HAW HAMBURG

BACHELOR THESIS
Jessica Moll

Quotierung und Priorisierung in einem Multi-Tenancy Kubernetes Cluster

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Jessica Moll

Quotierung und Priorisierung in einem Multi-Tenancy Kubernetes Cluster

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Martin Hübner
Zweitgutachter: Dr. Tobias Eichler
Fachlicher Betreuer: M. Sc. Jan Dalski

Eingereicht am: 28. März 2024

Jessica Moll

Thema der Arbeit

Quotierung und Priorisierung in einem Multi-Tenancy Kubernetes Cluster

Stichworte

Kubernetes, Multi-Tenancy, Quotierung, Limitierung, Prioritäten, Monitoring, Ressourcenmanagement, Kiosk, Hierarchical Namespaces, Policy Engine, Kyverno, GPU-Metriken

Kurzzusammenfassung

Sobald mehrere Menschen oder Anwendungen auf einem Cluster arbeiten, muss darauf geachtet werden, dass sich diese nicht ungewollt in die Quere kommen. Um solche Konflikte zu verhindern, untersucht diese Arbeit die Ressourcenverteilung in einem Multi-Tenancy Cluster.
Es wird ein Konzept gezeigt, welches die Pods, Namespaces und Gruppen von Namespaces in ihrem Ressourcenverbrauch limitiert. Durch diese Quotierung ist der Ressourcenverbrauch der Nutzenden, der Anwendungen und damit des Clusters besser kontrollierbar. Außerdem können durch Priorisierungskonzepte Requests in ihrer Wichtigkeit gesteuert werden. Dadurch können bestimmte Requests bevorzugt oder gezielt auf ruhigere Phasen des Clusters verschoben werden. Die Priorisierung hilft damit, die zeitliche Komponente der Ressourcenzuteilung zu steuern.
Um die Ressourcen des Clusters besser zu überwachen, werden ergänzende Monitoringmaßnahmen eingeführt. Hier sollen inaktive Pods erkannt werden, um unnötig besetzte Ressourcen zu erkennen. Außerdem wird ein Konzept zur Überwachung der ergänzenden GPU-Ressourcen vorgestellt, um ein umfassenderes Bild der Ressourcennutzung zu ermöglichen.
Im Rahmen dieser Arbeit wird zusätzlich die vorhandene Multi-Tenancy Anwendung Kiosk mit dem aktuelleren Hierarchical Namespace Controller abgelöst. Mit Admission Controllern wird dabei erreicht, die existierenden Ressourcen verlustfrei in die neue Struktur zu überführen.

Jessica Moll

Title of Thesis

Quotas and Priorities in a Multi-Tenancy Kubernetes Cluster

Keywords

Kubernetes, Multi-Tenancy, Quota, Limits, Priorities, Monitoring, Resource management, Kiosk, Hierarchical Namespaces, Policy Engine, Kyverno, GPU metrics

Abstract

When multiple users or applications are working on a single cluster, it is important to ensure they do not inadvertently interfere with each other. With the aim to minimize such interference, this paper investigates the resource allocation in a multi-tenancy cluster with static resources.
A concept is presented to limit the resources of pods, namespaces, and groups of namespaces, which helps to control the resource usage of users, applications, and thus the cluster itself. In addition, priorities allow indicating the importance of requests. This makes it possible to favour requests or defer them to later times with lower traffic. Therefore, the prioritization helps manage the temporal aspect of resource allocation.
To better monitor the cluster's resources, supplementary monitoring measures are introduced. They aim at identifying inactive pods to recognize unnecessarily occupied resources. Furthermore, a concept for monitoring additional GPU resources is presented to provide a more comprehensive view of the cluster's resource usage.
Additionally, this paper provides a solution to replace the existing multi-tenancy application Kiosk with the newer Hierarchical Namespace Controller. Admission controllers are used to transition existing resources into the new structure without loss.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Cloud Computing ist aus der heutigen IT-Landschaft nicht mehr wegzudenken. Dabei beschreibt Cloud Computing ein Modell, das es ermöglicht, jederzeit über ein Netzwerk auf einen geteilten Pool an konfigurierbaren IT-Dienstleistungen zugreifen zu können [18]. Dabei ist das Netzwerk, über den der Zugriff erfolgt, in den meisten Fällen das Internet. Die IT-Dienstleistungen, die darüber angefragt werden, können die Rechenressourcen sein, die man vom eigenen Computer kennt: Prozessorleistung (CPU), Hauptspeicher (RAM), persistenter Speicher (Festplatten, SSDs). Aber auch komplette Anwendungen und Plattformen können bereitgestellt werden. Diese Abstraktion der Infrastruktur bietet dabei viele Vorteile. Primär müssen sich die Nutzenden keine Gedanken über die zugrunde liegende physische Infrastruktur machen und können sich beispielsweise auf die Entwicklung neuer Funktionen und Anwendungen konzentrieren. Die Compute Cloud bietet zusätzlich Skalierungsmöglichkeiten, wodurch je nach Bedarf Ressourcen provisioniert und skaliert werden können. Dadurch können Anwendungen bei Bedarf mehr Ressourcen bekommen, um sich ändernden Anforderungen gerecht zu werden. Die Hochschule für Angewandte Wissenschaften in Hamburg (HAW) betreibt eine eigene Cloud-Compute-Plattform namens „Informatik Compute Cloud“ (ICC) für Forschungsgruppen und Studierende. Die ICC wird als Kubernetes [5] Cluster betrieben. Dabei stehen dem Cluster feste Ressourcen zur Verfügung, die an der HAW vor Ort betrieben werden. Die Mitglieder des Departments Informatik können mit einem eigenen Zugang zum Cluster containerbasierte Anwendungen mit den verfügbaren Ressourcen betreiben [56]. Zusätzlich betreibt die HAW auch weitere öffentlich zugängliche Anwendungen, die mit den Ressourcen der Cloud arbeiten.

1.1 Problemstellung und Motivation

Die ICC wird von unterschiedlichen Nutzungsgruppen und Anwendungen genutzt. Alle diese Entitäten werden als „Tenants“ also Mieter des Clusters bezeichnet. Das macht die ICC zu einem sogenannten „Multi-Tenancy“ Cluster, was bedeutet, dass verschiedene Tenants mit verschiedenen Anforderungen auf dem Cluster arbeiten. Dadurch ergeben sich spezielle Anforderungen an die Isolation zwischen den einzelnen Tenants. Das beginnt bei der Ressourcenverteilung mit der Frage, wer welche Clusterressourcen bekommen darf und wie viel davon. Weiter müssen Fragen zur Netzwerkisolierung geklärt werden, also wer mit wem innerhalb des Clusters oder sogar über die Grenzen des Clusters hinweg kommunizieren darf. Zusätzlich müssen die Daten der Tenants vor unberechtigtem Zugriff geschützt werden und Prozesse isoliert werden, um zu verhindern, dass sie sich nicht untereinander beeinträchtigen. Alle diese Aspekte sind wichtig für die Effizienz und Sicherheit der Plattform.

Diese Arbeit konzentriert sich auf die Herausforderung der Ressourcenverteilung innerhalb der ICC als Multi-Tenancy Cluster. Die ICC arbeitet nicht mit einem externen Cloud-Anbieter, sondern baut auf physikalisch in der Hochschule betriebenen Ressourcen auf. Das bedeutet Ressourcen für das Cluster können nicht einfach nach Bedarf angefordert werden, sondern sind durch die verfügbare Hardware festgelegt. Ohne ein Konzept die verfügbaren Ressourcen zu verteilen, können diese in ungewünschten Quantitäten an die Anwendungen und Nutzenden zugeteilt werden. Im schlimmsten Fall können sie ungeplant erschöpft werden.

1.2 Zielsetzung

Ziel der Arbeit ist es, ein Konzept für die ICC zu entwickeln, durch welches die Ressourcenverteilung besser gesteuert werden kann. Alle Tenants sollen entsprechend konfigurierbaren Regelungen Ressourcen zur Verfügung gestellt bekommen. Zusätzlich soll gezeigt werden, wie die Ressourcenverteilung im Cluster überwacht und kontrolliert werden kann. Zentral soll dafür mit einer Quotierung gesteuert werden, wie viele Ressourcen ausgewählten Tenants maximal zugeteilt werden. Daneben soll mit einer Priorität die Möglichkeit gegeben werden, bestimmte Ressourcenanfragen bevorzugt zu behandeln.

Durch die Messung der Aktivität von Pods und der GPU-Auslastung soll eine weitere Metrik zur Überwachung der genutzten Ressourcen geboten werden.

1.3 Struktur der Arbeit

Um eine Grundlage zu den in der Arbeit verwendeten Begriffen und Konzepten aufzubauen, beginnt die Arbeit in Kapitel 2 damit Container selbst und daraufhin Kubernetes als Container-Orchestrations-System und dessen Konzepte zum Ressourcenmanagement darzustellen. Dabei wird auch der Begriff der „Multi-Tenancy“ erklärt.

Da das erstellte Konzept für das Kubernetes Cluster der HAW entworfen wird, wird in Kapitel 3 dessen Architektur und die existierenden Anwendungen betrachtet. Dabei werden die Multi-Tenancy Anwendungen Kiosk, JupyterHub und GitLab Runner als primäre Ziele des Konzepts festgehalten und die benötigten Quotierungs-, Priorisierungs- und Monitoringfunktionalitäten durch Use-Cases und Qualitätsszenarien konkretisiert.

In Kapitel 4 werden unterschiedliche Lösungen für die aufgestellten Anforderungen untersucht. Dabei werden zuerst die bereits verfügbaren Mechaniken von Kubernetes selbst und den bestehenden Anwendungen betrachtet. Anschließend werden zusätzlich externe Erweiterungen betrachtet die fehlende Funktionalitäten ergänzen oder anderweitig das Konzept verbessern sollen.

Aufbauend darauf wird in Kapitel 5 das Konzept zur Lösung der unterschiedlichen Anforderungen beschrieben. Zuerst werden die ausgewählte neu eingeführten Erweiterungen vorgestellt. Anschließend werden neben clusterweiten Konzepten zur Priorisierung und Monitoring auch anwendungsbezogene Lösungen dargestellt.

Das Konzept wird im Anschluss in Kapitel 6 evaluiert, indem untersucht wird, ob die aufgestellten Qualitätsziele erfüllt wurden.

Schließlich wird in Kapitel 7 das Fazit zur Arbeit präsentiert mit einem Ausblick auf die Zukunft der ICC und mögliche Weiterentwicklungen.

2 Grundlagen

Kubernetes ist eine der meistgenutzten Container-Orchestrations-Systeme [8]. Das Ziel von Kubernetes ist es, den Prozess zur Bereitstellung und Verwaltung von containerisierten Anwendungen zu vereinfachen. Container sind dabei eine Art von Virtualisierungstechnologie, die es ermöglicht, Anwendungen und ihre Abhängigkeiten in isolierten Umgebungen auszuführen.

2.1 Container als Virtualisierungstechnologie

Im Informatikbereich beschreibt die Virtualisierung die Erstellung virtueller Computerbestandteile wie Hardware, Software oder Betriebssystemen. Bei der Containertechnologie können auf einem Host mehrere Container als eigenständige Computer betrieben werden. Dabei teilen sie sich das Betriebssystem und die darunterliegenden Ressourcen mit dem Host. Durch die Virtualisierung auf Betriebssystemebene haben Container einen entscheidenden Vorteil gegenüber den klassischeren virtuellen Maschinen, was die Performance und Portabilität betrifft [10].

2.2 Kubernetes als Container-Orchestrations-System

Die Vorteile von Containern haben sie zu einer bevorzugten Wahl für die Bereitstellung von Cloud-nativen Anwendungen und Mikrodienstarchitekturen gemacht. Allerdings sind mit der Zeit auch die Anforderungen der Nutzenden gewachsen und die containerisierten Applikationen größer und komplexer geworden. Das Management und die Koordination vieler Container wurde zur eigenen Herausforderung. Um diese Probleme anzugehen, wurde 2015 [58] Kubernetes als *Container-Orchestrations-Plattform* vorgestellt, um containerisierten Anwendungen in ihren Abläufen zu orchestrieren und ihre Stabilität zu

verbessern. Dabei unterstützt Kubernetes die Nutzenden dabei, konkrete Anforderungen an die Anwendungen wie Verfügbarkeit, Skalierbarkeit und Performance zu erreichen.

Dafür bietet Kubernetes ein Framework, mit dem Nutzende ihre gewünschten containerisierten Aufgaben beim Kubernetes Cluster einreichen. Dabei ist das Cluster eine Abstraktion von miteinander verbundenen Knoten (Nodes), die physische Rechenmaschinen oder virtuelle Maschinen sein können. Kubernetes kümmert sich nach einer Anfrage darum, die gewünschten Container auf den Nodes des Clusters zu erstellen und sicherzustellen, dass zu jedem Zeitpunkt die gewünschte Anzahl an Containern läuft.

2.2.1 Objekte

Die Container, die im Cluster laufen, werden dabei von Kubernetes abstrahiert. Kubernetes arbeitet dafür mit einer eigenen Definition von Objekten.

Pods
Pods sind die kleinste erstellbare Einheit in Kubernetes. Sie sind die erste Abstraktionsebene der tatsächlichen Container [26]. Ein Pod kann dabei einen oder mehrere zusammengehörige Container enthalten. Pods sind in Kubernetes *ephemere* Objekte. Das bedeutet, dass sie dafür entworfen sind, kurzlebig zu sein. Sie können schnell erstellt und entfernt werden. Dies ermöglicht Kubernetes beispielsweise, sie nach einem Absturz jederzeit zu ersetzen, auf anderen Nodes zu starten oder zu replizieren – je nach Anforderung des Systems.

Workloads
Pods sind die am häufigsten erstellten Objekte in Kubernetes, aber sie werden meist nie direkt von den Nutzenden des Clusters erstellt. Stattdessen bietet Kubernetes eine weitere Abstraktionsebene mit *Workloads* an. Mit einer Workload kann eine Gruppe an Pods erstellt werden, woraufhin sich Kubernetes darum kümmert, dass die definierte Anzahl und Art läuft [47].

Durch die Workloads namens **Deployment** werden Replikationen derselben Pods erstellt, um beispielsweise die Verfügbarkeit der laufenden Applikation zu verbessern. Statt manuell mehrere Pods derselben Anwendung zu erstellen, kann die Konfiguration des Pods und die gewünschte Anzahl in einem Deployment angegeben werden. Ein **Job** kümmert sich darum, die Ausführung der angegebenen Pods so lange zu wiederholen, bis

eine definierte Anzahl der Pods erfolgreich beendet wurden[1]. **CronJobs** werden genutzt, um in festgelegten Zeitabständen Jobs zu erstellen.

Daneben bietet Kubernetes noch viele weitere nützliche Definitionen, die alle unter dem Begriff *Workloads* zusammengefasst werden. Dabei stellen diese Workloads immer eine Abstraktion von Pods dar.

Service
Pods können auch untereinander kommunizieren. Dafür bietet Kubernetes ein eigenes virtuelles Netzwerk, indem jeder Pod seine eigene IP-Adresse bekommt. Wenn ein Pod abstürzt und neu gestartet wird, bekommt er jedes Mal eine neue IP-Adresse. Damit Pods dennoch zuverlässig miteinander kommunizieren können, gibt es **Services**. Services haben eine permanente IP-Adresse und können mit jedem Pod verknüpft werden. Dabei ist die Lebensdauer des Pods nicht mit dem des Service verbunden. Das bedeutet, selbst wenn der Pod abstürzt, behält der Service seine initiale Adresse [48].

Kubernetes verfolgt für die Verwaltung der Objekte einen deklarativen Ansatz. Die Objekte werden genutzt, um den Status des Clusters zu repräsentieren. Mit den Objekten und deren Konfigurationen beschreiben Nutzende den gewünschten Zustand ihrer Anwendungen, ohne selbst spezifische Anweisungen zu geben, wie dieser Zustand erreicht werden soll. Kubernetes kennt durch die Objekte den gewünschten Zustand und arbeitet als System konstant daran, diesen und die dazugehörigen Objekte herzustellen [40].

2.2.2 Architektur

An der Realisierung des gewünschten Zustands des Clusters sind in Kubernetes verschiedene Komponenten und Prozesse beteiligt, welche in Abbildung 2.1 gezeigt sind.

Node
Nodes stellen die Rechenressourcen bereit und können entweder physische Rechner oder virtuelle Maschinen sein. Alle Pods und damit auch alle Container des Clusters laufen auf diesen Nodes. Auf den Nodes sind dabei drei Prozesse aktiv.

[1] Ein Pod gilt als erfolgreich beendet, wenn alle enthaltenen Container mit einem Exit-Code 0 beendet wurden [30].

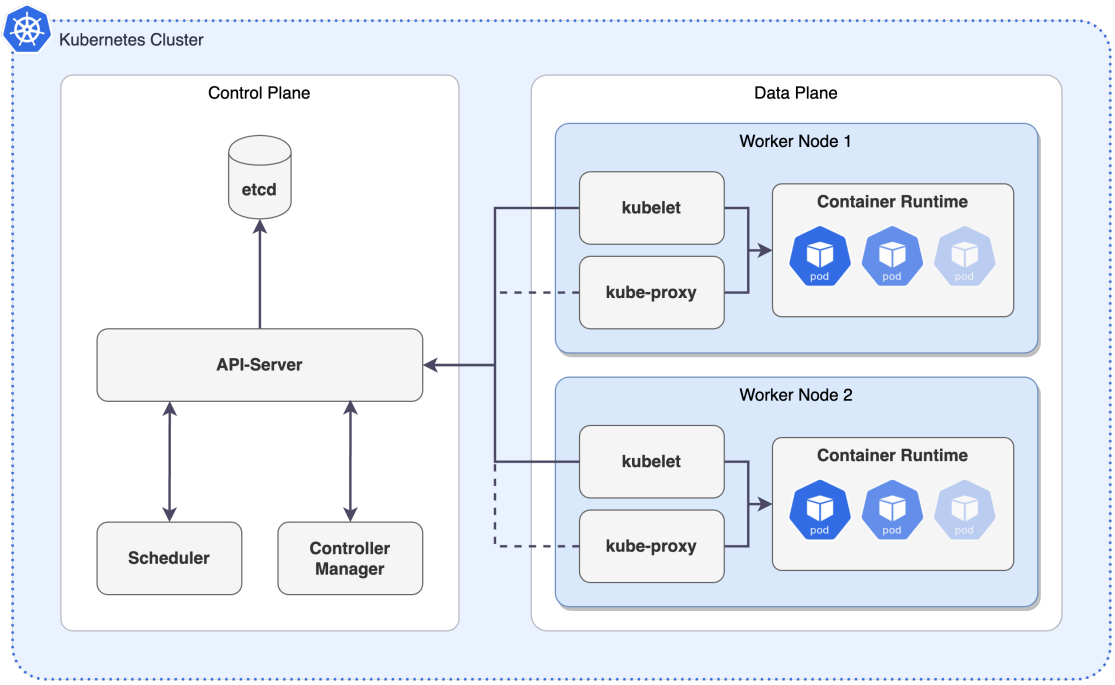

Abbildung 2.1: Kubernetes Architektur

- Eine **Container Runtime** ist notwendig, damit die Container der Pods auf den Nodes laufen können. Dabei kann jede Container Runtime genutzt werden, die den Container Runtime Interface (CRI) Standard implementiert [27].

- **kubelet** interagiert mit den Containern und dem Node. `kubelet` kümmert sich darum, dass die Pods und die darunterliegenden Container auf dem Node laufen. Es startet die Container und teilt ihnen die nötigen Ressourcen des Nodes zu.

- **kube-proxy** ist dafür zuständig, Anfragen an die Services zu den tatsächlich verknüpften Pods weiterzuleiten [34].

Control Plane

Die *Control Plane* verwaltet und steuert das Cluster. Die einzelnen Komponenten der Control Plane können dabei gemeinsam auf einem dedizierten Node oder verteilt auf unterschiedlichen Nodes des Clusters laufen [34]. Den Kern der Control Plane bilden dabei vier Komponenten.

- Der **API-Server** stellt das Frontend zum Cluster dar. Der API-Server ist der einzige Weg, mit dem Cluster zu kommunizieren [32, 31]. Endnutzer*innen können

Anfragen stellen, aber auch Komponenten des Clusters selbst und externe Dienste nutzen den API-Server, um mit dem Cluster und miteinander zu kommunizieren. Dabei bietet der API-Server eine ressourcenbasierte REST-Schnittstelle per HTTP an, über die mittels *Requests* Objekte angefragt und verändert werden können [33].

- Das **etcd** ist im Grunde ein großer Key-Value Store. Im etcd werden alle Informationen über den Status des Clusters gespeichert. Dazu gehören Informationen zu Konfigurationen, Berechtigungen und Ressourcenstatus. Wird über einen Request eine Änderung des Clusters angefragt, wird auch diese im etcd festgehalten.

- Ändert sich der gewünschte Status des Clusters, versucht der **Controller Manager** diesen Status zu realisieren. Dabei hört der Controller Manager auf Änderungen durch den API-Server, um anschließend den im etcd beschriebenen Zustand herzustellen. Dabei enthält der Controller-Manager mehrere einzelne *Controller*, die für die Logik bestimmter Objekte zuständig sind. Beispielweise ist es der Job-Controller, der sich darum kümmert, dass die im Job-Objekt definierten Pods erstellt und erfolgreich beendet werden.

- Der **Scheduler** entscheidet darüber, auf welchem Node des Clusters ein neuer Pod gestartet werden soll. Dabei ist zu beachten, dass der Scheduler nur die Entscheidung trifft und anschließend `kubelet` auf dem entsprechenden Node dafür zuständig ist, den Pod zu starten.

Um ein Objekt in Kubernetes zu erstellen oder zu bearbeiten, müssen Nutzende den gewünschten Zustand über die Kubernetes API einreichen. Der Body des API-Requests muss dabei das Objekt mit allen nötigen Eigenschaften als JSON enthalten. Meistens wird dabei mit dem Cluster über das CLI `kubectl` interagiert. Dieses bietet dabei als Komfortfunktion an, die Eigenschaften der Objekte in Form von Manifest-Dateien im **YAML**-Format anzugeben [40].

Schedulingprozess

Abbildung 2.2 zeigt die beteiligten Komponenten des Schedulingprozesses vom Request nach einem Pod bis zu dem Punkt, an dem er tatsächlich auf einem Node persistiert wird. Im Falle der Erstellung eines Pods ist der Controller Manager nicht direkt beteiligt. Die entsprechenden Controller sind erst bei der Erstellung höherer Abstraktionen wie Deployments zuständig, die dahinterliegenden Pods durch eigene Requests an den API-Server im Hintergrund zu erstellen.

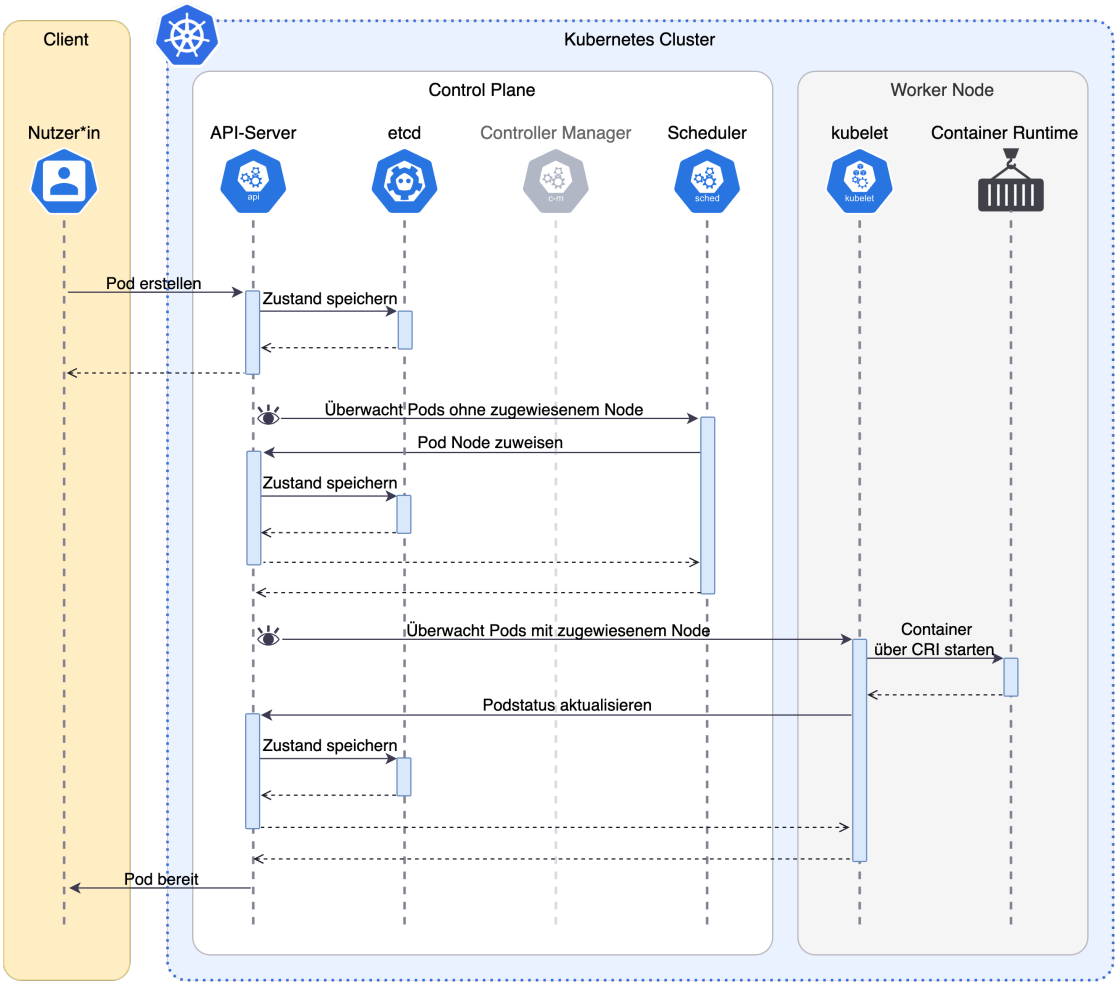

Abbildung 2.2: Kubernetes Schedulingprozess

2.3 Ressourcenmanagement in Kubernetes

In Kubernetes gibt es unterschiedliche Arten von Ressourcen, die verwaltet und angefragt werden können. Welche Eigenschaften die einzelnen Ressourcen haben und wie Pods ihre Ressourcenanforderungen gegenüber dem Cluster kommunizieren können, soll im folgenden Abschnitt erläutert werden.

2.3.1 Arten von Ressourcen

In Kubernetes wird generell zwischen Rechenressourcen und API-Ressourcen unterschieden. CPU und Arbeitsspeicher (RAM, in Kubernetes als *Memory* bezeichnet) sind klassische Rechenressourcen. Sie können in messbare Quantitäten angefragt, zugeteilt und verbraucht werden. Das sind die Ressourcen der Nodes, die letztlich von den laufenden Containern beansprucht werden.

Daneben sind API-Ressourcen alle Kubernetes Objekte, die mittels des API-Servers gelesen und modifiziert werden können [43]. Darunter fallen die bereits genannten Objekte wie Pods, Deployments oder Services.

Kubernetes unterscheidet dabei zwischen *namespace-weiten* und *cluster-weiten* API-Ressourcen, im Folgenden *Namespace-* und *Cluster-Ressourcen* genannt. Namespace-Ressourcen sind solche, die nur innerhalb eines *Namespaces* sichtbar sind. Namespaces können dabei wie Ordner für diese API-Ressourcen betrachtet werden, wobei jede Ressource zu genau einem Namespaces gehört. Diese Mechanik wird genutzt, um isolierte Gruppen an API-Ressourcen im Cluster zu definieren [38, 87]. Dies ermöglicht unterschiedlichen Anwendungen oder Teams in eigenen Namespaces zu arbeiten, ohne Namenskonflikte untereinander zu bekommen [38]. Außerdem wird von Kubernetes verhindert, dass die Ressourcen eines Namespaces ungewollt mit denen eines anderen über das virtuelle Netzwerk interagieren.

Die eben genannten Objekte wie Pods, Deployments und Services gehören zu den Namespace-Ressourcen. Cluster-Ressourcen sind dagegen global und über alle Namespaces hinweg sichtbar und nicht auf einen bestimmten Namespace begrenzt. Beispiele für diese Cluster-Ressourcen in Kubernetes sind Namespaces selbst oder Nodes. Eine besondere Art der API-Ressourcen sind *Custom Resources*. Mit diesen kann die Kubernetes-API um eigene Ressourcendefinitionen erweitert werden. Dafür wird eine `CustomResourceDefinition` (CRD) angelegt, wodurch Kubernetes einen neuen REST-Endpunkt für das definierte Objekt erstellt. Da die CRDs jedoch neben den so ermöglichten CRUD-Operationen meist noch zusätzliche Funktionen auslösen sollen, wird häufig auch ein eigener *Controller* für die neue Ressource erstellt. Dieser kann als Workload im Cluster direkt angelegt werden und reagiert auf Ereignisse des API-Servers, wie das Erstellen oder Verändern einer CRD.

Eine besondere Ressource stellt persistenter Speicher dar. Pods selbst sind ephemer und können keine Zustände langfristig speichern. Bei einem Neustart des Pods gehen alle Daten verloren. Doch da Anwendungen häufig Daten auch langfristig speichern wollen, bietet Kubernetes Mechanismen zum persistenten Speichern an. Die Kubernetes Objekte, die dafür genutzt werden, sind `PersistentVolumes` (PVs) und `PersistentVolumeClaims` (PVCs). PVs können dabei auf verschiedenen Speichermedien basieren, wie lokale Festplatten oder Cloud-basierte Speicherlösungen. Ein Pod kann den Speicher dann über einen PVC in seinen Anforderungen definieren, wodurch Kubernetes den Pod und das passende PV verknüpft. So kann ein physischer Speicher mit einem Pod verbunden werden [24]. Dadurch können Anwendungen Daten speichern, die über die Lebensdauer einzelner Pods hinaus bestehen bleiben.

2.3.2 Requests und Limits

Um dem Cluster mitzuteilen, welche Ressourcen ein Node für einen Pod verfügbar haben muss, können Pods bei ihrer Erstellung ihre Ressourcenanforderungen definieren. Standardmäßig können dabei die Anforderungen zu CPU, Memory und lokalem ephemeren Speicher definiert werden. Diese Anforderungen werden dann vom Scheduler für die passende Nodeauswahl berücksichtigt [82]. Daneben können durch entsprechende Plug-ins auch erweiternde Ressourcen wie GPUs angefordert werden. Die geforderten Rechenressourcen können konkret durch *Requests* definiert werden. Requests legen fest, welche Ressourcen die Container garantiert bekommen. Fordert ein Container anschließend mehr, als er im Request angegeben hatte, ist ihm das generell erlaubt, wobei er sich in einem „überprovisionierten“ Zustand befindet (siehe Abbildung 2.3). Standardmäßig laufen damit alle Pods, welche nur Requests definieren, mit unbegrenzten Rechenressourcen [36]. Um die Ressourcen besser zu kontrollieren, können Pods zusätzlich *Limits* definieren, wodurch die maximale Menge an Ressourcen tatsächlich begrenzt wird [9].

