Betrieb von mehreren JupyterHubs mit nbgrader auf einem Kubernetes Cluster

Operating multiple JupyterHub instances on a Kubernetes cluster

Bachelorarbeit

Jonas Matthias Kloos
Fachhochschule Südwestfalen
27. Juli 2023

Autor: Jonas Matthias Kloos
Referent: Prof. Dr. Giefers
Korreferent: Prof. Dr. Gawron
Eingereicht: 27. Juli 2023

Zusammenfassung

Diese Ausarbeitung basiert auf einem praktischen Projekt, welches in den kommenden Kapiteln beschrieben wird. Das Ziel des Projektes besteht darin, auf einem *Kubernetes Cluster* mehrere *JupyterHub* Umgebungen für verschiedene Lehrveranstaltungen zu betreiben. Diese Ziele wurden formuliert, da es mit den bisher vorhandenen Lösungen zu einem erheblichen Mehraufwand kommt, wenn auf einer *JupyterHub* Umgebung mehrere Kurse parallel betrieben werden sollen. Für die Erfüllung dieser Ziele werden neben *Kubernetes* eine Art "Cloud-Betriebssystem" und *JupyterHub* noch einige andere Technologien und Softwarelösungen eingesetzt. Des Weiteren sollte eine Möglichkeit entwickelt werden, die Erstellung und die Anwendung, diese Umgebungen so weit wie möglich zu automatisieren und den Verwaltungsaufwand weitestgehend zu minimieren. In den kommenden Kapiteln werden die verschiedenen zum Einsatz kommenden Technologien beschrieben und aufgezeigt, wie diese zusammen arbeiten, um die im Vorfeld definierten Ziele zu erreichen. Diese Ausarbeitung dient als eine technische Dokumentation der erstellten Lösung und als eine Diskussion über diese.

Inhaltsverzeichnis

1 Einleitung

1.1 Zielsetzung

Das Ziel dieser Arbeit ist es, eine Methode zu entwickelt, welche es erlaubt, die Erstellung von *JupyterHubs* für Kurse von Studierenden zu automatisieren. Es ist geplant, diese *JupyterHubs* in Verbindung mit *nbgrader* als Praktikumsumgebungen zu nutzten.
Diese Automatisierung soll dazu führen, dass der Arbeitsaufwand der Lehrende bei Erstellung und Verwaltung von Praktikumsumgebung im Vergleich zu gängigen Lösungen deutlich reduziert wird.
Die erstellte Lösung wird auf einem *Kubernetes* basiertem *Cluster* betrieben. Gleichzeitig wird die Nutzung so vereinfacht, dass es auch Personen ohne Vorkenntnisse im Umgang mit *Kubernetes* und den dazugehörigen Technologien möglich ist, eine Praktikumsumgebung aufzusetzen und zu verwalten.
Der Login in die einzelnen Praktikumsumgebungen erfolgt über die bereits vorhandene Lehrplattform Moodle.

1.2 Aufbau der Arbeit

Die Arbeit besteht aus mehreren Kapiteln, zu Beginn werden die technischen Grundlagen erklärt, die zum Verständnis der erstellten Lösung und der beschriebenen Vorgehensweise zum Erreichen dieser notwendig sind. Die Grundlagen basieren auf der zu Beginn der Arbeit recherchierten Literatur, welche aufgrund der Zielsetzung analysiert wurde.
Es folgt eine detaillierte Beschreibung der technischen Ausarbeitung, welche auf die konkreten Ziele, die Vorgehensweise, die getroffenen Entscheidungen und den Aufbau der erstellten Lösung eingeht.
Anschließend wird die erstellte Lösung mit anderen Lösungsansätzen zum Erreichen gleicher oder sehr ähnlicher Ziele verglichen.
Im nächsten Kapitel wird die erstellte Lösung evaluieren und vergleichen, welche zuvor gesetzten Ziele eingehalten werden konnten, welche nicht komplett erfüllt wurden und welche ganz liegen gelassen werden mussten.
Anschließend wird ein Ausblick auf mögliche Erweiterungen der Lösung gegeben. Es wird diskutiert was in Zukunft verändert werden könnte, um die Ergebnisse der Arbeit zu verbessern, wie etwa die Erweiterung der Anwendungsgebiete.

