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

Agentic AI for Legal Automation: A Modular Multi-Agent Platform for Contract Management

Realized by:

Naji Ilyass

Defended on June xx, 2025, in front of the jury composed of:

Mrs. El Akkaoui Zineb	INPT	- Supervisor
Mr. Hamad Ahmed	INPT	- Examiner
Mr. Hamad Ahmed	INPT	- Examiner

National Telecommunications Regulatory Agency
National Institute of Posts and Telecommunications

Class of 2024/2025

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 provides a comprehensive overview of the context surrounding the internship project. Divided into three major sections, it begins by presenting the host organization, Boston Consulting Group (BCG), specifically within its specialized technology and innovation unit, BCG X. This introduction includes insights into the organization's history, areas of expertise, business activities, organizational structure, and cultural attributes. The second section delves into the project framework, focusing on the use of Agentic AI within legal operations, the importance of digital contract management, the identified problems in existing legal workflows, and the specific objectives aimed at resolving these challenges. Finally, the third section describes the project management approach adopted during the internship, highlighting the Agile methodologies, collaborative tools, communication practices, and detailed sprint-based planning used to successfully manage and deliver the project.

1.1 Host Organization

My internship was hosted by Boston Consulting Group (BCG), specifically within its technology and innovation unit, BCG X. This section introduces both entities, providing insights into their history and areas of expertise.

1.1.1 Presentation

In today's rapidly evolving business environment, organizations across sectors are increasingly reliant on technology-driven solutions to enhance efficiency, innovation, and competitive advantage. Companies are leveraging advanced technologies such as Artificial Intelligence (AI), cloud computing, and digital transformation platforms to address complex operational challenges and unlock new growth opportunities.

Boston Consulting Group (BCG), a globally recognized management consulting firm, combines deep industry expertise and analytical rigor to help its clients navigate these challenges. Through its specialized tech build and design unit, BCG X, the firm integrates consulting excellence with advanced technological capabilities, delivering impactful and scalable digital solutions that transform business models and operational processes.

The following sections provide a comprehensive overview of Boston Consulting Group and BCG X, highlighting their history, business areas of activity, mission, and organizational culture.

1.1.2 Boston Consulting Group (BCG)

BCG BOSTON CONSULTING GROUP

Figure 1.1. BCG Logo
[1]

1.1.2.1 History

Boston Consulting Group was founded in 1963 by Bruce Henderson, pioneering many of the strategic concepts that shape modern management consulting today. Over the decades, BCG has consistently been at the forefront of business strategy, driving innovation and delivering transformative results for clients worldwide. Headquartered in Boston, Massachusetts, BCG operates globally with offices in more than 90 cities across over 50 countries. The firm has continuously evolved its offerings, focusing increasingly on integrating technology and digital solutions into traditional consulting services to meet the dynamic needs of modern businesses.

1.1.2.2 Business area of activity

BCG provides a wide array of strategic and operational consulting services, including business transformation, corporate strategy, digital transformation, sustainability, mergers and acquisitions, innovation management, and organizational effectiveness. The firm's extensive industry expertise covers sectors such as healthcare, financial services, consumer goods, energy, technology, and public sector.

In recent years, BCG has significantly expanded its digital and technology capabilities through its dedicated unit, BCG X, emphasizing digital solutions, advanced analytics, artificial intelligence, and large-scale digital platform developments to support clients' transformational journeys.

1.1.3 BCG X

Figure 1.2. BCG X Logo
[?]

1.1.3.1 Presentation

BCG X is the technology build and design unit of Boston Consulting Group, established to accelerate the digital transformation journeys of large organizations by integrating high-impact technology solutions with strategic business insight. BCG X acts as a force multiplier, enhancing BCG's traditional consulting services through state-of-the-art digital and technological solutions.

1.1.3.2 Business area of activity

BCG X specializes in the following core capabilities:

- **AI and Generative AI (GenAI)**
  Developing advanced AI solutions that streamline core business functions and enhance customer engagement through predictive analytics, automation, and intelligent decision-making.

- **Customer Experience for Growth**
  Designing and implementing digital strategies and customer experiences to drive business growth and foster deeper client engagement.

- **Venture and Business Builds**
  Launching and scaling disruptive new ventures and business models, providing end-to-end support from ideation to market entry.

- **Digital Platform Builds**
  Creating scalable, secure digital platforms and infrastructures that support extensive digital transformations and enable efficient integration of diverse technological solutions.

Through these activities, BCG X significantly contributes to transforming and modernizing businesses across industries, positioning them effectively within competitive and rapidly changing markets.

1.1.4 Organizational Structure

Boston Consulting Group (BCG) adopts a structured yet agile approach, promoting global collaboration through its tech unit, BCG X. Led by Sylvain Duranton (Global Leader) and Jon Ferris (Chief Financial Officer), BCG X's organizational structure includes regional and functional leads who oversee specialized areas and regional centers. This design ensures efficient global coordination and responsiveness to client needs.

Each regional center—including Casablanca, Berlin, Shanghai, Shenzhen, Singapore, and India—is managed by dedicated leaders who implement localized yet globally aligned strategies. This structured regional management facilitates effective service delivery tailored to local market requirements while maintaining consistent global standards.

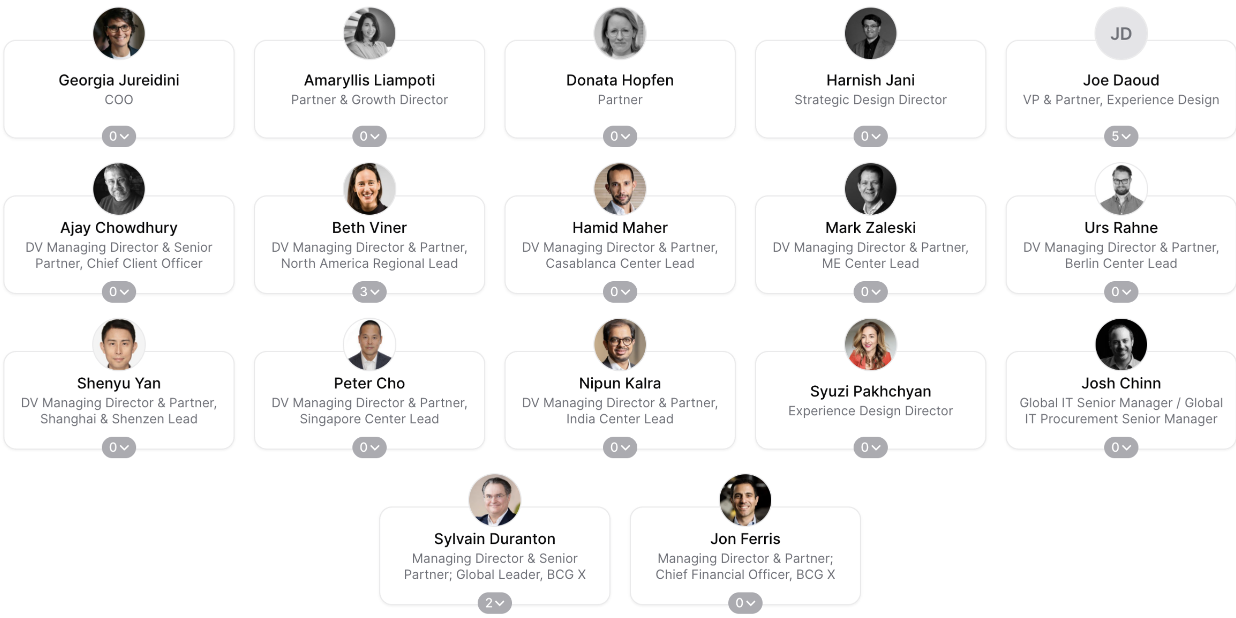


Figure 1.3. Executive Leadership Board
[2]

BCG X's organizational design promotes effective decision-making, seamless collaboration across global teams, and responsiveness to client needs, leveraging both hierarchical and regional structures.

1.1.5 Organizational Culture

BCG X fosters a dynamic organizational culture defined by innovation, inclusivity, and entrepreneurial spirit. Its culture encourages team members to embrace challenges, take risks, and collaborate openly to drive transformative solutions for clients.

Key aspects of BCG X's organizational culture include:

- **Innovation and Entrepreneurship**
  Innovation is central to BCG X's culture, which consistently emphasizes entrepreneurial

mindsets, encouraging individuals to explore new ideas, technologies, and methodologies. Employees are supported in developing digital solutions that significantly impact businesses and industries.

- **Inclusivity and Employee Well-being**
  BCG X prioritizes a culture of inclusivity and well-being, providing a supportive environment that values diverse perspectives and encourages personal and professional growth. The organization actively supports employees through wellness programs, career coaching, tuition reimbursement, and structured promotion pathways.

- **Open Communication and Collaboration**
  Transparent and open communication is encouraged within the organization, fostering a collaborative environment where ideas can flow freely. Employees are empowered to communicate openly, enabling swift decision-making and fostering strong internal relationships and teamwork.

- **Ambition and Growth Orientation**
  The organizational culture at BCG X highly values ambitious, growth-oriented individuals who are comfortable navigating ambiguity and pushing boundaries. This ambition is reflected in rapid innovation cycles, strategic client engagements, and a constant drive to achieve impactful outcomes.

Overall, BCG X's organizational culture underpins its ability to deliver high-impact digital transformation and innovation, driven by motivated, collaborative, and empowered teams.

1.2 Project Framework

1.2.1 Agentic AI in Legal Operations

Agentic AI represents an innovative shift from traditional imperative systems toward autonomous decision-making processes in various business functions. It involves specialized AI agents that are designed to autonomously determine actions based on high-level goals, dynamically adapting to new data and operational conditions. In legal operations specifically, Agentic AI significantly improves flexibility, scalability, and efficiency by automating complex workflows traditionally reliant on extensive manual intervention.

Agentic AI platforms in legal applications are equipped with capabilities such as automated clause selection, smart contract data mapping, continuous monitoring, and compliance verification. These autonomous capabilities streamline the traditionally manual processes involved in contract drafting, validation, and compliance monitoring, significantly reducing contract processing time and increasing accuracy.

1.2.2 Digital Contract Management

The process of contract management traditionally involves manual drafting, iterative feedback loops, and approval workflows that often lead to inefficiencies, errors, and delays. Digital Contract Management (DCM) solutions have emerged to address these issues, aiming to centralize, automate, and optimize the entire lifecycle of contract operations—from initial drafting to final execution and continuous monitoring.

DCM employs sophisticated AI-driven agents capable of real-time collaboration, intelligent clause insertion, and automated verification. This approach enhances operational efficiency, reduces manual effort in repetitive tasks, and improves compliance by automatically tracking regulatory changes and adapting contracts accordingly. Digital contract management thus transforms legal operations into agile, scalable, and error-resistant processes, significantly boosting organizational productivity and risk management.

1.2.3 Need for Intelligent Legal Workflow Systems

Traditional legal workflows suffer from manual bottlenecks, fragmented processes, and compliance risks due to frequent internal and external regulatory changes. The manual approach limits scalability and visibility, causing significant inefficiencies in managing contractual obligations and deadlines.

Implementing Intelligent Legal Workflow Systems, powered by Agentic AI, addresses these pain points by automating repetitive tasks, providing real-time monitoring of contract lifecycles, and offering predictive insights into compliance and risk management. These systems ensure seamless collaboration between various stakeholders, streamline approval processes, and mitigate risks through automated compliance checks. The result is a robust, scalable framework capable of handling complex legal workflows with improved transparency and accountability.

