

## KubeLab

Investigating the Use of Containers and Management Tools for Laboratory Environments

Florian Reitz

Master's Thesis

First Supervisor: Prof. Dr.-Ing. David Strippgen

Second Supervisor: Prof. Dr. Klaus Jung

Berlin, 30.08.2023

---

## Acknowledgments

I would like to use this opportunity to thank my supervisors, Professor Strippgen and Professor Jung, for the input they provided to this thesis, as well as all the friends who helped out by proofreading or providing support and an open ear. Specifically, I want to thank Gisella Vogel Sanchez and Martin Kock.

The inspiration behind this project stems from the personal experiences of both myself and fellow computer science students during our academic journey. Many of us, including myself, faced difficulties in the initial weeks of various courses. Some students even dropped out due to problems in making the provided samples and initial projects work correctly. Addressing these issues often meant investing considerable time into debugging. And even after solving a problem for one student, different computer setups led to various errors among other students.

Additionally, the exigencies of the pandemic further underscored the necessity for improvements in this regard. A vivid memory involves a review session with an instructor, which had extended into late hours, due to the lengthy and inconvenient review process at hand.

Berlin, 2023  
Florian Reitz

---

# Abstract

## *German*

In dieser Arbeit soll untersucht werden, ob Containerumgebungen als Alternative zu Computerlaboren infrage kommen und welche Vorteile sie bieten. Hierfür wurde die Frage gestellt: *Wie können Container für den Aufbau von Laborumgebungen genutzt werden?*

Zur Beantwortung der Frage wird zunächst Kubernetes untersucht, eine Software, die zur Orchestrierung einer großen Anzahl von Containern eingesetzt wird. Anschließend wird ein Operator angefertigt, der über eigene Einträge in der Kubernetes-API Ressourcen erstellt, die von Studierenden und Lehrenden genutzt werden können. Um den Studierenden die Möglichkeit zu geben, sich mit den Containern zu verbinden und deren Status zu verwalten, wird eine Webanwendung gebaut, welche die Authentifizierung zum Kubernetes-Cluster über einen OpenID-Connect-Provider vornimmt. Dozenten können über dieselbe Schnittstelle sowohl auf die Container ihrer Studenten zugreifen als auch neue Dateien hochladen. Um die Erstellung neuer Kubernetes-Objekte für den Operator zu vereinfachen, wurde Ansible gewählt, welches mithilfe einer CSV-Datei die Erstellung der Klassen und Studenten ermöglicht. Zudem werden Mechanismen untersucht, die zu der Verbesserung der Sicherheit verwendet werden können. Des Weiteren wird die Leistung von Containern ausgewertet. Abschließend werden die Ergebnisse und weitere Forschungsmöglichkeiten diskutiert.

Diese Masterarbeit veranschaulicht die Realisierbarkeit eines containerbasierten Labors innerhalb eines Kubernetes-Umfelds. Dennoch, wie die vorliegende Arbeit aufzeigt, bedarf es weiterer Forschung für einige generelle sowie Sicherheitsaspekte.

## *English*

This study aims to assess the feasibility and potential advantages of employing container environments as an alternative to traditional computer laboratories. The main inquiry that this contribution seeks to address is: *How can containers be of use in the development of computer laboratories?*

In order to answer it, this work sets its focus on Kubernetes, a software widely employed for orchestrating large volumes of containers. Within this framework,

an Operator is built, utilizing custom entries within the Kubernetes API to generate resources that can be readily accessed by both students and teaching staff. A web application is constructed, enabling students to establish connections and effectively manage the state of their container. The prototype manages Authentication to the Kubernetes Cluster via an OpenID-Connect provider. Additionally, it provides lecturers with access to their students' containers and afforded the capability to upload files to a shared space accessible to every student within the class. To simplify the creation of new Kubernetes resources for the Operator, Ansible was employed. This choice aims to make the process of generating classes and students more straightforward by utilizing CSV files. Following that, an examination of viable security mechanisms within the project is carried out. Additionally, the performance of containers is discussed. Concluding this study, an evaluation of the outcomes is presented and potential avenues for further research are outlined.

This master's thesis shows that building a container-based laboratory inside Kubernetes is feasible and offers a series of advantages. Nevertheless, some general and security concerns persist and need to be addressed in further research endeavors.

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>4</b>
<b>3</b>	<b>Fundamentals</b>	<b>8</b>
3.1	Container	8
3.1.1	Definition	8
3.1.2	Security	12
3.2	Kubernetes	14
3.2.1	Components	15
3.2.2	Concepts	17
3.2.3	Security	21
3.2.4	Custom Resources	24
3.2.5	Operators	25
3.3	Frameworks and Tools	27
3.3.1	SvelteKit	28
3.3.2	Management Tools	28
<b>4</b>	<b>Project Design</b>	<b>30</b>
4.1	Requirements	30
4.1.1	Should	30
4.1.2	Can	31
4.1.3	Workflows	31
4.2	Cluster	32
4.2.1	Hardware	32
4.2.2	Design	33
4.3	Operator	35
4.3.1	Kubelab-User	35
4.3.2	Classroom	37
4.3.3	Framework	41
4.4	Web-Application	41
4.4.1	Requirements	41
4.4.2	User, Class, and Deployment Management	42

---

4.5	Container .....	43
4.5.1	Student Container .....	43
4.5.2	Web Application Container .....	44
4.5.3	Operator Container .....	44
<b>5</b>	<b>Cluster Creation .....</b>	<b>45</b>
5.1	Setup .....	45
5.2	Add-ons .....	46
5.3	OpenID-Connect .....	49
5.3.1	Configuring Keycloak .....	49
5.3.2	Configuring Kubernetes .....	49
<b>6</b>	<b>Implementation .....</b>	<b>53</b>
6.1	Operator .....	53
6.1.1	Setup .....	53
6.1.2	Kubelab-User .....	54
6.1.3	Classroom .....	57
6.2	Web Application .....	59
6.2.1	Keycloak Integration .....	59
6.2.2	Kubernetes Integration .....	60
6.2.3	Functions .....	60
6.2.4	Challenges .....	62
6.3	Management Tools .....	64
6.3.1	Reviewed Tools .....	64
6.3.2	Chosen Tool .....	65
6.4	Containerization .....	66
6.4.1	Student Container .....	66
6.4.2	Web Application Container .....	68
6.4.3	Operator Container .....	70
<b>7</b>	<b>Security and Performance .....</b>	<b>72</b>
7.1	Security .....	72
7.2	Performance .....	75
<b>8</b>	<b>Results and Discussion .....</b>	<b>76</b>
<b>9</b>	<b>Conclusion .....</b>	<b>80</b>
	References .....	82
	Glossary .....	94
	Declaration of autonomy .....	95

---

## List of Figures

3.1	Comparison between containers and virtual machines [1] .....	8
3.2	ps-command in a new PID namespace .....	9
3.3	ps-command inside a simple container .....	10
3.4	Example usage of unshare -net .....	10
3.5	All containerized components of Kubernetes .....	15
4.1	Diagram of the Prototype .....	32
4.2	Diagram of the Kubelab-User Custom Resource .....	36
4.3	Diagram of Keycloak vs. Kubelab-Users .....	37
4.4	Diagram of the Classroom Custom Resource .....	38
4.5	Diagram of NodePort setup .....	40
5.1	Keycloak mapper configuration .....	50
5.2	Data received from Keycloak .....	50
5.3	OIDC options for Kubernetes .....	51
6.1	Folder structure of Kubebuilder .....	54
8.1	Teacher's user interface .....	76
8.2	Student's user interface .....	77

## Introduction

During the pandemic and with the rise of distance learning, the need for more flexible computer laboratories has increased. Furthermore, certain academic modules often require special software configurations for their assignments, these being sometimes multiple versions behind the newest software release. This, combined with many students' unfamiliarity with the technology in use, results in long debugging sessions to get the sample projects running.

The revision process introduced an additional technical hurdle, notably exacerbated by the circumstances of the pandemic. Lecturers mandated the submission of project files by students, who then faced the uncertainty of whether these projects would compile and execute as they did in their personal computing environments. An alternative method of assessment involved the use of screen sharing, a technique that occasionally resulted in a reduction in the pace of the process due to the instructors' limited ability to directly access the source code. In these circumstances, tutors relied on students to present the code segments that needed to be examined closer.

While there exist concepts for systems that attempt to address these problems, the available online resources do not appear promising. Many of these projects are closed source and their system descriptions often remain at a relatively superficial level, for example the studies by Irvine et al. [2] or Pandeewari et al. [3]. In light of the recent surge in container<sup>1</sup> usage, there remains a lack of extensive research concerning applications that lie beyond their primary intended scope. The only projects using Kubernetes<sup>2</sup> to a similar extent are Crownlabs [4], which differs from this work by the usage of the more resource intensive virtual machines and CvLabs [5], which aims to build an E-learning platform instead of the more unrestricted approach of this project.

In an endeavor to address the issues mentioned above, and add further research into other container use cases, this thesis aims to answer the following question: *How can containers be of use in the development of computer laboratories?*

To clarify this inquiry, the following theses are examined:

- Containers are a light and quick alternative to virtual and bare-metal machines.

---

<sup>1</sup> A software bundle that makes it possible to run software in different computing environments and isolating it from the host

<sup>2</sup> Container Orchestration and Management Tool



- Containers can be accessed remotely by students and lecturers to develop within the container.
- Containers are sufficiently isolated to allow safe access by students.

To assess these matters and test the feasibility of facilitating work within containers for both students and lecturers, a prototype will be developed. Additionally, a review of prevailing security considerations and container performance will be conducted.

Firstly, the background to this work is explained, delving into the current landscape of container laboratory research and also reviewing security and performance considerations. Key concepts are then defined, including the essential elements of a container itself and the container orchestration utility Kubernetes.

Secondly, the various components used to build the prototype are discussed in the design chapter. The prototype design consists of four different parts. The first one is the design of the server cluster, which includes the authorization and storage components, as well as Kubernetes itself. The second is the design of the Operator, which manages the Kubernetes resources that are needed for the prototype. For example, it assigns storage and creates the permissions needed for students to use their container. The third is the design of the web application. It offers two different workflows: one for students, where they can start and stop their class containers, as well as get the connection string to connect via SSH<sup>3</sup>. It is also possible to upload an SSH key to avoid having to authenticate using a password. The other workflow is designed for lecturers. After logging in, lecturers are presented with the following options: 1. To view all their classes and the list of students they contain. This allows them to start or stop their students' containers and connect to them with elevated rights. 2. To upload files to be used by the class and make them available to each of its students. Thereafter, the requirements for the management tool are outlined. Lastly, attributes that are essential for the container, students will connect to, are examined. This includes the mount points and SSH setup.

Once all the components have been designed, the setup of the cluster is discussed. This encompasses both the overall setup of the cluster and the modifications needed to ensure its operational functionality.

The following implementation chapter provides an overview of the procedure involved in developing the remaining components of the prototype, namely: A. The Operator, which manages the resources of the Kubernetes cluster; B. The web application, C. The management tool, which facilitates the creation of classes and students and D. The creation of the students' container, as well as the containerization of the written applications.

Upon the successful creation of a functional prototype, the project subsequently delves into an analysis of security and performance aspects. This section explores concepts that may be employed to enhance security. Examples include the usage of a second kernel to better manage system calls or the implementation of permission systems that enhance the default one.

---

<sup>3</sup> Secure Shell, a way to connect and control a system

Prior to finalizing the study, an assessment is conducted to validate the prototype's alignment with the initial design objectives, alongside an evaluation of the feasibility of a container-based system for educational purposes.

Before embarking on the project, it is essential to highlight a few remarks. Firstly, the project's source code can be found under: <https://github.com/troppes/kubelab>. Secondly, to ensure clear visibility, all commands are presented in the format `$ ps -a`. Lastly, it is noteworthy that all concepts belonging to Kubernetes are denoted with an initial capital letter.

## Background

The landscape of containers and the broader cloud environment is experiencing a rapid and significant progress. Year by year, the adoption of containers is steadily increasing, showcasing a notable upward trajectory [6]. To regulate the expanding industry's standards, a governing body known as the Open Container Initiative (OCI) was formed. As a component of the broader Linux Foundation, the Open Container Initiative works towards establishing vendor-neutral guidelines for containers. At present, OCI has formulated standards for container images, runtimes, and distribution. [7]

Another significant entity in the realm of containers and cloud technology is the Cloud Native Computing Foundation (CNCF), which extends support to open-source projects beyond code management. This assistance encompasses a range of services, such as marketing, travel reimbursements, and legal support, all aimed at sustaining the project's holistic requirements. This entity is also a part of the Linux foundation. [8]

A notable project emerging within this foundations' initiative is Kubernetes. Born as an open-source project led by Google, Kubernetes facilitates the efficient orchestration of extensive collections of containers. [9] The choice of Kubernetes for this endeavor is driven by its open-source nature and its inherent ability to be extended through Custom Resources and Operators. Furthermore, with a proven track record of over a decade and a large developer community, Kubernetes provides ample online support resources. While this system is not specifically made for the task of creating remote laboratories, its features set can be leveraged for this task, as the last of the projects to be subsequently mentioned shows.

The idea of creating remotely operated laboratory systems for educational purposes has been around for a long time, yet it received increased attention in the context of the coronavirus pandemic, as it highlighted the need to facilitate online learning. Similarly, containers have long been used in education, particularly in cybersecurity courses. For example, Irvine et al. [2] designed a framework that relies on Docker<sup>1</sup> containers to be used by students. Cano et al. [10] have developed and tested a remote laboratory to be used in their cybersecurity classes, based on Docker as well.

---

<sup>1</sup> Container Vendor <https://www.docker.com/>

Amidst the pandemic, Bußler et al. [11] created and evaluated a system, in which teachers can deploy collaboration tools through a web interface using containers.

Pandeeswari et al. undertook a study focused on creating a container-based laboratory using the Docker engine, aimed at providing students with containers tailored for their academic work [3]. They employed a personalized PHP-based backend to interface with the container engine. The study’s findings highlight that, in contrast to virtual machines, the container-based approach turns out to be more lightweight, utilizing fewer storage resources.

While aligned with preceding endeavors in its aim to establish a container-based laboratory for student use, this undertaking diverges in its approach. Unlike relying on the Docker engine and a PHP backend, this project opts for a tight integration with Kubernetes. This choice is driven by Kubernetes’ inherent capabilities for the creation of laboratory environments, such as supporting server clusters, intelligently assigning container workloads to different Nodes, and facilitating scalability.

There are already some projects using Kubernetes to deploy their container workloads in educational settings. Particularly noteworthy is the analysis and discussion put forth by Iorio et al. [4], who developed CrownLabs. In this system, KubeVirt<sup>2</sup> is employed to establish virtual machines, providing a collaborative environment for students.

Another relevant contribution stems from a project called CvLabs [5]. This initiative concentrates on establishing an inclusive learning platform, wherein tasks are assigned, and users can explore potential solutions within an accessible shell, conveniently hosted on a website.

Both CrownLabs and CvLabs utilize Kubernetes to manage the underlying machines. While the study presenting CvLabs [5] does not delve deeply into the system’s inner workings, Iorio et al. [4] elucidate CrownLabs’ implementation, employing Kubernetes Custom Resources alongside a Controller/Operator to manage the infrastructure. Both systems provide web interfaces, allowing users to allocate and decommission Containers/VMs.

Although CrownLabs [4] follows a similar approach to this endeavor, the key distinction lies in its core purpose. While this work concentrates on containers and SSH access to the system, CrownLabs currently employs VNC<sup>3</sup> and virtual machines to address its remote laboratory requirements. Iorio et al. propose examining and developing analogous features as a potential pathway for future exploration. However, they also express concern regarding potential isolation issues when employing containers.

Although the CrownLabs project is open-source and could have been considered as a starting point for this prototype, it was deemed unsuitable due to its extensive built-in functionalities that do not align with the objectives of this work. Utilizing the existing code would have demanded substantial alterations, including the removal of essential features like virtual machine creation and the corresponding VNC solution. Implementing these changes would have led to a distinct devia-

<sup>2</sup> Technology to deploy VMs into a Kubernetes cluster <https://kubevirt.io/>

<sup>3</sup> Virtual Network Computing, a solution to see and interact with a remote desktop environment

tion from the original project, making the integration of future updates into the new codebase considerably challenging. Nonetheless, the study presenting Crown-Labs is employed across various contexts within this thesis to offer an alternative viewpoint on specific challenges.

CvLabs [5] employs its system to establish a comprehensive learning platform, similar to commercial platforms like KodeKloud<sup>4</sup> or Katacoda, before its integration into O'Reilly in 2022 [12]. This diverges significantly from the platform objectives of this project, where containers are generated for unrestricted utilization rather than for a designated task.

Many of the aforementioned projects, at the very least, acknowledge some concerns about security implications associated with the utilization of containers. This, combined with the escalating damages in cyberattacks each year [13], creates an increasing demand for enhanced mitigation and security measures.

In line with this, numerous academic papers set their focus on container security. In a 2018 study, Xin Lin et al. highlighted the critical link between a container's security and the integrity of its underlying kernel. Their research involved subjecting 223 exploits to testing, revealing eleven vulnerabilities capable of breaching isolation. Significantly, they also proposed defense mechanisms to counter all identified vulnerabilities. [14]

Shu et al. created DIVA (Docker image vulnerability analysis), a tool designed for scrutinizing images on Docker Hub. In their research, they conducted scans on more than 350,000 images and discovered that 80% of all images contained at least one vulnerability at a high severity level. Moreover, approximately 50% of the images had not been updated in 200 days. [15]

In 2020, Wist et al. conducted a study, in which they found that 82% of certified images and 45.9% of official images contained at least one high-rated vulnerability. Particularly notable was the prevalence of high and critical vulnerabilities within Python and JavaScript library functions. The study also revealed that only 29.8% of the scanned images were up-to-date, having been updated within the last 14 days. [16]

Viktorsson et al. conducted an assessment of performance disparities between runc and more virtualized alternatives like Kata-Containers<sup>5</sup>, which generates a virtual machine around the container, and gVisor<sup>6</sup>, which introduces a guest kernel to intercept system calls. Despite the heightened security offered by both solutions, their adoption resulted in a reduction in performance, sometimes plummeting to as low as 20% of runc's efficiency. [17]

Abbas et al. introduced an intrusion detection system named PACED, tailored to identifying container escapes. Distinguished by its provenance-based approach, the tool documents the origin and evolution of data objects, facilitating the identification of anomalies. The system underwent rigorous testing against established

---

<sup>4</sup> A platform to learn DevOps Concepts <https://kodekloud.com/>

<sup>5</sup> <https://katacontainers.io>

<sup>6</sup> <https://gvisor.dev>

CVEs<sup>7</sup>, displaying an almost flawless capacity to detect container escapes. Notably, PACED is an open-source resource that can be accessed online<sup>8</sup>. [18]

Sultan et al. conducted an extensive examination of the existing landscape of container security, highlighting potential strategies for enhancing security through diverse tools. Their investigation explores multiple attack vectors and outlines plausible mitigation strategies, encompassing both software-based and hardware-based approaches, such as trusted platform modules. [19]

These studies stress the significance of giving precedence to security considerations in research concerning this topic. This aspect is also critical within the context of the present work to guarantee the safety of the platform being developed for both students and lecturers. Another relevant aspect for large-scale applications, such as the one discussed in this work, is performance.

Container performance has been extensively measured against virtual machines. For instance, Potdar et al. assessed Docker-based containers and demonstrated that their performance was superior across multiple metrics, including CPU, memory, Disk I/O, load testing, and operational speed. These evaluations were executed using benchmarking tools. [20] Similarly, Joy et al. arrived at a comparable conclusion in their own testing, highlighting containers' enhanced scalability due to their minimal overhead [21]. Sharma et al. yielded similar outcomes, showcasing containers' capacity for nearly bare-metal performance, while also expressing concerns about their relatively weaker isolation and security implications [22].

The increased performance and adaptability offered by containers present a compelling argument for their adoption in large-scale environments like educational institutions, as opposed to virtual or bare-metal machines. Despite this, the weaker isolation remains a significant topic, which will be further discussed in the upcoming chapters.

---

<sup>7</sup> Common Vulnerabilities and Exposures, database of known vulnerabilities

<sup>8</sup> <https://github.com/PACED-prov>

## Fundamentals

To provide a clearer understanding of the concerns discussed in the previous background chapter, this section aims to define containers and outline their key elements. Furthermore, within this chapter, an explanation of the various software and concepts employed in this study is presented.

### 3.1 Container

The objective of the current section is to establish a clear definition of containers and clarify mechanisms that enhance security.

#### 3.1.1 Definition

When researching the concept of containers or comparing them to virtual machines, a visual representation similar to figure 3.1 is commonly found. Such graphics illustrate that containers operate within the host operating system and are consequently more lightweight. However, a precise description of the fundamental components constituting containers is frequently lacking.

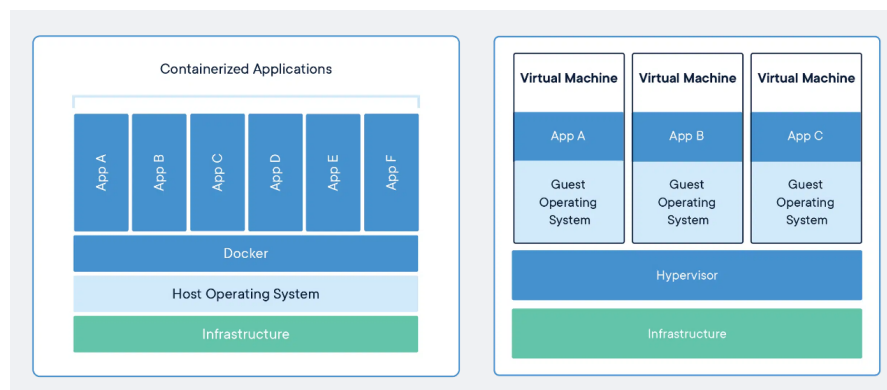


Fig. 3.1: Comparison between containers and virtual machines [1]

In a broad sense, containers represent processes that operate within the operating system and are isolated through the utilization of Linux namespaces and

additional tools. To better understand the security offered by containers, this section describes the various isolation mechanisms they are made of.

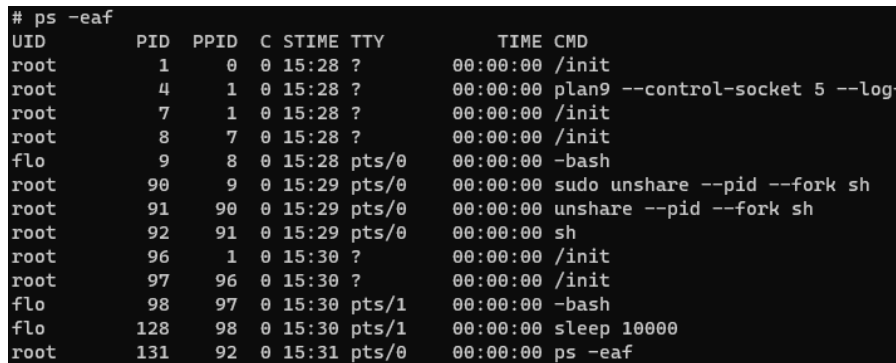
### *Unix Timesharing System Namespace*

Although the term may imply otherwise, this namespace allows the establishment of a distinct hostname and NIS<sup>1</sup> domain name compared to the host. To achieve this, the `$ unshare` command comes into play. This command enables the extraction of specific aspects from the execution context of the host and puts them into new isolated namespaces [23].

### *Process ID Namespace*

Secondly, the PID namespace allows the assignment of new IDs for processes. For instance, by executing `$ unshare` with appropriate parameters, PIDs can be reset to start from 1 again, even though they might already be in use when observed from the host's perspective. [24, p. 35ff.] It is noteworthy that solely employing the PID namespace and running the `$ ps` command within a namespaced shell could still reveal processes from the host system, as `$ ps` operates by accessing information from the `proc` directory [25].

To observe this process, the command `$ unshare --pid --fork sh` can be run. Subsequently, in a separate terminal window, the `sleep` command has to be executed. Within the namespaced shell, the presence of the `sleep` command becomes evident, as depicted in Figure 3.2.



UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	15:28	?	00:00:00	/init
root	4	1	0	15:28	?	00:00:00	plan9 --control-socket 5 --log-
root	7	1	0	15:28	?	00:00:00	/init
root	8	7	0	15:28	?	00:00:00	/init
flo	9	8	0	15:28	pts/0	00:00:00	-bash
root	90	9	0	15:29	pts/0	00:00:00	sudo unshare --pid --fork sh
root	91	90	0	15:29	pts/0	00:00:00	unshare --pid --fork sh
root	92	91	0	15:29	pts/0	00:00:00	sh
root	96	1	0	15:30	?	00:00:00	/init
root	97	96	0	15:30	?	00:00:00	/init
flo	98	97	0	15:30	pts/1	00:00:00	-bash
flo	128	98	0	15:30	pts/1	00:00:00	sleep 10000
root	131	92	0	15:31	pts/0	00:00:00	ps -eaf

Fig. 3.2: `ps`-command in a new PID namespace

The isolation of the file system can be achieved by employing the `$ chroot` command, which allows the current root directory to be relocated to a different folder. Further details on this process are presented below.

### *chroot*

The `$ chroot` command relocates the existing file system root to an alternative folder. Subsequently, any access to folders located above this new root is no longer

<sup>1</sup> Network Information Service



available. When used for a container environment, a file system must be present in the new root directory – for instance, an Alpine Linux. This isolates both processes and the entire file system, thus eliminating the possibility of accessing process information on the host system. [24, p. 38ff.]

