FH MÜNSTER

ETI FB Elektrotechnik und Informatik
Department of Electrical Engineering
and Computer Science

A Thesis Submitted in Partial Fulfilment of the Requirements
for the Degree of Master of Science in Computer Science

Evaluating (De-) Centralized Solutions for Kubernetes Multi-Cluster Management

Focused on Service Discovery and Peer-to-Peer Relaying

Leon Alexander Kraß

Student ID: 947743

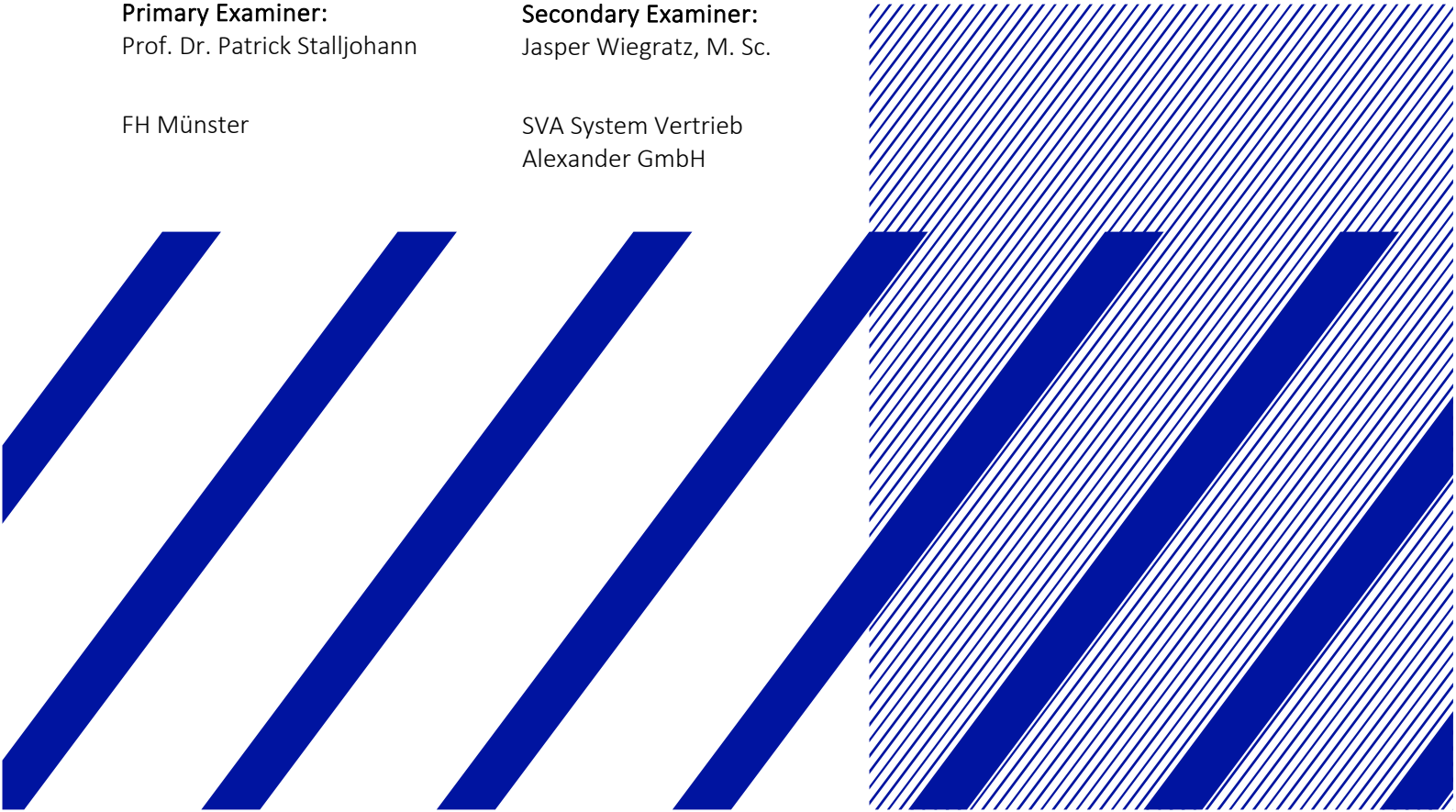
Submitted on the 7th of March 2025

Primary Examiner:
Prof. Dr. Patrick Stalljohann

FH Münster

Secondary Examiner:
Jasper Wiegratz, M. Sc.

SVA System Vertrieb
Alexander GmbH



Abstract

Kubernetes** multi-cluster management* is a challenging task, particularly in terms of *service discovery* for *clusters* and relaying the ***Kubernetes** API*. To address these challenges, various solutions showing different architectural approaches have been implemented. To evaluate these solutions and their corresponding *architectures*, an *evaluation model* based on the ***Systems and software Quality Requirements and Evaluation series (***ISO/IEC 25000** – **25099***) is proposed. This model is applied to ***Rancher***, a centralized solution, and ***kubeanchors***, a new decentralized solution based on ***Skupper***. The results of the evaluation are presented and discussed to provide further insights into the strengths and weaknesses of the assessed solutions and the *evaluation model* itself.

Acknowledgments

To address the acknowledged individuals more personally, the following words are written in German:

Die Ausarbeitung meiner Master-Thesis stellt für mich einen wichtigen Meilensteil meiner akademischen, beruflichen und persönlichen Entwicklung dar. Daher möchte ich allen Beteiligten mit diesen Worten herzlich danken, mich auf diesem Weg unterstütz zu haben!

Ein besonderer Dank gilt meinem Erstprüfer, Herrn Prof. Dr. Stalljohann. Vielen Dank, dass Sie mir stets mit Rat und Tat zur Seite standen, nicht nur bei dieser Arbeit, sondern auch über meine gesamte akademische Laufbahn an der FH Münster hinweg! Die Erfahrungen, die ich durch Sie sammeln durfte, haben meine fachliche Ausrichtung und Interessen nachhaltig positiv inspiriert.

Ich möchte Jasper Wiegratz für die tatkräftige Unterstützung als Zweitprüfer, Arbeitskollegen und besonders als Freund danken. Danke, Jasper, ohne Dich wäre ich nicht zu diesem Ergebnis gekommen! Deine Ideen, fachliche Kompetenz und Motivation sind beachtenswert und haben mir ungemein geholfen.

All meinen Kollegen bei der SVA System Vertrieb Alexander GmbH, besonders denen, die mir stets den Rücken freigehalten haben, gilt an dieser Stelle mein aufrichtiger Dank! Ohne Euch hätte ich es nicht geschafft, die Master-Thesis zu bestreiten. Auch für die Bereitstellung entsprechender Ressourcen für diese Arbeit möchte ich der SVA herzlich danken!

Zuletzt möchte ich meiner Familie und meinen engen Freunden danken. Ohne Eure wohltuenden Worte und Zusprüche und ständige, bedingungslose Unterstützung wäre mein Weg zur Master-Thesis deutlich steiniger gewesen. Ihr habt mich stets aufgemuntert und motiviert!

Editorial Notes

Differing from the initially submitted proposal, this thesis primarily focuses on the evaluation of solutions for ***Kubernetes** multi-cluster management* rather than solely on ***kubeanchors***. However, ***kubeanchors*** remains a relevant component, as it serves as one of the targets evaluated.

The proposed “concept of evaluation” and “test cases” became the central focus of this thesis. A different approach than the ones suggested in the “state of research” section was followed to implement ***kubeanchors***, still providing compliance to the mentioned caveats in the proposal.

Additionally, the distribution of pages per Chapter and the Chapter titles from the initially proposed thesis outline have been adjusted to better suite the final structure of the thesis.

Table of Contents

Appendix

Tables

Figures

Code

Formatting Conventions

In this thesis the terms Chapter, Section, Subsection and Paragraph correspond to each level of indentation of headlines. A **Chapter** is introduced by a headline numbered with a single digit (e. g., “1”). Within each Chapter, **Sections** are introduced with two-digit numbering (e. g., “1.1”). **Subsections** are further divided with three-digit numbering (e. g., “1.1.1”). Lastly, **Paragraphs** within Sections or Subsections are marked with lettered headlines (e. g., “a)”). Please note that these terms are spelled uppercase for distinction.

Different highlighting is used to convey specific meaning for certain terms:

Format	Explanation
own definitions	Own definitions are terms defined on behalf of this thesis to provide a more precise distinction of technical terms that may vary across different literature sources or to establish a common terminology. The full definitions can be found in Subsection 2.1.2.
technical term	Technical terms comply to an external definition. If not explained within the context of this thesis, a short explanation can be found in the glossary (Appendix B). Terms that require further explanation are outlined in Subsection 2.1.1.
proper names	Names of organizations, brands, products, *software* or similar. On first occurrence a reference for further details is accompanied.
command	(Linux) command that can be executed within a shell. The commands are further described per reference within the text.
fat words	Words that should be highlighted for various reasons.
Hyperlink	To ease navigation within a digital copy of this document, hyperlinks are provided.
Abbreviations (abbr.)	When abbreviations are introduced, their full form is provided upon first occurrence. Common abbreviations are not necessarily introduced within the text. Therefore, a complete list of abbreviations and their full forms can be found in Appendix A.

Figures leveraging the ***Unified Modeling Language*** (***UML***) use version 2.5.1 [1] of the standard.

1. Introduction

1.1. Motivation

More and more companies are leveraging ***Kubernetes*** [2] as their primary solution for *container* orchestration [3]. ***Kubernetes*** comes in many flavors, including managed cloud distributions such as ***Azure Kubernetes Service*** (***AKS***) [4], ***Amazon Elastic Kubernetes Service*** (***EKS***) [5], and ***Google Kubernetes Engine*** (***GKE***) [6], as well as self-hosted distributions like ***MicroK8s*** [7], ***K3s*** [8], or ***Red Hat OpenShift*** [9].

As a result, managing multiple ***Kubernetes*** *clusters* (*clusters*) and diverse ***Kubernetes*** distributions introduces new challenges. Two key challenges from the perspective of both *cluster administrators* and *cluster users* are the discovery of existing *clusters* and establishing reliable connections to these *clusters* across complex network infrastructures.

Solutions like ***Rancher*** [10] address these challenges by providing a centralized *multi-cluster management* service, where the *clusters* are registered. With proper network connectivity to the registered *clusters*, the ***Kubernetes*** *API* of these *clusters* can be relayed to different peers.

However, centralized solutions like ***Rancher*** may have reliability concerns. If the management service experiences an outage or is misconfigured, it becomes unavailable, preventing both the discovery of and access to the registered *clusters*.

Therefore, the challenges of *cluster* discovery and *cluster* connectivity should be solved without the downsides of a centralized solution. Thus, a new decentralized approach for *multi-cluster management* was developed. This solution is called ***kubeanchors.***

1.2. Objective

To provide evidence or disprove that a *multi-cluster management* solution with a *decentralized architecture* like ***kubeanchors*** provides advantages over a solution based on a *centralized architecture* like ***Rancher***, a thorough evaluation must be conducted. The objective of this thesis is to develop a generic *evaluation model*, applicable to ***Kubernetes*** *multi-cluster management* solutions that address the identified challenges.

The model will be based on ***ISO/IEC*** [11], [12] standards, which provide guidance on developing models for assessing *software quality*. A literature review will be carried out to determine *quality criteria* relevant to the given context.

Additionally, an *evaluation environment* will be developed to enable repeatable and standardized measurements. To further improve reproducibility and simplify the evaluation execution, *quality measurement scenarios* will be introduced.

Finally, the *evaluation model* will be applied to ***Rancher*** and ***kubeanchors*** as examples to demonstrate its usage and also to compare both solutions and their different *architectures*.

2. Foundations

To establish a common understanding of the foundations of this thesis, several definitions are provided within this Chapter. Additionally, the specific domain of this thesis is further explained so that non-specialists with fundamental knowledge in information technology can gain an understanding of the topics covered.

2.1. Definitions

2.1.1. Common Terms

a) Software

The ***IEEE Standard Glossary of Software Engineering Terminology*** [13] defines *software* as "computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system" [13, p. 66].

Synonyms:

- *software system*

b) (Software) Quality

ISO/IEC 25000 [14] defines *software quality* (*quality*) as the "capability of *software* […] to satisfy stated and implied needs when used under specified conditions" [14, p. 6]. Within this thesis, the *quality* of *evaluation targets* will be measured by the application of an *evaluation model*, providing quantitative metrics for various *quality criteria*.

c) Stakeholder

According to ***ISO/IEC 25000***, a *stakeholder* is an "individual or organisation having a right, share, claim or interest in a system […] that meet their needs and expectations" [14, p. 6].

2.1.2. Own Definitions

a) Centralized, Decentralized Architectures

An *architecture* is defined as "the organizational structure of a system" [13, p. 10]. In a *centralized architecture*, the core functionality of a *software system* is provided through a single central component. In contrast, a *decentralized architecture* distributes the main functionality across multiple components.

b) Evaluation Model, Target, Environment

An *evaluation model* is a conceptual framework designed to establish comparability between multiple *evaluation targets*. It achieves this by mapping one or more numeric values to each target and defining the methods for calculating these values. The process of evaluation is conducted within a unified *evaluation environment* to provide transparency and repeatability.

c) Quality Criterion

The term *quality criterion* is commonly used to describe a property of *software quality*. *Quality criteria* can be measured using *quality measures* and prioritized to determine whether a *stakeholder*'s requirements for a *software system* are met. Refer to [15, p. 39], [16, pp. 4–5], [17] and Section 3.1 for further details.

Synonyms:

- ***Bass et al.*** [15]: *quality attribute*
- ***ISO/IEC 25002*** [16]: *(quality) (sub-) characteristic*
- ***arc42 Quality Model*** [18]: *quality (attribute), (quality) property*

d) Quality Measurement Scenarios

Scenarios for assessing *quality measures*, refer to Section 3.5.

Synonyms:

- ***Bass et al.***: *quality attribute scenario*
- ***arc42 Quality Model***: *quality scenario*

e) Cluster Administrator, Cluster User

Groups of *stakeholders*, refer to Section 3.2.

f) Peer-to-Peer Relaying

Refer to Paragraph 2.2.2.b).

2.2. Domain

This Subsection explains the domain on which the research of this thesis is based. The details provided address to offer a deeper insight into the topics covered.

2.2.1. Kubernetes

Kubernetes is an orchestrator designed to automate the deployment, scaling, and management of containerized *software* [19, Ch. 1]. It provides a robust infrastructure for *service discovery*, load balancing, storage orchestration, secret and configuration management, and other common challenges encountered in *software* deployment [2]. To achieve this, ***Kubernetes*** offers an *API* that allows users to define the desired state of various artifacts, such as *pods*, *deployments*, *services*, and *ingresses*, which collectively serve as building blocks for modeling applications within a ***Kubernetes*** environment.

a) Common Artifacts

Figure 2.1, derived from [20], outlines a typical setup within a ***Kubernetes*** *cluster* spanning across two hosts. The smallest ***Kubernetes*** artifact in this setup is called a *pod* and consists of one or multiple *containers*. The *containers* within a *pod* share the same networking namespace and, consequently, the same IP and ports [19, Ch. 5]. *Containers* "that run along with the main application *container*" [21] are often referred to as sidecar *containers*.

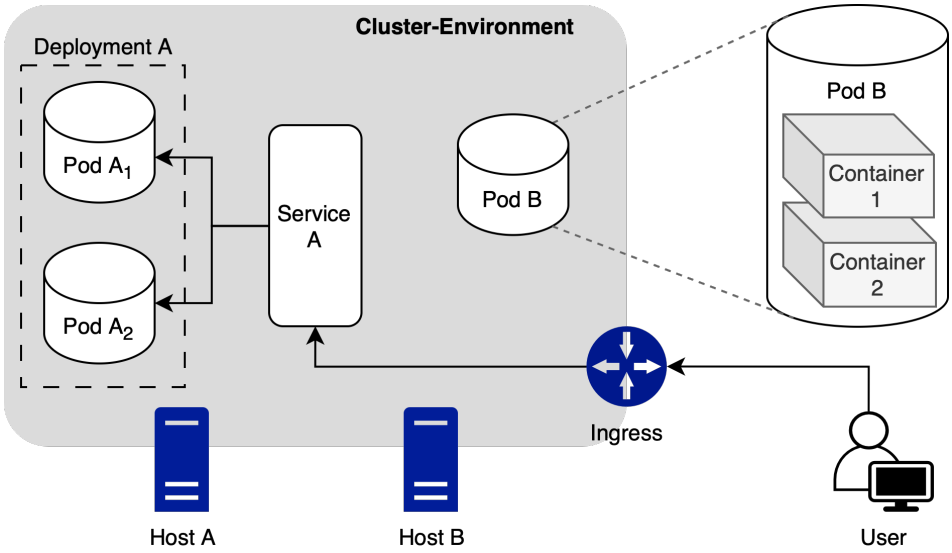


Figure 2.1: Kubernetes

Pods can be collectively maintained within a *deployment*, which defines the desired number of *pod* replicas and enables dynamic scaling [19, Ch. 10]. To provide a common networking endpoint for the *pods* in a *deployment*, a *service* can be used [19, Ch. 7].

For *services* that need to be accessible by users outside of the *cluster* environment, an *ingress* can be created. An *ingress* exposes a ***Kubernetes** service* via *HTTP(S)* on a defined endpoint, acting as entry point to the *cluster* [19, Ch. 8].

Typically, all ***Kubernetes** artifacts* are defined declaratively in one or multiple files, known as ***Kubernetes** manifests*. These *manifests* are applied to a *cluster* via the ***kube-apiserver*** [22], which provides the ***Kubernetes** API*, using the **`kubectl`** command [23].

b) Helm

Helm [24] bundles multiple ***Kubernetes** manifests* in so-called ***Helm** charts*, enabling efficient management and dynamic templating [19, Ch. 22]. When a ***Helm** chart* is applied to a *cluster*, a ***Helm** deployment* is created, allowing for organized modifications and updates to the contained artifacts.

Ready-to-use ***Helm** charts* for various applications can be obtained from ***Artifact Hub*** [25]. Thus, ***Helm*** functions as a package manager for ***Kubernetes***, similar to how **`apt`** [26] is used for ***Ubuntu*** [27] or **`dnf`** [28] for ***Red Hat Enterprise Linux*** [29].

c) Container Runtime

To run *containers* within a ***Kubernetes** cluster*, each host in the *cluster* must have a *container runtime* installed. The *container runtime* acts as the execution environment for *containers*. To create a *container*, a *container image* is required, serving as a blueprint that includes all necessary files, executables, and static configurations needed for the encapsulated *software* [19, Ch. 2].

Container images can be built from ***Dockerfiles*** [30] or pulled from *image* registries such as ***Docker Hub*** [31]. In ***Kubernetes***, an automated *image* pull is performed by default when a *pod* is created, and the specified *container image* is not already available within the *container runtime* [32].

2.2.2. Multi-Cluster Management

When managing not only multiple hosts within a single ***Kubernetes** cluster* but also multiple hosts across multiple *clusters*, the term *multi-cluster management* is commonly used in practice [33], [34], [35]. *Multi-cluster management* introduces various new challenges, including security and compliance, resource optimization, monitoring and logging, cross-cluster networking, and *service discovery* [33], [34], [35], [36]. This thesis specifically aims to evaluate solutions addressing issues related to *service discovery* and cross-cluster networking, particularly in terms of relaying the ***Kubernetes** API*.

a) Service Discovery

The term *service discovery* is defined as "a mechanism by which applications […] find each other's locations on the network" [37, p. 146]. Consequently, a *service discovery* system provides a way to map abstract addresses to concrete application endpoints and retrieve a list of all services. One of the best-known examples of a *service discovery* system is the *Domain Name System* (*DNS*), which resolves domain names of hosts on in the internet (or private networks) to their corresponding IP addresses [19, Ch. 7].

Within ***Kubernetes***, *service discovery* is facilitated through the *service* artifact. Each ***Kubernetes*** *service* maps to one or multiple *pods* and corresponding their ports, creating an entry within ***Kubernetes***' internal *DNS*, which is typically implemented using ***CoreDNS*** [38]. This internal *DNS* enables *pods* to resolve other *pods* and *services* within the *cluster*[1]. Each *service* can be accessed using its *service* name or in the format **`<service-name>.<service-namespace>`**, optionally followed by ".svc.cluster.local" to form a fully qualified hostname [37, p. 157].

When working with multiple ***Kubernetes*** *clusters*, two key *service discovery* challenges emerge: First, discovering *services* across multiple *clusters* may be required to enable inter-cluster communication. Second, *service discovery* for the *clusters* themselves is necessary to locate and address each *cluster* for management purposes. This thesis specifically focuses on evaluating solutions that address the second challenge.

b) Cross-Cluster Networking

Even if services can be discovered across multiple *clusters*, reliable network connectivity between *clusters* is still required for the *pods* to communicate. This can be challenging, as *clusters* may be distributed across various network segments or even geographical locations, potentially leading to connectivity and latency issues [34], [35].

