ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

SVLC: Service virtualization based on Linux Containers

Daniel Carvalho

Supervisors: Luis Osório
Mário Pinheiro

Project report carried out within the scope of Project and Seminar.
Degree in IT and Computer Engineering.

June, 2024

Instituto Superior de Engenharia de Lisboa

SVLC: Service virtualization based on Linux Containers

Student: 49419 - Daniel Martinho de Carvalho

Supervisores: António Luís Freixo Guedes Osório

Mario Justiniano Morais Pinheiro

Project report carried out within the scope of Project and Seminar.

Degree in IT and Computer Engineering.

Resume

The Linux container concept has experienced a notable surge in adoption over the past decade, largely propelled by the widespread adoption of Docker. As organizations increasingly seek efficient and scalable solutions for application deployment and management, Linux containers have emerged as a pivotal technology, offering lightweight, portable, and isolated environments for running applications.

In this project, our goal is to develop a comprehensive set of functionalities to facilitate the instantiation, management, and deployment of applications, or components thereof, within Linux containers. Furthermore, we aim to extend these capabilities to the Kubernetes network—a leading container orchestration platform—enabling seamless integration with cloud-native architectures.

Aligned with the principles of open collaboration and standardization, our project will embrace open source initiatives, particularly those endorsed by the Linux Foundation's Open Container Initiative (OCI). By leveraging open standards and community-driven development, we seek to foster interoperability, transparency, and innovation within the container ecosystem.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Context

The rapid evolution of technology has motivated the development of more efficient and scalable solutions for application deployment and management. Traditional virtualization techniques, while useful, often incur significant overhead in terms of resource consumption and operational complexity. This challenge has led to the emergence and subsequent popularity of containerization technologies, with Linux containers at the forefront. Over the past decade, Linux containers have transformed from niche tools to mainstream solutions, largely due to the introduction and widespread adoption of Docker [1].

1.2 The Rise of Containers

The concept of image repositories is considered the innovative element that catapulted the use of containers [1]. Another dimension was the simplicity in utilizing existing operating system services for container management, such as Linux Containers (LXC).

Docker, introduced in 2013, revolutionized the way developers build, ship, and run applications. By providing a simple yet powerful abstraction for containerization, Docker made it possible to package applications and their dependencies into a single, portable container that can run consistently across various environments [2]. This innovation has driven a surge in container adoption, as organizations recognize the benefits of lightweight, portable, and isolated application environments.

1.3 The Need for Container Orchestration

As container usage proliferated, the need for effective management and orchestration became evident. Managing a few containers is relatively straightforward, but as applications scale out to hundreds or thousands of containers, the complexity increases exponentially [3]. This complexity is addressed by container orchestration platforms, with Kubernetes being the most prominent. Kubernetes automates the deployment, scaling, and operation of containerized applications, making it an essential component of modern cloud-native architectures [3].

1.4 Project Overview

This project aims to develop a comprehensive set of functionalities to facilitate the instantiation, management, and deployment of applications, or components thereof, within Linux containers. By extending these capabilities to the Kubernetes network, we seek to enable seamless integration with cloud-native architectures, thereby enhancing scalability, reliability, and efficiency (see figure 1.1) [4].

1.5 Embracing Open Standards

In alignment with the principles of open collaboration and standardization, the project outcomes will embrace open-source initiatives, particularly those endorsed by the Linux Foundation's Open Container Initiative (OCI). The OCI provides industry standards for container formats and runtimes, ensuring interoperability and fostering innovation within the container ecosystem [5]. By leveraging these open standards and community-driven development, our project aims to contribute to the broader containerization landscape, promoting transparency and reducing vendor lock-in.

1.6 Key Objectives

The key objectives of this project are:

To develop suite of functionalities that simplify the creation, management, and deployment of Linux containers. To integrate these tools with Kubernetes, providing robust orchestration capabilities. To adhere to OCI standards, ensuring compatibility and interoperability with existing container technologies. To support and contribute to the open-source community, fostering a culture of collaboration and continuous improvement.