Vom Scheduler werden nur die im Request definierten Werte zur Nodeauswahl berücksichtigt. Diese Ressourcen sind danach aber auch fest für den Pod und die Container reserviert und für andere Pods blockiert.

Die Einhaltung der Limits wird vom `kubelet` des Nodes überwacht. Wenn `kubelet` einen Container als Teil eines Pods startet, übergibt es die Requests und Limits des Containers an die Container Runtime. Die Container Runtime kümmert sich dann darum, dass die definierten Limits eingehalten werden [43]. Überschreitet ein Container die

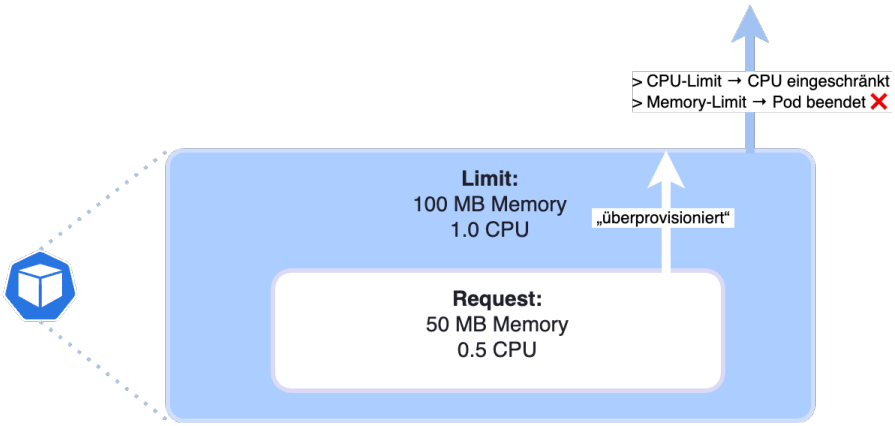

Abbildung 2.3: Limits und Requests eines Pods

festgelegten Grenzen, greift `kubelet` ein und kann mittels verschiedener Maßnahmen das Limit durchsetzen. Die CPU-Ressourcen können komprimiert werden, was bedeutet das bei einer Überschreitung, die CPU-Nutzung für den Pod entsprechend eingeschränkt wird. Der Arbeitsspeicher hingegen kann nicht komprimiert werden. Wenn ein Container das angegebene Memory-Limit überschreitet, wird er beendet [9].

Durch die Verwendung von Requests kann der Scheduler die Ressourcen des Clusters effizienter verteilen und durch Limits können zusätzlich Überlastungen vermieden werden. Insgesamt ist ein effektives Ressourcenmanagement für die Leistungsfähigkeit und Stabilität von Kubernetes Clustern essenziell.

2.4 Multi-Tenancy

Kubernetes Cluster können eine „Multi-Tenancy“ Architektur unterstützen. Das bedeutet, dass generell mehrere Personen gleichzeitig mit dem Cluster arbeiten können. Dies bringt allerdings eigene Herausforderungen mit sich.

2.4.1 Allgemein

Wie schon beim Begriff „Cloud Computing“ hat sich auch für den das Konzept „Multi-Tenancy“ keine allgemeingültige Definition etabliert. Meistens bezieht sich der Begriff auf eine Softwarearchitektur, in der eine einzige Instanz einer Applikation mehreren Nut-

zenden (genannt *Tenants*) zur Verfügung steht. Jeder Tenant kann dabei eine Person oder eine Gruppe an Nutzenden sein. Die Tenants können dabei unabhängig voneinander arbeiten und haben jeweils eine eigene Sicht auf die Applikation, indem jedem ein dedizierter Anteil der Instanz zugeordnet wird. Das kann sich auf die verfügbaren Daten, Konfigurationen oder Funktionalitäten beziehen. Diese unterschiedlichen Anteile sind auch im Bezug auf die Performance und den Datenschutz voneinander isoliert. Dabei teilen sich die Tenants die Infrastruktur der Anwendung wie Rechenressourcen, Datenbanken oder Netzwerkressourcen [23, 84].

2.4.2 Kubernetes

Auch Kubernetes kann als Multi-Tenancy Cluster betrieben werden. Ein Cluster zwischen mehreren Tenants zu teilen bringt mehrere Vorteile. Statt mehrere einzelne Cluster zu betreiben, werden die Kosten und die Administration durch die gemeinsame Nutzung verringert. Allerdings birgt ein Multi-Tenancy Cluster auch eigene Herausforderungen, die die Sicherheit und Fairness betreffen [37].

Kubernetes wurde initial nicht mit dem Gedanken an Multi-Tenancy, sondern für Single-Tenant Deployments entwickelt [7]. Aus diesem Grund existiert in Kubernetes auch kein direktes Konzept von getrennten Endnutzenden oder Tenants [37]. Deshalb muss bei Multi-Tenancy Architekturen in Kubernetes selbst auf eine angemessene Isolierung der Tenants geachtet werden.

Wichtig dafür – jedoch nicht Teil dieser Arbeit – ist die Trennung der Pods und Workloads der Tenants durch beispielsweise Netzwerk- und Speicherisolierung. Diese Arbeit konzentriert sich auf die Ebene der Ressourcenverteilung, welche durch Quotierungs- und Priorisierungskonzepte besser organisiert werden soll.

3 Anforderungsanalyse

3.1 Architektur und Anwendungen der ICC

Um die Funktionsweisen und Anforderungen der ICC besser zu verstehen, soll hier ein kurzer Überblick über die Architektur der ICC gegeben werden. Außerdem soll erläutert werden, warum die ICC als Multi-Tenancy Cluster bezeichnet wird, welche Anwendungen im Cluster laufen und wer das Cluster nutzt. Anschließend werden noch die bestehenden Regelungen zur Ressourcenverteilung betrachtet.

3.1.1 Architektur der ICC

Die ICC ist ein vom Labor für Allgemeine Informatik (AI-Labor) betriebenes Kubernetes-Cluster [55]. Die Hardware des Clusters wird dabei in der Hochschule selbst betrieben. Wie in Abbildung 3.1 gezeigt, werden die Worker bzw. CPU-Nodes durch virtuelle Maschinen abstrahiert, während GPU-Ressourcen und Speicher-Ressourcen durch entsprechende Nodes direkt in das Cluster integriert sind.

Dadurch, dass die Rechenressourcen des Clusters durch die tatsächlichen Rechenmaschinen an der Hochschule festgelegt sind, arbeitet die ICC mit einer statischen Kapazität. Das bedeutet, die Anzahl der Ressourcen ändert sich in der Regel nicht und es ist keine dynamische Skalierung möglich. In Clustern mit dynamischen Ressourcen ist eine effektive Ressourcenverteilung wichtig, da ein höherer Verbrauch mit höheren Kosten verbunden ist. In der ICC geht es dahingegen vor allem darum, die fest vorhandenen Ressourcen angemessen zu verteilen. Dabei müssen neben den Standardrechenressourcen wie CPU und Arbeitsspeicher der Worker-Nodes auch spezielle GPU- und Speicher-Ressourcen bei entsprechenden Konzepten berücksichtigt werden.

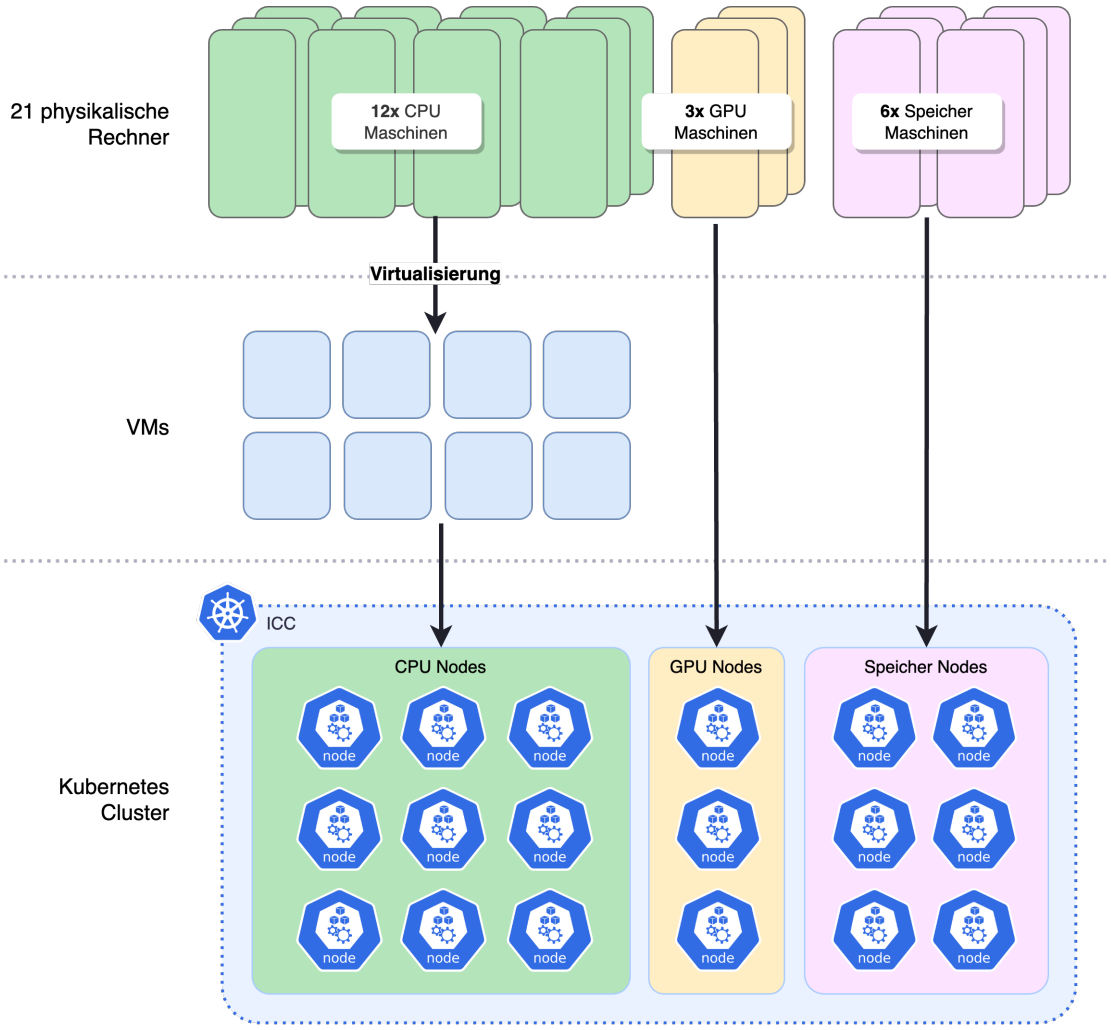

Abbildung 3.1: Aufbau der ICC

Abbildung 3.2: Namespaces der ICC

3.1.2 Anwendungen und Multi-Tenancy der ICC

Abbildung 3.2 zeigt alle Namespaces in der ICC. Die ICC ist in mehrerer Hinsicht ein Multi-Tenancy Cluster. Zum einen laufen generell unterschiedliche Applikationen in einem Cluster, zum anderen gibt es darunter auch eigene Multi-Tenancy Anwendungen, welche zusätzlichen Tenants Zugriff auf das Cluster geben.

Die Multi-Tenancy Anwendungen sind *Kiosk* [60], *JuypterHub* [78] und der *GitLab Runner* [89]. Die restlichen Anwendungen unterstützen die Administration des Clusters und werden als *Systemdienste* bezeichnet. Darunter befinden sich Anwendungen zur Bereitstellung erweiternder Ressourcen (*Rook und Ceph* [85, 81], *GPU Operator* [70]), zur Überwachung des Clusters (*Prometheus* [80], *Grafana* [57]) oder auch der Authentifizierungsdienst der ICC (*ICC System*). Über diesen Authentifizierungsdienst ist es möglich, sich als Mitglied des Informatikdepartments per HAW-Kennung über ein Webinterface bei der ICC anzumelden und anschließend das Cluster über Kiosk direkt zu nutzen.

Auch die anderen beiden Multi-Tenancy Anwendungen JupyterHub und GitLab Runner können von allen Mitgliedern des Informatikdepartments genutzt werden. Da vor allem durch die drei Multi-Tenancy Anwendungen viele Nutzende Zugriff auf die Ressourcen des Clusters bekommen können, werden diese im Folgenden genauer betrachtet. Konkret soll gezeigt werden, welche Funktionen sie den Tenants bieten und welche Rechenressourcen und Clusterressourcen im Betrieb erstellt werden. Zudem wird die Nutzung auf die ICC

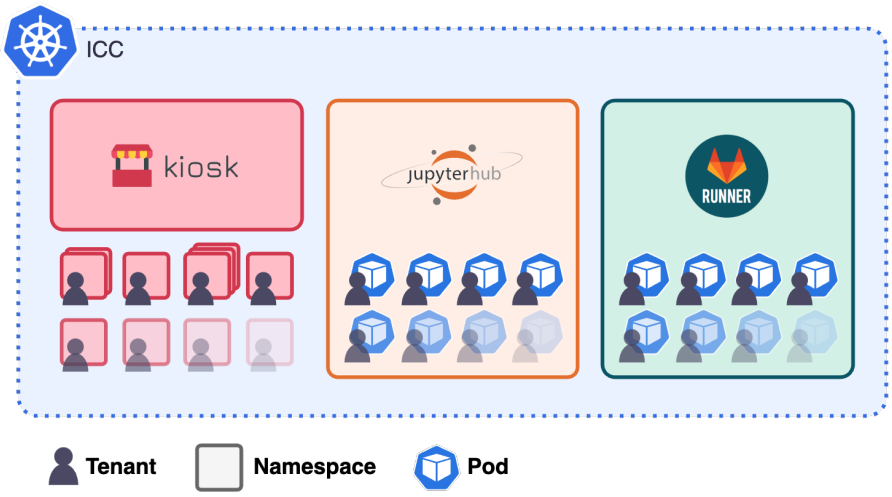

Abbildung 3.3: Für die Tenants der Multi-Tenancy Anwendungen werden Namespaces oder Pods erstellt

bezogen analysiert, indem beschrieben wird, *wie* auf das Angebot zugegriffen wird und wofür es meist genutzt wird.

Wie in Abbildung 3.3 gezeigt, werden durch die Nutzung von Kiosk ganze Namespaces erstellt, wohingegen durch JupyterHub und den GitLab Runner nur einzelne Pods in den entprechenden Namespaces angelegt werden.

Kiosk

Kiosk wurde als Multi-Tenancy Erweiterung für Kubernetes entwickelt. Die Kernfunktion von Kiosk ist es, den Tenants eine eingeschränkte Sicht auf die Namespaces des Clusters zu geben. Da Namespaces eine Cluster-Ressource sind, ist es normalerweise mit RBAC-Regelungen nur möglich, alle oder keine Namespaces zu sehen. Kiosk umgeht diese Einschränkung mit einer eigenen API-Ressource namens `Spaces`. Spaces repräsentieren genau einen Namespace und werden nicht durch Kubernetes selbst, sondern von Kiosk verwaltet. Kiosk übernimmt damit die Logik, den Tenants nur die ihnen zugewiesenen Spaces und damit eine eingeschränkte Sicht auf die Namespaces des Clusters zu zeigen [60]. Dabei ist es den Tenants möglich, mehrere Namespaces in Form von Spaces anzulegen. Innerhalb dieser können sie anschließend selbstständig Workloads mit allen verfügbaren Rechenressourcen erstellen. Kiosk selbst bietet dabei keine eigene Zugangsverwaltung. Dafür hat die ICC einen eigenen Authentifizierungsdienst entwickelt, bei dem sich die Tenants mit ihrer HAW-Kennung anmelden können. Anschließend wird ein `Account` für sie erstellt und sie erhalten eine Konfigurationsdatei mit einem Token, über

den sie sich beim API-Server der ICC authentifizieren können [88, p. 63] [56]. Kiosk wird von Studierenden und Forschungsgruppen genutzt, um direkt in der ICC zu arbeiten und eigene Anwendungen im Cluster laufen zu lassen.

JupyterHub

JupyterHub ermöglicht es den Tenants, sogenannte *Notebooks* über eine Weboberfläche zu erstellen [79]. In diesen Notebooks können unterschiedlichste Funktionalitäten abgebildet werden. Unter anderem kann Python-Code, berechnete Inputs, Outputs oder auch Bilder in einem Dokument angelegt werden. Codeabschnitte können über das integrierte Interface ausgeführt und die Ergebnisse anschließend direkt angezeigt werden [59]. JupyterHub wird von den Tenants vor allem zur Datenanalyse und für Machine-Learning-Prozesse genutzt. Da das Arbeiten mit großen Datenmengen und das Trainieren von Machine-Learning-Modellen sehr rechenintensiv ist, soll den Tenants mit JupyterHub vor allem der Zugang zu den GPU-Ressourcen der ICC ermöglicht werden. Im Cluster wird für jedes so angefragte Notebook ein eigener Pod mit festen Rechenressourcen, darunter besonders einer GPU-Ressource, im `jupyterhub` Namespace erstellt. Das Cluster selbst ist dabei für die Tenants nicht sichtbar. Sie nutzen JupyterHub über die angebotene Weboberfläche als Software as a Service (SaaS), wobei alle Tenants letztlich auf eine gemeinsame Instanz der Anwendung zugreifen. Tenants können sich dabei direkt über die Weboberfläche mit ihrer HAW-Kennung anmelden. Anschließend können sie aus vordefinierten Konfigurationen zu den verwendeten Grafikkarten auswählen. Die Tenants von JupyterHub sind zum einen Studierende, die innerhalb eines Kurses vorgegebene Aufgaben mithilfe von JupyterHub bearbeiten. Aber auch außerhalb eines konkreten Kurses kann JupyterHub von allen mit einer gültigen HAW-Kennung genutzt werden.

GitLab Runner

Mit dem GitLab Runner ist es möglich, in GitLab angelegte CI-Pipelines über ein Cluster laufen zu lassen. Dafür hört der Runner auf CI Jobanfragen der GitLab-Instanz. Wird so ein Job angefragt, erstellt der Runner über die Kubernetes API einen Pod in seinem Namespace, um den CI-Job vom Cluster ausführen zu lassen [12]. Die erstellten Pods können theoretisch alle verfügbaren Rechenressourcen anfragen. In der ICC ist die Nutzung der GitLab Runners für alle Mitglieder des Departments Informatik möglich. Das Angebot kann von den Tenants beispielsweise für eine in GitLab versionierte Anwendung genutzt werden. Diese kann über eine solche CI-Pipeline automatisch in einem Kiosk-Space bereitgestellt wird.

Vor allem die Multi-Tenancy Anwendungen erlauben einer potenziell hohen Anzahl an Tenants einen Zugriff auf das Cluster und müssen daher für die Quotierungs- und Priorisierungskonzepte betrachtet werden. Dabei bieten die Anwendungen den Tenants direkten oder indirekten Zugriff auf die Infrastruktur und Ressourcen des Clusters. Dadurch erhalten die Tenants unterschiedlich viel Kontrolle über die tatsächlich im Cluster erstellten Workloads. In Kiosk können die Tenants frei Ressourcenanfragen wählen, während diese in JupyterHub durch die angebotenen Konfigurationen vorgegeben sind. Der GitLab Runner erlaubt den Tenants eigene Ressourcenanforderungen in den Pipelinekonfigurationen angeben. Diese Unterschiede müssen bei der Konzeption berücksichtigt werden.

3.1.3 Nutzungsgruppen der ICC

Die ICC wird von unterschiedlichen Personen als Tenants genutzt. Um diese für die weitere Arbeit zu konkretisieren, werden diese in zwei Gruppen aufgeteilt. Zum einen gibt es das ICC-Team, das die ICC als System betreut. Das ICC-Team kümmert sich dabei um administrative Aufgaben zum Betrieb des Clusters, wie die Konfiguration der Nodes, die Bereitstellung der Ressourcen oder die Installation und Wartung der verschiedenen Anwendungen. Daneben gibt es die ICC-Nutzenden. Diese Gruppe besteht aus allen Mitgliedern des Departments Informatik, die die Anwendungen der ICC nutzen und damit direkt oder indirekt die Ressourcen des Clusters nutzen. Durch die unterschiedlichen Nutzungsgruppen ergeben sich unterschiedliche Anforderungen an das Konzept. Das ICC-Team selbst ist dabei als vertrauenswürdig zu betrachten, weshalb die exklusiv durch das Team verwalteten Systemdienste keine Vorgaben zur Quotierung und Priorisierung benötigen. Die ICC-Nutzenden dahingegen sind generell als nicht vertrauenswürdig anzusehen, sei es aus Unwissenheit oder tatsächlichen schädlichen Absichten. Vor allem diese Nutzungsgruppe soll durch die Konzepte angemessen eingeschränkt werden.

3.1.4 Bestehende Regelungen zur Ressourcenverteilung

Um neue Anforderungen an die Quotierung und Priorisierung aufzustellen, müssen auch die bestehenden Regelungen zum Ressourcenverbrauch der Multi-Tenancy Anwendungen in der ICC betrachtet werden.

Kiosk
In Kiosk werden aktuell keine Limits durchgesetzt. Das bedeutet, dass es den ICC-

Nutzenden hier freigestellt ist, Limits für ihre Pods zu definieren. Tun sie das nicht, laufen die erstellten Container mit unbegrenzten Rechenressourcen [36]. Um Ressourcen wieder freizugeben, existiert in der ICC ein Automatismus, der alle Clusterressourcen eines Accounts nach 90 Tagen ohne Anmeldung löscht [56].

JupyterHub
Alle auswählbaren Konfigurationen definieren Limits für die im Hintergrund durch JupyterHub erstellten Pods, was einen unbegrenzten Ressourcenverbrauch verhindert. Damit die limitierten GPU-Ressourcen des Clusters nicht durch inaktive JupyterHub-Pods besetzt werden, existieren mehrere Mechaniken. Die Nutzenden können ihrer Server aktiv über die Weboberfläche abschalten, wodurch der Pod beendet und die Ressourcen freigegeben werden. Da dies jedoch häufig vergessen wird, werden speziell die für Kurse der Hochschule vorgesehenen JupyterHub-Pods zusätzlich jeden Werktag der Woche automatisch um acht Uhr abgeschaltet. Damit soll sichergestellt werden, dass die GPU-Ressourcen auf jeden Fall zum Start der Kurse frei sind. Zudem prüft JupyterHub alle zehn Minuten, ob das gestartete Notebook noch als Fenster im Browser geöffnet ist. Wenn Jupyterhub das Notebook länger als acht Stunden nicht erreichen kann, dann wird dieses automatisch beendet [56]. Als letzte Maßnahme werden auch regelmäßig manuell JupyterHub-Pods abgeschaltet, wobei mehrere selbst definierte Metriken zur Bewertung der GPU-Auslastung zum Einsatz kommen.

GitLab Runner
Die GitLab-Jobs, die von einem Account angefragt werden können, sind auf eine feste Minutenanzahl pro Tag begrenzt. Zudem sind Limitierungen im Einsatz, wie viel Ressourcen gleichzeitig angefragt werden können.

Die vorhandenen Regelungen zeigen, dass vor allem in Kiosk eine Gefahr durch unlimitierte Ressourcenanfragen besteht. Die ICC-Nutzenden können über Kiosk eigenständig so viele Namespaces und Workloads mit so vielen Ressourcen wie sie wollen starten. Definieren sie gar keine Limits, laufen die Pods dazu standardmäßig mit unbegrenzten Ressourcen. Die ICC-Nutzenden, die über JupyterHub indirekt auf das Cluster zugreifen, können nur die durch die angebotenen Konfigurationen vorgegebenen Notebooks starten. Die im Hintergrund erstellten Pods sind über JupyterHub bereits mit einem Limit definiert. Wegen der Exklusivität der angefragten GPU-Ressourcen existieren schon mehrere Mechaniken, durch welche inaktive Pods abgeschaltet werden sollen, um die Ressourcen wieder freizugeben. Die vielen parallel existierenden Mechaniken zeigen jedoch, dass es ein Problem darstellt, die inaktive Pods und deren GPU-Auslastung zuverlässig zu erken-

nen. Der GitLab Runner ist durch das Minuten- und Ressourcenlimit aktuell ausreichend quotiert und bedarf keiner weiteren Begrenzungen. In keiner der Multi-Tenancy Anwendungen bestehen Regelungen zur Priorisierung von bestimmten Ressourcenanfragen.

3.2 Funktionen und Qualitäten

Um konkreten Ziele für diese Arbeit zu definieren, werden im folgenden funktionale und nicht-funktionale Anforderungen an das entwickelte Konzept aufgestellt. Dabei werden die funktionalen Anforderungen durch Use-Cases abgedeckt, welche die generellen Funktionen der Lösung definieren. In den anschließenden Qualitätszielen werden die nicht-funktionalen Anforderungen durch konkrete Szenarien festgehalten. Die Szenarien sollen helfen, das erstellte Konzept zu bewerten.

3.2.1 Use-Cases

Um die Funktionalität und Anforderungen des Systems klar zu definieren, wurde in Zusammenarbeit mit den Administratoren der ICC Use-Cases erstellt, die von der Lösung abgedeckt werden sollen (Abbildung 3.4).

Ein wichtiger Punkt war der Wunsch, Limits im Cluster zu definieren. Konkret soll es möglich sein, Namespaces in ihren Ressourcen zu quotieren (UC-1), sodass alle Workloads darin zusammen genommen, nicht mehr als ein im Voraus festgelegtes Limit nutzen können. Dadurch soll verhindert werden, dass ein Namespace mehr als seinen fairen Anteil der Ressourcen des Clusters nutzt. Ergänzend sollen auch logisch zusammengehörige Namespaces durch ein gemeinsames Limit quotiert werden können (UC-2). Damit unterliegen alle Workloads innerhalb dieser Namespaces einem aggregierten Limit. In diesem Zuge sollen auch generelle Limitierungen für Pods (UC-3) ermöglicht werden, damit nicht ein Pod die Ressourcen seines gesamten Namespaces oder der zusammengehörigen Namespaces monopolisieren kann. Sind Limits definiert, kümmert sich Kubernetes darum, diese durchzusetzen. Deswegen soll speziell das Konfigurieren von festgelegten Limits ermöglicht werden. Um zusätzlich die Wichtigkeit von eingehenden Pods im Bezug zu anderen zu bestimmen, soll es die Möglichkeit geben, Prioritäten festzulegen (UC-4). Diese Informationen sollen dann zum Scheduling der Pods verwendet werden, wobei höher priorisierte Pods bevorzugt behandelt werden.

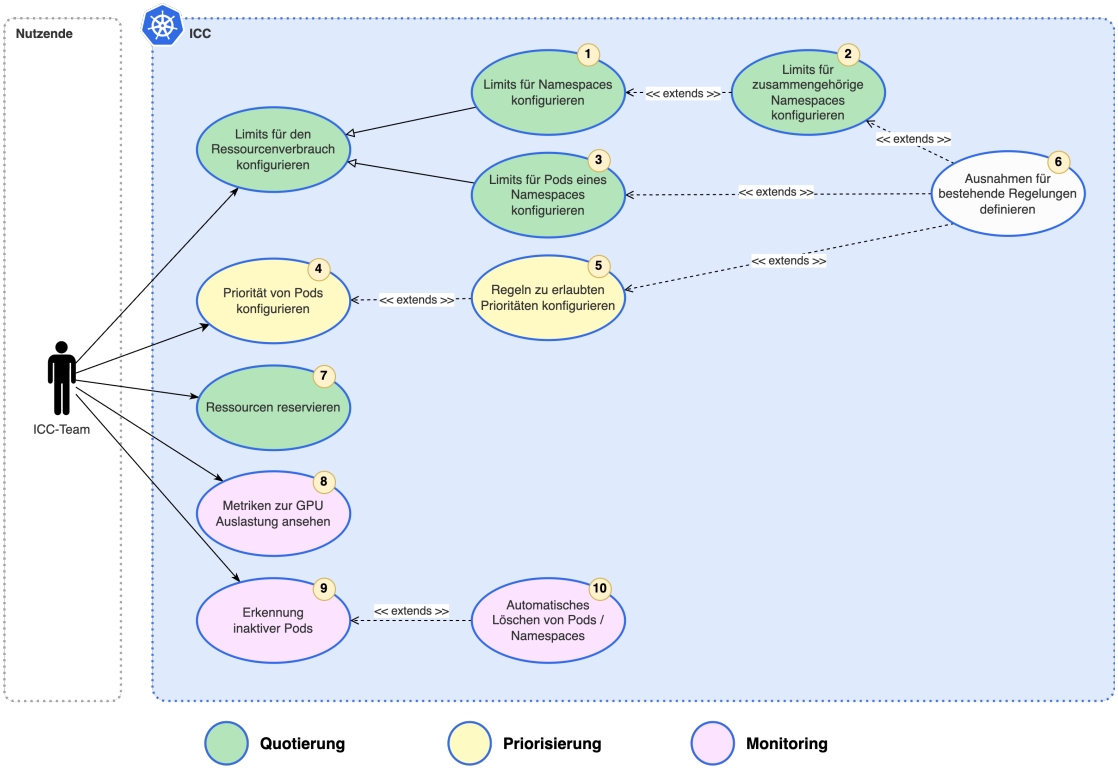

Abbildung 3.4: Use-Case-Diagramm

Aus diesen ersten Use-Cases haben sich noch weitere erweiternde Funktionalitäten ergeben. Zum einen soll es möglich sein, Regelungen zu den erlaubten Prioritäten zu erstellen (UC-5). Damit soll es nur festgelegten Nutzenden oder Namespaces erlaubt sein, bestimmte Prioritäten für ihre Pods festzulegen. Dadurch soll ein Missbrauch der Prioritätsfunktion verhindert werden. Für die vorgestellten Regelungen soll es auch möglich sein, Ausnahmen festzulegen (UC-6). Dadurch soll die Lösung auch im Nachhinein anpassbar bleiben. Ein speziellerer Fall von Quotierung stellt das Reservieren von Ressourcen (UC-7) dar. Hierbei sollen Ressourcen nur für bestimmte Namespaces zugänglich sein. Dies soll ermöglichen, Ressourcen für geplante Workloads freizuhalten, damit die Workloads garantiert laufen können.

