End of Year Internship Report
Major: Advanced Software Engineering for Digital Services

Oracle Linux Virtualization and System Testing for Oracle Cloud Infrastructure

Realized by:
NAJI Ilyass

Supervised by:
Mr. JORTI Yassine ORACLE R&D - External Mentor

National Telecommunications Regulatory Agency
National Institute of Posts and Telecommunications

Class of 2023/2024


بسم الله الرحمن الرحيم

Acknowledgements

I extend my heartfelt thanks to everyone who contributed to the success of my internship and the completion of this report.

First and foremost, I am deeply grateful to Allah the Almighty for providing me with the strength and perseverance required to excel in my academic journey.

I would like to express my sincere appreciation to **Pr. Driss ALLAKI**, my internal mentor, for his invaluable guidance, constructive feedback, and steadfast support throughout my internship. His mentorship has been crucial, and I am thankful for the opportunity to benefit from his expertise.

A special thanks goes to the team manager, **Ms. Carol TIAN**, and my external mentor, **Mr. Yassine JORTI**, for their insightful advice, patient guidance, and generosity in sharing knowledge during my time as an intern. Working closely with both of them greatly enhanced my learning experience and played a significant role in developing my professional skills.

I am also appreciative of **Ms. Fatima Zahra MOUAK** and **Mr. Youssef EL ABBASSI CHRAIBI** for their oversight of the internship program, ensuring a well-organized and smooth experience.

Thank you to everyone mentioned and to anyone else who played a part in this journey.

Abstract

This report offers a comprehensive summary of the internship project undertaken at Oracle Morocco Research & Development Center. The internship aimed to investigate and validate Oracle Linux Virtualization and System Testing procedures to ensure the stability and performance of virtualized environments. As part of the Oracle Linux Virtualization, System, and Hardware Testing team, I engaged in various tasks involving QEMU and Libvirt testing across multiple Oracle Linux environments.

The project's focus was on assessing the compatibility and functionality of virtualization modules, including QEMU and Libvirt, through rigorous testing. This involved validating their performance under different configurations and software versions, as well as implementing regression tests to identify and resolve potential issues. Key activities included manual sanity tests on Intel, AMD and ARM hosts, evaluating network interface hotplug/unplug scenarios, and assessing the effectiveness of memory and disk hotplug operations.

The outcomes of these tests were instrumental in ensuring that Oracle Linux virtualization technologies met the required performance and reliability standards. By providing detailed insights into each testing procedure and its results, the report contributes to the ongoing improvement and stability of Oracle's virtualization solutions, supporting a seamless integration and robust performance within diverse computing environments.

Key words: Oracle Linux, Virtualization, QEMU, Libvirt, Oracle Cloud Infrastructure, Testing Procedures, Regression Testing, Performance Evaluation

---

Résumé

Ce rapport présente le résumé du projet de stage d'application entrepris au Centre de Recherche & Développement d'Oracle Maroc. Le stage avait pour objectif d'étudier et de valider les procédures de virtualisation et de test des systèmes Oracle Linux afin d'assurer la stabilité et les performances des environnements virtualisés. Au sein de l'équipe de virtualisation, de systèmes et de tests matériels d'Oracle Linux, j'ai participé à diverses tâches impliquant des tests de QEMU et Libvirt sur plusieurs environnements Oracle Linux.

Le projet se concentrait sur l'évaluation de la compatibilité et de la fonctionnalité des modules de virtualisation, y compris QEMU et Libvirt, à travers des tests rigoureux. Cela impliquait de valider leurs performances sous différentes configurations et versions de logiciels, ainsi que de mettre en œuvre des tests de régression pour identifier et résoudre d'éventuels problèmes. Les activités clés comprenaient des tests manuels de validation sur des hôtes Intel, AMD et ARM, l'évaluation des scénarios de hotplug/unplug des interfaces réseau, ainsi que l'évaluation de l'efficacité des opérations de hotplug de mémoire et de disque.

Les résultats de ces tests ont été essentiels pour garantir que les technologies de virtualisation d'Oracle Linux répondaient aux normes de performance et de fiabilité requises. En fournissant des informations détaillées sur chaque procédure de test et ses résultats, le rapport contribue à l'amélioration continue et à la stabilité des solutions de virtualisation d'Oracle, soutenant une intégration transparente et des performances robustes dans divers environnements informatiques.

Mots clés : Oracle Linux, Virtualisation, QEMU, Libvirt, Oracle Cloud Infrastructure, Procédures de test, Tests de régression, Évaluation des performances

---

ملخص

يقدم هذا التقرير ملخصًا شاملاً لمشروع التدريب الذي تم إجراؤه في مركز أبحاث وتطوير Oracle المغرب. كان الهدف من التدريب هو دراسة وتقييم إجراءات اختبار النظام والافتراضية في Oracle Linux لضمان استقرار وأداء البيئات الافتراضية. كجزء من فريق اختبار الأجهزة والنظام والافتراضية في Oracle Linux ، شاركت في مهام مختلفة تتعلق باختبار QEMU و Libvirt عبر بيئات Oracle Linux متعددة.

ركز المشروع على تقييم التوافق والوظائف لوحدات الافتراضية، بما في ذلك QEMU و Libvirt ، من خلال اختبارات دقيقة. شمل ذلك التحقق من أدائها تحت تكوينات وإصدارات برمجية مختلفة، بالإضافة إلى تنفيذ اختبارات التراجع لتحديد المشكلات المحتملة وحلها. تضمنت الأنشطة الرئيسية إجراء اختبارات يدوية على مضيفات Intel و AMD و ARM ، وتقييم سيناريوهات hotplug/unplug لواجهات الشبكة، وتقييم فعالية عمليات hotplug للذاكرة والتخزين.

كانت نتائج هذه الاختبارات حاسمة لضمان أن تقنيات الافتراضية في Oracle Linux تلبي معايير الأداء والموثوقية المطلوبة. من خلال تقديم رؤى مفصلة حول كل إجراء اختبار ونتائجه، يساهم التقرير في التحسين المستمر واستقرار حلول الافتراضية الخاصة بـ Oracle ، مما يدعم تكاملاً سلسًا وأداءً قويًا ضمن بيئات الحوسبة المتنوعة.

الكلمات المفتاحية: Oracle Linux ، الافتراضية، QEMU ، Libvirt ، Oracle Cloud Infrastructure ، إجراءات الاختبار، اختبارات التراجع، تقييم الأداء.

Abbreviations

AAVMF	ARM Architecture Virtual Machine Firmware
AMD	Advanced Micro Devices
AMD-V	AMD Virtualization
API	Application Programming Interface
ARM	Advanced RISC Machine
CLI	Command Line Interface
CPU	Central Processing Unit
EDK2	EFI Development Kit II
GUI	Graphical User Interface
I/O	Input/Output
ISO	International Organization for Standardization
KVM	Kernel-based Virtual Machine
NUMA	Non-Uniform Memory Access
NVMe	Non-Volatile Memory Express
OL	Oracle Linux
OCI	Oracle Cloud Infrastructure
OS	Operating System
OVMF	Open Virtual Machine Firmware
PCI	Peripheral Component Interconnect
QCOW2	QEMU Copy On Write version 2
QEMU	Quick EMUlator
QMP	QEMU Machine Protocol
RAM	Random Access Memory
RAW	Raw Disk Image
RHEL	Red Hat Enterprise Linux
RPM	Red Hat Package Manager
SSD	Solid State Drive
SSH	Secure SHell
UEK	Unbreakable Enterprise Kernel
UEK6U3	Unbreakable Enterprise Kernel 6 Update 3
UEK7U2	Unbreakable Enterprise Kernel 7 Update 2
UEFI	Unified Extensible Firmware Interface
VGA	Video Graphics Array
VFIO	Virtual Function I/O
VNC	Virtual Network Computing
VNIC	Virtual Network Interface Card
XML	eXtensible Markup Language
VT-x	Intel Virtualization Technology for x86

Contents

List of Figures

General Introduction

In the ever-evolving landscape of cloud computing and virtualization, the need for robust and reliable systems has never been more critical. As organizations increasingly rely on virtual environments to support their operations, ensuring the stability and compatibility of these systems becomes paramount. Oracle Corporation, a global leader in enterprise technology, has been at the forefront of this evolution, providing cutting-edge solutions that empower businesses to leverage virtualization technologies effectively.

This report presents a comprehensive overview of my internship project, which focused on validating and testing virtualization modules within the Oracle ecosystem. The project was designed to ensure that core virtualization components, such as QEMU and Libvirt, function seamlessly across various configurations and versions, thereby maintaining the reliability and performance of Oracle Linux and Oracle Cloud Infrastructure.

The report is structured into four main chapters:

- The First chapter outlines the project's context, including an introduction to Oracle Corporation, the team involved, and the objectives of the project;
- The Second chapter delves into the technical foundations of the project, providing a detailed analysis of the virtualization technologies and tools utilized;
- The Third chapter discusses the specific tasks and testing procedures undertaken during the internship, highlighting the methodologies used to ensure thorough validation;
- The Fourth chapter presents the implementation of the tests, detailing the results and the impact on the overall stability and performance of the virtualization environment.

This report aims to offer insights into the complexities of virtualization testing and the critical role it plays in the success of Oracle's enterprise solutions. Through meticulous testing and validation, the project contributed to the ongoing efforts to enhance the reliability and efficiency of Oracle's virtualization offerings, ensuring that they meet the high standards expected by their customers.

Chapter 1

General Context of the Project

This chapter, divided into three sections, provides an overview of the project's context. The first section introduces Oracle Corporation, the host organisation, providing insights into its history, field of activity, services, organizational structure, and culture. The second section explores the framework of the project, outlining the team in which it took place, the topic overview, to problematic it solves and the objectives to be achieved. The final section focuses on project management, including agile practices and collaboration tools, and the internship planning process.

1.1 Host Organization

My internship was hosted by Oracle Corporation, specifically Oracle Morocco Research & Development Center. This section introduces both entities, providing insights into their history and services.

1.1.1 Presentation

In today's world, businesses, technology companies, and the public sector rely on databases to store their data, while utilizing cloud services to manage their IT infrastructure and other essential tools such as ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), EPM (Enterprise Performance Management), and SCM (Supply Chain Management). These technologies have taken a crucial role, making it inconceivable to function without them.

Oracle, a leading US multinational computer technology corporation, not only employs these tools in its day-to-day operations but also creates and provides them to customers. Moreover, the company places a high value on innovation and ensuring customer satisfaction.

The first part of this section provides a broad context by giving an overview of Oracle Corporation and Oracle Morocco Application & Development Center (MADC). This includes an understanding of their business area of activity, their mission, as well as their organizational culture, and structure.

1.1.2 Oracle Corporation

Figure 1.1. Oracle Logo
[1]

1.1.2.1 History

Oracle Corporation is a Tech company with headquarters in Austin, Texas. In 1977, Larry Ellison, Bob Miner, and Ed Oates founded Oracle Corporation, then called as Software Development Laboratories (SDL). The company first focused on creating Oracle Database, a relational database management system (RDBMS), that became its flagship product. Enterprise software, cloud computing, hardware systems, and consulting services are now all part of Oracle's growing list of services and product options. Sun Microsystems was a key acquisition in 2010, giving Oracle access to technologies such as Java and the Solaris operating system.[1]

Today, Oracle is a leading provider of cloud-based IT infrastructure and software that helps organizations grow, find new sources of efficiency, and improve their performance. To

help organize and secure client data, Oracle developed the first and only autonomous database in the world as well as Oracle cloud.

1.1.2.2 Business area of activity

Oracle provides a variety of goods and services to its clients. Software development tools, cloud services, training, consultancy, and credentials are just a few of the goods and services offered here. One of the brands Oracle offers is Oracle Database, which has held the top spot since 1979 and is supported by cloud artificial intelligence. SaaS (Software as a Service), PaaS (Platform as a Service), IaaS (Infrastructure as a Service), and DaaS (Data as a Service) are just a few of the integrated solutions offered by Oracle Cloud Infrastructure for business, IT, and development needs.

Oracle Middleware is also a platform that is integrated for creating and running intelligent, agile applications while increasing technical efficiency using contemporary software designs. Oracle Services, which include Oracle Advanced Customer Support Services, Oracle Premium Support, Oracle Consulting, Oracle Financing, and Oracle Managed Cloud Services.

1.1.2.3 Oracle Labs

The research and development division of Oracle is called Oracle Labs, which focuses primarily on creating technologies that will keep Oracle at the forefront of the IT industry. Researchers at Oracle Labs pursue innovative methodologies and approaches, frequently taking on challenges or projects that would be difficult to complete within a product development organization or that involved high risk or uncertainty.

Figure 1.2. Oracle Labs Logo
[2]

Real-world applications are the focus of Oracle Labs research, and the researchers are working to create technologies that will eventually have a big impact on how society and technology evolve. For instance, work done in Oracle Labs led to the development of chip multithreading and the Java computer language.

Oracle's quest for research and development is greatly supported by Oracle Labs. Oracle can recognize challenges, find creative solutions, and create new products that can spur growth and innovation in the future because of its well-rounded research portfolio.

1.1.2.4 Oracle MADC

The first Oracle R&D center in Africa and one of only a few in the world, the Oracle MADC Morocco Research and Development Center, is a software development center established by Oracle Corporation in Casablanca, Morocco.

Figure 1.3. Oracle MADC Location

The center was inaugurated in 2015 and is one of Oracle's key development centers in the EMEA region (Europe, Middle East, and Africa). The center is staffed by highly skilled and talented software developers, engineers, and project managers who work on a variety of projects across various Oracle product lines utilizing the newest trends in innovation, such as artificial intelligence, augmented/virtual/mixed reality, big data analytics, Blockchain, cloud computing, cybersecurity, internet of things, machine learning, mobile computing, serverless computing. The center's researchers make use of all these technologies to tackle the most pressing problems facing business, science, and the public sector.

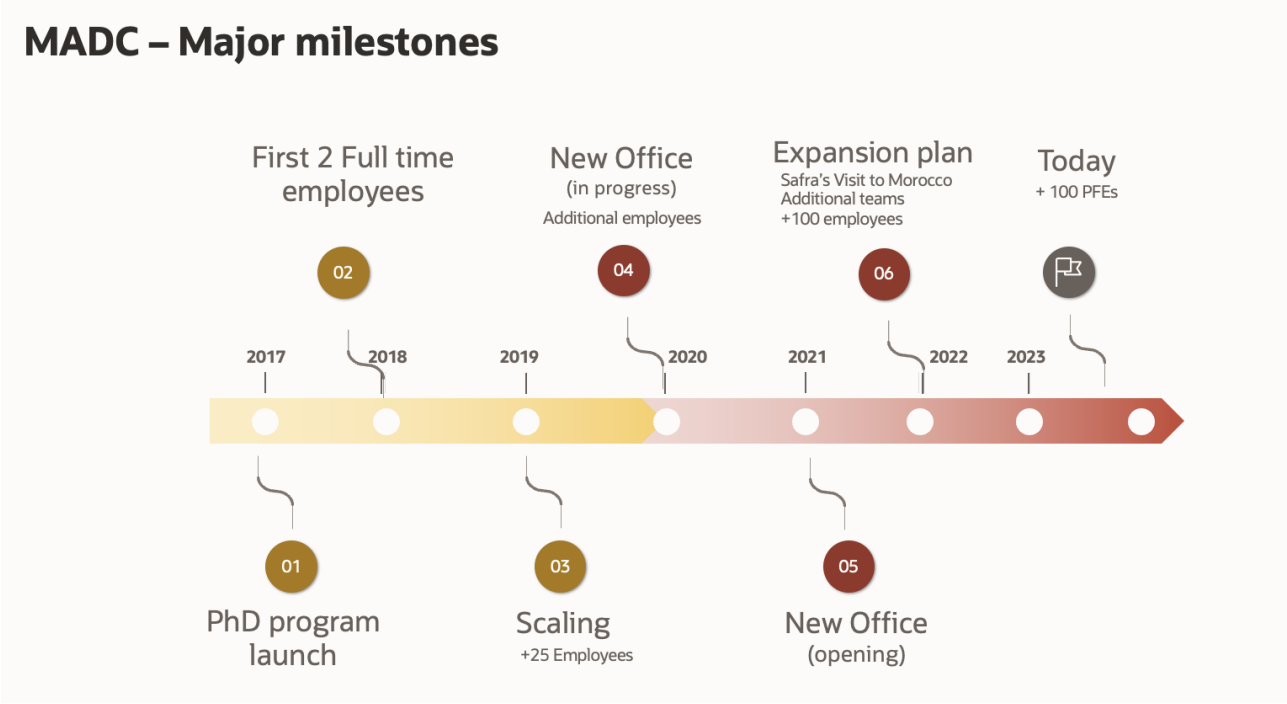


Figure 1.4. MADC Major Milestones

The global program includes expanding internship and graduate recruitment initiatives, as well as establishing joint research projects with regional universities. Oracle MADC has achieved success by delivering several high-quality products and solutions, such as Oracle Cloud Infrastructure (OCI) and Autonomous Data Warehouse (ADW).

1.1.3 Organizational Structure

Oracle Corporation has a hierarchical organizational structure with three main business divisions: Hardware, Services, Cloud and License. Each segment is divided into smaller business units focused on goods, services, or geographical areas. The Board of Directors oversees the company's performance and strategic direction, while the Executive Leadership Team manages daily operations.

Functional divisions, including sales, marketing, finance, human resources, legal, and IT, fall under the Executive Leadership Team. Business units are arranged by region, product, and service, with a senior VP or general manager overseeing each unit's operations. The employees are subordinated to both a functional manager and a business unit manager in the matrix-based organizational structure of the business units.

Figure 1.5. Executive Leadership Board

In general, Oracle's organizational structure is built to encourage teamwork, creativity, and client attention. While the matrix form enables people to work across many functions and business divisions to achieve shared goals, the hierarchical structure promotes effective decision-making and communication.

1.1.4 Organizational Culture

Oracle has a strong corporate culture that prioritizes innovation, client focus, and excellence. The company's history of innovation, emphasis on customer satisfaction, and dedication to fostering a friendly and collaborative work environment all have a bearing on the company's culture.

Oracle's culture places a strong emphasis on the following:

- **Innovation**
  Which is one of its defining characteristics. The business has a lengthy history of creating cutting-edge goods and technology that have revolutionized the tech sector. This culture of innovation is promoted by a dedication to research and development as well as by giving staff members the tools and encouragement they require to test out fresh concepts and create new products.

- **Customer Satisfaction**
  Oracle's customer-centric approach to business reflects its commitment to providing goods and services that satisfy the needs of its clients. Employees are encouraged to collaborate closely with clients to comprehend their demands and create solutions that meet their problems.

- **Quality**
  Oracle is dedicated to providing quality operations, products, and services, reflected in

its solid reputation for quality, dependability, and performance. The company expects every person to strive for excellence, providing them with the tools, resources, and encouragement to succeed.

- **Collaboration and Teamwork**
  Effective collaboration is essential for achieving Oracle's goals and delivering value to customers. The company encourages employees to work together across different departments and functions, providing them with tools and resources to facilitate collaboration and communication.

Overall, Oracle's organizational culture is characterized by a commitment to innovation, customer focus, excellence, collaboration, and teamwork. These values are reflected in the company's products, services, and operations, and are embraced by its employees at all levels of the organization.

1.2 Project Framework

1.2.1 Oracle Linux and Virtualization

Oracle Linux is a secure and high-performance operating environment used extensively by thousands of customers worldwide, both in the cloud and on-premises. It powers Oracle Cloud and Oracle Engineered Systems. As an operating system, Oracle Linux provides a solid foundation for virtualization technologies, including Oracle VM and other hypervisors like KVM and Xen. These technologies enable the creation and management of virtual machines (VMs) on Oracle Linux.

Oracle Cloud is Oracle's cloud computing platform offering a wide range of services, including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Within Oracle Cloud, customers can utilize virtualization capabilities to deploy and manage VMs in a cloud-based environment.

The Oracle Linux Virtualization, System, and Hardware Testing team is responsible for ensuring that all virtualization aspects are tested and function as intended before new Oracle Linux releases are submitted or delivered to Oracle Cloud Infrastructure. QA team works closely with the Development team for validating new features, tools etc, ensuring no regressions occur with previous systems and no bugs, issues, or performance problems arise in future releases. This process includes covering all scenarios and related test cases, and then developing a test plan that encompasses all aspects of the presented and new features.

1.2.2 Project Context

Oracle Linux, as a robust operating system, underpins various virtualization technologies, including Oracle VM, KVM, and Xen. These technologies facilitate the creation and management of virtual machines (VMs) on Oracle Linux. Oracle Cloud, Oracle's comprehensive cloud computing platform, offers services such as infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Within this cloud environment, Oracle Linux virtualization serves as a critical bridge, enabling the creation, configuration, and management of VMs. This integration allows businesses to leverage Oracle Linux's stability and flexibility to enhance their cloud computing capabilities.