1.7 Chapter Breakdown

The remainder of this report is structured as follows:

- **Chapter 2 -** provides a detailed review of existing container technologies, orchestration platforms, and theoretical concepts essential for understanding the project.
- **Chapter 3 -** outlines the design and implementation of our container management and deployment functionalities, along with the discussion of Kubernetes integration.
- **Chapter 4 -** evaluates the performance and scalability of our solutions through a series of tests and benchmarks.
- **Chapter 5 -** concludes the report with a summary of our findings, contributions, and suggestions for future work.

Through this project, we aim to advance containerization technology, offering practical tools and insights to benefit the broader community and drive further innovation in the field.

Chapter 2

Background Knowledge

The concept of Linux containers and container clusters has gained increasing importance in the realm of cloud computing. Cloud computing can be defined as the delivery of computing services -including servers, storage, databases, networking, software, analytics, and intelligence- over the Internet (”the cloud”) to offer faster innovation, flexible resources, and economies of scale [6].

Questioning the concept of application boundaries and the responsibilities of computational elements, as well as the notion of services (more recently microservices), underscores the need for the concept of an information system composed of computational elements designated as services. This is elucidated in the technological framework of ISoS (Informatics System of Systems), which abstracts technological artifacts of a computational nature and related aspects.

In reality, an application or informatics system it formally represents a set of computational logic and supporting resources, such as configuration files, that execute (following deployment) within an execution environment primarily based on the operating system. However, when applications are distributed—comprising parts running on networked computers—a solitary operating system proves insufficient. This necessitates a cluster of computers, particularly in the era of cloud computing, where the scale of users and volume of data demand a lighter infrastructure capable of coordinating the execution of distributed application parts across a network of operating systems.

A ”lightweight” operating system, containing only the necessary resources for executing a component (service) of an application, is therefore referred to as a container. The network or cluster of containers, which encompasses the elements of a distributed application, finds embodiment in the idea of a distributed operating system, exemplified by Kubernetes, a rapidly growing adoption in the field.

The remainder of the chapter is organized into the following sections:

1. Informatics System of Systems (ISoS);
2. Linux Containers and Podman;
3. Kubernetes Technology Framework;
   - Node Structure;
   - Kubernetes Cluster.
4. Continuous Integration and Continuous Deployment (CI/CD);
   - Continuous Integration;
   - Continuous Deployment.
5. Application Packaging (ISystem) in Kubernetes.
   - Helm Package Manager.

2.1 Informatics System of Systems (ISoS)

The ISoS framework addresses the need for a cohesive approach to integrate diverse technological components within an application, regardless of the underlying programming languages or runtime environments. It provides an integration model that accommodates multiple suppliers or vendors, ensuring compatibility and interoperability across various technological landscapes [7].

At its core, an ISoS comprises interconnected components, including ISystems, Cooperative Enabled Services (CES), and individual Services (see figure 2.1). These elements collectively encapsulate the computational logic and essential resources required for application execution. By leveraging the container model and encapsulating technological artifacts within service elements, ISoS enables the deployment of isolated and independent applications [8]. This approach promotes modularity, scalability, and flexibility, essential qualities in modern software development environments.

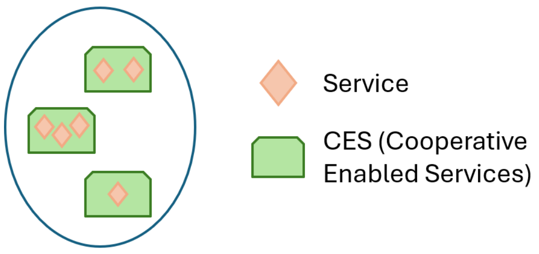

Figure 2.1: Example of an ISystem.

