology*

Abstract

Analyse and review the low-level container architecture, finds a solution for various Linux packaging systems, different operating systems with various packaging designs, focus on unifying the branching of Linux packaging system by utilising container environment.

Index Terms

Linux, Unix, Packaging, Repository, Operating System, Containerisation, Open-source, Dependency, Container, Podman, Docker, Kernel.

I. Introduction and Background

A. Introduction

LINUX dependency hell has been a problem since the dawn of the Linux operating system. Like Unix based Linux, the Windows NT based Windows operating system also has a DLL hell problem. Unlike Windows, which is a closed source and follows a linear development model, Linux's strength lies in its customisability and the freedom it offers to users. However, this flexibility comes at the cost of branching architecture with numerous distributions, ranging from Alpine, Arch, Debian, Slackware, SUSE, Red Hat, Android and many others. Different distributions have different packaging systems with different system designs, and maintaining packages for all the numerous systems has been a problem for software developers. Some distributions even have multiple packaging systems, such as Debian-based systems that use '.deb' packages and are managed by tools such as 'apt' or 'dpkg', while Red Hat-based systems use '.rpm' packages managed by 'yum' or 'dnf'. This diversity in packaging systems becomes a hassle for developers to produce applications to cover all of this. To make matters worse, many new Linux distributions are rolling out every day which use different packaging styles.

Windows NT partially solved DLL hell problem by Static linking: statically link own DLLs for each application and Portable Packages: applications bundle required DLLs with its own private DLLs. Unlike Windows, Linux operating systems are diverse. Various non-profit organisations and proprietary companies developed different kinds of Linux based operating systems with their own packaging system which becomes incompatible with one another.

This research paper aims to solve this problem by unionising all the packaging systems into containers which can be run on every Linux operating system with consistency and linearity. The concept of containers present a viable solution where containers are lightweight and portable, encapsulating an application and its dependencies into a single image unit that can run easily across every environment. This is achieved with containerisation technologies like Docker, Podman, LXC, LXD, containerd which leverages Linux kernel features such as namespaces, cgroups and jailing. For future reference, this research prototype is based on Podman.

B. Background and Motivation

The Linux ecosystem has long been a complex with packaging and dependency management. As a user who used multiple Linux distributions, starting the journey from Debian to various other distributions, the author has experienced the frustrations of dealing with different package managers. Each package manager has their own strengths and weaknesses, handling them in their own ways. While some distributions excel in certain areas such as package availability or update frequently, others usually leave the job of compiling the source code to user for compatibility and openness. E.g. Linux from Scratch, Gentoo, Arch (AUR).

As an open-source developer, this problem becomes frustrating and tedious. The diversity of Linux distributions becomes a double-edged sword where developers need to take account of all packaging systems, dependency chains, and versioning schemes. Even for standalone binaries, sometimes, the application has to rely on larger libraries. The advent of containerisation, particularly with Docker, has been a game-changer in this space.

By encapsulating an application and its dependencies within a container, developers can ensure reproducible and reliable application across different environments. The author's experience with containerisation began with a project where the author needed to deploy an application across multiple different servers. Containers not only simplified the deployment process but were also able to help the application run smoothly on any underlying Linux operating system. This experience sparked an idea that what if containers could be used to deliver all applications not only CLI (Command Line) but also GUI (Graphical User Interface), effectively decoupling them from the underlying distribution package manager?

The idea of using containers to deliver applications is both exciting and promising. However, there are some hurdles to overcome. As someone who has been solving the complexities of Linux for years, the author is eager to explore this approach and see how it might change the future of application delivery in the Linux ecosystem.

II. Literature Review

Eelco Dolstra and Andres Loh [1] [2] introduced NixOS, a purely functional Linux distribution, offers a distinct paradigm for managing software packages and system configurations. Unlike conventional distributions that manage packages imperatively, NixOS use the Nix package manager to build and manage all software in isolated, cryptographically hashed environments. This approach ensures that every package and its dependencies are uniquely identified and isolated, eliminating ”dependency hell” and guaranteeing bit-for-bit reproducibility across different systems. Furthermore, its declarative system configuration allows for atomic upgrades and rollbacks, which makes robust mechanisms for system stability and recovery from failed updates. However, a key distinction from container technologies, which are the focus of this paper, is that NixOS primarily addresses dependency and reproducibility challenges at the system and package build level. It does not inherently provide the same granular runtime process isolation, resource limiting, or lightweight application portability across arbitrary host operating systems that container runtime like Docker or Podman. While Nix can be used to build container images, its native main strength in managing the entire operating system’s state, rather than encapsulating individual applications for deployment in diverse, pre-existing host environments.

In ”A Study of Application Sandbox Policies in Linux,” [3] provide the first comprehensive analysis of application sandbox policies within the emerging Flatpak and Snap ecosystems on Linux desktop operating systems. The paper investigates 283 matching applications across both platforms, revealing that Flatpak and Snap significantly enhance Linux platform security, with a high percentage of applications (90.1% of Snaps and 58.3% of Flatpaks) contained by tamper-proof sandboxes. While the study finds evidence that package maintainers actively strive for least-privilege policies, often favouring fine-grained permissions, it also highlights the inherent difficulty and error-proneness in defining such policies, evidenced by frequent mismatches and instances of over- or under-privilege between Flatpak and Snap versions of the same application. Although the paper focus on theory to address the packaging security, it does not show the result and prototype to implement in real application.

This paper by Antunes and Vardasca (2014) [4] presents a performance analysis between FreeBSD jail environments and traditional virtualisation solutions for cloud computing implementations. The authors conducted tests using SSH, HTTP, and NetBIOS protocols to evaluate resource consumption and data access performance across different cases, including single and concurrent access patterns. Their findings demonstrate that jail environments significantly outperform virtualised solutions, achieving performance improvements of over 50% while consuming fewer hardware resources which requiring only minimal CPU (2.5%), memory (20MB), and storage (150MB) compared to virtual machines that demand considerably more resources for both the hypervisor and individual instances. The study shows that jail environments maintain consistent performance regardless of the number of running instances, whereas virtual machines experience performance degradation as additional instances are deployed. However, the authors acknowledge the limitation of jail-based solutions such as the loss of operating system flexibility while all containers must share the same kernel as the host system. Despite the constraint, they conclude that for specific cloud computing scenarios focused on data access and service provision rather than diverse operating system requirements, jail environments offer great alternative that can enhance performance while reducing infrastructure costs

In the paper ”OPIUM: Optimal Package Install/Uninstall Manager,” Tucker et al. [5] address the limitations of traditional Linux package managers like apt-get and yum, which rely on heuristics, which cause inability to optimise installations based on user preferences. The authors propose OPIUM, a package management tool that utilise SAT, pseudo-boolean, and Integer Linear Programming solvers to model package dependencies and conflicts as constraint and optimisation problems. OPIUM offers two key advantages, it guarantee a solution if one exists, and it can optimise installations according to user-defined objective functions, such as minimising download size or maximising package popularity. Through an evaluation against Debian’s apt-get using 600 real-world installation traces, the authors demonstrate that OPIUM is practically usable despite being slower than apt-get, OPIUM features provide concrete benefits where successfully handling cases where apt-get fails and finding better solutions in terms of packages removed and data downloaded.

Ryding and Johansson (2020), in their study ”Jails vs Docker: A performance comparison of different container technologies,” [6] conducted a performance evaluation comparing FreeBSD Jails and Docker on Linux. Utilising benchmarking tools for CPU, memory, disk I/O, network, and startup times across varying container counts (1 to 32), they aimed to determine the efficiency and stability of each technology. The findings revealed that Docker generally outperformed Jails in CPU efficiency and disk read performance across most test cases. Notably, Docker demonstrated better utilisation of shared resources and higher stability (lower result variance) when running multiple containers, contradicting some earlier research. Conversely, Jails proved significantly faster in container startup times and showed superior network performance with a low number of containers. The authors concluded that while Docker appears superior for overall performance and stability, Jails remain competitive or better in specific scenarios like startup speed and low-container network performance, also discussing how feature adoption (like Docker’s platform independence and repository system) could benefit Jails. A key limitation acknowledged by the authors was the use of the default file-systems (ZFS for FreeBSD and EXT4 for Linux), which may have influenced the disk I/O performance results.

III. Research Questions and Scoping

A. Research Questions

We articulate the research questions that directly address the core problem of Linux packaging and dependency management, as introduced in the introduction, and clarify the specific gap identified in the literature review regarding the potential of a comprehensive container-based solution. We also define the scope of this study to establish clear boundaries for the investigation, guiding the methodology and expected outcomes. Based on the identified challenges with traditional Linux packaging and the potential of container technology, this research seeks to answer the following questions:

RQ1: To what extent does a comprehensive containerisation approach, encompassing both command-line and graphical user interface (GUI) applications, effectively mitigate the ”dependency hell” problem in Linux compared to traditional packaging methods, and what specific dependency challenges persist within this model?

RQ2: What are the key security implications and necessary mitigation strategies when implementing a universal container-based system for managing diverse Linux applications, with a particular focus on the unique security considerations introduced by integrating graphical user interface (GUI) applications within this framework?

RQ3: What are the primary design considerations for building a container-based application packaging system for a comprehensive Linux environment (including GUI applications), and how does this approach impact the end-user experience in terms of application installation, execution, and integration?