2 Technische Grundlagen

2.1 Kubernetes

Kubernetes ist ein *Open-Source-Tool* zur *Container-Orchestrierung* und wurde im Juni 2014 von Google veröffentlicht. Anfangs wurde es unter dem Namen Borg entwickelt.[1] Der größte Anwendungsbereich von *Kubernetes* liegt darin, verteilte Systeme für *cloud-nativ* Anwendungen bereitzustellen. Hierbei ist es egal, ob auf einem *Cluster* bestehend aus *Raspberry Pis* oder in einem modernen Rechenzentrum oder aber in eine Hybrid-Umgebung gearbeitet werden soll.[2] Ein *Kubernetes-Cluster* besteht aus einem oder mehreren Rechnern, welche *Nodes* genannt werden. Je nach Größe des Clusters übernimmt eine *Node* die Rolle des *Masters* und die Übrigen treten als *Worker* bei. Bei größeren *Clustern* ist es aber auch durchaus üblich, dass mehrere *Nodes* die Rolle einer *Master-Node* übernehmen, dies sorgt für zusätzliche Ausfallsicherheit.
Eine weitere Stärke von *Kubernetes* liegt in der einfachen Skalierbarkeit, sowohl was das *Cluster* als auch die darauf laufenden Anwendungen angeht. Diese Skalierbarkeit in Verbindung mit einer hohen Ausfallsicherheit hat *Kubernetes* in den letzten Jahren zu einer großen Beliebtheit bei vielen Entwicklern und Anbietern von Hostingdiensten verholfen.[3] Die Skalierbarkeit kommt vor allem dadurch zustande, dass *deployte* Anwendungen automatisch nach Bedarf erweitert werden können. Ein weiterer Aspekt, in dem Kubernetes eine Skalierung vereinfacht, ist der, dass es simpel ist, einem laufenden *Cluster* neue *Nodes* hinzuzufügen.
Wenn eine *Node* ausfallen sollte, zum Beispiel bei einer Wartung, kann *Kubernetes* alle darauf laufenden *Pods* automatisch auf andere *Nodes* im *Cluster* verschieben. Dies führt dazu, dass eine möglichst geringe *Downtime* erreicht werden kann.
Ein *Pod* stellt dabei die kleinste Organisationseinheit in *Kubernetes* dar. Dabei handelt es sich um einen *Container* oder eine Gruppe von *Containern*, die immer zusammen *deployt* wird und eine eigene *Runtimeumgebung* hat. Alle *Container* in einem *Pod* werden auf den gleichen *Host deployt*, teilen sich ein Dateisystem und ein *Netzwerkstack*. Dieser Zusammenschluss erlaubt es den *Containern* innerhalb eines *Pods* über das Dateisystem oder über *localhost Networking* miteinander zu kommunizieren.[4] Das Verwenden von *Pods* bringt außerdem den Vorteil mit, dass die einzelnen Anwendungen deutlich stärker voneinander abgeschirmt sind, als wenn sie direkt auf einem *Host* laufen würden.

[1] vgl. Bai17, S. 14.
[2] vgl. BBH21, 1f.
[3] vgl. Wat22.
[4] vgl. IH23, S. 6.

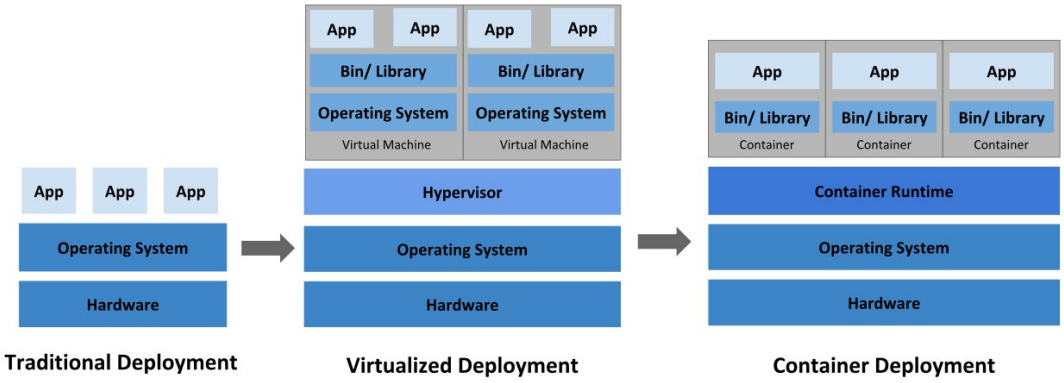

Abbildung 2.1: Verschiedene Deployment-Strategien, Quelle: [Aut23]

Die Abbildung 2.1 zeigt die drei wesentlichen Strategien, Dienste zu *hosten*.
Links ist die "traditionelle" Methode abgebildet, bei dieser führt ein Rechner einen oder mehrere Dienste direkt aus, ohne dass zusätzliche Steuerungssoftware eingesetzt wird.
In der Mitte ist die Methode der "virtuellen Maschinen" abgebildet, hier werden auf einem Rechner mehrere *virtuelle Maschinen* (VMs) ausgeführt, diese führen dann die gewünschten Dienste aus. Diese Methode bietet gegenüber dem direkten Ausführen auf einem physischen Server den Vorteil, dass die Dienste von denen auf anderen VMs abgetrennt sind. Wenn ein Dienst aufgrund von Problemen, einem Update oder aus sonstigen Gründen den Neustart des Servers erfordern würde, so kann in dieser Methode nur die betroffene VM neu gestartet werden und die anderen Dienste sind somit unbetroffen. Dieses Vorgehen bringt jedoch einen deutlich erhöhten Rechenaufwand mit sich, da für jede VM ein komplettes Betriebssystem ausgeführte werden muss. Außerdem werden Prozesse benötigt, welche die Kommunikation zwischen VM und Host ermöglichen.
Rechts im Bild ist die *Deploymentstrategie*, welche unter anderem von *Kubernetes* genutzt wird. Diese Strategie bringt, wie schon erwähnt, eine noch stärkere Abschirmung der einzelnen Anwendungen mit sich. Was zu einer erhöhten Sicherheit führt, da Anwendungen nur Zugriff auf bestimmte Bereiche des Dateisystems haben. Zudem ist die Ausfallsicherheit im Vergleich zur Strategie mit VMs noch einmal deutlich erhöht, da, wenn eine Anwendung ausfällt, nur der Betroffenen *Pod* neu gestartet werden muss. Der Rechenaufwand ist auch hier etwas erhöht, da eine zusätzliche Schicht erforderlich ist, jedoch ist er nicht ganz so hoch wie bei der Benutzung von VMs, da nicht für jeden Pod ein eigenes Betriebssystem simuliert werden muss.
Der letzte große Vorteil von *Kubernetes* ist der, dass die Infrastruktur stark abstrahiert wird, was dazu führt, dass die Entwickler von Anwendungen von bestimmten Rechnern getrennt werden, und auch der Wechsel zwischen verschiedenen Umgebungen wird deutlich vereinfacht, da die Entwickler mit einer API auf einem höheren Level interagieren und somit nur die deklarativen Konfigurationen auf ein neues Cluster übertragen werden müssen.[5]

[5] vgl. BBH21, S. 10.