1.2.4 Problematic

Current contract management and legal operations rely heavily on manual inputs, iterative feedback loops, and scattered stakeholder communications, leading to inefficiencies, compliance risks, and delays. Contract drafting and validation processes are especially impacted, often resulting in lengthy negotiation cycles, limited visibility into contract deadlines, and difficulty in ensuring consistent regulatory compliance across different jurisdictions. The lack of automated systems further exacerbates these issues, causing increased operational costs and elevated risk profiles.

1.2.5 Objectives

This project aims to deploy and evaluate the effectiveness of an AI-driven intelligent contract management platform leveraging Agentic AI to transform traditional legal operations into streamlined, scalable, and automated processes.

The specific objectives include:

- Automating repetitive and manual contract drafting tasks through AI pre-filling and intelligent clause selection.
- Establishing real-time, AI-driven monitoring and compliance validation of contracts.
- Enhancing collaboration and efficiency across multiple stakeholders via automated workflows and digital signatures.

- Reducing overall contract processing time and manual effort significantly (40-60% reduction target).
- Minimizing compliance risks through automated regulatory updates and monitoring, aiming for a 30% risk reduction.

Achieving these goals will provide a standardized, scalable solution capable of significantly enhancing efficiency, compliance, and risk management across legal operations.

1.3 Project Management

The project management approach employed during my internship was meticulously structured to ensure smooth execution and timely delivery. Our methodology was grounded in Agile principles, specifically utilizing the Scrum framework to manage the iterative development and deployment of the intelligent contract management platform.

1.3.1 Scrum Methodology

Scrum is an Agile project management framework designed to facilitate iterative progress through collaboration, flexibility, and responsiveness to feedback. This methodology was particularly suitable for our project due to the dynamic and innovative nature of AI-driven platform development.

1.3.1.1 Scrum Roles and Responsibilities

The Scrum framework clearly defines the roles and responsibilities within the team, enhancing clarity and collaboration:

- **Product Owner**: Defined the project vision, managed the product backlog, and prioritized tasks ensuring alignment with client expectations and project goals.
- **Scrum Master (Project Lead)**: Facilitated daily standups, sprint planning, reviews, and retrospectives; ensured adherence to Agile principles; and resolved any impediments affecting progress.
- **Development Team**: Included software engineers, UX designers, product specialists, and interns who executed the technical, product, and design tasks outlined in each sprint.

1.3.2 Project Execution

The project's lifecycle was divided into iterative phases (Scoping, Alpha, Beta, Minimum Lovable Product), each strategically managed through one-to-two-week sprints. Each sprint encompassed phases of research, design, implementation, testing, and deployment.

1.3.2.1 Sprint Structure

The typical sprint structure adopted for the project was as follows:

- **Sprint Planning**: Tasks were clearly defined and prioritized based on their impact and feasibility.
- **Daily Standups**: Conducted daily for 15 minutes, these sessions updated progress, identified blockers, and adjusted tasks as needed.
- **Weekly Demos**: One-hour weekly meetings where implemented features were demonstrated, allowing stakeholders to review progress, provide feedback, and realign future tasks accordingly.
- **Sprint Review and Retrospective**: Held at the end of each sprint to evaluate completed work, discuss improvements, and refine upcoming tasks.

1.3.3 Collaboration Tools

Our team utilized various collaborative and productivity-enhancing tools, significantly streamlining communication, documentation, and project management:

- **ClickUp**: Functioned as the central hub for managing diverse project activities including sprint planning, task assignment, documentation storage, issue tracking, and progress monitoring. It facilitated comprehensive documentation through organized folders for engineering, product, and design documentation, as well as centralized spaces for meeting notes and general project insights.
- **Slack**: Facilitated instant communication and collaboration among team members, enabling quick discussions and efficient problem-solving.
- **Zoom and Microsoft Teams**: Used extensively for virtual meetings, sprint demos, and daily standups, providing robust audio-visual communication capabilities.
- **Microsoft Outlook**: Managed meetings, reminders, and task scheduling, ensuring structured daily and weekly routines.
- **Miro**: Supported visual collaboration, particularly useful during ideation, design sessions, and feedback workshops.

1.3.4 Meetings and Communication

Regular meetings and structured communication were crucial to maintaining project momentum and addressing challenges promptly:

- **Steering Committee (SteerCo)**: Held every two to three weeks, these strategic meetings, involving leadership from BCG and the client, focused on validating use cases, reviewing project approaches, and aligning on vision and roadmap.
- **Weekly Demos**: Facilitated regular feedback loops with stakeholders to ensure alignment and adapt quickly to feedback and evolving requirements.
- **Ad-Hoc Sessions**: Dedicated weekly time to address emergent challenges, conduct user testing, validate product iterations, and ensure continuous alignment with legal and regulatory frameworks.

1.3.5 Project Timeline and Milestones

The project followed a clearly structured roadmap divided into four primary phases, each critical for incremental delivery and validation:

- **Scoping & Tech Foundations (Weeks 1-3)**: Defined objectives, system design, and initial architecture.
- **Alpha Build (Weeks 3-9)**: Developed the first operational version, enabling preliminary testing and stakeholder feedback.
- **Beta Build (Weeks 9-15)**: Refined the product based on real-world usage, addressing user feedback and enhancing system reliability.
- **Minimum Lovable Product (Weeks 15-21)**: Delivered an enterprise-grade, production-ready platform.

This structured and iterative management approach, supported by Agile Scrum methodologies and robust tool usage, enabled our team to effectively respond to changing requirements, maintain alignment with project objectives, and deliver a comprehensive, innovative solution tailored to enhancing efficiency and compliance in legal operations.

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

Analysis and Specifications

This chapter focuses on the detailed analysis and technical specifications of the intelligent contract management solution developed during my internship. It begins with a comparative benchmarking of existing solutions, providing justification for our technology selection. Subsequently, it delves into the technical background, describing the key technologies used including Large Language Models (LLM), Agentic AI, LangChain, LangGraph, and others. Finally, the chapter specifies the project requirements, identifying functional and non-functional expectations that shaped the solution's development.

2.1 Comparative Analysis

This section benchmarks existing contract management and legal automation platforms to highlight their features, strengths, and limitations, providing the rationale for developing our specialized Agentic AI solution.

2.1.1 Benchmarking Existing Solutions

To evaluate the landscape of intelligent contract management platforms, we analyzed three prominent solutions: PandaDoc, Harvey AI, and DocDraft. Each offers unique features catering to different aspects of contract management and legal automation.

2.1.1.1 PandaDoc

PandaDoc is a comprehensive document automation platform designed to streamline the creation, approval, and management of digital documents, including contracts, proposals, and quotes.

Figure 2.1. PandaDoc Logo
[?]

Here are the key features of PandaDoc:

- **Dynamic Contract Templates**: Create and manage reusable templates to ensure consistency and efficiency in document creation.
- **Integrated eSignatures**: Facilitate secure and legally binding electronic signatures within documents.
- **CRM Integrations**: Seamlessly integrate with platforms like Salesforce and HubSpot to synchronize data and streamline workflows.
- **Contract Repository**: Centralized storage for contracts, enabling easy access, tracking, and management.
- **Workflow Automation**: Automate approval processes and notifications to reduce manual intervention and accelerate deal closures.

2.1.1.2 Harvey AI

Harvey AI is a generative AI platform designed for legal professionals, enabling advanced legal research, document drafting, and compliance analysis with the support of large language models.

Figure 2.2. Harvey AI Logo
[?]

Here are the key features of Harvey AI:

- **AI-Powered Legal Research**: Provides fast, accurate legal research across jurisdictions with references and citations.
- **Document Drafting Automation**: Automates the generation and review of legal documents, reducing time and manual effort.
- **Predictive Analytics**: Leverages historical legal data to forecast case outcomes and support strategic decisions.
- **Secure Document Management**: Ensures safe handling and storage of sensitive legal documents.
- **Customizable AI Models**: Allows tailoring of legal AI agents to specific practice areas, workflows, or jurisdictions.

2.1.1.3 DocDraft

DocDraft is an AI-powered contract drafting solution focused on automating the generation, review, and management of legal documents for legal teams and firms.

Figure 2.3. DocDraft Logo
[?]

Here are the key features of DocDraft:

- **Automated Document Drafting**: Quickly generates contracts and legal documents using intelligent pre-built logic.
- **Clause Library**: Offers a rich repository of standard and customizable clauses for efficient contract assembly.
- **Compliance Checks**: Reviews documents to ensure alignment with relevant laws and regulations.
- **Collaboration Tools**: Enables multiple stakeholders to co-author and comment on documents in real-time.
- **Attorney Matching Service**: Connects users to legal experts for document review and legal consultation.

2.1.1.4 Comparative Analysis

The following table summarizes the key features of the three platforms:

Feature	**PandaDoc**	**Harvey AI**	**DocDraft**
Document Automation	Yes	Yes	Yes
eSignature Integration	Yes	No	No
Legal Research Capabilities	No	Yes	No
Predictive Analytics	No	Yes	No
Compliance Checks	Limited	Advanced	Advanced
CRM Integration	Yes	No	No
Collaboration Tools	Yes	Limited	Yes
Customizable AI Models	No	Yes	No
Clause Library	Limited	No	Yes
Attorney Consultation	No	No	Yes

Table 2.1: Comparison of Contract Management and Legal AI Platforms

2.1.2 Why We Chose to Build a New Solution

The comparative study clearly revealed that existing contract management platforms were not sufficient to meet our needs. Most tools offer limited automation capabilities and lack the flexibility required for complex legal workflows. Additionally, their inability to support deep integration with enterprise systems, limited orchestration features, and lack of control over agent behavior made them unsuitable for large-scale deployment.

To overcome these limitations, we chose to develop a custom solution based on agentic AI. This approach allows us to build a modular, scalable system capable of automating contract drafting, compliance checks, and collaboration, all within a secure and integrated enterprise environment. The goal was not only to improve current processes, but also to lay a solid foundation for future legal automation use cases.

A clear comparison between traditional (imperative) and agentic (declarative) approaches is illustrated in Figure 2.4, highlighting the benefits that guided our architectural direction.

Aspect	**Imperative Approach** (Traditional)	**Declarative Approach** (Agentic AI)
Definition	Step-by-step instructions to achieve a goal.	Specifies the goal and constraints, allowing AI agents to figure out the best steps.
Control	Explicitly defines every action and decision logic.	AI autonomously determines actions based on high-level goals.
Flexibility	Rigid and requires manual updates for changes.	Adaptive, dynamically adjusting based on new data and conditions.
Example	Writing explicit loops and conditionals to handle a process.	Using AI agents that decide when and how to execute tasks.
Scalability	Limited by human-defined logic; harder to extend.	Easily scales as AI agents can generalize solutions across tasks.
Error Handling	Requires explicit error handling in the code.	AI can self-correct and learn from past interactions.
Human Involvement	Heavy involvement in coding and debugging logic.	Minimal intervention; humans define objectives, and AI executes.
Efficiency	Time-consuming, as every scenario must be manually handled.	AI optimizes execution and improves efficiency over time.
Use Case Example	A chatbot that follows predefined decision trees.	An autonomous AI assistant that adapts to user queries and context dynamically.

Figure 2.4. Imperative vs. Agentic Approach Comparison
[?]

2.2 Technical Background

This section provides an overview of key technologies enabling the platform's intelligent functionalities.