B. Aim

The primary aim of this research is to propose and critically evaluate the feasibility and effectiveness of adopting a comprehensive container-based model for resolving Linux application packaging and dependency issues across the entire application spectrum, including GUI applications. Specifically, the study aims to assess the potential of containers to significantly reduce or eliminate dependency conflicts for all types of Linux applications and identify and analyse the security implications inherent in a universal containerised application ecosystem, proposing relevant mitigation strategies, particularly concerning GUI application integration. This research also aim to explore the architectural and design considerations required for such a system and evaluate its potential impact on the end-user experience.

Through these aims, the research seeks to determine if a comprehensive container strategy is a viable and beneficial paradigm shift for Linux application management.

C. Scope of the Study

The scope of this study is defined to provide a focused and achievable investigation into the proposed comprehensive container-based packaging model.

Included:

- Analysis of the ”dependency hell” problem within traditional Linux packaging (e.g., package managers like APT, YUM/DNF).
- Investigation of how container technology (using Docker and Podman as representative examples) fundamentally addresses dependency isolation.
- Examination of the challenges and potential solutions for containerising diverse application types, with a specific focus on the unique requirements and complexities of graphical user interface (GUI) applications (e.g., display server integration, desktop environment interaction).
- Analysis of the security implications of running a wide range of applications, including GUI apps, within containers, covering aspects like isolation, privilege escalation, and host system interaction.
- Exploration of high-level architectural and design principles for a system that utilises containers as the primary application of packaging and distribution mechanisms.
- Consideration of the end-user experience, including aspects like application discovery, installation, updates, startup times, and integration with the host environment, based on qualitative analysis and conceptual metrics.

Excluded:

- In-depth technical analysis of the internal implementation details of specific container runtime or image formats (unless directly relevant to dependency or security mechanisms discussed).
- Detailed performance benchmarking of containerised applications versus native installed applications beyond the scope of perceived user experience (e.g., micro-benchmarks not related to typical application usage).
- Comprehensive comparison with all alternative modern packaging formats (e.g., Snap, Flatpak) – while they may be mentioned in the literature review as related work, the focus is specifically on the universal container model.
- Analysis of enterprise-scale container orchestration platforms (e.g., Kubernetes) – the focus is on the application packaging and execution model for a single-user or desktop environment.
- Development of a fully functional prototype system – the study is primarily analytical and conceptual, though it may draw upon existing tools and experiments.
- Detailed legal or licensing implications of software distribution via containers.

D. Contributions of the study

This study aims to contribute to the understanding of Linux application management by providing a comprehensive analysis of the potential and challenges of using container technology as a universal packaging and dependency resolution mechanism, specifically addressing the often-overlooked complexities of incorporating GUI applications into such a model. By evaluating its effectiveness against dependency hell, analysing its security landscape, and exploring design and user experience considerations, this research offers valuable insights for future operating system design, application distribution strategies, and the ongoing effort to simplify the Linux desktop experience. It provides a structured framework for evaluating containerisation not just as an isolation tool for servers, but as a fundamental shift in how all applications, including graphical ones, can be delivered and managed on Linux systems.

IV. Research Method

A. Research Plan

1) Foundational Research Phase: This initial phase involves a comprehensive review of existing literature to:

- Identify and Characterise Current Challenges: Systematically analyse documented issues related to traditional Linux packaging (e.g., DEB, RPM), universal package formats (e.g., Flatpak, Snap, AppImage), and existing dependency management systems. This includes ”dependency hell,” library version conflicts, platform inconsistencies, and difficulties in application portability.
- Survey Existing Containerisation Solutions: Review the architecture, benefits, and limitations of prevalent containerisation technologies like Docker and Podman, focusing on their current use cases, particularly in server-side deployments.
- Examine Current Approaches to UI Application Containerisation: Investigate existing techniques, tools (e.g., X11 forwarding, Wayland integration, PipeWire for audio/video), and community efforts aimed at running graphical applications within containers. This will highlight current best practices and unresolved challenges.
- Establish a Baseline: Define the current state-of-the-art and identify gaps that the proposed universal containerisation approach aims to address.
- Choose a methodology: We will use systematic review methodology for this paper as it is the most compatible with our research questions and objectives. This methodology is more effective than others and strongly align with our approach.

2) Proof-of-Concept (PoC) Prototype Development and Implementation Phase:

- Selection of Target Applications: A curated set of applications will be chosen to represent various categories:
  - Complex UI applications: e.g., IDEs, web browsers, office suites, multimedia applications.
  - Applications with specific hardware dependencies: e.g., applications requiring GPU acceleration, specific device access (e.g., webcam, microphone).
- Containerisation Strategy:
  - Technology Choice: Both Docker and Podman will be utilised to create container images, allowing for a comparison of their suitability for this universal approach. Standard Dockerfiles/Containerfiles will be developed.
  - Base Image Selection: Minimal base images (e.g., Alpine Linux, Ubuntu) will be prioritised to reduce overhead, while also considering compatibility with application dependencies.
  - UI Integration Techniques: Implementation and testing of various methods for UI rendering and interaction, including:
    - X11 socket sharing and forwarding.
    - Wayland proxying solutions.
    - Integration with host sound systems (e.g., PulseAudio, PipeWire).
  - Persistence and Configuration: Strategies for managing user data, application configurations, and ensuring seamless updates will be designed and implemented (e.g., volume mounts, environment variables).
  - Security Considerations: Implementation of best practices for container security, including user namespacing, Seccomp profiles, and AppArmor/SELinux where appropriate, to ensure that containerised UI applications do not pose undue security risks.

3) Evaluation and Comparative Analysis Phase: The containerised applications developed in Prototype Development Phase will be rigorously evaluated against several key metrics and compared with traditional installation methods and other universal packaging formats.

- Evaluation Metrics:
  - Dependency Resolution and Portability:
    - Success rate of containerising selected applications.
    - Ease of deployment across different Linux distributions (e.g., Ubuntu, Fedora, Arch Linux).
    - Absence of host system library conflicts.
  - Performance:
    - Startup Time: Time taken for the application to launch and become responsive.
    - Resource Consumption: CPU usage, RAM footprint, and disk space (image size vs. installed size of traditional packages).
    - Runtime Performance: Responsiveness and perceived smoothness of UI applications, especially those requiring graphics or multimedia capabilities. Standardised benchmarks will be used where applicable (e.g., GPU tasks).
  - Usability and User Experience (UX):
    - Seamlessly integrable with the host desktop environment (e.g., file access, theme consistency, notifications).
    - Ease of installation, update, and removal for the end-user.
  - Developer Experience (DX):
    - Complexity of creating and maintaining container definitions (Dockerfiles/Containerfiles).

    - Ease of debugging applications running within containers.
  - Isolation and Security:
    - Effectiveness of process and file-system isolation.
    - Assessment of the attack surface compared to natively installed applications.
- Comparative Baselines:
  - Native Packaging: Applications installed via the system package manager (e.g., ‘apt‘, ‘dnf‘).
  - Other Universal Formats: Comparison with existing Flatpak or Snap versions of different applications, if available and mature.
- Data Collection Methods:
  - Benchmarking Tools: Standardised tools for performance measurement (e.g., ‘time‘ command, ‘htop‘, ‘iotop‘, custom scripting for UI responsiveness), manual user testing.
  - System Logs and Monitoring: Collection of system logs and resource usage data during application execution.
  - Qualitative Assessment: Documented on ease of use and overall user/developer experience. For UI applications.
  - Reproducibility: All container definitions, test scripts, and methodologies will be documented for reproducibility.

4) Data Analysis: The data collected in Evaluation Phase will be analysed as follows:

- Quantitative data (performance metrics, resource usage) will be analysed to identify differences between the containerised version and native. Results will be shown using tables, charts, and graphs.
- Qualitative data (usability, DX) will be analysed to identify patterns, strengths, and drawbacks of the proposed containerisation approach.

B. Research Prototype Development

The prototype will be developed using Podman, developed by RedHat which is Open Container Initiative (OCI)-compliant. Podman is based on the libpod library API, which is identical to Docker and the application images will be based on the Containerfile format.

1) Prototype 1(Brave Browser): This Prototype was developed for Brave Browser, based on an Ubuntu container image. Brave Browser is based on Chromium and this application is chosen to demonstrate that all chromium based applications are able to run in a container.

```
FROM ubuntu:latest

RUN apt update && apt install -yq --no-install-recommends \
        ca-certificates \
        gnupg \
        libgl1 \
        libpulse0 \
        curl \
        libegl1-mesa \
    && curl -fsSLo /usr/share/keyrings/brave-browser-archive-keyring.gpg https://brave-browser-apt-release.s3.brave.com/brave-browser-archive-keyring.gpg \
    && echo "deb [signed-by=/usr/share/keyrings/brave-browser-archive-keyring.gpg arch=amd64] https://brave-browser-apt-release.s3.brave.com/ stable main"|tee /etc/apt/sources.list.d/brave-browser-release.list \
    && apt update && apt install -yq --no-install-recommends brave-browser \
    && fc-cache -fv \
    && rm -rf /var/lib/apt/lists/* \
    && apt autoclean && apt autoremove --purge && apt clean \
    && find / -xdev -type f -perm /u+s -exec chmod -c u-s {} \; \
    && find / -xdev -type f -perm /g+s -exec chmod -c g-s {} \; \
    && echo "default-server = unix:/run/user/1000/pulse/native\nautospawn = no\ndaemon-binary = /bin/true\nenable-shm = false" > /etc/pulse/client.conf

ENTRYPOINT [ "brave-browser" ]
```

To ensure compatibility with Chromium's sand-boxing features, which mandate non-root execution, a dedicated non-privileged user (UID 1000) was provisioned for browser operation. The environment was prepared by updating apt package repositories and installing essential browser dependencies. Specifically, libpulse0 was included for audio support, and libegl1 library for graphics drivers, enabling hardware-accelerated rendering. The Brave browser package was sourced directly from its official website to ensure authenticity and the latest stable version. Post-installation, temporary package caches were purged to optimise the container image size, and file system permissions were normalised for security and consistency.