These issues are already addressed by extensions for *Container Networking Interface* plugins [39] available for ***Kubernetes***, as well as by dedicated tools. These solutions interconnect *clusters* by, for example, creating overlay networks, providing gateways, establishing *VPN* tunnels, or introducing virtual networks on the application layer [40].

[1] This is an optional component. If no *DNS* is installed for a ***Kubernetes*** *cluster*, *services* can still be resolved using environment variables. For each ***Kubernetes*** *service*, the variables **`<service-name>_SERVICE_HOST`** and **`<service-name>_SERVICE_PORT`** are injected into the *pods* [37, p. 157].

To further narrow the scope of cross-cluster networking to a specific issue, this thesis addresses relaying of the ***kube-apiserver*** of multiple *clusters* to enable access to the ***Kubernetes** API* through unified endpoints. The objective is to ensure that every *cluster* within a given environment, such as a company, can be managed using the **`kubectl`** command from one or multiple centralized endpoints.

To provide a common term, this functionality is referred to as *peer-to-peer relaying* throughout this thesis. Each *cluster* acts as a peer, as it not only provides its own ***Kubernetes** API*, which can be relayed, but can also relay the *API* of another *cluster*.

2.3. Multi-Cluster Management Solutions

For the challenges outlined in Subsection 2.2.2, existing solutions aim to address these issues. Two such solutions are ***Rancher*** and ***kubeanchors***. Essentially, both solutions provide similar functionalities. However, ***Rancher*** follows a *centralized architecture*, whereas ***kubeanchors*** employs a *decentralized architecture*.

2.3.1. Rancher

"***Rancher*** is a Kubernetes management tool to deploy and run *clusters* anywhere and on any provider. ***Rancher*** can provision ***Kubernetes*** from a hosted provider, provision compute nodes and then install ***Kubernetes*** onto them, or import existing ***Kubernetes*** *clusters* running anywhere" [41].

One of ***Rancher***'s capabilities is its web-based *user interface* (*UI*), which simplifies the management and setup of ***Kubernetes*** *clusters*. Existing ***Kubernetes*** *clusters* can be joined to ***Rancher*** by deploying an agent, which connects to the ***Rancher*** server [42].

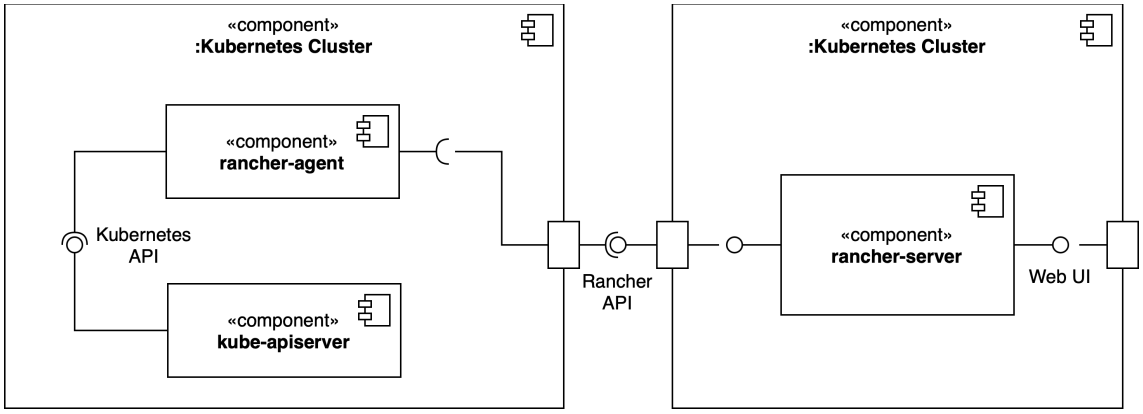


Figure 2.2: Rancher Components

Figure 2.2, leveraging ***UML*** and derived from [43], outlines the key components of ***Rancher***. The ***Rancher*** server is typically deployed on a dedicated ***Kubernetes*** *cluster*. Each *cluster* integrated into ***Rancher*** runs a ***Rancher*** agent, which acts as intermediary between the ***Rancher*** server and the ***kube-apiserver*** of the *cluster* where it is installed. *Clusters* running the ***Rancher*** agent are commonly referred to as *downstream clusters [44]*.

Downstream clusters imported into ***Rancher***, rather than being installed by ***Rancher***, must establish a connection to the ***Rancher*** server via *HTTPS*. Consequently, a *downstream cluster* must be able to actively reach the ***Rancher*** server for communication [45]. This applies to all *clusters* further considered within this thesis. Both, the ***Rancher*** *API* and ***Rancher*** *UI* provide the capability to discover the *downstream clusters* and to connect to their ***kube-apiserver*** via **`kubectl`**.

2.3.2. kubeanchors

As a lightweight alternative to ***Rancher***, ***kubeanchors*** is developed by the author of this thesis. Currently, ***kubeanchors*** is available as minimum viable product, providing *service discovery* for multiple ***Kubernetes*** *clusters*. Additionally, a concept for *peer-to-peer relaying* has already been developed but not yet been implemented.

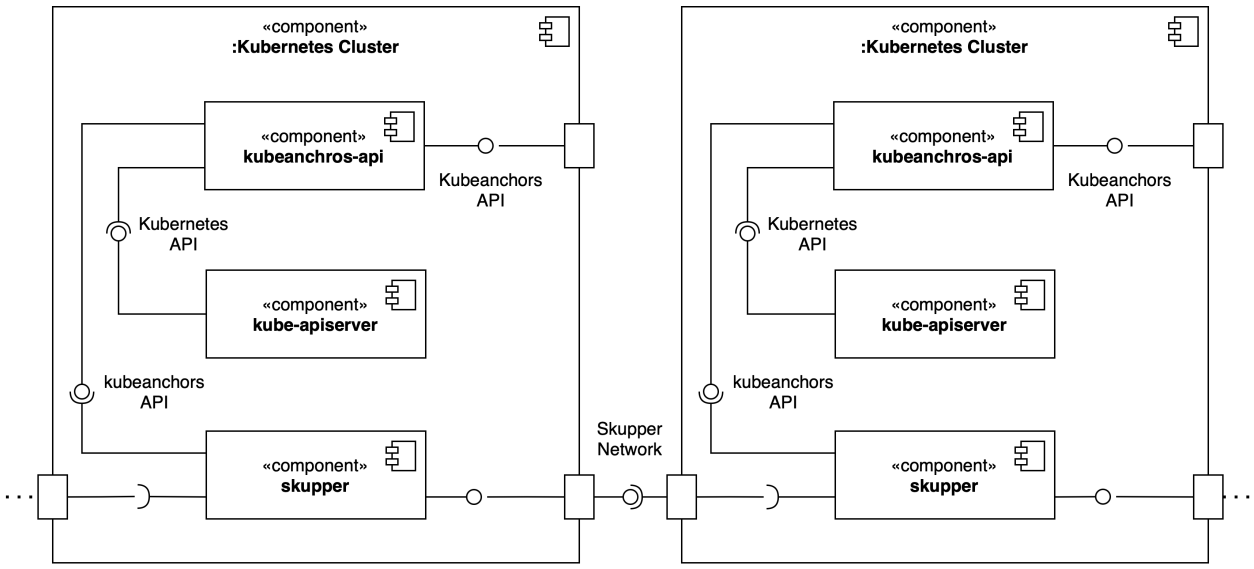


Figure 2.3: kubeanchors Components

kubeanchors is based on ***Skupper*** [46], which "enables secure communication across ***Kubernetes*** *clusters* with no *VPNs* or special firewall rules" [46]. To achieve this, ***Skupper*** establishes a *Virtual Application Network* (*VAN*). "*VANs* […] use […] *application routers* to route communication between […] *application addresses*" [47]. For ***kubeanchors***, ***Skupper*** is installed in each ***Kubernetes*** *cluster* that needs to be discovered, providing a unique *application address* per *cluster*.

Skupper enables ***Kubernetes*** *deployments*, and consequently the with the *deployment* associated *pods*, to be exposed to all other *clusters* within the *VAN*. To accomplish this, ***Skupper*** creates a ***Kubernetes*** *service* on each *cluster* within the *VAN*, which redirects traffic to the *cluster* containing the targeted *deployment.* As a result, the *pods* of the exposed *deployment* become reachable across all *clusters [48]*.

On top of ***Skupper***, the ***kubeanchors*** *API* is deployed to each *cluster* and exposed within the ***Skupper*** network to all other *clusters*. It connects to the local ***kube-apiserver*** to provide information about its *cluster* and displays the *application address* generated by ***Skupper***. The ***kubeanchors*** *APIs* running in other *clusters* can then collect this information.

Consequently, each ***kubeanchors** API* can query the ***kubeanchors** APIs* of other *clusters*, enabling decentralized *service discovery*. Additionally, ***Skupper*** can be leveraged to expose the ***kube-apiserver*** of a *cluster* within the *VAN*, making it accessible to other *clusters* and thereby providing *peer-to-peer relaying*.

Figure 2.3, leveraging ***UML***, presents the described ***kubeanchors*** components. The ***kubeanchors** API* is implemented in ***Python*** [49] using the ***FastAPI*** [50] library. The source code can be obtained from Appendix E.

3. Evaluation Model

To compare the proposed solutions to the initial problem, a suitable *evaluation model* is required. In general, the chosen model must be capable of establishing comparability between selected *quality criteria* of *evaluation targets*. In this case, the *evaluation targets* are *software* designed for *service discovery* and *peer-to-peer relaying* in ***Kubernetes*** *clusters*.

Various considerations have been introduced for assessing *software quality*, e. g.:

- ***ISO/IEC 25000*** introduces the ***Systems and software Quality Requirements and Evaluation (SQuaRE)*** [14] series, which consists of multiple standards (***ISO/IEC 25000*** – ***25099***) that define *quality criteria, quality* measurement, etc. concerning *software quality* [14, pp. 7–11].
- ***Bass et al.*** propose *quality criteria* in a scenario-based approach to ensure testability of *software quality* [15, pp. 41–44].
- The ***arc42 Quality Model (Q42)*** incorporates elements from both ***Bass et. al*** and the ***SQuaRE*** series but emphasizes practical application [18], [51].

These approaches will be leveraged in the development of the *evaluation model*. However, for full adoption as the *evaluation model,* the existing approaches are either too complex, overly extensive, or unsuitable. For example, ***Q42*** consists of 108 *quality criteria*, though only a subset may be relevant to the *evaluation model*. Determining which criteria are important will be addressed during the development process.

The ***SQuaRE*** series provides a framework for implementing *quality models* [16]. To ensure a standardized approach, this framework will serve as the foundation of the *evaluation model*. As a result, the developed model will be comparable and adhere to a well-defined structure.

Implementing ***ISO/IEC*** standards in derived models is a common practice. An example of this is the ***IT-Grundschutz*** [52] model developed by the ***German Federal Office for Information Security (BSI)*** [53], which is based on ***ISO/IEC 27001*** [54]. Similar approaches to implementing ***ISO/IEC 25000*** and its subordinate standards (***ISO/IEC 25001 – 25099***) have been proposed by ***Argotti et al.*** [55], ***Barletta et al.*** [56], ***Perdomo*** and ***Zapata*** [57], ***Polillo*** [58] and ***Ravanello et al.*** [59].

Bass et al. highlight the problem that definitions of *quality criteria* often lack testability. To resolve that issue, they introduce a scenario-based concept for the specification of *quality criteria* requirements [15, pp. 42–44]. The *evaluation model* will adopt this concept.

3.1. Model Structure

According to ***ISO/IEC 25002***, a "*quality model* is a defined set of *characteristics* […] that are quantified by *quality measures* that can be used to […] evaluate the *quality* properties of target entities" [16, p. 8]. If necessary, *characteristics* can be decomposed into *sub-characteristics* "that collectively cover […] that […] *characteristic*" [16, p. 8], forming a hierarchic structure. The leaf (*sub-) characteristics* linked to a *quality measure* reassemble *quality criteria*.

For the structure of the *evaluation model*,

- a set of *characteristics* and, if necessary, *sub-characteristics* that describe the *evaluation targets* **must** be chosen,
- for each *characteristic* and/or *sub-characteristic* at least one *quality measure* **must** be selected,

to conform ***ISO/IEC 25002*** [16]. The following figure derived from the given standard [16, p. 9] outlines the explained structure:

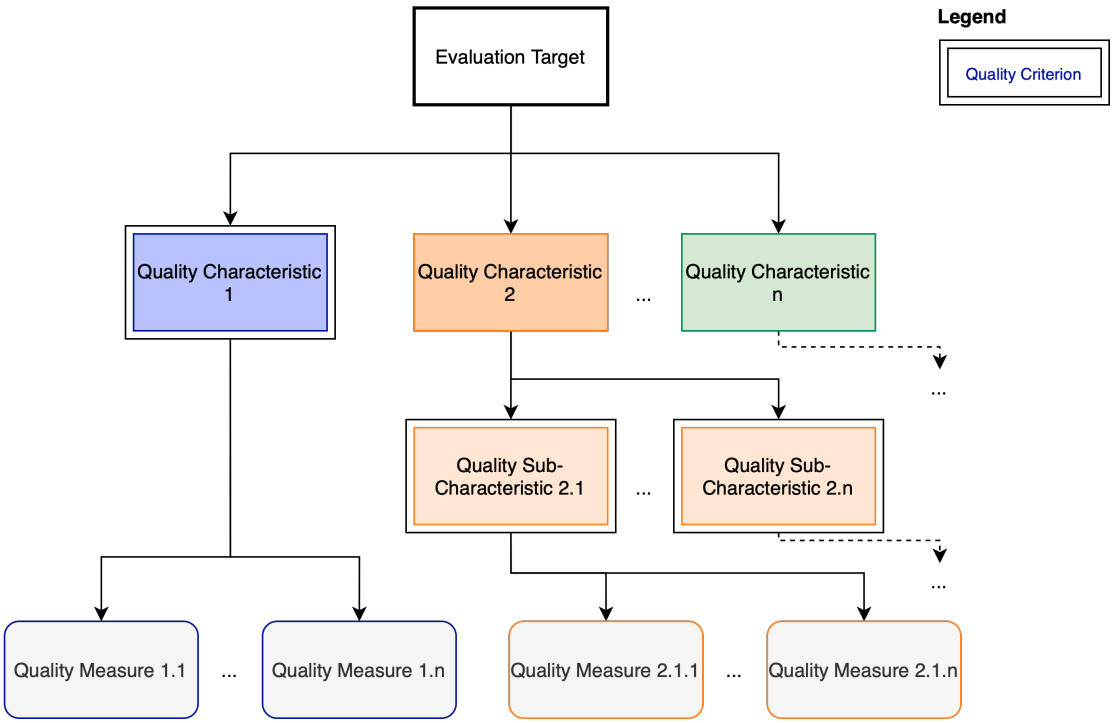


Figure 3.1: Evaluation Model Structure

3.2. Stakeholders

ISO/IEC 25002 states that the importance of *quality criteria*, and consequently the overall perceived *quality* of a *software system*, may vary depending on the *stakeholder* [16, Ch. 8.1]. Additionally, ***Q42*** proposes the *stakeholder* as a dimension of *quality* [17]. Therefore, the following *stakeholders*[2] will be considered for the *evaluation model*:

3.2.1. Cluster Administrator

A *cluster administrator* is responsible for maintaining the functionality of ***Kubernetes*** *clusters*, including networking, storage, and *container runtime*, from the perspective of the *cluster*. Their key responsibilities encompass *cluster* setup and maintenance, resource management, networking, security, storage, and backup and disaster recovery. In *multi-cluster management* contexts, they handle tasks such as *cluster* provisioning and orchestration, cross-cluster networking, centralized governance, and *service discovery*.

The *cluster administrator* corresponds to the *stakeholder* group referred to as “architects, developers, maintainers, and system integrators” [16, p. 14] in ***ISO/IEC 25002***. Skills relevant to this role are typically covered by the ***Certified Kubernetes Administrator Certification*** [60] provided by the ***Linux Foundation*** [61].

3.2.2. Cluster User

The *cluster user* interacts with ***Kubernetes*** *clusters* to run and manage applications. Unlike a *cluster administrator*, a *cluster user* does not manage infrastructure-level tasks but instead focuses on deploying, scaling, monitoring, and troubleshooting applications within the *cluster*. In *multi-cluster management* scenarios, *cluster users* frequently access various environments, such as development, testing, and staging. They require the ability to discover and connect to *clusters* and applications across these environments.

The *cluster user* aligns with the *stakeholder* group referred to as “users” [16, p. 13] in ***ISO/IEC 25002***. Skills relevant to this role are typically covered by the ***Certified Kubernetes Application Developer Certification*** [62] offered by the ***Linux Foundation***.

[2] In general, the *evaluation model* is not limited to the given *stakeholders* and can be extended.

3.3. Quality Criteria

For the *evaluation model*, a set of *quality criteria* must be selected to encapsulate the relevant attributes of the *evaluation targets* within the given context. To ensure a standardized approach, these criteria are chosen from the ***ISO/IEC 2501n*** division[3]. However, two key research questions need to be addressed:

(R1) Which of the given *quality criteria* from the ***ISO/IEC 2501n*** division are most relevant to applications providing *service discovery* and *peer-to-peer relaying* in ***Kubernetes*** *clusters*?

(R2) How should the chosen *quality criteria* be prioritized based to the perspective of the *stakeholders* identified in Section 3.2?

To answer these questions, a literature review will be conducted (analogous to [55] and [65, Ch. 3–4]). This review will analyze the statistical occurrence of the criteria outlined in the ***ISO/IEC 2501n*** division within literature focused on *service discovery* and *peer-to-peer relaying* in ***Kubernetes*** *clusters*. For a transparent, reproduceable, and comprehensive approach, the first five steps of the methodology proposed by ***Greetham*** [66, pp. 11–21] will be applied:

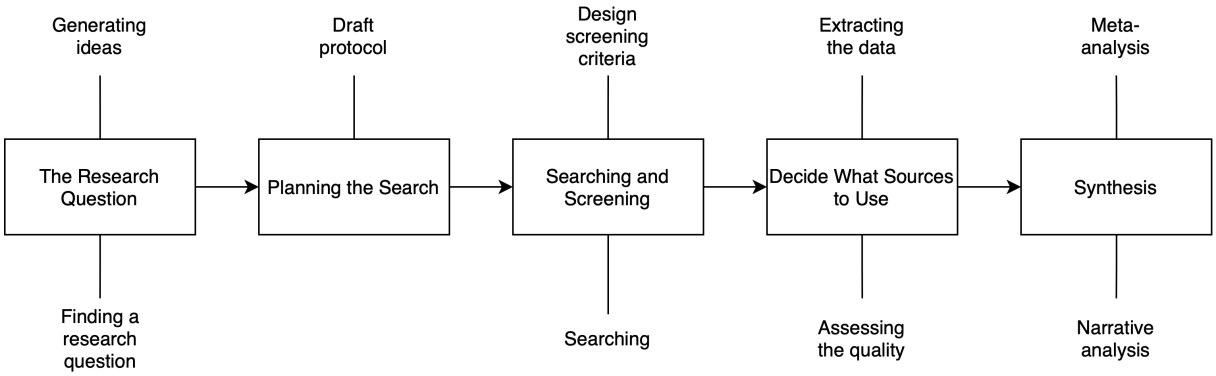


Figure 3.2: Literature Review Steps

a) Step 1: The Research Question

The research questions have already been implied previously. Thus, no additional work is required for this step.

b) Step 2: Planning the Search

To identify relevant literature, the databases ***IEEE Xplore*** [67], ***Scopus*** [68], and ***Web of Science*** [69] will be utilized. In each database, the search will encompass all metadata/fields available.

[3] As the given *evaluation targets* are software solutions, the respective standards are restricted to ***ISO/IEC 25010*** [63] and ***ISO/IEC 25019*** [64].

A preliminary search using search strings related to the ***SQuaRE*** series, such as

```
“Kubernetes” AND (“ISO/IEC 250*” OR “ISO 250*”)
```

Code 3.1: Preliminary Search String

produce limited results (< 5 per database). Therefore, a broader search strategy will be adopted, acknowledging a specific reference to the ***ISO/IEC*** standards will be neglected. Instead, the reference will be established within the further steps of the review. The revised search string is as follows:

```
“Kubernetes” AND (
      “service discovery” OR “service registry”
      OR
      “p2p” OR “peer-to-peer” OR “peer to peer”
      OR
      “multi cluster” OR “multi-cluster” OR “multicluster”
)
```

Code 3.2: Search String

The following table outlines the structure of the search string:

Part	Explanation
“Kubernetes”	Limits the search to articles specifically related to ***Kubernetes***.
AND (	Combines the additional keywords for contextualization.
“service discovery” OR “service registry”	Targets articles discussing *service discovery* with “service registry” included as potential synonym.
OR	Indicates alternate terms.
“p2p” or “peer-to-peer” OR “peer to peer”	Focuses on articles about *peer-to-peer relaying*, omitting “relaying” for more results; includes abbreviations and alternate spellings.
OR	Indicates alternate terms.
“multi cluster” OR “multi-cluster” OR “multicluster”)	Covers articles related to *multi-cluster management* for a broader search, omitting “management” for more results; includes alternate spellings.
)	Closes the contextualized search string.