Als Erweiterung einer aktiven Quotierung sind noch Use-Cases zum passiven Monitoring der Ressourcen aufgekommen. Zum einen kam der Wunsch nach zuverlässigeren Metriken zur GPU-Auslastung (UC-8) auf. Diese spezielle Ressource wird nicht von den Standardwerkzeugen zur Überwachung der Clusterauslastung von Kubernetes erfasst. Da die limitierten GPU-Ressourcen jedoch vor allem durch JupyterHub stark belastet sind,

soll hier eine Metrik helfen, den Bedarf der ICC zu überwachen, um zukünftig weitere Maßnahmen zu koordinieren. Außerdem soll es möglich sein, inaktive Pods zu erkennen (UC-9). Viele ICC-Nutzende sind bei der Nutzung von Kiosk initial nicht vertraut, in einem Cluster wie Kubernetes zu arbeiten oder wissen beim Nutzen von JupyterHub gar nicht, dass sie dadurch Ressourcen reservieren. Dabei kommt es nicht selten vor, dass laufende Workloads vergessen werden. Wenn inaktive Pods erkannt werden können, kann in einem nächsten Schritt auch ein Automatismus erstellt werden (UC-10), der so erkannte Pods zu regelmäßigen Zeiten löscht. Dies soll helfen, von ICC-Nutzenden gestartete und vergessene Workloads zu beenden und die unnötig besetzten Ressourcen wieder freizugeben. Diese Monitoring-Use-Cases beschäftigen sich dabei indirekt mit der Quotierung, indem sie die Ressourcenauslastung des Clusters greifbarer machen und versuchen ungenutzte, aber reservierte Ressourcen wieder freizugeben.

Die Konzepte der Lösung sollen dem ICC-Team helfen, die Ressourcen im Cluster strukturierter zu verteilen. Besonders die direkte Quotierung und Priorisierung sollen helfen, den Ressourcenverbrauch der Multi-Tenancy Anwendungen besser zu kontrollieren und so entstehende Ressourcenkonflikte zu reduzieren. Eine systematische Darstellung der Use-Cases ist in Anhang Abschnitt A.1 zu finden.

3.2.2 Qualitätsziele

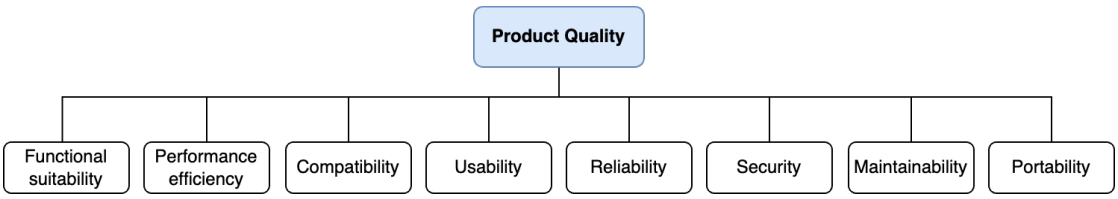

Abbildung 3.5: Qualitätsmerkmale der ISO/IEC 25010 Norm

Neben den im vorherigen Unterabschnitt 3.2.1 genannten funktionalen Anforderungen werden auch Anforderungen an die Qualität und Merkmale der Lösung gestellt. Diese nicht-funktionalen Anforderungen sollen beschreiben, **wie** die Lösung ihre Funktionen ausführen soll. Dafür werden im Folgenden Qualitätsziele gemäß der ISO/IEC 25010 Norm festgehaltenen. Diese werden durch Qualitätsszenarien ergänzt, die zur Veranschaulichung sowie als Metrik zur Erreichung der Qualitätsziele dienen sollen. Da es für die von der ISO/IEC festgelegten Qualitäten keine offizielle deutsche Übersetzung gibt, werden die originalen englischen Begriffe verwendet. Abbildung 3.5 zeigt die offiziellen

Tabelle 3.1: Qualitätsziele

Qualitätsziel	Motivation und Erläuterung
Functional Suitability: Bessere Ressourcenverteilung	Die Konzepte helfen, die verfügbaren Ressourcen der ICC strukturierter zu verteilen.
Compatibility: Existierende Anwendungen berücksichtigen	Die Konzepte funktionieren im Zusammenspiel mit den bereits existierenden Anwendungen und stören diese nicht in ihrer Arbeitsweise.
Usability: Schnelle Einarbeitung	Die Konzepte sind einfach und verständlich anzuwenden und zu bedienen.
Maintainability: Anpassbarkeit durch Mechaniken	Die Konzepte können im Nachhinein modifiziert werden, um besser auf aktuelle Situationen angepasst zu werden.

acht Qualitätsmerkmale des *Product Quality Model* der ISO/IEC 25010 Norm. Im Rahmen dieser Arbeit soll sich von diesen Merkmalen auf vier als Ziele für das Konzept konzentriert werden (Tabelle 3.1).

Im Folgenden werden die Qualitätsziele durch konkrete Szenarien charakterisiert, um zu beschreiben, wie sich die Lösung in bestimmten Situationen verhalten soll. Die Qualitätsszenarien beziehen sich dabei auf bestimmte Unterkategorien der ISO/IEC 25010 Qualitätsmerkmale.

Die zuerst definierten Qualitätsszenarien zur Korrektheit der Lösung mit der ID FC (von **F**unctional Suitabillity: **C**orrectness) in Tabelle 3.2, bieten messbare Szenarien für die durch die Use-Cases definierten funktionalen Anforderungen.

Tabelle 3.2: Functional Suitability: Correctness

ID	Szenario
FC-01	Überschreitet ein Request im entsprechenden Namespace eingestellte Limits, wird der Request abgelehnt und die Workload wird nicht erstellt.
FC-02	Überschreitet ein Request ein für mehrere Namespaces eingestelltes Limit, wird der Request abgelehnt und die Workload nicht erstellt.
FC-03	Überschreitet ein Request nach einem Pod die für ihn eingestellten Limits, wird der Request abgelehnt und der Pod nicht erstellt.
FC-04	Ein angefragter Pod mit höherer Priorität wird früher einem Node zugeteilt, als zeitlich vor ihm angefragte Pods mit niedriger Priorität.
FC-05	Wenn eine Priorität in einem bestimmten Namespace ausgeschlossen wurde, kann dort kein Pod mit dieser Priorität erstellt werden.
FC-06	Wird ein Pod von einem bestimmten Ressourcenlimit ausgenommen, wird die Ausnahmeregel bevorzugt.
FC-07	Wird ein Pod von einer Beschränkung der Priorität ausgeschlossen, wird die Ausnahmeregel bevorzugt und der Pod darf die im Namespace ausgeschlossene Priorität verwenden.
FC-08	Wurden Ressourcen reserviert, werden diese nur ausgewählten angefragten Workloads zugeteilt.
FC-09	Es können aussagekräftige Metriken zur GPU-Auslastung des Cluster angezeigt werden. Dabei ist ersichtlich, welche konkreten Pods welche Grafikkarte nutzen und wie ausgelastet diese Grafikkarten sind.
FC-10	Als inaktiv erkannten Pods können identifizierbar aufgelistet werden.
FC-11	Ein Pod, der durch festgelegte Metriken als inaktiv erkannt wird, wird spätestens am nächsten Sonntag gelöscht.

Die weiteren Qualitätsziele beziehen sich nicht auf konkrete Use-Cases, sondern besondere Aspekte, die bei der Konzeption berücksichtig werden sollen.

Tabelle 3.3: Functional Suitability: Appropriateness

ID	Szenario
FA-01	Alle ICC-Nutzenden haben einen **quotierten** Zugriff auf das Cluster.
FA-02	Das ICC-Team kann zeitkritische Pods schneller schedulen. Genauso können zeitunkritische Pods zurückgestellt werden, um keine dringlicher angefragten Pods zu blockieren.
FA-03	Durch die Konzepte werden die Pods von JupyterHub bevorzugt, die aufgrund eines geplanten Hochschulkurses angefragt werden.

Tabelle 3.4: Compatibility: Co-Existence

ID	Szenario
CC-01	Für die ICC-Nutzenden bleibt die Interaktion mit Kiosk unverändert, solange sie ihre Anfragen innerhalb der festgelegten Grenzen halten, keine unzulässigen Prioritäten setzen und keine Ressourcen anfordern, die nicht für sie reserviert sind.
CC-02	Für die Nutzenden bleibt die Interaktion mit JupyterHub unverändert, so lange die Anfrage keine Ressourcen anfordert, die nicht für sie reserviert sind.

Tabelle 3.5: Usability: Learnability

ID	Szenario
UL-01	Das ICC-Team kann die Konzepte selbstständig, effektiv und effizient anwenden, da die Funktionsweisen leicht verständlich sind und keine längere Einarbeitung voraussetzen.
UL-02	ICC-Nutzende werden so weit wie möglich über die aktiven Einschränkungen informiert.

Tabelle 3.6: Usability: Operatability

ID	Szenario
UO-01	Die Limits für Pods können für einen Namespace gesammelt eingestellt werden.
UO-02	Die Metriken zur GPU-Auslastung können visuell verständlich dargestellt werden, statt über reine Tabellen und Zahlenwerte.
UO-03	Das Definieren von Ausnahmen ist eindeutig mit der betroffenen Regelung verknüpft. Eine Einsicht in die Regelung zeigt direkt, ob eine Ausnahme definiert ist.

Tabelle 3.7: Maintainability: Modifiability

ID	Szenario
MM-01	Limits für Pods und Namespaces können durch maximal eine Anpassung an einer definierten Stelle effektiv angepasst werden.
MM-02	Prioritäten und Regeln zu erlaubten Prioritäten können durch maximal eine Anpassung an einer definierten Stelle effektiv angepasst werden.
MM-03	Das Konzept zur Erkennung inaktiver Pods bietet eine Mechanik, mit der die Metrik zur Erkennung im Nachhinein angepasst werden kann.

Die restlichen Qualitäten der ISO/IEC 25010 Norm wurden in dieser Arbeit nicht als primäre Ziele verfolgt. Die Qualitätsziele haben Einfluss auf Entscheidungen, weswegen es wichtig ist, passende Ziele zu definieren und festzuhalten. Die zusätzlichen Qualitätsszenarien konkretisieren die Qualitätsziele, machen diese messbar und helfen letztlich zu entscheiden, inwiefern sie erfüllt wurden [1].

3.3 Anforderungen

Auch wenn die ICC mehrere Charakteristiken der Multi-Tenancy Architektur aufweist, sind für eine bessere Ressourcenverteilung vor allem die konkreten Multi-Tenancy Anwendungen zu betrachten. Diese ermöglichen vielen potenziell nicht vertrauenswürdigen Nutzenden Zugang zum Cluster und damit den Ressourcen. Die Konzepte der Arbeit sollen sich auf Quotierungs- und Priorisierungsmaßnahmen für diese Anwendungen konzentrieren. Dabei sollen für die Quotierung aktiv Limits durchgesetzt werden und ergänzende Monitoringmaßnahmen zur passiven Kontrolle eingesetzt werden. Bei der Priorisierung soll ein Konzept zur bevorteilten Ressourcenzuteilung für entsprechend gekennzeichnete Pods erarbeitet werden. In der ICC gibt es die drei Multi-Tenancy Anwendungen Kiosk, JupyterHub und GitLab Runner. Dabei haben vor allem Kiosk Nutzende einen direkten Zugriff auf das Cluster und können aktuell uneingeschränkt Ressourcen anzufordern, was durch entsprechende Gegenmaßnahmen eingeschränkt werden soll. JuypterHub und der GitLab Runner bieten den Nutzenden nur indirekten Zugriff auf das Cluster. Aus diesem Grund existieren bereits anwendungsinterne Mechaniken, die angefragten Ressourcen zu beschränken. Diese Ressourcenverteilung des Clusters kann jedoch von Priorisierungskonzepten für diese Anwendungen profitieren. Die Monitoringmaßnahmen beziehen sich auf das gesamte Cluster. Hier soll vor allem die Auslastung der häufig benötigten, aber stark limitierten GPU-Ressourcen besser einsehbar werden. Zusätzlich sollen inaktive Pods identifizierbar sein und in einem nächsten Schritt automatisch gelöscht werden. Es wurden Qualitätsziele mit entsprechenden Szenarien aufgestellt, welche die Lösung erfüllen soll. Hierbei stehen die Qualitäten der funktionalen Eignung im Vordergrund, die helfen sollen zu bewerten, ob die aufgestellten Use-Cases im Bezug auf die ICC erfüllt werden. Aber auch die Kompatibilität mit den bereits existierenden Anwendungen des Clusters, eine schnelle Einarbeitung und die Anpassbarkeit sollen am Ende als Kriterien für die erarbeiteten Lösungen dienen.

4 Verfügbare Konzepte und Erweiterungen

Im folgenden Kapitel soll untersucht werden, wie man die aufgestellten Funktionalitäten und Qualitäten in der ICC umsetzen kann. Dabei werden verschiedene Ansätze betrachtet. Als erstes werden bereits vorhandene Mechaniken von Kubernetes und den Multi-Tenancy Anwendungen selbst betrachtet. Letztlich werden noch externe Erweiterungen vorgestellt, die zur Realisierung noch fehlender Funktionalitäten herangezogen werden oder hilfreiche Ergänzungen bieten können.

4.1 Verwandte Arbeiten

Da die Kombination aus aufgestellten Anforderungen und die Architektur der ICC sehr individuell sind, existiert keine einzige Arbeit, deren Lösung alle Aspekte betrachtet. Stattdessen sind die bisherigen Lösungen oft sehr spezialisiert und konzentrieren sich auf Teilaspekte der Ressourcenverteilung.

Mehrere Arbeiten gehen schon einen Schritt weiter und beschäftigen sich damit, die Werte der Ressourcenlimits zu optimieren [17, 15]. Dies ist definitiv sinnvoll, jedoch ist das Ziel dieser Arbeit in einem ersten Schritt fehlende Limits generell zu etablieren. Auf der anderen Seite gibt es Arbeiten, die sich bei der Verbesserung der Ressourcenverteilung auf eine spezielle Art von Anfragen spezialisieren [20, 19, 59]. Ziel dieser Arbeit ist es jedoch, ein erstes umfassendes Konzept für die ICC und ihre unterschiedlichen Anwendungen, vor allem die verschiedenen Multi-Tenancy Anwendungen zu schaffen.

Die spezialisierten Ergebnisse der Arbeiten sind im Rahmen eines gesamtheitlichen Konzepts zur Quotierung und Priorisierung nicht anwendbar, können jedoch für die zukünftige Entwicklung des hier initial erarbeiteten Konzepts zur Ressourcenverteilung herangezogen werden.

4.2 Kubernetes' interne Mechaniken

Für die Quotierung und Priorisierung ist vor allem der Schedulingprozess aus Unterabschnitt 2.2.2 zu betrachten. In erster Instanz kann dabei der API-Server eine Quotierung durchsetzen und der Scheduler anschließend gesetzte Prioritäten berücksichtigen.

4.2.1 Quotierung durch Admission Control

Nachdem ein Pod mit gesetzten Ressourcenlimits im Cluster gestartet wurde, kümmert sich `kubelet` darum, dass diese Limits eingehalten werden. Um eine Limitierung der Werte selbst – sei es pro Pod oder für ganze Namespaces – durchzusetzen, müssen die angefragten Werte überprüft werden *bevor* die Pods persistiert werden.

Um bestimmte Eigenschaften von API-Objekten zu kontrollieren, bevor sie persistiert werden, verweist die Kubernetes Dokumentation auf die Erweiterungsmöglichkeiten der Zugriffskontrolle des API-Servers [29]. Wie in Abbildung 4.1 dargestellt, durchläuft ein beim API-Server eingegangener Request nach erfolgreicher Authentifizierung und Autorisierung des Clients noch drei weitere Stationen. Diese dienen der Kontrolle der Eigenschaften des angefragten Objektes, bevor es im etcd festgehalten wird [28].

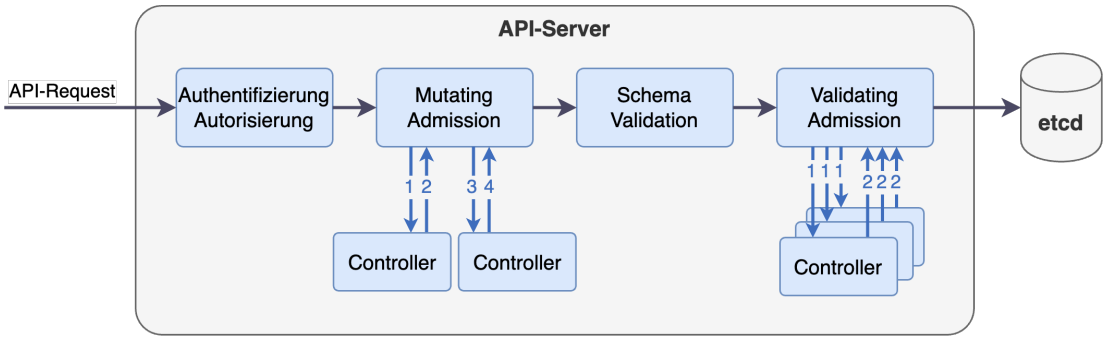

Abbildung 4.1: Phasen der Zugriffskontrolle des API-Servers

Die *Schema Validation* überprüft, ob die Struktur und Syntax des Objektes mit dem erwarteten Format der API übereinstimmt. Die tatsächlichen Eigenschaften des Objekts können durch definierte Regeln in der *Mutating Admission* angepasst und letztlich in der *Validating Admission* überprüft werden. In diesen Phasen leitet der API-Server die Objekte an die entsprechenden mutierenden und validierenden *Admission Controller* weiter. *Mutierende* Controller können die Eigenschaften der angefragten Objekte verändern,

während *validierende* Controller die Objekte aufgrund der Eigenschaften genehmigen oder ablehnen. Dazu sei erwähnt, dass auch mutierende Controller die Requests ablehnen können [25].

Die Reihenfolge der Controller garantiert dabei, dass die validierenden Controller den tatsächlich finalen Zustand des angefragten Objekts vorliegen haben. Dabei müssen die mutierenden Controller sequenziell arbeiten, um keine Konflikte durch Änderungen an denselben Eigenschaften zu bekommen. Die validierenden Controller können hingegen parallel aufgerufen werden, da diese nur lesend arbeiten.

Kubernetes bietet von sich aus eine Reihe von integrierten Admission Controllern an, die innerhalb des API-Servers laufen. Darunter gibt es auch zwei, die sich mit der Limitierung von Ressourcen für Namespaces beziehungsweise Pods beschäftigen [25].

ResourceQuota

Der *ResourceQuota* Admission Controller ist ein validierender Controller, der ein festgelegtes Limit für einen Namespace durchsetzt. Dabei arbeitet er mit einem im betroffenen Namespace definierten gleichnamigen API-Objekt `ResourceQuota` zusammen. Bei einem neuen Request für den Namespace gleicht der Controller die angefragten Ressourcen mit den bereits belegten Ressourcen des Namespaces ab. Neben der Anzahl verschiedener Namespace-Ressourcen können so auch die gesammelten Rechenressourcen eines Namespaces limitiert werden [44]. Würde mit der neuen Workload das im `ResourceQuota` Objekt festgelegte Limit überschritten werden, wird der Request abgelehnt [25]. Zu den standardmäßigen Rechenressourcen CPU und Memory können auch erweiternde Ressourcen wie GPUs limitiert werden. Für den angefragten persistenten Speicher können ebenfalls verschiedene Limitierungen gesetzt werden. Unter anderem, wie viel `PersistentVolumeClaims` im Namespace definiert werden dürfen und wie viel Speicher diese in Summe anfragen dürfen. Um festzustellen, wie viel Ressourcen aktuell im Namespace verbraucht sind, arbeitet der ResourceQuota-Controller nicht mit Echtzeitdaten, sondern mit den definierten Request- und Limitwerten der Pods[1]. Deswegen ist zu beachten, dass für die korrekte Funktion des Controllers alle Requests die limitierten Eigenschaften definieren müssen.

LimitRanger

Ein `ResourceQuota` beschränkt zwar den Ressourcenverbrauch des gesamten Name-

[1] Der ResourceQuota-Controller arbeitet mit einem weiteren Controller zusammen, der die Request- bzw. Limitwerte aller Objekte eines Namespaces festhält [5, pkg/controller/resourcequota/doc.g l.388].

spaces, jedoch könnte ein einziger Pod die Ressourcen für sich beanspruchen. Um dem entgegenzuwirken, kann der *LimitRanger* Admission Controller genutzt werden. Dieser kann die Werte für Ressourcenanfragen der angefragten Pods kontrollieren und setzen. Analog zum ResourceQuota-Controller arbeitet auch der LimitRanger-Controller mit einem dazugehörigen API-Objekt `LimitRange` innerhalb des Namespaces zusammen. Neben den Ressourcenanfragen zu CPU, Memory und GPU von Pods können auch die Speicheranfragen von `PersistentVolumeClaims` kontrolliert werden [36]. Konkret können für die Anfragen minimale und maximale Werte in der `LimitRange` definiert werden. Verstoßen die Requests dagegen, lehnt der Controller diese ab. Zusätzlich können automatisch Werte gesetzt werden, wenn die Requests keine eigenen definieren. Damit kann mit einer entsprechend definierten `LimitRange` garantiert werden, dass die quotierten Werte einer `ResourceQuota` gesetzt sind und damit der ResourceQuota-Controller das Gesamtlimit durchsetzen kann.

Mit den beiden integrierten Admission Controllern **LimitRanger** und **ResourceQuota** können Ressourcenlimitierungen für Pods und gesamte Namespaces realisiert werden.

4.2.2 Priorisierung durch Pod Priority

Auch eine Lösung zur Priorisierung muss greifen, bevor ein Pod auf einem Node persistiert wird. Die Komponente des Schedulingprozesses, die davor über die Sortierung der Requests entscheidet, ist der *Scheduler*.

Sobald ein Request durch die Admission Control genehmigt wurde, wird der gewünschte Zustand im etcd persistiert. Anschließend kümmert sich der Scheduler `kube-scheduler` darum, einen Node zu finden, der die Anforderungen der Pods erfüllt. Definieren die Pods keine Prioritäten, verwendet der `kube-scheduler` den First Come First Serve (FCFS) Algorithmus, um die Pods nach dem Zeitstempel ihres Requests zu sortieren. Gibt ein Pod jedoch eine Priorität an, arbeitet der `kube-scheduler` mit einem Plug-in, das statt der standardmäßigen FIFO-Sortierung eine Queue-Sortierung anwendet. Dabei werden die Pods in einem ersten Schritt nach ihren Prioritäten sortiert und anschließend innerhalb derselben Prioritäten nach ihren Zeitstempeln [2, 35].

Die Pods können dafür den Namen einer Prioritätsklasse `spec.priorityClassName` angeben, die davor im Cluster erstellt werden muss. Die sogenannten `PriorityClass` Objekte sind Cluster-Ressourcen. Dabei wird der Name der `PriorityClass` auf einen Integer-

Wert abgebildet, wobei höhere Werte eine höhere Priorität anzeigen. Dadurch werden höher priorisierte Pods in der Regel auch früher persistiert. Ausnahmen gibt es nur, falls ansonsten das Cluster in seiner Effizienz eingeschränkt werden würde. Um zu verhindern, dass ein Pod mit hoher Priorität und hohen Anforderungen die gesamte Schedulingqueue aufhält, gibt es ein Scheduler *back-off*. Das bedeutet, wenn der Scheduler aktuell keinen Node findet, der die Ressourcenanfrage des Requests bedienen kann, versucht der Scheduler es erst nach länger werdenden Abständen erneut. Dadurch kann es auch passieren, dass Pods mit geringerer Priorität vor dem höher priorisierten Pod persistiert werden [41]. Dies hilft, die Systemressourcen bestmöglich zu nutzen und die Wartezeiten im Cluster zu verbessern.

Mit den `PriorityClasses` und der integrierten Queue-Sortierung des Schedulers kann somit eine Priorisierung der Pods umgesetzt werden.

4.2.3 Ressourcenreservierung

Durch die `PriorityClasses` ergibt sich auch eine Lösung zur Reservierung von Ressourcen. Die dafür genutzte Mechanik nennt sich *Preemption*. `PriorityClasses` bieten die Möglichkeit, eine `preemptionPolicy` zu definieren. Wenn diese auf `Never` eingestellt ist, werden höher priorisierte Pods lediglich vor niedriger priorisierten Pods in der Schedulingqueue sortiert. Bei einem Wert von `PreemptLowerPriority` darf der Scheduler jedoch zusätzlich bereits laufende, niedriger priorisierte Pods beenden. Findet der Scheduler für einen Pod mit dieser Einstellung keinen Node mit ausreichend freien Ressourcen, überprüft er zusätzlich, ob auf einem Node durch das Beenden niedriger priorisierter Pods ausreichend Platz geschaffen werden könnte. Ist dies der Fall, werden die entsprechenden Pods beendet und der wartende Pod kann auf dem Node zugeteilt werden.

Diese Preemption kann indirekt zur Reservierung von Ressourcen genutzt werden. Dabei werden sogenannte Platzhalterworkloads erstellt, die die gewünschten Ressourcen besetzen und damit reservieren. Die Workloads, die die reservierten Ressourcen anfragen dürfen, bekommen eine höhere Priorität als die Platzhalter und dürfen diese zusätzlich durch Preemption beenden. Werden die Ressourcen durch eine solche Workload angefragt, werden entsprechend viele Platzhalterworkloads beendet und die Workload kann die freigewordenen Ressourcen nutzen. Workloads, die die so reservierten Ressourcen nicht nutzen dürfen, bekommen eine niedrigere Priorität oder eine, die keine Preemption erlaubt, wodurch sie die Platzhalterworkloads nicht beenden dürfen. Es ist wichtig zu

beachten, dass die `PriorityClasses` als Konzept nicht an Namespaces gebunden sind. Das bedeutet, dass Pods, die für einen Namespace angefragt sind, durch Preemption auch Pods eines anderen Namespaces beenden dürfen.

Dieses Konzept wird meist beim proaktiven Skalieren dynamischer Ressourcen eingesetzt und soll die Zeit von der Anforderung neuen Ressourcen bis zur Verfügbarkeit möglichst gering halten [11, p. 3]. Doch das Konzept ist wie beschrieben auch zur Reservierung von Ressourcen für Cluster mit unveränderlichen Ressourcen übertragbar.

4.3 Anwendungsbezogene Mechaniken

Für das Konzept sollen auch die angebotenen Mechaniken der Multi-Tenancy Anwendungen selbst untersucht werden. Da die Anwendungen speziell auf die Nutzung durch mehrere Nutzende ausgelegt sind, implementieren diese auch eigene Mechaniken, um die daraus entstehenden speziellen Anforderungen an die Ressourcennutzung zu steuern.

Kiosk
Da die ICC-Nutzenden mit Kiosk eigenständig mehrere Namespaces erstellen können, Namespaces jedoch kein Konzept zur gemeinsamen Quotierung besitzen, etabliert Kisok sogenannte `AccountQuotas`. Mit diesen kann für Kiosk-Nutzende eine Limitierung über die aggregierten Ressourcen aller ihrer Namespaces durchgesetzt werden. Diese Kiosk-eigene Art der Quotierung wurde bisher nicht in der ICC angewendet.

JupyterHub
Die Pods der einzelnen JupyterHub-Nutzenden werden mithilfe des *Kubespawners* erstellt. Dieser bietet dabei zusätzliche Konfigurationsmöglichkeiten zu den erstellten Pods [76]. Über den Kubespawner werden auch die in der ICC bereits angewendeten Limits der Pods festgelegt. Die Limits können dabei wie in Kubernetes selbst zu CPU, Memory oder spezielle Ressourcen wie GPU und Speicher definiert werden. Auch kann darüber die Prioritätsklasse der Pods beim Erstellen gesetzt werden [77]. Diese Einstellung wurde in der ICC bisher nicht genutzt. Zur Erkennung der Inaktivität gibt es den `jupyterhub-idle-culler`, der anhand der letzten Aktivität der Nutzenden entscheidet, ob ein Pod noch genutzt wird. Eben diese Mechanik ist in der ICC dafür verantwortlich die Pods nach acht Stunden Inaktivität zu beenden.

GitLab Runner
Limitierungen können für die vom GitLab Runner gestarteten Pods in der dafür vorgesehenen `config.toml` Konfigurationsdatei angegeben werden. Dabei können auch Angaben zu den Prioritätsklassen der Pods gemacht werden [12]. In der ICC werden so bereits Limitierungen, aber keine Prioritätsklassen gesetzt. Zusätzlich kann als eigene Quotierungsmaßnahme die Zeit festgelegt werden, die ein Job maximal laufen darf. Überschreitet der Job diese Zeit, wird der Pod beendet. Diese Mechanik kommt in der ICC bereits zum Einsatz.

4.4 Externe Erweiterungen

Manche Anforderungen der Funktionalitäten und Qualitäten sind nur mittels externer Erweiterungen realisierbar. Außerdem sollen Vorteile externer Lösungen gegenüber den bereits vorgestellten internen Mechaniken betrachtet werden.

4.4.1 Monitoring

Kubernetes selbst bietet nur limitierte Möglichkeite, Metriken zum Cluster auszulesen. Diese alleine sind nicht ausreichend für die Anforderung an das gewünschte Monitoring. Der von Kubernetes integrierte `metrics-server` kann zwar Daten zum CPU- und Memory-Verbrauch der Container auslesen allerdings ist es dem Server nicht möglich, die Auslastung der erweiterten GPU-Ressourcen zu bewerten, wodurch zur Lösung auf externe Tools zurückgegriffen werden muss [49]. Außerdem unterstützt der Server nur Echtzeitmetriken und bietet keine Unterstützung zu historische Daten, die für die Erkennung von Inaktivität vonnöten sind.

GPU-Auslastung
Zur besseren Bewertung der Ressourcenauslastung des Clusters sollen auch die erweiterten GPU-Ressourcen einbezogen werden, wofür eine aussagekräftige Metrik entwickelt werden soll. Da die GPU-Ressourcen allerdings nachträglich als CRD in das Cluster integriert werden, hat Kubernetes kein inhärentes Konzept, um die Auslastung auszulesen. Dies ist nur mit entsprechenden Erweiterungen möglich. Der Hersteller der Grafikkarten der ICC *NVIDIA* bietet dafür einen eigenen Exporter [69] an. Dieser ist Teil der

Werkzeug-Suite von NVIDIA zur Verwaltung der GPU-Ressourcen namens „Data Center GPU Manager“ [71].

Mit diesem können unterschiedliche Metriken zur GPU-Auslastung ausgelesen werden, wie beispielsweise der Speicherverbrauch, die Auslastung der Streaming-Multiprozessoren (SMs) oder die Schaltfrequenzen des Speichers und der SMs. Zusätzlich werden periphere Eigenschaften wie der Energieverbrauch in Watt, die Temperatur in Grad Celsius und die Drehgeschwindigkeit der Belüftungsventilatoren der GPUs erfasst. Diese Messwerte können anschließend exportiert und beispielsweise den bereits in der ICC integrierten Erweiterungen Prometheus und damit auch Grafana zur Verfügung gestellt werden [67].