Build and run the Containerfile using the following command.

```
podman build -t brave-browser -f Containerfile .
podman run
    --rm \
    --userns=keep-id \
    --group-add=video \
    --device=/dev/kfd \
    --device=/dev/dri \
    -e XDG_CURRENT_DESKTOP \
    -e XDG_RUNTIME_DIR \
    -e WAYLAND_DISPLAY \
    -v ${XDG_RUNTIME_DIR}/${WAYLAND_DISPLAY}:${XDG_RUNTIME_DIR}/${WAYLAND_DISPLAY} \
    -v ${XDG_RUNTIME_DIR}/pulse:${XDG_RUNTIME_DIR}/pulse \
    --transient-store \
    --shm-size=2g \
    --cap-add sys_chroot \
    brave-browser --enable-features=UseOzonePlatform --ozone-platform=wayland
```

The container only stops if we exit the program normally. To prevent this we added `--rm` to remove the container after used. To keep the container environment's permission the same as running user, the `--userns=keep-id:gid=$(id -u)` is added.

For graphic, `--group-add=video --device=/dev/kfd --device=/dev/dri` are used to access the GPU with required permission. To use the full potential of Linux display manager both Wayland and X11 are mounted with `-e XDG_CURRENT_DESTKOP -e XDG_RUNTIME_DIR -e WAYLAND_DISPLAY -v $XDG_RUNTIME_DIR/$WAYLAND_DISPLAY:$XDG_RUNTIME_DIR/$WAYLAND_DISPLAY`

For audio, pulseaudio server is shared with `-v $XDG_RUNTIME_DIR/pulse:$XDG_RUNTIME_DIR/pulse`.

The browser needs some additional permissions and shm access for better performance and used with `--shm-size=2g --cap-add sys_chroot`. The starting commands also need a flag to run specifically in Wayland by using `--enable-features=UseOzonePlatform --ozone-platform=wayland`

Additionally, `--transient-store` is used to run the container on memory only which can shorten start time.

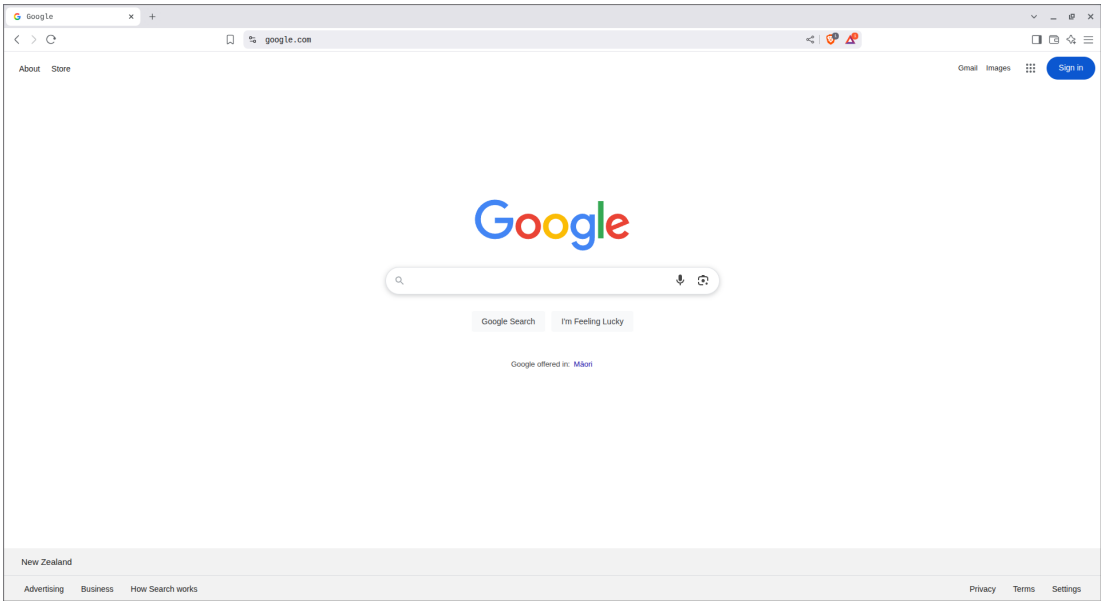


Fig. 1. Brave Browser in Container

2) Prototype 2 (Wine): Wine is a state of art application which is developed to run Windows application on Linux kernel with seamless design. In this prototype Winbox (MikorTik Router Management Software) will run on top of wine.

```
FROM alpine:latest as downloader
RUN apk add curl
RUN curl -JL https://download.mikrotik.com/routeros/winbox/3.41/winbox64.exe -o winbox.exe

FROM fedora:latest

RUN yum install wine -y --setopt=install_weak_deps=False \
    && yum clean all \
    && rm -rf /var/cache/yum \
    && find / -xdev -type f -perm /u+s -exec chmod -c u-s {} \; \
    && find / -xdev -type f -perm /g+s -exec chmod -c g-s {} \;

COPY --from=downloader --chown=user:user /winbox.exe /

WORKDIR /home/ubuntu

ENTRYPOINT [ "/usr/bin/wine","/winbox.exe" ]
```

And it can be built and run with

```
podman build -t winbox -f Containerfile .
podman run \
    --rm \
    --userns=keep-id \
    --group-add=video \
    --device=/dev/kfd \
    --device=/dev/dri \
    -e XDG_CURRENT_DESKTOP \
    -e DISPLAY \
    -v /tmp/.X11-unix:/tmp/.X11-unix
    --transient-store \
    winbox
```

Unlike the first prototype the Winbox in wine is running with X11 display server and the container base is Fedora.

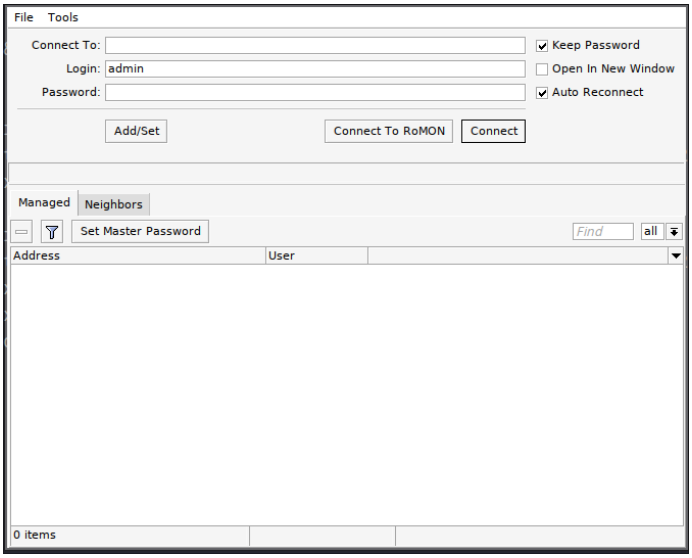


Fig. 2. Winbox on wine in Container

3) Prototype 3 (Obsidian): Obsidian is note taking app based on electron. This is a sample prototype for all electron based applications.

```
FROM alpine:latest as downloader
RUN apk add curl
RUN curl -JLO $(curl -s https://api.github.com/repos/obsidianmd/obsidian-releases/releases | grep -E ".*url
    .*amd64.deb" | cut -d '"' -f4 | head -1)

FROM ubuntu:latest

COPY --from=downloader --chown=ubuntu:ubuntu /obsidian* /obsidian.deb

RUN apt update && apt install -yq --no-install-recommends\
        libdrm2 \
        libgbm1 \
        libasound2t64 \
        libgtk-3-0 \
        libnotify4 \
        libnss3 \
        libxss1 \
        libxtst6 \
        xdg-utils \
        libatspi2.0-0 \
        libsecret-1-0 \
        ffmpeg \
        /obsidian.deb \
    && fc-cache -fv \
    && rm -rf /var/lib/apt/lists/* \
    && apt autoclean && apt autoremove --purge && apt clean \
    && find / -xdev -type f -perm /u+s -exec chmod -c u-s {} \; \
    && find / -xdev -type f -perm /g+s -exec chmod -c g-s {} \; \
    && rm -rf /obsidian.deb

ENTRYPOINT [ "obsidian" ]
```

And the command to build and run is

```
podman build -t obsidian -f Containerfile .
podman run
    --rm \
    --userns=keep-id \
    --group-add=video \
    --device=/dev/kfd \
    --device=/dev/dri \
    -e XDG_CURRENT_DESKTOP \
    -e DISPLAY \
    -v /tmp/.X11-unix:/tmp/.X11-unix
    obsidian
```