Table 3.1: Search String

The search results will undergo a four-step review process: starting with filtering by type, language, and availability, followed by the removal of duplicates, then screening by title and abstract, and finally, a full-text review. The first step will be handled within the search engines of the databases[4], and the last step will be automated, while the remaining steps will be executed manually. Screening will be based on the following criteria:

(C1) The document **must** be a book (chapter), journal article, or conference/proceeding paper.

(C2) The language of the document **must** be English.

(C3) Access to a full-text *PDF* file **must** be available through the respective database. This also includes references to third-party sources.

(C4) The literature **must** mainly[5] focus on ***Kubernetes*** and address at least *service discovery*, networking[6] or *multi-cluster management*.

(C5) The full-text *PDF* file **must** be readable by the ***PyMuPDF*** [70] ***Python*** library.

(C6) The content **must** allow conclusions related to *quality criteria* defined in ***ISO/IEC 25010*** [63] or ***ISO/IEC 25019*** [64].

Criteria (C1) – (C3) will be applied in the filtering step, criterion (C4) in the screening step and criteria (C5) – (C6) in the full-text review. To summarize, the figure below provides an overview of the review process:

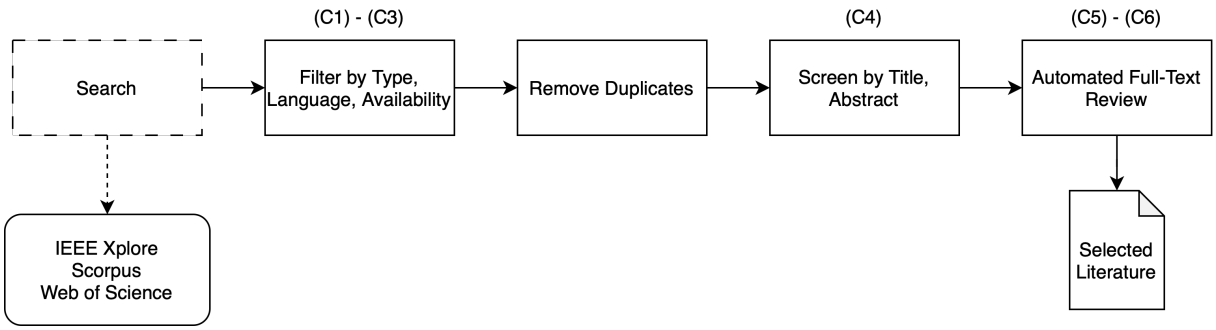


Figure 3.3: Search and Screening Process

The review process will be documented in an ***Excel*** [71] file, with found literature imported into ***Zotero*** [72] to ensure efficient tracking and management. The full-text review will leverage automation through ***Python*** and the ***PyMuPDF*** library. For transparency, the ***Excel*** file, a ***Zotero*** database export, and the ***Python*** code will be referenced in Appendix C.

[4] ***IEEE Xplore*** only publishes English articles, so no filtering by language is necessary.

[5] In this context, "mainly" indicates that the literature is primarily focused on addressing a ***Kubernetes***-related topic or on extending ***Kubernetes***. Literature that merely uses ***Kubernetes*** as a tool is excluded.

[6] The topic “networking” is applied as a less stringent criterion compared to “peer-to-peer relaying” to yield a greater number of matches. This is valid, as *peer-to-peer relaying* is a subset of networking.

c) Step 3: Searching and Screening

IEEE Xplore, ***Scopus***, and ***Web of Science*** were queried on the January 14, 2025, using the specified search string (refer to Code 3.2). Applying the outlined filters yielded 56 results from ***IEEE Xplore***, 638 from ***Scopus***, and 42 from ***Web of Science***, resulting in a total of 736 documents that satisfied at least criteria (C1) – (C3). After deduplication, 654 documents remained.

The subsequent manual screening of titles and abstracts, based on criterion (C4), further reduced the selection to 71 documents. During this screening process, the review of titles and abstracts was halted as soon as it became clear that the document had no relation to ***Kubernetes***, in order to accelerate the process.

d) Step 4: Decide What Sources to Use

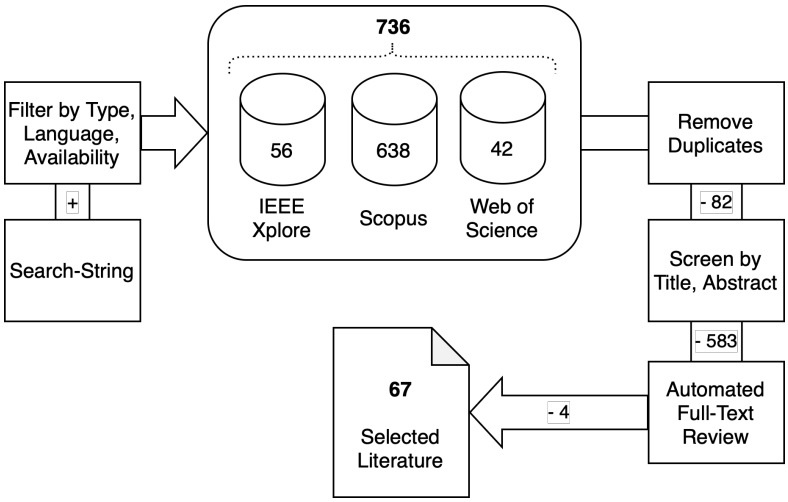


Figure 3.4: Literature Review Results

The full-text *PDF* files for the remaining 71 documents were downloaded. Due to inconsistent quality of the *PDF* metadata across different sources, ***ExifTool*** [73] was used to ensure that each file had at least an appropriate "title" metadata field. Additionally, *PDF* files containing whole magazine issues were cropped to include only the relevant articles using the ***Apple Preview*** [74] app.

To streamline the processing of the large number of documents, an automated full-text screening using ***Python*** was conducted. The text extraction from the *PDF* files was carried out with the ***PyMuPDF*** library[7]. The extracted text was then normalized by converting all characters to lowercase, removing hyphens and newlines, consolidating multiple whitespaces into single spaces, and discarding any content following and including the last occurrence of the word "references"[8].

[7] ***PyMuPDF*** was chosen from the results of a benchmark that compared various *PDF* libraries for ***Python*** [75].

[8] The references were discarded to minimize noise in the subsequent statistical analysis of word occurrences.

During text extraction, one document was excluded as it was unreadable by ***PyMuPDF***, violating criterion (C5). For the remaining readable documents, the extracted and normalized text was scanned for *quality criteria* from ***ISO/IEC 25010*** and ***ISO/IEC 25019*** through word matching. At this stage, three additional documents were excluded for containing no matches, thus violating criterion (C6). In sum, 67 documents were chosen. The results of the full-text scan were written to *CSV* files and imported into the documentary ***Excel*** file for further analysis within the next step.

e) Step 5: Synthesis

The *PDF* content-scan also determined the count of documents in which each *quality criterion* was mentioned. This resulted in a meta-analysis of the chosen literature. From the number of occurrences, a coverage metric was calculated for each *quality criterion*. The following table shows all *quality criteria* that appeared at least once[9]:

Model	Characteristic	Sub-Characteristic	Count of Docs.	Coverage
ISO/IEC 25010	reliability	**availability**	51	76.12%
ISO/IEC 25010	flexibility	**scalability**	45	67.16%
ISO/IEC 25010	performance efficiency	**resource utilization**	34	50.75%
ISO/IEC 25019	acceptability	**experience**	32	47.76%
ISO/IEC 25010	performance efficiency	**capacity**	30	44.78%
ISO/IEC 25010	reliability	**fault tolerance**	18	26.87%
ISO/IEC 25019	acceptability	**compliance**	13	19.40%
ISO/IEC 25010	compatibility	**interoperability**	9	13.43%
ISO/IEC 25010	interaction capability	**operability**	9	13.43%
ISO/IEC 25010	flexibility	**adaptability**	7	10.45%
ISO/IEC 25019	beneficialness	**usability**	7	10.45%
ISO/IEC 25010	security	**integrity**	6	8.96%
ISO/IEC 25010	maintainability	**modularity**	5	7.46%
ISO/IEC 25019	beneficialness	**suitability**	5	7.46%
ISO/IEC 25010	compatibility	**co-existence**	2	2.99%
ISO/IEC 25010	maintainability	**reusability**	2	2.99%
ISO/IEC 25010	reliability	**recoverability**	1	1.49%
ISO/IEC 25010	security	**accountability**	1	1.49%

[9] The full table, including *sub-characteristics* that did not appear in any document, is provided in Appendix C.

ISO/IEC 25010	security	**authenticity**	1	1.49%
ISO/IEC 25010	security	**resistance**	1	1.49%
ISO/IEC 25010	maintainability	**testability**	1	1.49%
ISO/IEC 25019	beneficialness	**accessibility**	1	1.49%

Table 3.2: Quality Criteria Coverage

Finally, the initial research questions can be addressed based on the results:

(R1): The *quality criteria* most relevant to applications providing *service discovery* and *peer-to-peer relaying* in ***Kubernetes*** *clusters* are the *sub-characteristics* from the ***ISO/IEC 2501n*** division that appeared at least once in the analyzed documents.

(R2): Since all analyzed documents demonstrate a relation to ***Kubernetes***, it is assumed that they align with the needs of the mentioned *stakeholders* in Section 3.2. Consequently, the coverage metric will be used to prioritize the *quality criteria* within the *evaluation model*.

For a definition of each *quality criterion* refer to ***ISO/IEC 25010*** [64, Ch. 3] or ***ISO/IEC 25019*** [64, Ch. 3].

3.4. Quality Measures

To quantify *quality criteria* during the evaluation of *evaluation targets*, *quality measures* will be employed. As defined in the ***ISO/IEC 2502n*** division, a *quality measure* "is a measurement function of two or more values of *quality measure elements*" [76, p. 4]. Moreover, *quality measure elements* are defined as "a property and the measurement method for quantifying it, including optionally the transformation by a mathematical function" [76, p. 4].

In simpler terms, a *quality measure element* is identified by systematically quantifying a specific property of an *evaluation target*. By combining multiple *quality measure elements* through a mathematical function, a *quality measure* is created. Table 3.3, derived from ***ISO/IEC 25023*** [77, App. A], lists the *quality measures* selected for the *evaluation model*, arranged by *sub-characteristic* priority. Further details on each *quality measure*, including their respective mathematical formulas and associated *quality measure elements*, can be referenced from the ***ISO/IEC 2502n*** division.

To ensure conformance, the selection of *quality measures* will adhere to the requirements specified in ***ISO/IEC 25022*** [76, Ch. 2] and ***ISO/IEC 25023*** [77, Ch. 2][10]. Justifications for inclusion or exclusion of certain measures can be obtained from Appendix D.

Model	Sub-Characteristic	Quality Measure Model	Quality Measures[11]		
ISO/IEC 25010	**availability**	ISO/IEC 25023	system availability	G	HR
			mean down time	S	R
ISO/IEC 25010	**scalability**	ISO/IEC 25023	undefined[12]		
ISO/IEC 25010	**resource utilization**	ISO/IEC 25023	mean processor utilization	G	HR
			mean memory utilization	G	R
			mean I/O devices utilization	G	R
			bandwidth utilization	S	UD
ISO/IEC 25019	**experience**	ISO/IEC 25022	undefined[12]		
ISO/IEC 25010	**capacity**	ISO/IEC 25023	transaction processing capacity	G	R
			user access capacity	G	R

[10] As the given *evaluation targets* are software solutions, ***ISO/IEC 25024*** [78] is omitted.

[11] G = General, S = Special. HR = Highly recommended, R = Recommended, UD = Used at users' discretion. Compare to [77, p. 31].

[12] ***ISO/IEC 25010*** and ***ISO/IEC 25019*** were reintroduced in 2023 whereas ***ISO/IEC 25023*** and ***ISO/IEC 25022*** have remained unchanged since 2016. No *quality measure* has been defined for this *sub-characteristic* to date.

ISO/IEC 25010	**fault tolerance**	ISO/IEC 25023	failure avoidance	G	HR
			redundancy of components	S	R
ISO/IEC 25019	**compliance**	ISO/IEC 25022	undefined[12]		
ISO/IEC 25010	**interoperability**	ISO/IEC 25023	data formats exchangeability	G	HR
			data exchange protocol sufficiency	G	R
			external interface adequacy	S	HR
ISO/IEC 25010	**operability**	ISO/IEC 25023	operational consistency	G	HR
			message clarity	G	R
			monitoring capability	S	UD
ISO/IEC 25010	**adaptability**	ISO/IEC 25023	hardware environmental adaptability	G	HR
			system software environmental adaptability	G	HR
ISO/IEC 25019	**usability**	ISO/IEC 25022	undefined[12]		
ISO/IEC 25010	**integrity**	ISO/IEC 25023	data integrity	G	HR
			internal data corruption prevention	G	R
ISO/IEC 25010	**modularity**	ISO/IEC 25023	coupling of components	G	R
ISO/IEC 25019	**suitability**	ISO/IEC 25022	undefined[12]		
ISO/IEC 25010	**co-existence**	ISO/IEC 25023	co-existence with other products	G	HR
ISO/IEC 25010	**reusability**	ISO/IEC 25023	reusability of assets	G	HR
ISO/IEC 25010	**recoverability**	ISO/IEC 25023	mean recovery time	G	HR
			backup data completeness	S	R
ISO/IEC 25010	**accountability**	ISO/IEC 25023	user audit trail completeness	G	HR
			system log retention	S	R
ISO/IEC 25010	**authenticity**	ISO/IEC 25023	authentication mechanism sufficiency	G	HR
ISO/IEC 25010	**resistance**	ISO/IEC 25023	undefined[12]		
ISO/IEC 25010	**testability**	ISO/IEC 25023	test function completeness	G	R
			autonomous testability	G	UD
ISO/IEC 25019	**accessibility**	ISO/IEC 25022	undefined[12]		

Table 3.3: Quality Measures

3.5. Quality Measurement Scenarios

Now that the necessary *quality criteria* and their corresponding *quality measures* have been identified, the context for applying these measures remains unclear. To address this, the *evaluation model* adopts *quality measurement scenarios* as introduced by ***Bass et al.*** (originally referred to as "quality attribute scenarios" [15, p. 42]). This concept offers the advantage of making *quality criteria* not only measurable but also testable. Additionally, *quality measurement scenarios* provide a more concrete interpretation of *quality criteria* than their abstract definitions do, and they ensure idempotency in the evaluation process when they are repeated.

By definition [15, pp. 42–43], a *quality measurement scenario* consists of six parts:

- A ***stimulus*** is an event that impacts the *evaluation target* in a specific way. It can manifest through various means, such as user or machine operation, cyber-attacks, or system modifications.
- Each *stimulus* originates from a ***stimulus source***, such as a user or another system. The *evaluation target* may handle the *stimulus* differently depending on its source.
- The *stimulus* is directed at an ***artifact***, which in this context refers to an *evaluation target*. By definition, the *artifact* can be specified more precisely, such as a particular system component, if needed.
- When a *stimulus* is directed at an *artifact*, the *artifact* generates a ***response***. They should not be considered purely technical. *Responses* can also include system modifications made by developers.
- The generation of a *response* must be quantifiable using a ***response measure*** to ensure testability. In this context, a *response measure* will be represented by one of the previously chosen *quality measures*.
- A *quality measurement scenario* occurs within a defined ***environment***, specifying the state of the *evaluation targets* and influencing their behavior on a *stimulus*. A unified *evaluation environment* is proposed to ensure consistent conditions and reproduceable scenarios.

The following figure, derived from ***Bass et al.*** [15, p. 44] outlines the explained structure:

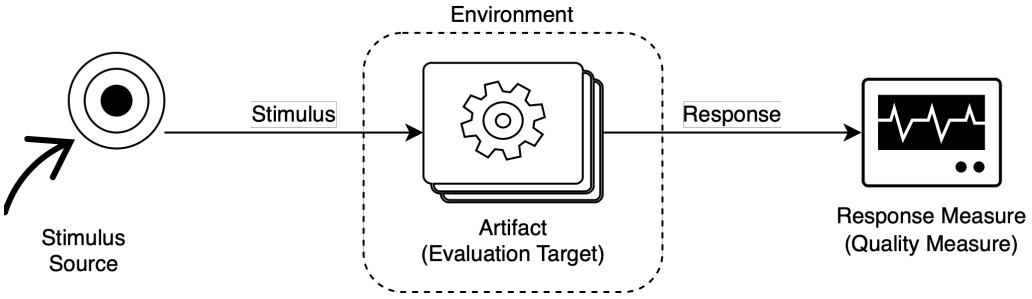


Figure 3.5: Quality Measurement Scenario

3.6. Evaluation Environment

To ensure consistent conditions for comparative analysis, a unified *evaluation environment* is proposed. This environment is based on the authors experience and replicates common *multi-cluster management* challenges, including hierarchically arranged subnets, network traffic restrictions due to firewalls, routing, etc., and variations in ***Kubernetes*** distributions.

The *evaluation environment* is structured into three vertical levels, each of which can be further divided into multiple segments, forming hierarchic structure. Network traffic between these levels and segments is controlled by a set of predefined rules.

The proposed environment serves as a conceptual blueprint for actual implementation. The following figure provides an overview of the environment:

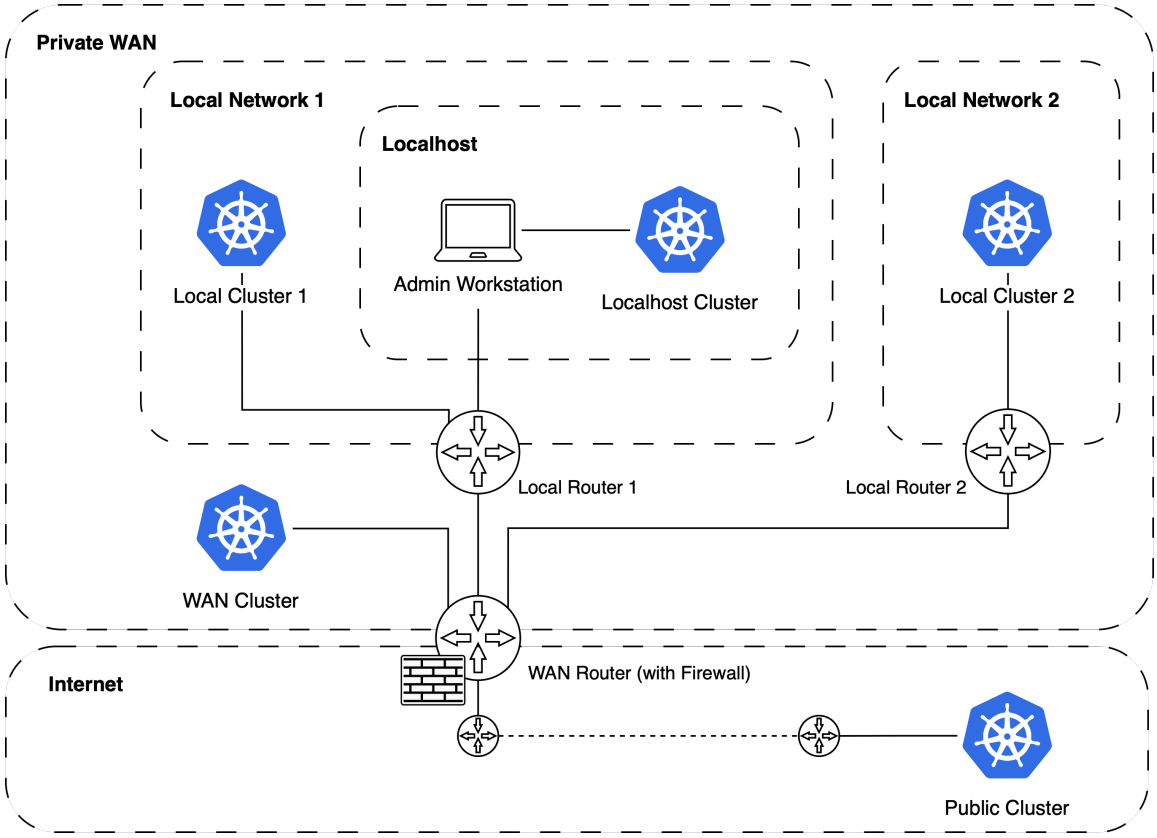


Figure 3.6: Evaluation Model

3.6.1. Structure

a) Level 0: Localhost

Level 0 represents the host-network of an admin workstation. This level is typically used for development purposes, including hosting short lived, self-managed ***Kubernetes clusters*** with tools like ***kind*** [79] or ***minikube*** [80].

b) Level 1: Local Network

In level 1, there are two horizontally divided networks: local network 1 and local network 2. Local network 1 contains the direct neighbors of the admin workstation, representing the subnet in which the workstation is located. Local network 2 is an adjacent subnet. To access it from local network 1, routing through level 2 is required. The environment must allow traffic to pass between local network 1 and local network 2. Within each local network, a self-managed ***Kubernetes** cluster* is running.

c) Level 2: Private WAN

The next higher level is the private wide area network (private WAN), where the subnets of level 1 are routed. The *evaluation environment* must allow traffic to pass between level 1 and level 2. Within the private WAN, another self-managed ***Kubernetes*** is running.

d) Level 3: Internet

Lastly, level 3 represents the internet. All outgoing traffic from level 2 to level 3 is allowed[13], while incoming traffic to the private WAN is restricted (by a firewall) to connections initiated from hosts in level 2. On the internet, a managed ***Kubernetes*** is running which is exposed to be accessible from the lower levels.