2.1.1 Kubernetes Networking

Kubernetes bildet ein Netzwerkmodell ab, siehe 2.2, welches die Grundlage für die Kommunikation zwischen den *Containern, Pods* und *Services* innerhalb von *Kubernetes* schafft.[6]
Das Kubernetes-Netzwerkmodell gibt einige grundlegenden Bedingungen vor:[7]

- Jeder *Pod* hat eine einzigartige IP Adresse
- *Container* innerhalb eines *Pods* teilen sich die IP Adresse des *Pods* und können frei untereinander kommunizieren
- *Pods* können mit allen anderen *Pods* innerhalb des *Clusters* kommunizieren
- Isolation wird über *Netzwerkpolicies* geregelt

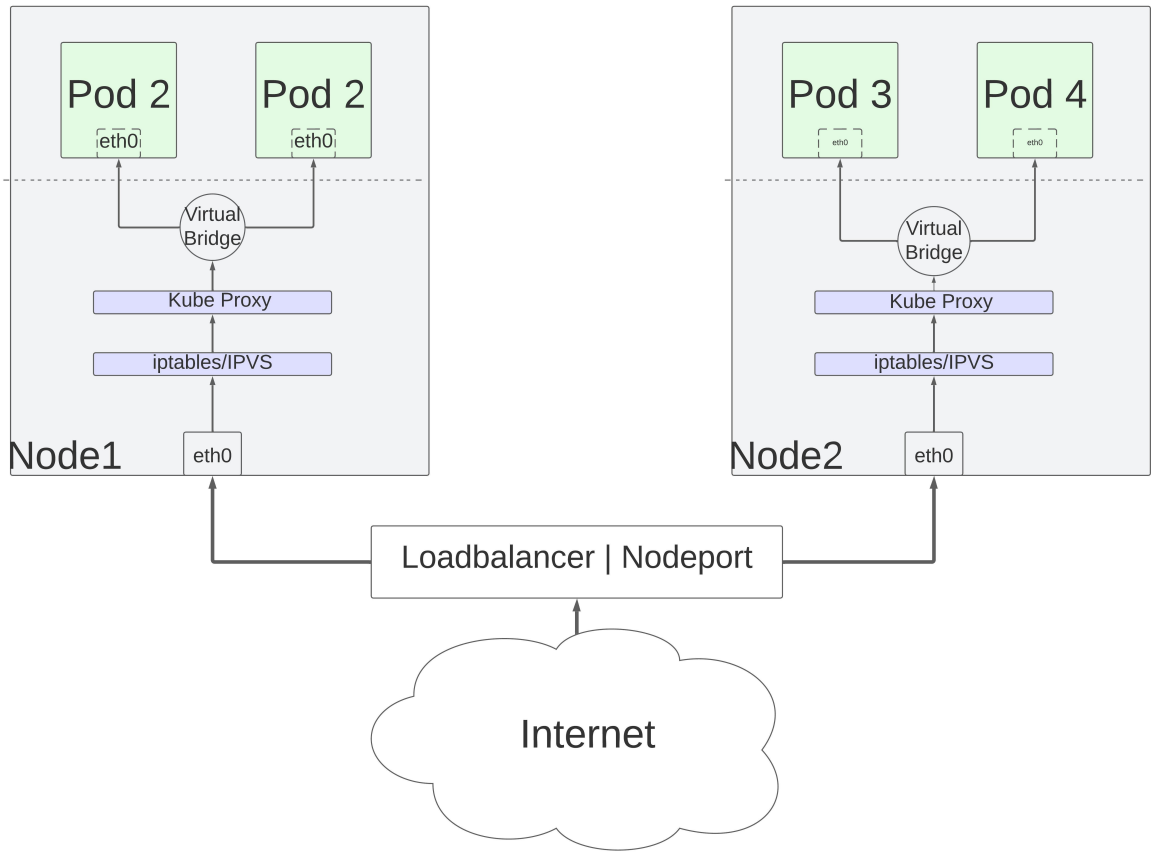

Abbildung 2.2: Netzwerkmodell und Services in Kubernetes, Quelle: [Vel22]

[6] vgl. Pol, S. 11.
[7] Pol, S. 11.

Services

Services sind in *Kubernetes* ein Weg, um eine Anwendung, welche in einer Gruppe von *Pods* läuft, abstrahiert freizugeben.[8]
Dabei gibt es in *Kubernetes* drei verschiedene Arten von *Services*. Die erste ist die *Cluster-IP*. Dieser *Service* schreibt der Anwendung eine eindeutige virtuelle IP zu und sorgt somit dafür, dass sie aus dem *Cluster* heraus erreicht werden kann.
Des Weiteren gibt es den *Nodeport*, dieser reserviert auf jeder *Node* im *Cluster* einen *Port* für den erstellten *Service*, sodass die *Pods* dann über die IP-Adresse der *Node* in Verbindung mit dem freigegebenem *Port* erreichbar sind.
Die letzte Art von *Service* ist der *Loadbalancer*, auch dieser vergibt eine virtuelle IP an die Anwendung, im Gegensatz zur *Cluster-IP* ist diese jedoch auch von außerhalb des *Clusters* zugänglich.