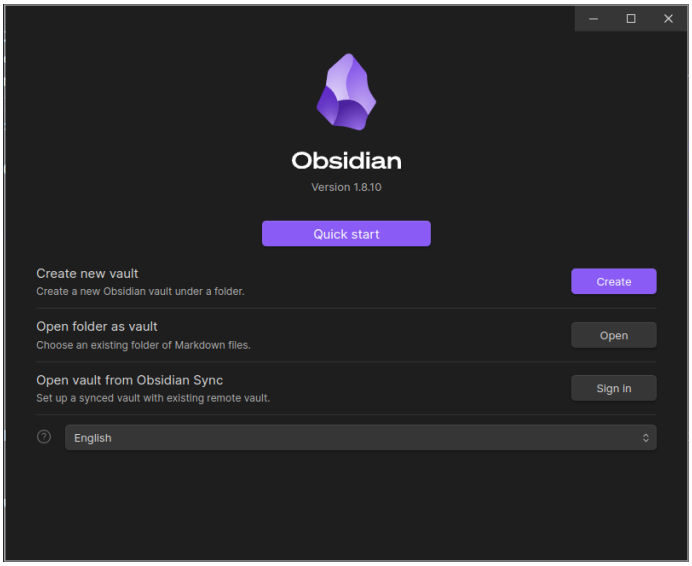


Fig. 3. Obsidian in Container

4) Prototype 4 (Zoom): Zoom is a popular meeting software and it is based on QT library for graphic.

```
FROM alpine:latest as downloader
RUN apk add curl
RUN curl -J -L -O https://zoom.us/client/latest/zoom_amd64.deb

FROM ubuntu:latest

COPY --from=downloader --chown=1000:1000 /zoom_amd64.deb /zoom_amd64.deb

RUN apt update && apt install -yq --no-install-recommends \
        libnss3 \
        libasound2t64 \
        libpulse0 \
        qt5dxcb-plugin \
        /zoom_amd64.deb \
    && fc-cache -fv \
    && rm -rf /var/lib/apt/lists/* \
    && apt autoclean && apt autoremove --purge && apt clean \
    && find / -xdev -type f -perm /u+s -exec chmod -c u-s {} \; \
    && find / -xdev -type f -perm /g+s -exec chmod -c g-s {} \; \
    && echo "default-server = unix:/run/user/1000/pulse/native\nautospawn = no\ndaemon-binary = /bin/true\
    nenable-shm = false" > /etc/pulse/client.conf

ENTRYPOINT [ "zoom" ]
```

And the command to build and run is

```
podman build -t zoom -f Containerfile .
podman run
    --rm \
    --userns=keep-id \
    --group-add=video \
    --device=/dev/kfd \
    --device=/dev/dri \
    -e XDG_CURRENT_DESKTOP \
    -e XDG_RUNTIME_DIR \
    -e WAYLAND_DISPLAY \
    -v ${XDG_RUNTIME_DIR}/${WAYLAND_DISPLAY}:${XDG_RUNTIME_DIR}/${WAYLAND_DISPLAY} \
    -v ${XDG_RUNTIME_DIR}/pulse:${XDG_RUNTIME_DIR}/pulse \
    --transient-store \
    zoom
```

Zoom application is specifically chose for prototype to demonstrate other peripheral functions such as camera and microphone.

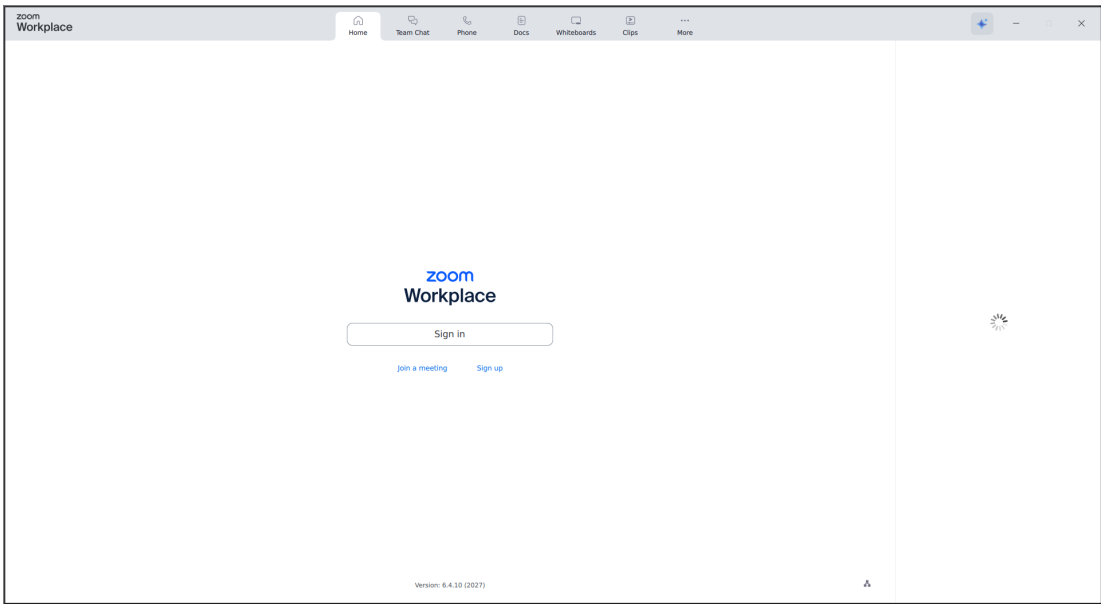


Fig. 4. Zoom in Container

5) Prototype 5 (Tor Browser): For the last prototype, we will demonstrate with Tor Browser, a privacy oriented Firefox based browser which is based on GTK library for graphic.

```
FROM alpine:latest as downloader
RUN apk add curl jq
RUN curl -fssLO $(curl https://aus1.torproject.org/torbrowser/update_3/release/downloads.json | jq -r '.
    downloads."linux-x86_64".ALL.binary') \
    && tar xvf tor-browser-*

FROM ubuntu:latest
RUN apt update && apt install -yq --no-install-recommends \
        keyboard-configuration \
        libx11-xcb1 \
        libxt6 \
        libgtk-3-0 \
        libdbus-glib-1-2 \
        file \
        libpulse0 \
        libasound2t64 \
    && fc-cache -fv \
    && rm -rf /var/lib/apt/lists/* \
    && apt autoclean && apt autoremove --purge && apt clean \
    && find / -xdev -type f -perm /u+s -exec chmod -c u-s {} \; \
    && find / -xdev -type f -perm /g+s -exec chmod -c g-s {} \; \
    && echo "default-server = unix:/run/user/1000/pulse/native\nautospawn = no\ndaemon-binary = /bin/true\
    nenable-shm = false" > /etc/pulse/client.conf

COPY --from=downloader --chown=1000:1000 /tor-browser/ /tor

ENTRYPOINT [ "/tor/Browser/start-tor-browser" ]
```

And the command to build and run is

```
podman build -t tor-browser -f Containerfile .
podman run
    --rm \
    --userns=keep-id \
    --group-add=video \
    --device=/dev/kfd \
    --device=/dev/dri \
    -e XDG_CURRENT_DESKTOP \
    -e XDG_RUNTIME_DIR \
    -e WAYLAND_DISPLAY \
    -e MOZ_ENABLE_WAYLAND=1 \
    -v ${XDG_RUNTIME_DIR}/${WAYLAND_DISPLAY}:${XDG_RUNTIME_DIR}/${WAYLAND_DISPLAY} \
    -v ${XDG_RUNTIME_DIR}/pulse:${XDG_RUNTIME_DIR}/pulse \
    --transient-store \
    tor-browser
```

The flag `-e MOZ_ENABLE_WAYLAND=1` is specifically added to force the browser to run on Wayland mode.

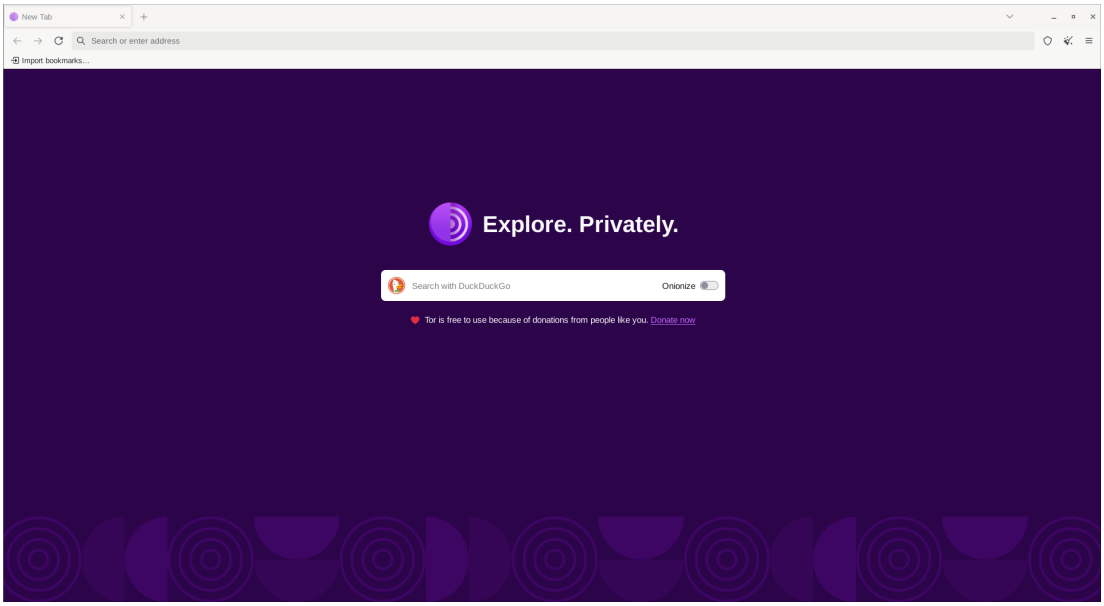


Fig. 5. Tor Browser in Container

The above prototypes are samples demonstrations using Podman container to run various applications with different configurations and environments.