3.6.2. Furter Recommendations

(E1) To ensure traceability, the *evaluation environment* implementation **must** be documented.

(E2) Different distributions of ***Kubernetes*** **should** be used across level 0, level 1 and 2, and level 3:

a. Level 0: Self-managed distributions suited for development purposes, such as ***kind*** or ***minikube***, are recommended.

b. Level 1 and 2: These should implement self-managed production-ready distributions like, ***MicroK8s*** [7], ***K3s*** [8] or ***Kubernetes*** via ***kubeadm*** [81].

c. Level 3: A managed distribution, such as ***AKS*** [4], ***EKS*** [5] or ***GKE*** [6], is proposed.

(E3) To increase complexity, different ***Kubernetes*** versions **can** be used.

[13] Due to transitivity, this also implies that outgoing traffic from level 0 and level 1 to level 3 is allowed.

3.7. Conclusion

Altogether, a structured *evaluation model* was developed to assess *software* for *service discovery* and *peer-to-peer relaying* in ***Kubernetes*** *multi-cluster management*. To ensure comparability, the model is built upon standardized frameworks: its structure follows ***ISO/IEC 25002***, *quality criteria* are systematically derived from ***ISO/IEC 25010*** and ***ISO/IEC 25019***, and corresponding *quality measures* are selected from ***ISO/IEC 25023.*** Following the proposal by ***Bass et al.***, the *quality measures* will be captured within *quality measurement scenarios*. A unified *evaluation environment* is proposed to ensure consistent conditions for a comparative analysis.

For conformity with the *evaluation model*, the proposed elements must be considered. The model intentionally allows flexibility for custom implementations of both the *evaluation model* and the identification methods for the *quality measures*. To ensure comparability across different implementations, proper documentation still is necessary.

4. Evaluation

As an example of application of the proposed *evaluation model*, it is applied to two concrete solutions for ***Kubernetes** multi-cluster management*: ***Rancher*** and ***kubeanchors***. ***Rancher*** employs a *centralized architecture* to manage multiple ***Kubernetes** clusters*, whereas ***kubeanchors*** follows a fully *decentralized architecture* to achieve the same objective. Both solutions enable *service discovery* and relaying of the ***kube-apiserver*** for *clusters* within a defined scope. For more information about the *evaluation targets*, refer to Section 2.3.

The evaluation process incorporates four tasks. First, the concrete implementation of the *evaluation environment* is specified. Second, *quality measurement scenarios* are defined to assess the *quality measures*. Third, the implementations of the *quality measurement scenarios* are described. Finally, the scenarios are executed, and the results are presented.

To ensure conformity with the *evaluation model*, each step is documented. Due to the time constraints for this thesis, only the highly recommended *quality measures* (refer to Table 3.3) will be applied. Additionally, *quality measures* that rely on a technical specification of the *evaluation target* are omitted. As ***kubeanchors*** is developed by the author of this thesis, an unbiased assessment of these measures is not feasible, since the specifications of ***kubeanchors*** could be adjusted to comply the measures. These omitted *quality measures* are:

- co-existence with other products
- data formats exchangeability
- external interface adequacy
- operational consistency
- authentication mechanism sufficiency
- reusability of assets

Nevertheless, the methodology of this evaluation can be considered as best practice for applying the *evaluation model* and can be adopted for further *evaluation targets*.

4.1. Evaluation Environment Implementation

For the execution of the evaluation, a concrete implementation of the environment proposed in Section 3.6 is introduced. This Section is divided into four Subsections: the first explains the toolchain used to set up the *evaluation environment*, the second defines common components of the environment and the last two detail the individual adjustments required for each *evaluation target*. Within this Section, figures leveraging ***UML*** are presented to outline the deployed components.

4.1.1. Toolchain

At its core, the used toolchain to implement the *evaluation environment* is composed of ***Terraform*** [82] and ***Ansible*** [83]. First, a set of ***Terraform*** code is executed against a ***Red Hat OpenStack Platform*** (***OpenStack***) [84] to provide *virtual networks*, *routers*, *routing rules*, *floating IPs* and *machines*. ***Terraform*** then generates an ***Ansible*** inventory file, which is used to connect to the *virtual machines* and install the required applications on these hosts. Once the applications are deployed via ***Ansible***, the deployment can be validated using tests implemented in ***Python*** with the ***Testinfra*** [85] framework.

This approach was chosen to ensure a reproduceable implementation, following the *infrastructure-as-code* methodology. It results in a set of files that define the desired state of the *evaluation environment* implementation. The code can be executed multiple times, consistently producing the desired state of the *evaluation environment*. Further details, including used *software* versions, dependencies and the implementation code, can be found in Appendix E.

To overcome access restrictions to additional platforms, such as ***Microsoft Azure*** [86], and reduce complexity, level 0 and level 3 of the initially stated *evaluation environment* are omitted in this implementation. This omission also simplifies the measurement process, making it easier to reproduce.

4.1.2. Common Components

For both *evaluation targets*, a common deployment providing base functionalities, including *SSH* access, *DNS*, and internet access, is introduced. In accordance with the *evaluation environment* definition, three *virtual networks* are created and interconnected by *virtual routers*. Additionally, the WAN router is connected to the *virtual network* “public”, which acts as gateway to the internet in ***OpenStack***. The routers are configured with static routes to enable traffic flow between the local networks, the WAN, and the internet.

Within each deployed *virtual network*, a *virtual machine* is provisioned for hosting ***Kubernetes***. All *virtual machines* run on the ***Ubuntu*** 24.04 cloud image [87]. An additional management server in the WAN acts as intermediate to access the servers within the *virtual networks*. This server has a *floating IP* assigned, making it is accessible outside of the virtualization platform.

The management server runs a ***BIND*** [88] *DNS* service to provide name resolution for the *virtual networks*. ***BIND*** is configured to forward requests to ***Google***'s [89] public *DNS* [90], except for the domain "kubeanchors.internal". All other hosts are set to use the management server for *DNS* resolution.

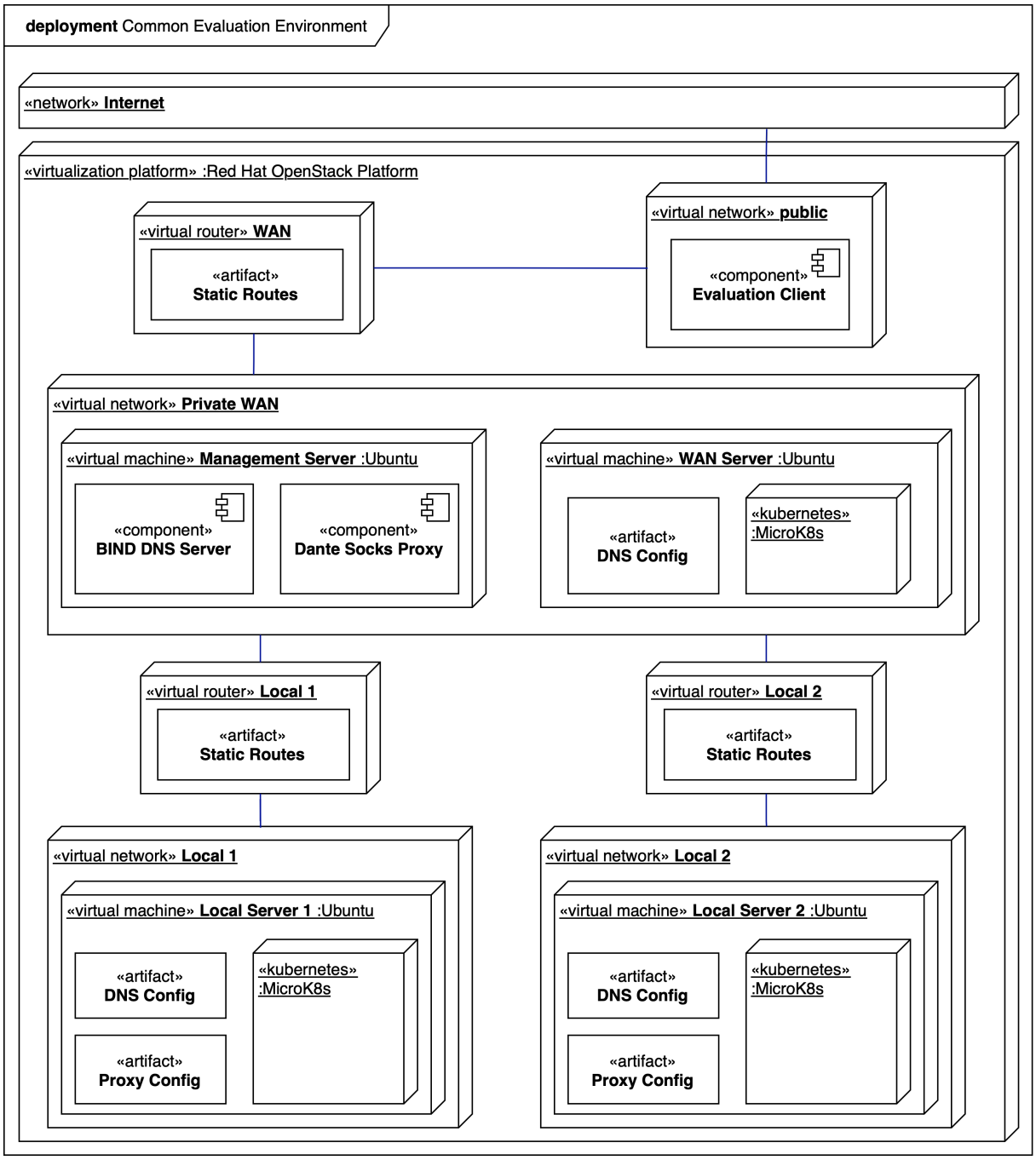


Figure 4.1: Common Evaluation Environment Deployment

All internet connections from the local networks must be proxied through the WAN. For this purpose, a ***Dante*** [91] *SOCKS* proxy is installed on the management server. All other hosts are configured to make use of the *SOCKS* proxy for *HTTP(S)*.

Except for the management server, ***MicroK8s*** v1.32.1 is installed on each host. The following plugins are enabled to ensure the base functionality of the ***Kubernetes*** *cluster*: **`hostpath-storage`**, **`rbac`**, **`metallb`**, and **`metrics-server`**.

The client to evaluate the given solutions runs within the virtual network “public”. From there, it can access the management server, which enables propagation to all other *virtual machines*.

Figure 4.1 summarizes all specifications. Further details, including the *software* versions used, can be found in Appendix E.

4.1.3. Rancher

For ***Rancher***, an additional ***MicroK8s*** *cluster* is deployed on the management server, as ***Rancher***’s *architecture* requires a centralized management cluster to supervise its *downstream clusters*. Due to restrictions to the ***Kubernetes*** version for the ***Rancher*** management cluster, the ***MicroK8s*** on the management server runs on version v1.31.5.

Rancher is deployed on this *cluster* using the ***Helm*** *chart* “rancher-latest/rancher” [92], which currently provides ***Rancher*** version 2.10.2. The ***Rancher*** server is exposed through a ***Kubernetes*** *ingress* to be reachable via *HTTPS*. For this purpose, the ***MicroK8s*** *ingress* plugin is enabled in addition to the same plugins running on the other ***MicroK8s*** *clusters*. It is important to note, that ***Rancher*** also can be deployed on a ***Kubernetes*** *cluster* spanning multiple hosts, which would improve reliability.

After setting up the ***Rancher*** server, again ***Terraform*** is used to register a lease for each ***Kubernetes*** *cluster* running on the other hosts. This creates an *API* endpoint on the ***Rancher*** server, allowing each *downstream cluster* to join. The *API* endpoints provide ***Kubernetes*** *manifests* that deploy a ***Rancher*** agent on each *downstream cluster*. Once deployed, the ***Rancher*** agents connect to the ***Rancher*** server, enabling *cluster* management through its *API*.

In summary, a centralized ***Rancher*** server is deployed on the management server, while the other *clusters* are joined to it by deploying ***Rancher*** agents on each target. This setup provides the discovery and relaying the ***kube-apiserver*** of the target *clusters* through the ***Rancher*** server. Figure 4.2 provides an overview of this deployment. Additional deployment details can be found in Appendix E.

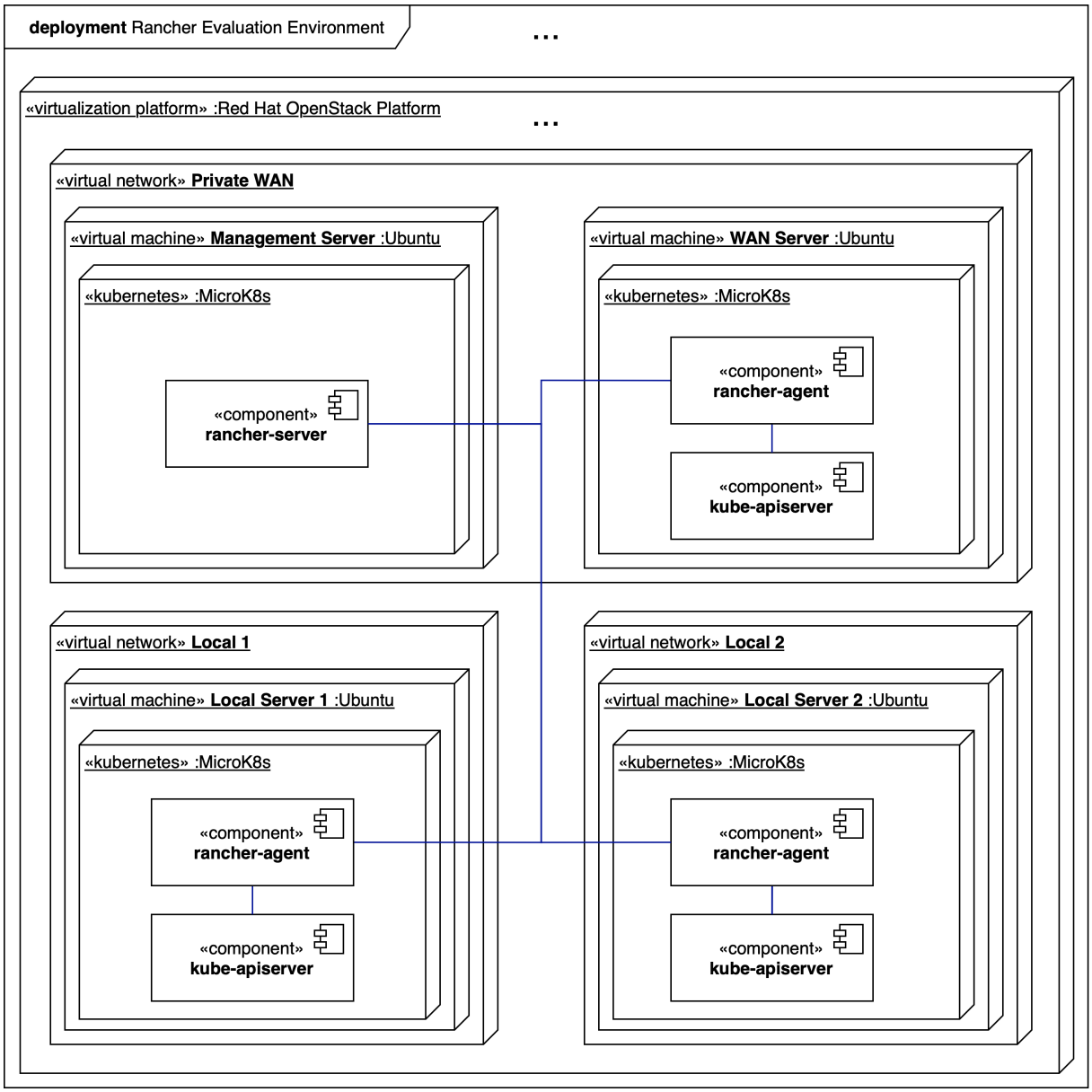


Figure 4.2: Rancher Evaluation Environment Deployment

4.1.4. kubeanchors

For ***kubeanchors***, ***Skupper*** is used to interconnect all ***MicroK8s clusters***. To set up ***Skupper***, the ***Skupper CLI*** is installed on each target host. The *CLI* is then used to deploy ***Skupper***'s components within the ***MicroK8s clusters***.

On the WAN server two ***Skupper*** join tokens are created and subsequently copied to the local servers allowing them to establish a link to the WAN server. As a result, the WAN server and the local servers form a hierarchic topology. It is worth noting that the local clusters additionally could be directly connected, forming a ring topology to enhance improved reliability.

Once the *clusters* are linked, the ***kubeanchors** API* is installed on each host and exposed to the other hosts by annotating the ***Kubernetes** deployments*. ***Skupper*** detects the annotated *deployment* and automatically creates a ***Kubernetes** service* on each host, redirecting request through the ***Skupper*** network to the corresponding host where the ***kubeanchors** API* for that *deployment* is running. This setup enables the ***kubeanchors** APIs* to query each other in a stateless manner to discover themselves.

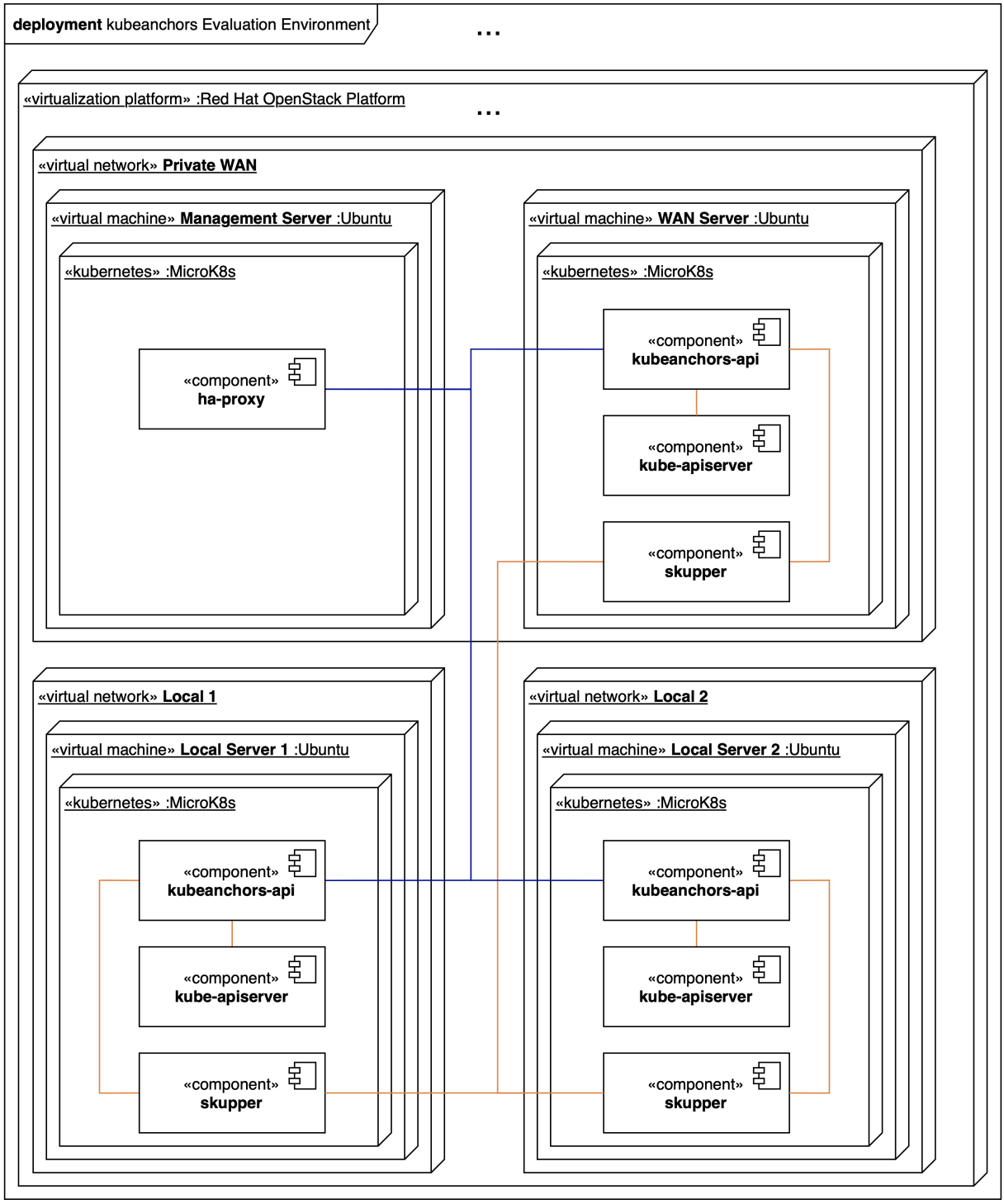


Figure 4.3: kubeanchors Evaluation Environment Deployment

Also, ***kubeanchors*** can access the ***kube-apiserver*** of the *cluster* it is running on. This capability can be used to relay the ***kube-apiserver*** to other hosts.