2.2.1 Large Language Models (LLMs)

2.2.1.1 Introduction

Language serves as the cornerstone of human interaction, significantly impacting how humans interface with machines. The evolution of Natural Language Processing (NLP), from early statistical approaches to sophisticated neural architectures, has culminated in the advent of Large Language Models (LLMs). These advancements have been driven by the introduction of transformer-based architectures, substantial improvements in computational resources, and vast amounts of training data. The resulting models exhibit near-human performance in numerous linguistic tasks such as translation, summarization, and conversational dialogues.

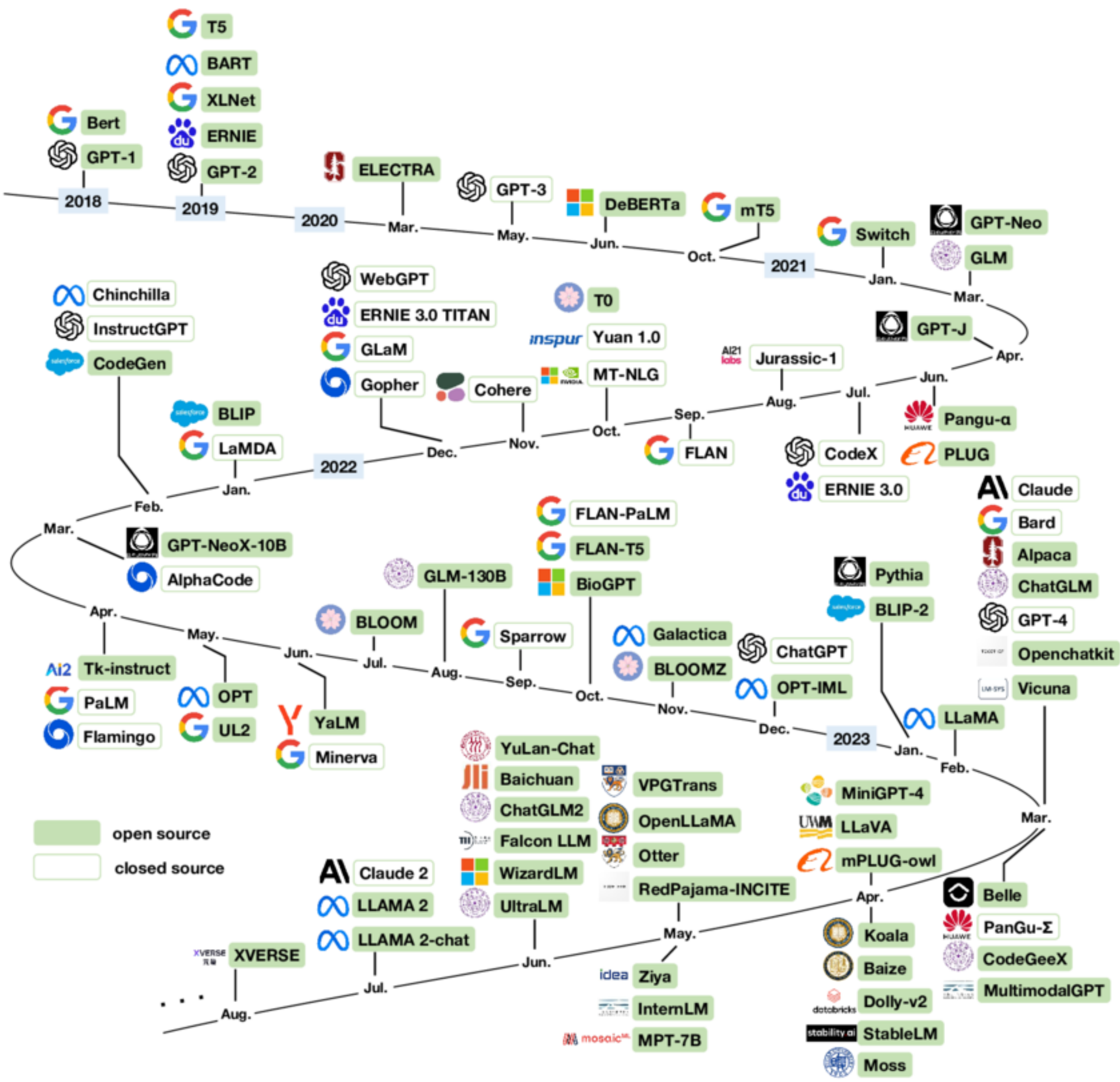


Figure 2.5. Chronological Evolution of Large Language Models (LLMs)
[?]

2.2.1.2 Capabilities and Emergent Properties

Large Language Models have significantly evolved, exhibiting capabilities such as reasoning, decision-making, and in-context learning, often emerging spontaneously at scale rather than through explicit programming or training. These emergent properties enable LLMs to effectively generalize across diverse tasks without extensive task-specific data, highlighting their robust adaptability. Techniques such as fine-tuning and prompt engineering further enhance model alignment, ensuring outputs align closely with user intents and expectations.

LLMs' adaptability and versatility have found applications in various sectors including healthcare, finance, legal operations, and customer service, underscoring their transformative potential across industries.

2.2.1.3 Challenges and Limitations

Despite these advancements, LLM deployment is constrained by several critical limitations. Key challenges include:

- **Computational Cost and Efficiency**: The resource-intensive nature of LLM training and inference demands considerable computational power, resulting in high operational costs.
- **Input Size Constraints**: Current LLM architectures typically handle context windows of up to 128k tokens, limiting their ability to directly process large-scale data sources comprising millions of tokens.
- **Inference Speed and Latency**: Longer inputs proportionally increase inference time and computational demands, challenging real-time or large-scale applications.
- **Ethical and Bias Considerations**: Models may inadvertently propagate biases present in training data or generate misleading and potentially harmful outputs, necessitating robust ethical oversight.

Figures 2.8, 2.7, and 2.6 summarize comparative metrics across popular LLMs, highlighting trade-offs in performance, cost, and processing speed.

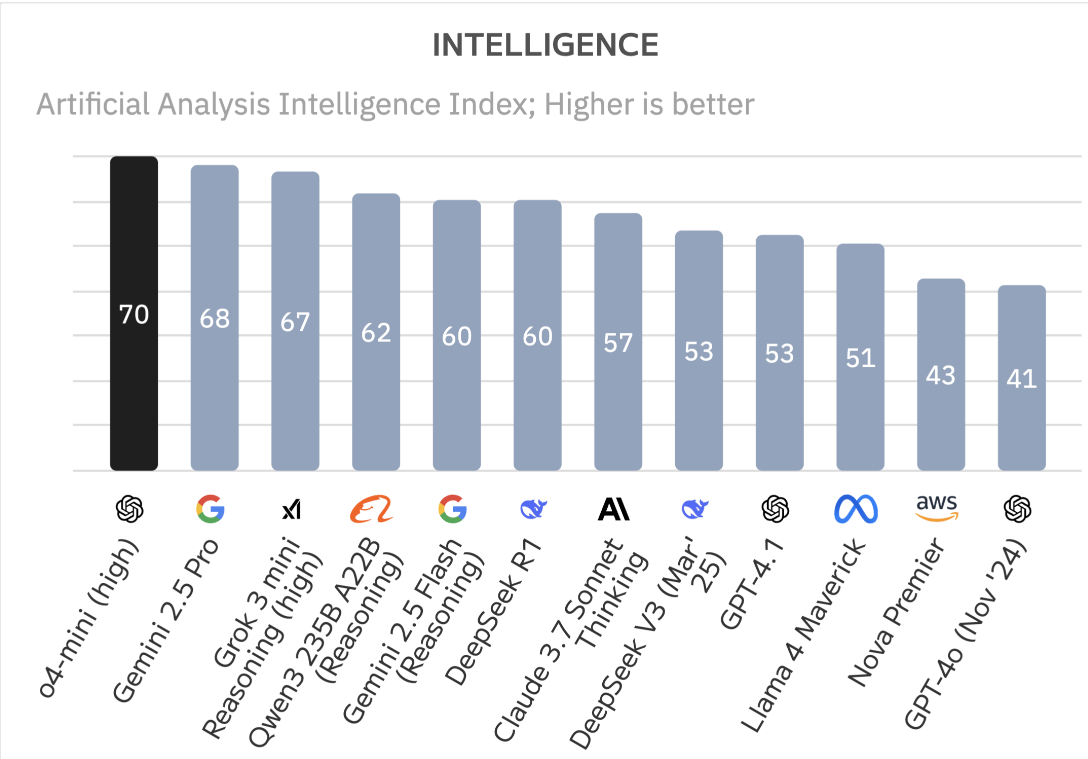


Figure 2.6. Comparative Intelligence Index of LLMs
[?]

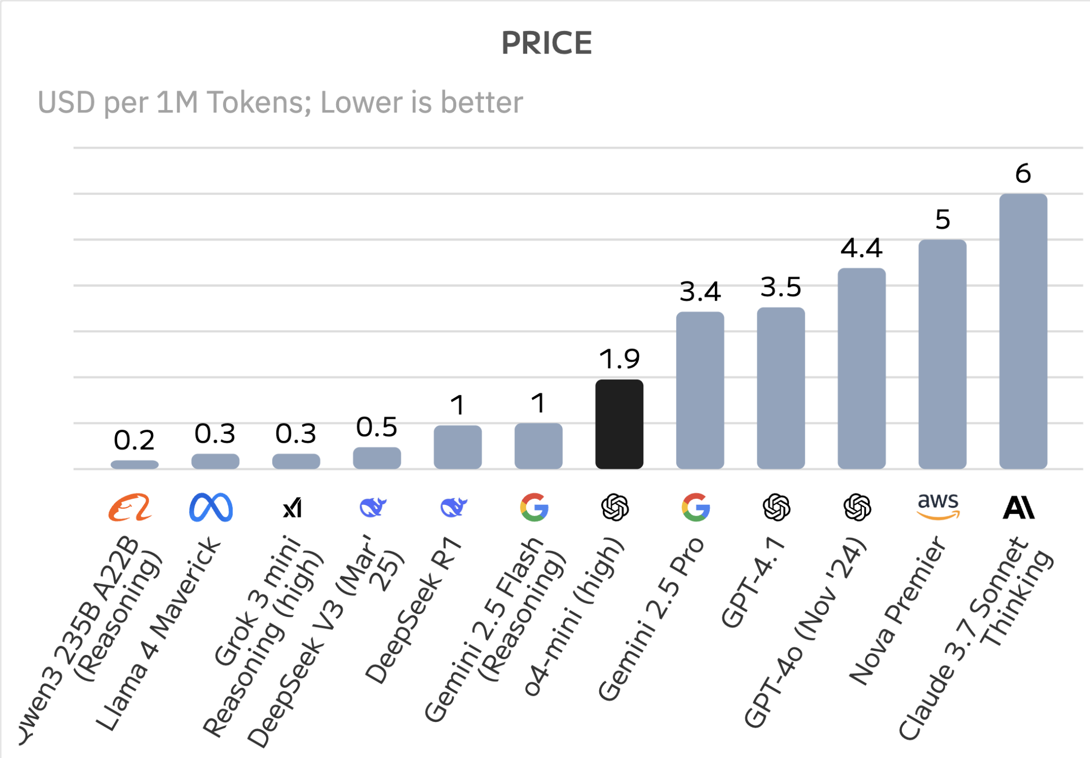


Figure 2.7. Cost Comparison per Million Tokens
[?]