1.2.3 Problematic

The primary challenge in this project lies in ensuring the compatibility of various virtualization modules (such as QEMU and Libvirt) with different software components. Virtualization modules are intricate systems that interact with multiple hardware and software layers, and updates or new releases can introduce bugs or compatibility issues. For instance, a new version of QEMU might inadvertently introduce defects or regressions, potentially impacting its stability, reliability, and compatibility with existing VM configurations. Ensuring that these components work harmoniously across different versions is crucial to maintaining the functionality and performance of the virtualization environment.

1.2.4 Objectives

To ensure the compatibility and stability of virtualization modules with various software components, this project focuses on validating the correct functioning of QEMU, Libvirt, and other related modules across different releases. These virtualization modules, integral to the Oracle ecosystem, require rigorous testing to maintain their reliability and performance.

The objectives are several-fold:

- Validate the functionality of QEMU, Libvirt, and other virtualization modules with both new and previous releases of software components such as UEK, Libiscsi, and OVMF;
- Implement regression testing to systematically identify and address potential defects or regressions introduced in new software versions;
- Ensure the stability, reliability, and compatibility of these virtualization modules across different versions and configurations;
- Uphold the core functionalities of the virtualization environment, including virtual machine creation, management, resource allocation, and networking;
- Mitigate risks associated with software updates by detecting and resolving issues early in the development cycle.

By achieving these goals, we aim to provide a dependable virtualization solution within the Oracle ecosystem, ensuring seamless integration and robust performance for businesses leveraging Oracle Linux virtualization and Oracle Cloud.

1.3 Project Management

1.3.1 Progress Monitoring

During my internship, we did not strictly adhere to an agile methodology. Instead, we implemented a weekly meeting schedule supplemented by urgent meetings for critical issues, such as troubleshooting problems or addressing pressing concerns. This approach ensured a consistent overview of project developments while allowing for the timely resolution of emergent needs. Additionally, I met one-on-one with my mentor weekly. These meetings allowed me to present the progress made, receive feedback, and discuss any open questions or issues. The

feedback provided valuable insights from different perspectives and played a crucial role in refining the project's functional requirements. Our sync-up meetings typically included status updates, demonstrations of functionality (if applicable), discussion of open questions or issues, and outlining next steps in the project.

1.3.2 Collaboration Tools

Using collaboration tools within a team offers several benefits. They play an important role in enhancing teamwork and communication within a team, enabling seamless communication through instant messaging and video conferencing. They also facilitate efficient collaboration on shared documents and projects, providing centralized repositories for knowledge sharing and documentation. Here is a list of collaboration tools used during my internship (*Figure 1.6*):

Figure 1.6. Communication Tools
Logos from Outlook[3], Slack[4], and Zoom[5].

- ***Slack***: A well-liked platform that offers file transfers, instant messaging, and robust message searching. It integrates with numerous other tools, including Zoom and Outlook, offering interesting features to facilitate communication;
- ***Zoom***: A cloud-based video conferencing tool that simplifies the hosting of virtual one-on-one or team meetings. This remote communication tool connects remote team members with one another, providing strong audio, video, and collaboration features;
- ***Outlook***: Serving as an email management platform, Outlook enables users to send and receive emails, manage calendars, store contact information, and track tasks effectively;
- ***Confluence***:

Figure 1.7. Confluence Logo
[6]

A collaborative wiki-style platform developed by Atlassian, serving as a central knowledge base and documentation tool. It allows us to document projects comprehensively, ensuring ease of understanding.

1.3.3 Internship Planning

My internship was conducted into two phases: the onboarding phase and the project phase.

1.3.3.1 Onboarding Phase

The first phase was dedicated to onboarding, ensuring a smooth integration into the company and providing the necessary training before starting the project. This phase included several key activities:

- **Account and Software Setup**
  The initial step involved setting up various accounts and software essential for daily operations, to ensure we had access to the necessary tools and platforms. This step included also familiarization with the company's processes and workflows, including understanding the organizational structure and communication tools.
- **Generic Training**
  The training provided an excellent opportunity to expand our skills and knowledge in various areas of software development. The generic training provided access to Oracle's learning resources and covered essential tools and platforms, including:
  - Communication tools (Slack);
  - Linux;
  - Oracle Cloud Infrastructure (OCI);
  - Developer tools (Git, Bitbucket, Jira);
  - DevOps;
  - Scripting;
  - APEX.
- **Specific Training**
  For my specific training, I focused on Oracle Linux and virtualization technologies, as these are essential for our team's system testing and validation efforts. This training involved gaining more experience in working with Oracle Linux and mastering virtualization tools like QEMU and Libvirt.

1.3.3.2 Project Phase

After the onboarding phase, the focus shifted to the project phase. The following Project Gantt Diagram (*Figure 1.8*) illustrates the timeline and sequence of the project phases.

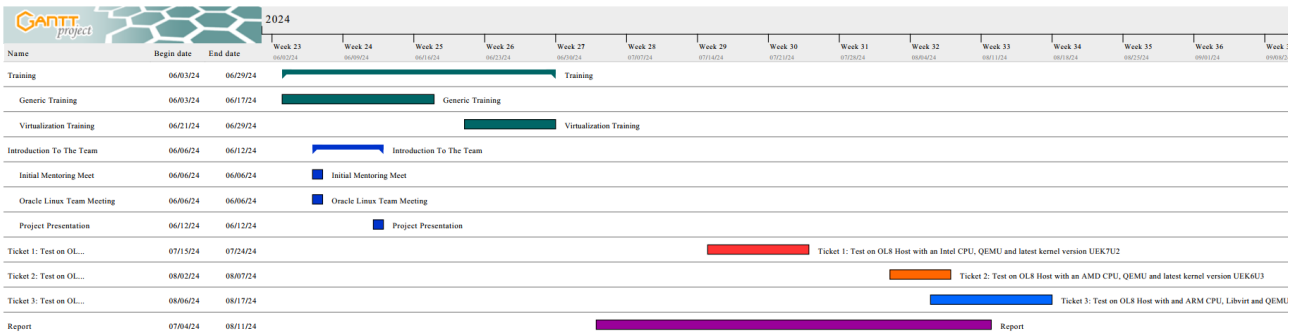


Figure 1.8. Gantt Diagram

1.4 Conclusion

This chapter provided the context of this project. We presented the host organization and discussed the general framework of the project, focusing on its context, problematic, and objectives. We also described the methodology employed during the internship and the project's planning.

Chapter 2

Technical Foundations and Project Essentials

This chapter will provide a comprehensive overview of the technical foundations essential to the project. It begins with an in-depth exploration of virtualization and Oracle Linux, detailing its core characteristics and key features. Following this, the chapter delves into the Unbreakable Enterprise Kernel (UEK) and its significance. The discussion then extends to the roles and functionalities of QEMU and Libvirt tools. The chapter concludes by outlining the project requirements, specifying the necessary features and criteria to achieve the project's objectives.

2.1 Technical Background

2.1.1 Virtualization

Virtualization serves as a cornerstone technology in modern IT infrastructure, enabling a single physical machine, or VM Host Server, to run multiple operating systems simultaneously as virtual machines (VM Guests). This approach abstracts the underlying hardware resources, creating isolated environments where each operating system can function independently, as though it were on dedicated hardware.

2.1.1.1 Mechanisms of Virtualization

The critical enabler of virtualization is the hypervisor, a software layer that directly interfaces with the hardware of the VM Host Server. The hypervisor manages the allocation of resources such as CPU, memory, and storage, distributing them among the various VM Guests. By presenting each guest with a virtualized hardware interface, the hypervisor ensures that multiple operating systems can coexist without interference. There are two primary types of hypervisors utilized in enterprise environments: Type 1 (bare-metal hypervisors) and Type 2 (hosted hypervisors).

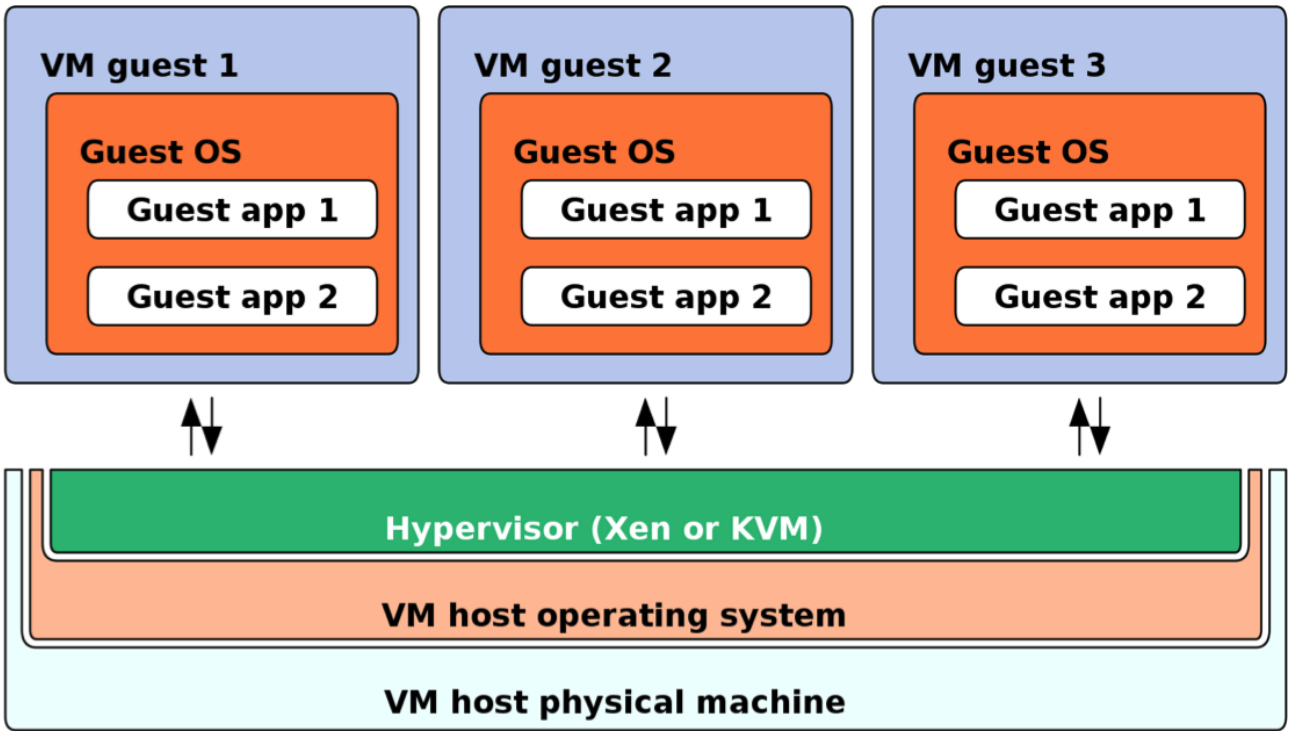


Figure 2.1. General Schema of Virtualization
[7]

2.1.1.2 Advantages of Virtualization in Cloud Infrastructure

In the context of Oracle Cloud Infrastructure (OCI), virtualization provides several strategic advantages:

- **Cost Optimization:** Virtualization reduces the need for additional hardware by allowing multiple operating systems to run on a single physical server, leading to significant savings in hardware, power, and cooling costs;
- **Enhanced Resource Utilization:** By maximizing the use of available resources, virtualization helps ensure that CPU, memory, and storage are used efficiently, reducing waste and improving overall system performance;
- **Operational Flexibility:** Virtualization facilitates rapid provisioning and deployment of virtual machines, enabling IT teams to quickly respond to changing business needs. Features such as live migration and snapshots further enhance the flexibility and resilience of virtualized environments;
- **Disaster Recovery and Business Continuity:** Virtualization supports advanced disaster recovery strategies by enabling features like VM snapshots, live migration, and automated failover, which are critical for maintaining business continuity in the event of hardware failures or other disruptions.

2.1.1.3 Virtualization Modes

- **Full Virtualization:** This mode emulates complete hardware, allowing unmodified operating systems to run in a virtualized environment. It is particularly useful when running legacy applications that require full emulation of underlying hardware;
- **Paravirtualization:** In this mode, the guest operating system is modified to interact directly with the hypervisor, leading to improved performance. Paravirtualization is advantageous in scenarios where high efficiency and performance are required;
- **Hardware-Assisted Virtualization (HVM):** This mode leverages hardware features such as Intel VT-x and AMD-V to enhance virtualization performance. HVM combines the compatibility of full virtualization with the efficiency of paravirtualization, making it ideal for high-performance computing environments.

2.1.2 Oracle Linux

Oracle Linux, often abbreviated as OL (previously known as Oracle Enterprise Linux or OEL), is a robust and versatile Linux distribution developed and freely distributed by Oracle Corporation. Since its initial release in 2006, Oracle Linux has been partially licensed under the GNU General Public License (GPL), making it an open-source platform. It is built using the source code from Red Hat Enterprise Linux (RHEL), with Oracle replacing Red Hat's branding with its own. This distribution plays a significant role in Oracle's ecosystem, serving as the foundation for Oracle Cloud Infrastructure (OCI) and Oracle Engineered Systems, including Oracle Exadata.

Figure 2.2. Oracle Linux Logo
[8]

2.1.2.1 Compatibility with Red Hat Enterprise Linux (RHEL)

Oracle Linux is designed to maintain binary compatibility with RHEL, ensuring that applications developed for RHEL can run seamlessly on Oracle Linux without modification. Oracle offers two kernel options within Oracle Linux:

- **Red Hat Compatible Kernel (RHCK):** This kernel is identical to the one provided by RHEL, offering users a stable and reliable environment that is fully compatible with the RHEL ecosystem;
- **Unbreakable Enterprise Kernel (UEK):** UEK is Oracle's optimized Linux kernel, tailored for high-performance environments. It enhances areas like OLTP, InfiniBand, and SSD access, and supports advanced features such as RDS, asynchronous I/O, and Btrfs. Since this kernel is used in our project, we will provide further details in the next sections.

Oracle's commitment to maintaining compatibility with RHEL allows enterprises to leverage the stability of RHEL while benefiting from Oracle's additional enhancements, particularly when using Oracle's hardware and cloud solutions.

2.1.2.2 Virtualization Support

Oracle Linux is fully equipped to support modern virtualization needs, offering the KVM hypervisor as part of its distribution. Additionally, it includes an oVirt-based management tool, which provides a robust interface for managing virtualized environments. Oracle Linux also supports other popular virtualization platforms, such as VMware and Oracle VM, which is based on the Xen hypervisor.

2.1.3 Unbreakable Enterprise Kernel (UEK)

The Unbreakable Enterprise Kernel (UEK) is a high-performance Linux kernel developed by Oracle, tailored specifically for enterprise-level workloads and Oracle environments. Designed to prioritize stability and efficiency, UEK closely follows the mainline Linux kernel while integrating Oracle's enhancements. It is extensively tested and is the recommended kernel

for Oracle's Engineered Systems, Oracle Cloud Infrastructure, and other large-scale enterprise deployments.

2.1.3.1 Capabilities and Features

UEK offers a range of advanced capabilities, making it particularly suitable for demanding enterprise applications. It incorporates a multitude of advanced features, such as:

- **Performance:** Engineered for Oracle workloads, UEK enhances the performance of databases and applications through optimized task scheduling, low-latency networking, and advanced memory management techniques;
- **Security:** It strengthens system protection with the latest security features and patches, including support for Security-Enhanced Linux (SELinux), control groups (cgroups), and other security-enhancing mechanisms;
- **Virtualization:** UEK supports modern virtualization technologies like KVM and Xen, offering paravirtualized drivers, optimized memory management, and advanced network and storage virtualization to maximize virtualization infrastructure efficiency;
- **Hardware Support:** Tailored for Oracle hardware, UEK includes support for cutting-edge technologies such as Intel Xeon Scalable processors and NVMe SSDs, as well as advanced features like NUMA and CPU hot-plugging to optimize hardware resource utilization;
- **Open Source:** As an open-source kernel, UEK is fully compatible with the Linux kernel and can be used across other Linux distributions, promoting transparency and flexibility in its deployment.

2.1.3.2 Key Versions: UEK7U2 and UEK6U3

Oracle's Unbreakable Enterprise Kernel continues to evolve with each release, bringing new features and optimizations. Two significant updates are used in this project : UEK7U2 and UEK6U3, each offering unique enhancements:

- **UEK7U2:** The second update of UEK7 introduces further improvements in performance and security, specifically designed for Oracle's cloud and on-premises environments. This update enhances support for the latest hardware, including next-generation processors and storage technologies. UEK7U2 also introduces refined scheduling algorithms and networking optimizations, making it an ideal choice for workloads that demand low latency and high throughput;
- **UEK6U3:** Update 3 in the UEK6 series builds on previous stability and performance enhancements. It focuses on extended hardware compatibility, especially for older Oracle systems, and includes critical security patches. This update is optimized for environments where reliability and backward compatibility are crucial, ensuring legacy applications run smoothly without sacrificing performance.

These updates underline Oracle's commitment to providing a flexible and high-performing kernel tailored to the evolving needs of enterprise environments.

2.1.4 QEMU

QEMU (Quick Emulator) is an open-source machine emulator and virtualizer that provides a comprehensive platform for hardware virtualization. As a crucial component in many virtualization environments, QEMU facilitates the creation and management of virtual machines by emulating hardware components and providing a range of virtual hardware devices.

2.1.4.1 Modes of Operation and Integration

QEMU can operate in two primary modes: full system emulation and user-mode emulation. In full system emulation mode, QEMU emulates an entire computer system, including CPU, memory, and peripheral devices, allowing users to run an entire operating system as if it were on physical hardware. In user-mode emulation, QEMU enables running applications compiled for different processor architectures on a host system, effectively providing cross-platform compatibility.

QEMU integrates seamlessly with various hypervisors and virtualization management tools, such as KVM (Kernel-based Virtual Machine) and Libvirt. When used in conjunction with KVM, QEMU leverages hardware acceleration features to enhance performance, delivering near-native execution speeds for virtualized environments.

2.1.4.2 Capabilities and Features

QEMU provides a diverse array of features that contribute to its versatility and effectiveness in virtualization:

- **Hardware Emulation:** Supports a wide range of hardware devices and peripherals, including CPUs, memory, storage controllers, and network interfaces, allowing users to create virtual machines with diverse configurations;
- **Virtualization Acceleration:** When paired with KVM, QEMU utilizes hardware acceleration technologies like Intel VT-x and AMD-V to enhance the performance of virtual machines, providing near-native execution speeds;
- **Live Migration:** Facilitates the movement of running virtual machines between physical hosts with minimal downtime, supporting continuous availability and load balancing;
- **Snapshot and Checkpoints:** Enables the creation of snapshots and checkpoints of virtual machines, allowing users to save and restore the state of a VM at any point in time, which is valuable for testing and disaster recovery;
- **Cross-Platform Compatibility:** Offers support for multiple processor architectures and operating systems, making it a flexible tool for diverse virtualization scenarios;
- **Customizable Devices:** Provides the ability to define and configure various virtual devices, including graphics cards, network adapters, and storage controllers, to tailor virtual machines to specific needs.

2.1.5 Libvirt

Libvirt is a comprehensive suite of software that provides a unified API (Application Programming Interface) for managing a variety of virtualization solutions, including KVM, QEMU, Xen, Virtuozzo, VMware ESX, and others. This suite comprises an API library, the system service libvirtd, and a command-line utility known as virsh. The modular design of libvirt facilitates extending its functionality, making it a versatile tool for managing virtual environments.

2.1.5.1 Libvirt Architecture

The libvirtd service operates on the virtual machine (VM) host, handling interactions between the host's configuration and the virtual environments it supports. Client libraries and utilities connect to libvirtd to gather configuration data and resource information for the host servers. Each virtual machine's configuration is managed through XML files, which can be manipulated using various methods. Libvirt supports both command-line and graphical user interface (GUI) tools for managing virtual machines. For command-line management, virsh provides a robust tool for configuring and controlling VMs, while virt-manager offers a graphical interface for those who prefer visual management. Additionally, other utilities like virt-install, virt-clone, and virt-image assist in creating, cloning, and managing virtual machines and their disks.