To access the ***kubeanchors*** *APIs*, an additional ***Kubernetes*** *service* is deployed, exposing the *API* on a local TCP port 30000 of each host[14]. On the management server, ***HAProxy*** [93] is installed to combine and load balance the ***kubeanchors*** *APIs* of each host in a common endpoint. This is not mandatory, as each ***Kubernetes*** host could be accessed directly, but introduces improved reliability and eases access from outside of the ***OpenStack*** environment.

In summary, a decentralized, stateless *multi-cluster management* solution is employed by leveraging ***Skupper*** and adding functionality by combining its networking capabilities with the ***kubeanchors*** *API*.

Figure 4.3 provides an overview of this deployment, omitting infrastructure related components like *virtual routers* for simplicity. Additional deployment details can be found in Appendix E.

[14] This port has no special role and could be any other port exposable by ***Kubernetes***.

4.2. Quality Measurement Scenarios

Within this Section, multiple scenarios are defined to assess the chosen *quality measures* for the *evaluation targets*. Each scenario includes concrete recommendations to the parts of a *quality measurement scenario* described in Section 3.4, along with relevant technical details to ensure accuracy and effectiveness. These scenarios can also be adopted for the analysis of other *evaluation targets* beyond ***Rancher*** or ***kubeanchors*** if they share similar implementations.

Where applicable, multiple *quality measures* are grouped within a single scenario. This approach results in five distinct *quality measurement scenarios*, each designed to determine the *quality measure elements*[15] necessary for calculating the respective *quality measures*.

4.2.1. Load Testing

Response Measure	The load testing scenario aims to quantify the system availability and mean processor utilization measure[16].
Artifact	Generated load will affect the *HTTP APIs* provided by ***Rancher*** [94] and ***kubeanchors***. To delimit the scope of this scenario to *service discovery*, only the *API* endpoints that provide information about all accessible *clusters* registered in the *evaluation target* are called.
Stimulus Source	Load will be generated by a simple *HTTP* client calling the desired *API* endpoints from multiple threads or processes. It is not mandatory, but possible, for the calls being executed on multiple hosts. A number of ten threads or processes is proposed for this scenario but can be adjusted if needed[17].
Stimulus	The *stimulus* manifests through the *HTTP* calls targeting the *API* endpoints of the *evaluation targets*. The time window of the scenario execution is split into fixed frames. Within each frame a defined number of calls is generated.

[15] For more details about the *quality measure elements* refer to ***ISO/IEC 25023***.

[16] This scenario would also suite other resource utilization measures, like mean memory utilization.

[17] The proposed numbers can be altered if necessary. E. g., when transferring the scenario to different targets than ***Rancher*** and ***kubeanchors***.

	A time window of one minute, with a frame interval of one second and ten calls per frame, is proposed. Therefore, with ten client threads or processes calling the *API*, each client will generate one call per second. Overall, within one minute, 600 calls are generated[17].
Response	The system availability will be measured through the results of each request made to the *APIs*. A successful call will result in a *HTTP* response with status code 200. For each frame containing at least one failed response, either by not returning status code 200 or not responding at all, the *evaluation target* is considered as unavailable. Within each time frame, the processor utilization is measured to determine the mean processor utilization.
Environment	No changes to the initially proposed *environment* are necessary.

Table 4.1: Load Testing Scenario

4.2.2. Fault Injection

Response Measure	Within the fault injection scenario**,** the failure avoidance and mean recovery time are quantified.
Artifact	A fault will be injected into what is considered the single point of failure for each *evaluation target*. In case of ***Rancher***, the management server is targeted, as it hosts the ***Rancher*** server. For ***kubeanchors***, the WAN server is targeted, since it interconnects the local clusters.
Stimulus Source	The fault is generated by a manual reboot of the host representing the single point of failure.
Stimulus	The *stimulus* manifests in a reboot of the host representing the single point of failure.
Response	The failure avoidance will be determined by immediately calling the health check *API* endpoint of each target after injecting a reboot. Both, ***Rancher*** and ***kubeanchors***, provide a ping *API* endpoint which returns “pong” if the service is operational. The test is considered successful, if the call still returns the string “pong” while the reboot is carried out.

	In contrast, the mean recovery time is determined by measuring the time from the reboot injection to the first successful call of the ping *API* endpoint to the respective target. To determine the latter, calling the ping endpoint will be repeated after reboot each second[18].
Environment	No changes to the initially proposed *environment* are necessary.

Table 4.2: Fault Injection Scenario

4.2.3. Hardware / Operating System Modulation

Response Measure	The hardware/operating system modulation scenario aims to quantify the hardware environmental adaptability and system software environmental adaptability measure.
Artifact	All *virtual machines* within the *evaluation environment* are targeted.
Stimulus Source	A manual change of the hardware or operating system used to deploy the *evaluation targets* on serves as *stimulus source*. To modulate the hardware, switching to an ***ARM***-based processor architecture [95] is proposed. For operating system modulation, ***Debian*** 12 [96] will be used instead of ***Ubuntu*** 24.04 [27][19].
Stimulus	The *stimulus* is represented by a change of the hardware or operating system the *evaluation target* is running on.
Response	For each *evaluation target* the respective setup routines provide a set of tests to ensure its functionality. These tests are executed after changing the hardware/operating system and setting up the solutions. If the tests succeed, the *evaluation target* is considered adaptable to the new hardware/operating system.
Environment	The hardware and/or operating system used are altered.

Table 4.3: Hardware / Operating System Modulation Scenario

[18] The proposed numbers can be altered if necessary. E. g., when transferring the scenario to different targets than ***Rancher*** and ***kubeanchors***.

[19] Still, other hardware changes or operating systems can be chosen if desired, when for instance applying the scenario to different targets than ***Rancher*** and ***kubeanchors***.

4.2.4. Audit Log Review

Response Measure	This scenario determines the user audit trail completeness.
Artifact	***Rancher*** generates audit logs for every request to its *API* when audit logging is enabled [97]. Also, ***kubeanchors*** generates a log entry for each call handled by its *API*. Therefore, the *APIs* of the services are targeted by this scenario. To delimit the scope of the scenario to *service discovery*, only the *API* endpoints to get information about all accessible *clusters* registered in the respective solution, are called.
Stimulus Source	A *HTTP* client will invoke the *API* endpoints to list all registered *clusters* within the *evaluation target*.
Stimulus	The *stimulus* is realized through the *HTTP* calls hitting the *evaluation target*'s *API*. For each solution, ten calls targeting the *API* endpoint for listing registered *clusters* are proposed. The number of calls can be adjusted if necessary.
Response	All calls to the respective *API* endpoint are expected to be captured within the respective log.
Environment	No changes to the initially proposed *environment* are necessary.

Table 4.4: Audit Log Review Scenario

4.2.5. Penetration Testing

Response Measure	The data integrity measure is assessed during the penetration testing scenario[20].
Artifact	***Rancher*** and ***kubeanchors*** in its entirety, but especially their data about the registered *clusters*, are targeted by this scenario.
Stimulus Source	A designated penetration testing team, to which the *evaluation targets* are presented as black boxes, must try to penetrate the given solutions and alter the data within them.

[20] Also, other integrity measures could be measured within this scenario.

Stimulus	The *stimulus* consists of changes to the data within the *evaluation targets* caused by the penetration testing team. The protected data items aimed to be changed consist of information about the *clusters* to be discovered, as well as information on how the *clusters* can be accessed.
Response	A report of the penetration testing team about which data items they were able to change serves as result. If the team was able to change data items either needed for *service discovery* or *peer-to-peer relaying*, these items are considered corrupted.
Environment	No changes to the initially proposed *environment* are necessary.

Table 4.5: Penetration Testing Scenario

4.3. Quality Measurement Scenario Implementations

This Section describes the implementation of the *quality measurement scenarios* in the *evaluation environment*. For each scenario, the following steps are carried out for both ***Rancher*** and ***kubeanchors***:

(M1) The *evaluation environment* for the respective *evaluation target* is set up.

(M2) The *evaluation target* is tested for functionality.

(M3) The *quality measurement scenarios* are executed, and the *quality measures* are determined.

For the first and second steps, ***Terraform***, ***Ansible*** and ***Testinfra*** code are provided, as described in Section 4.1. The following Subsections detail the actions performed to fulfill the third step. The code implemented can be obtained from Appendix E.

4.3.1. Load Testing

For the load testing scenario, a ***Python***-based load generator is implemented. The script uses the ***Requests*** [98] library to send *HTTP* requests to the *evaluation target*'s *API* endpoints for listing all registered *clusters* ("/v3/clusters" for ***Rancher***, "/all" for ***kubeanchors***).

The requests are executed concurrently across ten threads, with each thread sending a request every second[21] and enforcing a timeout of one second. Only requests that return within this timeframe and receive an *HTTP* 200 status code are considered successful. Any other response or timeout is counted as a failure. For further analysis, the script generates corresponding output.

While the load generator script is running, the **`sar`** [100] command is run on the relevant hosts to capture the processor utilization. To ensure synchronized execution of the load generator and **`sar`**, two **`tmux`** [101] panes are created, and both programs are started in the "synchronize-panes" mode. Since the *virtual machines* do not have time synchronization (e. g., via *NTP*) enabled, the system clocks of all hosts are manually synchronized using the **`date`** [102] command.

As access to ***Rancher***'s *API* requires authentication [94], an authorized ***Requests*** session [103] is created and shared across all threads for calling the *API*. An *API* key is manually created via the ***Rancher UI*** [104] and passed to the load generator script. Since ***Rancher*** runs over *HTTPS* by default, certificate validation is disabled[22] to minimize differences to ***kubeanchors***, which is tested over *HTTP*. The **`sar`** command is executed on the management server, as it hosts the targeted *API*.

[21] A ***GitHub Gist*** [99] served as template for the scheduling algorithm.

[22] The communication remains encrypted but the validity of the TLS certificate is not verified.

Except for ***kubeanchors*** not requiring authentication (yet) and operating over *HTTP*, no other significant modifications for the load generator script are notable. As ***kubeanchors*** is distributed over the WAN and local servers, the **`sar`** command is executed on all these hosts to monitor processor utilization across the entire setup.

4.3.2. Fault Injection

The fault injection scenario is triggered by rebooting the potential single point of failure host for each *evaluation target*. To monitor the outage, a ***Python*** script using the ***Requests*** library is implemented to call the *HTTP* "/ping" *endpoint* of each solution's *API*. This endpoint provides a basic health check to determine whether the application is operational. The script sends a request with a timeout of one second every second. Additionally, the rebooted host is pinged via *ICMP* [105] to measure the time until the host becomes available again.

The *evaluation target* is considered failure avoidant, if the call of the ping *API* endpoint continues to succeed while the reboot is in progress. In this case, the recovery time is set to zero. If the target does not avoid the failure, the recovery time is defined as the duration between the first successful *ICMP* ping and the first successful request of the ping *API* endpoint.

As before, ***Rancher***'s *API* operates over *HTTPS*, while ***kubeanchors*** operates over *HTTP*. The differences are accounted within the ***Python*** script. In addition, time synchronization is performed using the **`date`** command, and the execution of all commands (**`reboot`**, **`ping`** and ***Python*** script) is coordinated using **`tmux`**.

4.3.3. Hardware / Operating System Modulation

The hardware/operating system modulation scenario is implemented by replacing the operating system image of all hosts in the *evaluation environment* from ***Ubuntu*** 24.04 to ***Debian*** 12. As of the time of writing this thesis, no separate *virtualization environment* with different hardware, such as ***ARM***-based processors, is available. Therefore, this scenario focuses solely on operating system modulation.

After setting up the environment and installing all required components, including one of the *evaluation targets*, several tests are conducted to verify the basic functionality of the setup. These tests include checking *DNS* resolution, internet connectivity and, ***MicroK8s*** functionality, but more important checking whether the *evaluation target*'s *API* is operational.

The *evaluation target* is considered adaptable to environmental changes, if all tests pass successfully.

4.3.4. Audit Log Review

Within the audit log review scenario, a generator script written in ***Python*** is provided to call the *API* endpoints for *cluster* discovery of each *evaluation target* ("/v3/clusters" for ***Rancher***, "/all" for ***kubeanchors***) using the ***Requests*** library. In total, the script generates ten *HTTP* calls which are expected to appear in the respective log of the *evaluation target*.

For ***Rancher***'s *API*, authorization is required and provided by passing credentials, manually created in the ***Rancher UI***, to the script. Additionally, audit logging needs to be enabled for ***Rancher*** by a modification of its ***Helm deployment***. This adds a sidecar *container* to each ***Rancher*** server *pod* that writes the audit logs to the standard output. As ***kubeanchors*** does not implement an explicit audit log yet, *API* call logs are leveraged for this purpose.

After running the generator script, the respective logs are obtained via the **`kubectl`** command from each *pod* of the *evaluation target*. For ***Rancher***, all relevant *pods* are running on the management server, whereas for ***kubeanchors*** the *pods* are distributed across the WAN and local servers.

4.3.5. Penetration Testing

As within this thesis no resources for a penetration testing team are available, the execution of this scenario is omitted.

4.4. Measurement Results

The proposed *quality measurement scenarios* were executed as outlined in Section 4.3. This execution yielded several results, which are presented in the following Subsections, categorized by *quality measure*. No further interpretations are provided in this Section, as the results will be discussed in Chapter 5. The raw measurement results can be obtained from Appendix F.

4.4.1. System Availability

To assess the system availability the load testing scenario was executed. The results show that for both ***Rancher*** and ***kubeanchors***, all requests generated by the load testing script were successfully processed. Throughout each one-second interval within the 60-second execution window, both *evaluation targets* remained fully operational.

Consequently, the "operation time actually provided" precisely matches the "system operation time specified in the operation schedule" [77, p. 21]. By applying the formula from ***ISO/IEC 25023***, the system availability was calculated to be 1.0, representing 100% uptime.

4.4.2. Mean Processor Utilization

More significant insights were gathered from processor utilization measurements during the load testing scenario. For each one-second interval of the execution window, processor utilization was recorded for both targets:

- For ***Rancher***, processor utilization on a single host with eight virtual *CPU* cores was captured (the management server).
- For ***kubeanchors***, processor utilization was measured across three hosts with four virtual *CPU* cores (the WAN and the two local servers).

As **sar** already accounts the amount of available *CPU* cores, no further calculations are required to normalize the measures based on *CPU* count.

Figure 4.4 shows the processor utilization for ***Rancher*** and ***kubeanchors*** during the scenario execution. For ***kubeanchors***, the average utilization across all three hosts is displayed. To calculate the mean processor utilization as defined in ***ISO/IEC 25023***, the mean of each recorded data point is computed. Rounded to four decimal places, this results to 0.1027 (or 10.27%) for ***Rancher*** and 0.1287 (or 12,87%) for ***kubeanchors***.

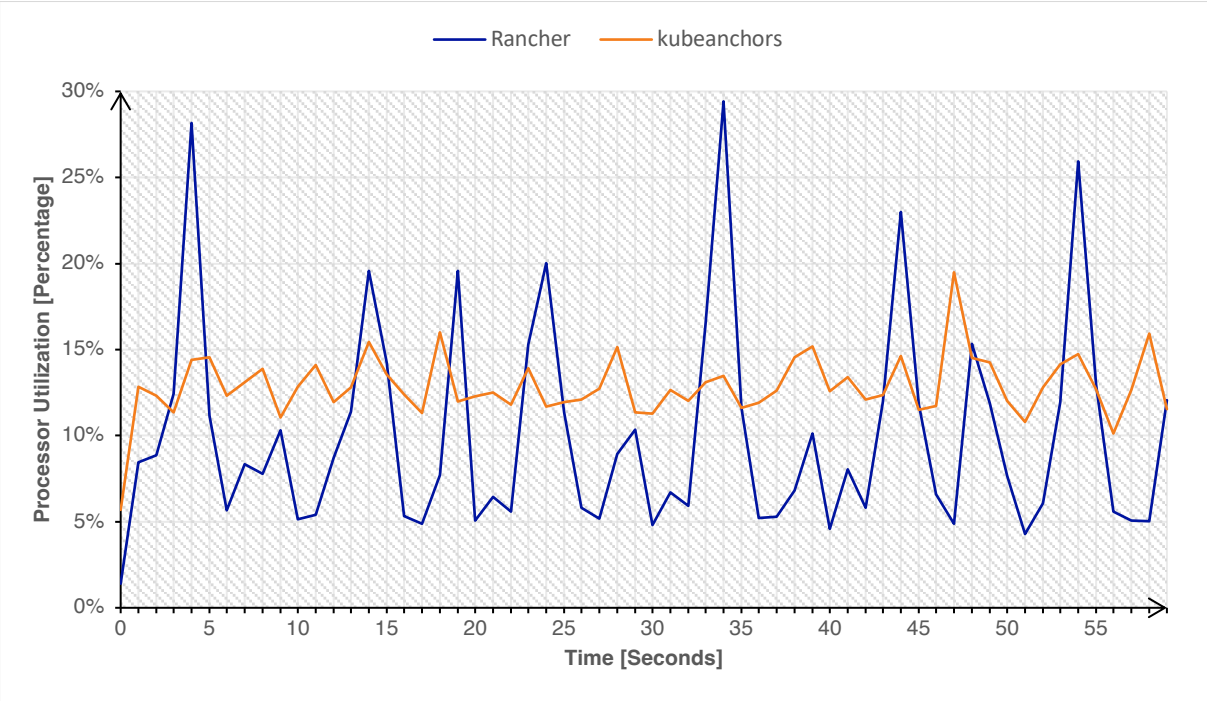


Figure 4.4: Processor Utilization

4.4.3. Failure Avoidance

The failure avoidance was obtained from the fault injection scenario. An *evaluation target* is considered failure avoidant, when the injected fault did not lead to a service outage. A service outage is defined by the ping *API* endpoint of the respective target not responding within one second.

The results show that ***Rancher*** did not sustain the reboot of the management server. In contrast, ***kubeanchors*** continued operating during the reboot of the WAN server. Consequently, applying the formula defined in ***ISO/IEC 25023***, the failure avoidance measures 0.0 (or 0%) for ***Rancher*** and 1.0 (or 100%) for ***kubeanchors***.

4.4.4. Mean Recovery Time

For the mean recovery time, the duration from the first successful *ICMP* ping response from the rebooted host to the first successful call of the ping *API* endpoint of each *evaluation target* was measured during the fault injection scenario. Since only a single fault has been injected, the mean recovery time is equal to the recovery time of that fault injection, as defined by ***ISO/IEC 25023***.

For ***Rancher*** it took 112 seconds to recover. Figure 4.5 illustrates the corresponding timeline of the fault injection scenario. In contrast, as the fault injection did not cause an outage for ***kubeanchors***, the mean recovery time amounts to 0.0 seconds.

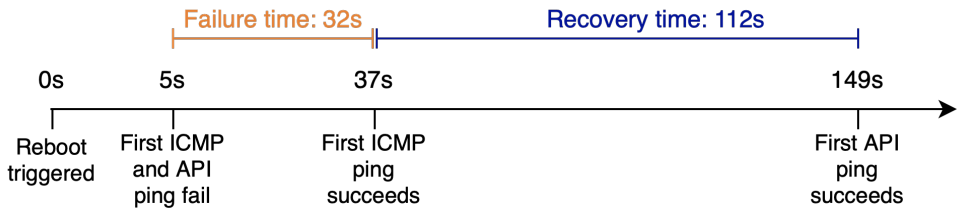


Figure 4.5: Rancher Fault Injection Timeline

4.4.5. System Software Environmental Adaptability

The system software environmental adaptability was assessed during the hardware/operating system modulation scenario. This scenario involved replacing the operating system of all hosts within the *evaluation environment* from ***Ubuntu*** 22.04 to ***Debian*** 12. To install ***Debian*** within ***OpenStack***, the ***Debian*** cloud image [106] was downloaded and then uploaded to the virtualization platform via ***Terraform***. The rest of the setup followed the same logic as described in Section 4.1, except for minor ***Debian***-related adjustments within the ***Ansible*** code, such as installing ***Snapcraft*** [107] for ***MicroK8s*** and changing the default username from “ubuntu” to “debian".