Testing this reveals that only the processes originating from inside the “container” are visible. Running a sleep command in a different shell does not result in the process being displayed in the `$ ps` output, as depicted in Figure 3.3.

```
/ # ps -eaf
PID   USER     TIME  COMMAND
    1   root      0:00   sh
    10   root      0:00   ps -eaf
/ # |
```

Fig. 3.3: `ps`-command inside a simple container

### *Mount Namespace*

Another isolation mechanism is the Mount Namespace. In the default configuration, all current system mounts are visible to processes. To restrict this visibility, a mount namespace can be established, similar to the PID namespace. However, the list of mounts is situated within the `/proc` directory, enabling the manual discovery of these mounts when not using `chroot`, as detailed in Rice’s work. [24, p. 42ff.]

### *Network Namespace*

To isolate the network resources, such as network devices and routing tables, the network namespace is used [26]. This isolation enables the creation of an entirely distinct networking setup, independent of the host’s networking configuration, as demonstrated in Figure 3.4.

```
flo@Flo-PC:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1280 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:84:71:98 brd ff:ff:ff:ff:ff:ff
    inet          brd          scope global eth0
        valid_lft forever preferred_lft forever
    inet6          scope link
        valid_lft forever preferred_lft forever
flo@Flo-PC:~$ sudo unshare --net ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Fig. 3.4: Example usage of `unshare --net`

### *Inter-Process Communication Namespace*

The inter-process communication (IPC) namespace is responsible for managing certain forms of inter-process communication [27]. By using this namespace, it

becomes possible to restrict container processes from utilizing inter-process communication mechanisms.

### *Cgroup Namespace*

The cgroup namespace limits the visibility of higher cgroups by configuring the cgroup-root according to the creating process's current cgroup. This measure prevents external paths from becoming visible within the container, a safeguard against potential exposure of sensitive data. [28] Additional insights into cgroups are elaborated upon in paragraph 3.1.2.

### *User Namespace*

The user namespace facilitates the mapping of identifiers, such as the root user ID (0), to an unprivileged user ID on the host system. This mapping minimizes potential harm in the event of a container breach, as the attacker does not immediately acquire root privileges beyond the container. Moreover, the creation of user namespaces can be achieved by a non-root user, enabling the sandboxing of processes and even the execution of containers without necessitating root privileges, as emphasized by Michael Kerrisk. [29]

### *Time Namespace*

The time namespace virtualizes the monotonic and booting clocks, excluding the real-time clock due to kernel complexities and overhead. This feature is added to ensure consistent values for these clocks during container migrations. [30]

To tie these concepts together to a container, container runtimes come into play.

### *Container Runtimes*

Container runtimes combine these earlier concepts to enable the smooth functioning of containers. It is worth highlighting that the term “container runtime” can be ambiguous, encompassing two different notions. On the one hand, there is the high-level runtime, orchestrating tasks like image retrieval from registries and managing container lifecycles post-deployment. On the other hand, the low-level container runtime constructs containers using an OCI-compliant bundle. Sometimes high-level runtimes are called “container engines”, as mentioned in sources like RedHat [31] and Bigelow [32]. For the sake of clarity, this work employs the same terminology.

An example of a container engine is CRI-O, which is the one used in this project.

### *CRI-O*

CRI-O<sup>2</sup> functions as a container engine, utilizing by default the low-level runtime, runc<sup>3</sup>. Nevertheless, the option to employ any OCI-compliant runtime remains available. Furthermore, it implements the Container Runtime Interface (CRI), needed to work with Kubernetes. It also follows the release cycle of Kubernetes to ensure compatibility. [33]

<sup>2</sup> <https://cri-o.io/>

<sup>3</sup> CLI tool for spawning and running containers

### 3.1.2 Security

As highlighted in the preceding section, containers inherently possess a substantial level of isolation from the Linux system. Nevertheless, there exist additional strategies to amplify and fortify this isolation, effectively reducing the potential for container vulnerabilities. The concepts outlined in this section are applied throughout this work to further strengthen the isolation of the prototype.

#### *Cgroups*

The primary method of imposing restrictions involves employing control groups (cgroups). These cgroups facilitate the grouping of processes, including containers, for effective monitoring. Additionally, these groups can be constrained by imposing limitations on the utilization of system resources, such as RAM and CPU allocation.

#### *Seccomp*

The secure computing mode (seccomp) is a tool to restrict system calls. Due to the fact that system calls are often used to escape the isolation of containers, a foundational comprehension of them is necessary, which the following paragraph seeks to establish.

#### *System Calls*

System Calls (syscalls) denote requests sent to the Kernel in order to execute actions that demand special privileges. An illustrative example of this is file operations, including file creation, reading, updating, or deletion. Additionally, system calls are used for tasks like initiating new processes. In total, the realm of system calls encompasses more than 300 functions. [24, p. 13f.]

Seccomp proves valuable for limiting the range of system calls. In its initial release, the tool provided the capability to restrict all syscalls except for read, write, and exit. Subsequent updates introduced a Berkeley Packet Filter, which can be configured to restrict the process solely to the required syscalls. [34] Furthermore, it is possible to monitor the system calls made by a process and subsequently enforce restrictions on any call that is not necessary for the program's execution [35].

#### *Capabilities*

Capabilities provide the capacity to grant certain privileges to processes that lack them. Currently, a collection of over 35 capabilities exists, extending rights that were conventionally exclusive to the root user. An example of this is the “net raw” capability, which permits the utilization of raw and packet sockets. It is essential for functionalities such as the `$ ping` command. It is crucial to note that privileged processes, specifically those executed by the root user, bypass the capabilities checks. [36]

Docker incorporates a list of default capabilities<sup>4</sup>, which are essential for running a Docker container [37]. This set of capabilities can vary between different vendors. For example, CRI-O has a more restrictive list<sup>5</sup>, omitting capabilities like “net raw”.

### *Linux Security Modules*

Linux Security Modules (LSM) were introduced to facilitate the implementation of improved access control mechanisms, without prescribing a specific implementation itself. Previous attempts to implement improved access controls encountered challenges due to a lack of consensus within the Linux community. The LSM framework offers an interface that allows the integration of alternative solutions. These implementations are integrated into the kernel through compilation. The mechanism involves embedding hooks within the kernel code, which invoke functions that must be implemented by the LSM. [38]

Mandatory Access Control extensions, designed to enforce security policies, constitute the primary users of this interface. The Linux capability system is provided by default, and while it can be substituted with an alternate solution, most LSMs prefer to extend this system rather than completely replace it. [39]

Numerous LSMs are accessible, and among the prominent choices are supported SELinux and AppArmor, which can integrate into Kubernetes.

### *AppArmor*

AppArmor<sup>6</sup>, a project supported in its ongoing development by Canonical since 2009, complements the default discretionary access control (DAC) system with a mandatory access control (MAC) system, thereby enhancing security. This mechanism involves the application of AppArmor policies (profiles) to applications, which serve to restrict their capabilities, limit network access, and impose resource constraints. [40, 41]

It operates in two modes: one enforces the applied profile and restricts resources, while the other mode merely raises complaints if the profile limits are exceeded. This latter mode serves as a tool for constructing customized profiles. It does so by initially learning the accessed resources and subsequently identifying which ones can be reasonably restricted. The Linux default discretionary access control system (DAC) is evaluated prior to AppArmor’s involvement. Applications lacking an applied profile are solely governed by the DAC system. Unlike SELinux, which permits unique profiles for individual users, AppArmor employs system-wide profiles. [40, 41]

### *SELinux*

Security Enhanced Linux<sup>7</sup> (SELinux), developed by the National Security Agency (NSA), was made open-source in 2000. It is now the default choice for RedHat

<sup>4</sup> <https://github.com/moby/moby/blob/master/oci/caps/defaults.go>

<sup>5</sup> <https://github.com/cri-o/cri-o/blob/main/docs/crio.conf.5.md>

<sup>6</sup> <https://gitlab.com/apparmor/apparmor>

<sup>7</sup> <https://github.com/SELinuxProject/selinux>

distributions. SELinux operates in two modes: enforcing mode, which actively enforces security policies, and permissive mode, which solely logs violations.

SELinux employs a security context that encompasses the user’s identity, the role, the type (which defines a domain for processes or a type for files), and optionally, the level. This context is instrumental in defining the regulations governing files or processes. To define rules for these files or processes, policies are created, which bundle multiple rules in them. The main permission control method of SELinux policies is type enforcement. Since all files are already labeled with a type, the latter can be used to define how they interact and access each other. [42]

Access needs to be given explicitly, as by default all access is denied. To avoid the manual definition of all rules, predefined sets of policies known as “Booleans” can be employed, which achieve specific tasks. As an example, the system can set whether to permit or deny FTP servers from accessing home directories. By further using role-based access control and user-based access control, the system can take more factors into consideration. [42]

Much like the SELinux assessments, AppArmor also requires the initial success of the system’s discretionary access control (DAC) check. To optimize computational resources, previously verified requests are stored within an access vector cache. In cases where permissions are not cached, the request is forwarded to the security server for examination of the security context. [43]

### *gVisor*

To reduce the risk associated with system calls, gVisor<sup>8</sup> aims to create a secondary kernel operating within the user space. In this configuration, instead of container system calls interfacing directly with the actual host kernel, they are redirected to the gVisor kernel, thereby enhancing security. [44]

The go-written system has two main components. The Sentry, serving as an equivalent to the kernel, intercepts, and processes system calls. Functioning as a user space application, the Sentry possesses constrained access to designated system functions. While it can directly interact with the kernel for proper operation, it does not transmit system calls from the container directly to the kernel. These Operations, that are beyond the capabilities of the Sentry, are delegated then to the second main component called Gofer, which serves as an intermediary for accessing file system resources and managing system calls. [44]

## 3.2 Kubernetes

This section aims to explain relevant components and fundamental concepts of Kubernetes that are important to this work. Subsequently, the security mechanisms inherent to Kubernetes are outlined. Lastly, the concept of Custom Resources and Operators is detailed.

NAME	READY	STATUS	RESTARTS	AGE
pod/coredns-787d4945fb-frn4d	1/1	Running	11	24d
pod/coredns-787d4945fb-wnq6h	1/1	Running	11	24d
pod/etcd-kube-master	1/1	Running	17	24d
pod/kube-apiserver-kube-master	1/1	Running	4	15d
pod/kube-controller-manager-kube-master	1/1	Running	4	15d
pod/kube-proxy-5w6jq	1/1	Running	4	15d
pod/kube-proxy-7xgdp	1/1	Running	4	15d
pod/kube-proxy-d95zq	1/1	Running	4	15d
pod/kube-scheduler-kube-master	1/1	Running	4	15d

Fig. 3.5: All containerized components of Kubernetes

### 3.2.1 Components

Kubernetes is structured into various components, which will be explained subsequently. These components can be installed directly on the host machine or virtualized within the Kubernetes cluster environment. In this project, the tool `kubeadm` is used to install the cluster.

#### *etcd*

`etcd` serves as a high-performance distributed key-value store, containing all Kubernetes cluster-related information, such as the servers of the cluster and the containers that are currently managed [45]. It is possible to either establish a single `etcd` instance or, to ensure redundancy, configure multiple `etcd` nodes.

`Etcd` handles writes from multiple instances by electing a leader, which is the only instance permitted to fulfill the write-request. Subsequently, the modification is propagated to all worker instances, which must acknowledge it. In cases where a request reaches a worker instance first, the request is redirected to the existing leader. [46]

#### *kube-apiserver*

At the heart of the Kubernetes cluster lies the `kube-apiserver`, functioning as the central command center. This component hosts the API that facilitates communication between all other elements. For instance, when a request is sent to create a Pod via the Kubernetes API, the `kube-apiserver` undertakes user authentication, request validation, and processing before making the resource available in the API, where the `kube-scheduler` can assign it to a Node. Subsequently, it updates the `etcd` store accordingly. [47]

#### *kube-scheduler*

The `kube-scheduler` efficiently arranges Pods onto an available Node by actively monitoring newly introduced Pods that await Node assignment. Employing a two-step approach, it begins by filtering out unsuitable Nodes, followed by assessing

<sup>8</sup> <https://github.com/google/gvisor>

the remaining options through a scoring mechanism to identify the most optimal one. Notably, both of these decision-making processes can be customized to incorporate individualized criteria, offering flexibility and adaptability to the scheduling process [48]. Furthermore, the scheduler can take Taints and Affinities 3.2.2 into account. It does not create Pods on Nodes, delegating this responsibility to the kubelet.

### *kubelet*

Kubelet, an essential cluster component, is directly installed on every Node within the infrastructure. Its primary role involves overseeing the containers present on the Node, establishing direct communication with the container engine. To achieve this, Kubelet relies on the utilization of PodSpec definition files. These files describe the Pods that should be running. Typically, these descriptive files are sourced from the kube-apiserver. [49] To oversee the proper functioning of the newly deployed Pods and the Node where kubelet resides, a Controller is needed.

### *kube-controller-manager*

The kube-controller-manager assumes the role of overseeing fundamental Controllers within Kubernetes, bundling them to reduce complexity [50]. One noteworthy example is the Node-Controller, which undertakes health evaluations across all Nodes. Should a Node encounter prolonged failures, this Controller acts by directing the eviction of existing workloads from the troubled Node. Subsequently, these workloads are rescheduled onto alternative Nodes, ensuring the continuity of operations. [51]

### *CoreDNS*

CoreDNS is the bundle's DNS-Server for the current Version (v1.27) of kubeadm and handles all DNS entries of the cluster. While not inherently a core component of Kubernetes, it is included in this context because it is, at present, the only option kubeadm supports. [52]

### *kube-proxy*

The kube-proxy runs on every Node and serves the function of forwarding traffic to and from a Service 3.2.2. [53]

### *kubectrl*

Kubectrl, the Kubernetes command line tool, allows users to administrate the cluster directly through command line interactions. Its functionality extends to configuring multiple clusters through a kubeconfig file, similar to an ssh-config file. This file defines essential information, such as the host and login tokens. [54]

The designated command for utilizing this tool is `$ kubectrl`, yet for the sake of brevity, the shorthand `$ k` will frequently be employed in this work. This is derived from the kubectrl cheat sheet [55], which advises creating an alias for quicker command entry.

### 3.2.2 Concepts

This section reviews key concepts of Kubernetes, encompassing elements employed within this work as well as those that hold significance for future expansions. The subsequent design chapter examines the practical application of these concepts.

#### *Resources and Manifests*

Resources encompass all Kubernetes API objects. When expressed in a YAML or JSON file, they take on the name “Manifests”. These Manifests can subsequently be transmitted to the API to create the described Resources. The responsible entities for housing these Resources are referred to as Nodes.

#### *Nodes*

Servers integrated into the cluster are designated as Nodes. The initial Node is referred to as the Control-Plane Node, and it typically abstains from hosting regular container loads, as it accommodates all essential Kubernetes components. The Control-Plane Node is primarily responsible for housing Kubernetes components, whether they are containerized or directly installed. For cluster expansion, additional Nodes can be introduced through self-registration or manual incorporation, achieved by generating a Node object on the Control-Plane. [56] To isolate users without creating multiple clusters, Namespaces can be used.

#### *Namespaces*

Namespaces serve as a mechanism for establishing isolation within the API, offering the capability to enforce access restrictions on all enclosed resources. Furthermore, aided by the Networking add-on 5.2 implemented within this project, it becomes feasible to establish policies that restrict the flow of traffic between different Namespaces, intensifying their isolation. In the context of this project, the utilization of Namespaces plays a pivotal role in ensuring the distinct isolation of each individual user from one another. [57]

#### *Pods*

Pods constitute the smallest unit within Kubernetes. They are assembled from a minimum of one to multiple containers, collectively sharing the same localhost environment and storage setup. They are sometimes known as sidecar-containers. [58] In this prototype, the Pods host the container that students can connect to.

#### *Labels and Selectors*

Labels serve as markers to identify specific Resource within Kubernetes. An illustration of their significance can be observed in the context of ReplicaSets 3.2.2. In that context, Labels function as discerning criteria, playing a crucial role in determining the eligibility of existing Pods to participate in replications. This evaluation involves comparing Labels on running Pods with those given in the Manifest.



Similar practices extend to Nodes too. Nodes can be designated for scheduling particular Pods by applying Labels. Furthermore, the introduction of a more complex logic can be achieved using Node Taints and Affinities, as elaborated upon in a later section 3.2.2. [59]

This work will include Labels to simplify the identification and accessibility of resources via the Kubernetes API.

### *ReplicaSet and ReplicationController*

Both of these mechanisms are engineered to manage the creation or termination of Pods in alignment with the specified replica counts. Additionally, they possess the ability to restart or rebuild Pods that encounter failures. They are not bound to a single Node and have the flexibility to extend across multiple Nodes. To accomplish this, ReplicaSets and ReplicationControllers write themselves in the owner's reference field in the metadata section and subsequently count the Pods which possess the matching metadata.

Besides, they also encompass the capability to oversee and manage pre-existing Pods. To facilitate this, the selector property exists. When Pods align with the Labels described in the selector, they are automatically incorporated into the ReplicaSet/Controller and are counted towards the replication limit.

The main difference between Sets and Controllers is that ReplicationControllers are not mandated to define a selector, while it is a prerequisite for ReplicaSets. Moreover, ReplicaSets utilize set-based selectors. Considering the prominence of these selectors as the modern practice, ReplicaSets emerge as the favored option. [60] Set-based selectors provide a broader spectrum of capabilities beyond basic equality-based operators such as `=` or `!=`. They extend their functionality to include advanced operations like “in,” “not in,” and “exists,” thereby improving label management. [59]

ReplicaSets offer the basis for Deployments, which are used in this work to provide Pods for students.

### *Deployments*

Deployments are a superset of ReplicaSets. They serve as an overarching framework that orchestrates ReplicaSets while introducing enhanced flexibility and encompassing functionalities like rolling updates, rollbacks, and application versioning. To conduct image version updates in a ReplicaSet without downtime, traditional methods involve manual creation of a new ReplicaSet with the updated image, followed by the deletion of the old one. Deployments automate this task by using the defined Rollout strategy. By default, they create a second ReplicaSet, which is scaled up one by one and the old ReplicaSet is scaled down one by one until every Pod runs on the new image(s). [61]

To address storage needs, the utilization of Persistent Volume claims facilitates the automated acquisition of the storage that is essential for the prototype.

### *Persistent Volume Claims, Persistent Volumes and StorageClasses*

A Persistent Volume represents a designated portion of allocated storage, which can be obtained through manual allocation or can be automatically provisioned using a Persistent Volume Claim (PVC) alongside a defined StorageClass. [62]

StorageClasses provide means to characterize the type of storage being provided. For instance, a StorageClass named “slow” might be associated with a provider that draws storage from an older storage array. Each StorageClass is associated with a specific provider responsible for fulfilling storage requests on demand. [63] Additional information about the process employed in this project will be provided in Section 5.2.

The Persistent Volume Claim allocates storage for a user, utilizing a designated StorageClass. For instance, a Pod can make a request to the PVC to obtain the allocated storage and mount it as needed.

### *Limits and Requests*

To allocate additional resources for each Pod, such as RAM and CPU power, it is possible to define Requests. These Requests establish the minimum resources required for the Pod to function. If these requirements are not met, the Pod will remain unscheduled until the necessary resources become available. [64]

In scenarios where a container requests two CPU Units, equivalent to two CPU cores, and there are four cores available, the container could potentially utilize all four cores. To address this, Limits can be set, representing the maximum resources permissible for the Pod’s utilization. This prevents the container from surpassing the defined upper threshold. [64]

In order to enforce these Limits, Kubernetes uses cgroups 3.1.2, a mechanism that restricts a Pod from surpassing its hard resource limit. In the case of the CPU, this may result in a slowdown of the application. However, when the memory limit is reached, the system might terminate the process, subsequently destroying the entire Pod. In such instances, Kubernetes will automatically restart the container, provided it is marked as restartable. [64]

### *Liveness-Probe*

Even in cases where a Pod does not experience a complete crash, it might still encounter internal malfunctions. To identify such issues, Liveness Probes come into play. These health checks, conducted by Kubernetes, assess the real-time condition of the container. In the event of a failed health check, the container is destroyed. For instance, a Liveness Probe could be configured to verify the reachability of a website hosted within the container. If this probe fails, indicating a potential web server crash, Kubernetes automatically initiates a container recreation to remedy this issue. [65]

### *ConfigMaps and Secrets*

To provide configurations to Pods or Deployments, ConfigMaps and Secrets serve as key-value stores. They facilitate the injection of data into containers as environment variables. While both can be employed similarly, a notable distinction lies

in the base64 obfuscation. This means that ConfigMaps can be directly read from the cluster in plain text, whereas Secrets encode their values in base64, rendering them less directly readable by humans. [66]

### *Services*

Services can be used to enable accessibility to Pods, serving to expose one or more Pods, either within or outside the cluster. [67] For example, in a scenario involving a database comprising two replicas, with the corresponding Pods successfully scheduled and operational, there exists a web application that needs to establish a connection with said database. This connectivity can be achieved by utilizing either the IP address or the default DNS entry, which comprises the Pod's IP by default, to connect to the database.

This approach introduces a potential issue: when a Pod is destroyed and reconstructed, its IP address may change. To address this concern, the utilization of a Service becomes essential. Kubernetes offers various Service types, yet in general, a Service load-balances traffic to a Pod or a group of Pods based on their Labels 3.2.2.

Upon the creation of a Service, the kube-proxy 3.2.1 is responsible for generating and consistently updating rules. These rules dynamically adapt to the addition of new Pods or the removal of existing ones. [53] With this setup, the web-Pod can conveniently use the Service's DNS entry to connect to the database. As a result, traffic is automatically directed to an accessible database-Pod. An example of a Service utilized within the project is the NodePort.

### *NodePort*

NodePort allows a Pod to be accessed from outside the cluster. By default, it allocates a port within the Range of 30000 to 32767. This port allocation is employed to load balance incoming traffic across the Pods within the associated Service. This arrangement enables the accessibility of user Pods from external sources. [67] However, the arbitrary port assignment may not be optimal for all resources. For instance, websites are usually accessed through URLs instead of IP and port combinations. To address this, the concept of Ingress is introduced.

### *Ingress*

In Kubernetes, Ingress serves as a resource type that facilitates external HTTP/S access to the cluster. By associating a URL with an internal Service, an Ingress resource renders the Service accessible. In scenarios necessitating different types of traffic, alternatives such as load balancers or NodePorts can also be considered. [68] To enable this functionality, the installation of an Ingress Controller is a requisite. The Ingress Controller oversees the management of the generated resources. Contour is a notable example of an Ingress controller, that is employed for the prototype.

### *Contour*

Contour<sup>9</sup> stands as a high-performance Ingress controller built on the Envoy<sup>10</sup> platform. This Ingress Controller operates within the OSI layer 7, making decisions based on elements of the message, such as HTTP metadata [69]. Notably, Contour is hosted by the CNCF. To ensure secure connections, the necessity of valid certificates arises. The certificate management process can be automated using tools like Cert-Manager.

### *Cert-Manager*

Cert-Manager<sup>11</sup> is another CNCF incubation project focused on the management of X.509 certificates. It provides capabilities for obtaining certificates from various issuers, including Let's Encrypt. Additionally, it features the ability to generate self-signed certificates. To store the resulting certificates, Cert-Manager employs ConfigMaps and Secrets. [70]

### *Taints, Tolerations, and Affinities*

To precisely assign workloads to specific Nodes, Kubernetes employs mechanisms such as Taints, Tolerations, and Affinities. For instance, the Control-Plane Node has an inherent Taint that prevents Pods from being scheduled on it by default. This measure simplifies the upgrade process 5.1 and ensures continuous availability of Pods, even when a configuration change disrupts the control-plane. [71]

Tolerations can be applied to Pods to bypass associated Taints and allow scheduling on designated Nodes. This functionality proves useful when adding specific critical Pods with the Control-plane. [71]

Lastly, Node Affinity serves to establish a preference for scheduling on a particular Node. This can be employed to ensure, for instance, that Pods are primarily scheduled on a server with superior external connectivity. Only when the designated server reaches full capacity would Pods be routed to the comparatively slower machines. Notably, Node Affinity Labels closely resemble Node Selectors, but they distinguish themselves through the ability to accommodate more complex configurations, such as weighted placements or the selection of values from a predefined set, rather than merely seeking specific Labels. [71]

## **3.2.3 Security**

This section defines essential security-related concepts within Kubernetes, which are required for a comprehensive understanding of this work. These concepts will be further revisited and put into context across the design and discussion chapters.

---

<sup>9</sup> <https://projectcontour.io/>

<sup>10</sup> <https://www.envoyproxy.io/>

<sup>11</sup> <https://cert-manager.io/>

### *Security Context*

The Security Context serves as a mechanism to configure privileges and access control parameters within Kubernetes Pods and containers. This feature encompasses a spectrum of customizable choices, spanning from establishing the user ID (UID) under which the container operates to configuring Seccomp profiles. A few of these options are explained below. [72]

First, to configure the Security Context, the “securityContext” field must be defined within the Spec section of the Pod. This step ensures the establishment of desired security parameters and controls.

As an example, it is feasible to impose the requirement of running the image as a non-root user. Failure to satisfy this condition while configuring the image leads to the container failing to start. Furthermore, the container’s capabilities can be extended by utilizing the “capabilities” field. This avenue offers the flexibility to either include or exclude capabilities within the container’s operational framework. Likewise, SELinux fields can be added to enable functionality. [72]

Seccomp and AppArmor work differently, wherein a predefined profile must be crafted and then applied to the container. Particularly noteworthy is that Seccomp provides the capability to designate a default profile for all workloads. [73, 74]

### *Pod Security Admission*

Pod Security Admission ensures the enforcement of isolation standards for Pods and replaces the deprecated Pod Security Policies system in Kubernetes 1.25. This streamlined approach combines a range of Security Context options along with settings that extend beyond the bounds of the Security Context itself. Pod Security constraints are applied within the context of individual Namespaces. [75]

The Pod Security Standards encompass three distinct levels: Privileged, Baseline, and Restricted. The Privileged standard lacks restrictions, potentially facilitating privilege escalations<sup>12</sup>. The Baseline standard enforces specific limitations to counteract well-known privileged escalations. Lastly, the Restricted standard applies all recommended hardening measures, although with the potential to introduce compatibility concerns. [76]

### *gVisor*

gVisor provides its own Container Runtime, called runsc, integrated through the Kubernetes Runtime-Class object. This approach permits the customization of runtimes for specific workloads. For instance, typical workloads employ the standard runc, while critical high-risk containers benefit from the enhanced isolation of gVisor’s runsc, thereby elevating container security. [77] For the Runtime-Class feature to operate correctly, the container engine must possess the ability to accommodate diverse runtimes. The container engine highlighted earlier, CRI-O, does support this behavior. [78]

---

<sup>12</sup> A type of cyberattack, where a bad actor escalates their rights to a higher level than intended

### *Authentication and Authorization*

Authentication for the Kubernetes cluster can be done in multiple ways. Broadly, these mechanisms fall into two distinct account categories: service accounts and regular user accounts.

Service accounts are meant for non-human processes and are stored within the Kubernetes API. For example, the logging service account is used by a logging application to access the Kubernetes API from within a Pod.

These accounts are accessed using tokens that are stored within the Kubernetes-API. However, in more recent versions of Kubernetes, these tokens need to be specifically requested through the TokenRequestAPI, and they are no longer generated by default. This change is a result of the Kubernetes Enhancement Proposal 2799 [79] and 1205 [80]. This alteration serves the purpose of enhancing security by refraining from generating unnecessary tokens. Instead, tokens now have a designated audience, a defined time limit, and are bound to specific objects. Lastly, a service account named “default” is automatically established for each Namespace. This service account is assigned to every Pod initiated within the Namespace, unless specifically configured otherwise. This association permits the Pod to establish communication with the Kubernetes cluster. [81]

In addition, regular users are not stored within the Kubernetes-API; instead, their authentication is based on possessing a valid certificate signed by the Kubernetes CA<sup>13</sup>. Once a user is successfully authenticated, the authorization process can proceed. [82]

Kubernetes has multiple authorization methods, the one that is being used here is called role-based access control (RBAC). This mechanism facilitates access to distinct Kubernetes objects through the utilization of Roles. These Roles can be either defined for a singular Namespace or for the whole cluster. For example, it is possible to grant privileges to exclusively list all Pods within a particular Namespace or solely access Pods with specific names. However, as of now, it is not possible to confer privileges to search for Pods based on specific Labels. [83]

As previously discussed, Kubernetes lacks an inherent user concept. To introduce this functionality, Single Sign-On solutions like Keycloak can be employed.

### *Keycloak*

Keycloak<sup>14</sup> provides Single-Sign On (SSO) functionality and serves as an OpenID-Connect provider. Initially developed and managed by RedHat starting in 2014, the project was later handed over to the CNCF in April 2023, with ongoing sponsorship from RedHat. [84] Due to its maturity, Keycloak has been selected as the designated authentication provider for this project. Further considerations are made in the design chapter. Before proceeding, it is relevant to understand the differences between authentication mechanisms.

<sup>13</sup> Certificate Authority, an entity that signs certificates

<sup>14</sup> <https://github.com/keycloak/keycloak>

### *SSO, OIDC and OAuth*

The acronyms OIDC and OAuth denote closely related concepts. Therefore, this passage aims to provide clarity on their meanings. The initial acronym, SSO, represents Single Sign On, which enables access to multiple applications using just one set of credentials. [85]

Moreover, OAuth, currently in its second version, assumes the role of an authorization framework, conferring precise permissions within a system. The OAuth provider enables users to access a predefined set of functions on the server by utilizing a token. [86]

Lastly, before the emergence of OpenID-Connect (OIDC), there existed OpenID, a system designed for identity verification during authentication processes. For instance, when a user aims to connect to a server that lacks prior knowledge of the user but places trust in an OpenID provider, the user can authenticate with said OpenID provider. Subsequently, the OpenID provider informs the server that the user is trustworthy and possesses the authorization to access the system. [86]

OIDC merges the dual capabilities of OpenID's authentication and OAuth2's authorization. This combination allows users to not only authenticate themselves, but also gain authorization for specific actions simultaneously. [87]

### *Network Policies*

Network Policies control the traffic flow at OSI layer 3 and 4, making it possible to allow or restrict any IP and Port combination. While Network Policies are a Kubernetes object, their execution is delegated to the utilized Network add-on. It is worth noting that the implementation of Network Policies is not mandatory, and certain add-ons might disregard them. [88]

Calico, the selected network add-on for this project, supports Network Policies.

### *Calico*

Project Calico<sup>15</sup> is an Open-Source Network add-on created by Tigera, which fully supports all restrictions set by Network Policies and assumes the role of the Network add-on for the prototype. [89]

#### **3.2.4 Custom Resources**

Custom resources enable the expansion of the Kubernetes API through the introduction of custom objects. This work will heavily rely on them in combination with an Operator to manage the Kubernetes resources mentioned before.

There are two distinct approaches to create a Custom Resource: Custom Resource Definitions (CRDs) and Aggregated API. Aggregated APIs are established by deploying a custom backend within the cluster. Subsequently, a URL is registered with the Kubernetes API, routing all requests to this backend. [90]

---

<sup>15</sup> <https://www.tigera.io/project-calico/>

CRDs, on the other hand, offer the capability to store customized data within the etcd cluster, granting direct accessibility through the Kubernetes API. Consequently, all functionalities, including authentication, are automatically extended to encompass Custom Resources as well. However, certain limitations exist when compared to the Aggregated API. For instance, there are format restrictions imposed on URLs, as they must align with the URLs of built-in resources. Additionally, size constraints are in place to prevent overloading the API. [90]

#### *Custom Resource Definition*

CRDs encompass three essential components: Spec, Status, and Metadata. The Spec section is employed to articulate the intended state, as defined by the user. The Status field is used to write down how the Controller views the object, for example, using a condition field in Status to indicate “Creating” while initializing the Pods. Lastly, the Metadata section provides additional information about the resource, for example the name of the Controller who created and manages it. [91]

### **3.2.5 Operators**

#### *Controllers and Operators*

In Kubernetes, Controllers have a pivotal role in system management, facilitating the transition from the current state to the desired state. For example, the creation of a Deployment illustrates this concept. Controllers continuously monitor for new objects and, upon detecting a new Deployment, assess the presence of required resources. If these resources are lacking, Controllers initiate the creation and configuration of the necessary ReplicaSet to align the system with the desired state. [92]

While Controllers are employed to handle built-in resources within Kubernetes, an Operator fulfills a similar role for Custom Resources. [93] Although Operators can be written in a multitude of languages, the Operator white paper [94] states that Go is widely favored as the predominant Operator language. This preference stems from Go’s adaptability, its orientation towards network operations, and its alignment with Kubernetes, which is also developed in Go.

#### *Go*

Go<sup>16</sup>, often referred to as golang due to its former domain [95], is a language developed by Google. It finds its primary application in Network Programming, owing to its robust concurrency mechanisms.

The syntax of Go has some resemblance to C, but does not have semicolons, similar to Python. Go is a compiled language with static typing. Notably, in Go, public variables and functions are designated in PascalCase, while private ones are named in camelCase. It is important to emphasize that all identifiers in Go are package-visible, unlike Java, where identifiers are by default protected. Go identifiers are public when written in PascalCase. [96]

<sup>16</sup> <https://go.dev/>



Another aspect to consider is the definition of methods and functions, illustrated in Listing 3.1. In the provided example, a receiver argument is placed between the `func` keyword and the function's name. This establishes the function as a method associated with a struct. Notably, the `*STRUCT` notation signifies a pointer receiver, indicating its ability to modify the struct's variables. If the star is omitted, it is referred to as a value receiver, which works with a copy of the struct. Following the name, the arguments are specified, all of which are of the same type. In this instance, the declaration of `arg1` can be shortened, since it has the same type as `arg2`. Lastly, all return values are written. [96]