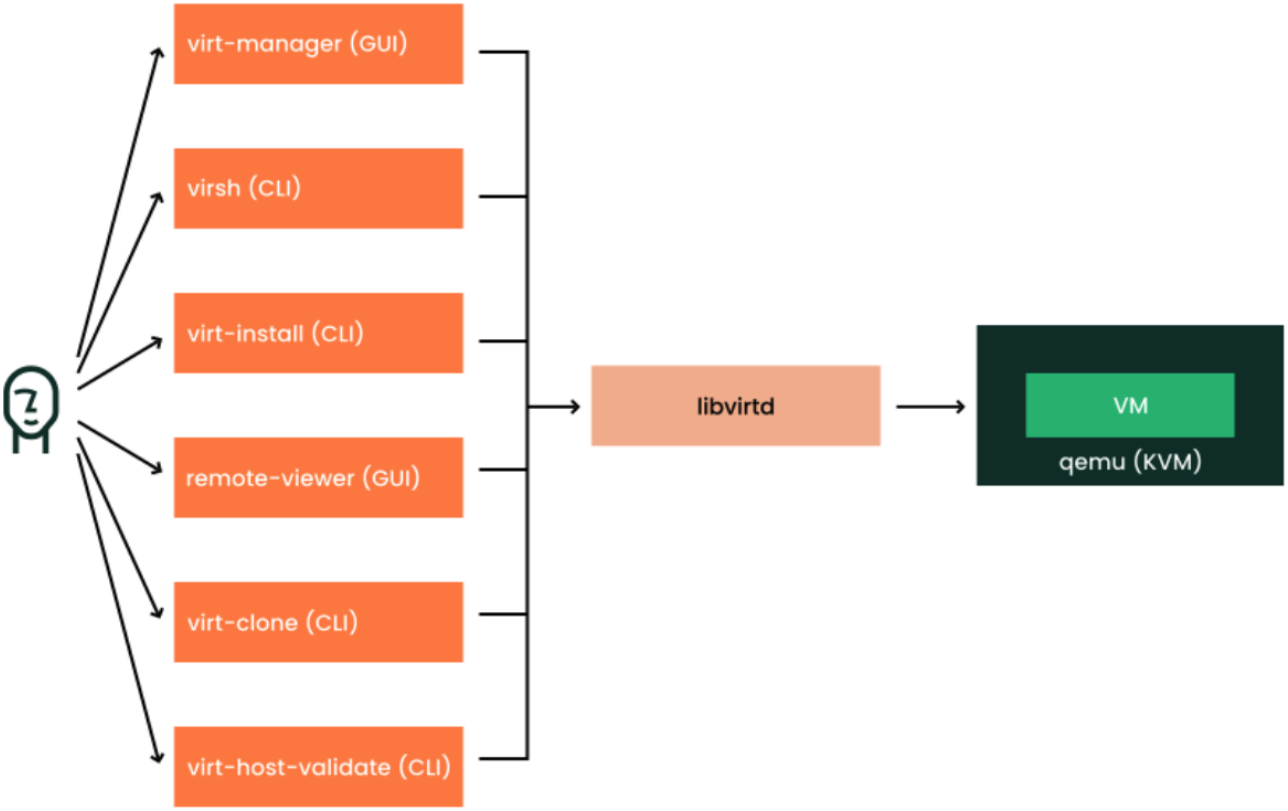


Figure 2.3. Libvirt Overview
[9]

2.1.5.2 Benefits of Using Libvirt

Libvirt provides several advantages that enhance the management and operation of virtual environments:

- **Basic Monitoring:** Allows for monitoring of both host systems and virtual machines, providing essential insights into their status and performance;
- **Automation and Scripting:** Facilitates automation of complex virtualization tasks and workflows through scripting, streamlining management processes;
- **Remote Management:** Enables remote management of virtualization hosts over secure protocols such as SSH, which is crucial for managing resources on remote servers;
- **Headless Server Management:** Offers management capabilities for headless servers that lack a graphical interface, using command-line tools as the primary method of control;
- **Custom Reporting:** Supports scripting and commands for generating custom reports, providing flexibility in monitoring and documentation;
- **Multi-Hypervisor Support:** Provides a unified tool for managing multiple hypervisors, ensuring compatibility and efficiency across different virtualization technologies.

By integrating libvirt into virtualization strategies, organizations can achieve greater control, efficiency, and flexibility in managing their virtual environments.

2.2 Project Requirements

Defining the project requirements is crucial to outline the expected outcomes and objectives clearly. The following are the main requirements to be addressed:

- **Validation of Functionality:** Ensure proper operation of QEMU and Libvirt across various Oracle Linux versions, including components like UEK, Libiscsi, and OVMF;
- **Regression Testing:** Detect and address any regressions or issues introduced by new software updates;
- **Compatibility Testing:** Verify interoperability of different QEMU and Libvirt versions with diverse host and guest OS configurations;
- **Performance Assessment:** Monitor and document the performance of virtualization modules to ensure they meet established standards;
- **Documentation:** Provide comprehensive reports on the testing procedures, results, and any issues encountered or resolved.

2.3 Conclusion

This chapter has delivered a thorough analysis of the technical aspects critical to the project. It started with an overview of virtualization and Oracle Linux, followed by a detailed look at the Unbreakable Enterprise Kernel and its features. The discussion then covered QEMU's modes of operation and integration, and examined the architecture and benefits of Libvirt. The chapter concluded by defining the project requirements necessary for meeting our goals.

Chapter 3

Internship Tasks and Testing Procedures

This chapter presents a detailed examination of the internship tasks and testing procedures undertaken throughout the project. It begins by outlining the primary tickets that guided my activities, which include manual QEMU and Libvirt tests across various Oracle Linux environments. Following this, the chapter categorizes the tasks into QEMU-focused and Libvirt-focused tests, emphasizing their respective objectives and methodologies. The subsequent sections will explore the steps taken to analyze ticket information, establish the testing environment, and execute the associated procedures, ensuring a thorough understanding of the project's technical foundations.

3.1 Analyzing Ticket Information

3.1.1 QEMU Tests

- **Ticket 1: Test on OL8 Host with an Intel CPU, QEMU and latest kernel version UEK7U2**
  This ticket involves performing a manual sanity test on Oracle Linux 8 with Intel CPU, using latest version of UEK7U2 and QEMU. Guest: Oracle Linux 8.9.
- **Ticket 2: Test on OL8 Host with an AMD CPU, QEMU and latest kernel version UEK6U3**
  Conducted a similar test on an AMD CPU with UEK7U2. Guest OS: Oracle Linux 7.9.

3.1.2 Libvirt Tests

Ticket 3: Test on OL8 Host with and ARM CPU, Libvirt, QEMU and Latest kernel version UEK7U2
Set up and tested on ARM architecture with Oracle Linux 8, Libvirt, and QEMU. Guest OS: Oracle Linux 8.10.

These summaries highlight the core differences between QEMU and Libvirt tests. While both require environment setup and VM management, Libvirt tests involve additional steps for integration with QEMU and offer more control over VM management through Libvirt-specific commands.

3.2 Environment Setup

3.2.1 Creation of the Instance

The first step in each test was creating the instance that would act as the host. This was done using Oracle Cloud Infrastructure (OCI), ensuring that the instance was running Oracle Linux 8 and equipped with the appropriate CPU architecture (AMD, ARM, or Intel). The instance needed to support virtualization and have adequate resources for testing.

3.2.2 Kernel Setup on Host

The next step involved installing the specified version of the Unbreakable Enterprise Kernel (UEK) on the host system. Depending on the requirements outlined in each ticket, either UEK7U2 or UEK6U3 was installed. Given that the tests were conducted in a controlled environment, the latest kernel versions available in the repositories were utilized to ensure up-to-date performance and security.

The process began by downloading the necessary RPM files directly from the repository using the `wget` command, as provided by the team. This ensured that the exact kernel versions required for the test were obtained.

```
[root@              kernel]# ls -l
total 1889016
-rw-r--r--. 1 root root   3512560 Aug  8 05:55 bpftool-                    .el8uek.aarch64.rpm
-rw-r--r--. 1 root root   3407384 Aug  8 05:55 bpftool-debuginfo-                    .el8uek.aarch64.rpm
-rw-r--r--. 1 root root   2573212 Aug  8 05:55 kernel-uek-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  29805044 Aug  8 05:55 kernel-uek-container-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root   3712364 Aug  8 05:55 kernel-uek-container-debug-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  59493268 Aug  8 05:55 kernel-uek-core-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root   2573284 Aug  8 05:55 kernel-uek-debug-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  58740004 Aug  8 05:55 kernel-uek-debug-core-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root 733950324 Aug  8 05:55 kernel-uek-debug-debuginfo-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  21302804 Aug  8 05:55 kernel-uek-debug-devel-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  71590160 Aug  8 05:55 kernel-uek-debug-modules-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root   5624036 Aug  8 05:55 kernel-uek-debug-modules-extra-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root 733770008 Aug  8 05:55 kernel-uek-debuginfo-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  72793780 Aug  8 05:55 kernel-uek-debuginfo-common-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  21275036 Aug  8 05:55 kernel-uek-devel-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  24788320 Aug  8 04:45 kernel-uek-doc-                  .el8uek.noarch.rpm
-rw-r--r--. 1 root root   3869196 Aug  8 05:55 kernel-uek-headers-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root  75770528 Aug  8 05:55 kernel-uek-modules-                  .el8uek.aarch64.rpm
-rw-r--r--. 1 root root   5761960 Aug  8 05:55 kernel-uek-modules-extra-                  .el8uek.aarch64.rpm
```

Figure 3.1. Kernel RPM Files Downloaded

Once the RPM files were downloaded, the kernel was installed on the host system, as illustrated below:

```
Dependencies resolved.
================================================================================
 Package                 Architecture  Version              Repository     Size
================================================================================
Installing:
 kernel-uek              aarch64       .el8uek              @commandline  2.5 M
 kernel-uek-core         aarch64       .el8uek              @commandline   57 M
 kernel-uek-debug        aarch64       .el8uek              @commandline  2.5 M
 kernel-uek-debug-core   aarch64       .el8uek              @commandline   56 M
 kernel-uek-debug-debuginfo aarch64    .el8uek              @commandline  700 M
 kernel-uek-debug-devel  aarch64       .el8uek              @commandline   20 M
 kernel-uek-debug-modules aarch64      .el8uek              @commandline   68 M
 kernel-uek-debug-modules-extra aarch64 .el8uek             @commandline  5.4 M
 kernel-uek-debuginfo    aarch64       .el8uek              @commandline  700 M
 kernel-uek-debuginfo-common aarch64   .el8uek              @commandline   69 M
 kernel-uek-devel        aarch64       .el8uek              @commandline   20 M
 kernel-uek-modules      aarch64       .el8uek              @commandline   72 M
 kernel-uek-modules-extra aarch64      .el8uek              @commandline  5.5 M
Upgrading:
 bpftool                 aarch64       .el8uek              @commandline  3.3 M
 bpftool-debuginfo       aarch64       .el8uek              @commandline  3.2 M
 kernel-uek-container    aarch64       .el8uek              @commandline   28 M
 kernel-uek-container-debug aarch64    .el8uek              @commandline  3.5 M
 kernel-uek-doc          noarch        .el8uek              @commandline   24 M
 kernel-uek-headers      aarch64       .el8uek              @commandline  3.7 M
Removing:
 kernel-uek-core         aarch64       .el8uek              @ol8_UEK2     204 M
 kernel-uek-modules      aarch64       .el8uek              @ol8_UEK2      60 M
Removing dependent packages:
 kernel-uek-modules-extra aarch64      .el8uek              @ol8_UEK2     2.8 M

Transaction Summary
================================================================================
Install  13 Packages
Upgrade   6 Packages
Remove    3 Packages

Total size: 1.8 G
Is this ok [y/N]: y
```

Figure 3.2. Kernel Installation Process

After installation, the host system was rebooted to apply the new kernel. Post-reboot, the correct kernel version was verified using commands such as `grubby` or `uname` to confirm that the system was running the desired kernel.

```
[root@              ]# grubby --default-kernel
/boot/vmlinuz-                .el8uek.aarch64
[root@              ]#
```

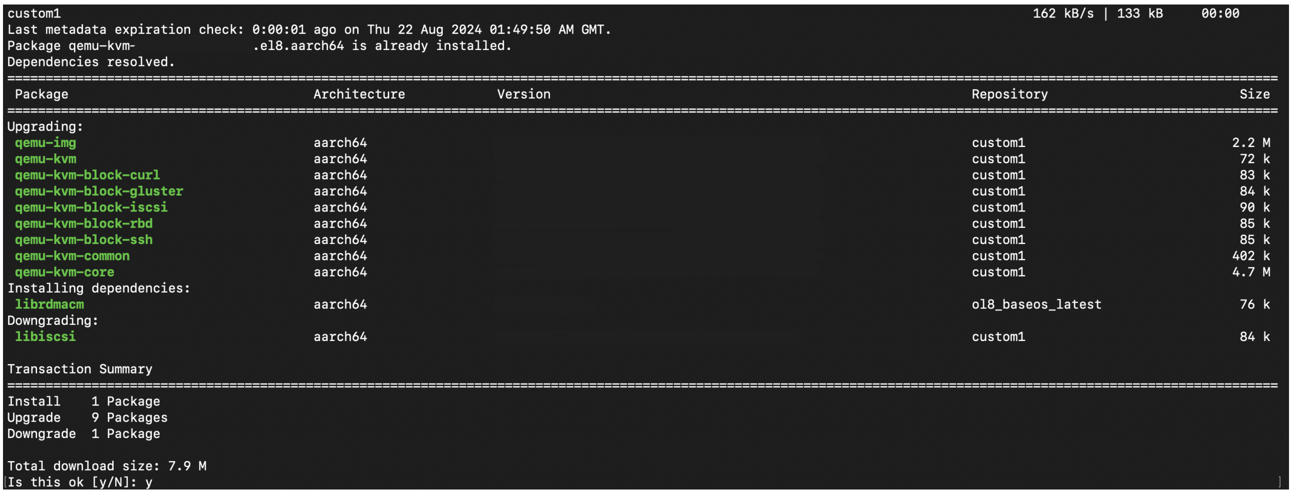
Figure 3.3. Verifying the Kernel Version

3.2.3 Installation of QEMU and Libvirt

The subsequent step involved the installation of the required versions of QEMU and Libvirt. For tickets focused exclusively on QEMU, only QEMU was installed. However, for tests

involving Libvirt, both QEMU and Libvirt needed to be installed to meet the test requirements.

To begin with, the repository link for QEMU was assigned to a variable named `Link`, after which the installation was executed using the `yum install` command, as shown below:



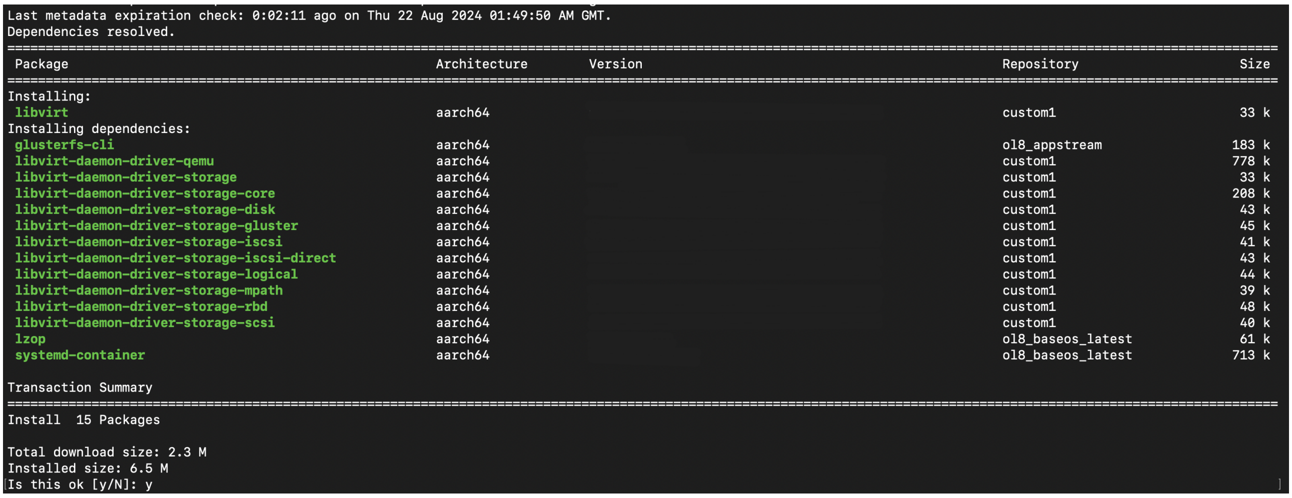
```
custom1                                                                               162 kB/s | 133 kB     00:00
Last metadata expiration check: 0:00:01 ago on Thu 22 Aug 2024 01:49:50 AM GMT.
Package qemu-kvm-[redacted].el8.aarch64 is already installed.
Dependencies resolved.
===================================================================================================================
 Package                  Architecture   Version      Repository            Size
===================================================================================================================
Upgrading:
 qemu-img                 aarch64        [redacted]   custom1               2.2 M
 qemu-kvm                 aarch64        [redacted]   custom1               72 k
 qemu-kvm-block-curl      aarch64        [redacted]   custom1               83 k
 qemu-kvm-block-gluster   aarch64        [redacted]   custom1               84 k
 qemu-kvm-block-iscsi     aarch64        [redacted]   custom1               90 k
 qemu-kvm-block-rbd       aarch64        [redacted]   custom1               85 k
 qemu-kvm-block-ssh       aarch64        [redacted]   custom1               85 k
 qemu-kvm-common          aarch64        [redacted]   custom1               402 k
 qemu-kvm-core            aarch64        [redacted]   custom1               4.7 M
Installing dependencies:
 librdmacm                aarch64        [redacted]   ol8_baseos_latest     76 k
Downgrading:
 libiscsi                 aarch64        [redacted]   custom1               84 k

Transaction Summary
===================================================================================================================
Install    1 Package
Upgrade    9 Packages
Downgrade  1 Package

Total download size: 7.9 M
Is this ok [y/N]: y
```

Figure 3.4. QEMU Installation Process

A similar process was followed for installing Libvirt:



```
Last metadata expiration check: 0:02:11 ago on Thu 22 Aug 2024 01:49:50 AM GMT.
Dependencies resolved.
===================================================================================================================
 Package                                   Architecture   Version      Repository            Size
===================================================================================================================
Installing:
 libvirt                                   aarch64        [redacted]   custom1               33 k
Installing dependencies:
 glusterfs-cli                             aarch64        [redacted]   ol8_appstream         183 k
 libvirt-daemon-driver-qemu                aarch64        [redacted]   custom1               778 k
 libvirt-daemon-driver-storage             aarch64        [redacted]   custom1               33 k
 libvirt-daemon-driver-storage-core        aarch64        [redacted]   custom1               208 k
 libvirt-daemon-driver-storage-disk        aarch64        [redacted]   custom1               43 k
 libvirt-daemon-driver-storage-gluster     aarch64        [redacted]   custom1               45 k
 libvirt-daemon-driver-storage-iscsi       aarch64        [redacted]   custom1               41 k
 libvirt-daemon-driver-storage-iscsi-direct aarch64       [redacted]   custom1               43 k
 libvirt-daemon-driver-storage-logical     aarch64        [redacted]   custom1               44 k
 libvirt-daemon-driver-storage-mpath       aarch64        [redacted]   custom1               39 k
 libvirt-daemon-driver-storage-rbd         aarch64        [redacted]   custom1               48 k
 libvirt-daemon-driver-storage-scsi        aarch64        [redacted]   custom1               40 k
 lzop                                      aarch64        [redacted]   ol8_baseos_latest     61 k
 systemd-container                         aarch64        [redacted]   ol8_baseos_latest     713 k

Transaction Summary
===================================================================================================================
Install  15 Packages

Total download size: 2.3 M
Installed size: 6.5 M
Is this ok [y/N]: y
```

Figure 3.5. Libvirt Installation Process

After both installations were completed, the versions of QEMU and Libvirt on the host were verified to ensure the correct versions were in place:

```
[root@                ]# yum info qemu-kvm
Last metadata expiration check: 3:43:23 ago on Wed 21 Aug 2024 10:11:04 PM GMT.
Installed Packages
Name         : qemu-kvm
Epoch        :
Version      :
Release      :
Architecture : aarch64
Size         : 0.0
Source       : qemu-kvm-                                    .src.rpm
Repository   : @System
From repo    : custom1
Summary      : QEMU is a machine emulator and virtualizer
URL          : http://www.qemu.org/
License      : GPLv2+ and LGPLv2+ and BSD
Description  : qemu-kvm is an open source virtualizer that provides hardware
             : emulation for the KVM hypervisor. qemu-kvm acts as a virtual
             : machine monitor together with the KVM kernel modules, and emulates the
             : hardware for a full system such as a PC and its associated peripherals.

[root@                ]# yum info libvirt
Last metadata expiration check: 3:43:31 ago on Wed 21 Aug 2024 10:11:04 PM GMT.
Installed Packages
Name         : libvirt
Version      :
Release      :
Architecture : aarch64
Size         : 0.0
Source       : libvirt-                                   .src.rpm
Repository   : @System
From repo    : custom1
Summary      : Library providing a simple virtualization API
URL          : https://libvirt.org/
License      : LGPLv2+
Description  : Libvirt is a C toolkit to interact with the virtualization capabilities
             : of recent versions of Linux (and other OSes). The main package includes
             : the libvirtd server exporting the virtualization support.
```

Figure 3.6. Verification of QEMU and Libvirt Versions

3.2.4 OVMF/AAVMF

To ensure proper virtualization support across different platforms, I verified the presence of essential files for OVMF (used for AMD and Intel hosts) and AAVMF (used for ARM hosts) on the system.