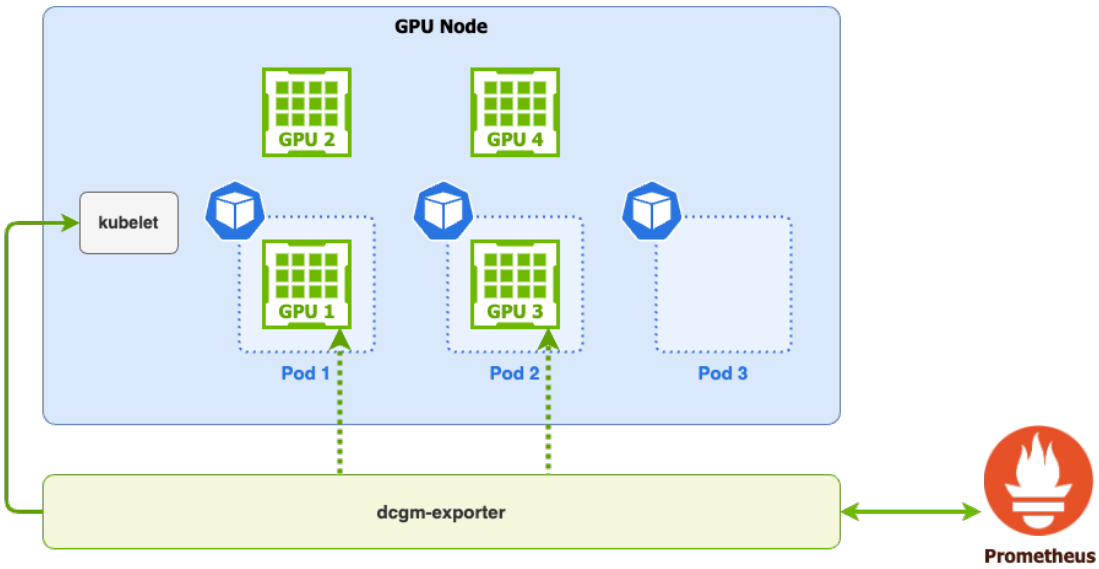

Abbildung 4.2: Per-Pod GPU-Metriken durch den DCGM-Exporter

In Kombination mit den Monitoringfähigkeiten von `kubelet` können die Informationen zur Auslastung mit dem Namen des Pods, dessen Namespace und der Device-ID verknüpft werden. Dafür verbindet sich der DCGM-Exporter wie in Abbildung 4.2 dargestellt mit dem jeweiligen `kubelet` des GPU-Nodes, um herauszufinden, welcher konkrete Pod welche GPU-Ressource nutzt. [75]

Inaktivität

Um die Inaktivität von Pods zu erkennen, sind aussagekräftige Metriken nötig. Dabei muss zuerst geklärt werden, was überhaupt unter Inaktivität zu verstehen ist. Nur weil ein Pod gerade nicht aktiv auf den CPU-Ressourcen rechnet, bedeutet dies nicht, dass er inaktiv ist. Genauso kann von einem einzigen Bild aller Pods und deren Ressourcen-

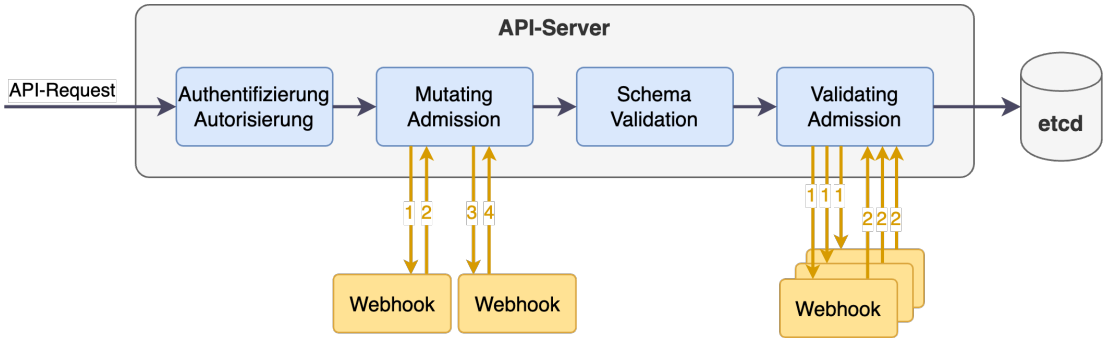

Abbildung 4.3: Dynamische Admission Controller außerhalb des API-Servers

nutzung im Cluster keine feste Aussage über die Aktivität der Pods getroffen werden. Vielmehr ist eine Kombination aus Ressourcennutzung und vergangener Zeit nötig. Das bedeutet, es müssen zusätzlich die historischen Daten zur Ressourcennutzung mit einbezogen werden. Da dies mit dem integrierten `metrics-server` von Kubernetes nicht möglich ist, muss auf externe Erweiterungen zurückgegriffen werden. Dabei bietet sich wieder Prometheus an, womit periodische Anfragen geschickt werden können, um Durchschnittswerte über einen längeren Zeitraum zu erstellen.

4.4.2 Policy Engine

Die neuen Funktionalitäten erfordern es, neue Regelungen im Cluster durchzusetzen. Prioritäten und Limitierungen zu Pods und einzelnen Namespaces sind über Kubernetes möglich, jedoch sollen diese Regelungen auch durch Ausnahmeregelungen erweitert werden können. Statt diese Erweiterungen selbst mittels eigens dafür implementierten Admission Controller umzusetzen, sollen in diesem Abschnitt dafür entwickelte *Policy Engines* betrachtet werden.

Mit Erweiterungsregeln sollen Ausnahmen zu im Voraus festgelegten Limitierungen und Prioritäten definiert werden können. Das Konzept hinter dem LimitRanger sieht keine Definition von Ausnahmen zu Pods vor. Es kann lediglich der gesamte Namespace von einer Kontrolle ausgenommen werden, indem die zugehörige `LimitRange` im Namespace gelöscht wird. Regelungen zu den verwendbaren Prioritätsklassen in Namespaces können in Kubernetes über `ResourceQuotas` realisiert werden [44]. Ausnahmen können jedoch anschließend nur auf einen Namespace bezogen definiert werden und nicht etwa für einzelne Pods oder Nutzende.

Eine Möglichkeit, diese zusätzlichen Reglungen in das Cluster zu integrieren, ist es, eigene *dynamische Admission Controller* zu schreiben. Dafür müssen diese Admission Controller als Endpunkte registriert werden, woraufhin sie zusätzlich zu den im API-Server integrierten Admission Controllern aufgerufen werden (siehe Abbildung 4.3). Ein so integrierter Controller könnte die Requests durch eigene implementierte Regeln und damit auch feingranulareren Ausnahmekonfigurationen validieren. Eine eigene Implementierung wäre jedoch mit zusätzlichem Aufwand verbunden. Die Controller müssten regelmäßig gewartet werden und Anpassungen direkt an der gewählten Codebasis erfolgen.

Statt selbst einen oder mehrere dynamische Admission Controller zu schreiben, gibt es eigens dafür entwickelte Policy Engines von verschiedenen Anbietern. Diese werden alle als dynamische Admission Controller in das Cluster integriert [42]. Dabei bieten die Policy Engines mehrere Vorteile gegenüber einer eigenen Implementierung. Zum einen abstrahieren die Policy Engines einen Großteil der Komplexität zur Implementierung und bieten eine vereinfachte Syntax zum Definieren der eigenen Regeln. Des Weiteren bieten die Policy Engines einen zentralen Punkt, alle definierten Regeln zu verwalten, was die Übersichtlichkeit verbessert. Zudem sind die Policy Engines auch mit Hinblick auf Skalierung in größeren Clustern entwickelt und sind als dynamischer Admission Controller optimiert, darauf performant und zuverlässig zu arbeiten.

Vergleich
Für diese Arbeit wurden die folgenden drei Policy Engines verglichen: OPA Gatekeeper, Kyverno und jsPolicy.

Tabelle 4.1 zeigt die Bewertungen der ausgewählten Konzepte, wobei ✔ für „ja“, **(✔)** für „eingeschränkt“ und ✘ für „nein“ steht. Dabei wurden zum einen die Spachen zur Definition der Regeln gegenübergestellt, da diese zur Bewertung der Anwendungsfreundlichkeit der Engine herangezogen werden. Da alle Policy Engines Mechaniken zur Definition von Ausnahmen bereitstellen, wurde verglichen, welche Arten von Regelungen generell möglich sind. Dabei wurde unterschieden zwischen Regeln zur Validierung, Mutation oder Generierung von Ressourcen. Bieten die Policy Engines eigene Bibliotheken an vorgefertigten Regelungen, können diese direkt angewendet werden oder als Grundstein für neue Regelungen herangezogen werden. Das erleichtert die Erstellung neuer Regelungen. Deshalb wurde verglichen, ob die Policy Engines eigene Beispielbibliotheken für Regeln bereitstellen. Letztlich wird die Größe der Community hinter den Open Source Projekten betrachtet. Die Community hinter den Projekten bestimmt über die Entwicklung und Verbesserung und damit der Qualität der Lösung.

Konzept	 OPA Gatekeeper	 Kyverno	 jsPolicy
Sprache	Rego	YAML	JavaScript
Validieren	✔	✔	✔
Mutieren	(✔)	✔	✔
Generieren	✘	✔	✔
Bibliothek	✔	✔	(✔)
Community	203	198	22

Tabelle 4.1: Gegenüberstellung der ausgewählten Policy Engines

Mit YAML-Dateien zur Definition von Regelungen bietet Kyverno einen großen Vorteil, da auch das Kubernetes Cluster selbst durch das Anwenden von YAML-Manifesten konfiguriert werden kann. Dadurch hat das ICC-Team eine Routine mit diesem Dateiformat und es fügt sich von den genutzten Formaten am besten in die Kubernetes Umgebung ein. Rego hingegen ist eine von OPA selbst entworfene funktionale Sprache, die einen hohen Anteil an Einarbeitung von den Nutzenden erfordert.

Alle Engines können validierende Regelungen erstellen. Auch mutierende Regelungen werden von allen Engines unterstützt, wobei diese bei OPA Gatekeeper stark eingeschränkt sind. Daneben sind generierende Regelungen in der OPA Gatekeeper Engine gar nicht vorgesehen. Bibliotheken, die häufig verwendeten Regelungen listen, gibt es sowohl für OPA Gatekeeper als auch für Kyverno. jsPolicy hingegen betreibt keine eigene Bibliothek, sondern verweist auf externe Quellen wie `npmjs` [73, 52, 62].

Die Community hinter allen drei Open Source Projekten ist aktiv mit regelmäßigen Commits pro Monat (jsPolicy) oder sogar pro Woche (OPA Gatekeeper, Kyverno). Die meisten Contributer mit 203 hat OPA Gatekeeper, wobei Kyverno mit 198 fast gleichauf ist. jsPolicy hat lediglich 22 gelistete Contributer [74, 54, 61].

Aufgrund der Kubernetes-ähnlichen Syntax, der zusätzlichen Möglichkeit zu mutierenden und generativen Regelungen, einer umfassenden Bibliothek und der aktiven Community bietet Kyverno in diesem Vergleich die meisten Vorteile.

4.4.3 Hierarchische Namespaces

Um allen Nutzenden einen quotierten Zugriff auf das Cluster zu ermöglichen, müssen nicht nur Pods und Namespaces, sondern auch zusammengehörige Namespaces in ihrem Ressourcenverbrauch limitiert werden. Dies ist vor allem für die Kiosk-Nutzenden nötig, da diese mehrere Namespaces erstellen können. Da Kubernetes dafür keine Mechanik bietet, stellt Kiosk eigene `AccountQuotas` bereit. Diese Funktionalität ist damit jedoch nicht außerhalb von Kiosk nutzbar. Im Folgenden soll noch eine weitere clusterweit anwendbare Möglichkeit zur Quotierung von mehreren Namespaces vorgestellt werden: der *Hierarchische Namespace Controller* (HNC) [65]. Der HNC ermöglicht es, Namespaces in Hierarchien anzuordnen und dabei auch Teilhierarchien gesammelt zu quotieren.

Der HNC wird von einer offiziellen „Working Group“ der Kubernetes-Community entwickelt und will die Multi-Tenancy Möglichkeiten von Kubernetes verbessern. Der HNC nimmt sich dem Problem an, dass Namespaces in Kubernetes als oberste Ebene der Organisation keine Konzepte zur gemeinsamen Verwaltung bieten. Speziell haben Namespaces kein Konzept darüber, wer für sie verantwortlich ist. Ist eine Person für zwei unterschiedliche Namespaces verantwortlich, weiß Kubernetes nichts davon, dass die Namespaces zu derselben Person gehören. Genauso wie ein Namespace dazu verwendet wird, um darzustellen, welche Objekte gemeinsam verwaltet werden, etabliert der HNC ein Konzept zur gemeinsamen Verwaltung mehrerer Namespaces.

Dafür organisiert der HNC die Namespaces in einer Hierarchie. Namespaces können Eltern- oder Kindknoten anderer Namespaces sein. Um einen Namespace als Kindnamespace zu erstellen, können „volle“ Namespaces oder die vom HNC integrierte CRD `SubnamespaceAnchor` verwendet werden. Wird ein `SubnamespaceAnchor` in einem Namespace erstellt, generiert der HNC einen Namespace unter diesem Namespace, welcher als „Subnamespace“ bezeichnet wird (Abbildung 4.4).

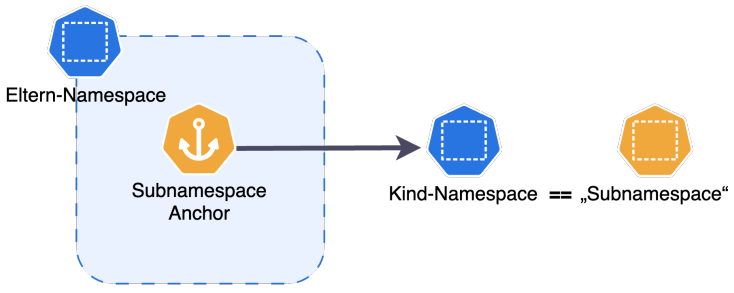

Abbildung 4.4: Begriff des „Subnamespaces“

Im Gegensatz zu einem „vollen“ Namespace ist dieser Subnamespace fest mit dem Eltern-Namespace verbunden und kann in der Hierarchie nicht nachträglich umgehangen werden. Zusätzlich ist der Lebenszyklus des Subnamespaces an den des Eltern-Namespaces gebunden. Wird der Eltern-Namespace gelöscht, wird auch der Subnamespace gelöscht.

Durch den HNC können zusäztlich bestimmte Objekte von einem Eltern-Namespace an alle Kind-Namespaces vererbt werden. Dadurch können Regelungen für eine Gruppe an Namespaces gleichmäßig angewendet werden [63]. Standardmäßig vererbt der HNC die RBAC-Einstellungen des Eltern-Namespaces durch das Propagieren der entsprechenden `Roles` und `RoleBindings`. Dadurch wird sichergestellt, dass eine Person alle Rechte, die sie in einem übergeordneten Namespace hat, auch in den angehängten Namespaces hat. Der HNC kann jedoch konfiguriert werden, alle möglichen Objekte und auch CRDs zu vererben. So können weitere Objekte zur Konfiguration zusammengehöriger Namespaces vererbt werden wie `NetworkPolicies`, `LimitRanges`, `Secrets` oder `ConfigMaps` [64].

Der HNC als eigene Multi-Tenancy Erweiterung, bietet wie Kiosk durch die Abstraktion der Namespaces in `SubnamespaceAnchors` die Möglichkeit, Nutzenden eine individuelle begrenzte Sicht auf die Namespaces des Clusters zu geben. Dabei bietet der HNC auch ein eigenes Konzept zur Quotierung von Teilhierarchien. Ein `HierarchicalResourceQuota` (HRQ) wird im gewünschten Eltern-Namespace erstellt, wodurch alle untergeordneten Namespaces gemeinsam quotiert werden.

Der HNC wird als validierender und mutierender dynamischer Admission Controller in das Cluster integriert. So kann der HNC die Logik der Hierarchien mithilfe von Labeln, Annotationen und eigener CRDs kontrollieren. Dazu kann er die vererbten Objekte erstellen und verhindern, dass diese verändert werden.

Durch den HNC kann die Übersichtlichkeit der Multi-Tenancy Namespaces durch Hierarchien verbessert werden. Auch die Möglichkeiten zur Vererbung können helfen, Regelungen auf zusammengehörige Namespaces anzuwenden. Speziell für diese Arbeit kann die Vererbung von `LimitRanges` für eine einheitliche Limitierung eingesetzt werden. Zudem kann mit den HRQ eine Quotierung über mehrere Namespaces realisiert werden.

4.5 Ergebnis

Es zeigt sich, dass es eine Vielzahl an Mechaniken und Möglichkeiten zur Erfüllung der einzelnen Qualitätsszenarien gibt. Auch wenn Kubernetes selbst schon einige Lösungen bietet, muss für Teilaspekte auf externe Erweiterungen zurückgegriffen werden. Durch Kubernetes können Limits für Pods und einzelne Namespaces sowie Priorisierungen realisiert werden. Zum Monitoring von inaktiven Pods und der GPU-Auslastung müssen externe Anwendungen integriert werden. Zusätzlich kann eine Policy Engine dabei helfen, individuelle Regelungen und Ausnahmen für das Konzept zu definieren und durchzusetzen und dabei gleichzeitig bei der Verwaltung der Regelungen unterstützen. Auch der HNC als Multi-Tenancy Lösung von Kubernetes bietet durch die hierarchische Anordnung von Namespaces, der damit verbundenen Vererbung von Objekten und Quotierung zusammengehöriger Namespaces interessante Ansätze, die gewünschten Funktionalitäten umzusetzen. Welche der vorgestellten Lösungsalternativen das Konzept für die ICC bilden und wie diese eingesetzt werden, soll im folgenden Kapitel erläutert werden.

5 Entwicklung eines Konzepts

Basierend auf der vorherigen Analyse der verfügbaren Lösungen soll im folgenden Kapitel ein Konzept entwickelt werden, welches die aufgestellten Qualitätsziele erfüllt. Dabei besteht das Konzept nicht aus einer Lösung, sondern aus einer Kombination mehrerer Teillösungen. Diese beziehen sich vor allem auf die Multi-Tenancy Anwendungen, durch welche die ICC-Nutzenden Zugriff auf die Ressourcen des Clusters haben. Besonders für Kiosk, das den Nutzenden erlaubt, direkt auf dem Cluster zu arbeiten, wird eine umfassende Umstrukturierung vorgestellt, die zum Ziel hat die Anwendung komplett zu ersetzen. Die ausgewählten Lösungsansätze werden begründet und in ihren Funktionen beschrieben. Dabei werden auch die konkreten Maßnahmen zur Umsetzung der einzelnen Lösungen in der ICC aufgezeigt.

5.1 Übersicht aller eingeführten Konzepte

Eine Übersicht aller eingeführten Konzepte ist in Abbildung 5.1 dargestellt. Für das gesamte Konzept wurden drei externe Tools eingeführt. **Kyverno** wird als Policy Engine eingeführt. Dadurch können Regelungen definiert werden, die validierend, mutierend oder generativ auf bestimmte auslösende Objekte reagieren. Die Regelungen können dabei durch individuelle Bedingungen auf die jeweiligen Situationen angepasst werden. Der **Hierarchical Namespace Controller** ist ein Projekt der offiziellen Kubernetes-Community und ermöglicht, Namespaces in Hierarchien anzuordnen. Dadurch können Objekte, die sich sonst auf einen Namespace beziehen, an weitere Namespaces vererbt werden. Zudem bietet der HNC eine gesammelte Quotierung für durch Hierarchien verwandte Namespaces an. Zuletzt wird der NVIDIA **DCGM-Exporter** integriert, welcher ermöglicht eine Vielzahl von NVIDIA bereitgestellten GPU-Metriken auszulesen.

Clusterweit werden neue Prioritätsklassen eingeführt, um es dem ICC-Team zu ermöglichen, administrative Anfragen in ihren Wichtigkeiten zu steuern. Auch das Monitoring

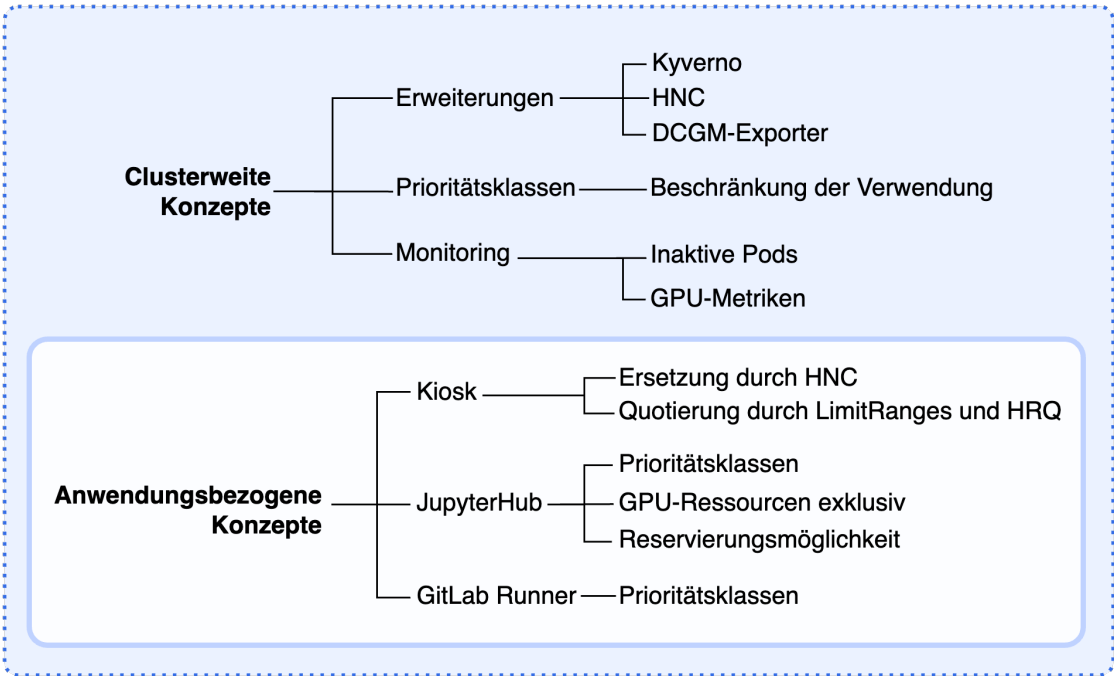

Abbildung 5.1: Eingeführte Konzepte

der Ressourcen zur Erkennung inaktiver Pods und der GPU-Auslastung bezieht sich auf das gesamte Cluster. Dabei wird in einem ersten Schritt eine Metrik zum Auswerten historischer Ressourcenauslastungen der Pods vorgestellt, wodurch die inaktivsten Pods gefunden werden sollen. Anschließend wird ein Konzept zur Integration des DCGM-Exporters erklärt.

Das umfangreichste Konzept betrifft die Ablösung von Kiosk als Multi-Tenancy Anwendung durch den HNC. Der HNC ist als Multi-Tenancy Anwendung konzipiert und wird von Kubernetes selbst aktiv weiterentwickelt. Zuerst werden die Struktur und die Abläufe der neuen HNC-Lösung für die ICC vorgestellt. Dabei werden zusätzlich neue Quotierungsmaßnahmen etabliert. Anschließend wird ein Konzept zum Parallelbetrieb der beiden Anwendungen entwickelt, um die bestehenden Ressourcen der ICC-Nutzenden in die neue Struktur zu überführen.

Im letzten Abschnitt werden die Konzepte für die beiden übrigen Multi-Tenancy Anwendungen JupyterHub und GitLab Runner vorgestellt. Mit entsprechenden Regelungen werden die GPU-Ressourcen exklusiv für JupyterHub vorgehalten. Zusätzlich werden den angefragten JupyterHub-Pods je nach ihrem Zweck andere Prioritäten zugeteilt. Dies soll

vor allem die Pods bevorzugen, die aufgrund eines geplanten Hochschulkurses angefragt werden. Für diese Praktikums-Pods wird auch ein Konzept zur Reservierung von GPU-Ressourcen mit Platzhalterworkloads entwickelt, um die nötigen Ressourcen für die Kurse zu garantieren.

Für den GitLab Runner wird eine weitere Prioritätsklasse erstellt, um sicherzustellen, dass die erstellten Jobs nicht aktiv benötigte Ressourcen aus Requests anderer Anwendungen blockieren.

5.2 Integration der ausgewählten Erweiterungen

Für die Gesamtlösung werden drei neue externe Erweiterungen in das Cluster integriert: Kyverno als Policy Engine, der Hierarchical Namespace Controller als neue Multi-Tenancy Lösung und der NVIDIA DCGM-Exporter zum Auslesen der GPU-Metriken.

5.2.1 Kyverno

Mit Kyverno soll es dem ICC-Team möglich sein, auf nutzungsfreundliche Art individuelle Regelungen für die ICC zu erstellen. Aber auch für die aufgestellten Qualitätsziele dieser Arbeit wird auf die Funktionalität von Kyverno zurückgegriffen. Da davor noch keine Policy Engine in der ICC aktiv war, ist vor allem die intuitive Nutzbarkeit ein wichtiger Faktor. Dies soll helfen, dass das Tool verstanden und auch in Zukunft weiter genutzt wird. Wie in Unterabschnitt 4.4.2 gezeigt, ist Kyverno vor allem aufgrund der intuitiven Syntax und der geboten Funktionalitäten eine gute Wahl für die Einführung einer Policy Engine. Dabei können mit Kyverno neben Regelungen zur Quotierung und Priorisierung in Zukunft auch weitere individuelle Regelungen für die ICC definiert werden. Zur Installation bietet Kyverno eine Helm-Chart[1] mit der zur Zeit aktuellsten Version v1.11.4 [54]. Dabei wird die Anwendung für den Produktiveinsatz in ihrer Verfügbarkeit verbessert, indem die enthaltenen Controller repliziert werden [51] *(Implementierung A.1)*.

[1] Helm ist ein Werkzeug, um Anwendungen vereinfacht in Kubernetes zu installieren. Ein Helm-Chart enthält gesammelt alle Ressourcendefinitionen, die notwendig sind, um die Anwendung in einem Kubernetes Cluster zu installieren [16].

5.2.2 Hierarchical Namespace Controller

Der HNC war initial als Lösung zur Quotierung mehrerer gemeinsamer Namespaces gedacht, stellt aber durch seinen Hierarchie-Ansatz mit Vererbungsmechaniken auch eine Alternative zur bisherigen Multi-Tenancy Anwendung Kiosk dar. Der HNC wird integriert, um die bisherige Multi-Tenancy Lösung mit Kiosk abzulösen. Da der HNC von der Kubernetes-Community selbst aktiv weiterentwickelt wird, stellt dies eine nachhaltigere Lösung zum nicht mehr weiterentwickelten Kiosk dar. Ein ausführliches Konzept zum HNC als Multi-Tenancy Lösung der ICC und die Ablösung von Kiosk findet sich in Abschnitt 5.4. Neben den in Unterabschnitt 4.4.3 aufgezeigten generellen Vorteilen der Übersicht und Kontrolle, soll durch den HNC die gesammelte Quotierung mehrerer Namespaces ermöglicht werden. Zum Zeitpunkt dieser Arbeit ist die aktuellste Version des HNC v1.1.0. Statt der Basis Version wird der HNC direkt mit der optionalen Erweiterung der hierarchischen Quotierung installiert.

5.2.3 NVIDIA DCGM-Exporter

Für aussagekräftige Metriken zur GPU-Auslastung bietet NVIDIA für Kubernetes den DCGM-Exporter. Da auch die Grafikkarten der ICC Modelle von NVIDIA sind, bietet sich der DCGM-Exporter an. Als NVIDIA-Entwicklung ist der DCGM konzipiert, nahtlos mit den NVIDIA-Grafikkarten zusammenzuarbeiten, wodurch es zu weniger Komplikationen bei der Integration kommen soll. Der DCGM bietet dabei eine große Auswahl an Metriken zur Auslastung. Dadurch können mehrere Metriken einzeln oder kombiniert ausgewertet werden, um ein besseres Bild zur tatsächlichen Auslastung der GPU-Ressourcen zu erhalten. Der DCGM-Exporter kann als Teil des für Kubernetes entwickelten *NVIDIA GPU-Operators* installiert werden. Der GPU-Operator ermöglicht, angeschlossene GPU-Ressourcen im Cluster verfügbar zu machen, indem er alle NVIDIA Softwarekomponenten automatisiert verwaltet. Da die Grafikkarten zum Zeitpunkt dieser Arbeit noch nicht wieder in der ICC zur Verfügung standen, konnten die folgenden Installationsschritte nicht vollständig getestet werden. Für diese Arbeit wird davon ausgegangen, dass die NVIDIA Softwarekomponenten wie Treiber und Container-Laufzeitumgebungen bereits in die ICC integriert sind.

Falls der DCGM-Exporter nicht mit dem GPU-Operator installiert wurde, kann er seperat mit einer Helm-Chart installiert werden. Dabei ist die Funktion, die Auslastung

auf konkrete Pods zurückzuführen, nicht standardmäßig aktiviert. Dafür muss in den Werten der `dcgm-exporter.values` Datei der Helm-Chart unter den Argumenten das entsprechende Flag mit `-k` gesetzt werden [68]. Um die Metriken Prometheus und damit implizit Grafana zur Verfügung zu stellen, muss zuerst die Helm-Chart von Prometheus angepasst werden. Der Servicetyp von Prometheus muss auf `NodePort` gestellt sein, um Prometheus auch außerhalb des Kubernetes Clusters zu erreichen. Zudem muss die Einstellung unter `prometheusSpec.serviceMonitorSelectorNilUsesHelmValues` auf `false` gesetzt werden. Dies ermöglicht dem DCGM-Exporter, die Information zum Finden seines eigenen Monitoring-Services an Prometheus zu übergeben. Außerdem dokumentiert NVIDIA, dass unter den `additionalConfigs` der Helm-Chart eine weitere `configMap` hinzugefügt werden muss *(Implementierung A.2)*. Diese Einstellungen passen das Verhalten zum Abfragen der GPU-Metriken von Prometheus an, indem sie den Endpunkt und die Intervalle festlegen [72]. Danach sollten die Daten für Prometheus mithilfe der neuen Variablen auslesbar sein wie beispielsweise eine generelle Information zur GPU-Auslastung mit `DCGM_FI_DEV_GPU_UTIL`. Um die Prometheus Daten in Grafana darzustellen, bietet NVIDIA ein vordefiniertes Dashboard an. Dieses kann über das Interface der Grafana-Homepage importiert werden [66]. Dieses Dashboard kann anschließend erweitert und angepasst werden.

5.3 Clusterweite Konzepte

Neben Konzepten, die sich auf bestimmte Anwendungen der ICC konzentrieren, sollen auch clusterweite Konzepte eingeführt werden. Diese werden im Folgenden dargestellt.