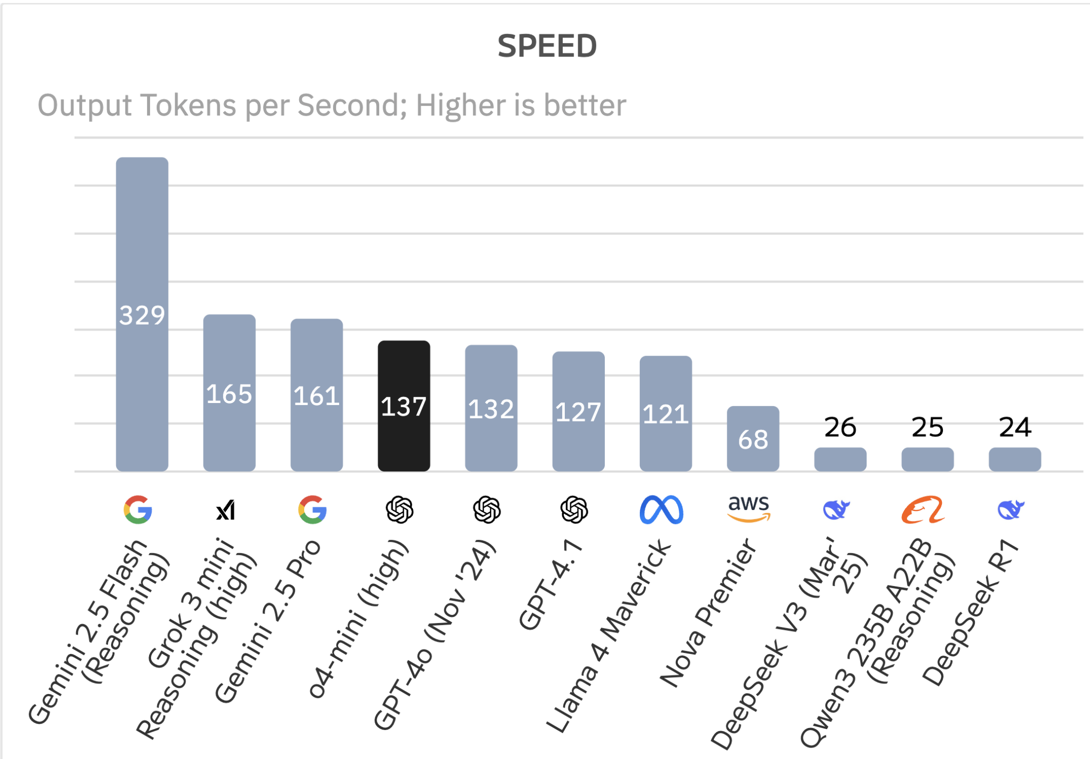


Figure 2.8. Inference Speed Comparison of Various LLMs
[?]

2.2.1.4 Future Directions

Addressing these constraints has become an active area of research. Innovations in model compression techniques, such as pruning, quantization, and distillation, aim to enhance efficiency and accessibility of LLMs. Moreover, emerging architectures and retrieval-augmented generation methods offer promising approaches for handling extensive datasets and large input contexts effectively.

Future advancements are expected to mitigate current limitations, further unlocking the full potential of Large Language Models across diverse, complex applications.

2.2.2 Agentic AI

2.2.2.1 Introduction

Agentic AI represents a significant evolution in artificial intelligence, transitioning from reactive systems to proactive, autonomous entities capable of setting goals, making decisions, and adapting to dynamic environments. Unlike traditional AI agents that operate within predefined parameters, Agentic AI systems exhibit autonomy, learning, and adaptability, enabling them to handle complex, multi-step tasks with minimal human intervention.

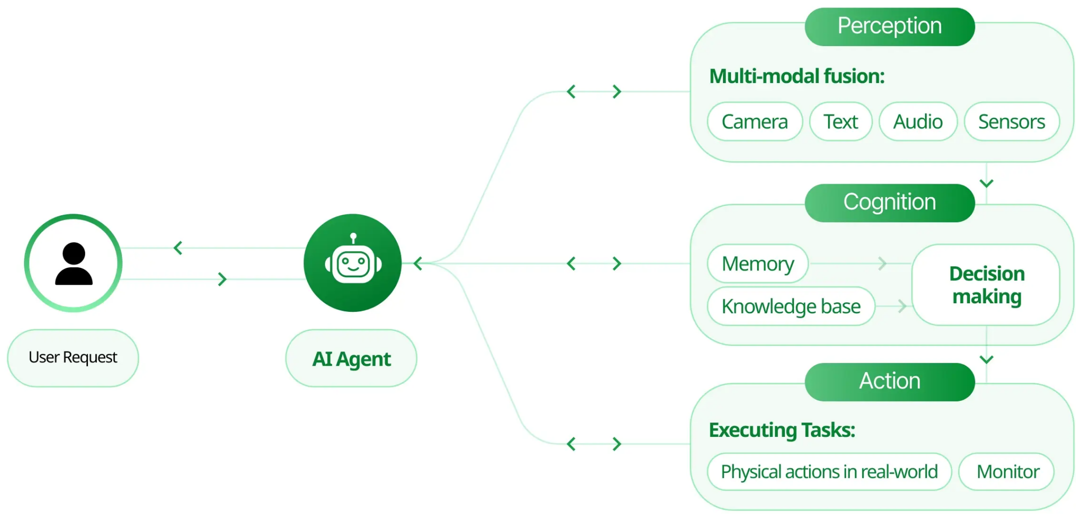


Figure 2.9. Agentic AI System Architecture
[?]

2.2.2.2 Core Characteristics

Agentic AI systems are distinguished by several key features:

- **Autonomy**: They operate independently, making decisions without continuous human oversight.
- **Goal-Oriented Behavior**: These systems can set, pursue, and adjust goals based on environmental feedback.
- **Adaptability**: Agentic AI learns from experiences, refining its strategies to improve performance over time.
- **Complex Decision-Making**: They evaluate multiple options and potential outcomes to make informed decisions.

- **Collaboration**: Agentic AI can coordinate with other agents or systems to achieve shared objectives.

2.2.2.3 Agentic AI vs. Generative AI

While both Agentic AI and Generative AI leverage advanced machine learning techniques, their functionalities differ significantly:

- **Generative AI**: Focuses on creating content (text, images, etc.) based on input data, operating primarily in a reactive manner.
- **Agentic AI**: Emphasizes autonomous decision-making and goal pursuit, enabling proactive interactions with the environment.

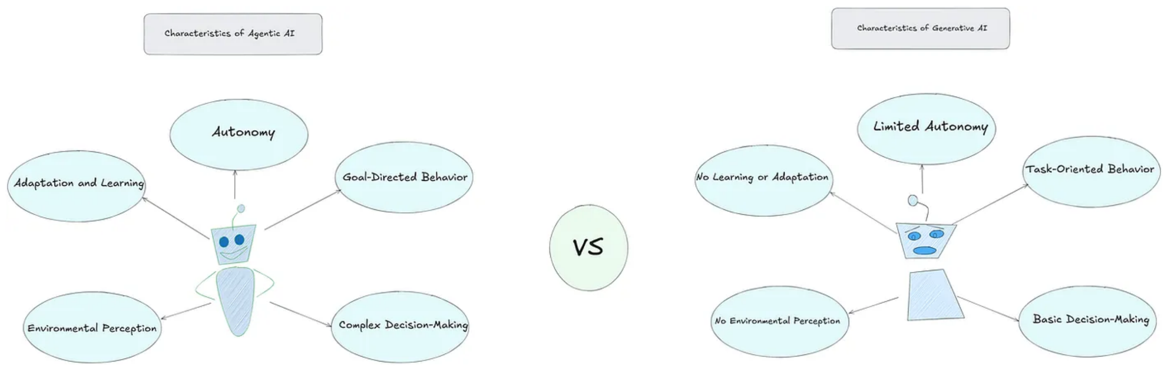


Figure 2.10. Comparison Between Agentic AI and Generative AI
[?]

2.2.2.4 Multi-Agent Systems

Agentic AI often operates within multi-agent systems, where multiple autonomous agents collaborate to solve complex problems. These systems benefit from:

- **Specialization**: Agents can focus on specific tasks, enhancing efficiency.
- **Scalability**: Systems can be expanded by adding more agents to handle increased complexity.
- **Robustness**: Collaboration among agents can compensate for individual agent failures or limitations.

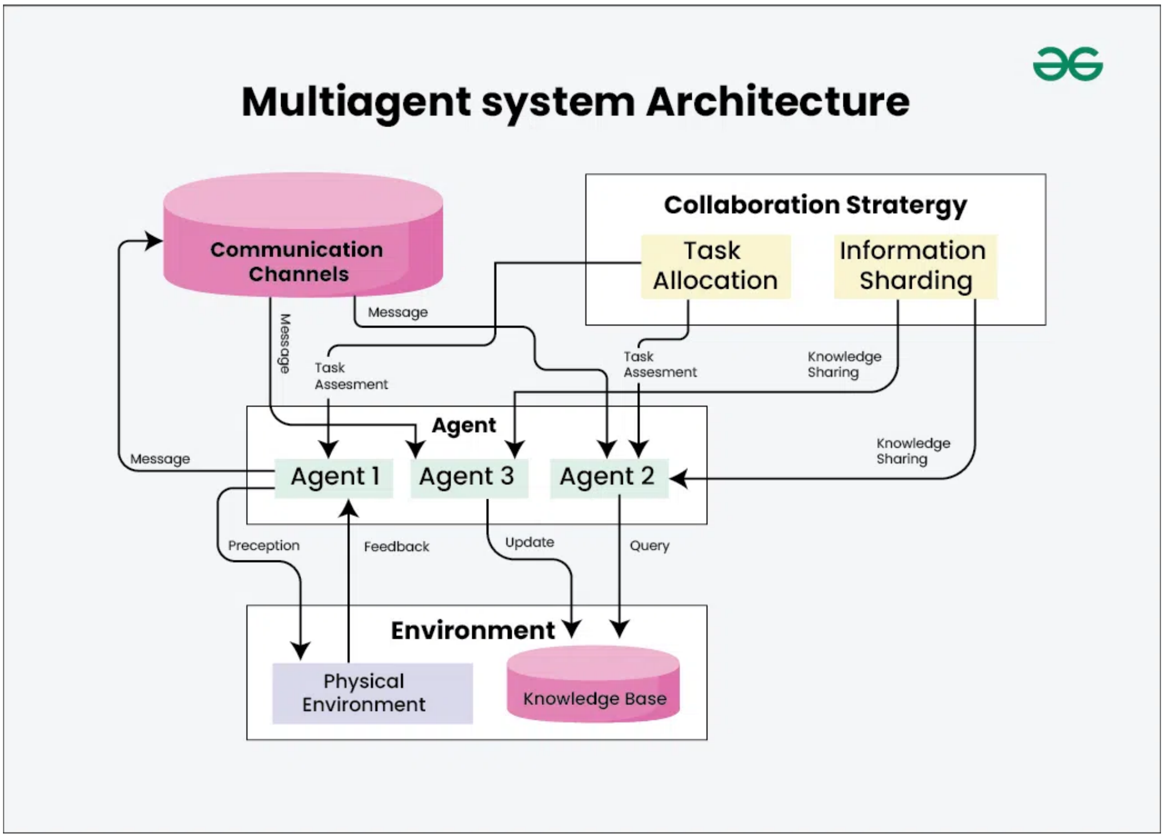


Figure 2.11. Multi-Agent System Collaboration
[?]