Initially, it was observed that the directory `/usr/share/AAVMF/` contained fewer than the expected number of files, with only six present:

```
[root@                ]# ll /usr/share/AAVMF/
total 393216
-rw-r--r--. 1 root root 67108864 Feb 18  2021 AAVMF_CODE.fd
-rw-r--r--. 1 root root 67108864 Feb 18  2021 AAVMF_CODE.pure-efi-debug.fd
-rw-r--r--. 1 root root 67108864 Feb 18  2021 AAVMF_CODE.pure-efi.fd
-rw-r--r--. 1 root root 67108864 Feb 18  2021 AAVMF_VARS.fd
-rw-r--r--. 1 root root 67108864 Feb 18  2021 AAVMF_VARS.pure-efi-debug.fd
-rw-r--r--. 1 root root 67108864 Feb 18  2021 AAVMF_VARS.pure-efi.fd
```

Figure 3.7. Incomplete AAVMF Files

To address this issue, a new yum repository was created to download the complete set of necessary files. The repository was configured as follows:

```
[[root@                    ]# cat /etc/yum.repos.d/edk2.repo
[edk2]
name=edk2
baseurl=http://...
gpgcheck=0
enabled=1
module_hotfixes=true
skip_if_unavailable=true
```

Figure 3.8. edk2 Yum Repository Configuration

The new files were then installed using the following command:

```
[root@               ]# yum install edk2-aarch64 edk2-tools --enablerepo=*
Last metadata expiration check: 0:10:37 ago on Thu 22 Aug 2024 02:10:56 AM GMT.
Dependencies resolved.
================================================================================
 Package                 Architecture        Version                 Repository      Size
================================================================================
Installing:
 edk2-aarch64            noarch                                      edk2            14 M
 edk2-tools              aarch64                                     edk2            86 k

Transaction Summary
================================================================================
Install  2 Packages

Total size: 14 M
Total download size: 86 k
Installed size: 915 M
Is this ok [y/N]: y
```

Figure 3.9. Installation of edk2 Files

Finally, the directory `/usr/share/AAVMF/` was checked again to confirm that all necessary files were now correctly installed:

```
[root@               ]# ls /usr/share/AAVMF/
AAVMF_CODE.fd                           AAVMF_CODE_2M.pure-efi-notpm.fd         AAVMF_VARS.secboot-debug.fd
AAVMF_CODE.fd.hmac                      AAVMF_CODE_2M.pure-efi-notpm.fd.hmac    AAVMF_VARS.secboot-debug.fd.hmac
AAVMF_CODE.pure-efi-debug.fd            AAVMF_CODE_2M.pure-efi.fd               AAVMF_VARS.secboot.fd
AAVMF_CODE.pure-efi-debug.fd.hmac       AAVMF_CODE_2M.pure-efi.fd.hmac          AAVMF_VARS.secboot.fd.hmac
AAVMF_CODE.pure-efi-notpm-debug.fd      AAVMF_CODE_2M.secboot-debug.fd          AAVMF_VARS_1M.pure-efi-debug.fd
AAVMF_CODE.pure-efi-notpm-debug.fd.hmac AAVMF_CODE_2M.secboot-debug.fd.hmac     AAVMF_VARS_1M.pure-efi-debug.fd.hmac
AAVMF_CODE.pure-efi-notpm.fd            AAVMF_CODE_2M.secboot.fd                AAVMF_VARS_1M.pure-efi-notpm-debug.fd
AAVMF_CODE.pure-efi-notpm.fd.hmac       AAVMF_CODE_2M.secboot.fd.hmac           AAVMF_VARS_1M.pure-efi-notpm-debug.fd.hmac
AAVMF_CODE.pure-efi.fd                  AAVMF_VARS.fd                           AAVMF_VARS_1M.pure-efi-notpm.fd
AAVMF_CODE.pure-efi.fd.hmac             AAVMF_VARS.fd.hmac                      AAVMF_VARS_1M.pure-efi-notpm.fd.hmac
AAVMF_CODE.secboot-debug.fd             AAVMF_VARS.pure-efi-debug.fd            AAVMF_VARS_1M.pure-efi.fd
AAVMF_CODE.secboot-debug.fd.hmac        AAVMF_VARS.pure-efi-debug.fd.hmac       AAVMF_VARS_1M.pure-efi.fd.hmac
AAVMF_CODE.secboot.fd                   AAVMF_VARS.pure-efi-notpm-debug.fd      AAVMF_VARS_1M.secboot-debug.fd
AAVMF_CODE.secboot.fd.hmac              AAVMF_VARS.pure-efi-notpm-debug.fd.hmac AAVMF_VARS_1M.secboot-debug.fd.hmac
AAVMF_CODE_2M.pure-efi-debug.fd         AAVMF_VARS.pure-efi-notpm.fd            AAVMF_VARS_1M.secboot.fd
AAVMF_CODE_2M.pure-efi-debug.fd.hmac    AAVMF_VARS.pure-efi-notpm.fd.hmac       AAVMF_VARS_1M.secboot.fd.hmac
AAVMF_CODE_2M.pure-efi-notpm-debug.fd   AAVMF_VARS.pure-efi.fd
AAVMF_CODE_2M.pure-efi-notpm-debug.fd.hmac AAVMF_VARS.pure-efi.fd.hmac
```

Figure 3.10. Verified AAVMF Files

3.2.5 ISO Installation and Image Creation

With the environment prepared, the next step involved downloading and installing the ISO files necessary for the guest OSs. These included ISOs for Oracle Linux 7.9, 8.9 and 8.10.

Creating the disk image was a critical process in both QEMU and Libvirt tests. A disk image is essentially a file that contains the complete contents and structure of a storage device, such as a hard drive. This image is used by virtual machines to simulate the presence of an actual physical disk, allowing the guest OS to function as if it were running on its own dedicated hardware.

For QEMU, the image was created using the following command:

```
[root@              images]# qemu-img create -f qcow2 disk-ol810.qcow2 20G
Formatting 'disk-ol810.qcow2', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib size=21474836480 lazy_refcounts=off refcount_bits=16
[root@              images]#
```

Figure 3.11. Creating Disk Image Using QEMU

In this case, the image format chosen was QCOW2 instead of RAW. The QCOW2 format offers several advantages, including the ability to take snapshots and compress the image to save space, which is especially useful in test environments where flexibility and storage efficiency are important. The RAW format, on the other hand, provides better performance but at the cost of increased storage space and fewer features, making QCOW2 the preferred choice for this project.

When using Libvirt, the process involves some additional steps. First, the Libvirt daemon must be started to manage the virtualization services:

```
[root@              images]# systemctl status libvirtd
● libvirtd.service - Virtualization daemon
   Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Thu 2024-08-22 02:35:45 GMT; 5s ago
     Docs: man:libvirtd(8)
           https://libvirt.org
  Process: 88345 ExecStart=/usr/sbin/libvirtd $LIBVIRTD_ARGS (code=exited, status=0/SUCCESS)
 Main PID: 88345 (code=exited, status=0/SUCCESS)
    Tasks: 2 (limit: 32768)
   Memory: 118.1M
   CGroup: /system.slice/libvirtd.service
           ├─6945 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/libexec/libvirt_leaseshelper
           └─6946 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/libexec/libvirt_leaseshelper

Aug 22 01:32:08                dnsmasq-dhcp[6945]: DHCPREQUEST(virbr0)
Aug 22 01:32:08                dnsmasq-dhcp[6945]: DHCPACK(virbr0)
Aug 22 01:52:14                dnsmasq[6945]: listening on virbr0(#4):
Aug 22 01:56:04                dnsmasq-dhcp[6945]: DHCPREQUEST(virbr0)
Aug 22 01:56:04                dnsmasq-dhcp[6945]: DHCPACK(virbr0)
Aug 22 02:21:37                dnsmasq-dhcp[6945]: DHCPREQUEST(virbr0)
Aug 22 02:21:37                dnsmasq-dhcp[6945]: DHCPACK(virbr0)
Aug 22 02:35:45                systemd[1]: Stopping Virtualization daemon...
Aug 22 02:35:45                systemd[1]: libvirtd.service: Succeeded.
Aug 22 02:35:45                systemd[1]: Stopped Virtualization daemon.
[root@              images]# systemctl start libvirtd
```

Figure 3.12. Starting Libvirt Daemon

Next, a storage pool needs to be created. A storage pool is a collection of storage volumes managed by Libvirt, where virtual machines can store their disk images. It acts as an abstraction layer, allowing easy management of storage resources for virtual environments. After creating the storage pool, it is started to make it available for use:

```
[root@              ilyass]# virsh pool-define-as pool_dir- dir --target /dev/shm/ilyass/image/
setlocale: No such file or directory
Pool pool_dir- defined

[root@              ilyass]# virsh pool-start pool_dir-
setlocale: No such file or directory
Pool pool_dir- started
```

Figure 3.13. Creating and Starting the Storage Pool

Finally, the required disk image is created within this storage pool:

```
[root@              ilyass]# virsh vol-create-as pool_dir- disk-ol89.qcow2 20G --format qcow2
setlocale: No such file or directory
Vol disk-ol89.qcow2 created
```

Figure 3.14. Creating Disk Image Using Libvirt

3.2.6 Creating the VM Using QEMU

For QEMU-based tests, the virtual machine (VM) was created using a detailed command that specified the necessary parameters, including CPU type, memory allocation, and the disk image. This command enabled the configuration and launch of the VM, ensuring it was tailored to meet the test requirements.

The command used to create and run the VM is shown below:

```
[root@                ]# /usr/libexec/qemu-kvm -boot d -machine virt,accel=kvm,gic-version=3 \
> -m 32G,slots=6,maxmem=64G -enable-kvm \
> -cpu host -smp cpus=32,cores=32,threads=1,sockets=1,maxcpus=32 \
> -drive file=/dev/shm/ilyass/image/disk-ol89.qcow2,if=virtio,aio=threads,format=qcow2 \
> -drive file=/usr/share/AAVMF/AAVMF_CODE.pure-efi.fd,if=pflash,format=raw,unit=0,readonly=on \
> -drive file=/tmp/AAVMF_VARS.pure-efi.fd,if=pflash,format=raw,unit=1 \
> -drive file=/dev/shm/ilyass/OracleLinux-R8-U10-aarch64-dvd.iso,media=cdrom \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=2,chassis=2,id=piciroot2,bus=pcie.0,addr=0x2 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=3,chassis=3,id=piciroot3,bus=pcie.0,addr=0x3 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=4,chassis=4,id=piciroot4,bus=pcie.0,addr=0x4 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=5,chassis=5,id=piciroot5,bus=pcie.0,addr=0x5 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=6,chassis=6,id=piciroot6,bus=pcie.0,addr=0x6 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=7,chassis=7,id=piciroot7,bus=pcie.0,addr=0x7 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=8,chassis=8,id=piciroot8,bus=pcie.0,addr=0x8 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=9,chassis=9,id=piciroot9,bus=pcie.0,addr=0x9 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=10,chassis=10,id=piciroot10,bus=pcie.0,addr=0xA \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=11,chassis=11,id=piciroot11,bus=pcie.0,addr=0xB \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=12,chassis=12,id=piciroot12,bus=pcie.0,addr=0xC \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=13,chassis=13,id=piciroot13,bus=pcie.0,addr=0xD \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=14,chassis=14,id=piciroot14,bus=pcie.0,addr=0xE \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=15,chassis=15,id=piciroot15,bus=pcie.0,addr=0xF \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=16,chassis=16,id=piciroot16,bus=pcie.0,addr=0x10 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=17,chassis=17,id=piciroot17,bus=pcie.0,addr=0x11 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=18,chassis=18,id=piciroot18,bus=pcie.0,addr=0x12 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=19,chassis=19,id=piciroot19,bus=pcie.0,addr=0x13 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=20,chassis=20,id=piciroot20,bus=pcie.0,addr=0x14 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=21,chassis=21,id=piciroot21,bus=pcie.0,addr=0x15 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=22,chassis=22,id=piciroot22,bus=pcie.0,addr=0x16 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=23,chassis=23,id=piciroot23,bus=pcie.0,addr=0x17 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=24,chassis=24,id=piciroot24,bus=pcie.0,addr=0x18 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=25,chassis=25,id=piciroot25,bus=pcie.0,addr=0x19 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=26,chassis=26,id=piciroot26,bus=pcie.0,addr=0x1A \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=27,chassis=27,id=piciroot27,bus=pcie.0,addr=0x1B \
> -chardev socket,id=mon_vm,path=/tmp/hmp_vm1,server=on,wait=off \
> -mon chardev=mon_vm,mode=readline \
> -qmp unix:/tmp/qmp_vm1,server=on,wait=off \
> -nodefaults \
> -vnc :2 -vga std \
> -netdev user,id=netdev1,hostfwd=tcp::6002-:22 \
> -device virtio-net-pci,id=net1,netdev=netdev1
```

Figure 3.15. Creating VM Using QEMU

This command includes several important components:

- **-boot d**: This option tells QEMU to boot from the first virtual CD-ROM drive;
- **-machine virt,accel=kvm,gic-version=3**: Specifies the machine type and enables KVM acceleration, which improves performance by allowing the VM to run directly on the host's hardware;
- **-m 32G,slots=6,maxmem=64G**: Allocates 32GB of memory to the VM with six additional memory slots, allowing the memory to be expanded up to 64GB;
- **-cpu host,+host-phys-bits**: This option uses the host CPU model and enables the use of physical address bits, which can improve performance by allowing the VM to use more memory addresses;

- **-smp cpus=32,cores=32,threads=2,sockets=1,maxcpus=64**: Configures the VM to use 32 CPUs, organized into 32 cores with 2 threads per core, and allows the number of CPUs to scale up to 64;
- **-drive file=/dev/shm/ilyass/image/disk-ol89.qcow2**: Specifies the QCOW2 disk image for the VM;
- **-drive file=/usr/share/AAVMF/AAVMF_CODE.pure-efi.fd,if=pflash,format=raw,unit=0,readonly=on**: Loads the UEFI firmware code for the VM, allowing it to boot using UEFI;
- **-drive file=/tmp/AAVMF_VARS.pure-efi.fd,if=pflash,format=raw,unit=1:** Specifies the UEFI variables file, which stores information like boot order and settings;
- **-device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=2,chassis=2,id=pciroot2,bus=pcie.0,addr=0x2**: Adds a PCIe root port to the VM, which is essential for hot-plugging devices like VNICs;
- **-chardev socket,id=mon_vm,path=/tmp/hmp_vm1,server=on,wait=off:** Configures a character device for QEMU's monitor interface, allowing real-time management of the VM;
- **-mon chardev=mon_vm,mode=readline**: Sets up the monitor interface for interactive commands;
- **-qmp unix:/tmp/qmp_vm1,server=on,wait=off**: Enables QEMU's QMP (QEMU Machine Protocol) server, providing a powerful API for VM management;
- **-nodefaults**: Disables QEMU's default devices, giving full control over the VM's configuration;
- **-vnc :2 -vga std**: Enables VNC (Virtual Network Computing) for remote access to the VM's display and sets up a standard VGA display;
- **-netdev user,id=netdev1,hostfwd=tcp::6002-:22**: Configures user-mode networking with port forwarding, allowing SSH access to the VM on port 6002;
- **-device virtio-net-pci,id=net1,netdev=netdev**: Adds a virtualized network interface to the VM, connecting it to the user-mode network.

This setup ensures that the VM is fully optimized for performance and scalability, making it suitable for the rigorous testing scenarios.

3.2.7 Creating the VM Using Libvirt

In the Libvirt-based tests, the virtual machine (VM) creation was handled through Libvirt commands, providing a streamlined approach for managing VM configurations, storage pools, and other settings. This method offers fine-grained control over the VM setup and facilitates the use of advanced features provided by Libvirt.

The command used to create the VM is illustrated in the image below:

```
GNU nano 2.9.8                    /dev/shm/ilyass/vm/create.bash

#!/bin/bash

# Defining variables
virsh='virsh --connect qemu:///system'
virt_install='virt-install --connect qemu:///system'
vm_name=vm_ol810

# Creating the VM
virt-install \
  --name $vm_name \
  --virt-type kvm \
  --boot loader=/usr/share/AAVMF/AAVMF_CODE.pure-efi.fd,loader_ro=yes,loader_type=pflash,nvram_template=/usr/share/AAVMF/AAVMF_VARS.pure-efi.fd,loader_secure=no \
  --memory=8192 \
  --vcpu=2 \
  --disk /dev/shm/ilyass/images/disk-ol810.qcow2.qcow2,bus=usb,size=25 \
  --cdrom=/dev/shm/ilyass/OracleLinux-R8-U10-aarch64-dvd.iso \
  --network network=default \
  --graphics vnc,listen=0.0.0.0 \
  --noautoconsole -v
```

Figure 3.16. Creating VM Using Libvirt

The command includes several key elements:

- **virsh='virsh –connect qemu:///system'**: Sets up the `virsh` command to interact with the QEMU system through Libvirt;
- **virt_install='virt-install –connect qemu:///system'**: Defines the `virt-install` command for creating and installing the VM;
- **vm_name=vm_ol810**: Specifies the name of the VM as 'vm_ol810';
- **–virt-type kvm**: Specifies that the VM will use KVM as the virtualization type;
- **–boot loader=...**: Configures the boot loader with UEFI firmware, including the secure boot options and NVRAM template;
- **–network network=default**: Connects the VM to the default virtual network;
- **–graphics vnc,listen=0.0.0.0**: Configures VNC for remote graphical access to the VM;
- **–noautoconsole -v**: Prevents automatic console connection and enables verbose output for better logging.

This setup provides a comprehensive approach to VM creation using Libvirt, ensuring proper configuration and access for subsequent testing and management.

3.2.8 Interaction with the VM

Interacting with a virtual machine (VM) is a crucial aspect of managing and utilizing virtualized environments. Various methods can be employed to access and control the VM, each offering different functionalities and levels of control. Below, we explore two primary methods: using VNC for graphical access and leveraging the QEMU Monitor for advanced management.

3.2.8.1 Using VNC

Virtual Network Computing (VNC) is a widely-used protocol for remote graphical access to a VM. By connecting through VNC, users can interact with the VM's desktop environment as if they were physically present at the machine. This approach is especially beneficial for:

- **Visual Interaction**: VNC allows for direct interaction with the VM's graphical user interface (GUI), making it easier to perform visual checks, configure settings, and troubleshoot issues from a remote location;
- **Basic Operations**: For tasks that require GUI interaction, such as software installations or configuration changes that are more intuitive with a graphical interface, VNC provides an effective solution;
- **Remote Access**: VNC facilitates remote access to the VM, which is advantageous for users who need to work from different locations or for administrators managing multiple VMs.

To utilize VNC for interacting with the VM, it is necessary to first configure the VM with a VNC server. Once configured, we can connect to the VM using a VNC client.

Figure 3.17. VNC Client

3.2.8.2 Using QEMU Monitor

The QEMU Monitor is a powerful tool that provides a command-line interface for managing and interacting with a VM. It offers several advanced features, making it an essential component for users who need deeper control over the VM's operations. Key functionalities include:

- **Hardware Configuration**: The QEMU Monitor allows for real-time adjustments to the VM's hardware configuration, such as adding or removing devices, modifying memory allocation, or changing CPU settings;
- **Snapshot Management**: Users can manage VM snapshots, enabling them to save and revert to previous states of the VM as needed;
- **Performance Monitoring**: The QEMU Monitor provides commands to monitor various performance metrics of the VM, such as CPU usage, memory utilization, and I/O statistics, helping in performance tuning and troubleshooting;

- **Real-time Commands**: Execute real-time commands to control the VM, such as pausing, resuming, or shutting down the VM.

To access the QEMU Monitor, a command-line interface is typically utilized, either directly from the terminal where the VM is running or through a dedicated monitor interface if configured. Commands are entered in a text-based format, providing precise control and configuration of the VM.

```
[root@                    ]# nc -U /tmp/hmp_vm1
QEMU 7.2.0 monitor - type 'help' for more information
(qemu) info status
info status
VM status: running
(qemu)
```

Figure 3.18. QEMU Monitor

Both VNC and QEMU Monitor provide essential methods for interacting with VMs, catering to different needs—from graphical user interaction to in-depth configuration and management.

3.3 Testing Procedures

3.3.1 Life Cycle Testing

Life cycle testing is a crucial step in validating the stability and reliability of a virtual machine (VM). This test involves performing a series of operations on the VM, such as starting, rebooting, resetting, stopping, and shutting down. Each operation tests a specific aspect of the VM's lifecycle management.

For instance, the reboot process ensures that the VM can restart without data loss or corruption, while the shutdown operation tests the VM's ability to terminate processes gracefully. The reset function is particularly important for checking if the VM can recover from an unresponsive state without requiring a full shutdown.

By systematically executing these operations, we verify the VM's robustness and its ability to handle various states of operation under different conditions. The results of these tests indicate whether the VM can consistently manage transitions between these states, which is essential for maintaining the overall health of the virtualization environment.

3.3.2 VNIC Hotplug/Unplug

The VNIC (Virtual Network Interface Card) Hotplug/Unplug test evaluates the VM's capability to dynamically add and remove network interfaces without requiring a reboot. In this test, some VNICs are sequentially added and removed from the VM, and the system's behavior

is closely monitored.