5.3.1 Prioritäten

Durch die Prioritätsklassen ist es in Kubernetes möglich, die Priorität von Pods untereinander anzugeben. Um eine Unterscheidung zwischen den Prioritäten zu erreichen, werden drei neue Prioritätsklassen erstellt *(Implementierung A.3)*.

- `high-icc-priority` mit einem Wert von 500
- `default-icc-priority` mit einem Wert von 0
- `low-icc-priority` mit einem Wert von -20

Dabei ist `default-icc-priority` als globaler Standardwert definiert. Das bedeutet, alle neuen Pods erhalten ohne Angabe einer eigenen Prioritätsklasse, die Prioritätsklasse `default-icc-priority` mit dem Wert 0 (Null) zugeteilt. Null wäre dabei auch der Standardwert, der von Kubernetes automatisch vergeben wird. Allerdings ist durch das Anlegen der eigenen Prioritätsklasse dieser Wert auch einsehbar festgehalten und kann zukünftig angepasst werden. Dem ICC-Team und den für die Funktionalität des Clusters wichtigen Diensten ist es erlaubt, jede Prioritätsklasse zu verwenden. Den ICC-Nutzenden der Multi-Tenancy Anwendungen soll es jedoch nicht möglich sein, höhere Prioritätsklassen als den Standardwert anzugeben. Das wird durch eine entsprechende `ClusterPolicy` sichergestellt, die die definierten Prioritätsklassen nur in den angegebenen Namespaces erlaubt *(Implementierung A.4)*. Mit `high-icc-priority` sollen zeitkritische Workloads schneller gestartet werden können, wohingegen `low-icc-priority` für zeitunkritische Workloads verwendet werden kann. Beispielsweise können Wartungsarbeiten mit `low-icc-priority` gezielt in ruhigeren Phasen des Clusters ausgeführt werden. Da selbst der Standardwert höher ist, werden Requests mit `low-icc-priority` in der Regel erst ausgeführt, wenn keine Requests aus der alltäglichen Nutzung des Clusters anstehen. Die Anpassung der Systemdienste, die vorgestellten Prioritätsklassen zu verwenden, ist nicht Teil dieser Arbeit und muss vom ICC-Team umgesetzt werden. Dafür muss in den ausgewählten Podspezifikationen die gewünschte Prioritätsklasse unter `spec.priorityClassName` angegeben werden.

5.3.2 Monitoring

Für das Monitoring ist davon auszugehen, dass Prometheus und Grafana bereits im Cluster installiert sind [86] und die Metriken pro Pod auslesbar vorliegen. Um inaktive Pods zu erkennen, muss geprüft werden, ob ein Pod über einen gewissen Zeitraum nicht aktiv Rechenressourcen nutzt. Da wie in Unterabschnitt 4.4.1 beschrieben, Kubernetes selbst keine Möglichkeit bietet, historische Daten zur Ressourcennutzung auszulesen, wird dabei auf die Funktionalitäten von Prometheus zurückgegriffen. Eine über Prometheus in der eigenen Abfragesprache *PromQL* definierte Anfrage könnte wie in Abbildung 5.2 aussehen. Zuerst wird der durchschnittliche Speicherverbrauch und die durchschnittliche CPU Zeitnutzung der letzten drei Tage aller Pods angefragt. Diese beiden Werte werden kombiniert und die niedrigsten zehn Ergebnisse ausgegeben. Dabei werden systemkritische Namespaces und die von Prometheus und Grafana selbst ausgenommen. Da Bytes für den Speicherverbrauch und Sekunden für die CPU sehr unterschiedliche Metriken dar-

```
bottomk (10,
    avg by (pod, namespace) (
        sum_over_time(container_memory_usage_bytes{namespace!~"(kube-
    system|kube-public|kube-node-lease|prometheus|grafana)"}[3d])
    )
    * on(pod, namespace) group_left()
    sum by (pod, namespace) (
        rate(container_cpu_usage_seconds_total{namespace!~"(kube-system|
    kube-public|kube-node-lease|prometheus|grafana)"}[3d])
    )
)
```

Abbildung 5.2: PromQL Anfrage die inaktivsten Pods aufzulisten

stellen, muss in der Anwendung einer solchen Abfrage darauf geachtet werden, ein gutes Verhältnis zu erreichen. Dies kann mit Multiplikationen der Unterergebnisse feinjustiert werden. Diese Prometheusanfrage kann anschließend mit Grafana genutzt werden, die Ergebnisse darzustellen.

GPU

Auch die GPU-Metriken können in Zukunft für die Erkennung von Inaktivität herangezogen werden. Entweder inkludiert in die obige Anfrage oder als eigene Metrik, um speziell ungenutzte GPU-Ressourcen wieder freizugeben. Dafür könnte beispielsweise die `DCGM_FI_DEV_GPU_UTIL` herangezogen werden.

Automatische Löschung

Mit den entsprechenden Metriken könnte in einem zweiten Schritt ein Skript erstellt werden, welches die Ergebnisse dieser Anfragen automatisch löscht. Das Skript würde dafür die Query an Prometheus senden und die identifizierten Pods mit einer Anfrage an den API-Server löschen. Das Skript kann dafür in ein eigenes Image verpackt werden und dem Container eines `CronJobs` übergeben werden. Der `CronJob` führt das Skript anschließend selbstständig zu den festgelegten Zeiten aus *(Implementierung A.5)*.

5.4 Übergang von Kiosk zu Hierarchischen Namespaces

Mit dem Konzept der Hierarchie für Namespaces des HNC können Quotierungen für mehrere Namespaces einer Person in Kiosk erreicht werden. Zusätzlich bietet die neue Ebene Übersichtlichkeit aller Kiosk-Namespaces in der ICC und die Möglichkeit für weitere ein-

heitliche Regelungen für die Nutzenden. Allerdings ist der HNC selbst als Multi-Tenancy Tool konzipiert worden. Statt also nur die organisatorischen Mechaniken des HNCs zu nutzen und damit zwei unterschiedliche Multi-Tenancy Lösungen parallel zu betreiben, kann der HNC Kiosk auch gänzlich ersetzen. Dies bietet vor allem zwei Vorteile: Der HNC ist eine zukunftsorientiertere Multi-Tenancy Lösung und durch den Wegfall einer Anwendung wird die Komplexität des Konzepts und damit die Fehleranfälligkeit und der Administrationsaufwand reduziert.

Kiosk wurde zu Beginn der ICC als Multi-Tenancy Lösung ausgewählt, da es keine offizielle Erweiterung diesbezüglich gab. Das Projekt wird allerdings seit 2022 nicht mehr weiterentwickelt [22]. Mit dem HNC ist jetzt jedoch eine Lösung verfügbar, die durch die Kubernetes-Community aktiv weiterentwickelt wird. Durch die offizielle Unterstützung und die kontinuierliche Weiterentwicklung werden die Sicherheit, Funktionalität und Stabilität der Anwendung erhöht. Aufkommende Sicherheitslücken können geschlossen und neue Funktionen und Verbesserungen integriert werden. Zusätzlich können Fehler behoben werden. Damit ist der HNC eine nachhaltige Lösung, die die Wartbarkeit der Infrastruktur verbessert und von neuen Funktionen profitieren kann.

Damit Kiosk jedoch mit dem HNC ersetzt werden kann, müssen die in der ICC genutzten Funktionalitäten von Kiosk mit dem HNC abbildbar sein. Dabei sind die Hauptfunktionalitäten, die in der ICC durch Kiosk realisiert sind:

1. Die Nutzenden haben eine eingeschränkte Sicht auf die Namespaces des Clusters.
2. Die Nutzenden können eigene Namespaces erstellen.
3. Die Nutzenden können andere Nutzende zu ihren Namespaces einladen.

Zusätzlich muss natürlich die initiale Anforderung zur Quotierung der ICC-Nutzenden erfüllt werden können (FA-01).

Da sich die ICC im laufenden Betrieb befindet und die Kiosk-Nutzenden aktiv darauf arbeiten, soll in einem ersten Schritt der Parallelbetrieb beider Anwendungen möglich sein. Dadurch wird ein fließender Übergang der Multi-Tenancy Anwendungen für die ICC ermöglicht. Bestehende Kiosk-Namespaces sollen in die neue Struktur überführt werden, um einen Verlust dieser und der darin laufenden Workloads zu verhindern. Dabei ist auch darauf zu achten, dass die Nutzenden in dieser Übergangsphase nur die Mechaniken von

Kiosk nutzen, um mit dem Cluster zu interagieren und keinen direkten Zugriff auf die Mechaniken des HNCs haben.

Im Folgenden soll zuerst ein Konzept für den HNC als die einzige Multi-Tenancy Anwendungen in der ICC vorgestellt werden. Dabei wird ein Konzept zu den Funktionalitäten von Kiosk und den Anforderungen an die Quotierung erstellt. Anschließend wird ein Konzept zum Parallelbetrieb der beiden Anwendungen vorgestellt, um die zukünftige HNC-Struktur im Hintergrund zum laufenden Betrieb von Kiosk aufzubauen.

5.4.1 Struktur der Multi-Tenancy Lösung mit dem HNC

Der HNC soll die neue Multi-Tenancy Anwendung für die ICC werden. Mit ihr soll es den ICC-Nutzenden ermöglicht werden, direkt auf dem Cluster zu arbeiten. Dafür soll eine Hierarchie für die erstellen Namespaces entwickelt werden. Dadurch sollen die von ICC-Nutzenden erstellten Namespaces von den Namespaces anderer Applikationen klar getrennt werden. Zudem sollen dadurch Regelungen effektiv vererbt werden können.

Wie in Abbildung 5.3 gezeigt, bildet ein Eltern-Namespace namens `icc-users` die Wurzel der Hierarchie. Anschließend wird für jede Person, die sich über das ICC-Tool anmeldet, ein Namespace unter `icc-users` mit dem Namen des Accounts („Account-Namespace“) angelegt. Dieser wird statt als Subnamespace als „voller“ Namespace erstellt und soll dem ICC-Team ermöglichen, diese Namespaces bei Bedarf umzuziehen. Zusätzlich erleichtert dies die Integration in das ICC-Tool, da bei einer Anmeldung lediglich der Namespace als Standardressource von Kubernetes erstellt wird. Somit entfällt der Mehraufwand, die neuen CRDs des HNCs in das Tool zu integrieren.

```
icc-users
 ├── account1
 │    └── [s] account1-default
 │    └── [s] my-project
 └── account2
      └── [s] account2-default
      └── [s] another-project

[s] zeigt einen Subnamespace an
```

Abbildung 5.3: Hierarchie der ICC-Nutzenden

Der Account-Namespace ist der Einstiegspunkt aller Nutzenden und die Ebene auf die sie Zugriff haben. Genauso wie in der jetzigen Kiosk-Nutzung wird automatisch ein erster Namespace – in diesem Fall ein Subnamespace – für die Nutzenden angelegt. Nutzenden selbst haben nur Rechte, die durch `SubnamespaceAnchor` markierten Subnamespaces, innerhalb ihres Account-Namespaces sehen und manipulieren. Dadurch wird, wie zuvor bei Kiosk, die eingeschränkte Sicht durch die Abstraktion der Namespaces erfüllt. Der Account-Namespace wird dabei so konfiguriert, dass beim Löschen alle seine Subnamespaces kaskadierend gelöscht werden. Dadurch wird das Löschen der gesammelten Ressourcen eines Accounts vereinfacht.

Der `icc-users` Namespace wird manuell erstellt. Die restliche in Abbildung 5.3 gezeigte Hierarchie kann automatisiert mit den generierenden Regeln von Kyverno erstellt werden. Dadurch bleibt die Lösung durch die entsprechenden `ClusterPolicy` Objekte im Cluster für das ICC-Team einsehbar und anpassbar. Für die Generierung der jeweiligen Objekte müssen dem `ServiceAccount` „Background-Controller“ von Kyverno entsprechende Rechte zugeteilt werden. Dabei kann die Aggregationsfunktion von `ClusterRoles` in Kubernetes genutzt werden. Dadurch können mehrere `ClusterRoles` zu einer einzigen zusammengefasst werden. Mit dieser Mechanik wird die neue `ClusterRole` mit einem entsprechenden Label `app.kubernetes.io/component: background-controller` erstellt, wodurch die neuen Rechte automatisch der `ClusterRole` des Background-Controllers hinzugefügt werden [53] *(Implementierung A.6)*.

Daneben werden die Objekte angepasst, welche das ICC-Tool für jede neu angemeldete Person erstellt. Statt einen `Account` und einen `Space` wird ein Namespace mit dem Label `created-for: icc-user` erstellt. Über dieses Label ist es möglich, die entsprechend erstellten Namespaces in einer Kyverno Regel auszuwählen. In den Regeln einer `ClusterPolicy` wird definiert, dass bei der Erstellung eines solchen Namespaces eine Hierarchie erstellt wird. Dafür werden zwei Dinge generiert: eine `HierarchyConfiguration` und ein `SubnamespaceAnchor` *(Implementierung A.8)*.

Die `HierarchyConfiguration` wird vom HNC genutzt, Informationen über die Hierarchie zu speichern. Die generierte `HierarchyConfiguration` definiert dabei den `icc-users` Namespaces als Eltern-Namespace für den Account-Namespace.

Mit der Erstellung des `SubnamespaceAnchor` wird der erste Subnamespace für die neu angemeldete Person erstellt. Dieser wird nach dem Accountnamen und der Ergänzung „default“ benannt. Dadurch wird der Einstieg in die Arbeit mit dem Cluster erleichtert, da

die Nutzenden von Beginn an einen Subnamespace haben, den sie sich anzeigen lassen können und in dem sie arbeiten können. Die Nutzenden können auch direkt in ihrem Account-Namespace arbeiten. Mit der Erstellung von Subnamespaces sollen sie jedoch für sich die Möglichkeit haben, voneinander isolierte Kontexte im Cluster zu schaffen – ganz wie es im vollen Cluster mit der direkten Erstellung von Namespaces geschieht. Dabei können die Nutzenden direkt über das CLI des HNCs neue Subnamespaces als Kinder ihres Account-Namespaces erstellen.

Um dies tun zu können brauchen die Nutzenden jedoch noch entsprechende Rechte. Diese werden in einer zweiten `ClusterPolicy` ebenfalls beim Erstellen des Namespaces erstellt. Dabei wird eine `Role` mit `RoleBinding` im Namespace erstellt *(Implementierung A.7)*. `Roles` sind eine Namespace-Ressource in Kubernetes und erlauben die definierten Rechte nur innerhalb des Namespaces in dem sie erstellt wurden. `Roles` und `RoleBindings` werden standardmäßig vom HNC vererbt. Da die Objekte im Account-Namespace erstellt werden, bedeutet das, alles, was die Nutzenden in ihrem Account-Namespace dürfen, dürfen sie automatisch in allen darunterliegenden Kind-Namespaces auch. Da sich `Roles` nur auf Namespaces beziehen und die entsprechenden Rechte nur dort gelten, können die ICC-Nutzenden die definierten Ressourcen nur in ihren Teilhierarchien verwalten.

Die Rechte ermöglichen den ICC-Nutzenden des HNCs als minimale Vorlage die Erstellung und Verwaltung von `SubnamespaceAnchors`, `Pods` und `RoleBindings`. An dieser Stelle können weitere Ressourcen ergänzt werden, die von den ICC-Nutzenden eigenständig in ihren Teilhierarchien verwaltet werden sollen.

Mit der Möglichkeit eigene `RoleBindings` zu erstellen, können ICC-Nutzenden anderen ICC-Nutzenden Zugriff auf ihre Teilhierarchie geben. Dafür müssen sie lediglich ein `RoleBinding` erstellen, dass als `User` auf den Accountnamen anderer ICC-Nutzenden verweist. Als Rolle wird die zu Beginn für den Account gerierte Rolle festgelegt *(Implementierung A.9)*. Dabei können sie das `RoleBinding` in ihrem Account-Namespace erstellen, um einen vollen Zugriff auf alle Subnamespaces zu ermöglichen. Es können aber auch beliebige andere Teilhierarchien unterhalb dieser Ebene geteilt werden. Da die ICC-Nutzenden nur das `RoleBinding` verwalten können, haben sie keinen Einfluss auf die tatsächlichen Rechte, die in der `Role` definiert sind.

Wie hier gezeigt, sind mit den Konzepten des HNC und der geplanten Struktur alle nötigen Funktionalitäten von Kiosk abbildbar. Durch die Subnamespaces ist eine eingeschränkte Sicht auf Namespaces sowie das eigenständige Erstellen von neuen Name-

spaces als Subnamespaces möglich. Auch das gemeinsame Arbeiten unterschiedlicher ICC-Nutzenden ist durch das Erstellen entsprechender RoleBindings möglich.

Quotierungsmaßnahmen

Um eine angemessene Quotierung zu ermöglichen, wird im `icc-users` Namespace eine `LimitRange` erstellt *(Implementierung A.11)*. Durch die Vererbung des HNC wird diese auch in allen untergeordneten Namespaces und Subnamespaces erstellt *(Implementierung A.10)*. Die LimitRange selbst stellt dabei keine Grenzen für die Requests oder Limits der Pods. Stattdessen definiert sie Standardwerte, falls die Informationen nicht gesetzt sind. Dies ist nötig, da die Quotierungen mit diesen Informationen arbeiten. Änderungen der originalen LimitRange werden dabei nach unten propagiert. Dabei werden automatisch gesetzte Request- und Limitwerte vom LimitRanger nicht auf Konsistenz überprüft. Dadurch kann es passieren, dass fehlende Limitwerte gesetzt werden, allerdings unter den Reqeuestwerten liegen. Solche Pods werden mit einer Fehlermeldung abgelehnt. Darüber sollen die ICC-Nutzenden auf der Dokumentationsseite der ICC hingewiesen werden:

> ⓘ Wenn keine Werte für Request oder Limit des Pods bzw. der Workload angegeben werden, werden Standardwerte gesetzt:
> `cpu: reqeust: 0.25, limit: 0.5`
> `memory: request: 256Mi, limit: 512Mi`
> Wenn Sie nur einen Request-Wert (`> cpu: 0.5, memory: 512Mi`) gesetzt haben und ein niedrigeres Limit automatisch gesetzt wird, kann die Workload nicht erstellt werden. Dies liegt daran, dass das Limit größer gleich dem Request sein muss. Auf der sicheren Seite sind Sie, wenn sie für alle Ressourcen sowohl den Request als auch das Limit angeben.

Zusätzlich wird eine weitere generierende `ClusterPolicy` erstellt. Diese generiert für jeden Account-Namespace ein `HierarchicalResourceQuota` in eben diesem *(Implementierung A.12)*. Dadurch wird jeder Account-Namespace und alle darunter erstellten Subnamespaces gesammelt quotiert. Dadurch sind alle ICC-Nutzenden des HNCs in ihren Ressourcen quotiert. Statt wie in den vorherigen Regeln die generierten Objekte direkt in den Regeln zu beschreiben (*Data Source*), wird das `HierarchicalResourceQuota` mit einer *Clone Source* erstellt. Dieser Begriff bezeichnet in Kyverno, dass die Informationen zum generierten Objekt von einem existierenden Objekt geklont werden [54]. Dies hat den Vorteil, dass die Quotierungen auch bei einem Löschen der Policy erhalten bleiben,

wodurch die Quotierungen für die ICC-Nutzenden garantiert werden sollen. Das entsprechende Objekt wird in einem dafür angelegten Namespace namens `clone-source` innerhalb des `icc-users` Namespaces erstellt *(Implementierung A.13)*. Änderungen an diesem Objekt werden von `Kyverno` direkt an alle geklonten Objekte übertragen.

Die Werte der Quotierung sollen dabei für die Standardanforderungen der ICC-Nutzenden ausreichen. Als Beispiel soll dafür eine von den Kiosk-Nutzenden der ICC häufig erstellte Kombination aus WordPress-Applikation mit verknüpfter Datenbank herangezogen werden. Die einzelnen Werte sind dabei aus den am meisten verwendeten Helm-Charts zu WordPress Installationen abgeleitet [3, 14, 4, 83]. Dabei muss berücksichtigt werden, dass die Werte der Charts nicht die Umstände einer Multi-Tenancy Architektur berücksichtigen, sondern als vollwertige Applikationen innerhalb eines Clusters aufgestellt sind. Für die Standardanforderungen der ICC-Nutzenden, die meist ein Projekt innerhalb eines Semesters betreiben, sollten Bruchteile dieser Werte ausreichen. Diese skalierten Werte wurden dann um einen kleinen Puffer erhöht, um eine eventuell zweite Anwendung zu ermöglichen. Zusätzlich ist die Anzahl der Namespace-Ressourcen selbst quotiert. Da jedes Objekt einen gewissen Verwaltungsaufwand für beispielsweise Scheduler und Controller Komponenten in Kubernetes bedeuten, kann dadurch die Belastung auf die Infrastruktur reduziert werden.

Die Request-Werte sind niedriger gewählt als die der Limits. Dadurch können ICC-Nutzende Pods mit höheren Limits als Requests anlegen. Dies soll den ICC-Nutzenden ermöglichen, kurze Hochzeiten ihrer Anwendungen mit ungenutzten Ressourcen des Clusters aufzufangen. Dabei können die Pods in ihren Ressourcenanforderungen über die im Request angegebenen Werte, jedoch nie über das Limit gehen. Solche Pods befinden sich in einem *überprovisionierten* Zustand. Überschreitet ein Pod das gesetzte Limit, wird er gedrosselt (CPU) oder abgeschaltet (Memory).

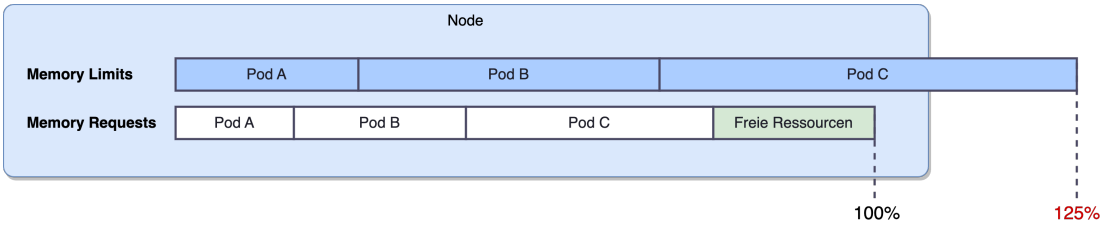

Abbildung 5.4: Die gesammelten Ressourcenlimits können die tatsächlichen Kapazitäten des Nodes überschreiten

Da der Scheduler die passenden Nodes jedoch nur nach den Request-Werten auswählt, kann es passieren, dass die Ressourcen des Nodes trotz der Einhaltung der Limits nicht für alle Pods ausreichen. (Siehe Abbildung 5.4) Werden die Kapazitäten des Nodes erreicht, werden Pods mit einem Verbrauch über ihren Requests zuerst beendet [39]. ICC-Nutzende können sich dadurch bewusst entscheiden, mehr als durch die Request-Grenzen vorgegebenen Ressourcen anzufragen. Reizt ein Pod jedoch kontinuierlich sein Limit aus, wird er bei mangelnden Ressourcen mit hoher Wahrscheinlichkeit abgeschaltet.

Ein entsprechender Hinweis soll auf der Dokumentationsseite darüber informieren:

> ⓘ Sie dürfen höhere Limits als Requests für Ihre Workloads definieren. Dadurch können Ihre Workloads zu Hochzeiten mehr als den gesetzten Request anfordern, jedoch nie mehr als das Limit. Dadurch laufen Ihre Pods allerdings Gefahr, bei einem „Node Pressure“ abgeschaltet zu werden (mehr Informationen dazu auf der offiziellen Kubernetes Doku: `https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/`) Auf der sicheren Seite sind Sie das Limit gleich dem Request zu setzen.

Letztlich haben die ICC-Nutzenden immer die Möglichkeit, mehr Ressourcen beim ICC-Team direkt anzufragen. Sollte ein Projekt mehr Ressourcen benötigen, als durch die Quotierung erlaubt, kann das ICC-Team den Namespace von der Standardquotierung durch Kyverno ausnehmen und anschließend eine individuelle Quotierung definieren. Auch darauf soll auf der Dokumentationsseite der ICC verwiesen werden:

> ⓘ Ihre Namespaces unterliegen einer festgelegten Quotierung zu angefragten Ressourcen und der Anzahl an bestimmten Objekten. Sollte Sie beim Versuch, ein neues Objekt zu erstellen, eine Fehlermeldung wie die folgende angezeigt bekommen, haben Ihre Namespaces die Grenzen der Quotierung erreicht.
> `Error from server (Forbidden): error when creating “your-pod.yaml”: pods “your-pod” is forbidden: exceeded quota: icc-user-hrq, ...`
> Wenn Sie für eines Ihrer Projekte mehr Ressourcen als die vorgegebene Quotierung erlaubt benötigen, kontaktieren Sie bitte das ICC-Team.

Da die `preconditions` einer existierenden Policy nicht bearbeitet werden können, muss die Policy für eine Änderung gelöscht und neu erstellt werden. Damit auch danach sichergestellt ist, dass alle nicht ausgenommenen Nutzenden das definierte HRQ bekommen, arbeitet die `ClusterPolicy` im `generateExisting` Modus. In diesem Modus untersucht Kyverno nach dem initialen Erstellen der Policy zuerst das Cluster auf existierende Auslöser und wendet die Regeln entsprechend an. Nach diesem initialen Durchlauf funktionieren die Regeln wie normale Regeln und werden bei einem `CREATE` oder `UPDATE` Befehl der Auslöser angewendet. In diesem Fall wird das HRQ auch rückwirkend für alle Account-Namespaces als Auslöser erstellt *(Implementierung A.12)*.

5.4.2 Parallelbetrieb beider Multi-Tenancy Anwendungen

In einer ersten Übergangsphase sollen beide Multi-Tenancy Anwendungen parallel laufen. Dadurch können die aktiven Kiosk-Nutzenden das Cluster vorerst wie gewohnt nutzen. Währenddessen werden im Hintergrund die bereits existierenden Namespaces und Workloads in die neue Struktur des HNC überführt. Zusätzlich sollen die Konzepte zur Quotierung eingeführt werden, um auch schon die Kiosk-Nutzenden in ihrem Ressourcenverbrauch zu limitieren. Ziel ist es, dass in einem letzten Schritt Kiosk deinstalliert werden kann und die Nutzenden ihre erstellten Ressourcen in der neuen HNC-Struktur vorfinden und direkt weiterarbeiten können. Eine Herausforderung der Koexistenz beider Multi-Tenancy Anwendungen ist dabei, dass diese nichts voneinander wissen und nicht direkt zusammenarbeiten können. Die entsprechenden Konzepte müssen selbst implementiert werden und sollen hier vorgestellt werden.

Um die in Unterabschnitt 5.4.1 vorgestellte Struktur abzubilden, wird der `icc-users` Namespace manuell erstellt. Auch die `LimitRange` zur Quotierung kann direkt dort erstellt werden, um sie an alle unterliegenden Namespaces zu vererben.

In der ersten Ebene unter `icc-users`, sollen die Account-Namespaces angelegt werden. Diese bekommen zusätzlich jeweils das bereits vorgestellte `HierarchicalResourceQuota`. Die tatsächlichen `Spaces` der Nutzenden sollen unter ihren entsprechenden Account-Namespace gehängt werden. Diese `Spaces` sollen zukünftig die Subnamespaces der Nutzenden werden. Das bedeutet, die Namespaces der Nutzenden müssen für diese Lösung von beiden Anwendungen als ihre Definition der Abstraktion von Namespaces erkannt werden. Sie müssen gleichzeitig `Spaces` und Subnamespaces sein. Dabei ist vorausgesetzt, dass die Erstellung der Namespaces durch Kiosk erfolgt und die HNC-Struktur

im Nachhinein ergänzt wird. Die Namespaces werden also zu `Spaces`, indem sie als solche über Kiosk angelegt werden. Dadurch legt Kiosk den entsprechenden Namespace im Hintergrund an. Subnamespaces können mit dem HNC erstellt werden, indem ein entsprechender `SubnamespaceAnchor` im Eltern-Namespace erstellt wird. Daraufhin erstellt der HNC den davon repräsentierten Namespace. Da jedoch in diesem Szenario Kiosk den Namespace bereits erstellt hat, muss betrachtet werden, wie der HNC im Nachhinein überzeugt werden kann, dass es sich um einen Subnamespace handelt.

Der HNC erkennt einen Namespace als Subnamespace an, wenn die folgenden zwei Bedingungen erfüllt sind:

1. Es muss ein `SubnamespaceAnchor` im Eltern-Namespace existieren mit demselben Namen wie der Namespace.

2. Der Namespace muss eine Annotation `hnc.x-k8s.io/subnamespace-of` auf den Namen des Eltern-Namespaces gesetzt haben [64].

Zuerst einen `SubnamespaceAnchor` für einen bereits existierenden Namespace anzulegen, wird durch den HNC mit einer entsprechenden Fehlermeldung unterbunden. Die einzige Möglichkeit ist es, den Namespace zuerst zu annotieren und anschließend den `SubnamespaceAnchor` zu erstellen. So ist es möglich, die von Kiosk erstellten Namespaces im Nachhinein als Subnamespaces in die Hierarchie einzuordnen.

Für das Konzept bedeutet das, es muss für alle angemeldeten ICC-Nutzenden ein Account-Namespace unter `icc-users` erstellt werden und anschließend alle durch `Spaces` angelegten Namespaces mit der beschriebenen Methode unter die jeweiligen Account-Namespaces gehängt werden. Dabei ist jedoch zu beachten, dass dies nicht nur für neue, sondern auch für bereits existierende angemeldete Nutzenden und deren `Spaces` geschehen muss.