V. Findings, Analysis and Evaluation

The results of the study are directly linked to the research problems which are mentioned in the introduction and research questions.

A. Findings

1) Mitigation of Host-Level Dependency Issues (Addressing RQ1): The development and successful execution of diverse GUI applications within containers demonstrated a significant mitigation of the ”dependency hell” problem at the host operating system level. By encapsulating applications and their specific library dependencies within the container image (Containerfiles explicitly listed required libraries like libgl1, libpulse0, libegl1-mesa, libnss3, libasound2t64, libgtk-3-0, etc.), the research confirmed that applications could run independently of the host distribution's installed library versions or packaging system. This approach effectively decouples the application from the host environment's dependency chain, allowing the same container image to theoretically run on any Linux distribution capable of running the container runtime (Podman), provided the necessary host-level integration points (display server, audio, devices) are accessible.

However, dependency management is not eliminated but rather shifted to the container image build process which means the dependency management is shifted to developer. Developers must now define and manage dependencies within the Containerfile. This solution is logical since the developer has the exact knowledge of which packages are required for the application developed.

2) Security Implications and Integration Challenges (Addressing RQ2): Containerisation inherently provides a layer of isolation using Linux kernel features like namespaces and cgroups. This isolates the application's file-system, processes, and network from the host system to a significant degree, improving security compared to natively installing potentially untrusted software directly onto the host. The use of user namespaces (–userns=keep-id) in the prototypes further enhanced security by preventing the container process from running as root on the host. Normalising file permissions within the container image also contributed to a more secure default posture.

However, integrating GUI applications necessitates granting the container access to host resources such as the display server (X11 or Wayland sockets), audio system (PulseAudio socket), and graphics hardware (/dev/dri, /dev/kfd, –group-add=video). These access points represent potential vectors for security vulnerabilities if the containerised application is compromised. But in this hypothetical scenario, if the bad actors can successfully infiltrate the application inside the container, they definitely can penetrate the application inside the host layer. Comparing to the native layer, the container gives extra layer to halt the attack.

To partially mitigate the potential security vulnerability, the container support more more granular security policies (e.g., Seccomp profiles, AppArmor/SELinux policies specific to containerised GUI apps) to the risks introduced in the container's isolation.

3) Design Considerations and End-User Experience Impact (Addressing RQ3): The prototype development process shows several key design considerations for a container-based application packaging system:

Image Structure: Images need to include the application binary and all its runtime dependencies not provided by a minimal base image. Multi-stage builds (used in Prototypes 2, 3, 4, 5) are effective for separating build dependencies from runtime dependencies and minimising final image size. Dependency Management: Shifting dependency management to the Containerfile requires a clear definition of required libraries for each application. Using a robust base image (like Ubuntu in most prototypes) simplifies this by providing common libraries, but increases image size. Host Integration: Robust mechanisms are required to handle the variability of host display servers (X11 vs. Wayland), audio systems, and hardware access. The prototype Podman run commands demonstrate the complexity involved, requiring specific volume mounts, environment variables, and device/group access flags. A user-friendly system would need to abstract this complexity. Persistence: While –transient-store was used for quick testing, a real system requires a strategy for persistent user data and configuration (e.g., using named volumes).

At the current stage, applications are portable across different Linux distributions without host dependency conflicts. Installation conceptually becomes pulling/building a single image. But there are some underdeveloped parts such as the command (Podman run with numerous flags for display, audio, devices, user namespaces, etc.) is significantly more complex than simply typing an application name or clicking an icon. Integration with the host desktop environment (e.g., file pickers, system notifications, theme consistency) was not fully seamless within the scope of the prototypes and represents a major usability challenge for a general-purpose desktop application delivery system.

B. Analysis

This section provides a detailed analysis and interpretation of the findings.

1) Analysis of Dependency Mitigation: The findings shows that containerisation offers a robust solution to the ”dependency hell” problem at the host operating system level. By bundling the application and dependencies within the container image, the approach decouples the application's runtime environment from the host distribution's library (e.g., APT, YUM/DNF). This was demonstrated by the ability of the same container images (e.g., the Brave Browser container based on Ubuntu) to run on different host distributions without encountering library conflicts or requiring specific host package installations beyond the container runtime itself.

2) Analysis of Superiority: Compared to traditional methods, this approach eliminates the need for complex dependency algorithms on the host (as discussed in relation to OPIUM [5]) and avoids the potential for system-wide library version clashes. Unlike universal package formats like Flatpak or Snap, which also provide isolation, the container approach leverages standard container runtime and image formats (OCI-compliant), potentially offering broader compatibility with existing container infrastructure and workflows

3) Analysis of Security: The container isolation utilise Linux kernel features like namespaces and cgroups, offers a significant security advantage compared to installing potentially untrusted applications directly onto the host system. The prototypes demonstrated effective isolation of the application's filesystem and processes. The findings point out that if an attacker can compromise the application inside the container, they might also be able to compromise it if installed natively. The container's isolation aims to confine the damage, preventing easy lateral movement or access to the entire host system, unlike a native application with full user privileges.

4) Analysis of Design and End-User Experience: The prototype development process demonstrate design considerations for building a container-based packaging system for a comprehensive Linux environment. One of the main improvement is the image structure, where multi-stage builds minimise final image size by separating build dependencies from runtime requirements. Managing application dependencies shifts entirely to the Containerfile, requiring developers to accurately define and maintain the required libraries.

Figure 6 shows some user from different backgrounds giving feedback from using the containerised applications of noticible difference in startup time.

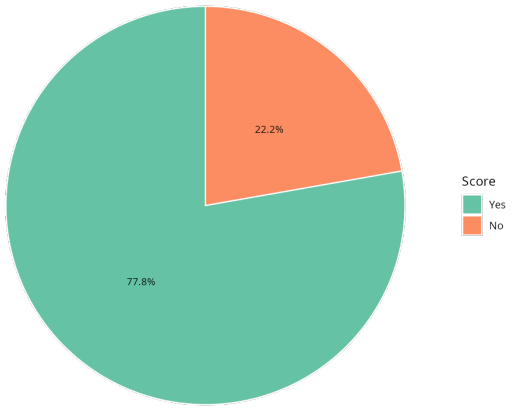
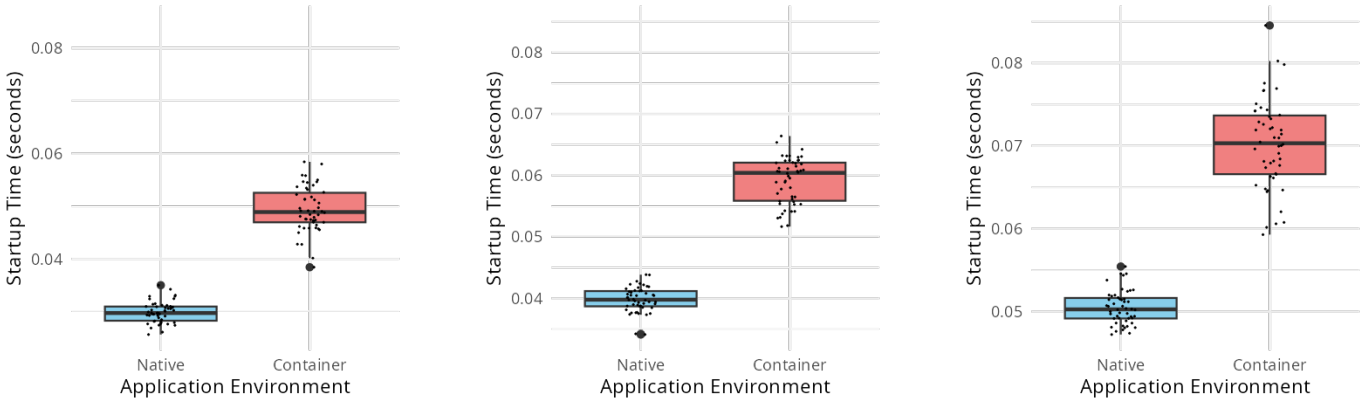


Fig. 6. Performance Difference Survey

5) Analysis of Performance Metrics: Startup Time: Figure 7 illustrates the startup time comparison between native installations and containerised executions for the selected applications (Brave Browser, Zoom, Winbox/Wine).



(a) Brave Browser (b) Zoom (c) Winbox(Wine)

Fig. 7. Startup Time Comparison

The Figure 7 indicate there are some container overhead delay for each applications. But the difference is only in the sub mill-second region which is not noticeable by end user. During the demonstration, the `--transient-store` flag is used to lessen the page storage size which might have some effect on the startup time.

As shown in Figure 8, the survey results indicate whether different users notice the difference between native and containerised application.