This test is critical for scenarios where network scalability and flexibility are required. The key focus is on ensuring that the VM can recognize new VNICs and reconfigure itself accordingly, as well as correctly release resources when VNICs are removed. This involves not only the successful attachment and detachment of the VNICs but also verifying that the network performance remains stable throughout the process.

3.3.3 VFIO-VNIC Hotplug/Unplug

The VFIO-VNIC Hotplug/Unplug test builds on the standard VNIC test by introducing Virtual Functions (VFs) through the VFIO (Virtual Function I/O) framework. VFs are essentially lightweight versions of physical network interfaces that provide dedicated resources for high-performance network operations. In this test, VFs are created from a physical VNIC and then hotplugged and unplugged in varying quantities.

The objective is to assess whether the VM can efficiently handle these operations and make use of the VFs without compromising performance or stability. The test also examines how the VM manages the allocation and deallocation of hardware resources associated with VFs. Successful execution of this test demonstrates the VM's ability to support advanced networking features, which is crucial for applications requiring high throughput and low latency.

3.3.4 VDisk Hotplug/Unplug

The vDisk Hotplug/Unplug test is designed to assess the system's ability to dynamically manage storage resources. During this test, multiple virtual disks (vDisk) are added and removed from the running virtual machine (VM) to observe how well the system adapts to these changes without requiring a reboot or shutdown.

The process begins by attaching one or more vDisks to the VM and verifying that the guest OS recognizes the newly added storage devices. This involves checking whether the disks are correctly identified in the OS, ensuring they can be mounted, partitioned, and used for file operations without errors. Afterward, these disks are detached, and the system's response is monitored to confirm that the detachment is clean, with no lingering references or errors in the OS logs.

3.3.5 Memory Hotplug/Unplug

The Memory Hotplug/Unplug test focuses on evaluating the VM's ability to handle changes in memory allocation during runtime. In this test, additional memory is dynamically added to a VM to see if the guest OS can recognize and utilize the extra RAM without requiring a restart. This is followed by the removal of the added memory to check whether the system can gracefully release the memory back to the host without any stability issues.

The test procedure includes monitoring system logs and performance metrics to ensure that memory management functions correctly during the hotplug and unplug processes. This testing is vital in scenarios where workloads vary and demand flexible memory management, such as in virtualized data centers.

3.3.6 VCPU Hotplug/Unplug

The vCPU Hotplug/Unplug test is performed to verify the system's capability to dynamically adjust processing power by adding or removing virtual CPUs (vCPUs) to the VM while it is running. The test starts by incrementally adding vCPUs to the VM and verifying that the guest OS detects and utilizes the additional processing resources. This involves running CPU-intensive tasks to observe if the additional vCPUs contribute to improved performance. Subsequently, vCPUs are removed, and the system's ability to redistribute the remaining processing workload without causing instability or performance degradation is assessed.

The process is carefully monitored for any signs of errors or inefficiencies in CPU scheduling and task handling. This test is particularly important in environments where computing demands fluctuate, necessitating a scalable processing capability.

3.3.7 Kdump Check

Kdump is a kernel crash dumping mechanism that captures system state information during a crash, aiding in post-crash analysis. To verify Kdump's functionality, the following steps were performed:

- Enable SysRq Trigger using: `sysctl -w kernel.sysrq=1`;
- A Kernel crash was manually induced using: `echo c > /proc/sysrq-trigger`;
- After rebooting, the presence of crash logs in `/var/crash` or `/var/oled/crash` was checked to confirm Kdump's functionality.

This process ensures that Kdump is correctly capturing crash data, essential for system diagnostics.

3.3.8 Big VM 500G-1T Boot Test

In the Big VM 500G-1T Boot Test, the VM's memory allocation is significantly increased, first to 500GB and then to 1TB, to test its ability to boot with such large memory configurations. This test is critical for verifying the scalability of the VM in handling large workloads that require substantial memory resources.

The boot process is carefully monitored to ensure that the VM can initialize and manage the expanded memory without encountering errors or significant delays. Successfully passing this test demonstrates the VM's capability to support high-memory environments, which is essential for enterprise-level applications and large-scale data processing tasks.

3.4 Conclusion

This chapter has detailed the primary tasks of the internship, focusing on QEMU and Libvirt tests. It covered the setup of environments, installation procedures, and the creation of virtual machines using both QEMU and Libvirt. These processes form the foundation for testing and validating virtualized environments, ensuring that the systems perform as expected across different configurations.

Chapter 4

Tests Realization and Results

In this chapter, we will delve into the implementation and validation of our testing procedures for the Oracle Linux Virtualization and System Testing project. We will begin by outlining the technical choices and configurations used in our tests. Subsequently, we will describe each test in detail, including the setup, execution, and analysis of results.

4.1 Test on OL8 Host with an Intel CPU, QEMU and latest kernel version UEK7U2

4.1.1 Host System Configuration

To begin the testing process, we set up an Intel-based host running Oracle Linux Server 8.10, which utilizes the x86-64 architecture. The system was further configured with the latest version of the UEK7U2. To verify the environment, the hostnamectl command was executed, confirming the successful installation and configuration:

```
[root@                                ]# hostnamectl
   Static hostname:
         Icon name: computer-server
           Chassis: server
        Machine ID:
           Boot ID:
  Operating System: Oracle Linux Server 8.10
       CPE OS Name: cpe:/o:oracle:linux:8:10:server
            Kernel: Linux                    .el8uek.x86_64
      Architecture: x86-64
```

Figure 4.1. Verification of Host Configuration Using hostnamectl Command

After the host environment was confirmed, QEMU was installed. The successful installation of QEMU and its associated packages was verified using the yum command, which displayed all the installed components, including qemu-img, qemu-kvm, and several QEMU-related modules:

```
[root@                          ]# yum list installed | grep qemu
qemu-img.x86_64                                    .el8                    @custom1
qemu-kvm.x86_64                                    .el8                    @custom1
qemu-kvm-block-curl.x86_64                         .el8                    @custom1
qemu-kvm-block-gluster.x86_64                      .el8                    @custom1
qemu-kvm-block-iscsi.x86_64                        .el8                    @custom1
qemu-kvm-block-rbd.x86_64                          .el8                    @custom1
qemu-kvm-block-ssh.x86_64                          .el8                    @custom1
qemu-kvm-common.x86_64                             .el8                    @custom1
qemu-kvm-core.x86_64                               .el8                    @custom1
qemu-kvm-docs.x86_64                                                       @ol8_appstream
```

Figure 4.2. Installed QEMU Version Verification

Furthermore, we confirmed the presence of OVMF files and checked the edk2 package to ensure that the necessary UEFI components were available. These files are critical for running UEFI-based virtual machines:

```
[root@                          ]# ls -l /usr/share/OVMF/
total 0
lrwxrwxrwx. 1 root root 34 Apr  9 15:29 OVMF_CODE.cc-debug.fd -> ../edk2/ovmf/OVMF_CODE.cc-debug.fd
lrwxrwxrwx. 1 root root 39 Apr  9 15:29 OVMF_CODE.cc-debug.fd.hmac -> ../edk2/ovmf/OVMF_CODE.cc-debug.fd.hmac
lrwxrwxrwx. 1 root root 28 Apr  9 15:29 OVMF_CODE.cc.fd -> ../edk2/ovmf/OVMF_CODE.cc.fd
lrwxrwxrwx. 1 root root 33 Apr  9 15:29 OVMF_CODE.cc.fd.hmac -> ../edk2/ovmf/OVMF_CODE.cc.fd.hmac
lrwxrwxrwx. 1 root root 25 Apr  9 15:29 OVMF_CODE.fd -> ../edk2/ovmf/OVMF_CODE.fd
lrwxrwxrwx. 1 root root 30 Apr  9 15:29 OVMF_CODE.fd.hmac -> ../edk2/ovmf/OVMF_CODE.fd.hmac
lrwxrwxrwx. 1 root root 40 Apr  9 15:29 OVMF_CODE.pure-efi-debug.fd -> ../edk2/ovmf/OVMF_CODE.pure-efi-debug.fd
lrwxrwxrwx. 1 root root 45 Apr  9 15:29 OVMF_CODE.pure-efi-debug.fd.hmac -> ../edk2/ovmf/OVMF_CODE.pure-efi-debug.fd.hmac
lrwxrwxrwx. 1 root root 46 Apr  9 15:29 OVMF_CODE.pure-efi-notpm-debug.fd -> ../edk2/ovmf/OVMF_CODE.pure-efi-notpm-debug.fd
lrwxrwxrwx. 1 root root 51 Apr  9 15:29 OVMF_CODE.pure-efi-notpm-debug.fd.hmac -> ../edk2/ovmf/OVMF_CODE.pure-efi-notpm-debug.fd.hmac
lrwxrwxrwx. 1 root root 40 Apr  9 15:29 OVMF_CODE.pure-efi-notpm.fd -> ../edk2/ovmf/OVMF_CODE.pure-efi-notpm.fd
lrwxrwxrwx. 1 root root 45 Apr  9 15:29 OVMF_CODE.pure-efi-notpm.fd.hmac -> ../edk2/ovmf/OVMF_CODE.pure-efi-notpm.fd.hmac
lrwxrwxrwx. 1 root root 34 Apr  9 15:29 OVMF_CODE.pure-efi.fd -> ../edk2/ovmf/OVMF_CODE.pure-efi.fd
lrwxrwxrwx. 1 root root 39 Apr  9 15:29 OVMF_CODE.pure-efi.fd.hmac -> ../edk2/ovmf/OVMF_CODE.pure-efi.fd.hmac
lrwxrwxrwx. 1 root root 39 Apr  9 15:29 OVMF_CODE.secboot-debug.fd -> ../edk2/ovmf/OVMF_CODE.secboot-debug.fd
lrwxrwxrwx. 1 root root 44 Apr  9 15:29 OVMF_CODE.secboot-debug.fd.hmac -> ../edk2/ovmf/OVMF_CODE.secboot-debug.fd.hmac
lrwxrwxrwx. 1 root root 33 Apr  9 15:29 OVMF_CODE.secboot.fd -> ../edk2/ovmf/OVMF_CODE.secboot.fd
lrwxrwxrwx. 1 root root 38 Apr  9 15:29 OVMF_CODE.secboot.fd.hmac -> ../edk2/ovmf/OVMF_CODE.secboot.fd.hmac
lrwxrwxrwx. 1 root root 34 Apr  9 15:29 OVMF_VARS.cc-debug.fd -> ../edk2/ovmf/OVMF_VARS.cc-debug.fd
lrwxrwxrwx. 1 root root 39 Apr  9 15:29 OVMF_VARS.cc-debug.fd.hmac -> ../edk2/ovmf/OVMF_VARS.cc-debug.fd.hmac
lrwxrwxrwx. 1 root root 28 Apr  9 15:29 OVMF_VARS.cc.fd -> ../edk2/ovmf/OVMF_VARS.cc.fd
lrwxrwxrwx. 1 root root 33 Apr  9 15:29 OVMF_VARS.cc.fd.hmac -> ../edk2/ovmf/OVMF_VARS.cc.fd.hmac
lrwxrwxrwx. 1 root root 25 Apr  9 15:29 OVMF_VARS.fd -> ../edk2/ovmf/OVMF_VARS.fd
lrwxrwxrwx. 1 root root 30 Apr  9 15:29 OVMF_VARS.fd.hmac -> ../edk2/ovmf/OVMF_VARS.fd.hmac
lrwxrwxrwx. 1 root root 40 Apr  9 15:29 OVMF_VARS.pure-efi-debug.fd -> ../edk2/ovmf/OVMF_VARS.pure-efi-debug.fd
lrwxrwxrwx. 1 root root 45 Apr  9 15:29 OVMF_VARS.pure-efi-debug.fd.hmac -> ../edk2/ovmf/OVMF_VARS.pure-efi-debug.fd.hmac
lrwxrwxrwx. 1 root root 46 Apr  9 15:29 OVMF_VARS.pure-efi-notpm-debug.fd -> ../edk2/ovmf/OVMF_VARS.pure-efi-notpm-debug.fd
lrwxrwxrwx. 1 root root 51 Apr  9 15:29 OVMF_VARS.pure-efi-notpm-debug.fd.hmac -> ../edk2/ovmf/OVMF_VARS.pure-efi-notpm-debug.fd.hmac
lrwxrwxrwx. 1 root root 40 Apr  9 15:29 OVMF_VARS.pure-efi-notpm.fd -> ../edk2/ovmf/OVMF_VARS.pure-efi-notpm.fd
lrwxrwxrwx. 1 root root 45 Apr  9 15:29 OVMF_VARS.pure-efi-notpm.fd.hmac -> ../edk2/ovmf/OVMF_VARS.pure-efi-notpm.fd.hmac
lrwxrwxrwx. 1 root root 34 Apr  9 15:29 OVMF_VARS.pure-efi.fd -> ../edk2/ovmf/OVMF_VARS.pure-efi.fd
lrwxrwxrwx. 1 root root 39 Apr  9 15:29 OVMF_VARS.pure-efi.fd.hmac -> ../edk2/ovmf/OVMF_VARS.pure-efi.fd.hmac
lrwxrwxrwx. 1 root root 39 Apr  9 15:29 OVMF_VARS.secboot-debug.fd -> ../edk2/ovmf/OVMF_VARS.secboot-debug.fd
lrwxrwxrwx. 1 root root 44 Apr  9 15:29 OVMF_VARS.secboot-debug.fd.hmac -> ../edk2/ovmf/OVMF_VARS.secboot-debug.fd.hmac
lrwxrwxrwx. 1 root root 33 Apr  9 15:29 OVMF_VARS.secboot.fd -> ../edk2/ovmf/OVMF_VARS.secboot.fd
lrwxrwxrwx. 1 root root 38 Apr  9 15:29 OVMF_VARS.secboot.fd.hmac -> ../edk2/ovmf/OVMF_VARS.secboot.fd.hmac
```

Figure 4.3. Checking UEFI Components in OVMF Directory

```
[root@                          ]# yum list installed | grep edk2
edk2-ovmf.noarch                                                    @edk2
edk2-tools.x86_64                                                   @edk2
ipxe-roms-hops.x86_64                                               @edk2
```

Figure 4.4. Verifying edk2 Version for UEFI Compatibility

To ensure the system had sufficient resources for running and testing virtual machines, we recorded the available storage and memory using the df -h and lsmem commands, respectively. This information is crucial for the development team to troubleshoot any issues related to resource allocation during the testing process:

```
[root@                          opc]# df -h
Filesystem                     Size  Used Avail Use% Mounted on
devtmpfs                       252G     0  252G   0% /dev
tmpfs                          252G   17G  235G   7% /dev/shm
tmpfs                          252G  787M  251G   1% /run
tmpfs                          252G     0  252G   0% /sys/fs/cgroup
/dev/mapper/ocivolume-root      36G   27G  8.8G  76% /
/dev/mapper/ocivolume-oled      10G  2.0G  8.1G  20% /var/oled
/dev/sda2                     1014M  794M  221M  79% /boot
/dev/sda1                      100M  6.0M   94M   6% /boot/efi
tmpfs                           51G     0   51G   0% /run/user/1000
tmpfs                           51G     0   51G   0% /run/user/986
```

Figure 4.5. Available Storage Capacity on Host System

```
[root@                                ]# lsmem
RANGE                                  SIZE  STATE REMOVABLE BLOCK
0x0000000000000000-0x000000007fffffff    2G online       yes     0
0x0000000100000000-0x000000807fffffff  510G online       yes 2-256

Memory block size:         2G
Total online memory:     512G
Total offline memory:      0B
```

Figure 4.6. Host Memory Information

4.1.2 Guest Virtual Machine Deployment

With the host system fully configured, we proceeded to run the guest VM. The VM was initiated using the following QEMU command, which defined various parameters including memory allocation, CPU cores, and disk images. This command ensured that the VM was properly configured to mirror a typical production environment:

```
[root@                          ]# /usr/libexec/qemu-kvm -boot d -machine q35,kernel_irqchip=split \
> -m 32G,slots=6,maxmem=64G -enable-kvm \
> -cpu host,+host-phys-bits -smp cpus=32,cores=32,threads=1,sockets=1,maxcpus=32 \
> -drive file=/dev/shm/test/OracleLinux-8.9-2024.01.26-0-uefi-x86_64.qcow2 \
> -drive file=/usr/share/OVMF/OVMF_CODE.pure-efi.fd,if=pflash,format=raw,unit=0,readonly=on \
> -drive file=/tmp/OVMF_VARS.pure-efi.fd,if=pflash,format=raw,unit=1 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=2,chassis=2,id=pciroot2,bus=pcie.0,addr=0x2 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=3,chassis=3,id=pciroot3,bus=pcie.0,addr=0x3 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=4,chassis=4,id=pciroot4,bus=pcie.0,addr=0x4 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=5,chassis=5,id=pciroot5,bus=pcie.0,addr=0x5 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=6,chassis=6,id=pciroot6,bus=pcie.0,addr=0x6 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=7,chassis=7,id=pciroot7,bus=pcie.0,addr=0x7 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=8,chassis=8,id=pciroot8,bus=pcie.0,addr=0x8 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=9,chassis=9,id=pciroot9,bus=pcie.0,addr=0x9 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=10,chassis=10,id=pciroot10,bus=pcie.0,addr=0xA \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=11,chassis=11,id=pciroot11,bus=pcie.0,addr=0xB \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=12,chassis=12,id=pciroot12,bus=pcie.0,addr=0xC \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=13,chassis=13,id=pciroot13,bus=pcie.0,addr=0xD \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=14,chassis=14,id=pciroot14,bus=pcie.0,addr=0xE \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=15,chassis=15,id=pciroot15,bus=pcie.0,addr=0xF \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=16,chassis=16,id=pciroot16,bus=pcie.0,addr=0x10 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=17,chassis=17,id=pciroot17,bus=pcie.0,addr=0x11 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=18,chassis=18,id=pciroot18,bus=pcie.0,addr=0x12 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=19,chassis=19,id=pciroot19,bus=pcie.0,addr=0x13 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=20,chassis=20,id=pciroot20,bus=pcie.0,addr=0x14 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=21,chassis=21,id=pciroot21,bus=pcie.0,addr=0x15 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=22,chassis=22,id=pciroot22,bus=pcie.0,addr=0x16 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=23,chassis=23,id=pciroot23,bus=pcie.0,addr=0x17 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=24,chassis=24,id=pciroot24,bus=pcie.0,addr=0x18 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=25,chassis=25,id=pciroot25,bus=pcie.0,addr=0x19 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=26,chassis=26,id=pciroot26,bus=pcie.0,addr=0x1A \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=27,chassis=27,id=pciroot27,bus=pcie.0,addr=0x1B \
> -chardev socket,id=mon_vm,path=/tmp/hmp_vm1,server=on,wait=off \
> -mon chardev=mon_vm,mode=readline \
> -qmp unix:/tmp/qmp_vm1,server=on,wait=off \
> -nodefaults \
> -vnc :2 -vga std \
> -serial mon:stdio \
> -netdev user,id=netdev1,hostfwd=tcp::6002-:22 \
> -device virtio-net-pci,id=net1,netdev=netdev1
```

Figure 4.7. Launching Guest VM with QEMU Command

After launching the guest, we confirmed that it was running Oracle Linux Server 8.9 with UEK7U2. This was verified by executing the hostnamectl and uname -r commands within the guest VM:

```
[root@OL89_VM ~]# hostnamectl
   Static hostname: OL89_VM
Transient hostname: localhost.localdomain
         Icon name: computer-vm
           Chassis: vm
        Machine ID:
           Boot ID:
    Virtualization: kvm
  Operating System: Oracle Linux Server 8.9
       CPE OS Name: cpe:/o:oracle:linux:8:9:server
            Kernel: Linux                  .el8uek.x86_64
      Architecture: x86-64
[root@OL89_VM ~]#
```

Figure 4.8. Verifying Hostname within the Guest

```
[root@OL89_VM ~]# uname -r
                  .el8uek.x86_64
```

Figure 4.9. Kernel Version Verification in Guest

This verification confirmed that the VM was correctly set up and ready for the subsequent testing procedures.

4.1.3 Test Execution and Performance Evaluation

To evaluate the stability and performance of the virtual environment, a series of tests were conducted. These tests focused on the VM's lifecycle operations, networking capabilities, and its ability to handle hotplug and unplug events for multiple virtual network interfaces.

4.1.3.1 Lifecycle Test

The lifecycle test aimed to verify the VM's ability to handle common operational states, such as rebooting, stopping and continuing operations, suspending and waking, and shutting down. Each of these states was carefully tested to ensure that the VM could transition smoothly without any issues.