Zur Umsetzung werden Kyverno `ClusterPolicies` im `generateExisting` Modus eingesetzt. Mit dieser Funktionalität sollen Regeln erstellt werden, die es ermöglichen, die bereits existierenden Kiosk Ressourcen und anschließend auch die neu erstellten Ressourcen in die HNC-Struktur zu überführen. Dafür kommt sowohl eine mutierende als auch eine generierende `ClusterPolicies` zum Einsatz. Beide sind dabei konfiguriert, rückwirkend auf existierende Auslöser zu reagieren. Zu beachten ist, dass Kyverno zwar die Reihenfolge der einzelnen Regeln innerhalb einer `ClusterPolicy` garantiert [50], jedoch nicht die Reihenfolge der angewendeten Regeln zwischen verschiedenen `ClusterPolicies`. Aus

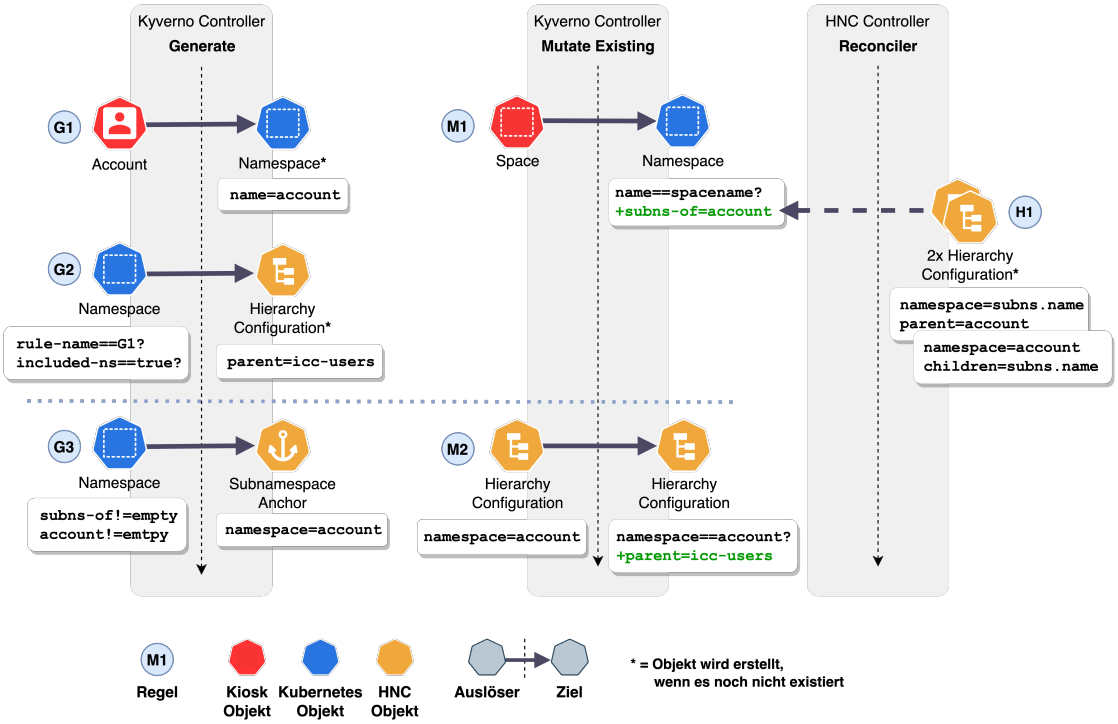

Abbildung 5.5: Sequenzdiagram Kiosk zu HNC Struktur

diesem Grund müssen erstellte Regeln unter den verschiedenen `ClusterPolicies` in der initialen Phase für existierende Auslöser parallel anwendbar sein. In Abbildung 5.5 sind die verschiedenen Prozesse mit Pseudocode zur Beschreibung der Bedingungen und Eigenschaften dargestellt. Ausschnitte der Manifeste sind in *Implementierung A.14* und *Implementierung A.15* zu finden.

Als Auslöser zur Erstellung der Account-Namespaces dienen dabei die `Accounts` (**G1**). Aus den `Accounts` lässt sich auch der Nutzungsname und damit der Name für den Account-Namespace ableiten. Ein so erstellter Namespace wird anschließend durch das Erstellen einer `HierarchyConfiguration` unter `icc-users` gehängt (**G2**). Um `Spaces` als Subnamespaces unter den Account-Namespace zu hängen, dienen die `Spaces` selbst als Auslöser. Der unterliegende Namespace bekommt dann die entsprechende Annotation, die ihn als Subnamespace des Account-Namespaces auszeichnet (**M1**). Anschließend muss noch der `SubnamespaceAnchor` im Account-Namespace erstellt werden. Dabei dient der durch den `Space` erstellte Namespace als Auslöser. Dieser ist durch ein Label `account` als von Kiosk erstellter Namespace identifizierbar. Zusätzlich kann hier sichergestellt werden, dass der `SubnamespaceAnchor` erst bei einem erfolgreichen Setzen der

`subnamespace-of` Annotation erstellt wird (**G3**). In diesem Szenario wäre gesichert, dass sowohl die Account-Namespaces unter `icc-users` erstellt werden, als auch die `Spaces` als Subnamespaces unter die Account-Namespaces gehängt werden. Der einzige Punkt, an dem die Regeln unter den `ClusterPolicies` in einer festen Reihenfolge arbeiten müssen, ist das Erstellen des `SubnamespaceAnchors`. Hierfür muss die Annotation bereits mutiert worden sein, damit kein Fehler durch den HNC ausgelöst wird. Dies wird durch die generierende Bedingung sichergestellt, dass die Annotation bereits gesetzt sein muss. Ist dies in einem ersten Durchgang noch nicht geschehen, wird die Regel durch das Setzten der Annotation als Modifikation des Auslösers erneut gestartet. Damit kann beim Ablauf garantiert werden, dass wenn **G3** ausgelöst wird, **M1** bereits geschehen ist. Allerdings kann Kyverno nicht alleine an der Umsetzung arbeiten, da sich in den beschriebenen Prozessen auch der Controller des HNCs einschaltet. Dieser erstellt selbstständig fehlende `HierarchyConfigurations`, wenn die Annotation `subnamespace-of` gesetzt wird. Dabei generiert er zwei `HierarchyConfigurations`. Eine davon wird im annotierten Namespace erstellt. Diese verweist auf den in der Annotation definierten Namespace als Eltern-Namespace. Zusätzlich wird auch die `HierarchyConfiguration` im Eltern-Namespace erstellt, die auf den annotierten Namespace als Kind-Namespace verweist (**H1**). Die `HierarchyConfiguration` (HC) für den annotierten Namespace selbst ist kein Problem. Jedoch stellt die Erstellung der zweiten HC im Eltern-Namespace und damit dem Account-Namespace ein Problem dar. Der HNC setzt durch, dass alle HCs mit dem Namen „hierarchy“ erstellt werden müssen. Aus diesem Grund kann es immer nur eine HC innerhalb eines Namespaces geben. Die in **G2** generierte HC kann also nicht erstellt werden, wenn es schon eine gibt. Allerdings enthält die HC aus **G2** die essenzielle Information, dass `icc-users` der Eltern-Namespace des Account-Namespaces werden soll. Die vom HNC generierte HC enthält diese Information nicht. Da nicht garantiert werden kann, dass **G2** vor **H1** ausgeführt wird, wird noch eine weitere mutierende Regel **M2** erstellt. Diese mutiert alle HC jener Namespaces, die nach einem `Account` benannt sind. Dabei ergänzt sie diese HCs um die Eigenschaft, dass der Eltern-Namespace `icc-users` ist. Dadurch ist es irrelevant, ob die HC durch **H1** oder **G2** erstellt wurde. Am Ende wird sie auf jeden Fall die Info zu ihrem Eltern-Namespace `icc-users` enthalten.

Mit diesen Regelungen kann die Hierarchie aus Abbildung 5.3 im Hintergrund zum aktuellen Kiosk Betrieb erstellt werden. Die ICC-Nutzenden bekommen von diesem Prozess nichts mit, da sie zu diesem Zeitpunkt noch keine Rechte haben, die Objekte des HNCs zu sehen.

Als einzige Einschränkung dürfen sie in der Übergangsphase ihre Spaces nicht löschen. Dies liegt daran, dass beim Löschen eines Spaces, Kiosk versucht, den dahinterliegende Namespace zu löschen. Dies wird allerdings vom HNC verhindert, da die Subnamespaces als Namespaces nicht direkt gelöscht werden sollen. Der HNC sieht vor, in diesem Fall den `SubnamespaceAnchor` zu löschen, wodurch der Subnamespace ebenfalls gelöscht werden würde. Die Löschung wäre möglich, indem die Annotation `subnamespace-of` des Subnamespaces wieder entfernt werden würde. Jedoch ist bei einer `DELETE` Anfrage zu einem Objekt keine Mutation des Objektes mehr möglich. Eine Mutation, die beim Löschen eines `Space` bereits versucht, den Namespace dahinter zu mutieren, kann nicht garantiert vor dem Löschen durch Kiosk ausgeführt werden. Aus diesem Grund wäre die einzige Möglichkeit, die Validierungskonfiguration des HNCs selbst anzupassen. Dabei kann eingestellt werden, dass dieser nicht mehr auf das Löschen von Namespaces reagiert. In Absprache mit den Administratoren wurde entschieden, dass die Einschränkung zum Löschen in der Übergangsphase hinzunehmen ist. Die Übergangsphase soll schließlich nur von kurzer Dauer sein und die angelegten Ressourcen der ICC-Nutzenden werden in letzter Instanz automatisch nach 90 Tagen Inaktivität gelöscht. Im Gegensatz dazu wären die negativen Auswirkungen eines in seiner Funktionalität eingeschränkten HNCs durch das vergessene Wiederherstellen der originalen Validierungskonfiguration dem Nutzen des Löschens nicht angemessen. Um den Kiosk-Nutzenden dennoch nicht die für sie unverständliche Fehlermeldung des HNCs bei einer Löschung anzuzeigen, wird eine eigene Validierung erstellt. Diese lehnt schon das Löschen des `Spaces` ab und zeigt eine verständlichere Fehlermeldung an *(Implementierung A.16)*.

In einem letzten Schritt kann Kiosk über helm deinstalliert werden. Dabei werden lediglich die Controller von Kiosk gelöscht. Die CRDs müssen separat manuell gelöscht werden. Doch dadurch, dass der Controller von Kiosk nicht mehr aktiv ist, werden dadurch auch keine existierenden Namespaces gelöscht. Die ICC-Nutzenden behalten dadurch ihre Namespaces und durch die `ClusterPolicies` sind diese schon in die neue Struktur des HNC eingegliedert. Nur das Objekt, das die ICC-Nutzenden als ihre Namespaces anfragen können, ändert sich von `Spaces` zu `SubnamespaceAnchors`. Die drei hier vorgestellten `ClusterPolicies` werden gelöscht, wobei das HRQ in den Account-Namespaces erhalten bleibt. Die neuen Regelungen aus Unterabschnitt 5.4.1 werden anschließend in das Cluster integriert. Dabei muss die `ClusterPolicy` zum Erstellen der `Roles` und `RoleBindings` auf `generateExisting` gestellt werden, um auch den bestehenden ICC-Nutzenden zu ermöglichen, ihre Subnamespaces zu sehen und zu verwalten.

Um die bereits von Kiosk erstellten RBAC-Objekte für alle ICC-Nutzenden zu löschen, kommt eine zusätzliche `ClusterCleanupPolicy` von Kyverno zum Einsatz. Dafür benötigt der Cleanup-Controller von Kyverno entsprechende Rechte *(Implementierung A.17)*.

Mit den hier vorgestellten Konzepten ist es möglich, die Struktur für eine Multi-Tenancy Lösung mit dem HNC parallel zum aktiven Kiosk-Betrieb aufzubauen. Die Nutzenden werden dabei so wenig wie möglich eingeschränkt. Die aufgebauten Strukturen und die unter Kiosk angelegten Ressourcen der Nutzenden bleiben auch nach einer Deinstallation von Kiosk erhalten.

5.5 Konzepte zu JupyterHub und GitLab Runner

In diesem Abschnitt sollen die Konzepte für die restlichen beiden Multi-Tenancy Anwendungen JupyterHub und GitLab Runner vorgestellt werden. Auch wenn beide Anwendungen in ihrem Ressourcenverbrauch limitiert sind, soll an dieser Stelle die Exklusivität der GPU-Ressourcen für JupyterHub als Quotierungsmaßnahme des Clusters vorgestellt werden. Außerdem werden Prioritätskonzepte für beide Anwendungen gezeigt, wobei diese für JupyterHub auch zur Reservierung genutzt werden.

5.5.1 JupyterHub

Für JupyterHub besteht, wie in der Anforderungsanalyse Unterabschnitt 3.1.4 gezeigt, kein konkreter Bedarf für eine weitere Quotierung. Da die Nutzenden nur indirekt über die JupyterHub-Oberfläche mit dem Cluster interagieren, können sie nur aus vordefinierten Podkonfigurationen mit definierten Limitierungen wählen. Jedoch können Prioritätsregelungen und Reservierungen dabei helfen, die GPU-Ressourcen besser zu verteilen.

Quotierung
Für die GPU-Ressourcen soll an dieser Stelle ein Limit für alle anderen Namespaces gesetzt werden. Da die GPU-Ressourcen primär für ML-Aufgaben über die JupyterHub-Oberfläche gedacht sind, soll es den anderen Namespaces nicht gestattet werden, GPU-Ressourcen anzufragen. Eine validierende `ClusterPolicy` überprüft, ob der Container eines Pods ein Limit für eine GPU angegeben hat. Dabei genügt die Überprüfung des Limits, da die GPU-Ressourcen nur über Limits angefragt werden können [45]. Fragt

ein Pod außerhalb der exkludierten Namespaces eine GPU an, wird der Request abgelehnt *(Implementierung A.18)*. Auf diese Weise können keine GPU-Ressourcen von den anderen Multi-Tenancy Anwendungen und deren Nutzenden angefragt werden. Falls in Sonderfällen ICC-Nutzenden der anderen Anwendungen GPU-Ressourcen zur Verfügung gestellt werden sollen, kann dies durch das Hinzufügen von Ausnahme-Namespaces in der `ClusterPolicy` geschehen. Diese Information soll auch auf der Dokumentationsseite der ICC festgehalten werden:

> ⓘ Den über Kiosk/den HNC erstellten Namespaces ist es nicht erlaubt, GPU-Ressourcen anzufordern. Wenn Sie für eines Ihrer Projekte GPU-Ressourcen benötigen, kontaktieren Sie bitte das ICC-Team.

Priorisierung

Es soll zwischen den Nutzenden unterschieden werden, die aufgrund eines geplanten Hochschulkurses Aufgaben mit JupyterHub bearbeiten und solchen, die aus eigener Entscheidung für individuelle Projekte auf das Angebot zurückgreifen. Dabei sollen die Pods der Praktikumsteilnehmenden eine höhere Priorität bekommen, um ihnen schneller und sicherer die GPU-Ressourcen zuzuteilen. Die Unterscheidung der Anfragen erfolgt dabei nach dem Vertrauensprinzip, wobei die Praktikumsteilnehmenden die entsprechend markierten Konfigurationen in der Weboberfläche auswählen. Um unterschiedliche Prioritäten zuzuteilen, muss die Konfiguration der entsprechenden Profile im Kubespawner geändert werden und eine Prioritätsklasse unter `kubespawner_override.priority_class_name` angegeben werden. Dabei muss beachtet werden, dass nicht die Priorität der Praktikums-Pods erhöht, sondern die der restlich angefragten JupyterHub-Pods verringert wird *(Implementierung A.19)*. Somit wird garantiert, dass die JupyterHub-Pods nicht die Pods aus anderen Namespaces in ihrem Scheduling beeinflussen. Da Prioritäten kein auf Namespaces beschränktes Konzept sind, würde eine höhere Priorität über allen angefragten Pods im Cluster mit niedrigerer Priorität stehen, wodurch der Scheduler die Praktikums-Pods vor allen anderen Pods des Clusters bearbeiten würde *(Implementierung A.20)*.

Reservierung

In JupyterHub gibt es auch einen konkreten Anwendungsfall zur Reservierung von Ressourcen. Für geplante Praktikumsgruppen aus Hochschulvorlesungen können GPUs reserviert werden, damit die Praktika garantiert eine bestimmte Anzahl der Ressourcen anfragen können. Dies ist möglich mit den in Unterabschnitt 4.2.3 erwähnten Platzhal-

terworkloads, die von höher priorisierten Pods beendet werden können. Auch hier ist es wichtig zu beachten, dass die Prioritäten und damit auch die Preemption-Logik keine auf Namespaces begrenzten Konzepte sind. Die Prioritäten müssen so abgestimmt sein, dass keine ungewünschten Workloads in anderen Namespaces beendet werden. Die kann mit einer vom Standardwert 0 (Null) verringerten Prioritätsklasse für beide Arten von JupyterHub-Pods erreicht werden. Dabei bekommen beide den Wert -1, wobei nur die Praktikums-Pods niedriger priorisierte Pods beenden dürfen *(Implementierung A.21)*. Die Platzhalterworkloads werden anschließend bei Bedarf in der gewünschten Anzahl mit einer Priorität von -50 im Namespace von JupyterHub gestartet *(Implementierung A.22)*. Die Pods beanspruchen dabei jeweils eine GPU-Ressource. Anschließend können diese Pods von angefragten Praktikums-Pods beendet werden, da diese eine höhere Prioritätsklasse mit der Fähigkeit zum vorzeitigen Beenden niedriger priorisierter Pods besitzen.

5.5.2 GitLab Runner

In der Anforderungsanalyse in Unterabschnitt 3.1.4 hat sich gezeigt, dass auch für den GitLab Runner keine weiteren Quotierungsmaßnahmen nötig sind. Doch auch hier können Prioritäten helfen, die Ressourcen des Clusters besser zu verteilen. Um mit den Jobs des GitLab Runners keine akut benötigten Ressourcen zu belasten, kann eine neue Prioritätsklasse eingeführt werden, die die Jobs hinter die `default-icc-priority` stellt. Die entsprechende Einstellung kann unter `priority_class_name` in der `config.toml` Konfigurationsdatei des GitLab Runners angegeben werden.

5.6 Ergebnis

Mit den Möglichkeiten von Kubernetes selbst und den neu eingeführten Erweiterungen wurde ein Konzept zur Quotierung und Priorisierung in der ICC erstellt. Das Konzept besteht dabei aus mehreren Lösungen, die clusterweit oder in den speziellen Multi-Tenancy Anwendungen der ICC eingesetzt werden. Inwiefern das Konzept die aufgestellten Qualitätsszenarien erfüllen, soll im nächsten Kapitel evaluiert werden.

6 Evaluation des Konzepts

In diesem Kapitel werden die aufgestellten Anforderungen aus Kapitel 3 mit dem Konzept aus Kapitel 5 gegenübergestellt. Da die Use-Cases selbst nicht messbar evaluiert werden können, soll das Konzept auf die Erfüllung der Qualitätsziele ausgewertet werden.

6.1 Testszenario

Die gezeigten Lösungen wurden für diese Arbeit mit Minikube [6] getestet. Minikube ist eine Open-Source Lösung, die entwickelt wurde, um ein lokales Kubernetes Cluster aufzusetzen. Dabei startet Minikube eine virtuelle Maschine auf dem Rechner, welche ein Cluster aus einem einzigen Node simuliert. Dieser Node ist dabei die Rechenmaschine des Clusters und enthält gleichzeitig auch alle Komponenten der Control Plane. In einem über Minikube gestarteten Kubernetes Cluster konnten die Mechaniken und externen Anwendungen lokal getestet werden. Eine GPU-Unterstützung ist dabei nur über NVIDIA-Grafikkarten möglich [46]. Da diese dem hier verwendeten Rechner nicht zur Verfügung standen, konnten dem Minikube Cluster keine GPU-Ressourcen zur Verfügung gestellt werden. Auch die GPU-Ressourcen der ICC, waren zum Zeitpunkt dieser Arbeit noch nicht wieder vollständig integriert, wodurch die Auswertungen der GPU bezogenen Qualitätsziele eingeschränkt ist.

6.2 Auswertung der Qualitätsszenarien

Die Qualitätsziele werden mittels einer Skala bewertet. Diese hat drei Bewertungen (Tabelle 6.1), die anzeigen sollen, ob das Qualitätsziel erfüllt wurde oder nicht. Zusätzlich gibt es eine Bewertung, welche aussagen soll, dass das Qualitätsziel nicht getestet wurde und deshalb keine Aussage zu Erfüllung getroffen werden kann.

Erfüllt	Nicht erfüllt	Nicht getestet
✅	❌	☐

Tabelle 6.1: Skala zur Bewertung der Erfüllung der Qualitätsszenarien

Zuerst sollen die Funktionalitäten der Use-Cases selbst durch die ersten Qualitätsziele bewertet werden.

ID	Szenario	Bewertung
FC-01	Überschreitet ein Request im entsprechenden Namespace eingestellte Limits, wird der Request abgelehnt und die Workload wird nicht erstellt.	✅

Durch entsprechende `ResourceQuota` Objekte von Kubernetes kann ein Namespace auf ein festes Limit quotiert werden. Überschreitet ein Request die im `ResourceQuota` angegebenen Werte, wird er vom entsprechenden validierenden Admission Controller abgelehnt und die Workload nicht erstellt.

FC-02	Überschreitet ein Request ein für mehrere Namespaces eingestelltes Limit, wird der Request abgelehnt und die Workload nicht erstellt.	✅

Mit den `HierarchicalResourceQuotas` des HNCs können durch eine Hierarchie zusammengehörige Namespaces quotiert werden. Überschreitet ein Request in einem der Namespaces dieses Limit, wird der Request durch den HNC abgelehnt.

FC-03	Überschreitet ein Request nach einem Pod die für ihn eingestellten Limits, wird der Request abgelehnt und der Pod nicht erstellt.	✅

Durch das `LimitRange` Objekt von Kubernetes können Pods in ihren Anfragen limitiert werden. In einem Namespace kann eine `LimitRange` mit Maximalwerten für Requests und Limits der Pods definiert werden. Überschreitet ein Request nach einem Pod diese Maximalwerte, wird der Pod vom entsprechenden Admission Controller abgelehnt.

FC-04	Ein angefragter Pod mit höherer Priorität wird früher einem Node zugeteilt, als zeitlich vor ihm angefragte Pods mit niedriger Priorität.	✅

Der Scheduler von Kubernetes sortiert Pods mit höherer Priorität in der Scheduling Queue vor niedriger priorisierten Pods ein. In der Regel werden diese damit auch früher Nodes zugeteilt. Allerdings kann das eingebaute Konzept des *back-off* dafür sorgen, dass nach einem erfolglosen Versuch, einen höher priorisierten Pod zu schedulen, ein niedriger priorisierter Pod vorgezogen wird. Dennoch wird das Kernziel des Qualitätsszenarios als erreicht betrachtet, da das Konzept des *back-off* die hier vereinfacht formulierte Priorisierung verbessert, indem es die Gesamteffizienz des Systems verbessert.

FC-05	Wenn eine Priorität in einem bestimmten Namespace ausgeschlossen wurde, kann dort kein Pod mit dieser Priorität erstellt werden.	✅

Mithilfe der Policy-Objekte von Kyverno kann eine Validierung erstellt werden, die Pods mit einer zuvor ausgeschlossenen Priorität in einem bestimmten Namespace ablehnt. Diese Regel kann dabei für einen Namespace (`Policy`) oder für das gesamte Cluster (`ClusterPolicy`) definiert werden. In den hier vorgestellten Konzepten wurde die Regel umgekehrt angewendet, indem bestimmte Prioritätsklassen prinzipiell abgelehnt werden und nur in ausgenommenen Namespaces erlaubt sind. In beiden Fällen verhindert Kyverno als dynamischer Admission Controller bei einem Verstoß das Erstellen des Pods.

FC-06	Wird ein Pod von einem bestimmten Ressourcenlimit ausgenommen, wird die Ausnahmeregel bevorzugt.	✅

Da das Konzept der `LimitRanges` von Kubernetes keine entsprechende Mechanik bietet, muss für eine solche Ausnahme das durch die `LimitRange` definierte Limit zuerst mit einer Kyverno `Policy` umgesetzt werden. Damit stehen die von Kyverno bereitgestellten Mechaniken zur Selektierung von Ressourcen zur Verfügung, um Pods aufgrund von speziellen Eigenschaften von der Limitierung auszuschließen *(Implementierung A.23)*. Auch die neu eingeführte Erweiterung HNC bietet Mechaniken zur Definition von Ausnahmeregelungen zur Vererbung eines Objekts. Dies ist über Annotationen am vererbten Objekt möglich. Danach können ausgenommene Namespaces eigene Definitionen der Objekte und damit auch für die `LimitRange` bestimmen.

FC-07	Wird ein Pod von einer Beschränkung der Priorität ausgeschlossen, wird die Ausnahmeregel bevorzugt und der Pod darf die im Namespace ausgeschlossene Priorität verwenden.	✅

Die Beschränkungen der Prioritätsklassen werden durch eine Kyverno `ClusterPolicy` definiert. Deshalb kann hierfür analog zur Lösung von FC-05 ein Pod über seine diversen Eigenschaften von der `ClusterPolicy` ausgenommen werden.

FC-08	Wurden Ressourcen reserviert, werden diese nur ausgewählten angefragten Workloads zugeteilt.	✅

Dieses Qualitätsszenario kann durch Platzhalterworkloads erreicht werden, die die reservierten Ressourcen blockieren. Dabei haben diese Platzhalterworkloads eine niedrigere Prioritätsklasse als die ausgewählten Workloads, welche die reservierten Ressourcen anfragen dürfen. Zusätzlich ist in der Prioritätsklasse der ausgewählten Workloads definiert, dass diese laufende niedriger priorisierte Workloads beenden dürfen, um sich selbst Platz zu machen. Dies wird durch die entsprechende Eigenschaft `preemptionPolicy` des `PriorityClass` Objekts von Kubernetes ermöglicht. Anschließend können die ausgewählten Workloads die laufenden Platzhalterworkloads beenden, um die reservierten Ressourcen zu beanspruchen.

FC-09	Es können aussagekräftige Metriken zur GPU-Auslastung des Cluster angezeigt werden. Dabei ist ersichtlich, welche konkreten Pods welche Grafikkarte nutzen und wie ausgelastet diese Grafikkarten sind.	☐

Die Funktionalität konnte nicht überprüft werden, da die Grafikkarten zum Zeitpunkt der Arbeit noch nicht wieder in die ICC integriert waren. Theoretisch sollte das Qualitätsszenario mit dem von NVIDIA bereitgestellten DCGM-Exporter möglich sein, da dieser auch Informationen zu den nutzenden Pods bereitstellen kann. Das Auslesen dieser Informationen kann anschließend über vordefinierte Prometheus Anfragen und damit auch graphisch über Grafana erfolgen.

FC-10	Als inaktiv erkannten Pods können identifizierbar aufgelistet werden.	✅

Durch die verfügbaren historischen Daten aus der Kombination von Prometheus und Grafana konnte eine erste mögliche Metrik zur Erkennung erstellt werden. Allerdings muss diese Metrik noch im laufenden Betrieb der ICC validiert werden, um sicherzustellen, dass diese zuverlässig tatsächlich inaktive Pods identifiziert.

FC-11	Ein Pod, der durch festgelegte Metriken als inaktiv erkannt wird, wird spätestens am nächsten Sonntag gelöscht.	☐

Dieses Szenario kann mit einem `CronJob` von Kubernetes realisiert werden. Dieser startet zum angegebenen Zeitpunkt einen Job, welcher ein entsprechendes Skript zur Löschung der identifizierten Pods startet. Das Szenario wurde jedoch noch nicht getestet, da zuerst die Metrik zur initialen Erkennung ausgiebig getestet werden muss.

FA-01	Alle ICC-Nutzenden haben einen **quotierten** Zugriff auf das Cluster.	✅

Die drei Anwendungen, mit denen die ICC-Nutzenden Zugriff auf das Cluster haben, sind Kiosk beziehungsweise der HNC, JupyterHub und GitLab Runner. JupyterHub und GitLab Runner bieten keinen direkten Zugriff auf das Cluster und sind bereits durch entsprechende Konfigurationen für alle Nutzenden limitiert. Durch die mit dem HNC eingeführte Hierarchie der Namespaces und dem erstellten hierarchischen Quotierungen ist limitiert, wie viele Ressourcen die einzelnen Nutzenden über egal wie viele Namespaces anfragen können.

FA-02	Das ICC-Team kann zeitkritische Pods schneller schedulen. Genauso können zeitunkritische Pods zurückgestellt werden, um keine dringlicher angefragten Pods zu blockieren.	✅

Durch die erstellen Prioritäten stehen den dem ICC-Team sowohl eine Priorität **über** und eine **unter** dem Standardwert zur Verfügung. Damit werden die zeitkritischen Pods vom Scheduler bevorzugt behandelt. Die niedrigere Priorität kann für zeitunkritische Pods verwendet werden, um vom Scheduler hinter alle höheren Anfragen sortiert zu werden.

FA-03	Durch die Konzepte werden die Pods von JupyterHub bevorzugt, die aufgrund eines geplanten Hochschulkurses angefragt werden.	✅

Durch die ergänzten Konfigurationen der erstellten Pods bekommen die Praktikums-Pods eine höhere Priorität und werden dadurch in der Regel früher einem Node zugeteilt, als die restlichen JupyterHub-Pods.

CC-01	Für die Nutzenden bleibt die Interaktion mit Kiosk unverändert, solange sie ihre Anfragen innerhalb der festgelegten Grenzen halten, keine unzulässigen Prioritäten setzen und keine Ressourcen anfordern, die nicht für sie reserviert sind.	☑

Auch im Parallelbetrieb zum HNC bleibt Kiosk vorerst die Anwendung, mit der die ICC-Nutzenden interagieren. Die HRQ, LimitRanges und Kyverno Regelungen zu Prioritäten greifen nur, falls die Anfragen dagegen verstoßen sollten, wodurch Kiosk in seiner Funktionalität nicht eingeschränkt wird. Einzig das Löschen der Spaces wird während des Parallelbetriebs verhindert. Da dies durch das Anpassen der Validierungskonfiguration des HNCs möglich wäre, jedoch in einer nachträglichen Entscheidung abgelehnt wurde, wird das Qualitätsziel als erfüllt betrachtet.

CC-02	Für die Nutzenden bleibt die Interaktion mit JupyterHub unverändert, so lange die Anfrage keine Ressourcen anfordert, die nicht für sie reserviert sind.	☑

Die Konzepte greifen nur in den genannten Fällen und ansonsten wurde nicht in die Funktionsweise von JupyterHub eingegriffen.

UL-01	Das ICC-Team kann die Konzepte selbstständig, effektiv und effizient anwenden, da die Funktionsweisen leicht verständlich sind und keine längere Einarbeitung voraussetzen.	☑

Die eingeführten Konzepte sind zum einen direkt über Kubernetes steuerbar, wodurch die Interaktion vertraut ist. Der HNC ist auch ein eigenes Kubernetes-Projekt und arbeitet mit den aus Kubernetes bekannten Syntax von YAML-Konfigurationen und CRDs. Zusätzlich kann die eigene CLI-Erweiterung installiert werden, um die Objekte mit vereinfachten Terminal-Befehlen zu erstellen. Kyverno arbeitet ebenfalls mit den bekannten YAML-Manifesten. Alle Policy-Objekte sind als CRD im Cluster festgehalten und können jederzeit ausgelesen werden. Es existiert zudem eine ausgiebige Dokumentation an bereits vorgefertigten und häufig verwendeten Policies. Diese können direkt verwendet

werden oder als Vorlage für eigene Regelungen dienen und erleichtern die Umsetzung individueller neuer Regeln. Der DCGM-Exporter arbeitet zusammen mit Prometheus und Grafana, welche beide bereits im Cluster verfügbar und bekannt sind. Sobald die Installation abgeschlossen ist, können die erstellten Metriken durch die dokumentierten Metriken von NVIDIA einfach angepasst werden.