2.2.2.5 Applications of Agentic AI

Agentic AI has diverse applications across various industries:

- **Healthcare**: Assisting in diagnostics, treatment planning, and patient monitoring.
- **Finance**: Managing portfolios, detecting fraud, and optimizing trading strategies.
- **Manufacturing**: Overseeing production lines, predicting maintenance needs, and ensuring quality control.
- **Transportation**: Enabling autonomous vehicles to navigate and make real-time decisions.
- **Customer Service**: Providing personalized support through adaptive virtual assistants.

2.2.2.6 Challenges and Future Directions

Despite its potential, Agentic AI faces several challenges:

- **Ethical Considerations**: Ensuring decisions align with human values and societal norms.
- **Transparency**: Making decision-making processes understandable to users.
- **Security**: Protecting systems from malicious manipulation or unintended behaviors.

- **Integration**: Seamlessly incorporating Agentic AI into existing infrastructures.

Ongoing research aims to address these challenges, focusing on developing frameworks for ethical AI, enhancing interpretability, and establishing standards for safe deployment.

2.2.3 Model Context Protocol (MCP)

2.2.3.1 Introduction

The Model Context Protocol (MCP) is an open standard developed by Anthropic in late 2024 to address the growing complexity of integrating large language models (LLMs) with diverse external tools, systems, and data sources. MCP provides a standardized interface that enables AI applications to interact seamlessly with various resources, facilitating context exchange and enhancing the capabilities of AI assistants.

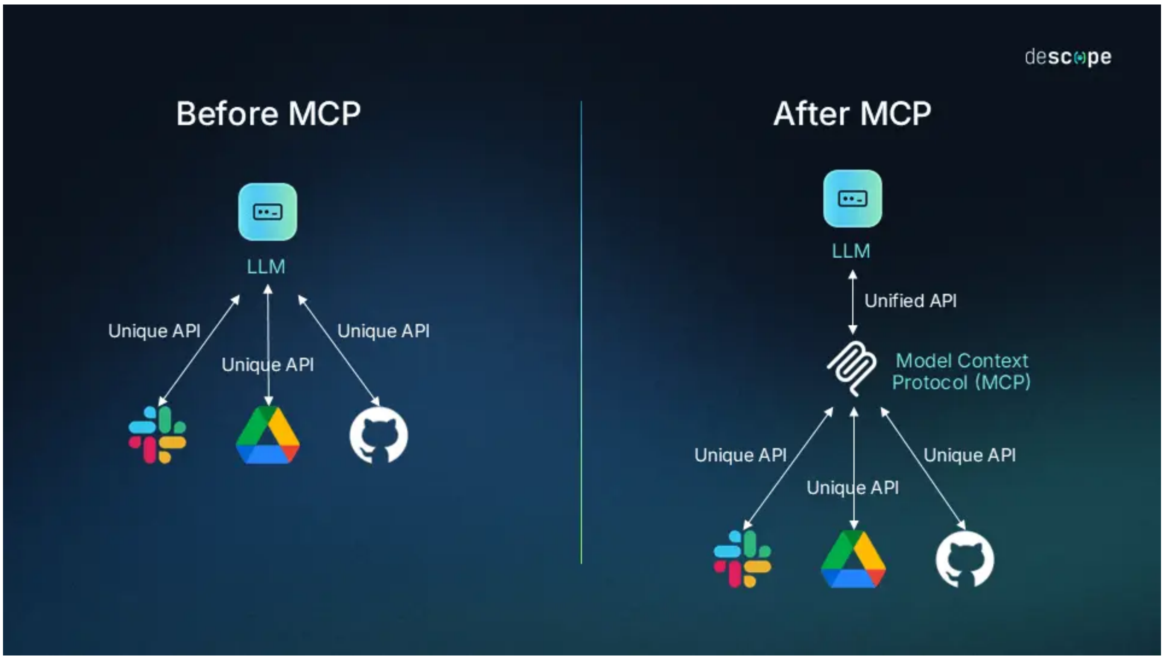


Figure 2.12. Before and After Model Context Protocol (MCP)
[?]

2.2.3.2 Architecture

MCP follows a client-host-server architecture, where:

- **Hosts** are LLM applications (e.g., Claude Desktop, IDEs) that initiate connections.
- **Clients** maintain 1:1 connections with servers within the host application.
- **Servers** provide context, tools, and prompts to clients.

This architecture allows for modular integration, enabling AI systems to access and utilize external resources efficiently.

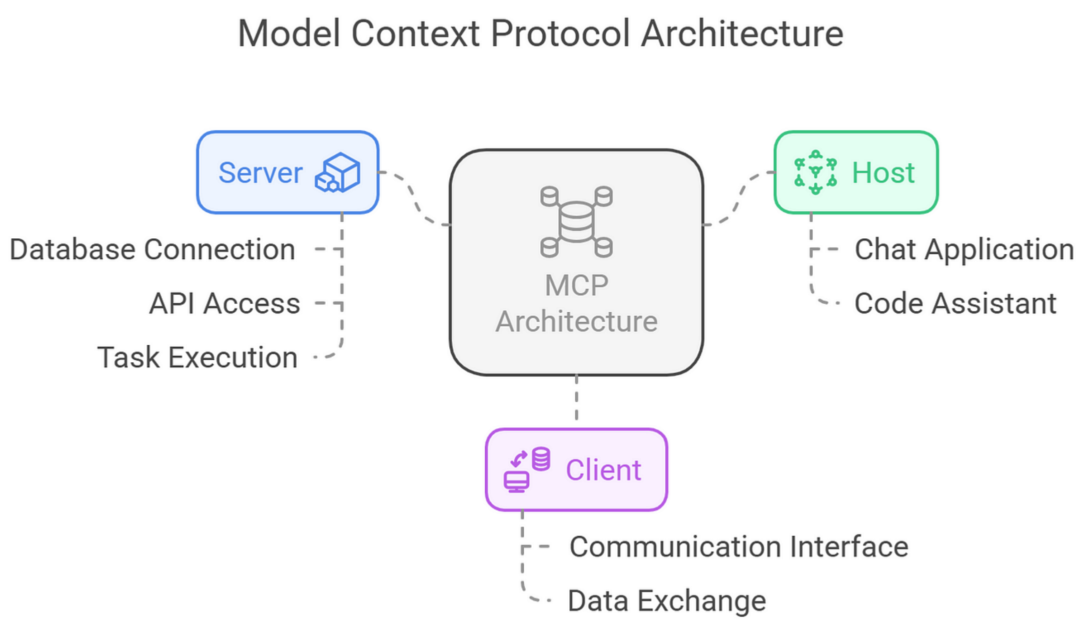


Figure 2.13. Model Context Protocol Architecture
[?]

2.2.3.3 Core Components

MCP servers can expose three main types of capabilities:

- **Resources**: File-like data that can be read by clients (e.g., API responses, file contents).
- **Tools**: Functions that can be called by the LLM with user approval.
- **Prompts**: Pre-written templates that assist users in accomplishing specific tasks.

These components enable AI assistants to perform complex tasks by leveraging external data and functionalities.

2.2.3.4 Advantages

MCP offers several benefits:

- **Standardization**: Provides a universal method for connecting AI assistants to external data sources and systems.
- **Scalability**: Simplifies integration across different AI systems and data, reducing the need for custom connectors.
- **Security**: Facilitates secure interactions between AI applications and local or remote resources.

By adopting MCP, developers can enhance the interoperability and efficiency of AI applications.

2.2.3.5 Applications

MCP has been applied across various domains:

- **Software Development**: IDEs like Zed and platforms like Replit use MCP to provide coding assistants with real-time code context.
- **Enterprise Assistants**: Companies utilize MCP to allow internal assistants to retrieve information from proprietary documents and systems.
- **Natural Language Data Access**: Applications leverage MCP to connect models with databases, enabling plain-language queries.

These applications demonstrate MCP's versatility in enhancing AI capabilities across different sectors.

2.2.3.6 Conclusion

The Model Context Protocol represents a significant advancement in AI integration, offering a standardized approach to connect LLMs with external tools and data sources. By facilitating seamless interactions and context exchange, MCP empowers AI applications to perform more complex and context-aware tasks, paving the way for more intelligent and versatile AI systems.

2.2.4 LangChain and LangGraph

2.2.4.1 Overview

LangChain and LangGraph are complementary frameworks designed to facilitate the development of applications powered by Large Language Models (LLMs). While LangChain provides a modular approach to constructing sequential workflows, LangGraph introduces a graph-based paradigm, enabling the creation of complex, dynamic, and stateful AI systems.

2.2.4.2 LangChain: Modular Workflow Construction

LangChain is an open-source framework that simplifies the integration of LLMs into applications by allowing developers to build chains—sequences of calls to LLMs and other utilities. Its modular design supports the composition of various components such as prompt templates, memory modules, and tool integrations. This structure is particularly effective for tasks that follow a linear progression, such as document summarization, question answering, and conversational agents.

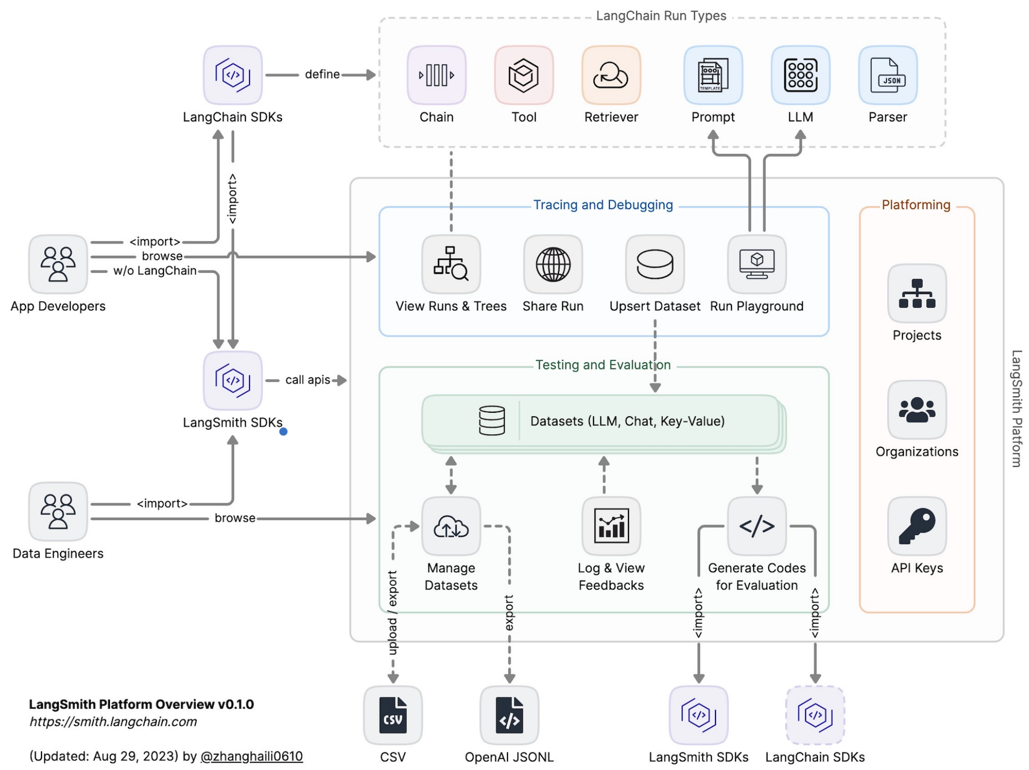


Figure 2.14. LangChain Architecture
[?]