Netzwerk Plugins

Das von *Kubernetes* implementierte *Netzwerkmodell* stellt jedoch nur die grundlegenden Funktionen zur Verfügung, sodass es üblich ist, zusätzlich ein *Netzwerk-Plugin* zu installieren. Diese *Plugins* funktionieren über das *Container Network Interface* (CNI). Es gib verschiedene *CNI Plugins*, diese lassen sich in zwei Gruppen unterteilen. Die *Network Plugins*, welche dafür sorgen, dass die *Pods* an ein Netzwerk angeschlossen werden und die *IP Address Management Plugins* (IPAM), welche den *Pods* IP-Adressen zuweisen.[9]

2.1.2 Helm

Helm (englisch: das Ruder, das Steuer) ist ein *Paketmanager* für *Kubernetes*, welcher es ermöglicht, Software, welche für *Kubernetes* entwickelt wurde, bereitzustellen, zu verteilen und zu nutzen.[10]
Im Gegensatz zu anderen *Paketmanagern*, welche z.B. *Executables* bereitstellen, arbeitet *Helm* mit sogenannten *Charts*.
Ein *Helm-Chart* kann man sich so vorstellen, dass mehrere *Kubernetes-Komponenten* in ein Paket zusammengefasst werden. Ein *Chart* beinhaltet also alle deklarativen Komponenten, mit deren Hilfe eine *Kubernetes Ressource* konfiguriert und erstellt werden kann.[11]
Helm arbeitet mit *Repositorys*, um die erstellten *Charts* den *Endusern* zur Verfügung zustellen. Ein *Helm-Chart* wird dabei in einer *YAML-Datei* erzeugt und definiert. Wenn ein *Helm-Chart* erstellt wurde, kann es mit *Helm* installiert werden. *Helm* bietet aber auch Möglichkeiten zu Versionsverwaltung, so können laufende Anwendungen auf eine neue Version upgedatet werden oder auf eine ältere gedowngradet werden.

[8] vgl. Min+21, S. 51; vgl. Pol, S. 12.
[9] vgl. Pol, 11f.
[10] vgl. Nat21.
[11] vgl. BD20, S. 23.

2.1.3 Persistant Volume

Ein *Persistant Volume* (PV) ist eine Speichereinheit innerhalb des *Clusters* und wird ähnlich zu *Nodes* durch einen eigenen *Ressourcentyp* abgebildet. PVs können unabhängig von *Pods* existieren.[12] Dies führt dazu, dass PVs auch dann weiter bestehen, wenn der *Pod*, welcher sie verwendet hat, nicht mehr laufen sollte. Sie sind, wie der Name vermuten lässt, persistent (persistant, engl.: beständig, bleibend). Die Beständigkeit ist für dieses Projekt wichtig, da ein *Pod*, welcher neu gestartet werden musste, wieder mit seinem PV verbunden werden kann und somit keine Dateien verloren gehen. Dies ist besonders von Vorteil, da die einzelnen Userumgebungen automatisch abgeschaltet werden, wenn diese längere Zeit nicht benutzt wurden. PVs können entweder vom *Cluster Administrator* manuell erzeugt werden, dann spricht man von einer *statischen Bereitstellung*, oder sie werden automatisch mit der Hilfen von *Storageklassen* erstellt, dann spricht man von einer *dynamischen Erzeugung*.

2.1.4 Persistant Volume Claim

Ein *Persistant Volume Claim* (PVC) ist eine Anfrage für bestimmte Speichereinheiten. PVCs ähneln *Pods* insofern, als dass sie *Cluster Ressourcen* konsumieren. Ein *Pod* konsumiert *Node Ressourcen* wie etwa CPU oder Arbeitsspeicher, ein PVC konsumiert *PV Ressourcen*.[13] Ein PVC fragt immer bestimmte Speichergrößen und Zugriffsmethoden an und kann nicht ohne ein korrekt konfigurierten PV eingesetzt werden.

In diesem Projekt werden für den *Hub* dynamisch erzeugte PVs bzw. PVCs genutzt, hierfür wurde eine *Storage Class* erstellt, welche lokalen Speicher auf den *Nodes* zur Verfügung stellt. Die dynamische Erzeugung vereinfacht die Anwendung insofern, als dass zum Erstellen einer neuen Umgebung nicht zuerst neue PVs erzeugt werden müssen. Diese Vereinfachung bringt im Wesentlichen zwei Vorteile mit sich. Erstens bietet es eine zeitliche Ersparnis und eliminiert eine potenzielle Fehlerquelle bei der Portierung auf ein anderes *Cluster*. Der zweite Vorteil liegt darin, dass somit weniger Kenntnisse über Kubernetes und das hier verwendete System notwendig sind.

Die Abbildung 2.3 verdeutlicht die beiden verschiedenen Möglichkeiten, in einem *Kubernetes Cluster* bleibenden Speicher für die Pods bereitzustellen. In dieser Abbildung wird als *Speicherressource* ein lokales, auf den *Nodes* vorliegendes Volumen genutzt. Hierfür gibt es aber auch Alternativen, so bieten einige der großen Cloudanbieter die Möglichkeit, Speicher für die eigenen *Pods* auf ihren Plattformen anzumieten.
Die obere Hälfte der Abbildung zeigt die statische Methode, bei dieser werden PVs im Voraus erstellt und wenn ein *Pod* mit einem entsprechendem PVC gestartet wird, so wird ein passendes PV an diesen PVC gebunden. In der unteren Bildhälfte ist die dynamische Methode abgebildet. Bei dieser kommt eine zusätzliche *Storageclass* zum Einsatz. Diese ist so konfiguriert, dass das *Cluster* bzw. der *API-Server* diese nutzen kann, um ein neues PV nach der vorgegebenen Definition zu kreieren. Wenn also jetzt ein neuer *Pod* mit einem PVC gestartet wird, schaut der *API-Server* erst, ob ein passendes PV zur Verfügung steht, ist das nicht der Fall, wird anhand der *Storageclass* ein neues PV erstellt und an den PVC gebunden.

[12] vgl. Aut23.
[13] vgl. Aut23.