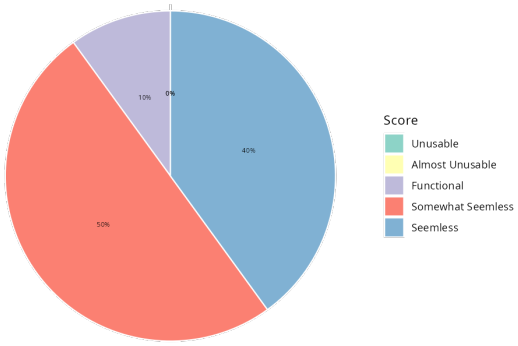
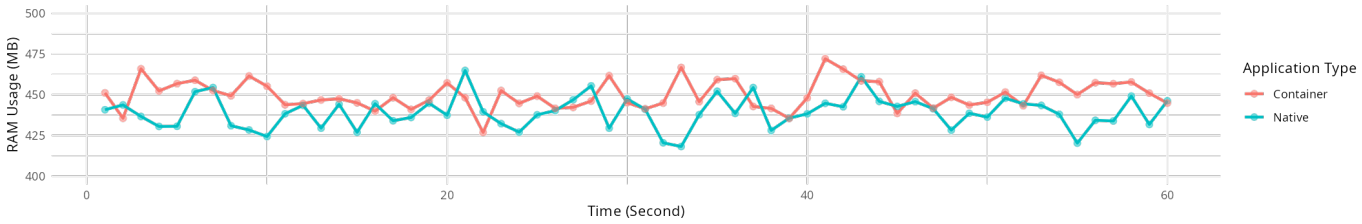
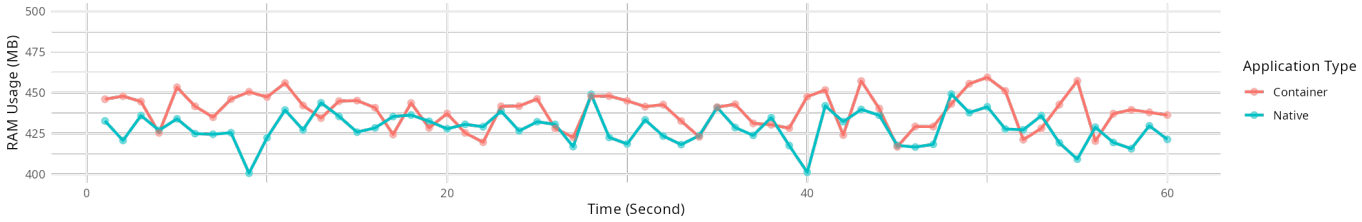


Fig. 8. Startup Time Difference Survey

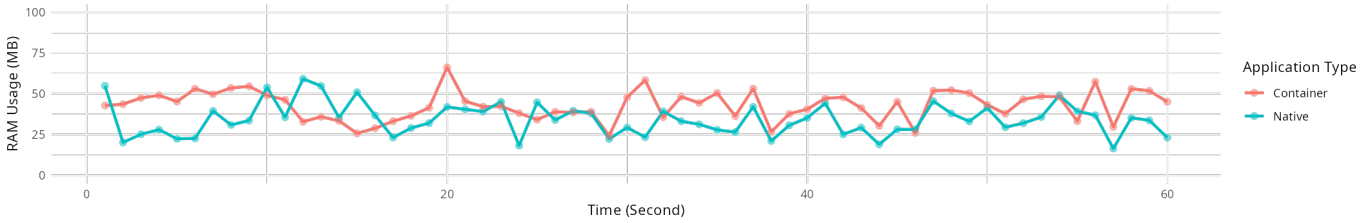
RAM Usage: Figure 8 presents the RAM usage comparison between native and containerised applications (Brave Browser, Zoom, Winbox/Wine).



(a) Brave Browser



(b) Zoom



(c) Winbox(Wine)

Fig. 9. RAM Usage Comparison

Regarding RAM usage, no significant difference was found between the native and container versions of the applications. Sudden dips were observed in the native version but not in the container version, suggesting that the container environment regulates RAM usage to maintain a consistent level.

Overall, the performance analysis suggests that while containerisation offers significant benefits in dependency management and isolation, it may introduce some overhead in startup time and RAM usage compared to highly optimised native installations. These trade-offs must be considered when evaluating the feasibility of a universal container-based system, particularly for performance-sensitive applications or resource-limited environments. The findings align somewhat with the performance trade-offs observed in the Jails vs. Docker study [6], where different container technologies showed strengths in different performance areas.

VI. Conclusion

The challenge of Linux ”dependency hell” rooting form the fragmentation of packaging systems and distribution-specific libraries, significantly hinder application portability and encountering a huge learning curve for new users. This research embarked on the potential of containerisation technology as a unifying system to resolve these issues across every spectrum of Linux branches, mainly complex graphical user interface (GUI) software. Our primary aim was to evaluate the effectiveness of decoupling applications form the host environment using container technology like Podman where offering a consistent and isolated environment.

Our findings validate the effectiveness of encapsulating applications and their required library within a single image. We demonstrated successful running of diverse GUI application across different host environments. For security, we explained the containers inherently offer a layer of isolation compared to native installation using through kernel features. We managed to shared the host resources such as display server, audio and devices where the container isolation layer still provides a border for outer attacks by utilising granular security policies.

From a design perspective, we constructed efficient images, leveraging multi-stage builds to minimise size, and developing robust, abstracted mechanisms for host resource integration. Performance analysis indicated that while containerisation introduces some minimal overhead in application startup time and potentially affects RAM usage consistency compared to highly optimised native installations, these differences were largely imperceptible to users which is suggesting that the performance cost is an acceptable trade-off for the benefits gained in dependency management and isolation.

The implications of these findings are significant for the future of Linux application distribution which can enable developers to target a single, containerised format that runs consistently across virtually any Linux distribution. This could accelerate software availability, simplify maintenance, and provide a more stable user experience by eliminating dependency conflicts.

Future research should focus on developing a user-friendly script or daemon that reduce complexities of container execution and host resource sharing which enable seamless application launching and deeper desktop integration. Further work is also needed to define standard security policies and best practices specifically tailored for containerised GUI applications to maximise isolation while maintaining necessary functionality. More extensive performance benchmarking across a wider range of hardware and application types would provide a deeper understanding of the performance trade-offs. Finally, exploring mechanisms for updates and distribution of containerised applications, using existing container registries, will be important for building a practical and scalable ecosystem. This study provides a strong foundation for exploring containerisation as a fundamental shift in Linux application delivery.

APPENDIX A
OTHER CONTAINERFILE PROTOTYPES AND EXECUTION COMMANDS

OBS Containerfile

```
FROM ubuntu:latest

RUN groupadd -g 1000 -r user \
    && useradd -u 1000 -s /bin/bash -r -G audio,video -g 1000 -m user

RUN apt update && apt install -yq --no-install-recommends \
        keyboard-configuration \
        software-properties-common \
        gpg-agent \
        ffmpeg \
        libpulse0 \
    && add-apt-repository -y ppa:obsproject/obs-studio \
    && apt update \
    && apt install -yq --no-install-recommends obs-studio \
    && fc-cache -fv \
    && rm -rf /var/lib/apt/lists/* \
    && apt autoclean && apt autoremove --purge && apt clean \
    && find / -xdev -type f -perm /u+s -exec chmod -c u-s {} \; \
    && find / -xdev -type f -perm /g+s -exec chmod -c g-s {} \; \
    && echo "default-server = unix:/run/user/1000/pulse/native\nautospawn = no\ndaemon-binary = /bin/true\
    nenable-shm = false" > /etc/pulse/client.conf

USER user

WORKDIR /home/user

ENTRYPOINT [ "obs" ]
```

Sample Container Image Builder

```
#!/bin/bash

BUILDER=podman

x=("brave" "gtkcord4" "libreoffice" "librewolf" "mullvad-browser" "obs" "qbittorrent" "staruml" "telegram"
    "tor" "ungoogled-chromium" "winbox" "zoom")

if [ "$1" = "selected" ] ; then
    for f in ${x[@]} ; do
        echo "=======================================>"
        echo "|"
        echo "|-----> Building $f"
        echo "|"
        echo "=======================================>"
        ${BUILDER} build -t peterzam/box-$f --no-cache -f https://codeberg.org/peterzam/box/raw/branch/main
    /$f/Containerfile .
    done
else
    ${BUILDER} build -t peterzam/box-$1 -f ./$1/Containerfile .
fi
```

Sample Runner

```
#!/bin/bash

if [ $# -le 1 ] || [ ${1:0:1} != '-' ]
then
    echo -e "Specify Display type"
    echo -e "Usage: \n-x (use X11 display) \n-w (use Wayland display)" >&2
    exit 1
fi


### Runner ###
runner='podman'

### Box ###
box=$2

### BOX PATH ###
BOXPATH='
/home/p/custom/box-mounts'