Both *evaluation targets* demonstrated full compability with the modified environment. All functionality tests executed against the setup for both ***Rancher*** and ***kubeanchors*** passed successfully. Therefore, applying the formula from ***ISO/IEC 25023***, both solutions exhibit a system software environmental adaptability of 1.0 (or 100%).

4.4.6. User Audit Trail Completeness

To examine the user audit trail completeness, a total of ten requests were generated against ***Rancher***’s and ***kubeanchors***’ *APIs*. It was expected that these requests would appear in the audit logs of the respective solutions. For ***Rancher***, the logs of the audit *sidecar containers* were inspected, while for ***kubeanchors***, the *API* call logs of each node were screened.

For both targets, all requests could be identified within the logs. Consequently, applying the formula from ***ISO/IEC 25023*** for the user audit trail completeness results a value of 1.0 (or 100%) for both solutions.

4.5. Conclusion

The evaluation provided valuable insights into ***Rancher***'s and ***kubeanchors***' *quality criteria*. Within a unified *evaluation environment*, several *quality measurement scenarios* were executed to assess the *quality measures* defined in the ***SQuaRE*** model. This standardized setup ensured consistent and repeatable measurements and minimized potential noise in the evaluation, leading to neutral and objective results.

Table 4.6 concludes the findings captured within this evaluation:

Quality Criterion	Quality Measure	Results: Rancher	Results: kubeanchors
reliability: availability	system availability	100%	100%
performance efficiency: resource utilization	mean processor utilization	10.27%	12.87%
reliability: fault tolerance	failure avoidance	0%	100%
reliability: recoverability	mean recovery time	112 seconds	0 seconds
flexibility: adaptability	system software adaptability	100%	100%
security: accountability	user audit trail completeness	100%	100%

Table 4.6: Evaluation Results

Even though several *quality criteria* were omitted from the scope of evaluation due to given constraints, this study still provides a best practice for applying the model developed in Chapter 3. Additional *quality measurement scenarios* can be incorporated to assess the full range of the chosen *quality criteria* and *quality measures* from the ***SQuaRE*** model (see Table 3.3).

Discussions about the evaluation results will be introduced in Chapter 5.

5. Discussion

After developing an *evaluation model* and applying it exemplarily to ***Rancher*** and ***kubeanchors*** within this Chapter, the measurement results of the evaluation are discussed. Furthermore, potential weaknesses and strengths of the *evaluation model* are outlined. Finally, advantages and shortcomings of the implementation considerations within the evaluation are presented.

The following Sections and Subsection are divided into Paragraphs, with each Paragraph representing a distinct argument to be discussed.

5.1. Measurement Results

Within this Subsection the results for each *quality criterion* measured in the evaluation will be interpreted and suggestions for improvement are listed.

5.1.1. System Availability

The system availability amounts to 100% for both *evaluation targets*. From this observation it can be concluded that both solutions are capable of handling at least ten *API* calls per second successfully. This also proves, ***kubeanchors*** has the same capability in terms of system availability as ***Rancher***.

a) Scenario Parameters

A possible critique to the underlying load testing scenario is that the parameters for the scenario were chosen solely by the author. Therefore, no external experience concerning load testing is incorporated within this scenario.

b) Scenario Focus

The scenario designed to determine the *quality measure elements* required to calculate the system availability measure according to ***ISO/IEC 25023***. Therefore, the scenario does not include variations in the number of *API* calls per second for the load generator.

In the author's opinion, a more meaningful approach would be to increase the number of *API* calls and employ multiple measurements to determine the maximum load the *evaluation targets* can handle. But this is not foreseen for the system availability measure in ***ISO/IEC 25023***.

Additionally, executing the load generation from multiple hosts would provide further insights into the system availability under distributed load conditions.

5.1.2. Mean Processor Utilization

The results for the mean processor utilization indicate that ***kubeanchors*** causes a slightly higher mean processor utilization than ***Rancher***. This increase could be attributed to the networking overhead introduced by ***kubeanchors***' *decentralized architecture*. Each call to one of the deployed ***kubeanchors*** *APIs* triggers additional calls to all other *APIs*, generating traffic on the ***Skupper*** network, which in turn contributes to *CPU* load.

However, the difference in mean processor utilization is only 2.60% higher for ***kubeanchors*** compared to ***Rancher***. Given this small margin, these interpretations should be considered with caution, as the impact may not be significant in practical scenarios.

Interestingly, the graph for ***Rancher***'s processor utilization shows a significantly higher standard deviation (0.0610) compared to ***kubeanchors*** (0.0184), indicating that ***kubeanchors*** produced a more consistent load during the scenario execution.

Part of this can be explained by the fact, that the graph for ***kubeanchors*** represents the average processor utilization across three servers, which naturally smooths out fluctuations. However, even when looking at the individual standard derivation for each host, these values remain closer to 0.0, further supporting the conclusion that ***kubeanchors*** generates a more stable load distribution:

$\sigma^2(\text{WAN}) = 0.0267$ $\sigma^2(\text{Local 1}) = 0.0416$ $\sigma^2(\text{Local 2}) = 0.0355$

Figure 5.1: Processor Utilization for kubeanchors

a) Scenario Parameters

Analogous to Paragraph 5.1.1.a).

b) Measurement Method

The processor utilization was measured by executing the **sar** command on each relevant host during the execution of the scenario. This approach offers the advantage of directly obtaining the processor utilization from each host without requiring additional calculations involving parameters like the *CPU* count.

However, more precise results could be obtained by querying the ***metrics-server*** [108] running within each ***Kubernetes*** *clusters*, as it provides a statistics on the *pod* level. The downside is that the ***metrics-server*** delivers results in a different unit (milli core), which would require several complex calculations to derive the processor utilization [108]. Therefore, the **sar** command was used for the evaluation within this thesis.

Nevertheless, when including level 3 in the *evaluation environment*, another approach besides the **sar** command, such as leveraging the ***metrics-server***, could become mandatory. This is because the hosts of managed ***Kubernetes*** distributions may not always be directly accessible, making it difficult to execute commands like **sar** on these hosts.

5.1.3. Failure Avoidance

For the failure avoidance, ***Rancher*** was unable to handle the reboot executed on the management server during the fault injection scenario. This is expectable, as it causes all *pods* running the ***Rancher*** server to be stopped simultaneously since they operate on the same host. On the other hand, ***kubeanchors*** was able to withstand the reboot of the WAN server. This is because the ***HAProxy*** running on the management server directs all requests to the remaining available servers.

a) Single Point of Failure Considerations

For ***Rancher*** the management server was assumed as the single point of failure for the setup. The fault injection scenario confirmed this assumption, as the reboot of the management server led to the failure of the ***Rancher*** service. Arguably, for ***kubeanchors***, the management server also could be considered the single point of failure. A reboot of the management server would cause the ***HAProxy*** to become unavailable, resulting in requests not being redirecting to the ***kubeanchors*** *APIs*.

Still, the WAN server is considered the single point of failure for ***kubeanchors***, as the ***HAProxy*** is an optional component in the setup. If ***HAProxy*** fails, the individual ***kubeanchors*** *APIs* could still be queried directly, ensuring service availability.

b) Data Consistency Check

More meaningful results would be available if the failure avoidance check included the data presented by the *APIs* during the failure injection. If data consistency were a mandatory criterion for the failure avoidance check, also ***kubeanchors***' results would be considered faulty. This is because *API* calls querying all available *clusters* (using the "/all" endpoint) would only show the *cluster* the *API* is running on. This is caused by the *clusters* not being able to access each other's *APIs* when the WAN server, which interconnects the *clusters*, is unavailable. As a result, the reboot of the WAN server leads to incomplete data being returned by the *APIs* of the local servers.

c) Scenario Repetition

The scenario was executed only once, but it could be run multiple times to provide more statistically significant results.

5.1.4. Mean Recovery Time

For ***Rancher*** a mean recovery time of 112 seconds was measured, which is significantly longer than the mean recovery time for ***kubeanchors***. Yet, ***kubeanchors*** mean recovery time is set to 0.0 seconds, as the fault injected within the respective scenario was avoided by ***kubeanchors***. Therefore, this value must be considered less representative.

a) Scenario Ontology

As outlined, setting the mean recovery time for ***kubeanchors*** to 0.0 seconds due its resistance to the injected fault and comparing this value to a real measure, provided by ***Rancher***'s outage, does not offer a meaningful correlation between the two values. Instead, faults should be injected that also cause ***kubeanchors*** to fail. Alternatively, the pass criteria for the *evaluation targets*' failure avoidance could be changed, e. g., by including data consistency (see 5.1.3.b)). Additionally, multiple faults could be introduced for both *evaluation targets* to providing a broader range of measurement results.

b) Scenario Repetition

Analogous to Paragraph 5.1.3.c)

5.1.5. System Software Environmental Adaptability

Both *evaluation targets* show being adaptable to a different *software* environment. The altered environment, which included ***Debian*** 12 as the operating system, was introduced within the hardware/operating system modulation scenario.

a) Operating System Considerations

As ***Ubuntu*** is a ***Debian***-based distribution [109], it is not surprising that the *evaluation targets* were also able to run on ***Debian***. Additionally, other operating systems like ***Red Hat Enterprise Linux*** or ***Microsoft Windows Server*** [110] could be considered for a broader test coverage.

b) Containers

As the majority of components in the *evaluation targets* run within *containers*, their portability to other operating systems is expected. Nevertheless, it is reasonable to validate this through testing.

5.1.6. User Audit Trail Completeness

The user audit trail completeness amounts to 100% for both *evaluation targets*. This means, each request generated is logged within the respective logs.

a) Scenario Parameters

Analogous to Paragraph 5.1.1.a).

b) Reviewed Logs

For ***Rancher*** a real audit log was reviewed to calculate the presented results. The generation of the audit logs had to be enabled within the ***Helm*** *deployment* of ***Rancher***. Contrary, only the *API* call logs of ***kubeanchors*** were screened for the user audit trail completeness. By time of carrying out the measurements, ***kubeanchors*** does not offer an audit log incorporating user information, as ***kubeanchors*** in general has no authentication implemented yet. To provide more comparable results, for ***kubeanchors*** also audit logs should be scanned, as soon as available.

c) Integration In Load Testing Scenario

Currently, an own scenario was developed to check the user audit trail completeness. The measurement of the user audit trail completeness could also be integrated into the load testing scenario, reducing the total amount of scenarios.

5.2. Evaluation Model

This Section discusses the *evaluation model* developed in Chapter 3. While the model is already well-engineered and well-founded, based on thorough research and modeling using ***ISO/IEC*** standards and additional literature, there is still room for improvement.

a) Practical Advice

The *evaluation model*, combined with the exemplary evaluation of ***Rancher*** and ***kubeanchors***, aims to provide guidance on applying the ***SQuaRE*** model in practice. This was an objective of this thesis, as the model's documentation only provides limited resources on this topic. ***Q42*** has also recognized these shortcomings of the ***SQuaRE*** model [111].

Even though the evaluation within Chapter 4 does not cover the full extent of the *evaluation model*, it still provides an overview of its application. While the scope of the evaluation was limited to ***Rancher*** and ***kubeanchors***, the model can be easily adapted to additional *evaluation targets* with minimal effort.

b) Model Structure

The structure of the *evaluation model* follows the hierarchic framework outlined in ***ISO/IEC 25002***. Both ***Q42*** and the author argue that grouping *sub-characteristics* within broader *characteristics* is not strictly necessary, as the *sub-characteristics* themselves provide sufficient information for *quality* assessment. Additionally, some *sub-characteristics* could arguably belong to multiple *characteristics*, which may lead to confusion [111].

Nevertheless, the *evaluation model* adheres to this structure to maintain conformity with ***ISO/IEC 25002***. While this is not inherently a disadvantage, it adds complexity and makes the model harder to understand.

c) Literature Review

As a central part of the *evaluation model*, a literature review was conducted to choose the *quality criteria* relevant to the given context. This review leveraged three databases: ***IEEE Xplore***, ***Scopus***, and ***Web of Science***. Also, ***ACM*** [112] and ***Google Scholar*** [113] were considered for the review but then excluded, as

- ***ACM***'s literature export could not be successfully imported into ***Zotero*** with complete metadata, and the library only allowed filtering for one literature type at a time.
- ***Google Scholar*** provided only limited filtering options, which were unsuitable for the review, and did not support a literature export.

Since several steps were performed manually, possibility of human error cannot be ruled out. To ensure transparency and traceability, Appendix C includes all resources to replicate the screening process. The only limitation may arise from restricted access to certain literature depending on the institution accessing it.

Upon reviewing the literature review results, it was observed that the full-text review considered only British English spellings of the *quality criteria*, such as “time behaviour” or “analysability”. For more comprehensive results, the full-text review should have also included the American English spelling. Additionally, searching for corresponding noun forms of *quality criteria* written as adjective could have further improved the review.

It is unsurprising that certain *quality criteria*, such as “freedom from health risk”, yielded no results, as their names are more complex than others. It should be considered how to equalize this difference. One possible approach is to analyze the statistical occurrence of the *quality criteria* names in English literature, using tools like ***Google Books Ngram Viewer*** [114] or ***NGRAMS*** [115].

Finally, leveraging a machine learning model or *AI*-based approach for the full-text screening could enhance the process by incorporating context from each literature source. Otherwise, each document must be manually reviewed to ensure no relevant mentions of *quality criteria* are overlooked.

d) Stakeholder Priorities

Within the *evaluation model*, several *stakeholders* relevant to the given context were identified. It is assumed that the literature review sufficiently captures their requirements. However, this assumption remains speculative. To ensure that *stakeholders*' needs are accurately considered, conducting a survey or expert interviews would have been a more reliable approach.

e) Quality Measures

Since the ***SQuaRE*** series provides only limited guidance on how to determine the *quality measures* precisely, this thesis introduced *quality measurement scenarios* to bridge that gap. However, these scenarios include interpretations by the author. As a result, it still is possible that they do not fully align with the definitions provided in the ***SQuaRE*** series. Nevertheless, the *quality measurement scenarios* offer a reproduceable, testable, and easy-to-understand approach for obtaining the desired *quality measures*, which can otherwise be difficult to grasp within the ***SQuaRE*** series.

5.3. Implementation Considerations

Several considerations were taken for the implementation of the *evaluation environment* and the *quality measurement scenarios*. The following Subsections provide justifications for the decisions made.

5.3.1. Evaluation Environment

a) Omissions

During the development of the *evaluation environment*, the decision was made to omit level 0 and level 3 from the initially proposed setup. This decision was based on several advantages that arise from excluding these levels.

By leaving out level 3, dependence on a managed ***Kubernetes*** provider, such as ***Microsoft Azure***, is avoided. As a result, only an ***OpenStack*** platform is required to set up the implementation of the environment. Additionally, since a *cluster* on level 3 would run within the internet, access is typically less restricted compared to a *cluster* within an internal network.

The omission of level 0 is justified by its lack of additional value compared to level 1. A level 0 host, such as one running ***minikube***, would have been deployed within the same network as local server 1. Therefore, apart from the difference in the ***Kubernetes*** distribution, it would not have introduced any significant distinction from the local clusters.

b) NAT

Within the *evaluation environment* implementation, *NAT* was expected functioning for packages being directed from a local network to the internet via the WAN router. Instead, *NAT* only functioned for packages originating from hosts within the WAN. This issue was bypassed by introducing the management server to provide a *SOCKS* proxy and *DNS* resolving.

In an ideal setup, the local servers would be able to access the internet directly. It is assumed that a bug in ***OpenStack*** caused this behavior, as a similar behavior is described in [116] for ***Ubuntu***'s ***Charmed OpenStack*** [117].

c) NTP

During the measurements, it was observed that not all servers were synchronized in time. This issue was temporarily addressed by using the **`date`** command simultaneously on all hosts to set the time. A more reliable solution would have been to configure time synchronization via *NTP*. However, the **`date`** command was preferred to proceed with the measurements quickly.

d) Docker Pull Limit

As the ***OpenStack*** used for the evaluation runs within a company network, a pull limit of 10 *image* pulls per hour from ***Docker Hub*** for the entire network restricted the setup of ***MicroK8s*** and deployment of *containers* [118]. This limitation was bypassed by collecting all required *container images* into an archive file and importing it into ***MicroK8s*** during the setup using the **`microk8s image import`** command. In an ideal setup, this workaround would not be necessary.

5.3.2. Quality Measurement Scenarios

a) Protocol Differences

All calls to ***Rancher***'s *API* used *HTTPS*, while calls to ***kubeanchors*** were made over *HTTP*. This introduced a more complex setup for ***Rancher***, as *HTTPS* includes encryption via *TLS*. To minimize differences, certificate validation was disabled for requests to ***Rancher***. However, *TLS* encryption within *HTTPS* still adds a significant overhead.

In an improved ***kubeanchors*** setup, *TLS* would also be enabled by exposing its *API* through a ***Kubernetes ingress*** with *TLS* configured via ***cert-manager*** [119], which is the same setup used by ***Rancher***. Adopting this setup would provide more comparable results at the *API*.

Nevertheless, this difference was tolerated within the evaluation, as the protocol (*HTTP*/*HTTPS*) is inherent to the ontology of the respective solutions.

b) Authentication

Similar to the protocol differences outlined in Paragraph 5.3.2.b), ***Rancher*** requires authentication before accessing the *API* endpoint to list all registered *clusters*, whereas ***kubeanchors*** does not yet implement authentication. This difference was also tolerated as part of the solutions ontology.

However, authentication is planned for ***kubeanchors***, and the measurements should be repeated once the implementation is completed. Repeating the measurements is straightforward, as they were designed to be reproduceable.

c) Peer-to-Peer Relaying for kubeanchors

Currently, the ***kubeanchors*** *API* primarily provides *service discovery* for ***Kubernetes*** *clusters*. *Peer-to-peer relaying* can be achieved by exposing the ***kube-apiserver*** of the targeted *cluster* via the ***Skupper*** network. However, this feature was not considered in the evaluation. No significant changes are expected for the *quality measurement scenarios* when incorporating explicit measurements for *peer-to-peer relaying*, as this only adds additional *API* endpoints for ***kubeanchors***.

The load testing scenario and audit log review scenario would be adjusted to include these endpoints, while the fault injection scenario would remain unchanged. The hardware/operating system modulation scenario only would include additional tests for ***kubeanchors***, since ***Rancher***'s *peer-to-peer relaying* already is tested.

6. Conclusion

6.1. Results

In summary, an *evaluation model* was developed to assess the *quality* of solutions for ***Kubernetes*** *multi-cluster management,* focusing on *service discovery* of ***Kubernetes*** *clusters* and *peer-to-peer relaying* of the ***Kubernetes*** *API*. The model adheres to various ***ISO/IEC*** standards from the ***SQuaRE*** series, ensuring standardization and comparability. A literature review was conducted to identify relevant *quality criteria*.

To ensure repeatable and unbiased measurement results, a unified *evaluation environment* was developed using the *infrastructure-as-code* methodology. Measurements are performed within reproduceable *quality measurement scenarios* to further enhance reliability and consistency of the results.

For the demonstration of the *evaluation model*, ***Rancher*** and ***kubeanchors*** were evaluated using the developed framework. The evaluation was successfully conducted, yielding results that enable a direct comparison of the two solutions and provide insights into their *quality criteria*.

Surprisingly, ***kubeanchors***, which by time of writing this thesis only represents a minimum viable product, could keep up with ***Rancher*** in terms of availability, adaptability, and accountability. Even though ***kubeanchors*** caused a higher processor utilization (2.6%), it outperformed ***Rancher*** in terms of failure avoidance. However, ***Rancher*** still provides significantly more features than ***kubeanchors*** and is a fully mature solution, suitable for production-use.

The discussion highlighted that the *quality measurement scenarios* require further refinement to produce more meaningful and insightful results. Several omissions present opportunities for enhancing the *evaluation model*. Nevertheless, the *evaluation model* is already mature enough to be adopted for further measurements.

6.2. Outlook

The thesis leaves room for improvement in several areas. On the one hand, ***kubeanchors*** can be further improved, for example, by…

- … implementing an *API* endpoint to relay the ***Kubernetes*** *API* of a *cluster* to any other *cluster* or an admin client.
- … implementing a heartbeat-based caching mechanism to overcome segmentations within the ***Skupper*** network.