- **Reboot Test:**
  The VM was subjected to a reboot operation using the system_reset command from the QEMU monitor. This command initiated a full system reboot, simulating scenarios where the VM might need to restart due to system updates or other maintenance tasks. Upon issuing the command, the VM successfully rebooted, with all services coming back online without errors. The logs were checked to confirm that no anomalies occurred during the reboot process.

```
QEMU 7.2.0 monitor - type 'help' for more information
(qemu) system_reset
system_reset
(qemu)
```

Figure 4.10. Execution of Reboot Command

After reboot, the guest was checked for uptime and kernel version to ensure that it was the same system with no inconsistencies:

```
[root@OL89_VM ~]# uptime
 01:32:36 up 1 min,  1 user,  load average: 0.00, 0.00, 0.00
[root@OL89_VM ~]# uname -r
                .el8uek.x86_64
[root@OL89_VM ~]#
```

Figure 4.11. Post-Reboot Guest Verification

- **Stop/Continue Test:**
  In this test, the VM was temporarily stopped using the stop command in the QEMU monitor. This action simulates a scenario where the VM might be paused during a maintenance window or to free up resources temporarily. The VM was then resumed using the cont command. Throughout this process, the VM's state was preserved, and it resumed operations seamlessly.

```
[root@OL89_VM ~]# ping localhost
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.030 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.019 ms
64 bytes from localhost (::1): icmp_seq=5 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=6 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=7 ttl=64 time=0.019 ms
```

Figure 4.12. Network Connectivity During VM Stop Operation

```
(qemu) stop
stop
(qemu) cont
cont
(qemu)
```

Figure 4.13. Stop/Cont Commands Executed on the Guest

```
[root@OL89_VM ~]# ping localhost
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.030 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.019 ms
64 bytes from localhost (::1): icmp_seq=5 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=6 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=7 ttl=64 time=0.019 ms
64 bytes from localhost (::1): icmp_seq=8 ttl=64 time=0.044 ms
64 bytes from localhost (::1): icmp_seq=9 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=10 ttl=64 time=0.019 ms
64 bytes from localhost (::1): icmp_seq=11 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=12 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=13 ttl=64 time=0.020 ms
64 bytes from localhost (::1): icmp_seq=14 ttl=64 time=0.060 ms
64 bytes from localhost (::1): icmp_seq=15 ttl=64 time=0.072 ms
64 bytes from localhost (::1): icmp_seq=16 ttl=64 time=0.060 ms
```

Figure 4.14. Network Connectivity After Resuming VM

Upon continuation, network connectivity, application services, and system responsiveness were all confirmed to be intact.

- **Suspend Test:**
  The suspend operation was tested to simulate putting the VM into a low-power state. The systemctl suspend command was used within the guest to initiate this state. The wakeup was triggered using the system_wakeup command from the QEMU monitor, bringing the VM back to an active state.

```
[root@OL89_VM ~]# systemctl suspend
[root@OL89_VM ~]# [ 1226.915253] PM: suspend entry (deep)
[ 1226.916127] Filesystems sync: 0.000 seconds
[ 1226.916628] Freezing user space processes ... (elapsed 0.000 seconds) done.
[ 1226.918126] OOM killer disabled.
[ 1226.918371] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[ 1226.920277] printk: Suspending console(s) (use no_console_suspend to debug)
```

Figure 4.15. Guest VM Suspension Executed

```
(qemu) system_wakeup
system_wakeup
(qemu)
```

Figure 4.16. Executing Wakeup Command via QEMU Monitor

```
[ 1275.425998] kvm-clock: cpu 30, msr 189c01781, secondary cpu clock
[ 1275.426304] kvm-guest: setup async PF for cpu 30
[ 1275.426308] kvm-guest: stealtime: cpu 30, msr 85fbb4080
[ 1275.427222] CPU30 is up
[ 1275.427264] smpboot: Booting Node 0 Processor 31 APIC 0x1f
[ 1275.427457] kvm-clock: cpu 31, msr 189c017c1, secondary cpu clock
[ 1275.427766] kvm-guest: setup async PF for cpu 31
[ 1275.427770] kvm-guest: stealtime: cpu 31, msr 85fbf4080
[ 1275.428745] CPU31 is up
[ 1275.430633] ACPI: PM: Waking up from system sleep state S3
[ 1275.485566] scsi host0: scsi_eh_0: waking up 1/0/0
[ 1275.485571] scsi host4: scsi_eh_4: waking up 1/0/0
[ 1275.485580] scsi host2: scsi_eh_2: waking up 1/0/0
[ 1275.485582] scsi host1: scsi_eh_1: waking up 1/0/0
[ 1275.485584] scsi host3: scsi_eh_3: waking up 1/0/0
[ 1275.485592] scsi host5: scsi_eh_5: waking up 1/0/0
[ 1275.485601] sd 0:0:0:0: [sda] Starting disk
[ 1275.645944] OOM killer enabled.
[ 1275.646368] Restarting tasks ... done.
[ 1275.647741] PM: suspend exit
[ 1275.796390] ata4: SATA link down (SStatus 0 SControl 300)
[ 1275.797388] scsi host3: waking up host to restart
[ 1275.797446] ata3: SATA link down (SStatus 0 SControl 300)
[ 1275.797913] scsi host3: scsi_eh_3: sleeping
[ 1275.798562] scsi host2: waking up host to restart
[ 1275.798644] ata2: SATA link down (SStatus 0 SControl 300)
[ 1275.798687] scsi host1: waking up host to restart
[ 1275.798689] scsi host1: scsi_eh_1: sleeping
[ 1275.798776] ata6: SATA link down (SStatus 0 SControl 300)
[ 1275.798808] scsi host5: waking up host to restart
[ 1275.798810] scsi host5: scsi_eh_5: sleeping
[ 1275.798889] ata5: SATA link down (SStatus 0 SControl 300)
[ 1275.798914] scsi host4: waking up host to restart
[ 1275.798916] scsi host4: scsi_eh_4: sleeping
[ 1275.798980] ata1: SATA link up 1.5 Gbps (SStatus 113 SControl 300)
[ 1275.799163] ata1.00: configured for UDMA/100
[ 1275.799630] scsi host2: scsi_eh_2: sleeping
[ 1275.800202] scsi host0: waking up host to restart
[ 1275.806309] scsi host0: scsi_eh_0: sleeping

[root@OL89_VM ~]#
```

Figure 4.17. Verifying the Guest After Wakeup

The VM successfully transitioned in and out of the suspend state, with all previously running processes and services continuing from where they left off. Network connectivity was reestablished without any manual intervention.

- **Shutdown Test:**
  Finally, the VM was cleanly shut down using the system_powerdown command in the QEMU monitor. This test is crucial as it ensures that the VM can safely terminate its operations and power down without data loss or corruption.

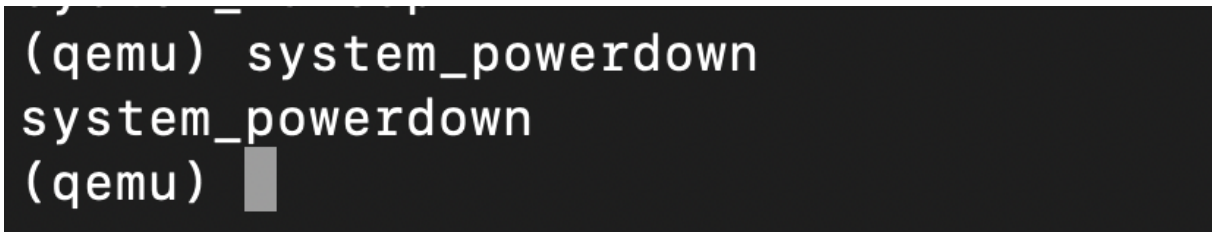


Figure 4.18. Executing Shutdown Test on the Guest

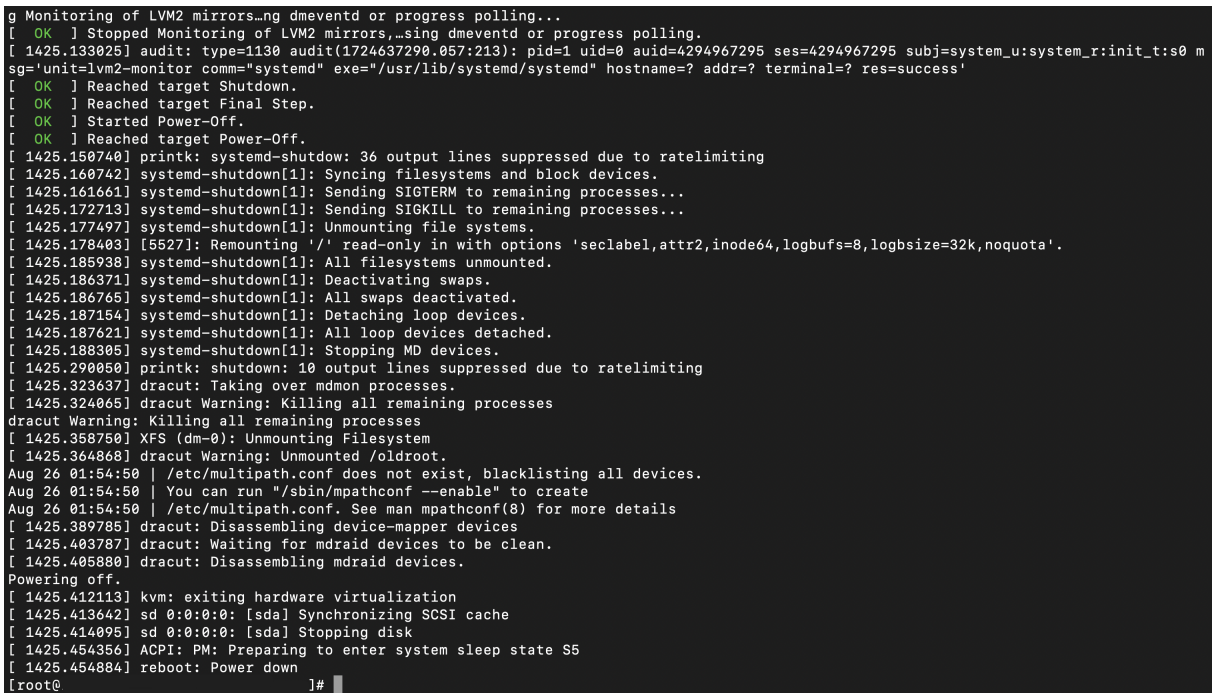


Figure 4.19. Verification of Shutdown Process in the Guest

The guest VM performed a clean shutdown, with all services stopping gracefully and the machine powering off as expected.

4.1.3.2 Some VNIC Hotplug/Unplug

One of the critical tests involved the hotplug and unplug of VNICs. This test simulates scenarios where a high number of network interfaces are dynamically added and removed from the VM, which is common in environments that require high networking throughput and flexibility.

Before starting the test, the existing network interfaces in the VM were listed using the ip a command to establish a baseline:

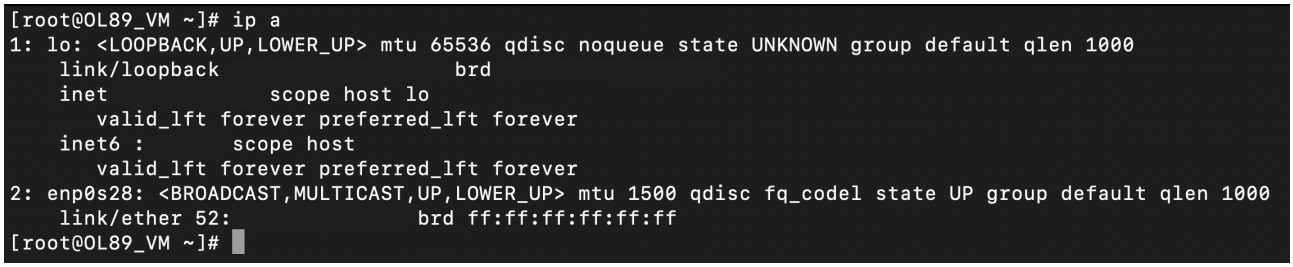


Figure 4.20. Baseline Network Interface Configuration in the Guest

A custom script, VNIC_Hotplug.sh, was developed to automate the process of adding and removing VNICs. This script used QEMU monitor commands to systematically add VNICs to the running VM, simulating a high-density networking environment.

```
[root@                              ]# cat    VNIC_Hotplug.sh
for i in {2..  };do echo -e "netdev_add tap,id=e100${i},script=no,downscript=no\n
device_add virtio-net-pci,id=e100${i},bus=pciroot${i}" | nc -U /tmp/hmp_vm1 ; done
[root@                              ]# sh    VNIC_Hotplug.sh
```

Figure 4.21. Executing VNIC Hotplug Script

During the hotplug process, each new VNIC was detected by the guest, with appropriate entries appearing in the system logs (dmesg) and the network configuration being updated dynamically.

The script iterated through each VNIC, ensuring that all interfaces were added without causing instability or significant performance degradation.

After the VNICs were added, the ip a command was run again to confirm that all interfaces were successfully hotplugged:

```
[root@OL89_VM ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                  brd
    inet            scope host lo
       valid_lft forever preferred_lft forever
    inet6 :       scope host
       valid_lft forever preferred_lft forever
2: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
4: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
5: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
6: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
7: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
8: enp6s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
9: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
```

Figure 4.22. Post-Hotplug Network Interface List in the Guestgit

Each VNIC was successfully added and configured, indicating that the VM's networking stack is robust and capable of handling dynamic changes in network configuration.

For the unplug process. The script then proceeded to remove each VNIC one by one, ensuring that the system could handle the removal process without any issues. The VM continued to operate normally, and the removal of VNICs was verified by checking the updated network interface list.

```
[root@                            ]# cat    VNIC_Unplug.sh
for i in {2..  };do echo -e "device_del e100${i}\n netdev_del e100${i}" | nc -U /tmp/hmp_vm1 ; done
[root@                            ]# sh    VNIC_Unplug.sh
```

Figure 4.23. Executing VNIC Unplug Script

```
[root@OL89_VM ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                  brd
    inet             scope host lo
       valid_lft forever preferred_lft forever
    inet6 :       scope host
       valid_lft forever preferred_lft forever
2: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
[root@OL89_VM ~]#
```

Figure 4.24. Updated Network Interface List After VNIC Unplug

4.1.3.3 Some VFIO-VNIC Hotplug/Unplug

For this test, the process involved the creation and management of some VFIO VNICs on the host system. The initial step required preparing the VFIO VNICs to be bound to the host, ensuring that they were correctly set up for use in the virtualization environment. This involved configuring the VFIO devices so they could be utilized by the guest VM.

```
[root@                          ]# lspci | grep Eth
4b:00.0 Ethernet controller:
4b:00.1 Ethernet controller:
4b:00.2 Ethernet controller:                                        Virtual Function
4b:00.3 Ethernet controller:                                        Virtual Function
4b:00.4 Ethernet controller:                                        Virtual Function
4b:00.5 Ethernet controller:                                        Virtual Function
4b:00.6 Ethernet controller:                                        Virtual Function
4b:00.7 Ethernet controller:                                        Virtual Function
4b:01.0 Ethernet controller:                                        Virtual Function
4b:01.1 Ethernet controller:                                        Virtual Function
4b:01.2 Ethernet controller:                                        Virtual Function
4b:01.3 Ethernet controller:                                        Virtual Function
4b:01.4 Ethernet controller:                                        Virtual Function
4b:01.5 Ethernet controller:                                        Virtual Function
4b:01.6 Ethernet controller:                                        Virtual Function
4b:01.7 Ethernet controller:                                        Virtual Function
4b:02.0 Ethernet controller:                                        Virtual Function
4b:02.1 Ethernet controller:                                        Virtual Function
4b:02.2 Ethernet controller:                                        Virtual Function
4b:02.3 Ethernet controller:                                        Virtual Function
4b:02.4 Ethernet controller:                                        Virtual Function
4b:02.5 Ethernet controller:                                        Virtual Function
4b:02.6 Ethernet controller:                                        Virtual Function
4b:02.7 Ethernet controller:                                        Virtual Function
```

Figure 4.25. Listing PCI Devices Bound to VFIO on Host

Once the VFIO VNICs were set up, the hotplug process was initiated. The script for adding VFIO-VNICs to the VM is as follows:

```
[root                              # cat    VFIO_VNIC_Hotplug.sh
#!/bin/bash

bus=0
device=2

for j in {2..  }; do
  # Format the host string
  host=$(printf "0000:4b:%02x.%x" $bus $device)

  echo "device_add vfio-pci,host=${host},id=vfio${j},bus=piciroot${j}" | nc -U /tmp/hmp_vm1

  device=$((device + 1))

  if [ $device -gt 7 ]; then
    device=0
    bus=$((bus + 1))
  fi
done

[root@                             |# sh    VFIO_VNIC_Hotplug.sh
```

Figure 4.26. Executing VFIO VNIC Hotplug Script

This script systematically adds VFIO VNICs to the VM by sending commands to the QEMU monitor to attach each device. Each VNIC was successfully detected by the guest VM, as confirmed by the updated network interface list after the hotplug operation.

```
[root@OL89_VM ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                 brd
    inet             scope host lo
       valid_lft forever preferred_lft forever
    inet6 :       scope host
       valid_lft forever preferred_lft forever
2: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
[root@OL89_VM ~]#
```

Figure 4.27. Network Interface List Before VFIO VNIC Hotplug

```
[root@OL89_VM ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                   brd
    inet             scope host lo
       valid_lft forever preferred_lft forever
    inet6 :       scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 8a:               brd ff:ff:ff:ff:ff:ff
3: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
4: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 6a:               brd ff:ff:ff:ff:ff:ff
5: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 96:               brd ff:ff:ff:ff:ff:ff
6: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether fe:               brd ff:ff:ff:ff:ff:ff
7: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 36:               brd ff:ff:ff:ff:ff:ff
8: enp6s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether fa:               brd ff:ff:ff:ff:ff:ff
9: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether d6:               brd ff:ff:ff:ff:ff:ff
10: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether f2:               brd ff:ff:ff:ff:ff:ff
11: enp9s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 56:               brd ff:ff:ff:ff:ff:ff
```

Figure 4.28. Post-Hotplug Network Interface List in Guest VM

The verification involved checking the network interfaces before and after the hotplug, which showed all VNICs were added successfully. The guest VM maintained stable network operations, and the hotplug persisted after a reboot.

For the unplug process, a different script was used:

```
[root@                              ]# cat    VFIO_VNIC_Unplug.sh
for i in {2..  };do echo -e "device_del vfio${i}\n" | nc -U /tmp/hmp_vm1 ; done
[root@                              ]# sh    VFIO_VNIC_Unplug.sh
```

Figure 4.29. Executing VFIO VNIC Unplug Script

This script removes the VFIO VNICs one by one by sending commands to the QEMU monitor. After unplugging the VNICs, the network interface list was checked to ensure the removal was successful. The guest VM continued to operate normally, and the network configuration was updated accordingly.

```
[root@OL89_VM ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                  brd
    inet            scope host lo
       valid_lft forever preferred_lft forever
    inet6 :       scope host
       valid_lft forever preferred_lft forever
2: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:              brd ff:ff:ff:ff:ff:ff
[root@OL89_VM ~]#
```

Figure 4.30. Updated Network Interface List After VFIO VNIC Unplug

4.1.3.4 VDisks Hotplug/Unplug

For this test, we prepared some virtual disk images on the host system. The creation of these disk images was accomplished with the following commands:

```
[root@                          ]# qemu-img create -f qcow2 img1.qcow2 5G
Formatting 'img1.qcow2', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib size=5368709120 lazy_refcounts=off refcount_bits=16
[root@                          ]# qemu-img create -f qcow2 img2.qcow2 5G
Formatting 'img2.qcow2', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib size=5368709120 lazy_refcounts=off refcount_bits=16
[root@                          ]# qemu-img create -f qcow2 img3.qcow2 5G
Formatting 'img3.qcow2', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib size=5368709120 lazy_refcounts=off refcount_bits=16
```

Figure 4.31. Creating vDisks on Host System

These commands generated three 5GB disk images in QCOW2 format, which were then used for hotplug testing.