```
1 func (struct *STRUCT) Reconcile(arg, arg2 ARGTYPE) (RETURNTYPE1, RETURNTYPE2)
```

Listing 3.1: Go function definition

Having covered these fundamental aspects of Go, the Operator's functions can be further discussed.

### *Reconciliation*

The primary objective of any Controller in Kubernetes is to ensure alignment between the user-declared state of an object and the actual state within the cluster. For example, every ReplicaSet needs to create as many Pods as stated by the user. To achieve this, the Controller ensures the creation of the necessary Pods. This process is referred to as reconciliation. [97]

### *Reconcile Function*

The Reconcile function serves as the core of a Kubernetes Controller. This function is triggered whenever a monitored object changes its state. This is accomplished by specifying in which way the Controller or Operator should pay attention to objects: "For", "Owns", and "Watches". The "For" directive dictates that the Operator should initiate the Reconciliation process whenever any of these objects are created, updated, or deleted. In contrast, "Owns" only triggers this action if the OwnersReference is configured to match the current Operator. This mechanism enables the management of common types utilized by multiple Operators. Lastly, the "Watches" directive serves as the foundation for both "For" and "Owns", providing enhanced flexibility in triggering the Reconciliation process. It has the capability to register any event from a specified source to initiate the Reconciliation-Loop. [98]

The Reconcile function must respond to changes and accordingly modify cluster resources. It is relevant to emphasize that this function must be idempotent. [97] This means that the function cannot have additional effects when called multiple times. For instance, the previously mentioned ReplicaSet is encountered twice by the Reconcile function: first during its creation, leading to Pod generation, and then again when a metadata field is modified. In this scenario, the function should abstain from spawning more Pods since the required count is already met. Introducing additional Pods would conflict with the user's intended state. [99]

An additional aspect to keep in mind is that Kubernetes APIs follow a level-based approach. In this system, the API reads the existing status of the cluster and contrasts it with the present specification. When discrepancies arise, reconciliation is set into motion. However, this approach brings forth several considerations. The Reconcile process can overlook outdated values declared in a previous version of the specification that had not yet triggered reconciliation, and multiple events can be batched together. [100]

Operators must account for these behaviors by consistently examining all aspects of the object. This means that, if an error is detected in any part of the object, it can be rectified directly, even without waiting for reconciliation to be triggered specifically for that segment of the object. Kubernetes calls the process where the system proactively addresses detected issues “self-healing”.

### *Finalizers*

Finalizers are a mechanism employed by Operators to facilitate a graceful deletion process. They wait until certain conditions are satisfied before the resource is completely removed. As an example, they might trigger actions such as deleting residual data from a Pod by interacting with a database. Typically, for internal Kubernetes objects, the OwnersReference is utilized. This mechanism ensures the deletion of all objects associated with the resource that is currently undergoing deletion. For example, a Deployment deletes all the ReplicaSets, which in turn delete all the Pods, hence, in the end, no Resource belonging to the Deployment is left over. Finalizers, on the other hand, are invoked when a deletion request reaches the Kubernetes API. They trigger a method specified in the Operator, which can be used to manage other resources not covered by the OwnersReference. Nonetheless, a notable drawback is that if the Operator encounters an unexpected crash, all resources in the process of deletion become stuck until the Operator regains availability. [101]

### *Kubebuilder*

To accelerate the development of an Operator, several frameworks are available. Among the prominent options is Kubebuilder<sup>17</sup>, which extends support for both Operator and Custom Resource Definition creation. Upon generating a new project, it automatically produces the essential boilerplate code, allowing the focus to be set on writing the Operator. [102]

## 3.3 Frameworks and Tools

This section explains the tools employed by the prototype, which do not find a direct fit within the preceding sections. It commences with an exploration of the chosen framework for the web application.

---

<sup>17</sup> <https://github.com/kubernetes-sigs/kubebuilder>

### 3.3.1 SvelteKit

SvelteKit<sup>18</sup> is a full-stack web framework, that builds upon the capabilities of Svelte<sup>19</sup> by introducing server-side components. These components facilitate routing and server-side rendering [103]. The server and client-side components coexist within the same codebase. Svelte itself operates as a reactive frontend framework, without the necessity for a virtual DOM. [104] Components, which are elements on the website, such as the Login-system, are clearly organized into Script, HTML, and Style sections. During the development process, SvelteKit builds on Vite, which provides several functionalities, including the capacity to build the application for deployment and enable Hot Module Replacement (HMR) in the development environment. HMR permits real-time application changes without the need to restart the server or reload the page [105].

SvelteKit offers a multitude of functions, among which two hold particular significance for this project: Environment Variables and Server-Side Rendering.

#### *Environment Variables*

SvelteKit introduces a distinction between private and public, static and dynamic environment variables. Private variables are exclusively accessible for import within server-side JavaScript files, thereby enabling the utilization of secrets without exposing them to users. In contrast, public variables can be employed on both the server and client sides. When a variable is labeled as static, it implies that it is statically replaced during compile time, preventing reassigning the variable. Conversely, dynamic variables can be replaced even after the build process is finalized, which is often needed for containerized applications. [106]

#### *Server-Side Rendering*

As previously noted, SvelteKit supports server-side rendering (SSR), enabling the application to retrieve data from the server-side and perform modifications before transmitting it to the client. For instance, data can be fetched from an API using sensitive credentials that should not be exposed to the client. Files intended for execution exclusively on the server side are designated by adding the “server” keyword to the file, before the file extension. [107]

### 3.3.2 Management Tools

Within the scope of this work, management tools refer to software designed to facilitate software configuration and deployment. A brief list of management tools is provided and a comparison between them is conducted in the implementation chapter of this work.

<sup>18</sup> <https://kit.svelte.dev/>

<sup>19</sup> JavaScript Framework <https://svelte.dev/>

### *Helm*

Helm<sup>20</sup> stands out as a well-known deployment tool, often referred to as a package manager in certain contexts [108]. Its widespread adoption is primarily attributed to its ability to deploy Kubernetes applications without necessitating the creation of multiple Manifest files. This open-source project is a graduate project within the CNCF [108], signifying stability. The source code for Helm is publicly available on GitHub<sup>21</sup>.

### *Puppet*

Puppet<sup>22</sup> is an open-source infrastructure and automation tool that prioritizes the maintenance of the desired system state rather than direct automation tasks. According to their blog, Puppet is most effective for intricate, long-term deployments within sizable organizations [109].

### *Ansible*

Ansible<sup>23</sup> is an agent-less, open-source configuration management tool that relies solely on the presence of Python on the remote server for execution. It is currently backed and developed by RedHat. Ansible employs modules, which are essentially libraries, to extend its functionality. Configuration files are written in YAML and executed via the command line. The files containing the designated tasks are termed Playbooks. By default, Ansible employs OpenSSH for establishing connections to remote systems. [110]

---

<sup>20</sup> <https://helm.sh/>

<sup>21</sup> <https://github.com/helm/helm>

<sup>22</sup> <https://www.puppet.com/>

<sup>23</sup> <https://www.ansible.com/>

## Project Design

This chapter defines the prototype's requirements and its design.

### 4.1 Requirements

The following list encompasses all functionalities intended to be tested through the creation of the prototype. The list is divided into “Should” criteria, denoting functionalities that must be tested, and “Can” criteria, signifying advantageous features that are not obligatory but beneficial to evaluate.

#### 4.1.1 Should

- S1. Student and Lecturer accounts  
Enabling students to connect to their individual containers requires the implementation of user accounts. Additionally, the capability to establish various permission levels, including roles such as lecturer, is crucial.
- S2. Private student shares  
Students should have a private folder to permanently store their files.
- S3. Common class shares  
A shared class space is essential, allowing lecturers to upload files that students can access. An illustrative use case is the distribution of weekly assignments.
- S4. Access to the containers for Students and Lecturers  
It should be possible for students to connect to their containers using SSH. Moreover, lecturers should have the capacity to access the container with heightened privileges in order to both evaluate and provide assistance to their students.
- S5. Web-Access restriction  
Lecturers should have the capability to control and limit web access within classrooms. This functionality is particularly useful, for instance, during closed-book exams.
- S6. Common Servers  
The system should support the deployment of a server, such as MySQL, that is accessible to all students within the class.

### 4.1.2 Can

- C1. SSO Integration  
Integrating a Single Sign-On (SSO) system can enhance the system's suitability for implementation in educational institutions, since every student already has an account.
- C2. Passwordless connections  
Incorporating a feature that enables direct container access without requiring a password would be advantageous.
- C3. GUI applications  
As development is not only done per console, having the ability to showcase GUI programs would be beneficial.
- C4. GPU scheduling  
Certain workloads, particularly those related to game development or artificial intelligence, demand GPU resources. Therefore, incorporating GPU scheduling capabilities would be a valuable addition.
- C5. Automatic Shutdown  
Containers that remain inactive for an extended duration could have an automated shutdown mechanism to optimize resource utilization.
- C6. Quick Class and Student creation  
Introducing a tool that facilitates the quick deployment of classes and student resources could simplify the processes involved.

The basic idea of the prototype is depicted in Figure 4.1. A detailed explanation of the specific processes is covered in the following sections, starting with an examination of the cluster setup itself.

### 4.1.3 Workflows

The proposed application should have two workflows from the user's perspective:

#### *Student-Workflow*

To access the web app, students are required to log in initially. The authentication procedure is managed by a Keycloak instance. Once the token is acquired from Keycloak, it assumes the role of authorizing specific Kubernetes requests. This includes actions such as listing all Deployments accessible to the student. For every class, the option to start and stop their container, as well as retrieve a connection string to put into their terminal, should appear. Once connected to the terminal, they can access the machine like any normal Linux host. If a student chooses to terminate the session and start a new class container, the old container should be destroyed to save system resources. Lastly, for a passwordless authentication (requirement C2), an upload button for an SSH public key should be made available.

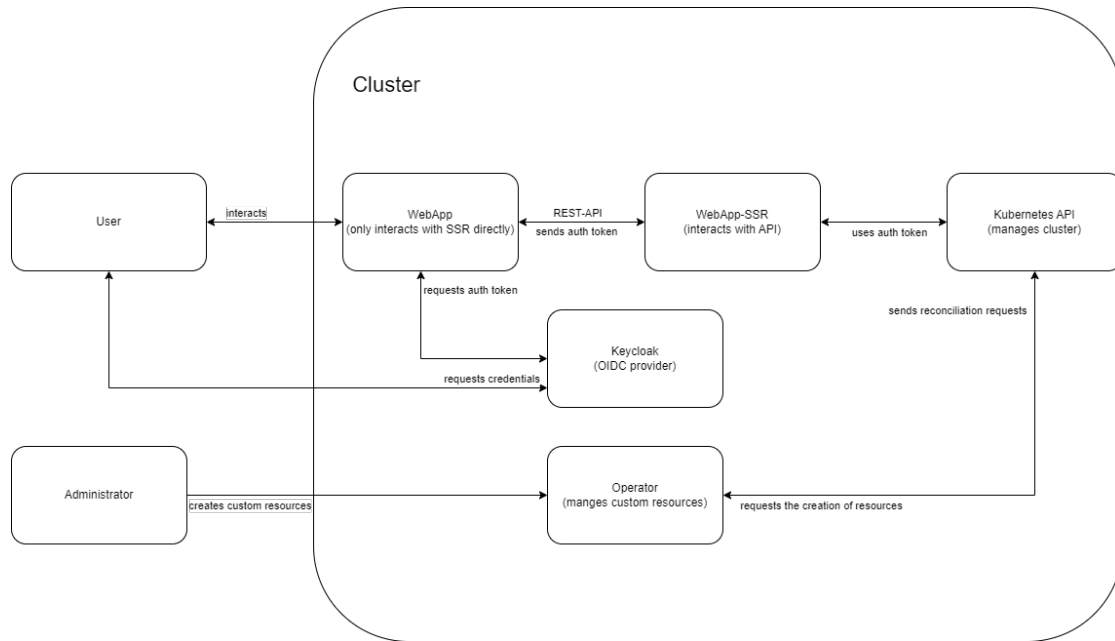


Fig. 4.1: Diagram of the Prototype

*Lecturer*

Lecturers should access the web interface through the same methods as students. Subsequently, they should have the ability to view all their classes. Within each class, they can access any students' container, start or stop it, as well as, log into it. Additionally, they possess the capability to upload files to the shared class storage, which can be accessed by all students. When opting to connect to a student's container, lecturers should be granted elevated privileges.

**4.2 Cluster**

This part proposes a cluster design and its setup choices.

**4.2.1 Hardware**

Four virtual machines with Ubuntu 22.04 are used for the Cluster. One is the Kubernetes control-plane, which stores most components 3.2.1 and is not used to schedule other Pods. The machine has 4 Cores and 8 Gigabyte of RAM allocated. Two of the remaining machines are workers that are used to schedule Pods not needed for the core system. Each machine has 4 Gigabytes of RAM as well as 2 Cores. All cores are from a Ryzen 3800XT at 3.9 GHz. Finally, one additional machine with 1 Core and 1 gigabyte of memory is used as an NFS-Server to permit permanent file storage.

### 4.2.2 Design

Kubernetes can be installed using multiple options, including Minikube, K3s, MicroK8s, and others.

For this project, the kubeadm tool is selected due to its capability to bootstrap a production-ready cluster while retaining full functionality. Consequently, on the three machines intended for the Kubernetes cluster, only the kubeadm tool and kubelet 3.2.1 are installed directly. Components like etcd and the kube-apiserver are deployed as containers within the Kubernetes cluster. This approach facilitates smoother management and debugging in the event of a component failure. However, before commencing the Kubernetes installation, it is essential to verify the presence of a container engine on each system.

#### *Container Engine*

Initially, the Docker engine was chosen due to its widespread usage. However, Kubernetes employs the Container Runtime Interface (CRI) standard to establish interactions with the container engine, which Docker does not directly implement [111]. In order to circumvent the necessity of using an adapter, CRI-O is selected. This open-source solution is lightweight, adheres to OCI standards, and aligns itself with the Kubernetes release cycle.

#### *Networking*

Given that Kubernetes does not include an inherent networking solution, an external one must be installed. This solution should align with the Cluster Network Interface (CNI) standard to ensure compatibility with Kubernetes. To fulfill these criteria, Calico is selected. This choice is influenced by its capability to provide extended security features, like Network Policies, that can help lock down the system further at a later stage. Moreover, Calico has a thorough documentation, robust support, and prominent status as a major networking project within the Kubernetes ecosystem [112].

Once the plan for the basic cluster is established, additional add-ons or software solutions are introduced. These serve to meet the requirements or address-specific needs of the prototype.

#### *Authentication*

The first add-on to be installed is an authentication provider. As previously noted in section 3.2.3, Kubernetes lacks inherent user account management capabilities. To address this, an OpenID-Connect provider called Keycloak is selected, aligning with requirement S1. Additionally, the chosen solution fulfills requirement C1, accommodating the need for Single Sign-On (SSO) integration, since the chosen system can be later replaced by any SSO that implements OpenID-Connect.

Keycloak is selected primarily due to its significant presence as a major open-source project and its backing by RedHat. Notably, this year it was contributed to the CNCF foundation to ensure ongoing development [113]. Keycloak will be



deployed within the Kubernetes cluster, capitalizing on Kubernetes' scalability and monitoring advantages. This approach also aligns with the security principle of minimizing direct software installations on machines to mitigate potential vulnerabilities.

The approach of the authentication is further explained in section 4.3.1, while the role definition is covered in section 4.3.1.

### *Ingress*

To access services like Keycloak through URLs from outside the cluster, an Ingress manager 3.2.2 is required. For this task, Contour has been chosen due to its status as one of the larger Ingress Controller projects, as well as its CNCF Incubation status. Moreover, Contour significantly implements the Gateway API, enhancing Kubernetes networking capabilities [114, 115].

Ensuring Keycloak's secure operation demands effective certificate management. For this purpose, cert-manager stands out as the chosen solution. Its affiliation with CNCF brings valuable project resources, while its adoption by multiple Ingress controllers, such as Contour [116] and NGINX [117], highlights its reputation as a widely recognized certificate management solution.

### *Storage*

Lastly, a robust storage solution is indispensable to address the requirements for private and public shares (S2/S3), as well as to provide permanent storage for the web application and Keycloak. As previously highlighted in this section, an NFS server has been designated for this purpose. This permanent file storage remains accessible even in the event of any disruptions to the cluster.

Enabling Kubernetes to handle NFS resources requires the installation of an NFS provisioner. Kubernetes presents two recommended solutions for this purpose [63]. The first involves setting up an NFS server within the cluster and granting access, while the second option utilizes subdirectories on an existing shared storage.

This work favors the latter option. This decision is driven by the desire to maintain a distinct boundary between the permanent storage and the Kubernetes cluster. For instance, the cluster might employ distinct backup mechanisms compared to those utilized for storage.

Lastly, in the event of a cluster failure, students should retain access to NFS data via alternative means rather than facing a complete loss of access. It is important to acknowledge that if a cloud deployment is preferred, the current NFS setup would likely be substituted with a storage solution specific to the vendor. To make it easier to swap out the technology, whenever possible, the choice is made to mount folder through Kubernetes mechanisms.

Subsequently, the storage solution needs to be configured to offer the shares to mount in a format that can be used by the Operator to correctly identify which mount belongs to which user. This entails organizing all private and class shares within designated subfolders. To accomplish this, the storage solution provides an annotation to the Persistent Volume Claim. This annotation specifies the intended

path where the object should be placed. Further elaboration on this process is provided in the following section.

## 4.3 Operator

The Operator is responsible for overseeing all necessary Kubernetes resources for establishing the system. This section will outline the Operator's purpose and clarify which Kubernetes concepts are employed to fulfill the requirements.

The initial decision involved considering the adoption of either Custom Resource Definitions (CRDs) or an Aggregated API. These concepts are outlined in section 3.2.4. Ultimately, CRDs were opted for, primarily due to their reduced maintenance burden as they are fully managed by the Kubernetes API. Furthermore, the additional flexibility provided by an Aggregated API was not deemed crucial for achieving the prototype's objectives.

In total, the system will utilize two distinct Custom Resource Definitions (CRDs). One CRD will be designated to manage users, while the other one will handle classes.

### 4.3.1 Kubelab-User

The first CRD is called Kubelab-User, encompassing both students and lecturers. The design of the Manifest is intentionally kept straightforward, as demonstrated in Listing 4.1. The essential parameters to configure include the object's name, ID, and permission level.