Key features of LangChain include:

- **Prompt Templates**: Facilitate the creation of dynamic prompts for LLMs.
- **Chains**: Enable the linking of multiple components to form a cohesive workflow.
- **Agents**: Allow for decision-making capabilities by selecting appropriate actions based on user input.
- **Memory**: Maintain context across interactions to support stateful conversations.

2.2.4.3 LangGraph: Graph-Based Workflow Orchestration

Building upon the foundations of LangChain, LangGraph introduces a graph-based approach to workflow orchestration, allowing for the modeling of complex, non-linear, and cyclical processes. In LangGraph, workflows are represented as directed graphs comprising nodes (representing operations or agents) and edges (defining the flow between nodes). This structure is particularly advantageous for applications requiring iterative processing, conditional branching, and multi-agent coordination.

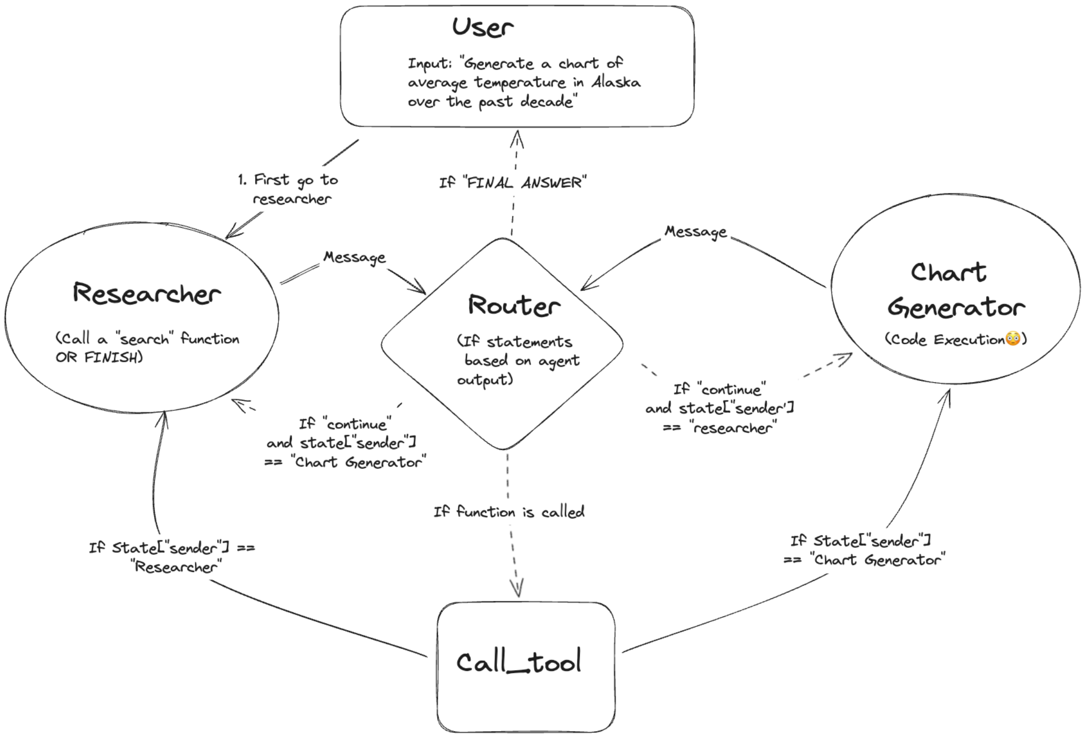


Figure 2.15. LangGraph: Multi-Agent Workflows
[?]

Notable capabilities of LangGraph include:

- **Cyclic Workflows**: Support for loops and iterative processes within workflows.
- **Conditional Routing**: Dynamic decision-making to determine the next node based on runtime conditions.
- **State Management**: Persistent tracking of the system's state throughout the workflow execution.
- **Multi-Agent Collaboration**: Facilitation of interactions between multiple agents, each with specialized roles.

2.2.4.4 Comparative Analysis

While both LangChain and LangGraph are designed to streamline the development of LLM-powered applications, they cater to different complexity levels and use cases. LangChain's linear and modular approach is well-suited for straightforward tasks with a clear sequence of operations. In contrast, LangGraph's graph-based architecture excels in scenarios requiring complex control flows, such as applications with iterative loops, conditional logic, or collaborative agent interactions.

2.2.4.5 Conclusion

LangChain and LangGraph offer robust frameworks for developing applications that leverage the capabilities of LLMs. The choice between them hinges on the complexity and requirements of the intended application. For linear and relatively simple workflows, LangChain

Feature	LangChain	LangGraph
Workflow Structure	Sequential chains	Dynamic graphs
Complexity Handling	Simple/moderate workflows	Moderate/complex workflows
Iterative Processing	Limited	Fully supported (loops)
Conditional Logic	Basic conditions	Advanced conditions
Multi-Agent Coordination	Basic support	Robust collaboration
State Management	Basic retention	Advanced tracking
Use Case Suitability	Straightforward tasks	Complex adaptive tasks
Flexibility	Rigid structure	Highly flexible
Debugging and Monitoring	Good support	Enhanced visibility
Real-time Decision Making	Limited	Extensive support
Ideal Applications	Simple Q&A, summarization	Complex multi-agent apps

Table 2.2: Comparison of LangChain and LangGraph

provides an efficient and straightforward solution. Conversely, for applications necessitating complex control flows, iterative processing, and multi-agent coordination, LangGraph presents a more suitable architecture.

2.2.5 Langfuse and Observability

2.2.5.1 Overview

Langfuse is an open-source observability platform designed specifically for applications powered by Large Language Models (LLMs). It provides developers with tools to trace, debug, and analyze the behavior of LLM applications, ensuring better performance and reliability.

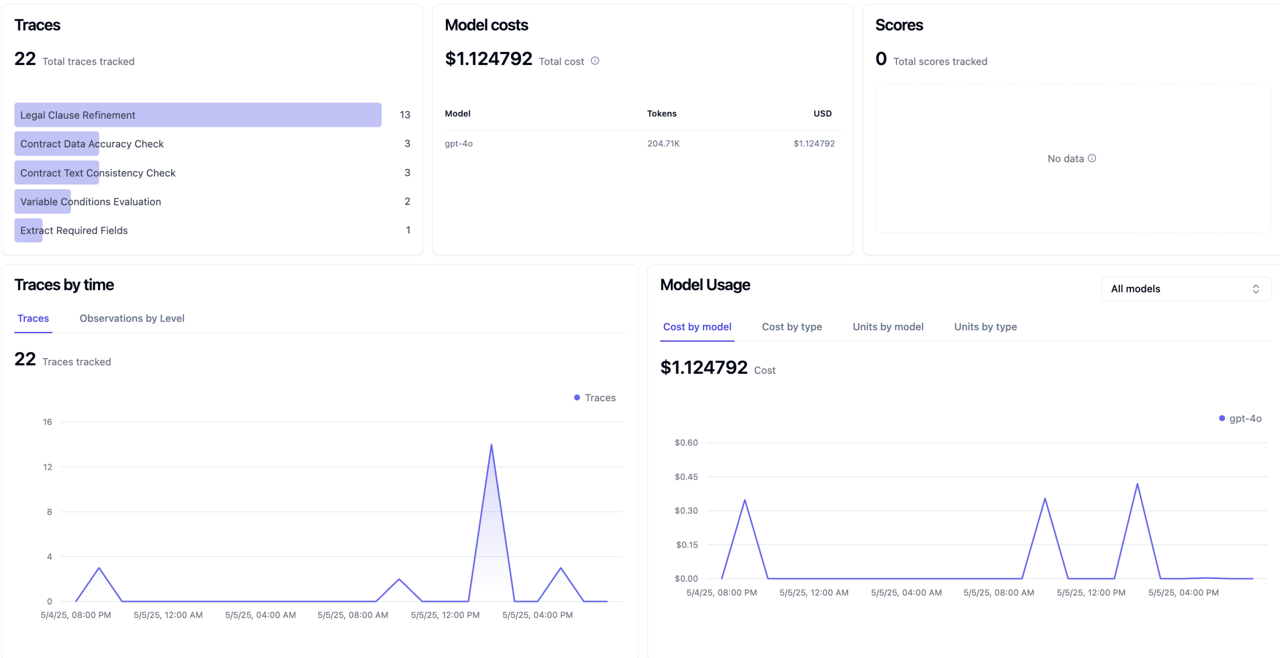


Figure 2.16. Langfuse Dashboard
[?]

2.2.5.2 Key Features

- **Comprehensive Tracing**: Langfuse captures detailed traces of LLM operations, including prompts, responses, and intermediate steps, allowing developers to understand the flow of data and identify issues effectively.
- **Session Management**: It groups related interactions into sessions, providing a holistic view of multi-turn conversations or complex workflows.
- **Analytics Dashboard**: Langfuse offers dashboards that display metrics such as latency, token usage, and cost, enabling teams to monitor and optimize application performance.
- **Prompt Management**: The platform includes tools for managing and versioning prompts, facilitating experimentation and iterative development.
- **Integration Support**: Langfuse integrates seamlessly with popular frameworks like LangChain, LlamaIndex, and OpenAI SDKs, as well as supports OpenTelemetry for broader observability needs.

2.2.5.3 Deployment Options

Langfuse can be deployed in various environments:

- **Cloud Hosting**: Utilize Langfuse's managed cloud service for quick setup and scalability.
- **Self-Hosting**: Deploy Langfuse on-premises or in private clouds using Docker, providing full control over data and configurations.

2.2.5.4 Use Cases

- **Debugging Complex Workflows**: By tracing each step of LLM operations, developers can pinpoint and resolve issues in complex applications.
- **Performance Monitoring**: Track metrics to identify bottlenecks and optimize resource usage.
- **Quality Assurance**: Analyze user interactions and feedback to improve response quality and user satisfaction.
- **Compliance and Auditing**: Maintain detailed logs for auditing purposes, ensuring compliance with regulatory standards.

2.2.5.5 Conclusion

Incorporating Langfuse into the development lifecycle of LLM applications enhances transparency, reliability, and efficiency, making it an invaluable tool for teams aiming to build robust AI solutions.

2.2.6 Tiptap for Rich Text Editing

2.2.6.1 Overview

Tiptap is a headless, open-source rich text editor framework built on ProseMirror, designed for developers seeking full control over their content editing interfaces. Its modular architecture and extensive extension ecosystem make it a powerful tool for building customized, collaborative, and AI-enhanced editing experiences.

Figure 2.17. Tiptap Editor
[?]

2.2.6.2 Architecture and Core Concepts

Tiptap is a headless, framework-agnostic rich text editor built on top of ProseMirror, designed to provide developers with full control over the editor's functionality and appearance. Its architecture is modular and extensible, allowing for the creation of customized editing experiences.

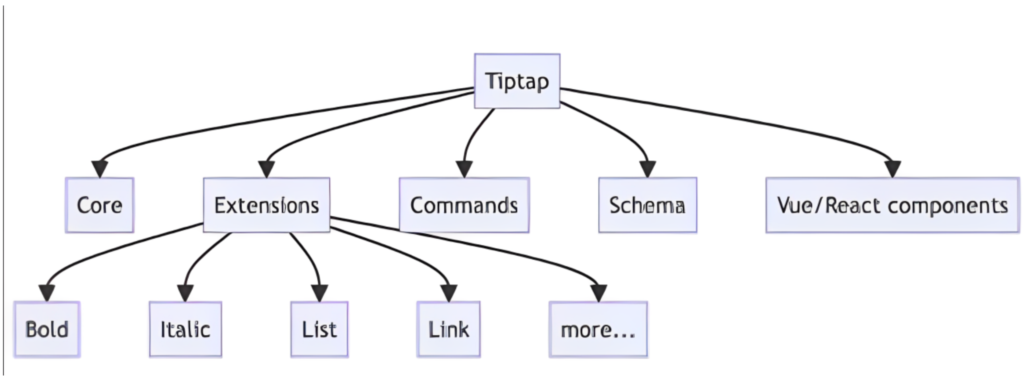


Figure 2.18. Tiptap Architecture
[?]