The hotplug process involved adding these virtual disks to the VM using the following QEMU monitor commands:

```
QEMU 7.2.0 monitor - type 'help' for more information
(qemu) drive_add 0 if=none,file=/home/opc/img1.qcow2,format=qcow2,id=blk-disk1
drive_add 0 if=none,file=/home/opc/img1.qcow2,format=qcow2,id=blk-disk1
OK
(qemu) device_add virtio-blk-pci,drive=blk-disk1,id=blk-dev1,bus=pciroot10
device_add virtio-blk-pci,drive=blk-disk1,id=blk-dev1,bus=pciroot10
(qemu) drive_add 1 if=none,file=/home/opc/img2.qcow2,format=qcow2,id=blk-disk2
drive_add 1 if=none,file=/home/opc/img2.qcow2,format=qcow2,id=blk-disk2
OK
(qemu) device_add virtio-blk-pci,drive=blk-disk2,id=blk-dev2,bus=pciroot6
device_add virtio-blk-pci,drive=blk-disk2,id=blk-dev2,bus=pciroot6
(qemu) drive_add 2 if=none,file=/home/opc/img3.qcow2,format=qcow2,id=blk-disk3
drive_add 2 if=none,file=/home/opc/img3.qcow2,format=qcow2,id=blk-disk3
OK
(qemu) device_add virtio-blk-pci,drive=blk-disk3,id=blk-dev3,bus=pciroot7
device_add virtio-blk-pci,drive=blk-disk3,id=blk-dev3,bus=pciroot7
(qemu)
```

Figure 4.32. Hotplugging vDisks Command Sequence

Before initiating the hotplug, the lsblk command was used to display the existing block devices:

```
[root@OL89_VM ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                   8:0    0 46.6G  0 disk
├─sda1                8:1    0  100M  0 part /boot/efi
├─sda2                8:2    0    1G  0 part /boot
└─sda3                8:3    0 45.5G  0 part
  ├─ocivolume-root  252:0    0 35.5G  0 lvm  /
  └─ocivolume-oled  252:1    0   10G  0 lvm  /var/oled
[root@OL89_VM ~]#
```

Figure 4.33. Block Device List Before vDisks Hotplug

During the hotplug, new entries appeared in the system logs, indicating that the disks were successfully detected and attached. After the hotplug, running lsblk again showed the new disks:

```
[root@OL89_VM ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                   8:0    0 46.6G  0 disk
├─sda1                8:1    0  100M  0 part /boot/efi
├─sda2                8:2    0    1G  0 part /boot
└─sda3                8:3    0 45.5G  0 part
  ├─ocivolume-root  252:0    0 35.5G  0 lvm  /
  └─ocivolume-oled  252:1    0   10G  0 lvm  /var/oled
vda                 251:0    0    5G  0 disk
vdb                 251:16   0    5G  0 disk
vdc                 251:32   0    5G  0 disk
[root@OL89_VM ~]#
```

Figure 4.34. Block Device List After vDisks Hotplug

The new disks were listed correctly, confirming the success of the hotplug operation.

For the unplug test, the disks were removed using similar QEMU monitor commands. This process was confirmed by checking the lsblk output again, which showed the disks were no longer present, verifying that the unplug operation was successful.

4.1.3.5 Kdump Check

To verify Kdump functionality, we configured the system to capture crash dumps during kernel failures. This involved setting up the `/etc/kdump.conf` file to designate a local dump directory and specifying memory allocation for dump files. We checked the Kdump service status with the command `systemctl status kdump.service` to ensure it was active.

To test the setup, a kernel panic was induced using `echo c > /proc/sysrq-trigger`, causing the system to reboot. Upon restart, we verified that dump files were present in `/var/oled/crash`.

```
[root@OL89_VM ~]# sysctl -w kernel.sysrq=1
kernel.sysrq = 1
[root@OL89_VM ~]# echo c > /proc/sysrq-trigger
```

Figure 4.35. Initiating Kdump Process

```
[root@OL89_VM ~]# ls -lr /var/oled/crash/
total 0
drwxr-xr-x. 2 root root 67 Aug 26 03:50 127.0.0.1-2024-08-26-03:50:06
```

Figure 4.36. Crash Dump Directory Verification

Analysis tools such as crash were used to inspect the dump files, confirming that Kdump was correctly capturing crash data.

4.1.3.6 Memory Hotplug/Unplug

The memory hotplug/unplug test was conducted to assess the VM's capability to dynamically add and remove memory. Initially, we aimed to add 16 GB of memory to the guest VM using QEMU commands. The following QEMU monitor commands were used to add memory:

```
(qemu) object_add memory-backend-ram,id=mem1,size=16G
object_add memory-backend-ram,id=mem1,size=16G
(qemu) device_add pc-dimm,id=dimm1,memdev=mem1
device_add pc-dimm,id=dimm1,memdev=mem1
```

Figure 4.37. Executing Memory Hotplug Process

Before initiating the test, we checked the VM's current memory configuration with the lsmem command:

```
[root@OL89_VM ~]# lsmem
RANGE                                  SIZE  STATE REMOVABLE  BLOCK
0x0000000000000000-0x000000007fffffff   2G online       yes   0-15
0x0000000100000000-0x000000087fffffff  30G online       yes 32-271

Memory block size:       128M
Total online memory:      32G
Total offline memory:      0B
[root@OL89_VM ~]#
```

Figure 4.38. Memory Status Before the Hotplugging

After the addition, the lsmem command showed the following updated memory configuration:

```
[root@OL89_VM ~]# lsmem
RANGE                                  SIZE  STATE REMOVABLE  BLOCK
0x0000000000000000-0x000000007fffffff   2G online       yes   0-15
0x0000000100000000-0x0000000c7fffffff  46G online       yes 32-399

Memory block size:       128M
Total online memory:      48G
Total offline memory:      0B
[root@OL89_VM ~]#
```

Figure 4.39. Memory Status After Hotplugging

After a reboot, the memory configuration remained at 48 GB, confirming that the hotplug operation was successful.

For the memory unplug test, the following QEMU monitor commands were used:

```
(qemu) device_del dimm1
device_del dimm1
(qemu) object_del mem1
object_del mem1
(qemu)
```

Figure 4.40. Executing Memory Unplug Process

Post-unplug, the memory configuration reverted to its initial state:

```
[root@OL89_VM ~]# lsmem
RANGE                                  SIZE  STATE REMOVABLE  BLOCK
0x0000000000000000-0x000000007fffffff   2G online       yes   0-15
0x0000000100000000-0x000000087fffffff  30G online       yes 32-271

Memory block size:       128M
Total online memory:      32G
Total offline memory:      0B
[root@OL89_VM ~]#
```

Figure 4.41. Memory Status After Unplugging

4.1.3.7 Overview of Testing Outcomes

In this section, we summarize the outcomes of the various tests conducted. The tests covered a range of functionalities, including lifecycle operations, hotplug/unplug of network interfaces, virtual disks, and memory. Each test aimed to validate the stability and performance of the VM in handling dynamic changes and typical operational states.

Key findings from the tests include:

- **Lifecycle Operations:** The VM successfully handled reboot, stop/continue, suspend, and shutdown operations without issues. The system remained stable and responsive through each state transition;
- **Hotplug/Unplug of VNICs:** Both standard and VFIO VNICs were tested. The guest VM efficiently managed the addition and removal of VNICs, demonstrating robustness in network interface handling;
- **VDisk Hotplug/Unplug:** The VM successfully attached and detached virtual disks, with the system recognizing new disks and removing them as expected;
- **Memory Hotplug/Unplug:** The VM handled the addition and removal of memory dynamically. The system maintained stability and correctly updated its memory configuration.

Performance metrics and system logs throughout the tests showed that the VM operated within expected parameters, confirming the effectiveness of the tested features and the stability of the virtual environment.

4.2 Test on OL8 Host with an AMD CPU, QEMU and latest kernel version UEK6U3

4.2.1 Host System and Guest VM Configuration

For the second testing scenario, we utilized an AMD-based OCI Instance running Oracle Linux Server 8. This setup involved configuring the latest version of UEK7U2 on the host. The first step was to verify the installation and configuration of QEMU, ensuring that all components were correctly installed and the environment was properly set up.

The initial verification included checking the host system's configuration using the hostnamectl command, listing all relevant QEMU components, verifying the presence of edk2 packages:

```
[root@                          ]# hostnamectl
   Static hostname:
         Icon name: computer-server
           Chassis: server
        Machine ID:
           Boot ID:
  Operating System: Oracle Linux Server 8.10
       CPE OS Name: cpe:/o:oracle:linux:8:10:server
            Kernel: Linux                 .el8uek.x86_64
      Architecture: x86-64
[root@                          ]# yum list installed | grep qemu
qemu-img.x86_64                                                          @custom1
qemu-kvm.x86_64                                                          @custom1
qemu-kvm-block-curl.x86_64                                               @custom1
qemu-kvm-block-gluster.x86_64                                            @custom1
qemu-kvm-block-iscsi.x86_64                                              @custom1
qemu-kvm-block-rbd.x86_64                                                @custom1
qemu-kvm-block-ssh.x86_64                                                @custom1
qemu-kvm-common.x86_64                                                   @custom1
qemu-kvm-core.x86_64                                                     @custom1
qemu-kvm-docs.x86_64                                                     @ol8_appstream
[root@                          ]# yum list installed | grep edk2
edk2-ovmf.noarch                                                         @edk2
edk2-tools.x86_64                                                        @edk2
ipxe-roms-hops.x86_64                                                    @edk2
[root@inaji-ol8-amd-instance opc]#
```

Figure 4.42. Host System Configuration Verification

With the host system fully prepared, we proceeded to deploy the guest VM running Oracle Linux 7.9 with UEK6U3. The guest was initiated using the QEMU command which ensured that the guest VM was correctly set up for the testing phase:

```
[root@                    ilyass]# /usr/libexec/qemu-kvm -boot d -machine q35,kernel_irqchip=split \
> -m 32G,slots=6,maxmem=64G -enable-kvm \
> -cpu host,+host-phys-bits,+topoext -smp cpus=64,cores=32,threads=1,sockets=4,maxcpus=128 \
> -drive file=/dev/shm/test/disk-OL-7.9.qcow2,if=virtio,aio=threads,format=qcow2 \
> -drive file=/usr/share/OVMF/OVMF_CODE.pure-efi.fd,if=pflash,format=raw,unit=0,readonly=on \
> -drive file=/tmp/OVMF_VARS.pure-efi.fd,if=pflash,format=raw,unit=1 \
> -drive file=/dev/shm/test/OracleLinux-R7-U9-Server-x86_64-dvd.iso,media=cdrom \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=2,chassis=2,id=pciroot2,bus=pcie.0,addr=0x2 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=3,chassis=3,id=pciroot3,bus=pcie.0,addr=0x3 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=4,chassis=4,id=pciroot4,bus=pcie.0,addr=0x4 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=5,chassis=5,id=pciroot5,bus=pcie.0,addr=0x5 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=6,chassis=6,id=pciroot6,bus=pcie.0,addr=0x6 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=7,chassis=7,id=pciroot7,bus=pcie.0,addr=0x7 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=8,chassis=8,id=pciroot8,bus=pcie.0,addr=0x8 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=9,chassis=9,id=pciroot9,bus=pcie.0,addr=0x9 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=10,chassis=10,id=pciroot10,bus=pcie.0,addr=0xA \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=11,chassis=11,id=pciroot11,bus=pcie.0,addr=0xB \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=12,chassis=12,id=pciroot12,bus=pcie.0,addr=0xC \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=13,chassis=13,id=pciroot13,bus=pcie.0,addr=0xD \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=14,chassis=14,id=pciroot14,bus=pcie.0,addr=0xE \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=15,chassis=15,id=pciroot15,bus=pcie.0,addr=0xF \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=16,chassis=16,id=pciroot16,bus=pcie.0,addr=0x10 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=17,chassis=17,id=pciroot17,bus=pcie.0,addr=0x11 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=18,chassis=18,id=pciroot18,bus=pcie.0,addr=0x12 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=19,chassis=19,id=pciroot19,bus=pcie.0,addr=0x13 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=20,chassis=20,id=pciroot20,bus=pcie.0,addr=0x14 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=21,chassis=21,id=pciroot21,bus=pcie.0,addr=0x15 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=22,chassis=22,id=pciroot22,bus=pcie.0,addr=0x16 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=23,chassis=23,id=pciroot23,bus=pcie.0,addr=0x17 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=24,chassis=24,id=pciroot24,bus=pcie.0,addr=0x18 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=25,chassis=25,id=pciroot25,bus=pcie.0,addr=0x19 \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=26,chassis=26,id=pciroot26,bus=pcie.0,addr=0x1A \
> -device pcie-root-port,io-reserve=0,pref64-reserve=32M,port=27,chassis=27,id=pciroot27,bus=pcie.0,addr=0x1B \
> -chardev socket,id=mon_vm,path=/tmp/hmp_vm1,server=on,wait=off \
> -mon chardev=mon_vm,mode=readline \
> -qmp unix:/tmp/qmp_vm1,server=on,wait=off \
> -nodefaults \
> -vnc :2 -vga std \
> -netdev user,id=netdev1,hostfwd=tcp::6002-:22 \
> -device virtio-net-pci,id=net1,netdev=netdev1
```

Figure 4.43. QEMU Command for Launching the Guest

4.2.2 Test Execution and Performance Evaluation

Following the setup, a series of tests were conducted to evaluate the stability and performance of the virtual environment. These tests focused on the VM's lifecycle operations, the dynamic management of virtual network interfaces, the kdump check and dynamic add and removal of virtual CPUs.

4.2.2.1 Lifecycle Test

The lifecycle test assessed the VM's ability to manage various operational states, including reboot, stop/continue, suspend, and shutdown. Each state transition was tested to ensure smooth operation and stability:

- **Reboot Test:**

```
QEMU 7.2.0 monitor - type 'help' for more information
(qemu) system_reset
system_reset
(qemu)
```

Figure 4.44. Execution of Reboot Command on the Guest

- **Stop/Continue Test:**

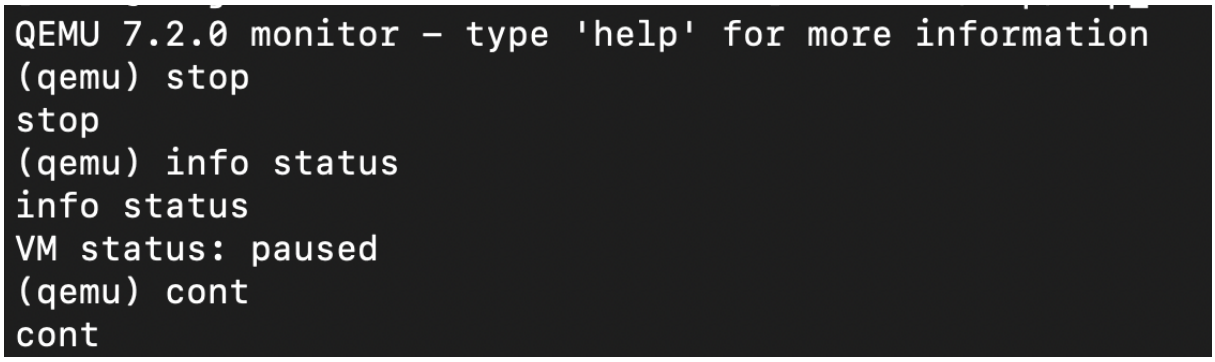


Figure 4.45. Stop and Continue Commands Executed on the Guest

- **Suspend Test:**

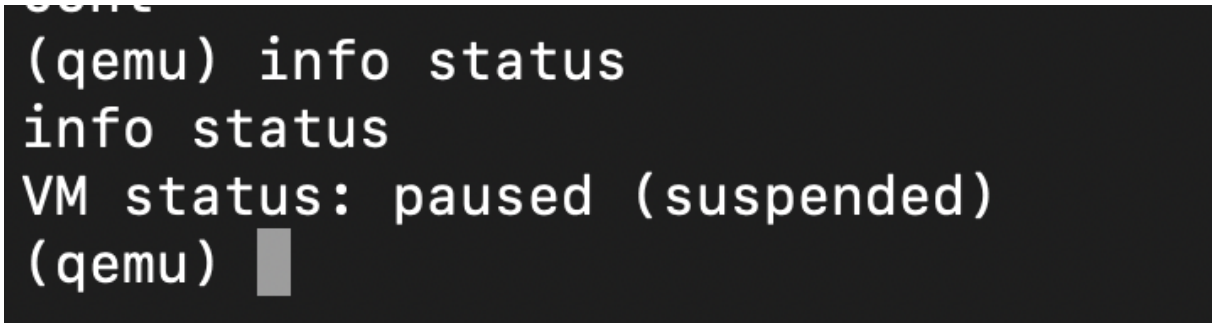


Figure 4.46. Guest Suspension via Systemctl Command

- **Shutdown Test:**

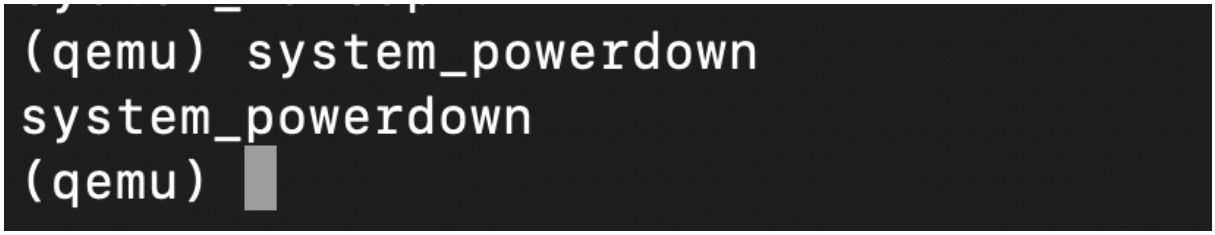


Figure 4.47. Executing Shutdown Test on the Guest

All lifecycle tests were executed successfully, demonstrating the VM's capability to handle various operational states without issues. Each command transitioned smoothly, confirming the stability and reliability of the virtual environment.

4.2.2.2 Some VNIC Hotplug/Unplug

The hotplug and unplug test of VNICs was conducted to simulate high-density networking scenarios. This involved dynamically adding and removing multiple VNICs from the VM.

Before initiating the test, we established a baseline by listing the existing network interfaces using the ip a command. The previous script, VNIC_Hotplug.sh, was used to automate the addition of VNICs.

The addition of VNICs was verified through system logs and updated network configurations. Following the hotplug, we confirmed that all interfaces were successfully added and configured:

```
[root@ol-7-9 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                 brd
    inet             scope host lo
       valid_lft forever preferred_lft forever
    inet6 :       scope host
       valid_lft forever preferred_lft forever
2: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
    inet6 fe80::5054:ff:fe12:3457/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
5: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
6: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
7: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
```

Figure 4.48. Post-Hotplug Network Interface List in the Guest

For the unplug process, the script VNIC_Unplug.sh removed each VNIC systematically. The removal was verified by checking the updated network interface list.

4.2.2.3 Some VFIO-VNIC Hotplug/Unplug

In this test, some VFIO VNICs were managed on the host system. The hotplug process was verified through the QEMU monitor commands (VFIO_VNIC_Hotplug.sh) and confirmed by updated network interface lists in the guest VM:

```
[root@ol-7-9 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                 brd
    inet             scope host lo
       valid_lft forever preferred_lft forever
    inet6 :       scope host
       valid_lft forever preferred_lft forever
2: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
    inet6 fe80::5054:ff:fe12:3457/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
5: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
6: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
7: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:                brd ff:ff:ff:ff:ff:ff
```

Figure 4.49. Network Interface List After VFIO VNIC Hotplug

The unplug process was similarly executed, and the network interface list was checked to confirm successful removal of VFIO VNICs.

4.2.2.4 VDisks Hotplug/Unplug

The vDisk images were prepared on the host and used for the hotplug/unplug test. The disks were created and added to the VM using QEMU commands are located on `/home/opc` path, with verification through the lsblk command before and after the hotplug:

```
[root@ol-7-9 ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                   8:0    0 46.6G  0 disk
├─sda1                8:1    0  100M  0 part /boot/efi
├─sda2                8:2    0    1G  0 part /boot
└─sda3                8:3    0 45.5G  0 part
  ├─ocivolume-root  252:0    0 35.5G  0 lvm  /
  └─ocivolume-oled  252:1    0   10G  0 lvm  /var/oled
vda                 251:0    0    5G  0 disk
vdb                 251:16   0    5G  0 disk
vdc                 251:32   0    5G  0 disk
[root@ol-7-9 ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                   8:0    0 46.6G  0 disk
├─sda1                8:1    0  100M  0 part /boot/efi
├─sda2                8:2    0    1G  0 part /boot
└─sda3                8:3    0 45.5G  0 part
  ├─ocivolume-root  252:0    0 35.5G  0 lvm  /
  └─ocivolume-oled  252:1    0   10G  0 lvm  /var/oled
[root@ol-7-9 ~]#
```

Figure 4.50. Updated Block Device List Post-Hotplug and Unplug

4.2.2.5 Kdump Check

To verify the Kdump functionality, we configured the system to capture crash dumps in the event of kernel failures:

```
[root@ol-7-9 ~]# sysctl -w kernel.sysrq=1
kernel.sysrq = 1
[root@ol-7-9 ~]# echo c > /proc/sysrq-trigger
```

Figure 4.51. Initiating Kdump Process

```
[root@ol-7-9 ~]# ls -l /var/oled/crash/
total 0
drwxr-xr-x. 2 root root 67 Aug 27 05:43 127.0.0.1-2024-08-27-05:43:40
[root@ol-7-9 ~]#
```

Figure 4.52. Crash Dump Directory Verification

The Kdump functionality was further validated by analyzing the crash dump files, confirming that the crash data was accurately captured and processed.

4.2.2.6 VCPUs Hotplug/Unplug

The vCPU hotplug test involved dynamically adding and removing virtual CPUs from the guest VM to evaluate its capability to handle changes in CPU resources.