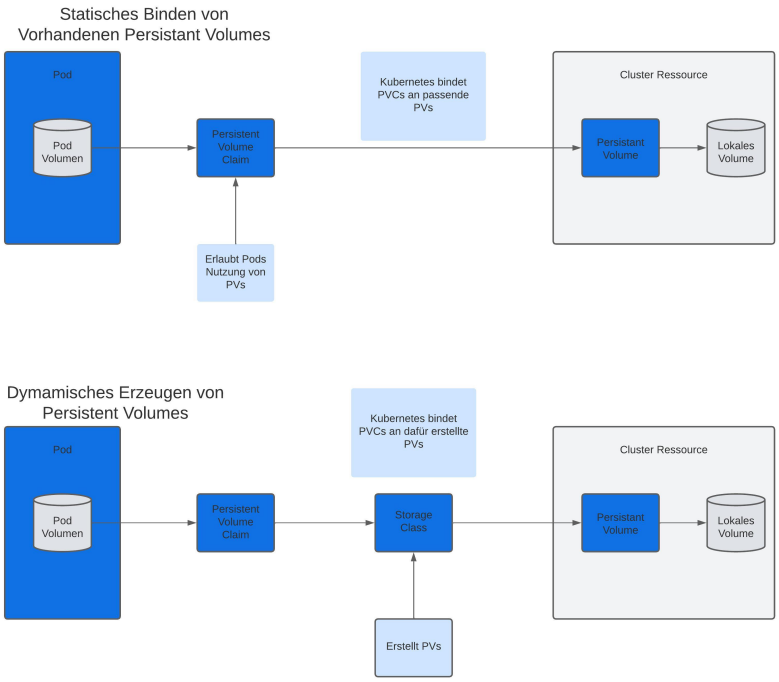

Abbildung 2.3: Speicher-Bindung in Kubernetes, Quelle: [Ran23]

2.2 JupyterHub

JupyterHub ist eine Umgebung, welche auf kooperatives Arbeiten ausgelegt ist. Es kann für verschiedene Zwecke eingesetzt werden, beispielsweise für die Datenverarbeitung innerhalb einer Forschungsgruppe oder aber für eine Gruppe von Studierenden.[14] Was jedoch alle Anwendungsfälle von *JupyterHub* gemeinsam haben, ist, dass der grundlegende Aufbau immer der gleiche ist. Ein grundlegendes *Deployment* von JupyterHub ist, dass es immer die gleichen Komponenten gibt 2.4. Wie bereits namensgebende ist die erste Komponente der *Hub*, dazu kommen dann ein *Proxy*, mehrere *Jupyter Notebook Server* für die einzelnen Nutzer und eine *Authentificator-Klasse*.
Der *Hub* ist der Kern des *Deployments*, der kommuniziert mit allen anderen Komponenten, er ist dafür zuständig, den *Proxy* und die *Notebooks* nach Bedarf für die Nutzer zu starten. Der Start geschieht mithilfe der sogenannten *Spawner*. Der *Hub* sorgt sich um die Loginfunktionen und verwaltet die Datenbanken. Alle Einstellungen werden in einer *Config-Datei* getroffenen, welche klassischerweise Config.py heißt.

Der *Proxy* ist dafür zuständig, alle Anfragen, die ein Nutzer über seinen *Browser* an den *Hub* oder die entsprechenden *Notebooks* schickt, weiterzuleiten. Er bildet somit die Schnittstelle zwischen *Hub* und *User*.
Die *Notebooks* bilden die Userumgebung. Sie sind der Bereich, in dem gearbeitet werden kann. Sie werden als eine *Webapp* ausgeliefert und bieten die Möglichkeit, Dateien zu bearbeiten und auch Programmcode zu erstellen und auszuführen. *Notebooks* können gezielt Ressourcen zugeschrieben werden. So können Anwender beispielsweise für aufwendige Berechnungen auf die starke Hardware eines Servers zurückgreifen, ohne selbst einen leistungsstarken Rechner verwenden zu müssen.
Die *Authentificatorkalsse* ist ein Mechanismus, um *User* im *Hub* oder auf *Notebook-Servern* zu *authentifizieren* und somit Zugriff zu gewähren.

[14] Con23.

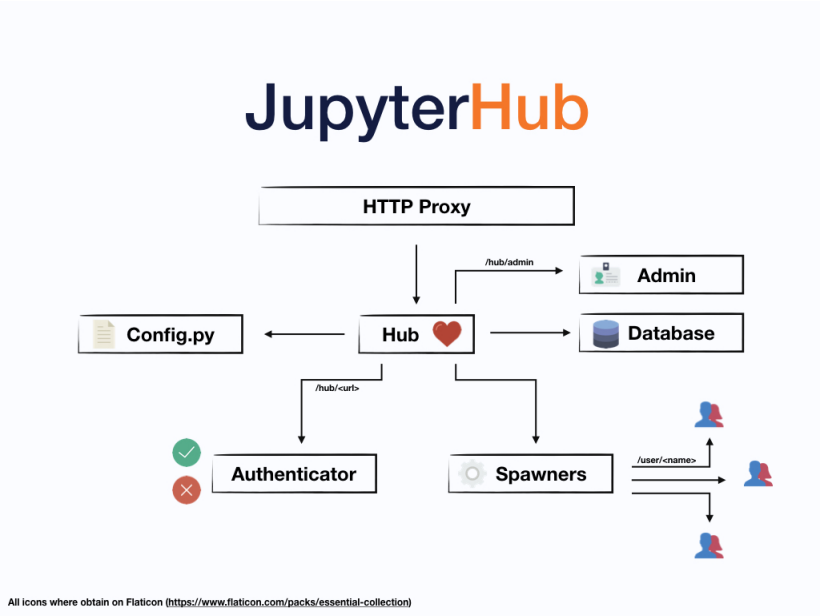

Abbildung 2.4: Grundlegender Aufbau einer JupyterHub Umgebung, Quelle:[Con23]