Key Architectural Components:

- **Schema**: Defines the structure of the document, specifying the allowed nodes (e.g., paragraphs, headings) and marks (e.g., bold, italic). This strict schema ensures content consistency and predictability.
- **State**: Represents the current content and selection within the editor. It is immutable and updated through transactions, enabling features like undo/redo and collaborative editing.
- **Transaction**: Encapsulates changes to the state, such as text insertions or formatting. Transactions ensure that state changes are predictable and manageable.
- **Extensions**: Modular units that add functionality to the editor, such as new nodes, marks, commands, or plugins. Tiptap provides a rich set of core and community extensions, and developers can create custom ones as needed.
- **Commands**: Functions that perform actions within the editor, often used to manipulate the document or respond to user input. Commands can be chained for complex operations.

2.2.6.3 Key Features

- **Extension-Based Modularity**: Tiptap offers over 100 extensions, including both open-source and Pro options, allowing developers to tailor the editor's functionality to specific needs.
- **Real-Time Collaboration**: Through integrations like Hocuspocus and CRDTs, Tiptap supports collaborative editing with features like live cursors and offline synchronization.
- **AI Integration**: The Content AI extension enables in-editor AI capabilities such as text suggestions, translations, and content generation, enhancing the writing experience.
- **Framework Agnostic**: Tiptap's design allows it to be integrated into various JavaScript frameworks, providing flexibility in application development.

2.2.6.4 Use Cases

Tiptap is suitable for a wide range of applications, including:

- Building Notion-like editors with custom blocks and interactions.
- Developing collaborative document editing platforms.
- Creating AI-assisted writing tools with real-time suggestions.
- Implementing rich text editors in CMS platforms.

2.2.6.5 Conclusion

Tiptap stands out as a versatile and developer-friendly rich text editor framework. Its combination of headless architecture, modular extensions, and support for real-time collaboration and AI integration makes it a compelling choice for modern web applications requiring customized content editing solutions.

2.3 Specification of Requirements

Given the agile nature of the project, the actors as well as functional and non-functional requirements are subject to evolve. The elements below reflect the current state of the project.

2.3.1 Actor Identification

An actor specifies a role played by a user or any other system interacting with the solution. Actors are always external to the system, interacting by initiating a use case, providing inputs, or receiving outputs. In the context of our platform, we have two types of actors:

Actor	**Role**
Sales User	Responsible for creating and managing contract documents and negotiations with clients.
Legal User	Oversees contract compliance, manages legal clauses, and reviews contracts.
Administrator	Manages platform users, roles, permissions, and oversees system configurations.

Table 2.3: Actors and Their Roles

2.3.2 Functional Requirements

2.3.2.1 Authentication

The system must allow users to authenticate using Azure Entra ID credentials.

2.3.2.2 Sales User Functionalities

Contract Generation:

- Initiate contract drafting using text input, voice transcription, or document upload.
- Verify client information from historical data.
- Select and adjust template inputs (Incoterm, Contract Type, Product).
- Adjust variables like price and volume.
- Generate final contract draft and move to editing mode.

Contract Editing:

- Access a rich-text editor (Tiptap-based).
- Track changes across versions.
- Request additional legal clauses.

Contract Review & Compliance:

- Trigger AI-powered compliance checks (data accuracy, text consistency, legal compliance).

- Receive detailed review summaries and recommendations.

Export and Sharing:

- Export contracts as downloadable files.
- Share contracts via secure links or attachments.

2.3.2.3 Legal User Functionalities

Clause Management:

- Review clause addition requests from Sales.
- Create and insert new legal clauses.
- Utilize AI assistance to refine clause content.

Contract Review & Compliance:

- Perform detailed legal compliance checks.
- Verify mandatory clauses, annexes, and specifications.
- Ensure legal alignment and minimize liabilities.

2.3.2.4 Administrator Functionalities

User Management:

- Add, modify, or remove platform users.
- Manage role-based permissions.

System Configuration:

- Oversee integration settings, security configurations, and monitoring dashboards.

2.3.3 Non-Functional Requirements

- **Usability**: The application should be intuitive, ensuring ease of use for Sales, Legal, and Admin users.
- **Maintainability**: The system should be modular and maintainable, supporting frequent updates.
- **Security**: Strict management of user access, data protection, and compliance with regulatory standards.
- **Performance**: Responsive interface, quick data retrieval, and AI analysis with low latency.
- **Scalability**: Easily scalable to support growing user base and additional functionalities.

2.3.4 Use Case Diagram

2.3.5 Textual Description

2.3.6 Analysis Sequence Diagram

2.3.7 Constraints and Assumptions

Constraints:

- Adherence to enterprise security and compliance guidelines.
- Compatibility with existing technological infrastructure.

Assumptions:

- Availability of clean and structured data.
- Consistent internet connectivity and cloud infrastructure reliability.

2.4 Conclusion

This chapter outlined the rationale behind our technology selection through comparative analysis, detailed the technical foundations and their roles, and defined the precise functional and non-functional project requirements. This structured approach ensures that the developed intelligent contract management solution robustly meets both user expectations and organizational objectives.

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.

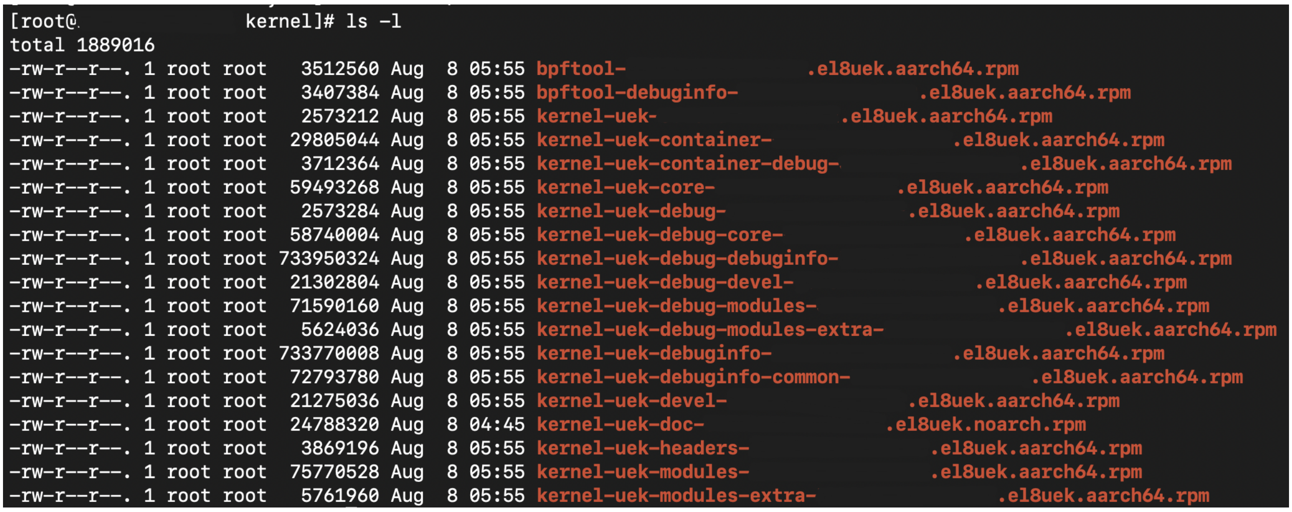


Figure 3.1. Kernel RPM Files Downloaded

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

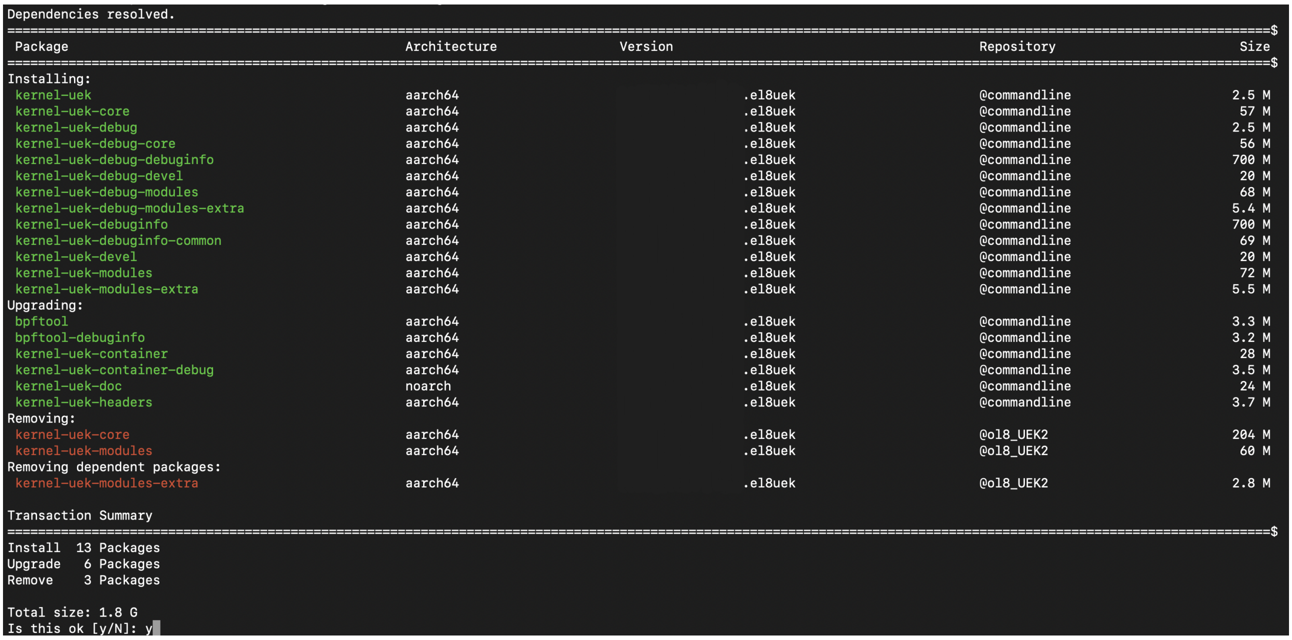


Figure 3.2. Kernel Installation Process

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.