Initially, the VM was configured with a certain number of CPUs. To perform the hotplug test, we used the following QEMU monitor commands to add three additional vCPUs:

```
QEMU 7.2.0 monitor - type 'help' for more information
(qemu) device_add host-x86_64-cpu,socket-id=2,core-id=0,thread-id=0,apic-id=64,id=cpu64
device_add host-x86_64-cpu,socket-id=2,core-id=0,thread-id=0,apic-id=64,id=cpu64
(qemu) device_add host-x86_64-cpu,socket-id=2,core-id=1,thread-id=0,apic-id=65,id=cpu65
device_add host-x86_64-cpu,socket-id=2,core-id=1,thread-id=0,apic-id=65,id=cpu65
(qemu) device_add host-x86_64-cpu,socket-id=2,core-id=2,thread-id=0,apic-id=66,id=cpu66
device_add host-x86_64-cpu,socket-id=2,core-id=2,thread-id=0,apic-id=66,id=cpu66
(qemu)
```

Figure 4.53. Executing vCPU Hotplug Process

Before the hotplug operation and after it, the VM's CPU configuration was as follows:

```
[root@ol-7-9 ~]# lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              64
On-line CPU(s) list: 0-63
Thread(s) per core:  1
Core(s) per socket:  32
Socket(s):           2
NUMA node(s):        1
Vendor ID:           GenuineIntel
BIOS Vendor ID:      QEMU
```

Figure 4.54. CPU Configuration Before Hotplug

```
[root@ol-7-9 ~]# lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              67
On-line CPU(s) list: 0-66
Thread(s) per core:  1
Core(s) per socket:  22
Socket(s):           3
NUMA node(s):        1
Vendor ID:           GenuineIntel
BIOS Vendor ID:      QEMU
```

Figure 4.55. CPU Configuration After Hotplug

Following the vCPU addition, we tested the vCPU unplug process using similar QEMU monitor commands to remove the added CPUs. The CPU configuration reverted to its original state, this confirmed the successful dynamic adjustment of vCPUs in the VM.

4.2.2.7 Overview of Testing Outcomes

The testing conducted for QEMU on AMD host with OL8 + UEK7U2 and OL 7.9 + UEK6U3 guest revealed several key insights:

- **Lifecycle Operations:** The guest VM managed various operational states, including reboot, stop/continue, suspend, and shutdown, effectively and without issues. Each state transition was smooth, demonstrating robust lifecycle management capabilities;
- **Hotplug/Unplug of VNICs:** The system successfully handled the dynamic addition and removal of virtual network interfaces (VNICs), with all interfaces being correctly recognized and configured. This indicates strong support for high-density networking scenarios;
- **VFIO-VNIC Hotplug/Unplug:** The VFIO VNICs were managed successfully, with the system effectively adding and removing the VFIO VNICs. The network interface list was updated as expected, showing the VM's capability to handle VFIO network interfaces;
- **VDisk Hotplug/Unplug:** The test demonstrated that the VM could efficiently attach and detach virtual disks, with the system accurately reflecting the changes in disk availability and configuration;
- **Kdump Functionality:** The Kdump test validated that the system could capture and store crash dumps correctly, with dump files being successfully created and analyzed following a kernel panic;
- **VCPU Hotplug/Unplug:** The dynamic addition and removal of vCPUs were executed successfully, with the VM reflecting changes in CPU configuration as expected. This tested the VM's capability to handle varying CPU resources effectively.

Overall, the tests confirmed that the virtual environment operated within the expected parameters, demonstrating stability, reliability, and proper functionality of key virtualization features.

4.3 Test on OL8 Host with and ARM CPU, Libvirt, QEMU and Latest kernel version UEK7U2

4.3.1 Host System and Guest VM Configuration

For this testing phase, an ARM-based OCI instance was utilized, running Oracle Linux Server 8. The host system was equipped with the latest UEK7U2 kernel, and the configuration process commenced with the installation of QEMU and Libvirt. Ensuring the correct installation and environment setup was paramount to the success of the tests.

The initial verification procedures involved confirming the host system's details via the `hostnamectl` command, followed by a thorough check of all QEMU and Libvirt components:

```
[root@                          ]# hostnamectl
   Static hostname:
         Icon name: computer-server
           Chassis: server
        Machine ID:
           Boot ID:
  Operating System: Oracle Linux Server 8.10
       CPE OS Name: cpe:/o:oracle:linux:8:10:server
            Kernel: Linux                 .el8uek.aarch64
      Architecture: arm64
[root@                          ]# yum info qemu-kvm
Last metadata expiration check: 2:24:37 ago on Tue 27 Aug 2024 03:32:50 AM GMT.
Installed Packages
Name         : qemu-kvm
Epoch        :
Version      :
Release      :
Architecture : aarch64
Size         : 0.0
Source       : qemu-kvm-                                        .src.rpm
Repository   : @System
From repo    : custom1
Summary      : QEMU is a machine emulator and virtualizer
URL          : http://www.qemu.org/
License      : GPLv2+ and LGPLv2+ and BSD
Description  : qemu-kvm is an open source virtualizer that provides hardware
             : emulation for the KVM hypervisor. qemu-kvm acts as a virtual
             : machine monitor together with the KVM kernel modules, and emulates the
             : hardware for a full system such as a PC and its associated peripherals.

[root@                          ]# yum info libvirt
Last metadata expiration check: 2:24:47 ago on Tue 27 Aug 2024 03:32:50 AM GMT.
Installed Packages
Name         : libvirt
Version      :
Release      :
Architecture : aarch64
Size         : 0.0
Source       : libvirt-                                       .src.rpm
Repository   : @System
From repo    : custom1
Summary      : Library providing a simple virtualization API
URL          : https://libvirt.org/
License      : LGPLv2+
```

Figure 4.56. Verification of Host System Configuration

With the host system confirmed to be in optimal condition, the next step was deploying the guest VM, which ran Oracle Linux 8.10 with UEK7U2. The VM was launched using a Libvirt script, ensuring proper initialization for the upcoming testing phase:

```
[root@                    ilyass]# cat create.bash
#!/bin/bash

# Defining variables
virsh='virsh --connect qemu:///system'
virt_install='virt-install --connect qemu:///system'
vm_name=vm_ol810

# Creating the VM
virt-install \
  --name $vm_name \
  --virt-type kvm \
  --boot loader=/usr/share/AAVMF/AAVMF_CODE.pure-efi.fd,loader_ro=yes,loader_type=pflash,nvram_template=/usr/share/AAVMF/AAVMF_VARS.pure-efi.fd,loader
_secure=no \
  --memory=8192 \
  --vcpu=2 \
  --disk /dev/shm/ilyass/OracleLinux-8.10-2024.06.30-0-uefi-aarch64.qcow2,bus=usb,size=25 \
  --network network=default \
  --graphics vnc,listen=0.0.0.0 \
  --noautoconsole -v
[root@                    ilyass]# bash create.bash
```

Figure 4.57. Deployment Script for the Guest via Libvirt

4.3.2 Test Execution and Performance Evaluation

Following the setup, various tests were executed to evaluate the stability and performance of the virtual environment. These tests focused on the VM's lifecycle management, the dynamic handling of VNICs, and kdump testing, and the booting of large VMs.

4.3.2.1 Lifecycle Test

The lifecycle test was designed to assess the VM's capability to handle different operational states, including start, reboot, suspend, resume, reset and shutdown. Each transition was meticulously tested to ensure seamless operation and system stability:

- **Start Test:**

```
[root@                    ilyass]# virsh start vm_ol810
setlocale: No such file or directory
Domain 'vm_ol810' started

[root@                    ilyass]# virsh list
setlocale: No such file or directory
 Id   Name       State
--------------------------
 2    vm_ol810   running
```

Figure 4.58. Execution of Start Command on the Guest

- **Reboot Test:**

```
[root@                      ilyass]# virsh reboot vm_ol810
setlocale: No such file or directory
Domain 'vm_ol810' is being rebooted

[root@                      ilyass]# virsh list
setlocale: No such file or directory
 Id   Name       State
--------------------------
 1    vm_ol810   running
```

Figure 4.59. Execution of Reboot Command on the Guest

- **Suspend Test:**

```
[root@                     ilyass]# virsh suspend vm_ol810
setlocale: No such file or directory
Domain 'vm_ol810' suspended

[root@                     ilyass]# virsh list
setlocale: No such file or directory
 Id   Name       State
-----------------------
 1    vm_ol810   paused
```

Figure 4.60. Suspension of the Guest Using Libvirt

- **Resume Test:**

```
[root@                    ilyass]# virsh resume vm_ol810
setlocale: No such file or directory
Domain 'vm_ol810' resumed

[root@                    ilyass]# virsh list
setlocale: No such file or directory
 Id   Name       State
--------------------------
 1    vm_ol810   running
```

Figure 4.61. Resuming the Guest After Suspension

- **Reset Test:**

```
[root@                      ilyass]# virsh reset vm_ol810
setlocale: No such file or directory
Domain 'vm_ol810' was reset

[root@                      ilyass]# virsh list
setlocale: No such file or directory
 Id   Name       State
--------------------------
 1    vm_ol810   running
```

Figure 4.62. Reset Command Execution on the Guest

- **Shutdown Test:**

```
[root@                       ilyass]# virsh list --all
setlocale: No such file or directory
 Id   Name       State
---------------------------
 -    vm_ol810   shut off
```

Figure 4.63. Guest Shutdown Initiated via Libvirt

The lifecycle tests were successfully completed, with each command executing flawlessly, confirming the VM's robust lifecycle management capabilities.

4.3.2.2 Some VNIC Hotplug/Unplug

A script was employed to automate the hotplugging of some VNICs using Libvirt:

```
[root@                    ilyass]# cat    VNIC_hotplug.bash
#!/bin/bash
for num in {1..  }
do
        virsh attach-interface vm_ol810 --type network --source default --model virtio --mac ac:          :$num --live
done
[root@                    ilyass]# bash    VNIC_hotplug.bash
```

Figure 4.64. Libvirt Script for VNIC Hotplug

The successful addition of the VNICs was verified by examining the system logs and updated network configurations. Post-hotplugging, all interfaces were confirmed to be properly recognized and configured:

```
[root@ol-8-10 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                   brd
    inet             scope host lo
       valid_lft forever preferred_lft forever
    inet6         scope host
       valid_lft forever preferred_lft forever
2: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
    inet6 fe80::5054:ff:fe12:3457/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
5: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
6: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
7: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
8: enp6s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
9: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
10: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
11: enp9s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
12: enp10s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
```

Figure 4.65. Post-Hotplug Verification of Network Interfaces

For the unplugging process, a script was executed to systematically remove each VNIC. The removal was validated by verifying the updated network interface list which contains just two.

4.3.2.3 Some VFIO-VNIC Hotplug/Unplug

This test focused on managing VFIO VNICs on the host system. The hotplugging process was carried out using Libvirt scripts, with the following steps verified:

```
[root@                      ilyass]# cat write_vfio_file.bash
#!/bin/bash

function_values=1
slot_value=0

for i in {1..  }
do
        filename="vfio-${i}.xml"

        if ((function_values > 7)); then
                ((function_values=0))
                ((slot_value+=1))
        fi

        echo "<interface type='hostdev'>
                <source>
                        <address type='pci' domain='0' bus='2' slot='${slot_value}' function='${function_values}'/>
                </source>
        </interface>" > "/tmp/${filename}"

        echo $function_values
        (( function_values += 1 ))
done
[root@                      ilyass]# bash write_vfio_file.bash
```

Figure 4.66. Populating VFIO Files via Script

```
[root@                              ]# cat plug_vm.bash
#!/bin/bash

function_values=1
nbr=0

for i in {1..  }
do
        if ((function_values > 7)); then
                ((function_values=0))
                ((nbr+=1))
        fi

        echo virsh nodedev-detach pci_0000_02_0$((nbr))_$((function_values))

        ((function_values+=1))
done

for i in {1..  }; do
        virsh attach-device vm_ol810 /tmp/vfio-$i.xml --live
done
[root@                              ]# bash plug_vm.bash
```

Figure 4.67. Hotplugging VFIO VNIC Using Libvirt

```
[root@ol-8-10 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback                  brd
    inet             scope host lo
       valid_lft forever preferred_lft forever
    inet6 :       scope host
       valid_lft forever preferred_lft forever
2: enp0s28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:               brd ff:ff:ff:ff:ff:ff
3: enp6s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0a:               brd ff:ff:ff:ff:ff:ff
4: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether ae:               brd ff:ff:ff:ff:ff:ff
    inet6 fe80:                        scope link noprefixroute
       valid_lft forever preferred_lft forever
5: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 9e:               brd ff:ff:ff:ff:ff:ff
6: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether d6:               brd ff:ff:ff:ff:ff:ff
7: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 2a:               brd ff:ff:ff:ff:ff:ff
8: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 66:               brd ff:ff:ff:ff:ff:ff
9: enp9s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether de:               brd ff:ff:ff:ff:ff:ff
10: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 5e:               brd ff:ff:ff:ff:ff:ff
11: enp10s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 4a:               brd ff:ff:ff:ff:ff:ff
12: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
```

Figure 4.68. Updated Network Interface List After VFIO Hotplug

The hotplugging operation was successfully completed, with the network interface list reflecting the newly added VFIO VNICs on the guest VM. The unplugging process was executed similarly, and the network interface list confirmed the successful removal of the VFIO VNICs.

4.3.2.4 VDisks Hotplug/Unplug

Some vDisks were created on the host in the `/dev/shm/ilyass/images/` directory for the hotplug/unplug tests. The disks were added to the VM using Libvirt commands, and the operation was verified using the `lsblk` command before and after the hotplug:

```
[root@ol-7-9 ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                   8:0    0 46.6G  0 disk
├─sda1                8:1    0  100M  0 part /boot/efi
├─sda2                8:2    0    1G  0 part /boot
└─sda3                8:3    0 45.5G  0 part
  ├─ocivolume-root 252:0    0 35.5G  0 lvm  /
  └─ocivolume-oled 252:1    0   10G  0 lvm  /var/oled
vda                 251:0    0    5G  0 disk
vdb                 251:16   0    5G  0 disk
vdc                 251:32   0    5G  0 disk
[root@ol-7-9 ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                   8:0    0 46.6G  0 disk
├─sda1                8:1    0  100M  0 part /boot/efi
├─sda2                8:2    0    1G  0 part /boot
└─sda3                8:3    0 45.5G  0 part
  ├─ocivolume-root 252:0    0 35.5G  0 lvm  /
  └─ocivolume-oled 252:1    0   10G  0 lvm  /var/oled
[root@ol-7-9 ~]#
```

Figure 4.69. Verifying Block Device List After Hotplug and Unplug

4.3.2.5 Kdump Check

To test Kdump functionality, the system was configured to capture crash dumps during kernel failures:

```
[root@ol-8-10 ~]# sysctl -w kernel.sysrq=1
kernel.sysrq = 1
[root@ol-8-10 ~]# echo c > /proc/sysrq-trigger
```

Figure 4.70. Executing Kdump Process on the Guest

```
Password:
Last login: Tue Aug 27 06:31:55 on ttyS0
[root@ol-8-10 ~]# ls -lr /var/oled/crash/
total 0
drwxr-xr-x. 2 root root 67 Aug 27 06:35 127.0.0.1-2024-08-27-06:35:50
```

Figure 4.71. Crash Dump Files Stored in Kdump Directory

The Kdump functionality was confirmed by analyzing the captured crash dump files, which verified that the crash data was accurately recorded and processed.

4.3.2.6 Big VM 500G-1T Boot Test

A VM with a configuration of 500G GB RAM and then 1000G GB RAM was deployed to test the environment's capability to handle large VM instances. The VM was successfully booted, and the expected resource allocation was verified via the `lsmem` command:

```
Last login: Tue Aug 27 13:57:37 2024
[root@ol-8-10 ~]# lsmem
RANGE                                  SIZE  STATE REMOVABLE  BLOCK
0x0000000040000000-0x0000007d3fffffff  500G online        no 8-4007

Memory block size:       128M
Total online memory:     500G
Total offline memory:      0B
[root@ol-8-10 ~]#
```

Figure 4.72. Resource Allocation Verification in Large VM (500G RAM)

```
Last login: Tue Aug 27 14:01:49 2024
[root@ol-8-10 ~]# lsmem
RANGE                                 SIZE  STATE REMOVABLE  BLOCK
0x0000000040000000-0x000000fa3fffffff 1000G online        no 8-8007

Memory block size:       128M
Total online memory:    1000G
Total offline memory:      0B
[root@ol-8-10 ~]#
```

Figure 4.73. Resource Allocation Verification in Large VM (1T RAM)

The successful boot of the large VM confirmed that the environment was capable of handling resource-intensive virtual machines without issues.

4.3.2.7 Overview of Testing Outcomes

In this testing phase, we evaluated the performance and stability of the virtual environment on an ARM-based OCI instance running Oracle Linux Server 8 with UEK7U2. The key aspects of this testing involved lifecycle management, virtual network interface handling, virtual disk operations, kdump functionality, and the booting of large VMs. The outcomes of these tests are summarized below:

- **Lifecycle Operations:** The VM demonstrated robust lifecycle management by successfully executing start, reboot, suspend, resume, reset, and shutdown operations. Each state transition was smooth, indicating that the environment can handle various operational states without disruptions;
- **Hotplug/Unplug of VNICs:** The VM efficiently managed the dynamic addition and removal of virtual network interfaces using Libvirt. All network interfaces were correctly recognized and configured, demonstrating the system's ability to support high-density networking requirements;

- **Hotplug/Unplug of VFIO VNICs:** The test confirmed that the VM could effectively manage VFIO VNICs. The successful hotplugging and unplugging operations reflected the environment's capability to handle VFIO-based network interfaces, which are critical for direct device access in virtualized environments;
- **VDisks Hotplug/Unplug:** The VM successfully attached and detached the virtual disks. The system accurately reflected the changes in disk availability, confirming the environment's ability to manage dynamic storage configurations effectively;
- **Kdump Functionality:** The Kdump test validated the system's ability to capture and store crash dumps. The dump files were correctly generated and analyzed, demonstrating the reliability of the crash recovery process;
- **Large VM Boot Test (500G-1T RAM):** The successful boot of a VM with 500G and then 1000G RAM confirmed the environment's capacity to handle large, resource-intensive virtual machines. The test validated the system's scalability and its ability to allocate significant resources without issues.

Overall, the testing confirmed that the virtual environment is stable, reliable, and capable of handling a wide range of demanding virtualization scenarios. The system operated within expected parameters, and all tested features performed as anticipated.

4.4 Conclusion

In conclusion, this chapter detailed the implementation and testing of QEMU with various configurations. It highlighted the technical choices made, the execution of tests, and the validation results, demonstrating the system's functionality and readiness for practical use.

General Conclusion & Perspectives

This report presented the results of my internship project, which was focused on validating Oracle Linux virtualization modules. The main objective was to ensure the compatibility, stability, and reliability of key virtualization components, such as QEMU and Libvirt, across various configurations and software releases. Throughout the project, I undertook a comprehensive series of tests, including manual sanity testing and regression analysis, to achieve these objectives.

This project provided me with a valuable opportunity to apply the theoretical knowledge I gained during my studies at National Institute of Posts and Telecommunications, while also allowing me to explore and work with advanced virtualization technologies in a real-world environment. My time at Oracle has been an enriching experience, contributing to my technical expertise, problem-solving abilities, and teamwork skills. The hands-on experience I gained in documenting and analyzing test results, coupled with the challenges of adapting to new technologies, has greatly enhanced my understanding of virtualization and cloud computing.

One of the most significant aspects of this project was navigating the complex interactions between different software components and virtualization layers. Addressing the challenges associated with system updates and ensuring that new releases do not introduce regressions was particularly demanding. These experiences have further developed my ability to approach technical problems systematically and with a focus on continuous improvement.

Looking ahead, several perspectives could guide the future development and enhancement of Oracle Linux virtualization. Continued efforts could be directed towards further refining testing methodologies, particularly in the area of automation, to increase the efficiency and coverage of regression tests. Additionally, exploring the integration of more advanced security measures, such as enhanced encryption protocols and better isolation techniques, could further improve the robustness of the virtualization environment. By pursuing these avenues, Oracle Linux virtualization can continue to evolve into a more reliable and secure platform, meeting the growing demands of enterprise customers and ensuring seamless operation within the Oracle Cloud ecosystem.

Bibliography

[1] The history of oracle corporation. https://www.oracle.com/.

[2] Oracle logo. https://labs.oracle.com/.

[3] Outlook logo. https://www.atlassian.com/software/confluence.

[4] Slack logo. https://slack.com/blog/news/say-hello-new-logo.

[5] Zoom logo. https://www.atlassian.com/software/confluence.

[6] Confluence logo. https://www.atlassian.com/software/confluence.

[7] General schema of virtualization. https://documentation.suse.com/smart/virtualization-cloud/pdf/virtualization_en.pdf.

[8] Oracle linux logo. https://images.app.goo.gl/5op3952h6PbJbnvV8.

[9] Libvirt overview. https://documentation.suse.com/smart/virtualization-cloud/html/manage-vm-on-commandline/.