```

1  apiVersion: kubelab.kubelab.local/v1
2  kind: KubelabUser
3  metadata:
4    name: teacher
5  spec:
6    id: "t01"
7    isTeacher: true

```

Listing 4.1: Kubelab-User Manifest

Figure 4.2 illustrates the necessary resources to be generated in order to meet the requirements. Every individual should be confined to a dedicated Namespace, encompassing all resources that are needed.

#### *Rights Management*

User privileges should adhere to the principle of least privilege. For students, this entails access only to resources within their designated Namespaces. Conversely, lecturers should possess rights to oversee both their classes and associated students, satisfying access requirement S4. To facilitate permission management, Roles can be employed, coupled with RoleBindings that tie them to Single Sign-On (SSO) users.

The Roles should contain permissions for students to list and scale Deployments. This empowers them to view, initiate, and stop their containers. Additionally, they

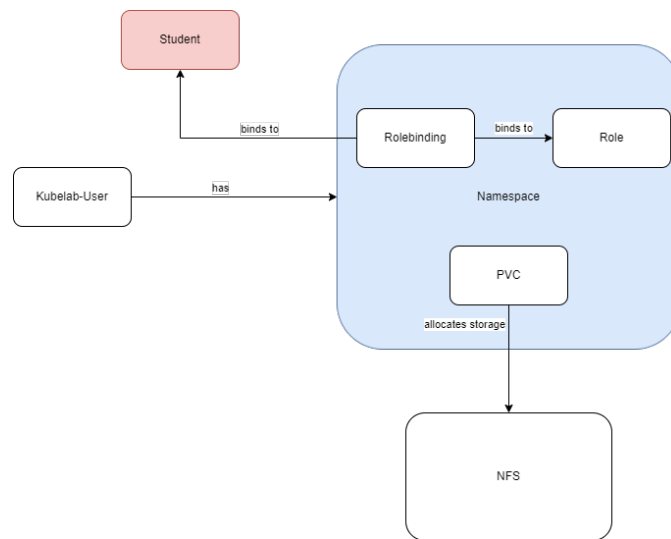


Fig. 4.2: Diagram of the Kubelab-User Custom Resource

require access to Service details to extract the information necessary to connect to their containers. Importantly, all privileges granted by the Role are confined to a single Namespace, ensuring that students possess no authority over resources beyond their respective Namespaces. A more comprehensive explanation of how these permissions are utilized can be found in the following Classroom section.

Lecturers, on the other hand, need a wider array of rights for multiple Namespaces, as they need to oversee containers belonging to their students. To achieve this, a Cluster Role without any Namespace restriction is implemented.

As for rights, lecturers need to be able to list, scale and update Deployments to manage the status of their students' containers. Additionally, they require the ability to retrieve and list Pods and Services, which grants them access to connection-related information. Finally, they should have the right to list all classroom Custom Resources in order to find the ones belonging to them.

These rights are broad and allow for interference with other classes when directly communicating with the cluster and not using the web-application. This decision stems from the limitations of RBAC controls, which presently lack the capability to filter based on Labels. Consequently, restricting rights while maintaining cluster-wide applicability becomes challenging. Thus, the chosen approach treats lecturers as trusted entities, assuming they will not alter classes belonging to others. Furthermore, the web application is designed to restrict access, as further described in Section 4.4.1. In a general overview, the authentication process functions as follows:

#### *Cluster Access*

The Keycloak-User must be manually configured within Keycloak and serves as the credential for both the web application and the cluster. To define the user's web application permissions, they are assigned either the "teacher" or "student" role.

To establish permissions within the cluster, a user must be associated with a group in Kubernetes. This group is subsequently linked to a Role through a RoleBinding. Users without a group are treated as anonymous and possess no privileges. As a Kubernetes group, the users' ID is used. The ID is then bound to assign the rights. Kubernetes determines the groups to which the current Keycloak-User belongs by referencing the “groups” attribute within the access\_token. To include the user\_id within the “groups” attribute, a mapper must be configured within Keycloak.

As previously mentioned, Kubernetes lacks an inherent user concept and instead grants permissions by analyzing the provided tokens. If the token is signed by the certificate authority (CA) and is deemed valid, permissions are allocated following the methodology that was described earlier. A visual representation of the entire process can be found in Figure 4.3.

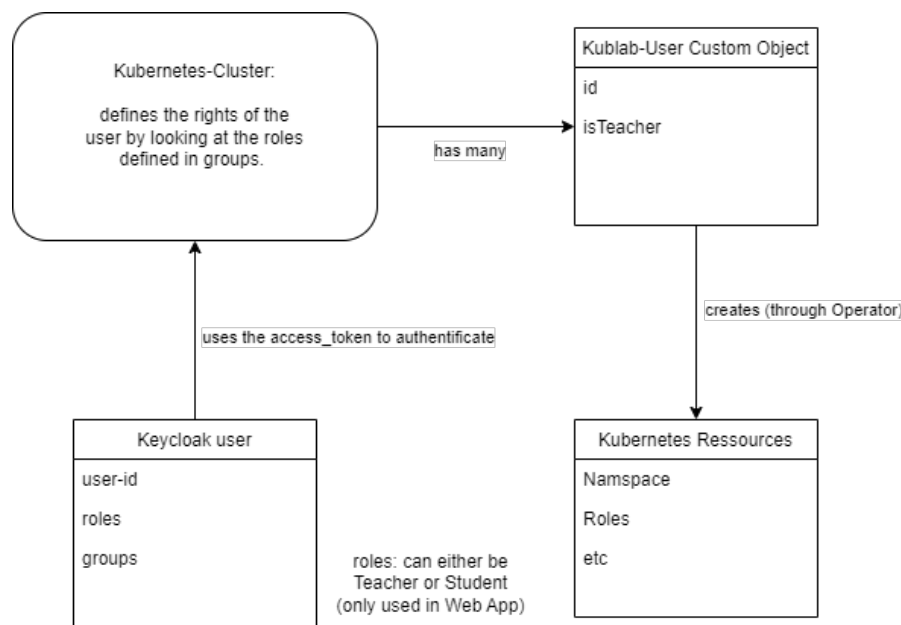


Fig. 4.3: Diagram of Keycloak vs. Kubelab-Users

### Storage

The subsequent requirement (S2) entails implementing user storage. To fulfill this, storage from the NFS server is employed. As detailed in the cluster design, a provisioner is set up, enabling automated storage allocation when creating and utilizing a Persistent Volume Claim. To enable this functionality, a StorageClass must be predefined, outlining the characteristics of the provisioner.

#### 4.3.2 Classroom

The second Custom Resource Definition defines the classroom. The fundamental Manifest for a classroom resource is illustrated in Listing 4.2. This configuration

allows lecturers to customize the class according to their preferences, granting the students root access or set the container-image utilized.

```

1  apiVersion: kubelab.kubelab.local/v1
2  kind: Classroom
3  metadata:
4    annotations:
5      nfs.io/storage-path: class
6    name: java-classroom
7  spec:
8    templateContainer: "floreitz/kubelab_base:latest"
9    allowUserRoot: "true"
10   rootPass: "toor"
11   teacher:
12     spec:
13       id: "t01"
14   enrolledStudents:
15     - spec:
16         id: "575103"
17     - spec:
18         id: "575104"

```

Listing 4.2: Classroom Manifest

Once this Manifest is deployed and picked up by an Operator, a configuration as seen in Figure 4.4 will be spawned.

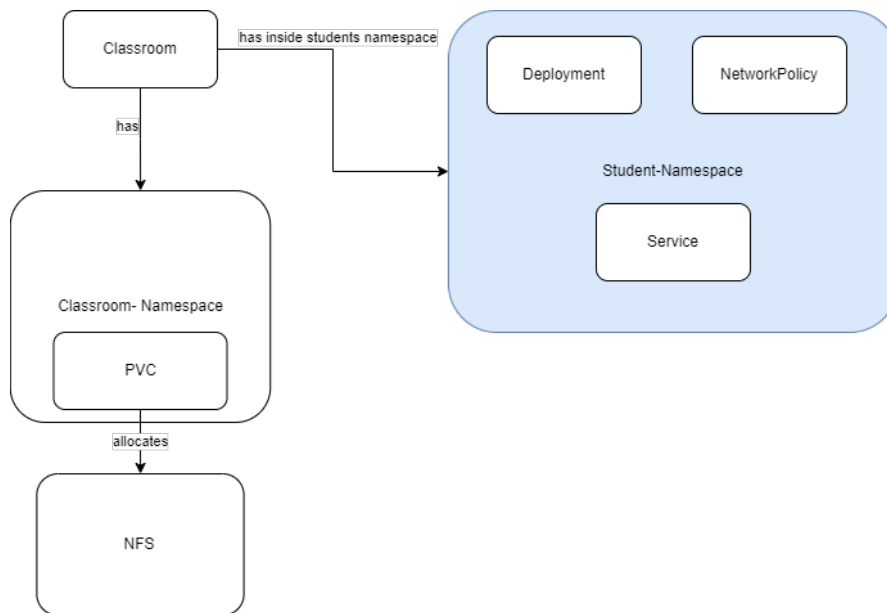


Fig. 4.4: Diagram of the Classroom Custom Resource

First, a Namespace for the classroom has to be created. Inside it, a PVC should be created to allocate storage for the classroom share, enabling access for all class members, as described in S3. The remaining classroom resources are directly placed inside the user's Namespaces, as can be seen in Figure 4.4. This approach ensures that students' resources are well isolated, instead of having them scattered around multiple Namespaces within the cluster.

### *Defining Students*

As can be seen in the Manifest 4.2, students are supplied as a list. This list is treated as a list of Kubelab-User objects, which are matched by their ID. The same principle applies to the teacher.

### *Student Container*

In this project, Deployments are used, rather than directly managing Pods for students' containers. Several factors contribute to this decision. First, as described in the fundamentals chapter, Deployments offer more functionalities, such as changing the image without recreating the Pods manually. Another reason is the possibility to instantly reset the container: In case a student destroys their Pod in a way that makes it inoperable, the student has the option to scale the Deployment down and up again in order to get a new container. By doing so, the new container has all the default configurations, except for their private storage. While this reduces long-term customization options for students, the benefit of resetting Pods instantly to the default setup could prevent long sessions of error hunting. Advanced students still have the option to create scripts that adapt certain settings to their liking and save them inside their private folder.

For the purposes of this work, the number of containers will be defined as one. However, this approach does present some drawbacks, as it does not take full advantage of the functions offered by multiple containers. An example is the use of specialized containers for Databases, which can be easily switched during lessons, or decoupling software versions from the main-container. Furthermore, Iorio et al. [4] suggests having a container inside a Pod specially for remote desktop support, which would fulfill C3. The main reason for this decision is the scope of this work, as its key purpose is to research whether the concept itself works properly.

Additionally, the option to grant users root rights for their container is available. The design for that, along with other container features, is further described in the Container section 4.5.

### *Storage*

Users' storage can, as described before, be mounted using the PVC inside their Namespace. The challenge lies in mounting the class share. By default, it is not possible to mount a PVC from another namespace, since it is a namespaced object. Currently, there exists a feature in early alpha that attempts to solve this issue, called Cross Namespace Data Sources [118]. However, since this feature remains under active development, an alternative solution was chosen. Given that the PVC under the management of this Operator determines the name and path of the NFS share, it becomes possible to get the complete mounting path to the NFS share. This information can then be used to mount the share directly through Kubernetes, ignoring the PVC object. To restrict students from writing to this share, the mount can be set to read-only by Kubernetes.

### *Network Restrictions*

The path used to fulfill requirement S5, namely restricting the network -for example for exam environments-, is using a Network Policy. In the prototype, this can be managed by setting the Exam-mode to true inside the Manifest. The Network Policy object deployed shall then restrict all traffic to and from the container export for SSH connections.

### *Accessing the Container*

Enabling access to the container, as described in S4, requires a system in which the container can be accessed externally. To achieve this, a NodePort Service is defined. Kubernetes then uses this Service to make the machine's SSH port accessible from outside the cluster. The primary function is to let all incoming traffic on the defined Port route to the container, as can be seen by Figure 4.5.

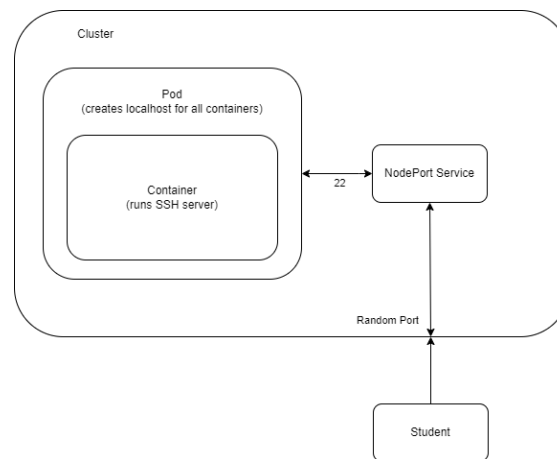


Fig. 4.5: Diagram of NodePort setup

### *GPU-Support*

For some classes, it may be advisable to schedule a GPU, as described in C4. This is possible using the limit system 3.2.2. To allocate a GPU device, there are plugins which detect the graphics unit. As GPUs presently do not support requesting, only the limit needs to be set, and it consequently functions as both the limit and the request. In cases involving multiple GPU types, differentiation can be reached through Labels 3.2.2. [119]

### *Common Pods*

Requirement S6, which sets the need for server-Pods that can be accessed by all students, can be satisfied using Services. For example, deploying Pods inside the class Namespace and creating a Service that exposes such Pods to the students. To restrict access, Network Policies that require the Pods to have the Label of the class to be accessed can be written. This makes it possible to have Pods that can be accessed by every student in the class.

### *Automatic Shutdown*

As outlined in requirement C5, an automatic shutdown of containers would be a useful way of saving resources. To achieve an automatic shutdown when a container remains unused, a liveness probe could be used. This probe can be configured to initiate the shutdown of containers if no SSH connection is established over a certain period.

### **4.3.3 Framework**

To implement the Operator, a framework to accelerate its development will be used. Iorio et al. [4] used a tool named Kubebuilder, which can boilerplate the CRDs as well as the Operator. This tool is further discussed in the Operator white paper [94]. The white paper also highlights the use of the Operator SDK, which embeds Kubebuilder and supports the implementation of Operators out of the box in Go, Ansible and Helm.

Both projects appear to exhibit close collaboration, even sharing some core developers. [120, 121] In a blog post where version 1.0 of the Operator SDK is celebrated, the relationship is further explained. The post states that, for Go Operators, Kubebuilder tools are used inside the Operator SDK [122]. This implies that the Operator SDK can be seen as an Extension for Kubebuilder, which has out-of-the-box support for Ansible and Helm Operators.

Since this project will focus on building a Go Operator, as it offers the most flexibility and has the most online resources, Kubebuilder will be used instead of the Operator SDK, as the former reduces the overhead.

## **4.4 Web-Application**

This section describes the design of the web application, which is used to provide the end user with a management interface. At the beginning of this chapter, the implementation of the prototype's two distinct workflows was explained. This subsection focuses on the requirements to achieve these workflows.

### **4.4.1 Requirements**

#### *Server*

Given that direct communication with the Kubernetes API is solely required by the website, a two-component structure is proposed for the application. This consists of a server component that establishes direct communication with the cluster and a client-side component responsible for displaying the frontend and the responses from the cluster. This design helps to protect the Kubernetes-API, since it may be restricted by firewalls or similar means to only communicate with the web-application server. It also safeguards the cluster from direct client-side requests, which are feasible due to the authentication design, by allowing the user to access the token from the OpenID-Connect provider.



A framework which enables the setup of this design in a single Codebase is SvelteKit. The server-side component uses the Kubernetes JavaScript Client<sup>1</sup> to communicate with the cluster. The server-side component should provide REST API routes for the client side, enabling the frontend to indirectly communicate with the cluster. This REST-based structure also facilitates the flexibility to switch out the frontend whenever necessary.

This design provides a mechanism to restrict direct calls made by the students to the cluster. For example, students have the rights to scale their Deployment, potentially to a number which could overload the server. By utilizing the REST route, the scaling capability is confined to just one or zero, eliminating the need to create an additional Custom Resource that wraps the original Deployment.

### *Authentication*

As stated before, the authentication provider should be able to be swapped out at any point. To avoid direct coupling to a specific vendor, a configuration should be established, which facilitates the substitution of the provider when needed.

There exist at least three providers that could achieve this: Lucia<sup>2</sup>, sk-auth<sup>3</sup> and Auth.js<sup>4</sup>. The native Keycloak library is not listed here, since it cannot be exchanged easily. All Frameworks have certain drawbacks: Auth.js is in the experimental phase for SvelteKit [123]. Lucia does not provide native OAuth integration; hence, the workload would be transferred to a second library. Finally, sk-auth did not receive any updates since July 2022 and a deprecation announcement was made by the developer [124]. Keeping these aspects in mind, Auth.js has been chosen, as it offers the most maturity, despite this not being directly the case for SvelteKit. Furthermore, it has the largest user base and the most robust documentation among the three options.

#### **4.4.2 User, Class, and Deployment Management**

To manage users and classes, as well as the deployment of the application itself, a management tool is required. The objective of this tool is to perform the following tasks without the requirement to start the development process entirely from the beginning.

The purpose of the management tool is to allow the creation of classes and students by using data from a list format, for example CSV. Moreover, it should enable viewing and deleting the Custom Resources as well. Another functionality should be to deploy the web-application and Operator directly to the cluster. Lastly, to assist lecturers in the creation of their own containers that align with this project, it should provide the functionality to take an existing container and modify it to fit the needs of the prototype.

---

<sup>1</sup> <https://github.com/kubernetes-client/javascript>

<sup>2</sup> <https://lucia-auth.com/>

<sup>3</sup> <https://github.com/Dan6erbond/sk-auth>

<sup>4</sup> <https://authjs.dev/>

General requirements for the tool are enterprise support, as well as an open-source structure that permits modifications to adapt to new requirements and facilitate continued development in the face of unforeseen circumstances.

## 4.5 Container

The current section explains the container layouts, required to execute the initial concept.

### 4.5.1 Student Container

This container should be seen as a blueprint for other class containers. It should have the capability to create users and configure them for SSH usage, alongside configuring all other essential settings. This can be done by creating an entry point script, which configures the container as needed and then tails the auth-log. `$ tail` becomes necessary, since the container shuts itself down once all processes that make out the container are completed. Tailing the authlog has the additional benefit of logging all connections made through SSH to the container's output console.

The container then can integrate with many IDEs such as IntelliJ IDEA, VS Code or Eclipse<sup>5</sup>.

#### *NFS Folders*

There exist two viable options for the location of the private folder: it can either be placed as the users' home folder or situated on a distinct mount. While putting the mount as the home folder has the benefit of persisting most files directly, it has some downsides too. Firstly, persisting configuration files could potentially result in errors when the classes are switched. Secondly, user configurations could be overwritten by other classes, without the user realizing this side effect. Lastly, the user folder could grow significantly in size as other programs are installed, which also place their configuration and project files in this folder. On the other side, using a separate folder creates more configuration expense, since the default paths for configuration files need to be manually modified to match the user folder every time the Deployment is scaled.

In preliminary testing, another problem appeared while setting the users' home folder to the NFS share. SSH is very peculiar about the rights on the folders preceding the `.ssh` folder, and it failed to work with the rights inherited from the NFS server, which are given by the `nfs-subdir-provider`. SSH mandates not only the `.ssh` directory to possess appropriate permissions, but also all preceding folders. This problem could potentially arise with other software applications as well. Upon reviewing the documentation, there appears to be no way to set the rights given by default from the provider. Attempting manual edits within the container could subsequently create complications concerning the provider's rights. Taking into

---

<sup>5</sup> Through the Remote System Explorer extension

consideration the aforementioned challenges, and in light of the issues detailed earlier, the optimal approach is to mount the folder within the user directory. The same principle applies to the public folder.

### *Skeleton*

Ubuntu offers the `/etc/skel` directory to alter the default folder for users when the user folder is either empty or has not been created yet. This behavior should be maintained, even though, as discussed in the preceding paragraph, the folder will already exist, since the mounts are set inside the user folder, which creates all preceding folders if they do not yet exist.

### *Users*

The group ID and user ID should be fixed to simplify the NFS access. Furthermore, passing the passwords as an environment variable has to be possible. This variable needs to be already encoded; otherwise it can be read out by the users. Consequently, the Operator must incorporate functionality to encrypt passwords, since the CRD, as seen in Listing 4.2, should include the password in clear-text to simplify management.

### *SSH access*

SSH should be configured to include root login for teachers and passwordless authentication. For the latter, a second file should be defined, to make it possible to continue using the normal `authorized_keys` file for other purposes.

## **4.5.2 Web Application Container**

The web application container must be containerized for deployment within the Cluster, enabling direct monitoring through the cluster environment. To achieve this, a container needs to be created with the capability to deploy the application. Additionally, it should support configurable settings, such as Keycloak URLs and Cluster URLs, allowing modifications without necessitating container reconstruction.

## **4.5.3 Operator Container**

To enable the Operator's deployment to any given cluster, without the need to rely on Kubebuilder, a container that can be deployed on any cluster should be built.

## Cluster Creation

This chapter will explain the creation and configuration of the server cluster to house both the web-application and Custom Resources.

### 5.1 Setup

The `$ kubeadm` tool creates a minimum viable cluster to be used as a basis. Initiating the installation requires a few prerequisites to be in place. First, swap must be disabled on all machines. Following that, the installation of the container engine is essential to allow the deployment of Kubernetes components as Pods.

The installation of CRI-O needs two environment variables. The first one is the current operating system and version, in this case `xUbuntu_22.04` and the second one is the CRI-O version, which needs to be set. Since the version is tethered to the Kubernetes version, the CRI-O version to be employed will align with the version utilized by Kubernetes. Afterward, the keyrings and the repository can be added and CRI-O, as well as the bundled runc version, can be installed through the package manager. Although CRI-O comes with its own version of runc to ensure compatibility, it is also possible to use the default version from the package manager or any other OCI-compliant runtime. Finally, the CRI-O Service needs to be enabled and started. [125]

The installation of `kubeadm` itself can be done through the distribution's package manager after installing the signing keys and repositories. The same applies to `kubelet 3.2.1`, which handles the communication between CRI-O and the Kubernetes API. To ensure the tools match the cluster's version, they are held back using `apt-mark hold`. Lastly, the cgroup driver needs to be set to allow Kubernetes to restrict resources for the Pods. The cgroups driver defaults to `systemd`, which is used by the Ubuntu distribution. [111]

To create the cluster on the control-plane machine, the `$ kubeadm init` command is used. This automatically creates the components required for Kubernetes to function. Having done this, the other Nodes can join the cluster by using `$ kubeadm join`. Once the Nodes have joined the cluster, the system is ready to deploy containers. To control the cluster from another system, a configuration file

can be exported. This file includes the private and public key to the cluster, which is loaded by `$ kubectl` to authenticate to the cluster.

### *Upgrading*

During this project, a new minor-version of Kubernetes was released. Kubernetes offers various strategies for cluster upgrades. However, given that the cluster was initially created using the `kubeadm` tool, the upgrade process from this tool was employed.

The upgrade process starts with one of the control-plane nodes. In this case, there is only one such node. First, `kubeadm` needs to be unhold and upgraded to the next version. Next, the command `$ kubeadm upgrade plan` can be used to generate a report detailing which version and components require upgrading. Afterward, the version can be upgraded. Lastly, the network add-on and container engine should be upgraded if an update is available. [126]

To proceed with Node upgrades, it is necessary to install the newest version of `kubeadm` and call the upgrade process that updates the configuration files. Next, since the Node is used to schedule Pods, it is necessary to drain it. `$ kubectl drain <node-to-drain> --ignore-daemonsets` ensures that no Pods are currently on the Node and no new Pods will be scheduled. The `ignore-daemonsets` is necessary, since Daemonsets cannot be drained, as they are designed to run on every Node. For example, Calico uses Daemonsets to connect all the Nodes. The downtime of one signals to Calico that this Node is currently not reachable. Next, `kubelet` and `kubectl` are upgraded. During this process, the Node will lose connection to the control-plane. After the upgrade is finished, the services need to be restarted and the Node is ready to host Pods again. To provide this information to the cluster, the command `$ kubectl uncordon <node-to-uncordon>` is used. `Uncordon` removes the Taint that Pods are not allowed to be scheduled here. [127]

## 5.2 Add-ons

Kubernetes itself is highly modular, allowing for customization. This means that several core-functions like networking are not included by default and require manual installation. The functions that are needed for the prototype will be discussed below.

### Networking – Calico

To set up a custom network IP address range for internal use while initializing the cluster via `kubeadm` the `‘-pod-network-cidr=192.168.178.0/24’` option was added. The process of setting up the Calico-Operator includes the installation of an official YAML Manifest. Following this, it is necessary to create a Custom Resource containing the correct CIDR. This enables the Operator to manage these settings correctly.

## Ingress – Contour

The Installation of Contour is similar to Calico, as it uses a Manifest. However, after testing the system, no connection could be established. The issue arose because Contour expects a load balancer to send the traffic to it. Yet, as this local project lacks an external load balancer, this aspect requires a different configuration. The official documentation states that NodePort or Host-Networking would be the recommended option in the absence of a load balancer. [128] NodePort exposes the ports for all cluster IPs and Host-Networking breaks the Network Isolation to the host. The chosen solution was to utilize external IP addresses. This redirected incoming traffic from the defined IPs and Ports to the intended Service. This approach helps limit incoming traffic to specific IPs, allowing for easier Firewall management. Although this option is not covered in Contour’s documentation, it functions without posing any issues.

### *DNS-Resolver*

The fact that this project currently runs on a local home network, made it not possible to use an official domain to route traffic. To remedy this issue, an installation of Pi-hole<sup>1</sup> was used for creating DNS entries, which route to the correct Services. An alternative to this would be modifying the HOST-file on the local system.

To create routes, Contour provides an HTTP-Proxy Object. It offers useful features, including support to restrict virtual host creation for specific namespaces. This can help prevent potential abuse of the prototype by avoiding the creation of too many hosts. [129] However, in this project, the Ingress Object from Kubernetes was employed. This choice was made because this generic object allows for the possibility to switch out the Ingress Controller at any point if the requirements happen to change.