2.2 Linux Containers and Podman

Linux containers represent an isolated set of one or more processes and related resources. These containers derive all necessary files from a distinct image, ensuring portability and consistency across various stages of development and production. This characteristic facilitates expedited usage compared to traditional "development pipelines reliant on replicating testing environments" [9].

Podman, short for Pod Manager, is a lightweight container management tool that has gained prominence as an alternative to Docker. Developed from Docker's open-source code, Podman maintains compatibility with Docker while following an open-source software (OSS) model. The Podman-desktop version offers a user-friendly interface for managing containers and pods, providing functionalities for creating, running, and managing containerized applications. Podman embraces the principles of containerization, enabling users to deploy and manage containers efficiently across diverse computing environments [10] that supports a compatible container runtime.

The concept of a Pod, as developed in the context of the Kubernetes infrastructure, constitutes an isolated set comprising one or more containers (see Figure 2.2) that share the same namespaces and cgroups (resource constraints). Namespaces provide isolation for system resources such as process IDs, network access, and file systems, ensuring that containers within a pod do not interfere with each other or with other pods. Cgroups (control groups) limit and isolate the resource usage (CPU, memory, disk I/O, etc.) of the containers, providing resource management and allocation. Upon the addition of a container to a pod, Podman incorporates the container process into the cgroups and namespaces. Moreover, each container within a pod features a container monitor process known as *conmon* [11].

To further illustrate the concept, consider a simple example using the system call *chroot*. The *chroot* system call changes the root directory of the current process and its children to a new location in the file system. This creates a contained environment where the process can only see and interact with files within the new root directory, similar to how containers operate with isolated file systems. For instance, when you use *chroot* to change the root directory to /var/mycontainer, any attempt by the process to access / will actually access /var/mycontainer. This isolation is fundamental to the operation of containers, ensuring that applications run in a consistent and controlled environment.

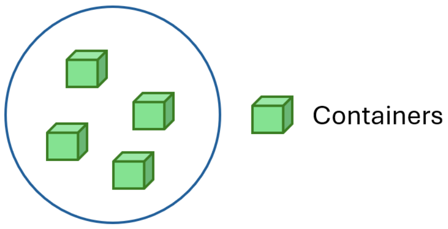

Figure 2.2: Example of a Pod.

2.3 Kubernetes Technology Framework

Kubernetes, a container orchestration platform, has emerged as a pivotal technology in modern software development, building upon previous contributions such as Apache Mesos [12]. As organizations seek efficient solutions for containerized application deployment and management, Kubernetes offers automated container deployment, scaling, and management capabilities [4].

The Kubernetes orchestrator boasts several essential runtime features that contribute to its effectiveness. These include service discovery, which provides stable DNS names and IP addresses to deployed services, facilitating seamless communication between components. Additionally, Kubernetes supports horizontal scaling, enabling applications to dynamically adjust resources based on workload demands. Load balancing functionality optimizes resource allocation and traffic distribution, enhancing performance and reliability.

Moreover, Kubernetes incorporates self-healing mechanisms to ensure high availability and fault tolerance. In the event of container failures, Kubernetes automatically restarts failed containers and redirects traffic away from temporarily unavailable instances. Configuration management features further streamline deployment processes, allowing for centralized configuration and efficient resource utilization [4].

Node Structure

A Node represents a single machine that can host one or more pods. These nodes can be part of a Kubernetes cluster if multiple pods are present (see Figures 2.3 and 2.4). In a production environment, a node is typically either a physical machine or a virtual machine hosted in the cloud [13].

Kubernetes Cluster

A Kubernetes cluster is a network of computer nodes that communicate and share resources, much like a honeycomb cluster (see Figure 2.4). The cluster and Kubernetes handle distributing work to those nodes. They also ensure that if a node is added or removed from the cluster, work will be shifted around as necessary [13].