- … redesigning the network topology of the ***Skupper*** network to improve reliability.
- … updating to ***Skupper*** v2, which, at the time of writing, is still a preview release, to take advantage of new features and optimizations.
- … rewriting ***kubeanchors*** in a resource-efficient language like ***Go*** [120] or ***Rust*** [121] for better performance and memory management.
- … providing an open-source development repository on ***GitHub*** [122] to encourage community contributions and facilitate further development.

On the other hand, the *evaluation model* can be improved, for instance, by…

- … re-running the literature review, including the proposed corrections within the discussion, to ensure that the most relevant and recent *quality criteria* are included.
- … incorporating the omitted *quality criteria*, which rely on the specification of the *evaluation target*.
- … implementing an optional level 3 for the *evaluation environment* by providing ***Terraform*** code for a managed *cluster* like ***AKS***, ***EKS*** and/or ***GKE***.
- … providing instructions, a script, or an appliance to set up an ***OpenStack*** platform for deploying the *evaluation environment* on.

Finally further *evaluation targets*, such as ***Paralus*** [123] or ***Portainer*** [124], can be assessed to gain additional insights into the behavior of the *evaluation model* and the evaluated targets, contributing to broader applicability and validation of the model.

Literature

[1] Object Management Group, *Unified Modeling Language*, formal/2017-12-05, Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/About-UML

[2] The Kubernetes Authors, “Production-Grade Container Orchestration,” Kubernetes. Accessed: Mar. 04, 2025. [Online]. Available: https://kubernetes.io/

[3] The Linux Foundation, “CNCF Annual Survey 2023,” CNCF. Accessed: Mar. 06, 2025. [Online]. Available: https://www.cncf.io/reports/cncf-annual-survey-2023/

[4] Microsoft, “Managed Kubernetes Service (AKS),” Microsoft Azure. Accessed: Jan. 31, 2025. [Online]. Available: https://azure.microsoft.com/en-us/products/kubernetes-service

[5] Amazon Web Services, Inc., “Amazon Elastic Kubernetes Service,” aws. Accessed: Jan. 31, 2025. [Online]. Available: https://aws.amazon.com/eks/

[6] Google, “Google Kubernetes Engine (GKE),” Google Cloud. Accessed: Jan. 31, 2025. [Online]. Available: https://cloud.google.com/kubernetes-engine

[7] Canonical Ltd., “The effortless Kubernetes,” MicroK8s. Accessed: Jan. 31, 2025. [Online]. Available: http://microk8s.io

[8] K3s Project Authors, “Lightweight Kubernetes,” K3s. Accessed: Jan. 31, 2025. [Online]. Available: https://k3s-io.github.io/

[9] Red Hat, Inc., “Red Hat OpenShift enterprise application platform,” Red Hat. Accessed: Mar. 06, 2025. [Online]. Available: https://www.redhat.com/en/technologies/cloud-computing/openshift

[10] Rancher, “Innovate Everywhere,” Rancher Labs. Accessed: Mar. 05, 2025. [Online]. Available: http://www.rancher.com

[11] ISO, “ISO - International Organization for Standardization,” ISO. Accessed: Mar. 06, 2025. [Online]. Available: https://www.iso.org/home.html

[12] IEC, “International Electrotechnical Commission,” International Electrotechnical Commission. Accessed: Mar. 06, 2025. [Online]. Available: https://www.iec.ch/homepage

[13] “IEEE Standard Glossary of Software Engineering Terminology,” *IEEE Std 61012-1990*, pp. 1–84, Dec. 1990, doi: 10.1109/IEEESTD.1990.101064.

[14] ISO/IEC JTC 1/SC 7, *ISO/IEC 25000:2014, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*, Mar. 2014.

[15] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Fourth Edition. in SEI Series in Software Engineering. Boston Amsterdam Munich: Addison-Wesley, Professional, 2021. ISBN: 978-0-13-688609-9

[16] ISO/IEC JTC 1/SC 7, *ISO/IEC 25002:2024, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality model overview and usage*, Mar. 2024.

[17] G. Starke, “How to Use this Site,” arc42 Quality Model. Accessed: Jan. 03, 2025. [Online]. Available: https://quality.arc42.org/how-to-use-this-site/

[18] G. Starke, “Home,” arc42 Quality Model. Accessed: Dec. 08, 2024. [Online]. Available: https://quality.arc42.org/

[19] B. Burns, K. Hightower, J. Beda, and L. Evenson, *Kubernetes*. Heidelberg: dpunkt.verlag, 2023. ISBN: 978-3-96910-962-5

[20] Leon Alexander Kraß, “Erweiterung des CAP-Theorems zur Charakterisierung von (Micro-)Service-Architekturen.” Aug. 02, 2021. [Online]. Available: https://leonkrass.dev/bachelor-thesis

[21] The Kubernetes Authors, “Sidecar Containers,” Kubernetes. Accessed: Mar. 04, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/

[22] The Kubernetes Authors, “kube-apiserver,” Kubernetes. Accessed: Feb. 22, 2025. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/

[23] The Kubernetes Authors, “Command line tool (kubectl),” Kubernetes. Accessed: Mar. 04, 2025. [Online]. Available: https://kubernetes.io/docs/reference/kubectl/

[24] Helm Authors and The Linux Foundation, “The package manager for Kubernetes,” Helm. Accessed: Feb. 23, 2025. [Online]. Available: https://helm.sh/

[25] The Linux Foundation, “Find, install and publish Cloud Native packages,” Artifact Hub. Accessed: Mar. 04, 2025. [Online]. Available: https://artifacthub.io/

[26] ubuntuusers.de, “APT,” ubuntuusers. Accessed: Mar. 04, 2025. [Online]. Available: https://wiki.ubuntuusers.de/APT/

[27] Canonical Ltd., “Enterprise Open Source and Linux,” Ubuntu. Accessed: Feb. 22, 2025. [Online]. Available: https://ubuntu.com/

[28] Red Hat, Inc., “Managing software with the DNF tool,” Red Hat Documentation. Accessed: Mar. 04, 2025. [Online]. Available: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html-single/managing_software_with_the_dnf_tool/index

[29] Red Hat, Inc., “Red Hat Enterprise Linux operating system,” Red Hat. Accessed: Mar. 02, 2025. [Online]. Available: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux

[30] Docker, Inc., “Dockerfile overview,” Docker Documentation. Accessed: Mar. 04, 2025. [Online]. Available: https://docs.docker.com/build/concepts/dockerfile/

[31] Docker, Inc., “Docker Hub Container Image Library,” Docker Hub. Accessed: Mar. 04, 2025. [Online]. Available: https://hub.docker.com

[32] The Kubernetes Authors, “Images,” Kubernetes. Accessed: Mar. 04, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/containers/images/

[33] S. Morellato, “How to Solve Common Kubernetes Multi-Cluster Deployment Issues,” DEV Community. Accessed: Mar. 05, 2025. [Online]. Available: https://dev.to/simone_morellato/how-to-solve-common-kubernetes-multi-cluster-deployment-issues-35fa

[34] S. Azulay, “Multi-cluster Kubernetes: Benefits, Challenges and Tools,” groundcover. Accessed: Mar. 05, 2025. [Online]. Available: https://www.groundcover.com/blog/kubernetes-multi-cluster

[35] Tigera, Inc., “Multi-Cluster Kubernetes: A Practical Guide,” Tigera. Accessed: Mar. 05, 2025. [Online]. Available: https://www.tigera.io/learn/guides/kubernetes-networking/kubernetes-multi-cluster/

[36] J. Walker, “Mastering Kubernetes Management: Challenges and Best Practices,” Rafay. Accessed: Mar. 05, 2025. [Online]. Available: https://rafay.co/the-kubernetes-current/mastering-kubernetes-management-challenges-and-best-practices/

[37] S. Dubey and M. J. Kulkarni, *Hands-On Kubernetes, Service Mesh and Zero-Trust: Build and Manage Secure Applications Using Kubernetes and Istio (English Edition)*, First Edition. London: BPB Online, 2023. ISBN: 93-5551-867-6

[38] The CoreDNS Authors and The Linux Foundation, “CoreDNS: DNS and Service Discovery,” CoreDNS. Accessed: Mar. 05, 2025. [Online]. Available: https://coredns.io/

[39] The Kubernetes Authors, “Network Plugins,” Kubernetes. Accessed: Mar. 05, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/

[40] G. Arbezzano and A. Palesandro, “Simplifying multi-clusters in Kubernetes,” CNCF. Accessed: Mar. 05, 2025. [Online]. Available: https://www.cncf.io/blog/2021/04/12/simplifying-multi-clusters-in-kubernetes/

[41] SUSE Rancher, “What is Rancher?,” Rancher. Accessed: Mar. 05, 2025. [Online]. Available: https://ranchermanager.docs.rancher.com/

[42] Rancher, “Use Cases of Rancher Prime,” Rancher Labs. Accessed: Mar. 05, 2025. [Online]. Available: http://www.rancher.com/use-cases

[43] SUSE Rancher, “Rancher Server and Components,” Rancher. Accessed: Mar. 05, 2025. [Online]. Available: https://ranchermanager.docs.rancher.com/reference-guides/rancher-manager-architecture/rancher-server-and-components

[44] SUSE Rancher, “Communicating with Downstream User Clusters,” Rancher. Accessed: Mar. 05, 2025. [Online]. Available: https://ranchermanager.docs.rancher.com/reference-guides/rancher-manager-architecture/communicating-with-downstream-user-clusters

[45] SUSE Rancher, “Port Requirements,” Rancher. Accessed: Mar. 05, 2025. [Online]. Available: https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/installation-requirements/port-requirements

[46] The Skupper Authors, “Skupper - Multicloud communication,” Skupper. Accessed: Feb. 23, 2025. [Online]. Available: https://skupper.io/

[47] The Skupper Authors, “Skupper - Skupper overview,” Skupper. Accessed: Mar. 05, 2025. [Online]. Available: https://skupper.io/docs/overview/index.html

[48] The Skupper Authors, “Skupper - Getting started,” Skupper. Accessed: Mar. 07, 2025. [Online]. Available: https://skupper.io/start/index.html#step-5-expose-your-services

[49] Python Software Foundation, “Welcome to Python.org,” Python. Accessed: Jan. 30, 2025. [Online]. Available: https://www.python.org/about/

[50] S. Ramírez, “FastAPI,” FastAPI. Accessed: Mar. 05, 2025. [Online]. Available: https://fastapi.tiangolo.com/

[51] Gernot Starke, “Quality Models,” arc42 Quality Model. Accessed: Jan. 07, 2025. [Online]. Available: https://quality.arc42.org/articles/quality-models

[52] Bundesamt für Sicherheit in der Informationstechnik, “IT-Grundschutz,” Bundesamt für Sicherheit in der Informationstechnik. Accessed: Dec. 14, 2024. [Online]. Available: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/it-grundschutz.html?nn=128656

[53] Bundesamt für Sicherheit in der Informationstechnik, “Bundesamt für Sicherheit in der Informationstechnik,” Bundesamt für Sicherheit in der Informationstechnik. Accessed: Mar. 06, 2025. [Online]. Available: https://www.bsi.bund.de/DE/Home/home_node.html

[54] ISO/IEC JTC 1/SC 27, *ISO/IEC 27001:2022, Information security, cybersecurity and privacy protection - Information security management systems - Requirements*, Oct. 2022.

[55] Y. Argotti, Y. Kenfaoui, C. Baron, A. Abran, and P. Esteban, “An Operational Quality Model of Embedded Software Aligned with ISO 25000,” *ACM Trans. Embed. Comput. Syst.*, vol. 24, no. 1, pp. 1–41, Jan. 2025, doi: 10.1145/3691642.

[56] V. S. Barletta, D. Caivano, L. Colizzi, G. Dimauro, and M. Piattini, “Clinical-chatbot AHP evaluation based on ‘quality in use’ of ISO/IEC 25010,” *Int. J. Med. Inf.*, vol. 170, p. 104951, Feb. 2023, doi: 10.1016/j.ijmedinf.2022.104951.

[57] W. Perdomo and C. M. Zapata, “Software quality measures and their relationship with the states of the software system alpha,” *Ingeniare Rev. Chil. Ing.*, vol. 29, no. 2, pp. 346–363, Jun. 2021, doi: 10.4067/S0718-33052021000200346.

[58] R. Polillo, “Quality Models for Web [2.0] Sites: A Methodological Approach and a Proposal,” in *Current Trends in Web Engineering*, vol. 7059, A. Harth and N. Koch, Eds., in Lecture Notes in Computer Science, vol. 7059. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 251–265. doi: 10.1007/978-3-642-27997-3_25.

[59] A. Ravanello, J.-M. Desharnais, L. E. Bautista Villalpando, A. April, and A. Gherbi, “Performance Measurement for Cloud Computing Applications Using ISO 25010 Standard Characteristics,” in *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, Oct. 2014, pp. 41–49. doi: 10.1109/IWSM.Mensura.2014.33.

[60] The Linux Foundation, “Certified Kubernetes Administrator (CKA),” Linux Foundation - Education. Accessed: Jan. 22, 2025. [Online]. Available: https://training.linuxfoundation.org/certification/certified-kubernetes-administrator-cka/

[61] The Linux Foundation, “Linux Foundation - Decentralized innovation, built with trust,” Linux Foundation. Accessed: Mar. 06, 2025. [Online]. Available: https://www.linuxfoundation.org

[62] The Linux Foundation, “Certified Kubernetes Application Developer (CKAD),” Linux Foundation - Education. Accessed: Jan. 10, 2025. [Online]. Available: https://training.linuxfoundation.org/certification/certified-kubernetes-application-developer-ckad/

[63] ISO/IEC JTC 1/SC 7, *ISO/IEC 25010:2023, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model*, Nov. 2023.

[64] ISO/IEC JTC 1/SC 7, *ISO/IEC 25019:2023, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality-in-use model*, Nov. 2023.

[65] M. Oriol Hilari, J. Franch Gutiérrez, and J. Marco Gómez, “Monitoring the quality of service to support the service based system lifecycle,” Universitat Politècnica de Catalunya, 2015. doi: 10.5821/dissertation-2117-95669.

[66] B. Greetham, *How to Write Your Literature Review*. in Palgrave Study Skills. London: Bloomsbury Academic, 2020. ISBN: 978-1-352-01104-3

[67] IEEE, “IEEE Xplore,” IEEE Xplore. Accessed: Jan. 11, 2025. [Online]. Available: https://ieeexplore.ieee.org

[68] Elsevier B.V., “Welcome to Scopus Preview,” Scopus. Accessed: Jan. 11, 2025. [Online]. Available: https://www.scopus.com

[69] Clarivate, “Welcome,” Web of Science. Accessed: Jan. 11, 2025. [Online]. Available: http://webofscience.com/

[70] Artifex, “Welcome to PyMuPDF,” PyMuPDF 1.25.2 documentation. Accessed: Jan. 22, 2025. [Online]. Available: https://pymupdf.readthedocs.io/en/latest/

[71] Microsoft, “Micorsoft Excel,” Microsoft 365. Accessed: Jan. 30, 2025. [Online]. Available: https://www.microsoft.com/de-de/microsoft-365/excel

[72] Corporation for Digital Scholarship, “Your personal research assistant,” Zotero. Accessed: Jan. 12, 2025. [Online]. Available: https://www.zotero.org/

[73] *exiftool/exiftool*. (Jan. 22, 2025). Perl. ExifTool by Phil Harvey. Accessed: Jan. 22, 2025. [Online]. Available: https://github.com/exiftool/exiftool

[74] Apple Inc., “Vorschau – Benutzerhandbuch für den Mac,” Apple Support. Accessed: Jan. 23, 2025. [Online]. Available: https://support.apple.com/de-de/guide/preview/welcome/mac

[75] *py-pdf/benchmarks*. (Jan. 21, 2025). Python. py-pdf. Accessed: Jan. 22, 2025. [Online]. Available: https://github.com/py-pdf/benchmarks

[76] ISO/IEC JTC 1/SC 7, *ISO/IEC 25022:2016, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of quality in use*, Jun. 2016.

[77] ISO/IEC JTC 1/SC 7, *ISO/IEC 25023:2016, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality*, Jun. 2016.

[78] ISO/IEC JTC 1/SC 7, *ISO/IEC 25024:2015, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of data quality*, Oct. 2015.

[79] The Kubernetes Authors, “kind,” kind. Accessed: Aug. 18, 2024. [Online]. Available: https://kind.sigs.k8s.io/

[80] The Kubernetes Authors, “minikube,” minikube. Accessed: Aug. 18, 2024. [Online]. Available: https://minikube.sigs.k8s.io/docs/

[81] The Kubernetes Authors, “Kubeadm,” Kubernetes. Accessed: Feb. 01, 2025. [Online]. Available: https://kubernetes.io/docs/reference/setup-tools/kubeadm/

[82] HashiCorp, Inc., “Automate infrastructure on any cloud with Terraform,” HashiCorp Terraform Community. Accessed: Feb. 22, 2025. [Online]. Available: https://www.terraform.io/

[83] Red Hat, Inc., “Ansible community documentation,” Ansible Documentation. Accessed: Feb. 22, 2025. [Online]. Available: https://docs.ansible.com/

[84] Red Hat, Inc., “Red Hat OpenStack Platform,” Red Hat Customer Portal. Accessed: Feb. 22, 2025. [Online]. Available: https://access.redhat.com/products/red-hat-openstack-platform

[85] Philippe Pepiot, “testinfra 10.1.2.dev2+ge1eb9c1 documentation,” Testinfra test your infrastructure. Accessed: Feb. 22, 2025. [Online]. Available: https://testinfra.readthedocs.io/en/latest/

[86] Microsoft, “Cloud Computing Services,” Microsoft Azure. Accessed: Feb. 22, 2025. [Online]. Available: https://azure.microsoft.com/en-us

[87] Canonical Ltd., “Ubuntu 24.04 LTS (Noble Numbat) daily [20250221],” Ubuntu. Accessed: Feb. 22, 2025. [Online]. Available: https://cloud-images.ubuntu.com/noble/current/

[88] Internet Systems Consortium, Inc., “BIND 9,” Internet Systems Consortium. Accessed: Mar. 06, 2025. [Online]. Available: https://www.isc.org/bind/

[89] Google, “Google - Alles über Google, unsere Unternehmenskultur und Neuigkeiten zum Unternehmen,” Google. Accessed: Mar. 06, 2025. [Online]. Available: https://about.google/intl/ALL_de/

[90] Google, “Google Public DNS,” Google for Developers. Accessed: Feb. 22, 2025. [Online]. Available: https://developers.google.com/speed/public-dns

[91] Inferno Nettverk A/S, “Dante - A free SOCKS server,” inet Dante. Accessed: Mar. 06, 2025. [Online]. Available: https://www.inet.no/dante/

[92] Resistance, “rancher 2.10.2 · Resistance/rancher-latest,” Artifact Hub. Accessed: Feb. 23, 2025. [Online]. Available: https://artifacthub.io/packages/helm/rancher-latest/rancher

[93] Willy Tarreau, “The Reliable, High Perf. TCP/HTTP Load Balancer,” HAProxy. Accessed: Feb. 23, 2025. [Online]. Available: https://www.haproxy.org/

[94] SUSE Rancher, “Previous v3 Rancher API Guide,” Rancher. Accessed: Feb. 22, 2025. [Online]. Available: https://ranchermanager.docs.rancher.com/v2.10/api/v3-rancher-api-guide

[95] Arm Ltd, “Architecture,” Arm | The Architecture for the Digital World. Accessed: Feb. 22, 2025. [Online]. Available: https://www.arm.com/architecture

[96] SPI and others, “The Universal Operating System,” Debian. Accessed: Feb. 22, 2025. [Online]. Available: https://www.debian.org/index.en.html

[97] SUSE Rancher, “Enabling the API Audit Log to Record System Events,” Rancher. Accessed: Feb. 22, 2025. [Online]. Available: https://ranchermanager.docs.rancher.com/v2.10/how-to-guides/advanced-user-guides/enable-api-audit-log

[98] MMXVIX, “Requests: HTTP for Humans™,” Requests 2.32.3 documentation. Accessed: Feb. 24, 2025. [Online]. Available: https://requests.readthedocs.io/en/latest/

[99] Allan Freitas, *the-best-way-to-repeatedly-execute-a-function-every-x-seconds-in-python*. Accessed: Feb. 24, 2025. [Online]. Available: https://gist.github.com/allanfreitas/e2cd0ff49bbf7ddf1d85a3962d577dbf

[100] Michael Kerrisk, “sar(1) - Linux manual page,” man7.org. Accessed: Feb. 24, 2025. [Online]. Available: https://man7.org/linux/man-pages/man1/sar.1.html

[101] *tmux/tmux*. Accessed: Feb. 24, 2025. [Online]. Available: https://github.com/tmux/tmux/wiki/Home