2.2.1 NBGrader

Nbgrader ist ein *Tool* zum Erstellen, Austeilen und Bewerten von *Assignments*, welches in *JupyterHub* integriert werden kann. *Nbgrader* ermöglicht es Lehrenden *Notebook* basierte *Assignments* zu erstellen, welche sowohl den Programmtext und dazugehörige Tests als auch textbasierte Freiantwortanteile beinhalten können. Darüber hinaus bietet *nbgrader Workflows* zum Bewerten und Benoten dieser *Assignments*. So können die erstellten Tests automatisch ausgeführt und bewertet werden, was die Notwendigkeit, alle Abgaben einzeln auszuführen, hinfällig macht. In Folge dessen müssen die Lehrenden nur noch die Freitextanteile bewerten. Dabei erstellt der/die Lehrende eine Version des *Assignments*, welches die Lösungen enthält, und aus dieser wird automatisch eine Version für die Studierenden erzeugt, sodass die Notwendigkeit zwei verschiedenen Versionen zu verwalten, wegfällt.[15]
Den Studierenden gibt es die Möglichkeit, neue *Assignments* automatisch zu erhalten und abgeschlossene *Assignments* einzureichen.

2.2.2 LTIauthentificator

LTI Authentificator ist eine *Authentificator-Klasse* für *JupyterHub*, welche den Login über verschiedenen Lehrplattformen wie etwa *Moodle* ermöglicht.
Dabei wird *JupyterHub* als ein *Tool-Provider* konfiguriert, diese arbeitet dann mit einem sogenannten *Tool-Consumer* zusammen. Dieser *Tool-Consumer* kann eine von verschiedenen Lehrplattformen sein, in diesem Projekt wird *Moodle* genutzt. Ein direkter Login über den *Hub* ist dann nicht mehr möglich, sodass der *Tool-Consumer* entsprechend konfiguriert werden muss. Hierzu wird in *Moodle* ein externes Tool angelegt, in welchem ein Link zum *LTI-Tool* auf dem *Hub* hinterlegt wird.

[15] vgl. Bla+18, 1f.

3 Technische Ausarbeitung

3.1 Zielsetzung

Das Ziel des Projektes besteht darin, auf einem *Kubernetes Cluster* mehrere *JupyterHub* Instanzen laufen zu lassen und eine Möglichkeit zu entwickeln, wie großen Aufwand neue Instanzen zu erstellt werden können, wenn diese benötigt werden.
Die *Hubs* sollen über *nbgrader* verfügen und der Login soll für die Studierenden über *Moodle* möglich sein. Dieses Vorgehen soll den Aufwand, der beim Betrieb von mehreren Kursen auf einem *Hub* entsteht, möglichst weit reduzieren. Daher ist ein weiteres wichtiges Kriterium, dass der Verwaltungsaufwand möglichst gering ausfällt. Die letzte wichtige Anforderung war es, die Erstellung der *Hubs* möglichst weit zu automatisieren.

Must-haves:

- Ein *JupyterHub* mit *nbgrader* als *Kubernetes Deployment*
- Mehrere *Deployments* für mehrere Kurse auf demselben *Cluster*
- Authentifizieren über *Moodle*
- Der *Hub* für einen Kurs soll möglichst leicht konfigurierbar sein
- Integration der Lehrmaterialien z.B. über *git* und *nbgitpuller*

Nice-to-haves:

- Webanwendung zum Erstellen von *Hubs*

3.2 Vorgehen

Vor Beginn der Arbeit wurden die Ziele des Projektes betrachtet und überlegt, welche Arbeitsabschnitte zum Erreichen dieser notwendig sind. Nach der Definition der Arbeitsschritte wurden sie in Arbeitspakete eingeteilt und mit deren Hilfe die zur Verfügung stehende Zeit geplant. Dieses Vorgehen wurde gewählt, um einen klar definierten Ablauf zu haben und somit einen besseren Vergleich zwischen den bereits erledigten Zielen und der Zeitplanung bieten. Folgende Planung wurde vorgenommen:

Woche	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Einarbeitung und erster Prototyp	x	x	x						
Integration der Gewünschten Funktionen		x	x	x					
Automatisierung des Deployments			x	x	x				
Entwicklung einer Webanwendung					x	x	x	x	

Die Tabelle zeigt, in welche Arbeitsabschnitte die Ziele unterteilt wurden und wie viel Bearbeitungszeit den einzelnen Paketen zugeteilt wurde. Die angegebenen Wochen sind nur Richtwerte und keinen festen Deadlines. Sie dienen lediglich der Kontrolle. Dieses strukturierte Vorgehen half dabei, einen besseren Überblick darüber zu behalten, welche Teile der Lösung bereits vorhanden waren und welche noch implementiert werden mussten. Aber auch welche Ziele aufgrund der zur Verfügung stehenden Zeit nicht mehr erreicht werden konnten.

3.2.1 Einarbeitung in das Thema

Die erste Phase der Bearbeitung stellte die Einarbeitung in das gegebene Thema dar. Hierzu wurden unter anderem die Dokumentationen der verschiedenen eingesetzten Softwares zurate gezogen. Dies half dabei, sich einen möglichst umfassenden Überblick zu verschaffen und eine optimale Strategie für die Implementierung finden zu können.
In dieser Zeit wurden ebenfalls kleinere Versuche gemacht, um sich mit der Materie vertraut zu machen und es wurden erste Prototypen entwickelt, die zu diesem Zeitpunkt noch sehr minimal ausfielen. Es wurde sich auch mit den verschiedenen Möglichkeiten des Auslieferns der erstellten Lösung befasst und schließlich der Entschluss gefasst, diese über ein *Helm-Chart* zu realisieren.