UL-02	ICC-Nutzende werden so weit wie möglich über die aktiven Einschränkungen informiert.	✅

Vor allem für die Nutzenden von Kiosk beziehungsweise des HNCs werden entsprechende Hinweise auf der offiziellen ICC-Dokumentationsseite bereitgestellt. Diese erklären, welche Einschränkungen es gibt, wie sich diese äußern und helfen bei entsprechenden Fehlermeldungen. Zudem zeigen HRQs und validierende Regelungen von Kyverno aussagekräftige Fehlermeldungen bei einem Verstoß an.

UO-01	Die Limits für Pods können für einen Namespace gesammelt eingestellt werden.	✅

Dies ist möglich durch das Erstellen einer `LimitRange` im gewünschten Namespace. Danach kontrolliert der `LimitRanger` Admission Controller jeden in diesem Namespace angefragten Pod auf die in der `LimitRange` angegebenen Werte zu Rechenressourcen.

UO-02	Die Metriken zur GPU-Auslastung können visuell verständlich dargestellt werden, statt über reine Tabellen und Zahlenwerte.	☐

Die durch den DCGM-Exporter zur Verfügung gestellten Metriken können theoretisch durch eine entsprechende Prometheus-Anfrage ausgelesen werden. Anschließend kann Grafana diese Daten auch in Graphen oder anschauliche Statistiken überführen. Dabei kann das von NVIDIA zur Verfügung gestellte Dashboard als erster Ansatz dienen.

UO-03	Das Definieren von Ausnahmen ist eindeutig mit der betroffenen Regelung verknüpft. Eine Einsicht in die Regelung zeigt direkt, ob eine Ausnahme definiert ist.	✅

Ausnahmen werden für Kyverno `Policies` und für vom HNC vererbte Objekte definiert. Wenn Ausnahmen für bestimmte Kyverno `Policies` definiert sind, stehen diese

in den entsprechenden Abschnitten der Objektkonfiguration selbst. Die Ausnahmen für den HNC sind durch Annotationen am obersten Objekt, das vererbt wird, einsehbar.

MM-01	Limits für Pods und Namespaces können durch maximal eine Anpassung an einer definierten Stelle effektiv angepasst werden.	✅

Alle Regelungen bezüglich der Limits werden durch entsprechende Objekte definiert (`ResourceQuotas` , `LimitRanges` , `HierarchicalResourceQuotas`). Die Regelungen können durch eine Modifikation der entsprechenden Objekte im Cluster direkt angepasst werden.

MM-02	Prioritäten und Regeln zu erlaubten Prioritäten können durch maximal eine Anpassung an einer definierten Stelle effektiv angepasst werden.	✅

Alle Regelungen bezüglich Prioritäten werden durch `Policy` oder `ClusterPolicy` Objekte definiert. Diese können direkt im Cluster in einem Schritt bearbeitet werden, um die Regelungen anzupassen.

MM-03	Das Konzept zur Erkennung inaktiver Pods bietet eine Mechanik, mit der die Metrik zur Erkennung im Nachhinein angepasst werden kann.	✅

Die Metrik besteht aus selbst definierten Prometheus Anfragen. Dabei können nicht nur die Anfragen selbst angepasst und weiterentwickelt werden, sondern bei Bedarf auch die Prometheus zur Verfügung gestellten Daten erweitert werden.

6.3 Ergebnisse

Die Qualitätsszenarien sind fast vollständig erfüllt. Lediglich FC-08 und UO-02 konnten nicht getestet werden, da die beschriebene Funktionsweise ohne ein Cluster mit GPU-Ressourcen nicht getestet werden konnte. Um FC-09 wie gewünscht zu erfüllen, muss die beschriebene Metrik noch über einen längeren Zeitraum und mit mehr aktiven Nutzenden evaluiert werden.

Durch die angewendeten Konzepte der Limitierung und Quotierung der Rechenressourcen und API-Ressourcen für Pods, Namespaces und zusätzlichen Hierarchien von Namespaces stehen dem ICC-Team effektive Mechaniken zur präzisen Kontrolle des Ressourcenverbrauchs der ICC zur Verfügung. Mit der Ablösung von Kiosk durch den HNC wird zusätzlich eine nachhaltige Multi-Tenancy Anwendung in die ICC integriert. Durch die Übergangsphase können bestehende Ressourcen der Nutzenden in die neue Struktur übertragen werden, um anschließend Kiosk ohne Verluste löschen zu können.

Die hier beschriebenen Werte zu Limitierungen sind nur Annäherungen. Gut gewählte Limits sind abhängig von vielen Faktoren, wie die generelle Kapazität des Clusters, die Anzahl der indirekten Nutzenden und die spezifischen Anforderungen an die Applikationen, die von diesen erstellt werden. Aus diesem Grund müssen die Limits im Vergleich zum tatsächlichen Verbrauch der ICC weiter beobachtet und angepasst werden, um die Effizienz des Clusters stetig zu verbessern.

Durch die Möglichkeit der Priorisierung kann zusätzlich das Scheduling der Workloads beeinflusst werden. Dadurch haben wichtige Workloads priorisierten Zugriff auf die verfügbaren Ressourcen der ICC. Im gleichen Zuge können Workloads auch niedriger priorisiert werden, um das Cluster zu Hochzeiten nicht zusätzlich zu belasten.

Mit Kyverno als Policy Engine können validierende, mutierende und generative Regelungen erstellt werden. Diese helfen nicht nur bei der Verfeinerung der Quotierung und Priorisierung, sondern auch dabei, die Konzepte mit den bereits bestehenden Anwendungen zu integrieren.

Durch Platzhalterworkloads können Rechenressourcen für geplante Veranstaltungen frühzeitig reserviert werden.

Mit dem von NVIDIA vorgestellten DCGM-Exporter können in Zukunft geeignete Metriken für die Auslastung der GPU-Ressourcen erstellt werden. Die Erkennung von inaktiven Pods kann durch entsprechende Langzeitdaten der Container erfolgen, die über Prometheus ausgelesen werden können. Die hier vorgestellte Metrik soll einen ersten Entwurf zeigen und muss über einen längeren Zeitraum im laufenden Betrieb der ICC mit mehr aktiven Nutzenden getestet werden. Die Metrik kann auf die Anforderungen der ICC angepasst und erweitert werden. Sobald die Metrik zuverlässige Daten liefert, kann die Löschung von inaktiven Pods durch ein Skript automatisiert werden.

7 Fazit

In einem Multi-Tenancy Cluster ist ein effektives Ressourcenmanagement essenziell, um eine faire Verteilung zu sichern. Da auf dem hier untersuchten Cluster verschiedenen Anwendungen und Nutzende arbeiten, sollte ein Konzept zur Quotierung und Priorisierung erstellt werden, um zu helfen, die Ressourcenverteilung besser zu steuern. Dabei standen die Anwendungen im Fokus, welche potenziell vielen Nutzenden ermöglichen, Zugriff auf die Clusterressourcen zu erhalten. Mit der Erkennung inaktiver Pods und einer Metrik zur GPU-Auslastung sollte die Ressourcenverteilung zusätzlich überwacht werden.

Während sich bisherigen Arbeiten auf sehr spezielle Details der Ressourcenverteilung selbst konzentrieren, betrachtet diese Arbeit das Cluster auf der Ebene der unterschiedlichen Anwendungen und Nutzenden. Dabei ist ein Konzept bestehend aus mehreren Teillösungen entstanden. Auch wenn dieses Konzept auf das Cluster der HAW abgestimmt ist, können die Teillösungen auch auf andere Cluster übertragen werden.

Den Ressourcenverbrauch zu beschränken kann für jede Art von Cluster hilfreich sein, um mit den verfügbaren Ressourcen besser planen zu können. Die Quotierungen können dabei auch für dynamische Cluster eingesetzt werden, um die Kosten einzugrenzen. Mit den hier gezeigten Quotierungsmaßnahmen können sowohl Pods, Namespaces und sogar Gruppen von Namespaces in ihrem Ressourcenverbrauch eingeschränkt werden. Dies erlaubt eine präzisere Kontrolle und Verwaltung der Ressourcen, was zu einer allgemein effizienteren Nutzung und faireren Verteilung führt. Die Reservierung von Ressourcen kann zusätzlich eingesetzt werden, um Ressourcen gezielt verfügbar zu halten.

Eine Priorisierung kann etabliert werden, um die zeitliche Komponente der Ressourcenzuteilung zu steuern. Ressourcen können geplant schneller zugeteilt werden und es kann verhindert werden, dass wichtigere Workloads von unwichtigeren aufgehalten werden. Zusätzlich können zeitunkritische Anfragen temporär zurückgestellt werden, um die Ressourcen nicht unnötig zu belasten. Dies betrifft in dieser Arbeit vor allem administrative Anfragen und solche, die indirekt über Anwendungen gestellt werden.

Inaktivität in einem Cluster zu erkennen, kann helfen, geeignete Maßnahmen zu ergreifen, um die Effizienz des Clusters zu verbessern. Mit historischen Auslastungsdaten über Prometheus sollen inaktive Pods erkannt werden. Die hier dargestellte Metrik soll dabei einen ersten Ansatzpunkt liefern. Für eine tatsächliche automatisierte Löschung müssen die Ergebnisse noch über einen längeren Zeitraum getestet und abgestimmt werden.

Eine aussagekräftige Anzeige der GPU-Auslastung ist für alle Cluster mit ergänzenden GPU-Ressourcen wichtig, um ein gesamtheitliches Bild der Ressourcennutzung abzubilden. Der DCGM-Exporter liefert für die GPUs von NVIDIA theoretisch aussagekräftige Metriken zur Auslastung. Hier muss die beschriebene Integration noch verifiziert werden, sobald die GPU-Ressourcen wieder bereitstehen.

Ergänzend wird für das betrachtete Cluster die Multi-Tenancy Anwendung Kiosk durch den HNC von Kubernetes abgelöst. Dies kann auch anderen Clustern einen Ansatz bieten, den HNC als Multi-Tenancy Lösung zu integrieren oder um ebenfalls von einem vorhandenen System umzusteigen.

7.1 Ausblick

Durch eine Quotierung wird zukünftig unnötigen Überprovisionierungen auf Kosten der anderen ICC-Nutzenden und Anwendungen entgegengesteuert. Dadurch wird die Verteilung fairer und das Nutzungserlebnis der ICC für alle verbessert. Die Administrationsebene kann durch die Maßnahmen weniger Zeit in das Auflösen von Ressourcenkonflikten stecken und dafür daran arbeiten Muster in der Ressourcennutzung selbst erkennen. Dadurch können entsprechend weiterführende Maßnahmen geplant werden, um die Stabilität der Anwendungen und des ganzen Systems weiter zu verbessern. Durch den Wegfall von Kiosk, wird der Administrationsaufwand verringert und die ICC kann von der aktiven Weiterentwicklung der von Kubernetes entwickelten Erweiterung profitieren.

Auf technischer Seite bieten sich zusätzlich zu den ausstehenden Tests zur Erkennung inaktiver Pods und der GPU-Auslastung noch weitere Möglichkeiten, das hier beschriebene Konzept zur Quotierung und Priorisierung weiter auszubauen.

Auswertung der Quotierungen
Die hier vorgestellten Werte der unterschiedlichen Quotierungen müssen im weiteren Betrieb der ICC getestet und ausgewertet werden. Dabei könnte eine weitere Metrik helfen,

die Qualität der definierten Limits zu bewerten. Das Ziel ist es, eine Quotierung aufzustellen, sodass alle Workloads flüssig laufen, ohne zu viel Ressourcen zuzuteilen. Diese Balance in einer Quotierung festzuhalten stellt eine große Herausforderung dar. Dabei kann es hilfreich sein, das Verhältnis der verfügbaren Ressourcen mit den angefragten und genutzten Ressourcen gegenüberstellen [21]. Damit können potenziell überprovisionierte Ressourcen sichtbar gemacht werden, um effektive Gegenmaßnahmen zu steuern. Entsprechende Formeln zur Berechnung könnten über Prometheus formuliert werden und in Grafana überwacht werden.

JupyterHub mit Jobs starten

Wie in der Arbeit von Lee u. a. [59] vorgestellt, könnte ein offline Scheduling für JupyterHub die Verfügbarkeit der GPU-Ressourcen verbessern. Dabei können die GPU-Ressourcen nur durch einen `Job` über die Notebooks angefragt werden. Diese Jobs werden über eine clusterweite Queue abgearbeitet. Dieser Ansatz kann helfen, die ressourcenintensiven ML-Prozesse auf eigene Jobs auszulagern, die dadurch nacheinander abgearbeitet werden können. Dies würde jedoch einen größeren Umbau der Infrastruktur rund um JupyterHub und das Einführen weiterer Tools erfordern.

Literaturverzeichnis

[1] arc42 Documentation: arc42 Documentation. 10. Quality Requirements, GitHub, Inc., 2023. – URL https://docs.arc42.org/section-10/. – Zugriffsdatum: 15.12.2023. – This work is licensed under the Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/4.0/.

[2] Beltre, Angel ; Saha, Pankaj ; Govindaraju, Madhusudhan: Framework for Analysing a Policy-driven Multi-Tenant Kubernetes Environment. In: *2021 IEEE Cloud Summit (Cloud Summit)*, 2021, S. 49–56

[3] Broadcom Inc.: *Bitnami package for WordPress. Version 19.2.0.* 2024. – URL https://artifacthub.io/packages/helm/bitnami/wordpress?modal=values. – Zugriffsdatum: 22.01.2024. – Open-source project to expose GPU metrics. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0

[4] Clay, Risser: *wordpress. Version 0.0.1.* 2024. – URL https://artifacthub.io/packages/helm/rock8s/wordpress?modal=value. – Zugriffsdatum: 22.01.2024

[5] Cloud Native Computing Foundation: *Kubernetes (K8s). Version v1.29.0.* 2023. – URL https://github.com/kubernetes/kubernetes. – Open-source Container Orchestration System. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0.

[6] Cluster Lifecycle Special Interest Group: *minikube. Version v1.32.0.* 2023. – URL https://github.com/kubernetes/minikube. – Zugriffsdatum: 02.25.2024. – Open-source-Tool to run a local Kubernetes cluster. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0.

[7] Şenel, Berat C. ; Mouchet, Maxime ; Cappos, Justin ; Fourmaux, Olivier ; Friedman, Timur ; McGeer, Rick: EdgeNet: A Multi-Tenant and Multi-Provider Edge Cloud. In: *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*. New York, NY, USA : Association for Computing Machinery, 2021 (EdgeSys '21), S. 49–54. – URL https://doi.org/10.1145/3434770.3459737. – ISBN 9781450382915

[8] Datadog: *Kubernetes runs in half of container environments.*. – URL https://www.datadoghq.com/container-report-2020/. – Zugriffsdatum: 11.02.2024

[9] Dinesh, Sandeep: Kubernetes best practices: Resource requests and limits, Google Cloud, Blog, 2018. – URL https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits. – Zugriffsdatum: 22.12.2023

[10] Felter, Wes ; Ferreira, Alexandre ; Rajamony, Ram ; Rubio, Juan: An updated performance comparison of virtual machines and Linux containers. In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, S. 171–172

[11] Gao, Peng: A Predictive Autoscaler for Elastic Batch Jobs. In: *CoRR* abs/2010.05049 (2020). – URL https://arxiv.org/abs/2010.05049

[12] GitHub, Inc.: *GitLab Docs. Kubernetes executor. Version v16.8*. 2023. – URL https://docs.gitlab.com/runner/executors/kubernetes/index.html. – Zugriffsdatum: 23.12.2023. – DevSecOps Platform.

[13] Google Ireland Limited: *Provision extra compute capacity for rapid Pod scaling*. 2024. – URL https://cloud.google.com/kubernetes-engine/docs/how-to/capacity-provisioning. – Zugriffsdatum: 23.01.2024. – This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0

[14] Göran, Pöhner: *Wordpress. Version 0.10.5*. 2023. – URL https://artifacthub.io/packages/helm/groundhog2k/wordpress?modal=values. – Zugriffsdatum: 22.01.2024

[15] Hamzeh, Hamed ; Meacham, Sofia ; Khan, Kashaf: A New Approach to Calculate Resource Limits with Fairness in Kubernetes. In: *2019 First International Conference on Digital Data Processing (DDP)*, 2019, S. 51–58

[16] Helm Authors.: Using Helm, The Linux Foundation., 2024. – URL https://helm.sh/docs/intro/using_helm/. – Zugriffsdatum: 20.01.2024

[17] Hilman, Muhammad H. ; Rodriguez, Maria A. ; Buyya, Rajkumar: Multiple Workflows Scheduling in Multi-Tenant Distributed Systems: A Taxonomy and Future Directions. In: *ACM Comput. Surv.* 53 (2020), feb, Nr. 1. – URL https://doi.org/10.1145/3368036. – ISSN 0360-0300

[18] Informationstechnik, Bundesamt für Sicherheit in der: *Cloud Computing Grundlagen.* – URL https://www.bsi.bund.de/dok/6622124. – Zugriffsdatum: 30.01.2024

[19] Jayaram, K. R. ; Muthusamy, Vinod ; Dube, Parijat ; Ishakian, Vatche ; Wang, Chen ; Herta, Benjamin ; Boag, Scott ; Arroyo, Diana ; Tantawi, Asser ; Verma, Archit ; Pollok, Falk ; Khalaf, Rania: FfDL: A Flexible Multi-Tenant Deep Learning Platform. In: *Proceedings of the 20th International Middleware Conference.* New York, NY, USA : Association for Computing Machinery, 2019 (Middleware '19), S. 82–95. – URL https://doi.org/10.1145/3361525.3361538. – ISBN 9781450370097

[20] Jyothi, Sangeetha A. ; Curino, Carlo ; Menache, Ishai ; Narayanamurthy, Shravan M. ; Tumanov, Alexey ; Yaniv, Jonathan ; Mavlyutov, Ruslan ; Goiri, Inigo ; Krishnan, Subru ; Kulkarni, Janardhan ; Rao, Sriram: Morpheus: Towards Automated SLOs for Enterprise Clusters. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16).* Savannah, GA : USENIX Association, November 2016, S. 117–134. – URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/jyothi. – ISBN 978-1-931971-33-1

[21] Kleemaier, Gerhard: SLOs for Kubernetes clusters: Optimize resource utilization of Kubernetes clusters with SLOs, Dynatrace LLC, 2023. – URL https://www.dynatrace.com/news/blog/optimize-resource-utilization-of-your-kubernetes-clusters-with-slos/. – Zugriffsdatum: 29.01.2024

[22] Kramm, Fabian: Commit 0a94288, Loft Labs, Inc., 2022. – URL https://github.com/loft-sh/kiosk/commit/0a94288f6a1b2a18a4a206c9846714926a07ccde. – Zugriffsdatum: 20.02.2024. – Multi-Tenancy Extension For Kubernetes. This work is licensed under the Apache License Version 2.0. To view a

copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0

[23] Krebs, Rouven ; Momm, Christof ; Kounev, Samuel: Architectural Concerns in Multi-tenant SaaS Applications. Walldorf, Karlsruhe, GER : SciTePress, 2012. – URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/jyothi. – ISBN 978-1-931971-33-1

[24] Kubernetes Authors, The: Authorization Overview, The Linux Foundation, 2022. – URL https://kubernetes.io/docs/concepts/storage/persistent-volumes/. – Zugriffsdatum: 24.12.2023

[25] Kubernetes Authors, The: Admission Controllers Reference, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/. – Zugriffsdatum: 21.12.2023

[26] Kubernetes Authors, The: Authorization Overview, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/workloads/pods/. – Zugriffsdatum: 25.12.2023

[27] Kubernetes Authors, The: Container Runtimes, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/setup/production-environment/container-runtimes/. – Zugriffsdatum: 02.12.2024

[28] Kubernetes Authors, The: Controlling Access to the Kubernetes API, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/security/controlling-access/. – Zugriffsdatum: 21.12.2023

[29] Kubernetes Authors, The: Extending Kubernetes, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/extend-kubernetes/. – Zugriffsdatum: 21.12.2023

[30] Kubernetes Authors, The: Jobs, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/workloads/controllers/job/. – Zugriffsdatum: 19.01.2024

[31] Kubernetes Authors, The: kube-apiserver, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/. – Zugriffsdatum: 18.12.2023

[32] Kubernetes Authors, The: The Kubernetes API, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/overview/kubernetes-api/. – Zugriffsdatum: 20.12.2023

[33] Kubernetes Authors, The: Kubernetes API Concepts, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/reference/using-api/api-concepts/. – Zugriffsdatum: 23.12.2023

[34] Kubernetes Authors, The: Kubernetes Components, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/overview/components/. – Zugriffsdatum: 05.02.2024

[35] Kubernetes Authors, The: Kubernetes Scheduler, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/. – Zugriffsdatum: 02.01.2024

[36] Kubernetes Authors, The: Limit Ranges, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/policy/limit-range/. – Zugriffsdatum: 23.12.2023

[37] Kubernetes Authors, The: Multi-tenancy, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/security/multi-tenancy/. – Zugriffsdatum: 06.02.2024

[38] Kubernetes Authors, The: Namespaces, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. – Zugriffsdatum: 05.02.2024

[39] Kubernetes Authors, The: Node-pressure Eviction, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/scheduling-eviction/node-pressure-eviction/. – Zugriffsdatum: 21.02.2024

[40] Kubernetes Authors, The: Objects In Kubernetes, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/overview/working-with-objects/. – Zugriffsdatum: 05.02.2024

[41] Kubernetes Authors, The: Pod Priority and Preemption, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/. – Zugriffsdatum: 02.01.2024

[42] Kubernetes Authors, The: Policies, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/policy/. – Zugriffsdatum: 23.12.2023

[43] Kubernetes Authors, The: Resource Management for Pods and Containers, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/. – Zugriffsdatum: 12.01.2024

[44] Kubernetes Authors, The: Resource Quotas, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/policy/resource-quotas/. – Zugriffsdatum: 03.02.2024

[45] Kubernetes Authors, The: Schedule GPUs, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/. – Zugriffsdatum: 25.12.2023

[46] Kubernetes Authors, The: Using NVIDIA GPUs with minikube, The Linux Foundation, 2023. – URL https://minikube.sigs.k8s.io/docs/tutorials/nvidia/. – Zugriffsdatum: 25.02.2024

[47] Kubernetes Authors, The: Workloads, The Linux Foundation, 2023. – URL https://kubernetes.io/docs/concepts/workloads/. – Zugriffsdatum: 05.02.2024

[48] Kubernetes Authors, The: Service, The Linux Foundation, 2024. – URL https://kubernetes.io/docs/concepts/services-networking/service/. – Zugriffsdatum: 15.03.2024

[49] Kubernetes Authors, The: Tools for Monitoring Resources, The Linux Foundation, 2024. – URL https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-usage-monitoring/. – Zugriffsdatum: 01.02.2024

[50] Kyverno Authors, The: Kyverno Documentation. Applying Policies., The Kyverno Project, 2023. – URL https://kyverno.io/docs/applying-policies/. – Zugriffsdatum: 09.01.2024

[51] Kyverno Authors, The: Kyverno Documentation. Installation., The Kyverno Project, 2023. – URL https://kyverno.io/docs/installation/. – Zugriffsdatum: 15.12.2023

[52] Kyverno Authors, The: Kyverno Documentation. Version v1.11.0, The Kyverno Project, 2023. – URL https://kyverno.io/docs/. – Zugriffsdatum: 14.01.2024

[53] Kyverno Authors, The: Kyverno Documentation. Configuring Kyverno., The Kyverno Project, 202j. – URL https://kyverno.io/docs/installation/customization/. – Zugriffsdatum: 22.01.2024

[54] Kyverno Project: *kyverno. Version v1.11.4*. 2023. – URL https://github.com/kyverno/kyverno/. – Zugriffsdatum: 14.01.2024. – Open-source Kubernetes Native Policy Management. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0.

[55] Labor für allgemeine Informatik: Dokumentations-Wiki des Departments Informatik der HAW Hamburg. Informationen, die Labor-übergreifend gelten., Fakultät TI - Department Informatik., 2023. – URL https://userdoc.informatik.haw-hamburg.de/doku.php?id=allgemein:start. – Zugriffsdatum: 20.12.2023

[56] Labor für allgemeine Informatik: ICC - Informatik Compute Cloud, HAW Hamburg, 2023. – URL https://icc.informatik.haw-hamburg.de/. – Zugriffsdatum: 20.12.2023

[57] Labs., Grafana: Grafana. Version 10.4.0., URL https://github.com/grafana/grafana. – Zugriffsdatum: 08.03.2024, 2024. – Open-source tool for querying and visualizing time series and metrics. This work is licensed under the GNU Affero General Public License v3.0 only. To view a copy of this license, visit https://spdx.org/licenses/AGPL-3.0-only.html

[58] Lardinois, Frederic: *As Kubernetes Hits 1.0, Google Donates Technology To Newly Formed Cloud Native Computing Foundation*. 2015. – URL https://techcrunch.com/2015/07/21/as-kubernetes-hits-1-0-google-donates-technology-to-newly-formed-cloud-native-computing-foundation-with-ibm-intel-twitter-and-others/. – Zugriffsdatum: 02.12.2024

[59] Lee, Chun-Hsiang ; Li, Zhaofeng ; Lu, Xu ; Chen, Tiyun ; Yang, Saisai ; Wu, Chao: Multi-Tenant Machine Learning Platform Based on Kubernetes. In: *Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence*. New York, NY, USA : Association for Computing Machinery, 2020 (ICCAI '20), S. 5–12. – URL https://doi.org/10.1145/3404555.3404565. – ISBN 9781450377089

[60] Loft Labs, Inc.: kiosk – Multi-Tenancy Extension For Kubernetes. Version v0.2.11, URL https://github.com/loft-sh/kiosk. – Zugriffsdatum: 12.01.2024, 2022

[61] Loft Labs, Inc.: *jspolicy. Version v0.2.2.* 2023. – URL https://github.com/loft-sh/jspolicy. – Zugriffsdatum: 14.01.2024. – Open-source Projekt for Policies using JavaScript or TypeScript. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0.

[62] Loft Labs, Inc.: *Why jsPolicy?. Version v0.2.2.* 2023. – URL https://www.jspolicy.com/docs/quickstart. – Zugriffsdatum: 14.01.2024

[63] Ludwin, Adrian: Introducing Hierarchical Namespaces, The Kubernetes Authors, The Linux Foundation, 2020. – URL https://kubernetes.io/blog/2020/08/14/introducing-hierarchical-namespaces/. – Zugriffsdatum: 22.12.2023

[64] Ludwin, Adrian ; Bezdicek, Ryan ; Rampal, Sanjeev ; Tasha: HNC: Concepts. Version v1.2, GitHub, Inc., 2023. – URL https://github.com/kubernetes-sigs/hierarchical-namespaces/blob/master/docs/user-guide/concepts.md. – Zugriffsdatum: 24.12.2023

[65] Ludwin, Adrian ; Bezdicek, Ryan ; Rampal, Sanjeev ; Tasha: The Hierarchical Namespace Controller (HNC), GitHub, Inc., 2024. – URL https://github.com/kubernetes-sigs/hierarchical-namespaces. – Zugriffsdatum: 11.02.2024. – Multi-Tenancy Extension For Kubernetes. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0.

[66] NVIDIA Corporation.: *NVIDIA DCGM Exporter Dashboard.* 2021. – URL https://grafana.com/grafana/dashboards/12239-nvidia-dcgm-exporter-dashboard/. – Zugriffsdatum: 21.01.2024

[67] NVIDIA Corporation: *About GPU Telemetry.* 2024. – URL https://docs.nvidia.com/datacenter/cloud-native/gpu-telemetry/latest/about-telemetry.html. – Zugriffsdatum: 01.20.2024

[68] NVIDIA Corporation: *DCGM Exporter*. 2024. – URL https://docs.nvidia.com/datacenter/cloud-native/gpu-telemetry/latest/dcgm-exporter.html. – Zugriffsdatum: 01.20.2024

[69] NVIDIA Corporation.: DCGM-Exporter. Version 3.3.0-3.2.0, URL https://github.com/NVIDIA/dcgm-exporter. – Zugriffsdatum: 20.01.2024, 2024. – Open-source project to expose GPU metrics. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0

[70] NVIDIA Corporation: *GPU Operator. Version 23.9.1*. 2024. – URL https://github.com/NVIDIA/gpu-operator. – Zugriffsdatum: 01.03.2024. – Open-source project to manage GPU nodes. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0

[71] NVIDIA Corporation.: *NVIDIA Data Center GPU Manager. Version v3.3.5*. 2024. – URL https://github.com/NVIDIA/DCGM. – Zugriffsdatum: 18.03.2024. – Suite of tools for managing and monitoring NVIDIA datacenter GPUs in cluster environments. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0.

[72] NVIDIA Corporation: *Setting up Prometheus*. 2024. – URL https://docs.nvidia.com/datacenter/cloud-native/gpu-telemetry/latest/kube-prometheus.html. – Zugriffsdatum: 01.20.2024

[73] Open Policy Agent. Gatekeeper Project: *Gatekeeper. Version v.3.14.0*. 2023. – URL https://open-policy-agent.github.io/gatekeeper/website/docs/. – Zugriffsdatum: 14.01.2024

[74] Open Policy Agent Gatekeeper Project: *gatekeeper. Version v.3.14.0*. 2023. – URL https://github.com/open-policy-agent/gatekeeper. – Zugriffsdatum: 14.01.2024. – Open-source Policy Controller for Kubernetes. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0.

[75] Pramod, Ramarao ; Ahmed, Al-Sudani ; Swati, Gupta: Monitoring GPUs in Kubernetes with DCGM, NVIDIA Corporation, NVIDIA Developer. Technical Blog.,

2020. – URL https://developer.nvidia.com/blog/monitoring-gpus-in-kubernetes-with-dcgm/. – Zugriffsdatum: 01.13.2024

[76] Project Jupyter: *Kubespawner. Version 6.1.0.* 2021. – URL https://jupyterhub-kubespawner.readthedocs.io/en/latest/. – Zugriffsdatum: 23.12.2023

[77] Project Jupyter: *KubeSpawner. Version 6.1.0.* 2021. – URL https://jupyterhub-kubespawner.readthedocs.io/en/latest/spawner.html. – Zugriffsdatum: 02.01.2024

[78] Project Jupyter: *JupyterHub. Version 4.0.2.* 2024. – URL https://github.com/jupyterhub/jupyterhub. – Zugriffsdatum: 01.03.2024. – Open-source multi-user Hub for single-user Jupyter notebooks. Licensed under the 3-Clause BSD License. To view a copy of this license, visit https://opensource.org/license/bsd-3-clause/.