[102] Michael Kerrisk, “date(1) - Linux manual page,” man7.org. Accessed: Feb. 24, 2025. [Online]. Available: https://man7.org/linux/man-pages/man1/date.1.html

[103] MMXVIX, “Advanced Usage,” Requests 2.32.3 documentation. Accessed: Feb. 24, 2025. [Online]. Available: https://requests.readthedocs.io/en/latest/user/advanced/

[104] SUSE Rancher, “API Keys,” Rancher. Accessed: Feb. 24, 2025. [Online]. Available: https://ranchermanager.docs.rancher.com/reference-guides/user-settings/api-keys

[105] Michael Kerrisk, “ping(8) - Linux manual page,” man7.org. Accessed: Feb. 24, 2025. [Online]. Available: https://man7.org/linux/man-pages/man8/ping.8.html

[106] SPI and others, “Debian Official Cloud Images,” Debian. Accessed: Feb. 23, 2025. [Online]. Available: https://cdimage.debian.org/images/cloud/

[107] Canonical Ltd., “Snapcraft - Snaps are universal Linux packages,” Canonical Snapcraft. Accessed: Feb. 26, 2025. [Online]. Available: https://snapcraft.io/

[108] *kubernetes-sigs/metrics-server*. (Feb. 21, 2025). Go. Kubernetes SIGs. Accessed: Feb. 21, 2025. [Online]. Available: https://github.com/kubernetes-sigs/metrics-server

[109] Canonical Ltd., “Debian,” Ubuntu. Accessed: Mar. 02, 2025. [Online]. Available: https://ubuntu.com/community/governance/debian

[110] Microsoft, “Windows Server Operating System,” Microsoft. Accessed: Mar. 02, 2025. [Online]. Available: https://www.microsoft.com/en-us/windows-server

[111] G. Starke, “Shortcomings of ISO 25010,” arc42 Quality Model. Accessed: Mar. 03, 2025. [Online]. Available: https://quality.arc42.org/articles/iso-25010-shortcomings

[112] ACM, Inc., “ACM Digital Library.” Accessed: Jan. 11, 2025. [Online]. Available: https://dl.acm.org/

[113] Google, “Google Scholar,” Google Scholar. Accessed: Mar. 03, 2025. [Online]. Available: https://scholar.google.com/

[114] Google, “Ngram Viewer Exports,” Google Books. Accessed: Jan. 21, 2025. [Online]. Available: https://storage.googleapis.com/books/ngrams/books/datasetsv3.html

[115] NGRAMS, “NGRAMS,” NGRAMS. Accessed: Jan. 21, 2025. [Online]. Available: https://ngrams.dev

[116] G. Petralia, “[OVN] SNAT only happens for subnets directly connected to a router,” Ubuntu neutron package. Accessed: Mar. 03, 2025. [Online]. Available: https://bugs.launchpad.net/bugs/2051935

[117] Canonical Ltd., “Charmed OpenStack,” Canonical OpenStack. Accessed: Mar. 06, 2025. [Online]. Available: https://ubuntu.com/openstack

[118] Docker, Inc., “Pulls,” Docker Documentation. Accessed: Mar. 03, 2025. [Online]. Available: https://docs.docker.com/docker-hub/usage/pulls/

[119] The cert-manager Authors and The Linux Foundation, “Cloud native certificate management,” cert-manager. Accessed: Mar. 03, 2025. [Online]. Available: https://cert-manager.io/

[120] Google, “Build simple, secure, scalable systems with Go,” The Go Programming Language. Accessed: Mar. 06, 2025. [Online]. Available: https://go.dev/

[121] Rust Team, “A language empowering everyone to build reliable and efficient software,” Rust Programming Language. Accessed: Mar. 06, 2025. [Online]. Available: https://www.rust-lang.org/

[122] GitHub, Inc., “GitHub · Build and ship software on a single, collaborative platform,” GitHub. Accessed: Mar. 06, 2025. [Online]. Available: https://github.com/

[123] The Linux Foundation, “Paralus,” Paralus. Accessed: Mar. 06, 2025. [Online]. Available: https://paralus.io/

[124] Portainer, “Kubernetes and Docker Container Management Software,” portainer.io. Accessed: Mar. 06, 2025. [Online]. Available: https://www.portainer.io/

[125] Gartner, Inc., “Information Technology Gartner Glossary,” Gartner. Accessed: Mar. 07, 2025. [Online]. Available: https://www.gartner.com/en/information-technology/glossary

[126] National Institute of Standards and Technology, “Glossary | CSRC,” NIST. Accessed: Mar. 07, 2025. [Online]. Available: https://csrc.nist.gov/glossary/term/information_technology

[127] Tigera, Inc., “Kubernetes CNI,” Tigera. Accessed: Mar. 07, 2025. [Online]. Available: https://www.tigera.io/learn/guides/kubernetes-networking/kubernetes-cni/

[128] Python Software Foundation, “csv — CSV File Reading and Writing,” Python documentation. Accessed: Mar. 07, 2025. [Online]. Available: https://docs.python.org/3/library/csv.html

[129] Dataprise, “IT Terms Glossary | Information Technology Definitions,” Dataprise. Accessed: Mar. 07, 2025. [Online]. Available: https://www.dataprise.com/it-glossary/

[130] Fortinet, Inc., “What is ICMP (Internet Control Message Protocol)?,” Fortinet. Accessed: Mar. 07, 2025. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/internet-control-message-protocol-icmp

[131] D. L. Mills and M/A-COM Linkabit, “Network Time Protocol (NTP),” Internet Engineering Task Force, Request for Comments RFC 958, Sep. 1985. doi: 10.17487/RFC0958.

[132] Michael Kerrisk, “ssh(1) - Linux manual page,” man7.org. Accessed: Mar. 07, 2025. [Online]. Available: https://man7.org/linux/man-pages/man1/ssh.1.html

[133] Microsoft, “Visual Studio Code - Code Editing. Redefined,” Visual Studio Code. Accessed: Mar. 07, 2025. [Online]. Available: https://code.visualstudio.com/

[134] OpenAI, “ChatGPT,” ChatGPT. Accessed: Mar. 07, 2025. [Online]. Available: https://chatgpt.com

[135] Microsoft, “Microsoft Copilot: Ihr KI-Begleiter,” Microsoft Copilot. Accessed: Mar. 07, 2025. [Online]. Available: https://copilot.microsoft.com

[136] DeepL SE, “DeepL Übersetzer: Der präziseste Übersetzer der Welt,” DeepL. Accessed: Mar. 07, 2025. [Online]. Available: https://www.deepl.com/translator

[137] Microsoft, “Microsoft Word,” Microsoft 365. Accessed: Mar. 07, 2025. [Online]. Available: https://www.microsoft.com/de-de/microsoft-365/word

[138] JGraph Ltd, “Security-first diagramming for teams,” draw.io. Accessed: Mar. 07, 2025. [Online]. Available: https://app.diagrams.net/

Appendix A: Abbreviations

Sorted alphabetically:

Abbrev.	Full Form	Notes
AI	Artificial Intelligence	
AKS	Azure Kubernetes Service	
API	Application Programming Interface	
BSI	German Federal Office for Information Security	Ger.: Bundesamt für Sicherheit in der Informationstechnik
Cluster	Kubernetes cluster	usage depends on context: refer to *highlighting*
CPU	Central Processing Unit	
CSV	Comma Separated Values	
DNS	Domain Name System	
EKS	Amazon Elastic Kubernetes Service	
GKE	Google Kubernetes Engine	
HTTP(S)	Hypertext Transfer Protocol (Secure)	
ICMP	Internet Control Messaging Protocol	
NTP	Network Time Protocol	
OpenStack	Red Hat OpenStack Platform	Other OpenStack distributions exist; these are mentioned explicitly
PDF	Portable Document Format	
Private WAN	private wide area network	
Q42	arc42 Quality Model	
quality	software quality	usage depends on context: refer to *highlighting*
SOCKS	Socket Secure	
Software	software system	usage depends on context: refer to *highlighting*
SQuaRE	Systems and Software Quality Requirements and Evaluation	
SSH	Secure Shell	
TLS	Transport Layer Security	

UI	user interface	
UML	Unified Modeling Language	
VAN	Virtual Application Network	
VPN	Virtual Private Network	

Appendix B: Glossary

Terms, which were not explained within context, sorted alphabetically:

Term	Context	Description
AI	General	"Artificial intelligence (AI) applies advanced analysis and logic-based techniques, including machine learning, to interpret events, support and automate decisions, and take actions" [125].
API	General	"An application programming interface (API) is an interface that provides programmatic access to service functionality and data within an application or a database. It can be used as a building block for the development of new interactions with humans, other applications or smart devices" [125].
Container	Kubernetes	"A method for packaging and securely running an application within an application virtualization environment. Also known as an application container or a server application container" [126].
Container Networking Interface	Kubernetes	The "Container Network Interface (CNI) is a framework for dynamically configuring networking resources. […] When used with Kubernetes, CNI can […] automatically configure the network between pods" [127].
CPU	General	"The component of a computer system that controls the interpretation and execution of instructions. […] The term 'processor' is often used to refer to a CPU" [125].
CSV	General	"The […] CSV […] format is the most common import and export format for spreadsheets and databases" [128].
DNS	General	A "service for accessing a networked computer by name rather than by numerical, (IP) address" [129].
HTTP	General	"HTTP […] is […] the protocol that governs the transfer of documents between servers and clients on the World Wide Web" [125].
HTTPS	General	"HTTPS […] is an extension of Hypertext Transport Protocol (HTTP) that provides security services for transaction confidentiality, authenticity and integrity between HTTP servers and clients" [125].

ICMP	General	"The Internet Control Message Protocol (ICMP) is a protocol that devices within a network use to communicate problems with data transmission" [130].
Infrastructure-as-Code	General	"The process of managing and provisioning an organization's IT infrastructure using machine-readable configuration files, rather than employing physical hardware configuration or interactive configuration tools" [126].
NTP	General	A "protocol for synchronizing a set of network clocks using a set of distributed clients and servers" [131].
PDF	General	A "type of formatting that enables files to be viewed on a variety [of] computers regardless of the program originally used to create them" [129].
SOCKS	General	"An Internet protocol to allow client applications to form a circuit-level gateway to a network firewall via a proxy service" [126].
SSH	General	SSH "is a program for logging into a remote machine and for executing commands on a remote machine. It is intended to provide secure encrypted communications between two untrusted hosts over an insecure network" [132].
TLS	General	"Internet-based transaction security provided by the Secure Sockets Layer (SSL) protocol" [125].
UI	General	"The physical or logical means by which users interact with a system, device or process" [126].
VPN	General	"A virtual network built on top of existing networks that can provide a secure communications mechanism for data and IP information transmitted between networks" [126].

Appendix C: Literature Review

Device

The code was developed and executed on a MacBook Pro (2023) with:

```
Chip:       Apple M2 Pro
Memory:     32 GB
OS:         macOS Sequoia 15.3.1
```

Also, the manual review steps were conducted on this device.

Review Protocol Excel File

The review protocol ***Excel*** file can be obtained from https://leonkrass.dev/master-thesis or the attached SD card.

Zotero Database Export

The review ***Zotero*** database export file can be obtained from https://leonkrass.dev/master-thesis or the attached SD card.

Automated Full-Text Review Code

The ***Python*** code for the automated full-text review can be obtained from https://leonkrass.dev/master-thesis or the attached SD card. The code was developed using ***JetBrains PyCharm Professional Edition*** version 2024.3.4 with the following plugins:

- .ignore
- Direnv Integration
- JetBrains AI Assistant
- Kubernetes
- NixIDEA
- Rainbow Brackets
- Space

Full Quality Criteria Table

Model	Characteristic	Sub-Characteristic	Count of Docs.	Coverage
ISO/IEC 25010	reliability	**availability**	51	76.12%
ISO/IEC 25010	flexibility	**scalability**	45	67.16%
ISO/IEC 25010	performance efficiency	**resource utilization**	34	50.75%
ISO/IEC 25019	acceptability	**experience**	32	47.76%
ISO/IEC 25010	performance efficiency	**capacity**	30	44.78%
ISO/IEC 25010	reliability	**fault tolerance**	18	26.87%
ISO/IEC 25019	acceptability	**compliance**	13	19.40%
ISO/IEC 25010	compatibility	**interoperability**	9	13.43%
ISO/IEC 25010	interaction capability	**operability**	9	13.43%
ISO/IEC 25010	flexibility	**adaptability**	7	10.45%
ISO/IEC 25019	beneficialness	**usability**	7	10.45%
ISO/IEC 25010	security	**integrity**	6	8.96%
ISO/IEC 25010	maintainability	**modularity**	5	7.46%
ISO/IEC 25019	beneficialness	**suitability**	5	7.46%
ISO/IEC 25010	compatibility	**co-existence**	2	2.99%
ISO/IEC 25010	maintainability	**reusability**	2	2.99%
ISO/IEC 25010	reliability	**recoverability**	1	1.49%
ISO/IEC 25010	security	**accountability**	1	1.49%
ISO/IEC 25010	security	**authenticity**	1	1.49%
ISO/IEC 25010	security	**resistance**	1	1.49%
ISO/IEC 25010	maintainability	**testability**	1	1.49%
ISO/IEC 25019	beneficialness	**accessibility**	1	1.49%
ISO/IEC 25010	functional suitability	**functional completeness**	0	0.00%
ISO/IEC 25010	functional suitability	**functional correctness**	0	0.00%

ISO/IEC 25010	functional suitability	**functional appropriate-ness**	0	0.00%
ISO/IEC 25010	performance efficiency	**time behavior**	0	0.00%
ISO/IEC 25010	interaction capability	**appropriateness recog-nizability**	0	0.00%
ISO/IEC 25010	interaction capability	**learnability**	0	0.00%
ISO/IEC 25010	interaction capability	**user error protection**	0	0.00%
ISO/IEC 25010	interaction capability	**user engagement**	0	0.00%
ISO/IEC 25010	interaction capability	**inclusivity**	0	0.00%
ISO/IEC 25010	interaction capability	**user assistance**	0	0.00%
ISO/IEC 25010	interaction capability	**self-descriptiveness**	0	0.00%
ISO/IEC 25010	reliability	**faultlessness**	0	0.00%
ISO/IEC 25010	security	**confidentiality**	0	0.00%
ISO/IEC 25010	security	**non-repudiation**	0	0.00%
ISO/IEC 25010	maintainability	**analyzability**	0	0.00%
ISO/IEC 25010	maintainability	**modifiability**	0	0.00%
ISO/IEC 25010	flexibility	**installability**	0	0.00%
ISO/IEC 25010	flexibility	**replaceability**	0	0.00%
ISO/IEC 25010	safety	**operational constraint**	0	0.00%
ISO/IEC 25010	safety	**risk identification**	0	0.00%
ISO/IEC 25010	safety	**fail safe**	0	0.00%
ISO/IEC 25010	safety	**hazard warning**	0	0.00%
ISO/IEC 25010	safety	**safe integration**	0	0.00%
ISO/IEC 25019	freedom from risk	**freedom from economic risk**	0	0.00%
ISO/IEC 25019	freedom from risk	**freedom from environ-mental and societal risk**	0	0.00%
ISO/IEC 25019	freedom from risk	**freedom from health risk**	0	0.00%
ISO/IEC 25019	freedom from risk	**freedom from human life risk**	0	0.00%
ISO/IEC 25019	acceptability	**trustworthiness**	0	0.00%

Appendix D: Quality Measures

As special *quality measures* are optional by definition (see [76, Ch. 2], [77, Ch. 2]), the following justifications provided are kept concise and may reflect the authors perspective. The entries are sorted analogous to Table 3.2.

Selected Special Quality Measures

a) Resource Utilization: Bandwidth Utilization

The *evaluation targets* resemble management solutions, where low bandwidth utilization is crucial to ensure the managed *software* can utilize the majority of available bandwidth.

b) Fault Tolerance: Redundancy Of Components

In distributed systems, such as the *evaluation targets*, the redundancy of components is a key measure for assessing a systems fault tolerance.

c) Operability: Monitoring Capability

For *multi-cluster management*, the monitoring capability is vital. *Decentralized architectures* rely on centralized monitoring planes to aggregate and correlate data from numerous endpoints, making this measure essential its targets.

d) Recoverability: Backup Data Completeness

Without adequate backups, system recovery is impossible. Consequently, backup data completeness is an important consideration.

e) Accountability: System Log Retention

System logs are invaluable for troubleshooting and auditing access in administrative tools like the *evaluation targets*. Therefore, the system log retention is a relevant measure.

Omitted Special Quality Measures

a) Capacity: User Access Increase Adequacy

Adding a large number of users is generally uncommon for the specified *evaluation targets*, except in environments such as those managed by large cloud providers. Thus, user access increase adequacy is excluded from the *evaluation model* by default. If, for instance, a large cloud provider uses the *evaluation model*, the *quality measure* still can be incorporated.

b) Fault Tolerance: Mean Fault Notification Time

Since the given *evaluation targets* usually do not involve notification functions, the mean fault notification time is omitted. Instead, the monitoring capability measure is included, as monitoring systems generally provide this functionality.

c) Operability: Functional Customizability

The *evaluation targets* provide a defined set of functionalities for administrative tasks. Management solutions are commonly selected for their defined range of features, which generally do not require the level of customization often associated with end-user *software*.

d) Operability: User Interface Customizability

Analogous to Paragraph c).

e) Operability: Undo Capability

Analogous to Paragraph c).

f) Operability: Understandable Categorization of Information

Management solutions, such as the defined *evaluation targets,* demand an extensive expertise regarding the presented information. As a result, prior understanding of the given information is essential.

g) Operability: Appearance Consistency

Analogous to Paragraph c).

h) Operability: Input Device Support

Analogous to Paragraph c).

i) Adaptability: Operational Environment Adaptability

The given definition for “operational environment” in ***ISO/IEC 25023*** lacks clarity: The term “operational environment” is not further described. Consequently, this measure is dropped.

j) Integrity: Buffer Overflow Prevention

Buffer overflow prevention is classified as an internal measure. Since the *evaluation model* focuses solely on external measures, this measure is excluded.

k) Modularity: Cyclomatic Complexity Adequacy

Analogous to Paragraph j).

l) Reusability: Coding Rules Conformity

Analogous to Paragraph j).

m) Authenticity: Authentication Rules Conformity

Analogous to Paragraph 0.

n) Testability: Test Restartability

Analogous to Paragraph 0.

Appendix E: Evaluation

Device

The code was developed and executed on a MacBook Pro (Nov. 2023) with:

```
Chip:       Apple M3 Pro
Memory:     36 GB
OS:         macOS Sequoia 15.3.1
```

Code

The code for the evaluation can be obtained from https://leonkrass.dev/master-thesis or the attached SD card. It code was developed using ***Visual Studio Code*** [133] version 1.97.2 with the following plugins:

```
Pylance
Python
Python Debugger
Remote - SSH
YAML
1Password
Ansible
direnv
German Language Pack for Visual Studio Code
HashiCorp HCL
HashiCorp Terraform
Nix IDE
Prettier - Code formatter
Remote - SSH: Editing Configuration Files
Remote Explorer
```

Refer to the **`README.md`** file in the "kubeanchors" folder to obtain further information on the project structure, project usage and on the *software* versions used. Used third-party libraries are referenced within the source code.

For the hardware/operating system modulation scenario, a separate folder "kubeanchors-debian" was created, containing the adjustments for ***Debian***.

Appendix F: Measurement Results

The raw measurement results (log files) can be obtained from https://leonkrass.dev/master-thesis or the attached SD card.

Zusätzliche Hilfsmittel

Neben den in der Arbeit referenzierten Hilfsmitteln, wurden ***OpenAI ChatGPT*** [134] und ***Microsoft Copilot*** [135] ausschließlich zur Überprüfung der sprachlichen Korrektheit verwendet. Übersetzungen einzelner Ausdrücke wurden per ***DeepL*** [136] recherchiert.

Zur Verfassung dieses Dokumentes wurde ***Microsoft Word*** [137] inkl. Features zur Prüfung der sprachlichen Korrektheit verwendet. Abbildungen wurden mit ***draw.io*** [138] und Diagramme mit ***Microsoft Excel*** [71] erstellt. Zitation und Quellverzeichnis wurden mit ***Zotero*** [72] erstellt.

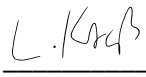
Im Quellcode verwendete Bibliotheken und Abhängigkeiten sind den entsprechenden Anhängen zu entnehmen.

Eigenständigkeitserklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen und ist noch nicht veröffentlicht worden. Alle zu deren Beurteilung verwendeten, auch elektronischen Kopien dieser Arbeit haben genau den gleichen Inhalt wie dieses gedruckte Exemplar. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben wird.

Stade, 07.03.2025

Ort, Datum



Unterschrift