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-               .el8.aarch64 is already installed.
Dependencies resolved.
=========================================================================================================================
 Package                               Architecture           Version                       Repository              Size
=========================================================================================================================
Upgrading:
 qemu-img                              aarch64                                              custom1                2.2 M
 qemu-kvm                              aarch64                                              custom1                 72 k
 qemu-kvm-block-curl                   aarch64                                              custom1                 83 k
 qemu-kvm-block-gluster                aarch64                                              custom1                 84 k
 qemu-kvm-block-iscsi                  aarch64                                              custom1                 90 k
 qemu-kvm-block-rbd                    aarch64                                              custom1                 85 k
 qemu-kvm-block-ssh                    aarch64                                              custom1                 85 k
 qemu-kvm-common                       aarch64                                              custom1                402 k
 qemu-kvm-core                         aarch64                                              custom1                4.7 M
Installing dependencies:
 librdmacm                             aarch64                                              ol8_baseos_latest       76 k
Downgrading:
 libiscsi                              aarch64                                              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                                     custom1                 33 k
Installing dependencies:
 glusterfs-cli                                  aarch64                                     ol8_appstream          183 k
 libvirt-daemon-driver-qemu                     aarch64                                     custom1                778 k
 libvirt-daemon-driver-storage                  aarch64                                     custom1                 33 k
 libvirt-daemon-driver-storage-core             aarch64                                     custom1                208 k
 libvirt-daemon-driver-storage-disk             aarch64                                     custom1                 43 k
 libvirt-daemon-driver-storage-gluster          aarch64                                     custom1                 45 k
 libvirt-daemon-driver-storage-iscsi            aarch64                                     custom1                 41 k
 libvirt-daemon-driver-storage-iscsi-direct     aarch64                                     custom1                 43 k
 libvirt-daemon-driver-storage-logical          aarch64                                     custom1                 44 k
 libvirt-daemon-driver-storage-mpath            aarch64                                     custom1                 39 k
 libvirt-daemon-driver-storage-rbd              aarch64                                     custom1                 48 k
 libvirt-daemon-driver-storage-scsi             aarch64                                     custom1                 40 k
 lzop                                           aarch64                                     ol8_baseos_latest       61 k
 systemd-container                              aarch64                                     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@[redacted] 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@[redacted] 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@[redacted] 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 [redacted] dnsmasq-dhcp[6945]: DHCPREQUEST(virbr0)
Aug 22 01:32:08 [redacted] dnsmasq-dhcp[6945]: DHCPACK(virbr0)
Aug 22 01:52:14 [redacted] dnsmasq[6945]: listening on virbr0(#4):
Aug 22 01:56:04 [redacted] dnsmasq-dhcp[6945]: DHCPREQUEST(virbr0)
Aug 22 01:56:04 [redacted] dnsmasq-dhcp[6945]: DHCPACK(virbr0)
Aug 22 02:21:37 [redacted] dnsmasq-dhcp[6945]: DHCPREQUEST(virbr0)
Aug 22 02:21:37 [redacted] dnsmasq-dhcp[6945]: DHCPACK(virbr0)
Aug 22 02:35:45 [redacted] systemd[1]: Stopping Virtualization daemon...
Aug 22 02:35:45 [redacted] systemd[1]: libvirtd.service: Succeeded.
Aug 22 02:35:45 [redacted] systemd[1]: Stopped Virtualization daemon.
[root@[redacted] 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@[redacted] ilyass]# virsh pool-define-as pool_dir- dir --target /dev/shm/ilyass/image/
setlocale: No such file or directory
Pool pool_dir- defined

[root@[redacted] 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@[redacted] 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@[redacted]]# /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=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 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;

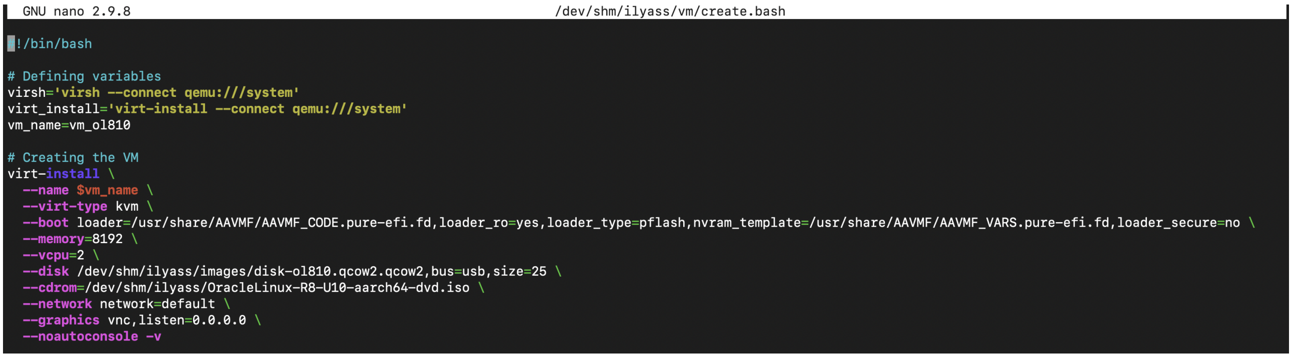
- **-smp cpus=32,cores=32,threads=2,sockets=1,maxcpus=64**: Configures the VM to use 32 CPUs, organized into 32 cores with 2 threads per core, and allows the number of CPUs to scale up to 64;
- **-drive file=/dev/shm/ilyass/image/disk-ol89.qcow2**: Specifies the QCOW2 disk image for the VM;
- **-drive 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@[redacted]]# hostnamectl
   Static hostname: [redacted]
         Icon name: computer-server
           Chassis: server
        Machine ID: [redacted]
           Boot ID: [redacted]
  Operating System: Oracle Linux Server 8.10
       CPE OS Name: cpe:/o:oracle:linux:8:10:server
            Kernel: Linux [redacted].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@[redacted]]# yum list installed | grep qemu
qemu-img.x86_64                 [redacted].el8          @custom1
qemu-kvm.x86_64                 [redacted].el8          @custom1
qemu-kvm-block-curl.x86_64      [redacted].el8          @custom1
qemu-kvm-block-gluster.x86_64   [redacted].el8          @custom1
qemu-kvm-block-iscsi.x86_64     [redacted].el8          @custom1
qemu-kvm-block-rbd.x86_64       [redacted].el8          @custom1
qemu-kvm-block-ssh.x86_64       [redacted].el8          @custom1
qemu-kvm-common.x86_64          [redacted].el8          @custom1
qemu-kvm-core.x86_64            [redacted].el8          @custom1
qemu-kvm-docs.x86_64            [redacted]              @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: [redacted]
           Boot ID: [redacted]
    Virtualization: kvm
  Operating System: Oracle Linux Server 8.9
       CPE OS Name: cpe:/o:oracle:linux:8:9:server
            Kernel: Linux [redacted].el8uek.x86_64
      Architecture: x86-64
[root@OL89_VM ~]#
```

Figure 4.8. Verifying Hostname within the Guest

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

```
(qemu) system_powerdown
system_powerdown
(qemu)
```

Figure 4.18. Executing Shutdown Test on the Guest

```
g Monitoring of LVM2 mirrors…ng dmeventd or progress polling...
[  OK  ] Stopped Monitoring of LVM2 mirrors,…sing dmeventd or progress polling.
[ 1425.133025] audit: type=1130 audit(1724637290.057:213): pid=1 uid=0 auid=4294967295 ses=4294967295 subj=system_u:system_r:init_t:s0 m
sg='unit=lvm2-monitor comm="systemd" exe="/usr/lib/systemd/systemd" hostname=? addr=? terminal=? res=success'
[  OK  ] Reached target Shutdown.
[  OK  ] Reached target Final Step.
[  OK  ] Started Power-Off.
[  OK  ] Reached target Power-Off.
[ 1425.150740] printk: systemd-shutdow: 36 output lines suppressed due to ratelimiting
[ 1425.160742] systemd-shutdown[1]: Syncing filesystems and block devices.
[ 1425.161661] systemd-shutdown[1]: Sending SIGTERM to remaining processes...
[ 1425.172713] systemd-shutdown[1]: Sending SIGKILL to remaining processes...
[ 1425.177497] systemd-shutdown[1]: Unmounting file systems.
[ 1425.178403] [5527]: Remounting '/' read-only in with options 'seclabel,attr2,inode64,logbufs=8,logbsize=32k,noquota'.
[ 1425.185938] systemd-shutdown[1]: All filesystems unmounted.
[ 1425.186371] systemd-shutdown[1]: Deactivating swaps.
[ 1425.186765] systemd-shutdown[1]: All swaps deactivated.
[ 1425.187154] systemd-shutdown[1]: Detaching loop devices.
[ 1425.187621] systemd-shutdown[1]: All loop devices detached.
[ 1425.188305] systemd-shutdown[1]: Stopping MD devices.
[ 1425.290050] printk: shutdown: 10 output lines suppressed due to ratelimiting
[ 1425.323637] dracut: Taking over mdmon processes.
[ 1425.324065] dracut Warning: Killing all remaining processes
dracut Warning: Killing all remaining processes
[ 1425.358750] XFS (dm-0): Unmounting Filesystem
[ 1425.364868] dracut Warning: Unmounted /oldroot.
Aug 26 01:54:50 | /etc/multipath.conf does not exist, blacklisting all devices.
Aug 26 01:54:50 | You can run "/sbin/mpathconf --enable" to create
Aug 26 01:54:50 | /etc/multipath.conf. See man mpathconf(8) for more details
[ 1425.389785] dracut: Disassembling device-mapper devices
[ 1425.403787] dracut: Waiting for mdraid devices to be clean.
[ 1425.405880] dracut: Disassembling mdraid devices.
Powering off.
[ 1425.412113] kvm: exiting hardware virtualization
[ 1425.413642] sd 0:0:0:0: [sda] Synchronizing SCSI cache
[ 1425.414095] sd 0:0:0:0: [sda] Stopping disk
[ 1425.454356] ACPI: PM: Preparing to enter system sleep state S5
[ 1425.454884] reboot: Power down
[root@                    ]#
```

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:

```
[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 4.20. 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
```

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
```

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=pciroot${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@[redacted]]# hostnamectl
   Static hostname: [redacted]
         Icon name: computer-server
           Chassis: server
        Machine ID: [redacted]
           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@[redacted]]# 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@[redacted]]# 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:**

```
QEMU 7.2.0 monitor - type 'help' for more information
(qemu) stop
stop
(qemu) info status
info status
VM status: paused
(qemu) cont
cont
```

Figure 4.45. Stop and Continue Commands Executed on the Guest

- **Suspend Test:**

```
(qemu) info status
info status
VM status: paused (suspended)
(qemu)
```

Figure 4.46. Guest Suspension via Systemctl Command

- **Shutdown Test:**

```
(qemu) system_powerdown
system_powerdown
(qemu)
```

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@[redacted]]# hostnamectl
   Static hostname: [redacted]
         Icon name: computer-server
           Chassis: server
        Machine ID: [redacted]
           Boot ID: [redacted]
  Operating System: Oracle Linux Server 8.10
       CPE OS Name: cpe:/o:oracle:linux:8:10:server
            Kernel: Linux [redacted].el8uek.aarch64
      Architecture: arm64
[root@[redacted]]# 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        : [redacted]
Version      : [redacted]
Release      : [redacted]
Architecture : aarch64
Size         : 0.0
Source       : qemu-kvm-[redacted].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@[redacted]]# 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      : [redacted]
Release      : [redacted]
Architecture : aarch64
Size         : 0.0
Source       : libvirt-[redacted].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@[redacted] 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@[redacted] 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@[redacted] ilyass]# virsh start vm_ol810
setlocale: No such file or directory
Domain 'vm_ol810' started

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

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

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

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

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

function_values=1
nbr=0

for i in {1..[redacted]}
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..[redacted]}; do
        virsh attach-device vm_ol810 /tmp/vfio-$i.xml --live
done
[root@[redacted]]# 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 [redacted] brd [redacted]
    inet [redacted] scope host lo
       valid_lft forever preferred_lft forever
    inet6 :[redacted] 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:[redacted] 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:[redacted] 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:[redacted] brd ff:ff:ff:ff:ff:ff
    inet6 fe80:[redacted] 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:[redacted] 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:[redacted] 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:[redacted] 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:[redacted] 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:[redacted] 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:[redacted] 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:[redacted] 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] Executive leadership board. `https://theorg.com/org/bcg-x`.