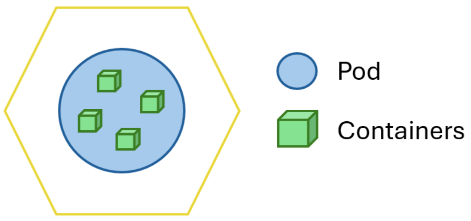

Figure 2.3: Example of a single node.

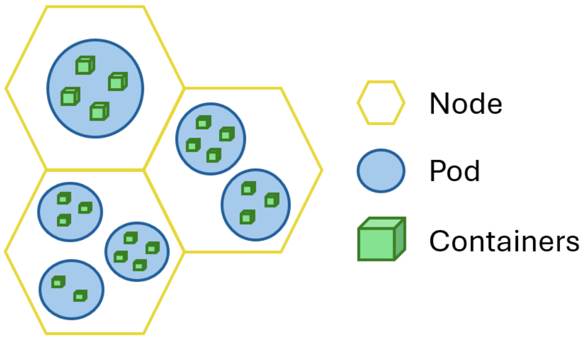

Figure 2.4: Example of a Kubernetes cluster.

2.4 Continuous Integration and Continuous Deployment (CI/CD)

Continuous Integration (CI) and Continuous Deployment (CD) are essential practices in modern software development. CI/CD automates the process of integrating code changes and deploying them to production environments, ensuring rapid and reliable software releases [14].

2.4.1 Continuous Integration

Continuous Integration involves the automated merging of code changes from multiple contributors into a shared repository. Tools such as Jenkins, Travis CI, and CircleCI facilitate this process by running automated tests and validating code changes to ensure they do not break the existing codebase [15].

2.4.2 Continuous Deployment

Continuous Deployment extends CI by automatically deploying code changes to production after they pass the automated tests. This practice minimizes manual intervention and reduces the time between development and release, leading to more efficient and frequent updates [16].

2.5 Application Packaging (ISystem) in Kubernetes

Application packaging in Kubernetes involves bundling applications and their dependencies into a deployable unit. Helm is a popular tool used for managing Kubernetes applications.

2.5.1 Helm Package Manager

Helm is a package manager for Kubernetes that simplifies the deployment and management of applications. It uses a packaging format called charts, which are collections of files that describe a related set of Kubernetes resources [**?**].

Chapter 3

SVLC Architecture and Implementation

The Service Virtualization based on Linux Containers (SVLC) suite builds on the ISoS framework to streamline the development and deployment of ISystems images. Our primary objective is to maximize automation in this process. By implementing CI/CD (continuous integration and continuous delivery) practices, we automate the manual tasks traditionally required to move new code from a commit into production. This includes the build, test (integration tests, unit tests, and regression tests), and deployment phases, as well as infrastructure provisioning [17].

To ensure a seamless deployment process, we develop mechanisms for deploying projects or ISystems while managing all dependencies and configurations (see figure 3.1). This guarantees that once deployed, the applications require no additional preparations to function correctly.

To illustrate the design and implementation, we use a simple CRUD (Create, Read, Update, Delete) server as our example. Initially, this server is deployed as a monolithic application on a single node using Podman. As the project progresses, we transition to a microservices architecture and deploy it in a Kubernetes cluster. This approach demonstrates the scalability and flexibility of containerization and orchestration solutions.