```
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: keycloak-ingress
5    namespace: keycloak
6    labels:
7      app: keycloak
8  spec:
9    rules:
10     - host: keycloak.kubelab.local
11       http:
12         paths:
13           - pathType: Prefix
14             path: /
15             backend:
16               service:
17                 name: keycloak-svc
18                 port:
19                   number: 8080
```

Listing 5.1: Example Ingress Entry

<sup>1</sup> Open source DNS sinkhole with ad blocking <https://pi-hole.net/>

To create an Ingress Entry in Kubernetes, a YAML-file specifying to which Service an incoming request of a given domain is routed was created, as can be seen in Listing 5.1. The setup defines both the URL and the Service for routing traffic. To also enable the use of HTTPS, Cert-Manager was introduced.

## Cert-Manager

Cert-Manager<sup>2</sup> provides the certificates for Contour. The installation process is similar to the previous two methods. After installation, an Issuer needs to be created. Cert-Manager differentiates between namespace issuers, called “Issuer” and cluster-wide issuers, called “ClusterIssuer” [130]. For the purpose of simplicity, a ClusterIssuer is utilized to generate certificates for all namespaces. To enable successful generation, creating a Certificate Authority (CA) first is imperative. To achieve this, an Issuer is established within Cert-Manager, responsible for creating the CA. Subsequently, the second cluster-wide issuer employs this CA to generate all further certificates. The CA certificate can then be exported using `$ k get -n cert-manager secret root-secret -o yaml | grep ca.crt`. The string received then needs to be decoded with base64. The resulting certificate can then be imported into any Browser or OS to trust all certificates created by this cluster.

To use the ClusterIssuer for Ingress objects, certain fields in the configuration need adjustment, as shown in 5.2.

```

1  [...]
2  metadata:
3  [...]
4  annotations:
5    cert-manager.io/cluster-issuer: kubelab-issuer
6    ingress.kubernetes.io/force-ssl-redirect: "true"
7    kubernetes.io/ingress.class: contour
8    kubernetes.io/tls-acme: "true"
9  spec:
10   tls:
11   - secretName: keycloak
12     hosts:
13     - keycloak.kubelab.local
14   rules:
15   [...]
```

Listing 5.2: Ingress TLS

## Storage

### *Server Setup*

To set up NFS on the storage server, the NFS-server packages are installed, and a shared folder is created. A restriction is set to only allow the Kubernetes subnet to connect to the share. To simplify rights’ management, all connecting users are squashed to UID and GID 1000. This does not pose an issue, since the mounts are later restricted through Kubernetes when mounting. After this configuration, the

<sup>2</sup> <https://cert-manager.io/>

NFS service was restarted and validated by using `$ showmount -e IP` from the Kubernetes subnet, which showed that establishing a connection was only possible from inside the subnet.

### *NFS-Provisioner*

Before beginning, the `nfs-common` packages needed to be installed on all machines to enable access to NFS functionalities within Kubernetes.

To install the provisioner, there are sample files given on their GitHub<sup>3</sup> that can be modified to align with the requirements of this project. Furthermore, a custom Namespace was defined to ensure clear organization within the cluster. Additionally, the file system structure pattern was adapted, allowing the Operator to configure a specific sub-folder for its shares. To clean up obsolete folders when a user account is no longer active, those folders are archived with a distinctive prefix.

## 5.3 OpenID-Connect

Having the cluster in a state, where it can be used to deploy applications, Keycloak can be integrated.

The installation is done by creating a Namespace and a PVC for storage. To expose the application, a Service is written to make the Pods accessible. This Service is then exposed using an Ingress Object. To run Keycloak itself, a Deployment is created.

### 5.3.1 Configuring Keycloak

First, a realm needs to be created. Realms are used to manage a set of users. Then a client is created, which is an entity that can request an authentication through Keycloak. To modify the token, given out after authentication is fulfilled, to the correct format, a token mapper was used. It allows changing the format of the token by mapping a value that can be set through Keycloak to a specific part of the token. In this case, as described in the design, the user ID was mapped to the `groups-claim`. Furthermore, the role that is used for the frontend to differentiate between students and lecturers, was mapped as well. The configuration can be seen in Figure 5.1.

To test out if the configuration made above is working properly, the `$ kubectl oidc` plugin can be used. With this, it is possible to see the full token sent by Keycloak after authentication. An example output can be seen in figure 5.2.

### 5.3.2 Configuring Kubernetes

To make Kubernetes accept Keycloak as a valid way to authenticate, it is necessary to add it as the OIDC provider. For this, a few changes needed to be made to the

<sup>3</sup> <https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner>



Mapper type	User Client Role
Name * ?	groups
Client ID ?	kubelab
Client Role prefix ?	
Multivalued ?	<input checked="" type="checkbox"/> On
Token Claim Name ?	groups
Claim JSON Type ?	String
Add to ID token ?	<input checked="" type="checkbox"/> On
Add to access token ?	<input checked="" type="checkbox"/> On
Add to userinfo ?	<input checked="" type="checkbox"/> On
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

Fig. 5.1: Keycloak mapper configuration

```
{
  "exp": 1682345578,
  "iat": 1682345278,
  "auth_time": 1682345247,
  "jti": "3cb11ff9-454a-4f69-a5af-82d5d0bc7905",
  "iss": "https://keycloak.kubelab.local/realms/kubelab",
  "aud": "kubelab",
  "sub": "9a7a2e46-4856-430d-b16f-4f39fa691abf",
  "typ": "ID",
  "azp": "kubelab",
  "nonce": "NgWfE_vxXV-pUPmrPGG8LWCv4ISwHIhRpjaVusNwqL4",
  "session_state": "fd743cfd-9991-4870-a701-c1fa8600ba60",
  "at_hash": "8o6ShENkFQdDnsphBi_Ajg",
  "acr": "0",
  "sid": "fd743cfd-9991-4870-a701-c1fa8600ba60",
  "email_verified": true,
  "name": "Florian Reitz",
  "groups": [
    "admin"
  ],
  "preferred_username": "flo",
  "given_name": "Florian",
  "family_name": "Reitz",
  "email": "flo@reitz.dev"
}
```

Fig. 5.2: Data received from Keycloak

Kubernetes-API server. To make these changes, the Manifest for the API server was changed, which forces the server to redeploy the API automatically. These Manifest files, located at “/etc/kubernetes/manifests”, are used by Kubernetes to create their core-components. If a change is detected, the component will be shut down and redeployed with the new configuration.

While identifying the correct configuration, the API Server did not restart properly. As a result, accessing the cluster through the Kubernetes API became unfeasible. This meant tools such as \$ `kubectl` ceased to function, thereby making it impossible to access and read the logs of the Pod.

### *Debugging without Kubernetes*

To gain insight into this problem, it is important to first comprehend the inner workings of Kubernetes. As mentioned earlier, kubelet utilizes Manifest files to establish Pods for the core components. It does not check, before deploying, whether the files are correct. Despite an erroneous configuration, the kubelet component directs CRI-O to deploy the Pods, causing Kubernetes’ installation to malfunction. To identify and rectify the error, it is necessary to review the error logs of the Container Engine. This can be accomplished using a command-line tool. For CRI-O this is \$ `crictl`. It operates similarly to \$ `docker`.

The issue stemmed from an invalid option added to the Kubernetes-API. The prefix used to identify the roles ended with a colon that was picked up by the YAML interpreter. To remedy the problem, the whole command has been wrapped in quotation marks, which can be seen in Figure 5.3. Afterward, the API server started to work correctly again.

```
- --oidc-issuer-url=https://keycloak.kubelab.local/realms/kubelab
- --oidc-client-id=kubelab
- "--oidc-groups-prefix=keycloak:"
- --oidc-groups-claim=groups
- --oidc-ca-file=/etc/kubernetes/kubelab-ssl/root.ca
- --oidc-username-claim=email
```

Fig. 5.3: OIDC options for Kubernetes

To configure the rights that the Keycloak user has inside the cluster, a RoleBinding is created. In the example 5.2, it can be seen that the groups-claim has one entry: admin. When read into Kubernetes, this group is added to the user with the prefix keycloak, which results in keycloak:admin. Next, a RoleBinding can be established to grant rights to this group. An example for the admin Role is provided in Listing 5.3. [131]

```
1  apiVersion: rbac.authorization.k8s.io/v1
2  kind: ClusterRoleBinding
3  metadata:
4    name: keycloak:admin-role-binding
5  subjects:
```

```
6      - kind: Group
7        name: keycloak:admin
8        apiGroup: rbac.authorization.k8s.io
9      roleRef:
10       kind: ClusterRole
11       name: cluster-admin
12       apiGroup: rbac.authorization.k8s.io
```

Listing 5.3: Rolebinding for Keycloak Admin

The certificate file needed for this process was the CA generated by cert-manager. This file is then mounted into the API Pod.

To test whether rights are correctly assigned, the `$ kubectl` tool is configured to use Keycloak as the authentication mechanism. To set this up, the kubeconfig file housing crucial details to connect to the cluster, such as certificates and server URLs, must be modified to employ the OIDC plugin. When using `$ kubectl` afterward, a browser window will open, requesting login. If the user possesses the required rights, Kubernetes will execute the command.

### *Permanent Storage*

Upon the Pod's destruction, the Keycloak database is lost. To create a permanent storage for the database, the previously configured NFS-server was used.

To achieve this, a PVC was included in the Keycloak YAML file to request 100Mi of storage on the NFS-server, which can be observed in Listing 5.4. The access mode permits multiple containers to both read from and write to the storage, enabling the utilization of multiple Pods for this storage.

```
1  kind: PersistentVolumeClaim
2  apiVersion: v1
3  metadata:
4    name: keycloak-claim
5    namespace: keycloak
6  spec:
7    storageClassName: nfs-client
8    accessModes:
9      - ReadWriteMany
10   resources:
11     requests:
12       storage: 100Mi
```

Listing 5.4: PVC for Keycloak

## Implementation

Having the cluster properly set up, the implementation chapter reviews the process of creating the Operator, web-application, management tool and the different containers.

### 6.1 Operator

This section explains the implementation process for the previously designed Operator.

#### 6.1.1 Setup

As mentioned earlier, Kubebuilder provides scaffolding for both the Custom Resource Definition (CRD) and the Operator.

Before installing Kubebuilder, it is necessary to have both the Go language and Docker installed. Since the machine used for implementation is a Windows Linux Subsystem installation, Docker Desktop is employed. Due to the outdated Go version in the package distribution, it becomes necessary to manually download the latest binary and perform the installation. The installation of Kubebuilder itself can then be conducted using only a few commands. Once the installation is completed, the system is ready to be used. During the development phase, it is important to keep in mind that Kubebuilder utilizes the current context of the kubeconfig file for all communication with the cluster.

The folder structure provided by Kubebuilder is quite ample, as can be seen in Fig. 6.1. In this work, two folders hold particular significance. Firstly, within the internal folder, there is a controller folder that contains all the created controllers. Secondly, within the API folder under the correct version number, all created Custom Resources can be found. Controllers are bound by the main.go, which is located in the cmd folder and works as the entry point.

There is an important distinction between the controller that is mentioned here and the Kubernetes Controller. Although both concepts share some similarities, in this context, the controllers refer to objects that monitor a single Custom Resource. On the other hand, Kubernetes Controllers are, in this context, full Operators that watch over multiple built-in resources.

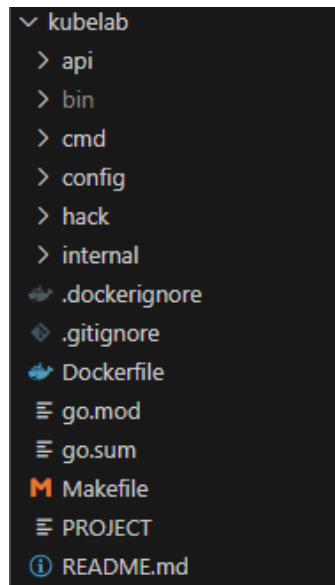


Fig. 6.1: Folder structure of Kubebuilder

The project creation is initiated with the following command: `$ kubebuilder init --domain kubelab.local --repo kubelab.local/kubelab --plugins=go/v4`. Domain specifies the domain used for groups, which can be seen when interacting with `kubectl`. The `repo`-argument defines the name of the go module. Lastly, `plugins`-argument defines the go version used. Currently, the default choice is `go/v3`, which is deprecated for Kubernetes versions above 1.25. To ensure compatibility, `go/v4` is selected.

### 6.1.2 Kubelab-User

#### Custom Resource

The creation of the Custom Resource involves running the `$ kubebuilder create api --group kubelab --version v1 --kind Kubelab-User`, which provides the scaffolding for the CRD and Controller.

To configure the Custom Resource, the file `kubelabuser_types.go` is automatically generated in the “`api/v1`” folder, and can be edited to include the fields specified in the earlier designed Manifest. One issue discovered during the process is related to uniqueness. By default, the fields can be set to any value without a restriction on uniqueness. This issue can be overcome using validation hooks that check beforehand, for example, if the uniqueness of the ID is given. [132] Currently, no validation hook has been integrated, requiring the administrator to ensure the uniqueness of the IDs.

To set the current status of the resource that the administrator can view, a field in the status section is added with the type `metav1.Condition`. This type automatically imports the default Kubernetes conditions that the resource can assume.

After editing the file, `$ make` has to be run to generate the new Manifests that will be deployed to the cluster. The generated Manifest can be found inside “config/crd/bases”.

## Controller

To implement the controller, the first step involves defining the managed resources. The controller can be found at “internal/kubelabuser\_controller.go”. At the bottom of that file, a function called `SetupWithManager` 6.1 can be found. Inside it, the resources being managed have been defined.

```
1 func (r *KubelabUserReconciler) SetupWithManager(mgr ctrl.Manager) error {
2     return ctrl.NewControllerManagedBy(mgr).
3         For(&kubelabv1.KubelabUser{}).
4         Owns(&v1.Namespace{}).
5         Owns(&v1rbac.Role{}).
6         Owns(&v1rbac.RoleBinding{}).
7         Owns(&v1.PersistentVolumeClaim{}).
8         Complete(r)
9 }
```

Listing 6.1: Manager Setup

After the setup, the reconciliation loop is triggered each time a watched object is created, updated, or deleted. To have the permissions needed to watch over these resources, RBAC rules have to be set. In Kubebuilder, this can be done through annotations 6.2.

```
1 //+kubebuilder:rbac:groups=kubelab.kubelab.local,resources=kubelabusers,
2   verbs=get;list;watch;create;update;patch;delete
3 [...] // Rules for finalizers and status
4 // RBAC group to create and delete namespaces
5 //+kubebuilder:rbac:groups="",resources=namespaces,verbs=get;list;watch;
6   create;update;patch;delete
7 [...]
```

Listing 6.2: RBAC rules for Kubelab-User controller

To properly work with the reconciliation, as described in the fundamentals section 3.2.5, the controller checks every part of the object every time it reconciles.

To achieve this, at first, the object in question is fetched. When fetching is not possible, it can be inferred that the object has been deleted, leading to the termination of the reconciliation process.

Otherwise, a check is performed to determine if the required resources associated with this object, such as the Namespace, already exist. If not, they are created. An illustration of a function responsible for resource creation is provided in Listing 6.3. First, this function creates the object, then it sets the `OwnersReference`, which deletes the object when the owner is deleted. After every resource has been created, the status is updated accordingly. The status can be viewed by the administrator using `$ kubectl`, which shows useful information in case the reconciliation fails at any point.

```
1 (r *KubelabUserReconciler) namespaceForUser(user *kubelabv1.KubelabUser) (*
2   v1.Namespace, error) {
3     ls := labelsForUser(user.Spec.Id)
```

```

3      ns := &v1.Namespace{
4          ObjectMeta: metav1.ObjectMeta{
5              Name:      user.Spec.Id,
6              Labels:    ls,
7          },
8      }
9      if err := ctrl.SetControllerReference(user, ns, r.Scheme); err != nil {
10         return nil, err
11     }
12     return ns, nil
13 }

```

Listing 6.3: Function to create Namespace

## Challenges

### *Namespaced vs. Non-Namespaced resources*

After deploying the Operator for the first time, an error occurred, indicating that namespaced resources cannot have ownership over cluster resources. This error originates from the fact that, by default, the newly created Kubelab-User is a namespaced resource, while Namespaces are cluster-wide resources. This mismatch is the cause of the error. To remedy this, an annotation was introduced in the type definition of the CRD, which scoped the Kubelab-User to a cluster-wide resource. Consequently, ownership of Namespaces becomes feasible.

### *Deletion*

Another challenge was Namespace deletion. Typically, a mechanism deletes all objects with the OwnersReference set to the deleted object. While this worked flawlessly for Pods and Deployments, the Namespace remained undeleted. To identify whether the OwnersReference is not correctly set, the Operator has to be debugged.

The Go-plugin in VS Code supports debugging by installing delve. After setting up breakpoints and configuring the launch.json to run the 'cmd\_main.go' file, debugging becomes functional and the reconciliation-request can be paused at any point. This revealed that the OwnersReference was correctly set.

To address the deletion issue, finalizers were incorporated into the code. They manage the removal of all cluster-wide objects by manually deleting them. This approach offers an additional advantage: it ensures that any resource within the Namespace is also deleted. Consequently, even if other objects might encounter difficulties with self-deletion, the Namespace deletion forcibly removes them.

To add the finalizer, there is a function that takes the resource as well as an identifier. This identifier signals which finalizer is handled at which function of the code. Upon the addition of the finalizer, an additional check must be implemented. When the object has a deletion timestamp, it is checked, if the object has a finalizer with the correct identifier. If this is the case, the resources requiring cleanup can be eliminated. After this process concludes, the finalizer is removed, prompting Kubernetes to destroy the object.

### 6.1.3 Classroom

#### Custom Resource

The Custom Resource Definition implements all the fields that are shown inside the Manifest, as displayed in Listing 4.2. Similar to the Kubelab-User discussed before, it implements the conditions object.

#### Controller

The classroom controller operates in a manner similar to the Kubelab-User controller, fetching the object and generating the necessary resources. However, distinctions exist, as this controller might spawn items within Namespaces that may not exist, especially if incorrect IDs are employed when referencing Kubelab-User objects. To avoid this, a check is incorporated to determine whether the referenced Kubelab-User resources truly exist before attempting to create resources within the Namespace. In the first version, all Kubelab-User objects are fetched and compared against the list from the Manifest. This logic may be outsourced to the Validation Hook mentioned in the first controller.

Subsequently, to generate the student Deployments, the controller loops through all the students in the class and creates a Deployment if it does not already exist. While within the loop, the image is also examined. If the image of the student's Deployment does not align with the current image of the class, it undergoes an update.

Furthermore, to ensure that students who leave the class lose access to their container, a comparison is made between the list of all students with a class Deployment and the current class list. These operations can pose a burden on the API when thousands of objects are queried. To remedy this issue, an Indexer was introduced.

#### *Indexer*

An indexer works by caching the objects, thus lightening the load on the Kubernetes API [133]. This solution enables the listing of all cached items without imposing any additional load on the API. It is important to note that the “Get” operation, which returns a single object, does not support field indexers. Therefore, a “List” operation is necessary, which checks that only one result is returned after the call.

The addition of the indexer requires it to be first included in the setup function. This function takes four arguments, as seen in Listing 6.4. The first argument is the context in which it needs to be run. The second argument defines the object-type that needs to be indexed, which here would be Kubelab-User. This is followed by the third argument, a string-key that is used to access the index later. Lastly, the fourth argument of the function is the Kubelab-User's field, which should be used for the index. In this case, the user ID is employed.

Wrapping around the function call in Listing 6.4 is an error assignment, that allows assigning the error, if one occurs during the indexing, and returning it



without having a variable assigned outside the if-construct. The assignment and the actual if condition are separated by a semicolon.

```

1  if err := mgr.GetFieldIndexer().IndexField(context.Background(),
2    &kubelabv1.KubelabUser{}, userOwnerKey,
3    func(rawObj client.Object) []string {
4      user := rawObj.(*kubelabv1.KubelabUser)
5      return []string{user.Spec.Id}
6    }); err != nil {
7    return err
8  }

```

Listing 6.4: Controller Indexer

To retrieve the objects, the indexer's string-key can be used inside the list function.

## Challenges

### *Garbage-Collection*

As discussed before, cluster-wide resources are not deleted without a finalizer. However, at a certain stage, the deletion of even the namespaced resources was not occurring as expected. Despite conducting a thorough examination of the Operator, no issues could be identified. In order to confirm that the Operator was not the underlying cause, a second Operator was developed. This second Operator was designed to manage only one Pod, yet even in this scenario, the Pod was still not being deleted. After skimming through all the logs that Kubernetes offers, the root cause was identified, namely, the garbage collector was constantly crashing. Further research indicated that the issue stemmed from an update of the networking add-on, Calico, which had created a service account with insufficient rights. This rights issue resulted in a crash of the complete garbage collection. [134]

At the time of writing, this issue had been addressed within the GitHub repository; however, no new release of Calico was available. To proceed with this work, the correct rights were manually edited into the service accounts responsible for the crash. Subsequently, the garbage collector began functioning as expected. It is relevant to note that this process needs to be repeated each time the cluster is restarted.

### *NFS-Archival*

During the setup of the Operator 3.2.5, an error occurred, wherein the storage provisioner failed to archive the folders properly belonging to deleted users. While investigating the Deployment of the NFS provisioner, an error was discovered, which stated the Operator was unable to locate the folder for archiving. This error stems from an update of the provisioner, which was conducted a few days before. The project's team overlooked custom path patterns during archiving, which led to the Operator errors. [135] Since no image had been released, after the error was fixed, the only way to remedy this issue was to build the container locally and push it to a private docker registry. Following this adjustment, the folders are once again correctly archived.

### Shutting down Containers

The automated procedure for shutting down containers did not function as anticipated, thus presenting an additional challenge. The liveness probe, as can be seen in Listing 6.5, checks whether a current SSH connection is active. In case it is not, after a delay, the Pod is declared dead and destroyed. The problem lied within the Deployment that manages the Pod. Upon detecting the destruction of the Pod, it healed itself by recreating it. The liveness probe hence needs to scale the Deployment instead of destroying the Pod.