3.2.2 Implementierung

In der Phase der Implementierung wurde mit der kleinstmöglichen Komponente begonnen und diese immer weiter mit den notwendigen Modulen erweitert. Hierfür wurde anfangs auf einer *Minikube* Instanz gearbeitet, einem Tool zum Simulieren von kleinen *Clustern*, welches es erlaubt, schnell Änderungen am *Cluster* vorzunehmen und somit die Entwicklung erleichtert. Hier wurden die ersten kleineren Versionen getestet, bis diese zu groß wurden und das physische Cluster erstellt wurde.
Dieses *Cluster* wurde konfiguriert, sodass mit der Entwicklung der Lösung fortgefahren werden konnte.

Dockerimage

Der erste Schritt der Implementierung besteht darin, ein passendes *Docker Image* für den *Hub* zu erstellen. Dieses basiert auf dem offiziellen *Dockerimage* für *JupyterHub* und wurde um Laufe der Zeit um notwendige Komponenten erweitert. So wurde als erstes *nbgrader* installiert und konfiguriert. Des Weiteren wurden dem *Image* für den *Hub* die notwendigen Pakete für den *LTI-Authentificator* hinzugefügt.

Helm-Chart

Dieses minimale *Image* wurde anschließend erst als *Docker Container* ausgeführt, um die Funktionalität zu überprüfen. Sobald alle Funktionen des *Docker-Images* vorhanden waren, wurde mit dem Entwickeln des *Helm-Charts* angefangen. Dieses *Helm-Chart* beinhaltet alle Grundkomponenten des *JupyterHub Deployments* und bietet die Möglichkeit, diese zu konfigurieren, so können der *Hub* selbst, der *Proxy* und auch die Einzel-Userumgebungen nach Bedarf angepasst werden. Hier werden alle Einstellungen getroffen, die das *Deployment* betreffen. So können Einstellungen vorgenommen werden, den *Usern* bestimmte Ressourcen zugesichert werden oder diese zu beschränken, die Netzwerkeinstellungen für den *Proxy* getroffen und den einzelnen *Pods* Speicher zugewiesen werden.
Nachdem dieses *Helm-Chart* erstellt war, wurde mit der Automatisierung der Installation begonnen.

3.3 Aufbau der erstellten Lösung

Die erstellte Softwarelösung besteht aus mehreren Komponenten, die zum Teil automatisch aus den im *Helm-Chart* enthaltenen Definitionen installiert werden.

Im Verlauf der Arbeit wurden verschiedene Komponenten erstellt, diese werden in den kommenden Abschnitten beschrieben.

3.3.1 Kubernetes Cluster

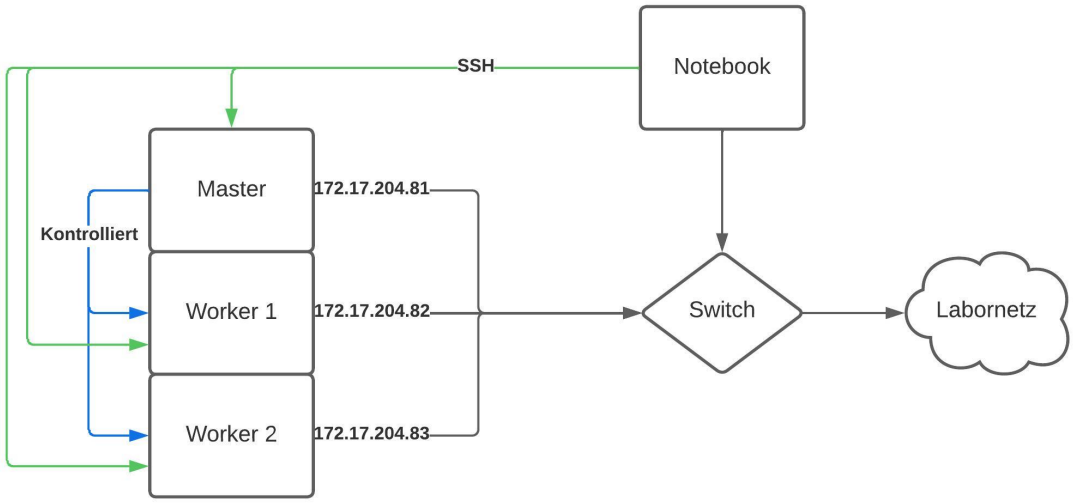

Abbildung 3.1: Aufbau des Clusters mit Netzzwerk, Quelle: Eigene Darstellung

Für die Arbeit wurde ein *Cluster* bestehend aus drei physischen *Nodes* erstellt, Abbildung 3.1, wobei eine *Node* die Rolle des Masters übernommen hat. Als Betriebssystem wurde eine *Ubuntu Server* Version gewählt, da diese die meisten benötigten Pakete bereits beinhaltet. Auf diesen *Nodes* wurden zunächst alle erforderlichen Pakete installiert, sodass im Anschluss das *Cluster* initialisiert werden konnte. Hierzu wurde *Kubeadm* genutzt, ein Tool zum Erstellen von *Clustern* und Hinzufügen von weiteren *Nodes*.[1]
Bei dem zum Einsatz kommendem *Container Runtime Interface* (CRI) handelt es sich um *Containerd*, eine *High-Level-Container-Runtime*, welche von *Docker* entwickelt wurde. *Containerd* ist dafür zuständig, *Images* aus dem *Repository* zu ziehen, diese zu verwalten und an eine untergeordnete *Runtime* zu übergeben. Diese erstellt die *Container* und führt sie aus.[2] Die Zusammenhänge zwischen *Kubernetes* und *Containerd* werden deutlich, wenn man sich Abbildung 3.2 ansieht. Hier ist zu erkennen, dass es verschiedene CRIs gibt, welche den *Open Container Initiative Standard* (OCI) implementieren und somit die Möglichkeit bieten, mit der *Container Runtime* zu kommunizieren.