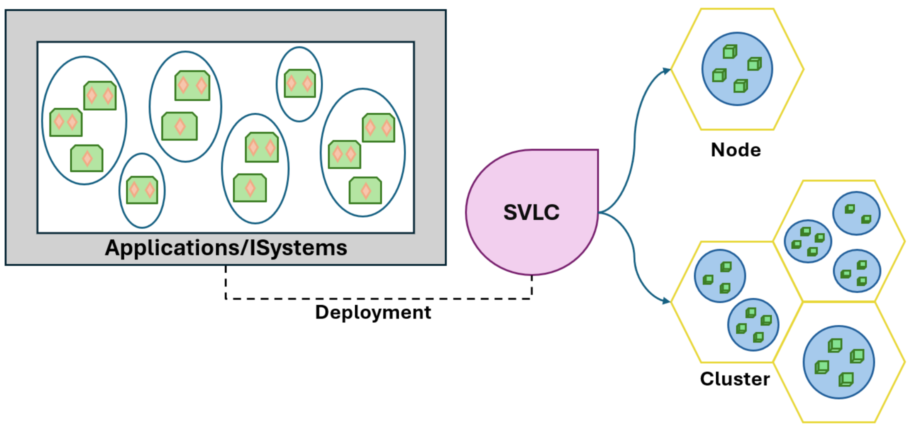

Figure 3.1: Implementation overview

The remainder of the chapter is organized into the following sections:

1. Implementation on a Single Node;
2. Implementation in a Kubernetes Cluster;
3. Challenges and Solutions;
4. Summary.

3.1 Implementation on a Single Node

3.2 Implementation in a Kubernetes Cluster

3.3 Challenges and Solutions

3.4 Summary

Chapter 4

Performance Evaluation

In this chapter, we're diving into the evaluation of our solutions performance and scalability, putting our systems through a battery of tests and benchmarks.

Performance evaluation is crucial for any system, especially when it comes to managing and deploying containers. We're subjecting our solutions to different workloads and conditions to see how they fare [18].

We using benchmarks to measure metrics like response times, throughput, and resource efficiency. These metrics give us valuable insights into our system's ability to handle peak loads.

By the end of this chapter, we'll have a clear understanding of our solutions' strengths and weaknesses, paving the way for future improvements and optimizations.

4.1 Tests and results

Chapter 5

Conclusion and Future Directions

References

[1] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[2] James Turnbull. *The Docker Book: Containerization is the new virtualization.* Lulu.com, 2014.

[3] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *ACM Queue*, 14(1):70, 2016.

[4] John Clingan and Ken Finnigan. *Kubernets Native Microservices - with Quarkus and MicroProfile.* Manning, 2022.

[5] Opencontainers.org. Open container initiative (oci), 2024. Retrieved from https://opencontainers.org/.

[6] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing, communications of the acm. *ACM Journals*, 2010.

[7] Carlos Gonçalves, Tiago Dias, A. Osório, and Luis Camarinha-Matos. *A Multi-supplier Collaborative Monitoring Framework for Informatics System of Systems*, pages 44–53. Springer, 09 2022.

[8] A. Osório, A. Belloum, Hamideh Afsarmanesh, and Luis Camarinha-Matos. Agnostic informatics system of systems: The open isos services framework. In *Collaboration in a Data-Rich World*, pages 407–420, 09 2017.

[9] Red Hat. What's a linux container? `https://www.redhat.com/en/topics/containers/whats-a-linux-container`, 2022. [Online; accessed 14-March-2024].

[10] Sphinx Technologies. What is podman? `https://docs.podman.io/en/latest/`, 2019. [Online; accessed 14-March-2024].

[11] Daniel J. Walsh. *Podman In Action - The next generation of container engines.* Manning, 2023.

[12] Benjamin Hindman, Andrew Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. *USENIX Association*, 44(2), 2011.

[13] Daniel Sanche. Kubernetes 101: Pods, nodes, containers, and clusters. `https://medium.com/google-cloud/kubernetes-101-pods-nodes-containers-and-clusters-c1509e409e16`, 2018. [Online; accessed 16-April-2024].

[14] Martin Fowler. Continuous integration, 2006.

[15] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley Professional, 2010.

[16] Mazin Y. Shahin, Peng Liang, and Mohammad A. Babar. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 2017.

[17] GitLab. What is ci/cd? `https://about.gitlab.com/topics/ci-cd/`. [Online; accessed 17-March-2024].

[18] John Smith. Performance evaluation of container management systems. *Journal of Container Computing*, 10(2):45–56, 2024.