[79] Project Jupyter: *JuypterHub.* 2024. – URL https://jupyter.org/hub. – Zugriffsdatum: 14.01.2024

[80] Prometheus Monitoring Community.: *Prometheus. Version 2.50.1.* 2024. – URL https://github.com/prometheus/prometheus. – Zugriffsdatum: 08.03.2024. – Open-source Monitoring Service. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0

[81] Red Hat: *Ceph. Version v16.2.15.* 2024. – URL https://github.com/ceph/ceph. – Zugriffsdatum: 01.03.2024. – Open-source distributed storage system. Licensed under the GPL-2.0, LGPL-2.1 and LGPL-3.0 license To view a copy of these licenses, visit https://www.gnu.org/licenses/old-licenses/gpl-2.0.html, https://www.gnu.org/licenses/old-licenses/lgpl-2.1.de.html, https://www.gnu.org/licenses/lgpl-3.0.de.html.

[82] Rejiba, Zeineb ; Chamanara, Javad: Custom Scheduling in Kubernetes: A Survey on Common Problems and Solution Approaches. In: *ACM Comput. Surv.* 55 (2022), dec, Nr. 7. – URL https://doi.org/10.1145/3544788. – ISSN 0360-0300

[83] Riftbit: *WordPress. Version 12.1.16.* 2021. – URL https://artifacthub.io/packages/helm/riftbit/wordpress?modal=values. – Zugriffsdatum: 22.01.2024

[84] Rodriguez, Maria A. ; Buyya, Rajkumar: *Container-based Cluster Orchestration Systems: A Taxonomy and Future Directions*. 2018

[85] Rook Contributors: *Rook. Version v1.13.5*. 2024. – URL https://github.com/rook/rook. – Zugriffsdatum: 01.03.2024. – Open-source cloud-native storage orchestrator for Kubernetes. This work is licensed under the Apache License Version 2.0. To view a copy of this license, visit https://www.apache.org/licenses/LICENSE-2.0

[86] Roth, Felix: *Monitoring eines Multi-Tenancy Kubernetes Clusters am Beispiel der HAW Hamburg Compute Cloud*, Hochschule für Angewandte Wissenschaften Hamburg. Fakultät Technik und Informatik. Department Informatik., Bachelorarbeit, November 2020

[87] Singh, Sachchidanand ; Singh, Nirmala: Containers & Docker: Emerging roles & future of Cloud technology. In: *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, 2016, S. 804–807

[88] Stäps, Florian: *Sicherheitskonzeption für eine mandantenfähige Cloudumgebung*, Hochschule für Angewandte Wissenschaften Hamburg. Fakultät Technik und Informatik. Department Informatik., Masterarbeit, November 2020

[89] Trzciński, Kamil ; GitLab Inc.: *GitLab Runner. Version v16.9.1*. 2024. – URL https://gitlab.com/gitlab-org/gitlab-runner. – Zugriffsdatum: 01.03.2024. – Open-source continuous integration service. Licensed under the MIT license. To view a copy of this license, visit https://gitlab.com/gitlab-org/gitlab-runner/blob/main/LICENSE.

A Anhang

A.1 Use-Cases

ID	UC-1
Titel	Limits für Namespaces konfigurieren
Akteur*in	ICC-Teammitglied
Ziel	Das Erstellen von Namespace-API-Ressourcen unterliegt den für diesen Namespace eingestellten Limits für Rechenressourcen und API-Ressourcen
Vorbedingung	1. ICC-Teammitglied ist in der Rolle „Admin“ bei der ICC angemeldet (autorisiert und authentifiziert).
Nachbedingung	1. Es gibt nicht mehr als die vom Limit vorgegebenen Anzahl an API-Ressourcen. 2. Die Summe der angefragten Rechenressourcen der einzelnen Workloads eines Namespaces überschreitet nicht das eingestellte Limit.
Erfolgsszenario	1. ICC-Teammitglied erstellt ein Limit für einen Namespace. 2. Ein Request geht ein, eine API-Ressource in diesem Namespace zu erstellen, übersteigt weder die Limits für API-Ressourcen noch die Limits für die akkumulierten Rechenressourcen des Namespaces. 3. Der Request wird genehmigt und die API-Ressource erstellt.
Erweiterungsfall	2a. Der Request, eine neue API-Ressource in diesem Namespace zu erstellen, würde das Limit des Namespaces für diese Art von API-Ressourcen oder Rechenressourcen übersteigen. 1. Der Request wird abgelehnt und die API-Ressource nicht erstellt.
Beispiel	Ein ICC-Teammitglied will, dass das es maximal 10 Deployments in einem Namespace gibt und der Namespace selbst maximal 100 GB beansprucht. Nachdem die Person ein einsprechendes Limit eingestellt hat, können im entsprechenden Namespace maximal 10 Deployments erstellt werden und die Rechenressourcen übersteigen nie die 100 GB.

ID	UC-2
Titel	Limits für zusammengehörige Namespaces konfigurieren
Akteur*in	ICC-Teammitglied
Ziel	Logisch zusammengehörtige Namespaces können mit einem Limit in ihren aggregierten Ressourcen beschränkt werden.
Vorbedingung	1. ICC-Teammitglied ist in der Rolle „Admin“ bei der ICC angemeldet (autorisiert und authentifiziert).
Nachbedingung	1. In allen Namespaces gesammelt, gibt es nicht mehr als die vom Limit vorgegebenen Anzahl an API-Ressourcen. 2. Die Summe der angefragten Rechenressourcen der einzelnen Workloads der zusammengehörigen Namespaces überschreitet nicht das eingestellte Limit.
Erfolgsszenario	1. ICC-Teammitglied erstellt ein Limit für zusammengehörige Namespaces. 2. Ein Request geht ein, eine API-Ressource in einem dieser Namespace zu erstellen, die weder die Limits für API-Ressourcen noch die Limits für die akkumulierten Rechenressourcen überschreitet. 3. Der Request wird genehmigt und die API-Ressource erstellt.
Erweiterungsfall	2a. Der Request, eine neue API-Ressource in diesem Namespace zu erstellen, würde das gemeinsame Limit der Namespaces für diese Art von API-Ressourcen oder Rechenressourcen übersteigen. 1. Der Request wird abgelehnt und die API-Ressource nicht erstellt.
Beispiel	Ein ICC-Teammitglied will, dass das drei Namespaces, die derselben Person gehören, zusammen nicht mehr als 50 GB beanspruchen. Nachdem ein entsprechendes Limit eingestellt ist, werden Anfragen in allen drei Namespaces abgelehnt, sollten sie das Limit überschreiten.

ID	UC-3
Titel	Limits für Pods eines Namespaces konfigurieren
Akteur*in	ICC-Teammitglied
Ziel	Erstellte Pods in diesem Namespace unterliegen den eingestellten Limits für Rechenressourcen pro Pod
Vorbedingung	1. ICC-Teammitglied ist in der Rolle „Admin“ bei der ICC angemeldet (autorisiert und authentifiziert).
Nachbedingung	1. Kein in diesem Namespace neu erstellter Pod bekommt mehr Rechenressourcen als durch das festgelegte Limit vorgegeben.
Erfolgsszenario	1. ICC-Teammitglied erstellt für die Pods eines Namespaces ein Limit pro Pod für eine oder mehrere Rechenressourcen. 2. Ein Request, einen Pod in diesem Namespace zu erstellen, der weniger oder gleich dem Limit für Rechenressourcen anfragt, wird erfüllt und der Pod im Cluster erstellt.
Erweiterungsfall	2a. Ein Request für einen neuen Pod in diesem Namespace fragt mehr Rechenressourcen an, als durch das Limit erlaubt. 1. Der Pod wird abgelehnt.
Beispiel	Ein ICC-Teammitglied will, dass das Limit für die angefragte Memory neuer Pods im Namespace `kiosk` unter 50 GB liegt. Nachdem ein Limit von 50 GB eingestellt ist, werden Pods mit einem Request über diesem Limit abgelehnt und nicht erstellt.

ID	UC-4
Titel	Priorität von Pod konfigurieren
Akteur*in	ICC-Teammitglied
Ziel	Pods werden ihrer Wichtigkeit geordnet geschedult
Nachbedingung	1. Wichtigere Pods werden früher als weniger wichtige Pods geschedult.
Erfolgsszenario	1. ICC-Teammitglied gibt in der Konfiguration eines Requests nach einem Pod eine Priorität an. 2. Der Pod wird nach seiner relativen Wichtigkeit geschedult.
Beispiele	1. Ein ICC-Mitglied möchte einen Job starten, der nicht kritisch ist, jedoch viele Ressourcen benötigt. Um die Ressourcen des Clusters in Hochzeiten nicht noch mehr zu belasten, gibt er/sie dem Job eine niedrige Priorität. 2. Ein systemkritischer Pod ist ausgefallen und wird von Kubernetes automatisch wieder geschedult. Da dieser eine höhere Wichtigkeit als alle anderen Requests in der Schedulingqueue hat, wird er an den Anfang der Queue gesetzt und damit als erstes geschedult.

ID	`UC-5`
Titel	Regeln zu erlauben Prioritäten konfigurieren
Akteur*in	ICC-Teammitglied
Ziel	Bestimmte Prioritäten dürfen nur von festgelegten Usern oder in festgelegten Namespaces vergeben werden.
Vorbedingung	1. ICC-Teammitglied ist in der Rolle „Admin“ bei der ICC angemeldet (autorisiert und authentifiziert). 2. Pods haben eine Eigenschaft, die ihre Wichtigkeit im Vergleich zu anderen Pods widerspiegelt (UC-3).
Erfolgsszenario	1. ICC-Teammitglied stellt ein, dass eine Priorität P nur für Pods eines speziellen Namespaces N verwendet werden kann. 2. Es wird ein Request für einen Pod mit der Priorität P gestellt. 3. Der Request will den Pod im vorher eingestellten Namespace N erstellen. 4. Der Request wird genehmigt und der Pod mit der eingestellten Priorität P erstellt.
Erweiterungsfall	3a. Der Request will den Pod in einem anderen Namespace $!N$ erstellen. 1. Der Request wird abgelehnt und der Pod nicht erstellt.
Beispiel	Ein ICC-Teammitglied will sicherstellen, dass nur im `kube-system` Namespace die höchste Priorität S für Pods verwendet werden kann. Dafür erstellt er/sie eine entsprechende Regel, wonach die Priorität S exklusiv für Pods im `kube-system` Namespace verwendet werden kann. Requests für Pods mit Priorität S in anderen Namespaces werden abgelehnt.

ID	UC-6
Titel	Ausnahmen für bestehende Regelungen definieren
Akteur*in	ICC-Teammitglied
Ziel	Um feinere Einstellungen für übergeordnete Regeln zu ermöglichen, können Ausnahmen definiert werden.
Vorbedingung	1. Es können übergeordnete Konfigurationen eingestellt werden (UC-1, UC-4).
Nachbedingung	1. Die speziellere Regel wird gegenüber der generellen Regel bevorzugt.
Erfolgsszenario	1. ICC-Teammitglied definiert eine Ausnahmeregel für die Limits der Pods innerhalb eines Namespaces. 2. Alle von der Ausnahmeregel betroffenen Requests werden mittels der neu definierten Limits überprüft. 3. Ist der Request nicht von der Ausnahmeregel betroffen, greifen eventuell allgemeinere Konfigurationen.
Erweiterungsfall	1a. ICC-Teammitglied definiert eine Ausnahmeregel zu erlaubten Prioritäten. 1. Bei Requests mit einer gesetzten Priorität wird diese Ausnahmeregel zuerst geprüft. 2. Greif diese Regel nicht, wird eine eventuell allgemeinere Regel angewendet.
Beispiel	Ein ICC-Teammitglied will, dass er/sie in einem Namespace N Pods mit einem Limit von 10 CPU erstellen kann. In dem Namespace N dürfen laut einer Regel nur Pods mit einem Limit von 5 CPU erstellt werden. Die Person erstellt eine Ausnahmeregel, die es allen Nutzenden mit der Rolle „Admin“ erlaubt, im Namespace N Pods mit einem Limit von 10 CPU zu erstellen. Allen anderen Rollen bleibt dies verboten.

ID	UC-7
Titel	Ressourcen reservieren
Akteur*in	ICC-Teammitglied
Ziel	Reservierte Ressourcen sind nur für bestimmte Namespaces verfügbar
Vorbedingung	1. ICC-Teammitglied ist in der Rolle „Admin“ bei der ICC angemeldet (autorisiert und authentifiziert).
Nachbedingung	1. Die reservierten Ressourcen sind für die definierten Namespaces verfügbar. 2. Es ist anderen Namespaces nicht möglich, die Ressourcen zu schedulen.
Erfolgsszenario	1. ICC-Teammitglied stellt für eine bestimmte Ressource R ein, dass diese nur für einen bestimmten Namespace N verfügbar ist. 2. Die Einstellung wird gespeichert. 3. Es wird eine Anfrage an die Ressource R gestellt. 4. Es wird geprüft, ob die Anfrage aus dem definierten Namespace N kommt. 5. Die Anfrage kommt aus Namespace N. 6. Die Anfrage wird freigegeben.
Erweiterungsfall	5a. Die Anfrage kommt aus einem Namespace, der nicht N ist. 1. Die Anfrage wird abgelehnt.
Beispiel	Zum Start des ML-Praktikums sollen für alle Studierenden GPUs verfügbar sein. Aus diesem Grund reserviert ein ICC-Teammitglied 60 GPUs für den JupyterHub Namespace der ICC. Diese GPU-Ressourcen sind damit nur noch für diesen verfügbar.

ID	UC-8
Titel	Metriken zur GPU Auslastung ansehen
Akteur*in	ICC-Teammitglied
Ziel	Durch die Metrik kann erkannt werden, wie ausgelastet die GPU-Ressourcen des Clusters sind
Vorbedingung	1. Das GPU Scheduling von Kubernetes wurde konfiguriert, um die GPU-Ressource im Cluster verfügbar zu machen. 2. Aussagekräftige Metrik zur Auslastung der GPU-Ressourcen sind festgelegt und können ausgelesen werden.
Erfolgsszenario	1. ICC-Teammitglied fragt die Metrik zur GPU Auslastung über eine bekannte Schnittstelle an. 2. Es wird eine Metrik zur Auslastung der GPU angezeigt.
Beispiel	Ein ICC-Teammitglied will kontrollieren, ob es Pods gibt, die GPU-Ressourcen blockieren, obwohl sie diese nicht mehr nutzen. Er/Sie ruft die Metrik auf und erkennt daraus, dass es mehrere GPU-Ressourcen gibt, die reserviert, aber nicht aktiv genutzt werden. Aus der Metrik kann auch abgelesen werden, welche Pods diese GPU-Ressourcen reserviert haben, wodurch er/sie diese gezielt abschalten und löschen kann.

ID	UC-9
Titel	Erkennung inaktiver Pods
Akteur*in	ICC-Teammitglied
Ziel	Pods erkennen, die seit gewisser Zeit nicht genutzt werden
Vorbedingung	1. Aussagekräftige Metriken zur Lasterkennung eines Pods sind festgelegt und können ausgelesen werden.
Erfolgsszenario	1. Die Pods eines bestimmten Namespaces werden überprüft. 2. Anhand der Metrik wird erkannt, dass Pods inaktiv sind. 3. Die Info über die inaktiven Pods wird über eine bekannte Schnittstelle zur Verfügung gestellt.
Beispiel	Ein ICC-Teammitglied will überprüfen, ob es in einem Namespace, der an sein Ressourcenlimit kommt, inaktive Pods gibt, um diese abzuschalten. Nach einer Anfrage an die Schnittstelle hat er/sie Informationen über inaktive Pods, die manuell abgeschaltet werden können.

ID	UC-10
Titel	Automatisches Löschen von Pods und Namespaces
Akteur*in	Automatisierter Prozess (z. B. CronJob)
Ziel	Pods, die seit gewisser Zeit nicht genutzt werden erkennen
Vorbedingung	1. Inaktive Pods können zuverlässig erkannt werden. (UC-8)
Erfolgsszenario	1. Ein regelmäßig automatisch ausgeführter Prozess überprüft die Pods eines Namespaces. 2. Es werden inaktive Pods erkannt. 3. Die entsprechenden Pods werden abgeschaltet und gelöscht.
Beispiel	Der automatisierte Job überprüft zu seiner festgelegten Zeit den Namespace N. Er erkennt, das ein Pod P inaktiv ist, woraufhin dieser über einen entsprechenden Aufruf an den API-Server gelöscht wird.

A.2 Implementierung

A.2.1 Kyverno

Hoch verfügbare Installation von Kyverno.

Implementierung A.1: Terminalbefehle

```
$ helm repo add kyverno https://kyverno.github.io/kyverno/
$ helm repo update
$ helm install kyverno kyverno/kyverno -n kyverno --create-namespace \
--set admissionController.replicas=3 \
--set backgroundController.replicas=2 \
--set cleanupController.replicas=2 \
--set reportsController.replicas=2
```

A.2.2 DCGM Exporter

`configMap` unter `additionalScrapeConfigs` definieren.

Implementierung A.2: Auschnitt der angepassten Helm-Chart

```
1  additionalScrapeConfigs:
2  - job_name: gpu-metrics
3    scrape_interval: 1s
4    metrics_path: /metrics
5    scheme: http
6    kubernetes_sd_configs:
7    - role: endpoints
8      namespaces:
9        names:
10       - gpu-operator
11   relabel_configs:
12   - source_labels: [__meta_kubernetes_pod_node_name]
13     action: replace
14     target_label: kubernetes_node
```

A.2.3 Prioritäten

Drei neue Prioritätsklassen.

Implementierung A.3: three-priorities.yaml

```yaml
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-icc-priority
value: 500
preemptionPolicy: Never
description: "This priority class is higher than the default of 0 and
     can be used for high priority pods."
---
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: default-icc-priority
value: 0
preemptionPolicy: Never
globalDefault: true
description: "This is the default priority class and is set for pods
    that do not explicitly specify a priority class."
---
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low-icc-priority
value: -20
preemptionPolicy: Never
description: "This priority class is lower than the default of 0 and
    can be used for low priority pods."
```

Systemkritische Prioritäten sind nur in bestimmten Namespaces erlaubt.

Implementierung A.4: deny-system-critical-priorities.yaml

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
...
    exclude:
      any:
        - resources:
            namespaces:
              - kube-system
              - kube-public
              - kyverno
              - cert-manager
              ...
    validate:
      message: >-
        The Pod PriorityClass {{ request.object.spec.template.spec.
    priorityClassName }} is not allowed in this Namespace.
      deny:
        conditions:
          any:
          - key: "{{ request.object.spec.priorityClassName || '' }}"
            operator: AnyIn
            value:
            - system-cluster-critical
            - system-node-critical
            - icc-critical
...
```

A.2.4 Monitoring

`CronJob`, der jeden Sonntag das angegebene Skript ausführt.

Implementierung A.5: cronjob-delete-pods.yaml

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: delete-inactive-pods
spec:
  schedule: "* * * * 0"
  jobTemplate:
    spec:
      template:
        spec:
          serviceAccountName: service-account-can-modify-pods
          containers:
          - name: container-with-script
            image: container-with-script-image:1.0
            command: ["/path/to/delete-inactive-pods-script.sh"]
          restartPolicy: OnFailure
```

A.2.5 HNC

Rechte zur Erstellung der Hierarchie durch den Background-Controller von Kyverno.

Implementierung A.6: hnc-multi-tenancy-clusterrole.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: hnc-multi-tenancy-clusterrole
  labels:
    app.kubernetes.io/component: background-controller
    app.kubernetes.io/instance: kyverno
    app.kubernetes.io/part-of: kyverno
rules:
  - apiGroups: ["hnc.x-k8s.io"]
    resources:
      - hierarchyconfigurations
      - hierarchicalresourcequotas
      - subnamespaceanchors
    verbs: ["*"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources:
      - roles
      - rolebindings
    verbs: ["*"]
```

Rechte der ICC-Nutzenden

Implementierung A.7: generate-role-and-binding-icc-user.yaml

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
...
spec:
  ...
  - name: create-role-icc-user
    match:
      ...
              - Namespace
    generate:
      ...
      kind: Role
      name: "hnc-user-default"
      namespace: "{{ request.object.metadata.name }}"
      data:
        rules:
          - apiGroups:
              - hnc.x-k8s.io
            resources:
              - subnamespaceanchors
            verbs: ["get", "list", "delete", "create", "update", "watch"]
              ...
  - name: create-rolebinding-icc-user
    match:
      ...
              - Namespace
    preconditions:
      any:
        - key: "{{ request.object.metadata.labels.\"created-for\" || '' }}"
          operator: Equals
          value: "icc-user"
    generate:
      ...
      kind: RoleBinding
      ...
      data:
        roleRef:
          apiGroup: rbac.authorization.k8s.io
          kind: Role
          name: hnc-user-default
        subjects:
          - kind: ServiceAccount
            name: default
            namespace: "{{ request.object.metadata.name }}"
```

`ClusterPolicy` zur Generierung der Hierarchie und Limitierung für die ICC-Nutzenden.

Implementierung A.8: generate-hierarchy-icc-user.yaml Hierarchie

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: generate-for-icc-user
  ...
spec:
  background: false
  rules:
  - name: create-hc-icc-user
    match:
      all:
        - resources:
            kinds:
              - Namespace
            operations:
            - CREATE
    preconditions:
      all:
        - key: "{{ request.object.metadata.labels.\"created-for\" || '' }}"
          operator: Equals
          value: "icc-user"
    generate:
      apiVersion: hnc.x-k8s.io/v1alpha2
      kind: HierarchyConfiguration
      name: "hierarchy"
      namespace: "{{ request.object.metadata.name }}"
      data:
        spec:
          parent: "icc-users"
          allowCascadingDeletion: true
  ...
```

`RoleBinding` durch welches `user1` seine/ihre Namespaces mit `user2` teilt.

Implementierung A.9: share-account-namespace.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user-2
  namespace: user-1
subjects:
- kind: User
  name: user-2
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: hnc-user-default
  apiGroup: rbac.authorization.k8s.io
```

Aktivieren der Vererbung von `LimitRanges` im HNC.

Implementierung A.10: Terminalbefehl

```
$ kubectl hns config set-resource limitranges --mode Propagate
```

Erstellen einer `LimitRange` für alle ICC-Nutzenden des HNC.

Implementierung A.11: icc-users-limit-range.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: icc-users-limit-range
spec:
  limits:
  - type: Container
    defaultRequest:
      cpu: "0.25"
      memory: 256Mi
    default:
      cpu: "0.5"
      memory: 512Mi
```

Generierung des `HRQ` aus einer Vorlage im Namespace `0-clone-source`

Implementierung A.12: generate-hrq-icc-user.yaml HRQ

```
...
  generateExisting: true
    ...
    preconditions:
      all:
        - key: "{{ request.object.metadata.labels.\"created-for\" || '' }}"
          operator: Equals
          value: "icc-user"
        # contitional exception
        # - key: "{{ request.object.metadata.name || 'NO EXCEPTION' }}"
        #   operator: NotEquals
        #   value: "excepted-username"
    generate:
      synchronize: true
      kind: HierarchicalResourceQuota
      ...
      clone:
        namespace: 0-clone-source
        name: icc-user-hrq
```

`HRQ` für die Account-Namespaces der ICC-Nutzenden.

Implementierung A.13: icc-user-hrq.yaml

```yaml
apiVersion: hnc.x-k8s.io/v1alpha2
kind: HierarchicalResourceQuota
metadata:
  name: icc-user-hrq
  namespace: 0-clone-source
spec:
  hard:
    pods: "10"
    services: "3"
    requests.cpu: "300m"
    requests.memory: 512Mi
    limits.cpu: "2"
    limits.memory: 2Gi
    persistentvolumeclaims: "2"
    requests.storage: 5Gi
```

Generierende Regeln für den Parallelbetrieb von Kiosk und dem HNC.

Implementierung A.14: generate-kiosk-to-hnc.yaml

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
...
spec:
  generateExisting: true
  rules:

  - name: g1-generate-account-ns # name is referenced in g2 rule below. If changed, change in g2 aswell
    match:
      ...
              - Account
    generate:
      apiVersion: v1
      kind: Namespace
      name: "{{ request.object.spec.subjects[0].name }}"

  - name: g2-generate-hc-for-account-ns
    match:
      ...
              - Namespace
    ...
    generate:
      apiVersion: hnc.x-k8s.io/v1alpha2
      kind: HierarchyConfiguration
      name: "hierarchy"
      namespace: "{{ request.object.metadata.name }}"
      data:
        spec:
          parent: "icc-users"
          allowCascadingDeletion: true

  - name: g3-generate-subns-for-kiosk-ns
    match:
      ...
              - Namespace
    context:
    - name: account_name
      variable:
        jmesPath: (request.object.metadata.labels."kiosk.sh/account" | to_string(@) | split(@, '-') | [0]) || 'NO KIOSK NS'
    ...
    generate:
      apiVersion: hnc.x-k8s.io/v1alpha2
      kind: SubnamespaceAnchor
      name: "{{ request.object.metadata.name }}"
      namespace: "{{ account_name }}" # parent
```

Mutierende Regeln für den Parallelbetrieb von Kiosk und dem HNC.

Implementierung A.15: mutate-kiosk-to-hnc.yaml

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
...
spec:
  mutateExistingOnPolicyUpdate: true
  rules:
  - name: m1-set-parent-label-for-kiosk-ns
    match:
      ...
          - Space
    mutate:
      targets:
        - apiVersion: v1
          kind: Namespace
          ...
      patchStrategicMerge:
        metadata:
          annotations:
            hnc.x-k8s.io/subnamespace-of: "{{ target.metadata.labels
    .\"kiosk.sh/account\" | to_string(@) | split(@, '-') | [0] }}"

  - name: m2-set-parent-of-kiosk-hc
    match:
      ...
            - HierarchyConfiguration
    mutate:
      targets:
        - apiVersion: hnc.x-k8s.io/v1alpha2
          kind: HierarchyConfiguration
          name: "hierarchy"
          namespace: "{{ request.object.metadata.namespace }}"
      patchStrategicMerge:
        spec:
          allowCascadingDeletion: true
          parent: "icc-users"
```

Validierende Regel für den Parallelbetrieb von Kiosk und dem HNC.

Implementierung A.16: validate-delete-kiosk-space.yaml

```
...
spec:
  validationFailureAction: Enforce
  rules:
  - name: deletion-of-space-not-allowed
    match:
      ...
          - Space
          operations:
          - DELETE
    validate:
      message: "The ICC is currently in a transitional phase to
    improve the multi-tenancy experience. Space deletion is
    temporarily disabled."
      ...
```

`ClusterRole` für den Cleanup-Controller von Kyverno und eine `CleanupPolicy` zum Löschen der übriggebliebenen Kiosk RBAC-Objekte

Implementierung A.17: delete-role-and-binding-clusterrole.yaml, cleanup-kiosk-role-and-binding.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
  ...
  labels:
    app.kubernetes.io/component: cleanup-controller
    ...
rules:
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings", "clusterroles", "
    clusterrolebindings"]
    verbs: ["delete", "list"]
---
apiVersion: kyverno.io/v2beta1
kind: ClusterCleanupPolicy
...
spec:
  match:
    ...
          - RoleBinding
    ...
    - key: "{{ target.roleRef.name }}"
      operator: Equals
      value: "kiosk-space-admin"
  schedule: "*/1 * * * *"
  ...
          - ClusterRoleBinding
          - ClusterRole
    ...
    - key: "{{ target.metadata.name }}"
      operator: Equals
      value: "kiosk-account-*"
    - key: "{{ target.metadata.name }}"
      operator: Equals
      value: "kiosk-creator"
  schedule: "*/1 * * * *"
```

A.2.6 Jupyterhub

Validierungsregel, dass nur die ausgenommenen Namespaces GPUs anfragen dürfen.

Implementierung A.18: allow-gpu-only-jupyterhub.yaml

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: allow-gpu-only-jupyterhub
  ...
spec:
  background: true
  validationFailureAction: Enforce
  rules:
  - name: check-gpu-request
    match:
      ...
          - Pod
    exclude:
      ...
            namespaces:
              - kube-system
              - kube-public
              - default
              - kyverno
              - jupyterhub-system
              - jupyterhub-praktikum
              - jupyterhub-default
              # - other-exempted-namespace
    validate:
      message: Containers may not define GPU Ressources.
      ...
          containers:
          - (name): "*"
            =(resources): # if tag is specified, continue processing
              =(limits):
                X(nvidia.com/gpu): null # tag cannot be specified
```

Priorität für JupyterHub-Pods außerhalb eines Praktikums.

Implementierung A.19: jupyterhub-lower-priority.yaml

```yaml
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: jupyterhub-lower-priority
value: -1
preemptionPolicy: Never
description: "Default priority for JupyterHub pods requested outside
    of study courses."
```

Priorität für JupyterHub-Pods mit der Erlaubnis niedriger priorisierte Pods zu beenden.

Implementierung A.20: jupyterhub-praktikum-preemption.yaml

```yaml
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: jupyterhub-praktikum-preemption
value: -1
preemptionPolicy: PreemptLowerPriority
description: "Priority for JupyterHub pods that are started as part
    of a study course. Can preempt lower priority Pods."
```

Prioritäten für Platzhalterworkloads.

Implementierung A.21: jupyterhub-placeholder-priority.yaml

```yaml
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: placeholder
value: -50
preemptionPolicy: Never
globalDefault: false
description: "Priority for placeholder workloads"
```

Platzhalter-Deployment, das GPU-Ressourcen reserviert [13].

Implementierung A.22: jupyterhub-placeholder-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reserve-gpus
spec:
  replicas: 30
  selector:
    matchLabels:
      app: gpu-placeholder
  template:
    metadata:
      labels:
        app: gpu-placeholder
    spec:
      priorityClassName: placeholder
      terminationGracePeriodSeconds: 0
      containers:
      - name: ubuntu
        image: ubuntu
        command: ["sleep"] # Tell the Pods to remain active until evicted
        args: ["infinity"]
        resources:
          limits:
            cpu: "500m"
            memory: "128Mi"
            nvidia.com/gpu: "1"
```

A.2.7 Limits durch Kyverno Policy

Limits durch Kyverno Policy mit Ausnahme spezieller Pods.

Implementierung A.23: exclude-pod-from-limit.yaml

```
...
    exclude:
      any:
      - resources:
          names:
          - "prod-*"
    ...
    validate:
      ...
      pattern:
        spec:
            ...
                =(cpu): <={{maxCpuUsage}}
                =(memory): <={{maxMemoryUsage}}
              =(limits):
                =(cpu): <={{maxCpuUsage}}
                =(memory): <={{maxMemoryUsage}}
```

A.3 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
minikube	Tool zum Aufsetzen eines lokales Kubernetes Clusters verwendet zum Testen der Konfigurationen und externer Kubernetes-Tools
draw.io	Anwendung für das Erstellen von Diagrammen verwendet zur Erstellung ebensolcher. Speziell zum Einsatz kam die VS Code Integration von Henning Dieterichs
	Docker Image polinux/stress verwendet, um CPU und Memory Auslastung zu simulieren. Zur Verfügung gestellt von Przemyslaw Ozgo.

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original