```

1  LivenessProbe: &v1.Probe{
2      ProbeHandler: v1.ProbeHandler{
3          Exec: &v1.ExecAction{
4              Command: []string{
5                  "/bin/bash",
6                  "-c",
7                  'active_connections=
8                      $(netstat -tnpa | grep 'ESTABLISHED.*sshd' | wc -l)
9                      if [ "$active_connections" -eq 0 ]; then
10                         exit 1
11                     fi
12                     exit 0',
13              },},},
14      PeriodSeconds: 30,
15      FailureThreshold: 2,
16      InitialDelaySeconds: 300, // Initial delay of 5 minutes
17  },

```

Listing 6.5: Liveness probe to shut down the container

To achieve the scaling, a Horizontal Pod Autoscaler (HPA) can be employed. This HPA can be configured by defining a custom metric to monitor instances of Pod failure. [136] An alternative path would be to watch the Pods inside the Operator and scale the Deployment down in case of a Pod failure. Since additional research on this is needed, it was not possible to implement this behavior within this work.

## 6.2 Web Application

This section explains the implementation of the web application, as outlined in the design chapter, as well as the challenges during this process.

### 6.2.1 Keycloak Integration

The integration of Keycloak was performed using Svelte hooks. Hooks are app-wide functions that are invoked after specific events. In this case, the handle hook that is called every time a request is made to the website was used. This hook is only available on the server side. [137] In the hook, the Keycloak-provider is set up inside the Auth.js library, by supplying the issuer-url, the client ID and secret. All three are supplied via dynamic private environment variables.

Auth.js allows modifying the callbacks to contain specific fields. In this scenario, the JWT callback, which is called whenever a JWT token is received (for example

after logging in), had to be modified. Commonly, Auth.js strips many fields by default to expose as little information as possible, yet the Kubernetes library needs access to special fields from the Keycloak response. To gain access to these fields, the callback that supplies the JavaScript object is overwritten. For example, the `id.token` is made available, to send it through the REST API for authenticating against the cluster. Furthermore, since the token is saved inside a session, that callback has to be overwritten as well to accommodate the newly modified token and correctly save it.

### 6.2.2 Kubernetes Integration

After obtaining the token within the session, the subsequent step is to authenticate against the cluster to retrieve the Deployments. As described in the design section, REST-API routes are used to connect the front and backend. Svelte itself offers an option to create these routes inside the codebase. For this purpose, a folder structure, for example `"/api/kubelab"`, is created. This structure houses an `+server.js` file that includes a GET function. If dynamic routes with parameters, for instance, scaling a specific Deployment, are needed, a `[slug]`-folder may be created. For example, the `"/api/kubelab/scale/[slug]"` folder structure allows a URL parameter that can be read out inside the `+server.js` file. The GET function is invoked whenever a GET request is sent to an API route that matches the folder structure. [138]

These routes use the Kubernetes JavaScript library to communicate with the cluster. For example, inside the scale route, users can start and stop their container. For this, they need the scale permission inside Kubernetes, which theoretically allows for any number of Pods. However, since the scaling is done on the server side, the user can only choose between 1 or 0. The following steps broadly describe the basic design of any given route.

First, the token is passed on inside the request. This token is then decoded by using a Base64-algorithm, which reads out the necessary data, such as the ID of the user. To make the request, the Kubernetes URL and Port, the CA, as well as the token, are required. If the user had altered the token prior to submitting it to the API, the request would fail at this stage. This is because Kubernetes can detect any modifications made to the token. Notably, the URL and Port combination, along with the CA, are server-side stored and never transmitted to the client. Afterward, the received response is forwarded to the client.

### 6.2.3 Functions

This section will focus on explaining the inner workings of several notable functions supplied by the web app.

#### *Class Table*

The class table component allows the user to start or stop their container, which in the background scales up or down the Deployment. If one container is already

running and another is initiated by clicking the start button, the former container will shut down automatically. This ensures that only one container can be active at any given time. To display all available classes, the REST API makes a request to the cluster, which lists all Deployments available inside the users' namespace. In order to start or stop the container, a request can be issued to scale the Deployment to either one or zero, depending on the current state. The user cannot alter this value, since it is hard-coded inside the server-side code. The scale route works by utilizing a slug that supplies the name of the Deployment to be started or stopped. All other Deployments are set to the scale of zero to ensure that only one class is active at a time.

Once a container is active, the connection button can be pressed. This button forms an SSH-string that the user can paste inside their terminal. To achieve this, a call to the REST-API is made, which returns a string the user can use to connect. This route uses the slug as well to get the name of the Service, which is equal to the name of the Deployment that should be connected to. Afterward, it reads out the NodePort given by the system. This, along with the load balancer's IP (provided by the environment) and the username (retrieved from the decoded token), is combined to construct the connection string, similar to the following example: `$ ssh -p22222 user@host`

### *File-Upload*

File uploads take place on the students side for the SSH-Upload and on the lecturers side to hand out files to the class by using a NFS-share. Both of them operate in a similar manner, as they both mount the NFS shares directly to their file system and use normal file operations to modify the files. A notable security concern arises from the fact that the cluster does not need to verify the validity of the access token. This vulnerability could potentially allow an attacker to alter the token, substituting it with another student's name, thereby gaining unauthorized access. This issue can be mitigated by making a request against the cluster, checking whether the user is entitled to read out all the Pods in their supposed Namespace. If this request fails, the token is tampered with and the upload is stopped. If, on the contrary, the user is entitled to read the Pods, it can be assumed that they possess the ability to upload data into the folder corresponding to the name within the token.

### *Teacher Table*

The teacher's class table lists all classes the lecturer teaches. To achieve that, first, all classes are fetched and scanned to identify whether the ID of the user and the lecturer listed in the Manifest matched. This approach was chosen because the library does not permit searching through the "Spec"-fields.

To improve performance, it was discovered that searching through Labels may be a viable solution. A check was added in the Operator, which includes a Label for the teacher as defined in the "Spec"-section. While it is feasible to manually set the Label within the Manifest, the intention is to maintain simplicity in these files

and prevent end users from duplicating values across different sections. Therefore, the responsibility has been delegated to the Operator. By introducing the Label to the Deployment, the function is limited to fetching only the Classrooms that correspond directly to the teacher's ID. This leads to a quicker method of fetching all classes, even when numerous classes are present within the cluster.

In order to manage the users of the classes, all Deployments that have the same name as the class are searched. From that point onward, the identical table, as described in the class table section, is utilized. The only distinction lies in the fact that when requesting the connection string, the string for the root user is provided, instead of the string for the regular user. To accomplish this, the API that handles the creation of the connection string is extended to have one body argument: A boolean, which defines if the user has teacher status. If this is true, the name of the student is exchanged to root.

Moreover, when a teacher scales up a student's Deployment, it does not trigger the shutdown of any other Deployments belonging to that student. This approach is taken to guarantee that the student is not unexpectedly logged out while actively engaged in another class.

#### 6.2.4 Challenges

##### *Keycloak*

One major challenge at the beginning of the project's implementation were the certificates. Whenever connection attempts were established using Keycloak, an "unable to verify leaf signature"—error occurred. To ensure this was not a general issue, a GitHub SSO client, which worked directly without a hitch, has been created. This highlighted that the problems stemmed from the self-signed certificates, which are used to ensure encrypted communication. In Node-based projects, there typically exist two ways to overcome these issues: either supplying the certificate through the `NODE_EXTRA_CA_CERTS` [139] environment variable, or ignoring TLS issues with `NODE_TLS_REJECT_UNAUTHORIZED` [140]. The variables were set through the `.env`-file that is automatically read by SvelteKit. However, both of these approaches failed. Another attempt involving configuring NPM to ignore strict-ssl, with the `$ npm set strict-ssl true` command, had no effect either. [141]

The first solution was to disable the forced SSL redirect, so Keycloak could be accessed by HTTP. Editing the Kubernetes cluster Manifest accomplished this. Although successful for login, Keycloak internally redirected SSO to HTTPS, causing unintended side effects, despite avoiding the leaf error. On the callback, the issuer URL in the JWT token pointed to the HTTPS page instead of the called upon HTTP page. This resulted in a verification error, as the library expected the request to be handled by an HTTP page. One way of addressing this consists of changing the issuer URL inside the response before verification. However, this approach would meddle inside the libraries and, given that it hurts the library's integrity, it is not recommendable.

In the process of exploring different approaches, the original idea was revisited. This time, instead of relying on the `.env` file to be read, the variable was directly injected into the code, thanks to the `process-object`. By doing this, the problem disappeared and an SSL connection could be made. As this is a problem that would only occur in development environments -since production environments normally have valid certificates for their SSO and the chance of man in the middle attacks inside the home network is very low-, it has been opted to ignore invalid and non-trusted certificates.

### *Unauthorized-Error*

During the initial implementation of the Kubernetes JavaScript library into the project, a response indicating that the server could not authorize the request was provided. While conducting tests, it became evident that the issue extended beyond authorization and there were problems with the authentication, as the system was unfamiliar with the user. This could be determined due to the fact that Kubernetes returns the username in any error messages. In order to investigate further, it became necessary to debug the server-side code.

Given that SvelteKit acts both a server and a client, each side presents different debugging processes. While the client-side code can be debugged either through the developer console or browser integrations inside the editor, the backend cannot be as easily debugged. Under normal conditions, logs are shown on the console, as well as debugger statements, but breakpoint usage was not possible. The root of the problem appears to be Vite, which does not support the proper creation of `ssr-sourcemaps`. [142] To enable these functions, the “`vavite-loader`”<sup>1</sup> package had to be installed for development environments. This package enables breakpoints by adding `sourcemaps` support. In order to use breakpoints, the application has to be loaded through the VS Code debugger, while using the `vavite-loader` [143].

To identify the origin of the issue, a service account and a token were created inside Kubernetes. This step is essential, as newer versions of Kubernetes do not create a token to increase security. Using this token, authentication through the website succeeded. This implies Kubernetes functions properly, suggesting the problem lied within Keycloak’s authentication.

This issue can be narrowed down further by keeping in mind that in the OIDC setup, which can be found in chapter 5.3.2, authentication against the cluster operated flawlessly. That points to the acquired token of the web-application as the source of the issue.

The problem was found inside Keycloak’s configuration. The cause was that it had falsely been assumed that clients shared the same tokens, as they share users. The so-called Realms manage users with the same credentials, while every client created generates their own different token. This means that, instead of creating one client for the OIDC client and one for the web application, both shall use the same client. [144]

---

<sup>1</sup> <https://github.com/cyco130/vavite/tree/main/packages/node-loader>

### *File Upload*

The file upload also presented a challenge, since text-based files were successfully written, yet encoded files like PDFs were broken. To remedy this, a binary file upload was written using blobs. By doing this, every file that is selected is converted into a blob and sent to the container through the API. Afterward, the Node file writer creates the blob onto the file system that mounts the NFS-share.

## **6.3 Management Tools**

After having created a functional prototype to prove the concept, a management tool is implemented to facilitate the resource creation process. The focus of this chapter is to evaluate a few well-known management tools that either support Kubernetes or are directly built for it.

### **6.3.1 Reviewed Tools**

This subsection will briefly discuss two tools that were analyzed, but for several reasons, did not fit the profile needed.

#### *Helm*

As Helm is a community built project that lacks a managing company, it offers no enterprise support. Helm has a strong focus on deploying Kubernetes apps, yet it does not support container extension. Its Custom Resources' deployment represents another limitation for the purposes of this project. Its documentation states that Custom Resource Definitions as well as Custom Resources are only supported for installation, but not deletion or upgrades. The solution to this used in the documentation is two to create two different charts, one for the CRD and one for the resource itself, which creates more complexity. The reason provided concerning this inconvenience is that the community does not have a common consensus on how to handle that type of data yet. [145]

#### *Puppet*

Puppet does offer enterprise support, although no direct pricing is visible on its website [146]. While the required tasks for this project are all possible, it does need more time to learn and understand, especially since it uses its own Ruby-based language to write puppet plans. These plans are lists of tasks the system should achieve. Should Kubelab eventually be scaled out in a way that makes the configuration more intricate, it may be worth revisiting Puppet as an alternative with monitoring capabilities.

### 6.3.2 Chosen Tool

The chosen tool for the prototype is Ansible, developed by RedHat, which, furthermore, offers enterprise support [147]. It has been chosen for this task because its Playbooks are human-readable, as they are written in YAML with support of Python functions. Moreover, all tasks can be carried out without generating a lot of overhead. Similar projects also relied on Ansible, for example the one developed by Iorio et al. [4]

In order to use Kubernetes functionality inside Ansible, the Kubernetes Python library has to be installed on the target system. The following paragraphs describe how the functions mentioned in the design chapter are implemented.

#### *Deploy Kubelab*

Kubelab is deployed in two stages: first, the Operator is created and afterward the web-application. To deploy the Operator, Ansible transfers the Manifest that is created by Kubebuilder and further described in the container section 6.4.3 to the target system. There, it will be deployed to the cluster. To accomplish this behavior in Ansible, the k8s module is used. This module requires the desired state, as well as the file that contains the Manifests as input.

The deployment of the web-app works similarly. The main difference is that, in the end, the certificates need to be transferred into the Deployment. Thanks to `$ k wait`, the process can be suspended until specific cluster events occur. In this case, it waits until the application Pod is deployed. Once the Pod is running, it is possible to use `$ k cp` to copy the required files inside the container. Thereafter, the container can start up normally with the correct certificates in place. This approach uses the container to move the files to the NFS server instead of using NFS directly. This makes it easier, at a later point, to switch the storage provider, as stated in the design chapter.

#### *Managing Custom Resources*

There are two Playbooks per Custom Resource. One to list the resource and one to manage it. As an example, the classroom Custom Resource is used. To list the current Custom Resource, first all classrooms are fetched using the k8s module. Afterward, a JSON query is used to filter the data. The management Playbook, on the other hand, allows the creation or deletion of classrooms. To accomplish this, it starts by reading in a CSV file 6.6, which contains all classes that should be deployed. As seen in the listing, enrolled students encompass one string that is internally separated using commas.

```
1 name,templateContainer,allowUserRoot,rootPass,teacher,enrolledStudents
2 javascript-class,floreitz/kubelab-node:latest,true,toor,t03,"575103,575111"
```

Listing 6.6: Example of classes.csv

To make the Custom Resources, Ansible expects the string to be a JSON list in the form `$ [{spec: {id: "575103"}},{spec: {id: "575111"}}]`. To achieve this, the string is first separated into a list of strings. This list is then modified by



a regex into the correct form. Lastly, a JSON parser is used to obtain the object. The script that leads to this can be seen in Listing 6.7. With that set, Ansible can loop through all the rows in the CSV and create the objects.

```

1  - name: Read users from CSV file and return a list
2    community.general.read_csv:
3      path: classes.csv
4      register: classes
5      delegate_to: localhost
6
7  - set_fact:
8      newClass: "{{ newClass | default([]) + [
9                    item | combine({
10                       'enrolledStudents': item.enrolledStudents.split
11                                     (' ','')
12                                     | map('regex_replace', '^(.*)$', '{\"spec
13                                     \":{\"id\": \"\\1\"}}')
14                                     | map('from_json')
15                                     | list
16                       }]) }}"
17  loop: "{{ classes.list }}"

```

Listing 6.7: CSV-Parsing

### *Extend Docker-Image*

To build the Docker image, the user needs to specify their base image as well as the name of the output image and the local user. The playbook then creates a new Dockerfile, which modifies the base image to properly work with the prototype. For this purpose, it first installs and configures the software needed for the prototype. Subsequently, it places the entry point script 6.4.1 inside the container, overwriting the entry point of the base image to use the entry point script. Once this is accomplished, the built image is pushed directly onto the registry. To build and deploy the image, the `docker_image` module is used.

## 6.4 Containerization

This section provides an overview of the implementation and challenges related to the three containers that have been created. First, the container used as an example for students' container is explained. Afterward, different containers for the web application and the Operator are discussed in more detail.

### 6.4.1 Student Container

As a base for the container, the official Ubuntu container was chosen. This is because for many students, Ubuntu is their first contact point to Linux, offering familiarity. Furthermore, Ubuntu has a support structure, which should ensure up-to-date base images without vulnerabilities. The following paragraph describes the workings of the Dockerfile that was created.

## Dockerfile

The Dockerfile starts by un-minimizing the container of Ubuntu to make it possible for users to log into, since the container is built for continuous integration (CI) usage normally [148]. Subsequently, locales and basic programs such as ping and VIM are installed, and bash completion is configured. Thereafter, the “inklog” module is disabled. This prevents rsyslog from capturing kernel logs, which would otherwise increase system calls and require additional rights [149]. Lastly, root login is allowed in the SSHD config and the entry point script serves as the command executed upon container startup.

## Entrypoint.sh

The script runs as root. Some of the script’s functions are explained in more detail in the following paragraphs.

### *NFS Mounting*

Since Persistent Volumes can only have a one-to-one relation with Persistent Volume Claims, and both are namespaced resources, it is not possible to make a class mount work for every student. To overcome this issue, a specific pattern for the creation of the NFS folders (/share/class/“CLASSNAME”) was used. This pattern is then being used by the Operator to create a Deployment, which mounts the class-folder directly per NFS with read-only access to every student within the class. The students’ private folder works without that caveat, since it is only used inside the student’s namespace. Therefore, the PVC is mounted normally.

### *User*

The container has 4 environment variables: username and password, root password and lastly whether the user should have sudo rights. The passwords are already encrypted by the Operator using the Go bcrypt-library before being passed into the container. Once the user is created, the skeleton directory is manually copied over and elevated rights are given if the environment variable is set. It is important to change the home folder’s rights to 755 and set the owner to the newly created user. This is necessary because, by default, the owner is set as root during the mounting process, as explained in paragraph 4.5.1.

### *SSH*

As described in the section dedicated to the application, an SSH key can be uploaded to facilitate accessing the machine. The website uploads the key inside a specific folder of the share. From there on, the entry point script checks whether an uploaded key can be found in the folder. If this is the case, this key then is copied to the .ssh folder under a special name, which is set in the sshd\_config. Thus, the .ssh folder contains the normal authorized\_users file, as well as a new file that contains only the key uploaded from the web application. For SSH to accept the

key, the rights need to be changed too. Having done this, the SSH key can operate effectively. It is important to state that this transfer only occurs within the script. Therefore, every time a new public key is uploaded, the current container stops and undergoes recreation.

### *SSH-Fingerprint*

Another challenge regarding SSH was that the SSH fingerprint changed every time the container was deleted and recreated. The known host file recognized this and prevented the connection, since SSH host keys should remain constant, and any alteration suggests a potential man-in-the-middle attack. To remedy this, the host key is copied into the mounted private user folder and kept there. The path of this key can be set inside the `sshd.config`. When the container is recreated, the file is not recopied if it already exists. As a result, the host key remains consistent upon container recreation, preventing any SSH fingerprint verification errors from occurring.

## **6.4.2 Web Application Container**

To release the app, the first step was to create an adapter, which defines how to build the webpage. Since this project is a containerized Node server, the Node adapter was chosen. This can be changed inside the `Svelte-config`. With `$ npm run build`, a folder is created that contains the built server and website, which can be containerized. [150]

### *Dockerfile and Docker-Compose*

The Dockerfile employs a multi-stage build. In the initial step, the project is copied into a container and subsequently built. In the following step, the built files are copied into the second container, which gets released. Furthermore, the customized `server.js` is employed to initiate the server. Further details about this process can be found in 6.4.2.

Considering that the web application requires two mounted folders— one for students and one for lecturers— a Docker-Compose configuration was created to provide an example of container utilization. The `compose-file` creates the container and mounts the two NFS volumes. Another volume is defined to mount the folder containing all certificates. Further details regarding the required certificates are provided in the following paragraph. The `compose file` also reads in all environment variables needed for the web-application to work, such as the Keycloak details or the Kubernetes server IP.

### *Auth.JS*

After building the app, `Auth.JS` fails to function correctly without SSL access. When deploying the Docker-Compose file, every request is rejected, since they come from an insecure source. By default, `SvelteKit` does not offer HTTPS access [151]. To solve this issue, a Custom Server [152] that accepts HTTPS requests was

written. The custom server itself is an Express.js app that listens both to HTTP and HTTPS. The HTTPS-routes fetches the certificates from the path defined in the environment. Once a request goes in, it will be sent to the SvelteKit Handler, which takes care of processing it. Creating a custom server has an additional advantage, as it offers to provide extra routes. In this case, a health check route was created, which supplies an okay when called. This can be used to allow Kubernetes to check whether the container is ready and healthy. In order to use the custom server, the latter is copied into the release folder post-building, and then employed to initiate the server.

The certs needed for HTTPS are supplied from the mount defined before. To generate the certificates, the command: `$ openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256 -days 3650 -nodes -subj "SUBJ"` can be used. After having done this, the server inside the Docker-Compose works as expected and integrates correctly with Keycloak and the cluster.

### *Deployment inside Kubernetes*

Inside the cluster, certificates are not needed, since Contour and Cert-Manager generate their own. Any incoming request can be then sent to the HTTP route inside the container. To deploy the app, first, a Namespace is created. Afterward, A ConfigMap with the environment variables is added to this Namespace.

Next, a Service is generated to enable a route to the Deployment, and a PVC is set up to provide storage for the certificates. The only certificate needed for this is the Kubernetes CA. Afterward, the Deployment 6.8 is created. It uses the ConfigMap to pass the variables into the container. Moreover, as visible in the listing, a readiness probe was included to check when the route is available to make calls. Lastly, the volumes are mounted through Kubernetes, which enables an easier exchange. To permit external access to the Deployment, an Ingress object was written, which allows calls coming from the browser to reach the website.

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    [...]
5  spec:
6    [...]
7    spec:
8      containers:
9        - name: kubelab-web-pod
10          image: floreitz/kubelab-web:latest
11          envFrom:
12            - configMapRef:
13              name: webapp-config
14          ports:
15            - name: http
16              containerPort: 80
17          readinessProbe:
18            httpGet:
19              path: /healthcheck
20              port: 80
21          volumeMounts:
22            - mountPath: /students
23              name: students

```

```

24     [...]
25     - mountPath: /certs
26       name: kubelab-certs
27   volumes:
28     [...]
29     - name: students
30       nfs:
31         server: 192.168.188.13
32         path: /srv/kubernetes/student
33     - name: kubelab-certs
34       persistentVolumeClaim:
35         claimName: kubelab-web-claim

```

Listing 6.8: Deployment for the web application

### 6.4.3 Operator Container

Kubebuilder added functions to directly deploy the Operator. Using `$ make docker-build docker-push IMG=<some-registry>/<project-name>:tag` the image is built and pushed to the docker registry. From there, it can be deployed with `$ make deploy IMG=<some-registry>/<project-name>:tag`. It automatically creates the Custom Resource and sets up a single Deployment with two Pods to initiate container execution. It is important to notice that the second Pod's purpose is not to distribute load; instead, it serves as a backup in case the first Pod fails. After this procedure, any Custom Resource of the type Classroom or Kubelab-User that is applied to the cluster will be processed by the Operator. [153]

To obtain a Manifest file, when using `$ make deploy`, the make file was edited. Under the hood, it uses Kustomize<sup>2</sup>. This tool manages Manifests, by combining them and defining patches based on the environment. After using the build command of Kustomize, it creates a large file, rather than the file folders created by Kubebuilder. Instead of the default behavior of directly piping the result of the build into the API, it was changed to write the file onto the disk. Following this action, the file can be shared to be used in any cluster.

#### *Apply vs. Create*

Upon the initial application of the file, Kubernetes did not acknowledge Kubelab-User as a valid CRD. The problem lied within the command that was used to add the resource to the cluster: `$ k apply`.

Kubernetes has two ways of adding resources to the cluster. On the one hand, “apply”, which is declarative. For example, if a small part suffers modifications and the Manifest is applied again, the edits would be recognized and incorporated. This way, Kubernetes saves the last applied Manifest inside the cluster and searches for differences if it is applied again. On the other hand, if the imperative command “create” is used, the defined resource cannot be changed by employing “create” once more. This would prompt Kubernetes to raise an error, indicating that the resource has already been created. In this context, Kubernetes does not save the

<sup>2</sup> <https://kustomize.io/>

Manifest inside the cluster. To implement the changes, the “replace” command can be employed. This command deletes the existing resource and subsequently recreates it using the provided Manifest.

Typically, “apply” is used as it offers more convenience. In this instance, the challenge arises from the operator’s Manifest being massive, and the “apply” command not effectively managing such extensive files. This problem is mentioned in Calico’s documentation and fixed using the create command, which has no size limitation. [154] After having used “create”, all resources were correctly created.

### *Rights*

After successfully deploying the Operator, nothing occurred when a CRD was added to the Cluster. By checking the Operator logs, it became evident that the Operator was not entitled to create Deployments. This error stemmed from a wrong API group that was set inside the Go-Code. This issue had never arisen before, even though the Operator had created multiple Deployments in the past. The reason behind it is that Kubebuilder uses the rights of the user defined in the kubeconfig. This results in an inconsistency between the development environment and the production environment. This issue was solved by adding the correct Go-code annotation. Subsequently, the container was rebuilt and it functioned correctly.

## Security and Performance

This chapter will discuss configurations that have been implemented as well as those that could be implemented to improve security and performance.

### 7.1 Security

#### *Application Security*

There are direct security features that can be used to further lock down the system directly from the codebase. Due to the structure of the web application, one involves modifying answers before passing them on to the clients. This shall be used to remove any information that is not necessary for the client. This includes, for example, the root password annotation from the Deployment.

Inside the controller, security can be improved using validation hooks. They can be employed to avoid wrong data being pushed to the Kubernetes API. In this context, hooks can ensure the uniqueness of the user IDs. Furthermore, they can ensure that the class object contains only students and lecturers that have already been created. This reduces the possibility of the Operator mishandling objects that rely on non-existent data. Regarding the API, it is also possible to configure the Server itself to minimize the attack vector.

#### *Server Security*

A potential measure, which has been mentioned earlier, is to lock down the Kubernetes API, in a way that only the web application's server-side rendering can access it. This lockdown ensures that students cannot abuse the system, for example, by scaling their Deployment up above one replica. This can be done by using network add-on features, which directly restrict Pod traffic. Furthermore, it is possible to restrict the server itself, by setting up a firewall that blocks all incoming requests against the Kubernetes port from unknown IPs.

Another attack vector is the host system itself. To reduce the risk of vulnerabilities that may affect container workloads, the host system should be kept up to date and as minimal as possible. There are some distributions that are specifically made for containers, for example Fedoras CoreOS<sup>1</sup>. Furthermore, no other

---

<sup>1</sup> <https://docs.fedoraproject.org/en-US/fedora-coreos/>

workloads should be run on the OS, as this helps to reduce the risk of external break-ins. Additionally, Kubernetes can be configured in several ways to lower the risk of attacks that stem from breaking out of the container instead of externally breaking in.

### *Kubernetes Security*

As stated by Xin Lin et al. [14], a crucial way of escaping containers' boundaries is through system calls. There are multiple ways to reduce this risk. One of them is employing gVisor, a software developed by Google, which creates a second kernel in the user space for certain file operations. It catches all system calls, and handles the ones it can directly in the user space. All other system calls are sent through to the real kernel. This approach reduces the risk of system call vulnerabilities, nonetheless it adds a performance penalty, as tested by Viktorsson et al. [17]. To use the system, it needs to be installed and the container runtime that is supplied with the gVisor core needs to be used to run workloads. This behavior can be defined inside CRI-O. Furthermore, Kubernetes supports the creation of runtime classes, where the runtime for every workload can be set. This makes it possible to restrict student containers, while still allowing other containers, such as the web application, to run performant.