```

```
### name ###
NAME='
--name='${box}'-box-'${RANDOM}

### Remove ###
RM='
--rm'

### User ###
USERNS='
--userns=keep-id:gid=1000
--group-add=video' ## for gpu access

### Display ###
DISPLAY_X='
--device=/dev/kfd
--device=/dev/dri
-e DISPLAY
-v /tmp/.X11-unix:/tmp/.X11-unix'

DISPLAY_WAYLAND='
--device=/dev/kfd
--device=/dev/dri
-e XDG_CURRENT_DESKTOP
-e XDG_RUNTIME_DIR
-e WAYLAND_DISPLAY
-v '${XDG_RUNTIME_DIR}'/'${WAYLAND_DISPLAY}':'${XDG_RUNTIME_DIR}'/'${WAYLAND_DISPLAY}

### Audio ###
AUDIO_PULSE='
-v '${XDG_RUNTIME_DIR}'/pulse:'${XDG_RUNTIME_DIR}'/pulse'

### Font ###
FONT='
-v /usr/share/fonts:/usr/share/fonts:ro'

### CHROOT ###
CHROOT='
--cap-add sys_chroot'

### SHM ###
SHM='
--shm-size=2g'

### USB/Serial ###
USB='
-v /dev:/dev --device-cgroup-rule="c 188:* rmw" --device-cgroup-rule="c 81:* rmw"'

### Camera ### #video[int] for other cams
CAMERA='
--device /dev/video0'

### SHARED PATH ###
SHARED='
-v '${BOXPATH}'/shared/'${box}':/home/user/Shared'

###  Transient Storage ###
TRANSIENT='
--transient-store'

### No Network ###
NONET='
--network=none'

### QT WAYLAND###
QT_WAYLAND='
-e QT_QPA_PLATFORM=wayland'

### Mozilla WAYLAND###
MOZILLA_WAYLAND='
-e MOZ_ENABLE_WAYLAND=1'

### Chromium WAYLAND###
CHROMIUM_WAYLAND='
--enable-features=UseOzonePlatform --ozone-platform=wayland'
```

```
98
99  DEFAULT_DISPLAY=${DISPLAY_WAYLAND}
100
101
102 while getopts 'xwh:' OPTION; do
103   case "$OPTION" in
104     x)
105         DEFAULT_DISPLAY=${DISPLAY_X}
106         CHROMIUM_WAYLAND=""
107         MOZILLA_WAYLAND=""
108         QT_WAYLAND=""
109         ;;
110     w)
111         ;;
112     *)
113         echo -e "Usage: \n-x (use X11 display) \n-w (use Wayland display)" >&2
114         exit 1
115         ;;
116   esac
117 done
118
119 COMMON="${NAME} ${RM} ${USERNS} ${DEFAULT_DISPLAY} ${SHARED}"
120
121 case ${box} in
122     brave)
123         ${runner} run \
124         ${COMMON} ${AUDIO_PULSE} ${TRANSIENT} ${FONT} ${SHM} ${CHROOT} \
125         -v ${BOXPATH}/configs/brave:/home/user/.config/BraveSoftware \
126         peterzam/box-brave ${CHROMIUM_WAYLAND}
127         ;;
128     brave-incognito)
129         ${runner} run \
130         ${COMMON} ${AUDIO_PULSE} ${TRANSIENT} ${SHM} ${CHROOT} \
131         peterzam/box-brave ${CHROMIUM_WAYLAND}
132         ;;
133     gtkcord4)
134         ${runner} run \
135         ${COMMON} ${FONT} \
136         -v ${BOXPATH}/configs/gtkcord4:/home/user/.config/gtkcord4 \
137         peterzam/box-gtkcord4
138         ;;
139     libreoffice)
140         ${runner} run \
141         ${COMMON} ${AUDIO_PULSE} ${TRANSIENT} ${FONT} \
142         peterzam/box-libreoffice
143         ;;
144     librewolf)
145         ${runner} run \
146         ${COMMON} ${AUDIO_PULSE} ${TRANSIENT} ${FONT} ${CHROOT} ${MOZILLA_WAYLAND} \
147         -v ${BOXPATH}/configs/librewolf:/home/user/.librewolf \
148         peterzam/box-librewolf
149         ;;
150     mullvad-browser)
151         ${runner} run \
152         ${COMMON} ${AUDIO_PULSE} ${TRANSIENT} ${CHROOT} ${MOZILLA_WAYLAND} \
153         peterzam/box-mullvad-browser --verbose # Mullvad Browser run as background process in default
154         ;;
155     obs)
156         ${runner} run \
157         ${COMMON} ${AUDIO_PULSE} ${NONET} \
158       -v ${BOXPATH}/configs/obs:/home/user/.config/obs-studio \
159         peterzam/box-obs
160         ;;
161     obsidian)
162         ${runner} run -it \
163         ${COMMON} ${TRANSIENT} ${NONET} \
164       -v ${BOXPATH}/configs/obsidian:/home/user/.config/obsidian \
165         peterzam/box-obsidian
166         ;;
167     qbittorrent)
168         ${runner} run \
169         ${COMMON} ${TRANSIENT} ${QT_WAYLAND} \
170       -v ${BOXPATH}/configs/qbittorrent:/home/user/.config/qBittorrent \
171         peterzam/box-qbittorrent
172         ;;
173     staruml)
174         ${runner} run \
```

```
        ${COMMON} ${FONT} ${SHM} \
        peterzam/box-staruml
        ;;
    telegram)
        ${runner} run \
        ${COMMON} ${AUDIO_PULSE} ${FONT} ${QT_WAYLAND} \
      -v ${BOXPATH}/configs/telegram:/home/user/.local/share/TelegramDesktop \
        peterzam/box-telegram
        ;;
    tor)
        ${runner} run \
        ${COMMON} ${AUDIO_PULSE} ${TRANSIENT} ${CHROOT} ${MOZILLA_WAYLAND} \
        peterzam/box-tor
        ;;
    ungoogled-chromium)
        ${runner} run \
        ${COMMON} ${AUDIO_PULSE} ${TRANSIENT} ${FONT} ${SHM} ${CHROOT} \
        -v ${BOXPATH}/configs/ungoogled-chromium:/home/user/.config/chromium \
        peterzam/box-ungoogled-chromium ${CHROMIUM_WAYLAND} --disable-top-sites
        ;;
    ungoogled-chromium-incognito)
        ${runner} run \
        ${COMMON} ${AUDIO_PULSE} ${TRANSIENT} ${FONT} ${SHM} ${CHROOT} \
        peterzam/box-ungoogled-chromium ${CHROMIUM_WAYLAND}
        ;;
    winbox)
        ${runner} run \
        ${COMMON} ${TRANSIENT} \
    -v ${BOXPATH}/configs/winbox:/home/user/.wine \
        peterzam/box-winbox
        ;;
    zoom)
        ${runner} run \
        ${COMMON} ${AUDIO_PULSE} ${CHROOT} \
      -v ${BOXPATH}/configs/zoom/data:/home/user/.zoom \
      -v ${BOXPATH}/configs/zoom/config:/home/user/.config/ \
        peterzam/box-zoom
        ;;
esac


# Experimental feature - -v ${XDG_RUNTIME_DIR}/pipewire-0:${XDG_RUNTIME_DIR}/pipewire-0 \
```

APPENDIX B
BENCHMARKING AND SURVEY

1) Benchmarking Methodology Details: Exact commands used for measurement (e.g., time -p ¡command¿, htop for peak RAM, specific perf commands). Used tools : time, htop, perf, grafana, prometheus

A. Podman System Detail

```
host:
  arch: amd64
  buildahVersion: 1.40.1
  cgroupControllers:
  - cpu
  - memory
  - pids
  cgroupManager: systemd
  cgroupVersion: v2
  conmon:
    package: conmon-1:2.1.13-1
    path: /usr/bin/conmon
    version: 'conmon version 2.1.13, commit: 82de887596ed8ee6d9b2ee85e4f167f307bb569b'
  cpuUtilization:
    idlePercent: 90.4
    systemPercent: 1.1
    userPercent: 8.5
  cpus: 16
  databaseBackend: sqlite
  distribution:
    distribution: arch
    version: unknown
  eventLogger: journald
```

```
  freeLocks: 2047
  hostname: arch
  idMappings:
    gidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
    uidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
  kernel: 6.14.10-arch1-1
  linkmode: dynamic
  logDriver: journald
  memFree: 768892928
  memTotal: 15518031872
  networkBackend: netavark
  networkBackendInfo:
    backend: netavark
    dns:
      package: aardvark-dns-1.15.0-1
      path: /usr/lib/podman/aardvark-dns
      version: aardvark-dns 1.15.0
    package: netavark-1.15.2-1
    path: /usr/lib/podman/netavark
    version: netavark 1.15.2
  ociRuntime:
    name: crun
    package: crun-1.21-1
    path: /usr/bin/crun
    version: |-
      crun version 1.21
      commit: 10269840aa07fb7e6b7e1acff6198692d8ff5c88
      rundir: /run/user/1000/crun
      spec: 1.0.0
      +SYSTEMD +SELINUX +APPARMOR +CAP +SECCOMP +EBPF +CRIU +YAJL
  os: linux
  pasta:
    executable: /usr/bin/pasta
    package: passt-2025_06_06.754c6d7-1
    version: |
      pasta 2025_06_06.754c6d7
      Copyright Red Hat
      GNU General Public License, version 2 or later
        <https://www.gnu.org/licenses/old-licenses/gpl-2.0.html>
      This is free software: you are free to change and redistribute it.
      There is NO WARRANTY, to the extent permitted by law.
  remoteSocket:
    exists: true
    path: /run/user/1000/podman/podman.sock
  rootlessNetworkCmd: pasta
  security:
    apparmorEnabled: false
    capabilities: CAP_CHOWN,CAP_DAC_OVERRIDE,CAP_FOWNER,CAP_FSETID,CAP_KILL,CAP_NET_BIND_SERVICE,
    CAP_SETFCAP,CAP_SETGID,CAP_SETPCAP,CAP_SETUID,CAP_SYS_CHROOT
    rootless: true
    seccompEnabled: true
    seccompProfilePath: /etc/containers/seccomp.json
    selinuxEnabled: false
  serviceIsRemote: false
  slirp4netns:
    executable: ""
    package: ""
    version: ""
  swapFree: 12883607552
  swapTotal: 12884897792
  uptime: 3h 45m 29.00s (Approximately 0.12 days)
  variant: ""
plugins:
  authorization: null
  log:
```

```
  - k8s-file
  - none
  - passthrough
  - journald
  network:
  - bridge
  - macvlan
  - ipvlan
  volume:
  - local
registries: {}
store:
  configFile: /home/p/.config/containers/storage.conf
  containerStore:
    number: 1
    paused: 0
    running: 1
    stopped: 0
  graphDriverName: overlay
  graphOptions: {}
  graphRoot: /home/p/.local/share/containers/storage
  graphRootAllocated: 511017222144
  graphRootUsed: 207637196800
  graphStatus:
    Backing Filesystem: btrfs
    Native Overlay Diff: "true"
    Supports d_type: "true"
    Supports shifting: "false"
    Supports volatile: "true"
    Using metacopy: "false"
  imageCopyTmpDir: /var/tmp
  imageStore:
    number: 1
  runRoot: /run/user/1000/containers
  transientStore: false
  volumePath: /home/p/.local/share/containers/storage/volumes
version:
  APIVersion: 5.5.1
  Built: 1749464740
  BuiltTime: Mon Jun  9 10:25:40 2025
  GitCommit: 850db76dd78a0641eddb9ee19ee6f60d2c59bcfa
  GoVersion: go1.24.4
  Os: linux
  OsArch: linux/amd64
  Version: 5.5.1
```


B. Survey Questions

- What is your experience level with Linux? (Beginner, Intermediate, Advanced)”
- What is your experience level with containerization technologies (Docker/Podman)? (None, Basic, Intermediate, Advanced)
- On a scale of 1 to 5, how noticeable was the difference in startup time between the native and containerized versions of Brave Browser? (1=Not noticeable at all, 5=Very noticeable)
- Did you perceive any difference in responsiveness, smoothness or performance when using the containerized application compared to the native one? Please elaborate.
- What was your general impression of using applications delivered via containers? (Positive, Negative, Neutral, Why?)

C. Alternative Approaches and Technologies Considered

- Other Container Runtimes/Technologies
  - LXC/LXD: more system-level virtualization, less emphasis on application packaging portability.
  - systemd-nspawn: capabilities and limitations for GUI application isolation.
- Alternative GUI Integration Methods
  - VNC/SPICE servers within the container: adds overhead, less native.
  - SSH X forwarding: performance, setup complexity.
  - Nested Wayland compositors : Complicated and not stable.

D. Glossary of Technical Terms and Acronyms

- OCI: Open Container Initiative

- cgroups: Control Groups
- Namespaces: Linux kernel feature for resource isolation (PID, NET, MNT, UTS, IPC, USER)
- Wayland: A display server protocol
- X11: The X Window System, a display server protocol
- PulseAudio: A sound server for POSIX and Windows systems
- PipeWire: A server for handling audio, video streams, and hardware on Linux
- Seccomp: Secure Computing Mode
- AppArmor/SELinux: Linux security modules
- DLL Hell: Dynamic Link Library conflict issue on Windows
- Dependency Hell: Software dependency conflict issue on Linux
- APT: Advanced Package Tool (Debian/Ubuntu)
- YUM/DNF: Yellowdog Updater, Modified / Dandified YUM (Red Hat/Fedora)
- AUR: Arch User Repository
- CLI: Command Line Interface
- GUI: Graphical User Interface
- PoC: Proof of Concept
- UX: User Experience
- DX: Developer Experience
- Containerd: Industry-standard container runtime
- libpod: Library for Podman, providing container management functionality

ACKNOWLEDGMENT

The author wish to thank Dr. Jian Yu [7], Associate Professor, for his guidance and supervision throughout this research. The author would also like to thank the following projects for their knowledge and support.

- x.org [8]
- zoom.com [9]
- brave.com [10]
- winehq.org [11]
- kernel.org [12]
- ubuntu.com [13]
- obsidian.md [14]
- flatpak.org [15]
- appimage.org [16]
- apparmor.net [17]
- overleaf.com [18]
- mikrotik.com [19]
- snapcraft.io [20]
- r-project.org [21]
- torproject.org [22]
- alpinelinux.org [23]
- github.com/docker [24]
- fedoraproject.org [25]
- latex-project.org [26]
- github.com/containers [27]
- ggplot2.tidyverse.org [28]
- wayland.freedesktop.org [29]
- github.com/peterzam/box [30]
- freedesktop.org/wiki/Software/PulseAudio [31]

Thanks are also extended to the survey respondents and the authors of the works cited in this paper.

REFERENCES

[1] E. Dolstra and A. Löh, "Nixos: a purely functional linux distribution," in *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 367–378. [Online]. Available: https://doi.org/10.1145/1411204.1411255

[2] ——, "Nixos: a purely functional linux distribution," *SIGPLAN Not.*, vol. 43, no. 9, p. 367–378, Sep. 2008. [Online]. Available: https://doi.org/10.1145/1411203.1411255

[3] T. Dunlap, W. Enck, and B. Reaves, "A study of application sandbox policies in linux," in *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*, ser. SACMAT '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 19–30. [Online]. Available: https://doi.org/10.1145/3532105.3535016

[4] C. Antunes and R. Vardasca, "Performance of jails versus virtualization for cloud computing solutions," *Procedia Technology*, vol. 16, pp. 649–658, 2014, cENTERIS 2014 - Conference on ENTERprise Information Systems / ProjMAN 2014 - International Conference on Project MANagement / HCIST 2014 - International Conference on Health and Social Care Information Systems and Technologies. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212017314002400

[5] C. Tucker, D. Shuffelton, R. Jhala, and S. Lerner, "Opium: Optimal package install/uninstall manager," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07. USA: IEEE Computer Society, 2007, p. 178–188. [Online]. Available: https://doi.org/10.1109/ICSE.2007.59

[6] C. Ryding and R. Johansson, "Jails vs docker : A performance comparison of different container technologies," Ph.D. dissertation, Mid Sweden University, 2020. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:miun:diva-39517

[7] D. J. Yu, "Dr jian yu," 2025. [Online]. Available: https://academics.aut.ac.nz/jian.yu

[8] "X.Org Foundation," https://www.x.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.x.org/

[9] "Zoom Video Communications," https://zoom.com/, 2025, accessed: June 10, 2025. [Online]. Available: https://zoom.com/

[10] "Brave Browser," https://brave.com/, 2025, accessed: June 10, 2025. [Online]. Available: https://brave.com/

[11] "WineHQ - The Wine Project," https://www.winehq.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.winehq.org/

[12] "The Linux Kernel Archives," https://www.kernel.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.kernel.org/

[13] "Ubuntu - The Linux Distribution," https://ubuntu.com/, 2025, accessed: June 10, 2025. [Online]. Available: https://ubuntu.com/

[14] "Obsidian - A powerful knowledge base," https://obsidian.md/, 2025, accessed: June 10, 2025. [Online]. Available: https://obsidian.md/

[15] "Flatpak - A universal packaging system," https://flatpak.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://flatpak.org/

[16] "AppImage - Universal Linux applications," https://appimage.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://appimage.org/

[17] "AppArmor - Mandatory Access Control," https://apparmor.net/, 2025, accessed: June 10, 2025. [Online]. Available: https://apparmor.net/

[18] "Overleaf - Online LaTeX Editor," https://www.overleaf.com/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.overleaf.com/

[19] "MikroTik - Routers and Wireless," https://mikrotik.com/, 2025, accessed: June 10, 2025. [Online]. Available: https://mikrotik.com/

[20] "Snapcraft - Snaps for Linux," https://snapcraft.io/, 2025, accessed: June 10, 2025. [Online]. Available: https://snapcraft.io/

[21] "The R Project for Statistical Computing," https://www.r-project.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.r-project.org/

[22] "The Tor Project," https://www.torproject.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.torproject.org/

[23] "Alpine Linux," https://www.alpinelinux.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.alpinelinux.org/

[24] "Docker on GitHub," https://github.com/docker, 2025, accessed: June 10, 2025. [Online]. Available: https://github.com/docker

[25] "Fedora Project," https://fedoraproject.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://fedoraproject.org/

[26] "The LaTeX Project," https://www.latex-project.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.latex-project.org/

[27] "Containers organization on GitHub," https://github.com/containers, 2025, accessed: June 10, 2025. [Online]. Available: https://github.com/containers

[28] "ggplot2 - Elegant Graphics for Data Analysis," https://ggplot2.tidyverse.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://ggplot2.tidyverse.org/

[29] "Wayland - A display server protocol," https://wayland.freedesktop.org/, 2025, accessed: June 10, 2025. [Online]. Available: https://wayland.freedesktop.org/

[30] "peterzam/box on GitHub," https://github.com/peterzam/box, 2025, accessed: June 10, 2025. [Online]. Available: https://github.com/peterzam/box

[31] "PulseAudio on freedesktop.org Wiki," https://www.freedesktop.org/wiki/Software/PulseAudio/, 2025, accessed: June 10, 2025. [Online]. Available: https://www.freedesktop.org/wiki/Software/PulseAudio/