Another way to restrict system calls is seccomp, which outright blocks predefined system calls. To find out which system calls are needed and which ones can be blocked, seccomp can be configured to log all activity. For this project, this would mean recording all needed system calls for either one semester or while someone does all the tasks of the module. Once the profile has been created, it can be applied to a class and block all system calls that are not needed. An issue can arise if students adopt new workflows that need different syscalls than the ones the current profile allows.

To restrict groups of system calls or, more generally, to drop certain permissions completely within a container, capabilities can be used. They can be used to drop or give certain rights to processes. Broadly, all capabilities that are not needed are dropped by default and need to be added manually again. A few of them are directly added by the container runtime to function correctly, yet they differ from vendor to vendor. For example, the `NET_RAW` capability, required for commands such as `$ ping` is included directly by Docker, while CRI-O does not need it to work.

In this project, three capabilities that are not directly given by CRI-O were required for the container to function. First, to ensure proper use of SSH, the `SYS_CHROOT` capability is needed, as it allows setting a new root folder inside the container. SSH uses this to lock users to certain directories. Secondly, `AUDIT_WRITE` was required for SSH as well, since SSH needs to log who connects to the system. Lastly, `NET_RAW` is used, as explained above, to allow users to employ ping. All these three capabilities are included in the Docker runtime by default. Therefore, with a different runtime, it may not be necessary to manually add them to the Deployment.



Another way of restricting containers inside Kubernetes are LSMs, specifically SELinux and AppArmor. They enable the definition of rights for files and applications, allowing for more granular restrictions, when compared to the default DAC system. For example, it is possible to restrict file access and network resource access. Both systems use profiles that can be applied to the container. To observe which restrictions can be made, similar to seccomp, there is a mode in which all access is logged.

In relation to networking, the implementation of Network Policies within Kubernetes can enhance security by restricting the internal network of Kubernetes. It is important to state that not every Kubernetes network add-on supports this functionality. One example of usage is built into the prototype. The exam mode restricts all traffic in and out of the container, except for the SSH connection. It does this by disallowing all calls that go outside the container (Egress-traffic) and solely allowing connection through the SSH port (Ingress-traffic).

Another way to decrease the risk of container escapes is the PACED system presented by Abbas et al. [18]. The results of this paper showcase a near perfect accuracy for detecting container escapes. To validate these statements, the system may, due to its Open-Source nature, be deployed and reviewed.

Lastly, if the aforementioned measures fail, it is important that a student possess only the bare minimum rights on the host system. To accomplish this, the user namespace can be employed. This namespace can map the users inside the container, including the root user, which may be needed for certain classes, to a non-root UID on the host system. Even if a student manages to break out, this contributes to limiting the effect of it. For Kubernetes, this function has only been launched in an experimental state [155]. CRI-O, the engine that is used for the prototype, already supports this functionality. Its usage is highly recommended, since it bares no downside in terms of performance or functionality inside the container. Having configured Kubernetes, an additional potential threat is the container itself.

### *Container Security*

Shu et al. [15] and Wist et al. [16] have shown that there are numerous vulnerabilities inside the images that are uploaded to container registries. Moreover, the current ecosystem where software comprises packages from various developers, for example NPM<sup>2</sup>, facilitates the introduction of vulnerabilities.

Having this in mind, a vulnerability scanner could be included in the workflow when building the container image. This should reduce the risk of container escapes through vulnerabilities. Furthermore, a CI that regularly rebuilds the container with the most recent security updates should be created. Many vendors allow users to continue to use an older version and still receive security hotfixes, at least for a certain period. This should make the containers safe to deploy for students.

---

<sup>2</sup> <https://www.npmjs.com>

## 7.2 Performance

While no previous performance measurements were carried out during this work, steps have been taken to improve the project's scalability and efficiency. This section takes a look at the measures taken for this purpose, as well as related performance testing of containers.

An important improvement, as described in the implementation chapter 6.1.3, is resource indexing. This allows the Operator to work with cached entries instead of burdening the API with large requests.

To restrict resources, the Kubernetes Limits system is used. This system relies on cgroups to restrict resources, such as RAM and CPU power. This ensures that every student has a set limit of resources, which the server can handle. It is also possible to scale horizontally by adding servers without the need for major configuration changes. The kube-scheduler will verify that all resources are split between the Nodes, meaning that the new servers are directly employed for workloads. The web application can also be scaled up and down as needed, since Kubernetes' internal load balancer can send incoming requests to any Pod of the web application Deployment.

To optimize resource usage, the Deployments can be terminated when no SSH connection persists. While the process is not as simple as adding a Liveness-probe, as described in the implementation, it is possible to accomplish this behavior within Kubernetes.

Generally, containers achieve a much lower footprint compared to virtual machines, as demonstrated by Podar et al. [20], Joy et al. [21] and Sharma et al. [22]. However, using a virtual laboratory does not only burden the server, but also the network connection. Iorio et al. [4] have tested their solution, which uses virtual machines with a VNC client. They came to the conclusion that two thirds of the time the bandwidth consumption ran under 200 Kbps<sup>3</sup> and was spiking up 20% of the time above one Mbps<sup>4</sup>. They calculated an average of 1 Mbps for the usage of the virtual desktop. The highest throughput was required to render full-HD videos from the desktop at up to 60 Mbps. Similar throughput can be expected for this project if a virtual desktop is required. This prototype is based on the shell and requires less bandwidth, as it only needs to transfer characters. Using verbose commands, such as `$ apt update && apt upgrade`, spiked the used bandwidth to 10 Kbps. In general, when employing standard read and write operations through SSH, the average bandwidth usage measured was at around 3 Kbps. These results show, that, when not in need of a virtual desktop, the network bandwidth required for this project can be neglected.

---

<sup>3</sup> kilobit per second

<sup>4</sup> megabit per second

## Results and Discussion

This chapter will discuss the results of this work. It will begin by comparing the finished prototype to the requirement list 4.1.

The prototype shows that it is possible to build a remote computer laboratory relying on Kubernetes. Students, as well as lecturers, may manage their containers, as can be seen in Figure 8.1 and 8.2. Furthermore, an account management system (Req. S1) with SSO integration (C1) can be provided through Keycloak. This system can be substituted by the SSO an educational institution may possess. Furthermore, student and class shares, that are used for requirements S2 and S3 work as anticipated in the design part. Generating a string that can be directly pasted into the terminal to establish a container connection is effective and satisfies S4, the SSH access requirement. It is also possible to, as a student, upload an SSH key to use passwordless authentication (C2).



Fig. 8.1: Teacher's user interface

Other functions, such as the exam mode (S5) and the creation and deletion of classes, can currently only be performed by interacting with the cluster directly or through an intermediary, such as Ansible, that simplifies the resource creation process (C6). Requirement S6, which states the need for servers that every student in the class can connect to, is feasible, yet it demands manual creation of the resources in Kubernetes. In order to achieve this, a Deployment featuring the server

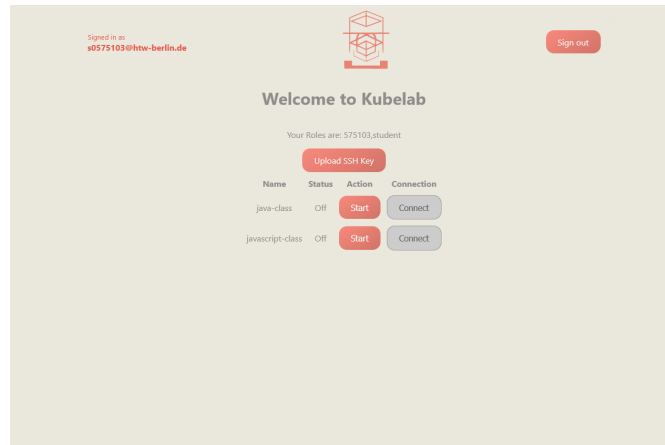


Fig. 8.2: Student's user interface

image should be created. Thereafter, based on the Labels of students' Deployments, only Ingress traffic should be allowed by writing a Network Policy.

Both of these functions can be included into the web application at a later stage to simplify the process for lecturers.

Explicit testing of GPU scheduling (C4) was hindered by hardware limitations. Nonetheless, its feasibility is plausible, given that Kubernetes does offer the functionality to schedule a container only if a GPU is available. GUI applications (C3) are possible too. During the tests, X11 was used to display simple interfaces through SSH. This approach presents some drawbacks, since X11 appears sluggish through the shell. Further research could explore the utilization of either Wayland, the successor to X11, or the implementation of a full-fledged VNC client. Iorio et al. [4] describes, for this purpose, the possibility to run a sidecar container with a VNC client.

An area open for further expansion within the prototype is the SSH connection itself. In this project, NodePorts were used to handle port assignment throughout the cluster. In the future, implementing a system where each student is allocated a designated URL, accessible through port 22 instead of relying on a random port assignment would be advantageous. This has not been done, owing to the fact that the Ingress object only supports HTTP/S. There are vendor-specific Custom Resources that combat this problem, however, Kubernetes itself does not yet have an official object that allows Ingress for other types of traffic.

While the prototype currently only supports Linux distributions, the server structure can be extended to include Windows Nodes, which enables the utilization of Windows containers. Once a Windows Node is added to the cluster, annotations can be made to the Deployment, only permitting it to schedule on Windows systems. Nevertheless, before using them, further research addressing security implications of Windows containers is encouraged. Other operating systems, such as macOS, currently do not support Kubernetes.

When the details of containers mentioned in the fundamentals section are examined, it becomes evident that they are not inherently unsafe. The risk that

the namespace isolation is broken still exists. For example, in 2022 a flaw in the cgroup system was discovered [156], which made it possible to break out of the namespaces. Nonetheless, after the introduction of the user namespace, it can be assumed that these threats will have a lower impact.

Kubernetes itself has some problems, which may be addressed in later versions. The RBAC system is not sufficiently granular for this purpose, which poses a challenge. Restricting teachers to only their resources without the use of Labels makes the process convoluted. It is possible to restrict only using the names as identifiers, but that would lead to a multitude of files for every student of the class, rendering administration much more difficult.

Another aspect that requires consideration is, in general, the technology's early stage. Kubernetes itself is evolving, and the feature set is expanding. For this reason, some of this work's relevant features, such as cross namespace storage and the user namespace, are still in alpha. The difficulties arising from new technology also become evident in the general ecosystem. As mentioned in the implementation section, two of the add-ons that were used presented severe bugs, which hindered the prototype's operation. The bug in the network add-on even crashed garbage collection for the whole cluster, impeding the cluster's ability to delete resources correctly. These issues can be negated by having a staging cluster, in which updates can be tested before going into production. On the negative side, this demands a lot more resources, human as well as computational.

Furthermore, due to its Open-Source nature, Kubernetes has no licensing fees and supplies a management interface for the prototype, without presenting the need to write a complete new software. Moreover, its continued development ensures patches with fixes and the availability of new features.

As described in the previous chapter, container security is a topic that can be extensively discussed. All in all, measures to minimize the attack vector must be implemented, such as setting the least number of privileges and blocking unnecessary system calls. Furthermore, LSMs can be configured to operate similarly to an intrusion detection system, writing down every attempted access to critical files (such as the clusters configurations files or general files of the host system), while also blocking access outright.

The web application, as well as the cluster and the control plane, can be scaled to accommodate any number of users. The only non-scalable part of the system is the Operator, as it has, at this stage, only one instance managing the state of the Custom Resources as described in the best practices of the Operator Framework [157]. The rationale for this is that it can result in complications regarding the state of a Custom Resource. However, adjustments can still be made to lighten the load on the Operator, such as splitting it into two halves and making one Operator per Custom Resource Definition. The reason this was done differently in this case was the extended development time to create and manage two different Operators.

Further testing regarding the two Operators and CRDs's is another avenue for further research. If this approach is not successful with larger workloads, an alternative would be using Aggregated APIs instead, which in turn require more

administration and development effort. Throughout this project, the creation, and placement of up to 50 users into classes did not pose any problems.

Finally, due to their lightweight nature, containers considerably save on costs, especially compared to bare-metal machines. The automatic provision of resources can also be used to more efficiently manage loads. It is conceivable that when the system is not fully loaded, students' containers could use extra CPU and RAM resources. However, these resources might be reduced if other students also start their containers. While virtual machines can at present, especially for RAM usage, scale as well, they are not as flexible as containers. Furthermore, while the separate OS kernel gives the system stronger isolation, it also creates more overhead, especially for storage. There exist other solutions, such as LXC containers and Amazons Firecracker, which strike a middle ground between containers and virtual machines and could thus be further investigated.

## Conclusion

This research aimed to examine the usages and advantages of containers for computer science laboratories. Drawing from the developed prototype, it can be affirmed that containers have the potential to grant students access to computational resources, regardless of whether they are within or outside the institution.

The developed prototype fulfills all the main requirements set at the beginning of this work, allowing, on the one hand, lecturers to manage their classes by uploading files and accessing their students' containers. Students, on the other hand, can connect to their containers, either by using a password or via SSH key. An advantage of this approach is the possibility to connect multiple users to one container, which facilitates the grading processes. Furthermore, although some optional requirements could not be implemented due to several reasons stated earlier in this work, this contribution does discuss options and ideas to solve those issues and fulfill those additional requirements.

Containers offer advantages over other solutions, as they allow a more flexible resource management and have a smaller footprint. One drawback worth noting is their considerably weaker form of isolation when compared to virtual machines. Nonetheless, to counterbalance this, it is possible to add a multitude of restrictions, for example with the use of LSMs and similar tools.

Similar approaches have already shown that Kubernetes can be leveraged for more functionality, especially a study conducted by Iorio et al. [4]. This thesis comes to a similar result, while also adding a few caveats. In this work, Kubernetes is being leveraged in a multitude of ways, for example, it supplies functionalities that facilitate management, such as a data store in the form of the Custom Resources. It also aids with abilities that can be employed to create a more flexible learning experience, such as GPU scheduling and Windows Nodes, hence removing the need to find a computer laboratory that suits a class' specific requirements. GPU scheduling, for instance, may be particularly important for game development and artificial intelligence classes, which demand additional computing power.

Furthermore, Kubernetes' automatic storage allocation and its ability to restrict all network traffic on a granular level helps to establish a secure and automated system. Additionally, the creation of more elaborate environments can be achieved through the employment of Network Policies, combined with containerized servers that students may use. For example, a cybersecurity class may host a vulnerable

container, restricted to be accessed only by the class for learning purposes. A weekly task could involve obtaining a flag from the aforementioned system, while tracking the time taken by each student.

One challenge this work encountered is that some features relevant to this work, especially the user namespace, are not yet fully supported. Overall, the ecosystem in some regards appears to be in its infancy, as can be judged based on the bugs encountered in Section 6.1.3.

Comparable observations can be made regarding security aspects. In particular, system calls are often used to break isolation. In this regard, the current research identified a multitude of approaches to achieve more secure environments. Nonetheless, many of the projects this work mentions, such as gVisor, are still undergoing development and expansion. The recent rise of security incidents in the industry has extended the need for more and better security tools, which may translate into better container tooling as well.

Another contentious aspect is that the prototype requires a lot of administrative work to function properly. Therefore, lecturers' workload would initially rise because of the necessity to create deployable containers and redesign parts of their courses. Its utilization would also increase the need for trained IT personnel, either to deploy and restrict common servers manually, or to develop further solutions that include these functionalities in interfaces for the lecturers. In addition to that, large-scale tests to verify its correct functioning would become necessary.

The Kubernetes concept of Pods, which provides opportunities to incorporate additional containers on the localhost (such as interchangeable databases based on the ongoing task) may be a fruitful area for further research. In this context, even the SSH component may be delegated into a sidecar container, making the system more modular. Furthermore, while it is possible to run desktop applications inside the Pod through X11, more research is needed into the performance and other possible approaches to deliver a proper desktop experience. While easy access to the students' work is possible for the lecturer, further automation could be introduced by researching the possibilities of automatic evaluation, either through connecting and running code analysis tools, or by hosting applications on open ports, which can be automatically evaluated. Further research efforts could be directed towards determining whether these considerations hold value.

The question: *How can containers be of use in the development of computer laboratories?* was addressed in this work by creating a prototype that demonstrates the feasibility and advantages of container-based remote laboratories. However, certain concerns persist that demand additional testing and assessment. Notwithstanding these limitations, the results of this research support the idea that containers managed by Kubernetes as an orchestrator are a suitable backbone for creating laboratories in an educational environment and an endeavor worth following.

The pandemic showed the need for more flexible laboratories. This study aspires to have contributed useful ideas concerning the design and development of such systems, while also highlighting directions for future research and enhancement.



---

## References

1. Docker Inc., “What is a container? — docker,” *Docker*, 11/11/2021. [Online]. Available: <https://www.docker.com/resources/what-container/> [Accessed: 7/7/2023]
2. C. Irvine, M. Thompson, and J. Khoslim, “Labtainers: A framework for parameterized cybersecurity labs using containers,” *National Cyber Summit*, 2017. [Online]. Available: <https://louis.uah.edu/cyber-summit/ncs2017/ncs2017papers/5> [Accessed: 4/27/2023]
3. S. Thiruchadai Pandeewari, S. Padmavathi, M. Sanjaybabu, S. S. Srilakshmi, and K. Sabari Priya, “Container-based lab-as-a-service application,” in *Internet of Things and Its Applications*, Operator SDK, Ed. Springer, Singapore, 2022, pp. 133–143. [Online]. Available: [https://link.springer.com/chapter/10.1007/978-981-16-7637-6\\_13](https://link.springer.com/chapter/10.1007/978-981-16-7637-6_13) [Accessed: 4/27/2023]
4. M. Iorio, A. Palesandro, and F. Risso, “Crownlabs—a collaborative environment to deliver remote computing laboratories,” *2169-3536*, vol. 8, pp. 126 428–126 442, 2020. [Online]. Available: <https://ieeexplore.ieee.org/document/9136697> [Accessed: 4/27/2023]
5. G. Chen, “Cvllabs: A container based interactive virtual lab for it education.” [Online]. Available: <https://smartech.gatech.edu/handle/1853/64898> [Accessed: 4/27/2023]
6. Cloud Native Computing Foundation, “Cncf annual survey 2022 — cloud native computing foundation,” 2023. [Online]. Available: <https://www.cncf.io/reports/cncf-annual-survey-2022/> [Accessed: 7/11/2023]
7. Open Container Initiative, “About the open container initiative - open container initiative,” 7/21/2023. [Online]. Available: <https://opencontainers.org/about/overview/> [Accessed: 7/21/2023]
8. Cloud Native Computing Foundation, “Services for cncf projects — cloud native computing foundation,” 7/21/2023. [Online]. Available: <https://www.cncf.io/services-for-projects/#introduction> [Accessed: 7/21/2023]
9. Kubernetes, “Production-grade container orchestration,” 8/14/2023. [Online]. Available: <https://kubernetes.io/> [Accessed: 8/14/2023]
10. L. Tobarra, A. Robles-Gómez, R. Pastor, R. Hernández, A. Duque, and J. Cano, “Students’ acceptance and tracking of a new container-based virtual laboratory,” *Applied Sciences*, vol. 10, no. 3, p. 1091,

2020. [Online]. Available: <https://www.mdpi.com/2076-3417/10/3/1091> [Accessed: 4/27/2023]
11. S. Strickroth, D. Bußler, and U. Lucke, "Container-based dynamic infrastructure for education on-demand," *1617-5468*, 2021. [Online]. Available: <https://dl.gi.de/handle/20.500.12116/37012> [Accessed: 4/27/2023]
  12. Laura Baldwin, "Leveraging katacoda technology - o'reilly media," 2022. [Online]. Available: <https://www.oreilly.com/online-learning/leveraging-katacoda-technology.html> [Accessed: 4/27/2023]
  13. Statista, "Cyber crime: reported damage to the ic3 2022 — statista," 7/11/2023. [Online]. Available: <https://www.statista.com/statistics/267132/total-damage-caused-by-by-cyber-crime-in-the-us/> [Accessed: 7/11/2023]
  14. X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, "A measurement study on linux container security," in *Proceedings of the 34th Annual Computer Security Applications Conference*. New York, NY, USA: ACM, 2018, pp. 418–429. [Online]. Available: <https://csis.gmu.edu/ksun/publications/container-acsa18.pdf> [Accessed: 08/25/2023]
  15. R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Record 1379410797 Unavailable*, G.-J. Ahn, A. Pretschner, and G. Ghinita, Eds. New York, NY, USA: ACM, 03222017, pp. 269–280. [Online]. Available: <http://dance.csc.ncsu.edu/papers/codaspy17.pdf> [Accessed: 7/11/2023]
  16. K. Wist, M. Helsen, and D. Gligoroski, "Vulnerability analysis of 2500 docker hub images," *arXiv: Cryptography and Security*, pp. 307–327, 2020. [Online]. Available: <https://typeset.io/papers/vulnerability-analysis-of-2500-docker-hub-images-3z4ezyud5j> [Accessed: 7/11/2023]
  17. W. Viktorsson, C. Klein, and J. Tordsson, "Security-performance trade-offs of kubernetes container runtimes," *Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, vol. 2020, pp. 1–4, 2020. [Online]. Available: <https://typeset.io/papers/security-performance-trade-offs-of-kubernetes-container-4u5c6erxby> [Accessed: 7/11/2023]
  18. Mashal Abbas, Shahpar Khan, Abdul Monum, Fareed Zaffar, Rashid Tahir, D. Eysers, Hassaan Irshad, A. Gehani, V. Yegneswaran, and Thomas Pasquier, "Paced: Provenance-based automated container escape detection," *2022 IEEE International Conference on Cloud Engineering (IC2E)*, 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/9946350> [Accessed: 7/11/2023]
  19. S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *2169-3536*, vol. 7, pp. 52 976–52 996, 2019. [Online]. Available: <https://ieeexplore.ieee.org/document/8693491> [Accessed: 08/25/2023]
  20. A. M. Potdar, N. D. G., S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050920311315> [Accessed: 24/7/2023]

21. A. M. Joy, “Performance comparison between linux containers and virtual machines,” in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 342–346. [Online]. Available: <https://ieeexplore.ieee.org/document/7164727> [Accessed: 08/25/2023]
22. P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, “Containers and virtual machines at scale,” in *Proceedings of the 17th International Middleware Conference*. New York, NY: ACM, 2016, pp. 1–13. [Online]. Available: [https://people.cs.umass.edu/~prateeks/papers/mw\\_submitted.pdf](https://people.cs.umass.edu/~prateeks/papers/mw_submitted.pdf) [Accessed: 08/25/2023]
23. Sebastien Godard and others, “unshare(2) - linux manual page,” 6/24/2023. [Online]. Available: <https://man7.org/linux/man-pages/man2/unshare.2.html> [Accessed: 7/4/2023]
24. L. Rice, *Container security: Fundamental technology concepts that protect containerized applications* / Liz Rice, 1st ed. Sebastopol, CA: O’Reilly Media, 2020.
25. Sebastien Godard and others, “ps(1) - linux manual page,” 6/24/2023. [Online]. Available: <https://man7.org/linux/man-pages/man1/ps.1.html> [Accessed: 7/4/2023]
26. —, “network\_namespaces(7) - linux manual page,” 6/24/2023. [Online]. Available: [https://man7.org/linux/man-pages/man7/network\\_namespaces.7.html](https://man7.org/linux/man-pages/man7/network_namespaces.7.html) [Accessed: 7/4/2023]
27. —, “ipc\_namespaces(7) - linux manual page,” 6/24/2023. [Online]. Available: [https://man7.org/linux/man-pages/man7/ipc\\_namespaces.7.html](https://man7.org/linux/man-pages/man7/ipc_namespaces.7.html) [Accessed: 7/7/2023]
28. —, “cgroup\_namespaces(7) - linux manual page,” 6/24/2023. [Online]. Available: [https://man7.org/linux/man-pages/man7/cgroup\\_namespaces.7.html](https://man7.org/linux/man-pages/man7/cgroup_namespaces.7.html) [Accessed: 7/7/2023]
29. Michael Kerrisk, “Namespaces in operation, part 6: more on user namespaces [lwn.net],” 2/13/2019. [Online]. Available: <https://lwn.net/Articles/540087/> [Accessed: 7/7/2023]
30. Sebastien Godard and others, “time\_namespaces(7) - linux manual page,” 6/24/2023. [Online]. Available: [https://man7.org/linux/man-pages/man7/time\\_namespaces.7.html](https://man7.org/linux/man-pages/man7/time_namespaces.7.html) [Accessed: 7/7/2023]
31. RedHat, “Using the cri-o container engine — cri-o runtime — openshift container platform 3.11,” 2023. [Online]. Available: [https://docs.openshift.com/container-platform/3.11/crio/crio\\_runtime.html](https://docs.openshift.com/container-platform/3.11/crio/crio_runtime.html) [Accessed: 3/7/2023]
32. S. J. Bigelow, “A breakdown of container runtimes for kubernetes and docker,” *TechTarget*, 10/11/2021. [Online]. Available: <https://www.techtarget.com/searchitoperations/tip/A-breakdown-of-container-runtimes-for-Kubernetes-and-Docker> [Accessed: 3/7/2023]
33. cri-o team, “cri-o,” 11/28/2022. [Online]. Available: <https://cri-o.io/> [Accessed: 5/1/2023]
34. The kernel development community, “Seccomp bpf (secure computing with filters) — the linux kernel documentation,” 10/22/2018. [Online]. Available:

- [https://www.kernel.org/doc/html/v4.19/userspace-api/seccomp\\_filter.html](https://www.kernel.org/doc/html/v4.19/userspace-api/seccomp_filter.html) [Accessed: 7/10/2023]
35. R. Gagliardi, “Container mit syscalls-filters unter kontrolle,” 7/10/2023. [Online]. Available: <https://www.scip.ch/?labs.20220203> [Accessed: 7/10/2023]
  36. Sebastien Godard and others, “capabilities(7) - linux manual page,” 6/24/2023. [Online]. Available: <https://man7.org/linux/man-pages/man7/capabilities.7.html> [Accessed: 7/10/2023]
  37. Docker Documentation, “Docker security,” 7/18/2023. [Online]. Available: <https://docs.docker.com/engine/security/#linux-kernel-capabilities> [Accessed: 7/18/2023]
  38. Chris Wright, Crispin Cowan, James Morris, Stephen Smalley, and Greg Kroah-Hartman, “Linux security modules: General security support for the linux kernel,” 2002, 2002. [Online]. Available: <https://www.usenix.org/legacy/event/sec02/wright.html> [Accessed: 7/10/2023]
  39. The kernel development community, “Linux security module usage — the linux kernel documentation,” 4/1/2018. [Online]. Available: <https://www.kernel.org/doc/html/v4.16/admin-guide/LSM/index.html> [Accessed: 7/10/2023]
  40. Steve Beattie, “Gettingstarted · wiki · apparmor / apparmor · gitlab,” 7/17/2023. [Online]. Available: <https://gitlab.com/apparmor/apparmor/-/wikis/GettingStarted> [Accessed: 7/17/2023]
  41. R. Hertzog and R. Mas, *The Debian administrator’s handbook: Debian 11 : Debian Bullseye from discovery to mastery*, 1st ed. [Sorbiers, France]: Freexian SARL, 2021. [Online]. Available: <https://debian-handbook.info/browse/stable/> [Accessed: 08/25/2023]
  42. Gentoo Authors, “Selinux/quick introduction - gentoo wiki,” 7/17/2023. [Online]. Available: [https://wiki.gentoo.org/wiki/SELinux/Quick\\_introduction](https://wiki.gentoo.org/wiki/SELinux/Quick_introduction) [Accessed: 7/17/2023]
  43. Red Hat, Inc., “What is selinux?” 7/17/2023. [Online]. Available: <https://www.redhat.com/en/topics/linux/what-is-selinux#how-does-it-work> [Accessed: 7/17/2023]
  44. Google, “What is gvisor?” 13.07.2023. [Online]. Available: <https://gvisor.dev/docs/> [Accessed: 13.07.2023]
  45. Kubernetes, “Operating etcd clusters for kubernetes,” 1/11/2023. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/> [Accessed: 4/10/2023]
  46. D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC’14. USA: USENIX Association, 2014, pp. 305–320. [Online]. Available: <https://raft.github.io/raft.pdf> [Accessed: 08/25/2023]
  47. Kubernetes, “The kubernetes api,” 10/24/2022. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/kubernetes-api/> [Accessed: 4/10/2023]

48. —, “Kubernetes scheduler,” 12/6/2022. [Online]. Available: <https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/> [Accessed: 4/14/2023]
49. —, “kubelet,” 12/23/2022. [Online]. Available: <https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/> [Accessed: 4/10/2023]
50. —, “Kubernetes components,” 10/24/2022. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/components/#kube-controller-manager> [Accessed: 4/14/2023]
51. —, “Nodes,” 3/31/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/architecture/nodes/#node-controller> [Accessed: 4/14/2023]
52. —, “Using coredns for service discovery,” 1/11/2023. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/coredns/#upgrading-an-existing-cluster-with-kubeadm> [Accessed: 4/14/2023]
53. —, “Kubernetes components,” 10/24/2022. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/components/#kube-proxy> [Accessed: 4/14/2023]
54. —, “Command line tool (kubectl),” 4/14/2023. [Online]. Available: <https://kubernetes.io/docs/reference/kubectl/> [Accessed: 4/14/2023]
55. —, “kubectl cheat sheet,” 3/30/2023. [Online]. Available: <https://kubernetes.io/docs/reference/kubectl/cheatsheet/> [Accessed: 4/14/2023]
56. —, “Nodes,” 4/1/2021. [Online]. Available: <https://kubernetes.io/docs/concepts/architecture/nodes/> [Accessed: 8/25/2023]
57. —, “Namespaces,” 29.06.2023. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/> [Accessed: 10.08.2023]
58. —, “Pods,” 17.08.2023. [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/pods/> [Accessed: 17.08.2023]
59. —, “Labels and selectors,” 2/19/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#set-based-requirement> [Accessed: 5/1/2023]
60. —, “Replicaset,” 2/8/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/> [Accessed: 5/1/2023]
61. —, “Deployments,” 2/18/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#pausing-and-resuming-a-deployment> [Accessed: 5/1/2023]
62. —, “Persistent volumes,” 3/7/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/> [Accessed: 6/9/2023]
63. —, “Storage classes,” 5/24/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/storage-classes/> [Accessed: 6/9/2023]
64. —, “Resource management for pods and containers,” 3/11/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/> [Accessed: 5/1/2023]
65. —, “Configure liveness, readiness and startup probes,” 4/4/2023. [Online]. Available: <https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/> [Accessed: 6/13/2023]

66. C. Collins, “An introduction to kubernetes secrets and configmaps,” 6/20/2023. [Online]. Available: <https://opensource.com/article/19/6/introduction-kubernetes-secrets-and-configmaps> [Accessed: 6/20/2023]
67. Kubernetes, “Service,” 3/22/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/services-networking/service/> [Accessed: 4/14/2023]
68. —, “Ingress,” 8/1/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/services-networking/ingress/> [Accessed: 8/25/2023]
69. Envoy, “What is envoy — envoy 1.27.0-7bba38 documentation,” 8/25/2023. [Online]. Available: [https://www.envoyproxy.io/docs/envoy/v1.27.0/intro/what\\_is\\_envoy](https://www.envoyproxy.io/docs/envoy/v1.27.0/intro/what_is_envoy) [Accessed: 8/25/2023]
70. cert manager, “cert-manager,” 8/29/2023. [Online]. Available: <https://cert-manager.io/docs/> [Accessed: 8/29/2023]
71. Kubernetes, “Taints and tolerations,” 6/29/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/> [Accessed: 7/24/2023]
72. —, “Configure a security context for a pod or container,” 3/15/2023. [Online]. Available: <https://kubernetes.io/docs/tasks/configure-pod-container/security-context/> [Accessed: 7/14/2023]
73. —, “Restrict a container’s access to resources with apparmor,” 5/16/2023. [Online]. Available: <https://kubernetes.io/docs/tutorials/security/apparmor/#objectives> [Accessed: 7/14/2023]
74. —, “Restrict a container’s syscalls with seccomp,” 3/10/2023. [Online]. Available: <https://kubernetes.io/docs/tutorials/security/seccomp/> [Accessed: 7/14/2023]
75. —, “Pod security admission,” 11/5/2022. [Online]. Available: <https://kubernetes.io/docs/concepts/security/pod-security-admission/> [Accessed: 7/14/2023]
76. —, “Pod security standards,” 4/29/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/security/pod-security-standards/> [Accessed: 7/14/2023]
77. —, “Runtime class,” 29.10.2022. [Online]. Available: <https://kubernetes.io/docs/concepts/containers/runtime-class/> [Accessed: 13.07.2023]
78. GitHub, “cri-o/docs/crio.conf.5.md at main · cri-o/cri-o,” 13.07.2023. [Online]. Available: <https://github.com/cri-o/cri-o/blob/main/docs/crio.conf.5.md> [Accessed: 13.07.2023]
79. Shihang Zhang, “enhancements/keps/sig-auth/2799-reduction-of-secret-based-service-account-token at master · kubernetes/enhancements,” 5/1/2023. [Online]. Available: <https://github.com/kubernetes/enhancements/tree/master/keps/sig-auth/2799-reduction-of-secret-based-service-account-token> [Accessed: 5/1/2023]
80. —, “enhancements/readme.md at master · kubernetes/enhancements,” 5/1/2023. [Online]. Available: <https://github.com/kubernetes/enhancements/blob/master/keps/sig-auth/1205-bound-service-account-tokens/README.md> [Accessed: 5/1/2023]

81. Kubernetes, “Service accounts,” 3/28/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/security/service-accounts/> [Accessed: 5/1/2023]
82. —, “Authenticating,” 3/5/2023. [Online]. Available: <https://kubernetes.io/docs/reference/access-authn-authz/authentication/#openid-connect-tokens> [Accessed: 5/1/2023]
83. —, “Authorization overview,” 5/5/2022. [Online]. Available: <https://kubernetes.io/docs/reference/access-authn-authz/authorization/> [Accessed: 5/1/2023]
84. Bruno Oliveira, “New keycloak maintainer: Alexander schwartz - keycloak,” 7/18/2023. [Online]. Available: <https://www.keycloak.org/2023/06/alexander-schwartz> [Accessed: 7/21/2023]
85. Cloudflare, “What is sso? — how single sign-on works — cloudflare,” 5/18/2023. [Online]. Available: <https://www.cloudflare.com/learning/access-management/what-is-sso/> [Accessed: 5/18/2023]
86. Information Security Stack Exchange, “authentication - difference between oauth, openid and openid connect in very simple term? - information security stack exchange,” 5/18/2023. [Online]. Available: <https://security.stackexchange.com/questions/44611/difference-between-oauth-openid-and-openid-connect-in-very-simple-term/44614#44614> [Accessed: 5/18/2023]
87. OpenID - The Internet Identity Layer, “Openid connect — openid,” 2011. [Online]. Available: <https://openid.net/connect/> [Accessed: 5/18/2023]
88. Kubernetes, “Network policies,” 5/15/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/services-networking/network-policies/> [Accessed: 7/14/2023]
89. Calico, “Get started with calico network policy — calico documentation,” 8/29/2023. [Online]. Available: <https://docs.tigera.io/calico/latest/network-policy/get-started/calico-policy/calico-network-policy> [Accessed: 8/29/2023]
90. Kubernetes, “Custom resources,” 10/8/2022. [Online]. Available: <https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/> [Accessed: 5/2/2023]
91. P. Wittrock, “What is a resource · the kubebuilder book,” 5/9/2023. [Online]. Available: [https://book-v1.book.kubebuilder.io/basics/what\\_is\\_a\\_resource.html](https://book-v1.book.kubebuilder.io/basics/what_is_a_resource.html) [Accessed: 5/9/2023]
92. Kubernetes, “Controllers,” 10/24/2022. [Online]. Available: <https://kubernetes.io/docs/concepts/architecture/controller/> [Accessed: 5/2/2023]
93. —, “Operator pattern,” 1/16/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/extend-kubernetes/operator/> [Accessed: 5/2/2023]
94. NCF TAG App-Delivery Operator Working, “Cncf operator,” 2021. [Online]. Available: [https://www.cncf.io/wp-content/uploads/2021/07/CNCF\\_Operator\\_WhitePaper.pdf](https://www.cncf.io/wp-content/uploads/2021/07/CNCF_Operator_WhitePaper.pdf) [Accessed: 5/2/2023]
95. Go, “Frequently asked questions (faq) - the go programming language,” 5/11/2023. [Online]. Available: <https://go.dev/doc/faq#go-or-golang> [Accessed: 5/11/2023]

96. —, “A tour of go,” 5/11/2023. [Online]. Available: <https://go.dev/tour/welcome/1> [Accessed: 5/11/2023]
97. Kubebuilder, “Controllers and reconciliation - the cluster api book,” 5/11/2023. [Online]. Available: [https://cluster-api.sigs.k8s.io/developer/providers/implementers-guide/controllers\\_and\\_reconciliation.html](https://cluster-api.sigs.k8s.io/developer/providers/implementers-guide/controllers_and_reconciliation.html) [Accessed: 5/11/2023]
98. Y. Kukreja, “Develop on kubernetes series — demystifying the for vs owns vs watches controller-builders in controller-runtime,” *Medium*, 8/10/2022. [Online]. Available: <https://yash-kukreja-98.medium.com/develop-on-kubernetes-series-demystifying-the-for-vs-owns-vs-watches-controller-builders-in-c11ab32a046e> [Accessed: 5/9/2023]
99. Stack Overflow, “language agnostic - what is an idempotent operation? - stack overflow,” 5/11/2023. [Online]. Available: <https://stackoverflow.com/questions/1077412/what-is-an-idempotent-operation> [Accessed: 5/11/2023]
100. P. Wittrock, “What is a controller · the kubebuilder book,” 5/16/2023. [Online]. Available: [https://book-v1.book.kubebuilder.io/basics/what\\_is\\_a\\_controller.html](https://book-v1.book.kubebuilder.io/basics/what_is_a_controller.html) [Accessed: 5/16/2023]
101. Kubernetes, “Finalizers,” 1/8/2022. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/working-with-objects/finalizers/> [Accessed: 5/11/2023]
102. GitHub, “kubernetes-sigs/kubebuilder: Kubebuilder - sdk for building kubernetes apis using crds,” 8/29/2023. [Online]. Available: <https://github.com/kubernetes-sigs/kubebuilder> [Accessed: 8/29/2023]
103. SvelteKit, “Introduction • docs • sveltekit,” 6/16/2023. [Online]. Available: <https://kit.svelte.dev/docs/introduction> [Accessed: 6/16/2023]
104. Rich Harris, “Virtual dom is pure overhead,” 6/16/2023. [Online]. Available: <https://svelte.dev/blog/virtual-dom-is-pure-overhead> [Accessed: 6/16/2023]
105. Vite, “Vite - features,” 6/16/2023. [Online]. Available: <https://vitejs.dev/guide/features.html#hot-module-replacement> [Accessed: 6/16/2023]
106. SvelteKit, “Modules • docs • sveltekit,” 5/29/2023. [Online]. Available: [https://kit.svelte.dev/docs/modules#\\$env-dynamic-private](https://kit.svelte.dev/docs/modules#$env-dynamic-private) [Accessed: 5/29/2023]
107. —, “Page options • docs • sveltekit,” 8/29/2023. [Online]. Available: <https://kit.svelte.dev/docs/page-options> [Accessed: 8/29/2023]
108. Helm, “Helm,” 6/20/2023. [Online]. Available: <https://helm.sh/> [Accessed: 6/20/2023]
109. S. P. Potter, “Ansible vs. puppet: How does each stack up?” *Perforce*, 22.3.2023. [Online]. Available: <https://www.puppet.com/blog/ansible-vs-puppet> [Accessed: 6/27/2023]
110. R. H. Ansible, “How it works,” 8/26/2023. [Online]. Available: <https://www.ansible.com/overview/how-ansible-works> [Accessed: 8/29/2023]
111. Kubernetes, “Container runtimes,” 2022. [Online]. Available: <https://kubernetes.io/docs/setup/production-environment/container-runtimes/> [Accessed: 3/7/2023]



112. Tigera, “Project calico — tigera,” 3/11/2023. [Online]. Available: <https://www.tigera.io/project-calico/> [Accessed: 5/1/2023]
113. Cloud Native Computing Foundation, “Keycloak joins cncf as an incubating project — cloud native computing foundation,” 2023. [Online]. Available: <https://www.cncf.io/blog/2023/04/11/keycloak-joins-cncf-as-an-incubating-project/> [Accessed: 7/21/2023]
114. Project Contour Authors, “Contour,” 5/1/2023. [Online]. Available: <https://projectcontour.io/> [Accessed: 5/1/2023]
115. Kubernetes, “Introduction - kubernetes gateway api,” 5/1/2023. [Online]. Available: <https://gateway-api.sigs.k8s.io/> [Accessed: 5/1/2023]
116. Project Contour Authors, “Deploying https services with contour and cert-manager,” 8/11/2023. [Online]. Available: <https://projectcontour.io/guides/cert-manager/> [Accessed: 8/11/2023]
117. G. Jacobs, “Automating certificate management in a kubernetes environment,” *NGINX, Inc*, 5/10/2022. [Online]. Available: <https://www.nginx.com/blog/automating-certificate-management-in-a-kubernetes-environment/> [Accessed: 8/11/2023]
118. Kubernetes, “Kubernetes v1.26: Alpha support for cross-namespace storage data sources,” 2023. [Online]. Available: <https://kubernetes.io/blog/2023/01/02/cross-namespace-data-sources-alpha/> [Accessed: 7/31/2023]
119. —, “Schedule gpus,” 3/17/2023. [Online]. Available: <https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/> [Accessed: 5/1/2023]
120. GitHub, “Contributors to kubernetes-sigs/kubebuilder,” 5/2/2023. [Online]. Available: <https://github.com/kubernetes-sigs/kubebuilder/graphs/contributors> [Accessed: 5/2/2023]
121. —, “Contributors to operator-framework/operator-sdk,” 5/2/2023. [Online]. Available: <https://github.com/operator-framework/operator-sdk/graphs/contributors> [Accessed: 5/2/2023]
122. D. Messer, “Operator sdk reaches v1.0,” 5/2/2023. [Online]. Available: <https://cloud.redhat.com/blog/operator-sdk-reaches-v1.0> [Accessed: 5/2/2023]
123. Auth.js, “Auth.js - sveltekit,” 5/18/2023. [Online]. Available: <https://authjs.dev/reference/sveltekit> [Accessed: 5/18/2023]
124. GitHub, “Dan6erbond/sk-auth: Authentication library for use with sveltekit featuring built-in oauth providers and zero restriction customization!” 5/18/2023. [Online]. Available: <https://github.com/Dan6erbond/sk-auth/issues/112> [Accessed: 5/18/2023]
125. cri-o team, “cri-o/install.md at main · cri-o/cri-o,” 5/1/2023. [Online]. Available: <https://github.com/cri-o/cri-o/blob/main/install.md> [Accessed: 5/1/2023]
126. Kubernetes, “Upgrading kubeadm clusters,” 3/1/2023. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/> [Accessed: 5/1/2023]

127. —, “Upgrading linux nodes,” 3/2/2023. [Online]. Available: <https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/upgrading-linux-nodes/> [Accessed: 5/1/2023]
128. Project Contour, “Documentation,” 5/1/2023. [Online]. Available: <https://projectcontour.io/docs/1.24/deploy-options/#running-without-a-kubernetes-loadbalancer> [Accessed: 5/1/2023]
129. —, “Documentation,” 5/1/2023. [Online]. Available: <https://projectcontour.io/docs/v1.19.1/config/fundamentals/> [Accessed: 5/1/2023]
130. cert manager, “Selfsigned,” 4/10/2023. [Online]. Available: <https://cert-manager.io/docs/configuration/selfsigned/> [Accessed: 4/10/2023]
131. A. Pejakovic, “Kubernetes — authenticating to your cluster using keycloak,” *ELMO Software*, 2/7/2021. [Online]. Available: <https://medium.com/elmo-software/kubernetes-authenticating-to-your-cluster-using-keycloak-eba81710f49b> [Accessed: 4/10/2023]
132. Kubebuilder, “Implementing defaulting/validating webhooks - the kubebuilder book,” 6/9/2023. [Online]. Available: <https://book.kubebuilder.io/cronjob-tutorial/webhook-implementation.html> [Accessed: 6/9/2023]
133. —, “Implementing a controller - the kubebuilder book,” 5/16/2023. [Online]. Available: <https://book.kubebuilder.io/cronjob-tutorial/controller-implementation.html> [Accessed: 5/16/2023]
134. GitHub, “kube-controller-manager unable to perform normal gc · issue #7598 · projectcalico/calico,” 6/9/2023. [Online]. Available: <https://github.com/projectcalico/calico/issues/7598> [Accessed: 6/9/2023]
135. —, “Fix ondelete option for subdirectories by forselli-stratio · pull request #221 · kubernetes-sigs/nfs-subdir-external-provisioner,” 6/9/2023. [Online]. Available: <https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner/issues/272> [Accessed: 6/9/2023]
136. Kubernetes, “Horizontal pod autoscaler,” 3/27/2023. [Online]. Available: <https://kubernetes.io/de/docs/tasks/run-application/horizontal-pod-autoscale/> [Accessed: 6/13/2023]
137. SvelteKit, “Hooks • docs • sveltekit,” 5/29/2023. [Online]. Available: <https://kit.svelte.dev/docs/hooks#server-hooks-handle> [Accessed: 5/29/2023]
138. —, “Routing • docs • sveltekit,” 5/29/2023. [Online]. Available: <https://kit.svelte.dev/docs/routing> [Accessed: 5/29/2023]
139. NodeJS, “Command-line api — node.js v20.2.0 documentation,” 5/16/2023. [Online]. Available: [https://nodejs.org/api/cli.html#cli\\_node\\_extra\\_ca\\_certs\\_file](https://nodejs.org/api/cli.html#cli_node_extra_ca_certs_file) [Accessed: 5/19/2023]
140. Stack Overflow, “javascript - unable to verify leaf signature - stack overflow,” 5/19/2023. [Online]. Available: <https://stackoverflow.com/questions/20082893/unable-to-verify-leaf-signature> [Accessed: 5/19/2023]
141. —, “node.js - nodejs unable to verify leaf signature - stack overflow,” 5/19/2023. [Online]. Available: <https://stackoverflow.com/questions/17200391/nodejs-unable-to-verify-leaf-signature> [Accessed: 5/19/2023]

142. GitHub, “feat(ssr): better dx with sourcemaps, breakpoints, error messages by aleclarson · pull request #3928 · vitejs/vite,” 5/22/2023. [Online]. Available: <https://github.com/vitejs/vite/pull/3928> [Accessed: 5/22/2023]
143. —, “Debug server side code using –inspect · issue #1144 · sveltejs/kit,” 5/22/2023. [Online]. Available: <https://github.com/sveltejs/kit/issues/1144> [Accessed: 5/22/2023]
144. Stack Overflow, “devops - keycloak realm vs keycloak client - stack overflow,” 6/16/2023. [Online]. Available: <https://stackoverflow.com/questions/56561554/keycloak-realm-vs-keycloak-client> [Accessed: 6/16/2023]
145. Helm, “Custom resource definitions,” 6/23/2023. [Online]. Available: [https://helm.sh/docs/chart\\_best\\_practices/custom\\_resource\\_definitions/](https://helm.sh/docs/chart_best_practices/custom_resource_definitions/) [Accessed: 6/23/2023]
146. Puppet, “Puppet pricing, plans & packages — puppet by perforce,” 6/26/2023. [Online]. Available: <https://www.puppet.com/pricing> [Accessed: 6/27/2023]
147. Ansible, “Red hat ansible automation platform pricing and deployment options,” 6/27/2023. [Online]. Available: <https://www.redhat.com/en/technologies/management/ansible/pricing> [Accessed: 6/27/2023]
148. Ubuntu, “Minimal - ubuntu wiki,” 6/20/2023. [Online]. Available: <https://wiki.ubuntu.com/Minimal> [Accessed: 6/20/2023]
149. Stack Overflow, “openthread/environment docker rsyslogd: imklog: cannot open kernel log (/proc/kmsg): Operation not permitted - stack overflow,” 6/20/2023. [Online]. Available: <https://stackoverflow.com/questions/56609182/openthread-environment-docker-rsyslogd-imklog-cannot-open-kernel-log-proc-km> [Accessed: 6/20/2023]
150. SvelteKit, “Adapters • docs • sveltekit,” 6/20/2023. [Online]. Available: <https://kit.svelte.dev/docs/adapters> [Accessed: 6/20/2023]
151. Stack Overflow, “node.js - how to deploy sveltekit app in node server with https? - stack overflow,” 6/20/2023. [Online]. Available: <https://stackoverflow.com/questions/68767798/how-to-deploy-sveltekit-app-in-node-server-with-https> [Accessed: 6/20/2023]
152. SvelteKit, “Node servers • docs • sveltekit,” 6/20/2023. [Online]. Available: <https://kit.svelte.dev/docs/adapters-node#custom-server> [Accessed: 6/20/2023]
153. Kubebuilder, “Running and deploying the controller - the kubebuilder book,” 6/19/2023. [Online]. Available: <https://book.kubebuilder.io/cronjob-tutorial/running.html> [Accessed: 6/19/2023]
154. Calico, “Quickstart for calico on kubernetes — calico documentation,” 6/19/2023. [Online]. Available: <https://docs.tigera.io/calico/latest/getting-started/kubernetes/quickstart> [Accessed: 6/19/2023]
155. Kubernetes, “User namespaces,” 6/2/2023. [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/pods/user-namespaces/> [Accessed: 7/7/2023]
156. Red Hat, Inc., “Nvd - cve-2022-0492,” 7/7/2023. [Online]. Available: <https://nvd.nist.gov/vuln/detail/CVE-2022-0492> [Accessed: 7/7/2023]

- 
157. Operator SDK, “Operator best practices,” 21.06.2023. [Online]. Available: <https://sdk.operatorframework.io/docs/best-practices/best-practices/> [Accessed: 08.08.2023]

# A

---

## Glossary

### *General*

VNC	Virtual Network Computing
SSH	Secure Shell
NIS	Network Information Service
PID	Process ID
IPC	Inter-Process Communication
CA	Certificate Authority
OIDC	OpenID Connect
RBAC	Role Based Access Control
IPC	Inter Process Communication
LSM	Linux Security Modules
CI	Continuous Integration
SSR	Server-Side Rendering
MAC	Mandatory Access Control
DAC	Discretionary Access Control
HMR	Hot Module Replacement
CVE	Common Vulnerabilities and Exposures

### *Container*

CRI	Container Runtime Interface
OCI	Open Container Initiative
CNCF	Cloud Native Computing Foundation

### *Kubernetes*

PVC	Persistent Volume Claim
PV	Persistent Volume
CRD	Custom Resource Definition

B

---

## Declaration of autonomy

Ich erkläre hiermit, dass

- ich die vorliegende wissenschaftliche Arbeit selbstständig und ohne unerlaubte Hilfe angefertigt habe,
- ich andere als die angegebenen Quellen und Hilfsmittel nicht benutzt habe,
- ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe,
- die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfbehörde vorgelegen hat.

Berlin, 30.08.2023



---

Florian Reitz