[1] Aut21.
[2] Don21.

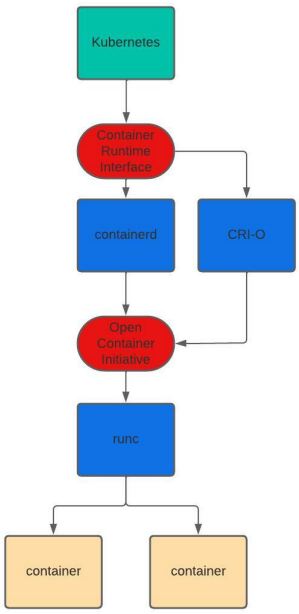

Abbildung 3.2: Zusammenhang zwischen Kubernetes und Containerd, Quelle: [Don21]

3.3.2 JupyterHub

Der *JupyterHub* wird in dem *Helm-Chart* definiert, konfiguriert und über *Helm* deployed. Dabei wird für den *Hub* ein eigener Pod gestartet und automatisch ein *Persistant Volume Claim* (PVC) und das dazugehörende *Persistant Volume* (PV) erzeugt. In diesem PV ist die Datenbank gespeichert, in welcher der *Hub* alle Daten über die registrierten Nutzer und auch die Daten von *nbgrader* abgelegt. Gleichzeitig wird für den *Hub* ein *Service* vom *Typ ClusterIP* erstellt, sodass dieser von den anderen *Pods* innerhalb des *Clusters* erreichbar ist.

3.3.3 Jupyternotebook

Die Userumgebungen in Form von *Jupyternotebooks* werden automatisch auf Anfrage erstellt. Auch hier werden dynamisch PVs und PVCs erzeugt. Anfragen der User werden über *Moodle* erzeugt. Hierfür wird ein Link eingebunden, der auf den *Hub* verweist. Wird dieser Link durch einen Nutzer ausgelöst/betätigt, wird die Authentifizierung angestoßen und der Nutzer angemeldet. Nachdem der *User* authentifiziert wurde, wird ein neuer *Pod* erstellt und mit dem PV verbunden. Sobald dieser läuft, wird der Nutzer vom *Proxy* an seinen *Notebook-Server* weitergeleitet und kann diesen nutzen.

3.3.4 Proxy

Wie auch der *Hub* und die *Notebooks* läuft der *Proxy* in einem eigenem *Pod*. Für den *Proxy* wird ein *Loadbalancer Service* erstellt, der es ermöglicht, den *Proxy* über die externe IP-Adresse zu erreichen. Der *Proxy* leitete dann alle Anfragen an den *Hub* und die einzelnen Notebook-Server weiter. Der *Proxy* benötigt keine eigenen PVs oder PVCs, da keinerlei Daten gespeichert werden, die erhalten werden müssten.

3.3.5 Networking

Den physischen Aufbau des Netzwerks kann man in Abbildung 3.1 erkennen. Hier sind nur die Komponenten abgebildet, die das *Cluster* direkt betreffen bzw. über die mit dem *Cluster* kommuniziert werden kann. Als *Netzwerk Plugin* wurde für dieses Projekt *Calico* gewählt. Eine *Open Source* Lösung, welche *Networking* und *Netzwerkpolicies* für *Container* ermöglicht.[3] *Calico* wurde gewählt, da es eins der besten *Featuresets* von allen *Networking Plugins* für *Kubernetes* hat und somit die besten Voraussetzungen schafft, um dieses Projekt in Zukunft noch um weitere Komponenten zu erweitern. Da lediglich der *Proxy* über einen *Service* des Typs *Loadbalancer* verfügt, ist auch nur dieser von außerhalb des *Clusters* erreichbar. Dementsprechend muss der Zugriff auf die anderen Komponenten des *Deployments* nicht explizit verboten werden.

3.3.6 NGShare

*Ngshar*e ist ein *Backend-Server*, welcher entwickelt wurde, um den Betrieb von *JupyterHub* mit *nbgrader* auf *Kubernetes* zu ermöglichen.[4] Da *nbgader* für seine Funktion normalerweise darauf angewiesen ist, ein gemeinsames Dateisystem für die Studierenden und die Lehrenden zuhaben, welches in einem *Kubernetes Cluster* nicht existiert.
Auf diesem gemeinsamen Dateisystem wird im Normalfall ein sogenannter *Exchange-Ordner* angelegt, auf welchen alle laufenden *Notebook-Server* Zugriff haben, *nbgrader* kopiert dann von Lehrenden veröffentlichte *Assignments* in diesen Ordner, und von hier aus werden sie weiter in die Ordner der Studierenden kopiert.
Dieses Vorgehen ist jedoch auf *Kubernetes* so nicht möglich, da wie erwähnt, der *Hub* und die einzelnen *Notebook-Server* jeweils in einem eigenem *Pod* laufen und somit kein gemeinsames Dateisystem existiert. *Ngshare* ersetzt diesen *Exchange-Ordner* mit einem *REST API Server*, welcher die Aufgabe des *Exchange-Ordners* übernimmt und dafür sorgt, dass die entsprechenden *Assignments* in die Dateisysteme der einzelnen *Pods* für die *Notebook-Server* kopiert werden. *Ngshare* verwendet für die Authentifizierung von *Usern* den *Hub*, somit ist keine weitere Authentifizierung seitens der *User* notwendig. *Ngshare* ist dafür vorgesehen, als eigener *Pod* und *Service* ausgeführt zu werden, der *Proxy* des *JupyterHub Deployments* wird auch für den *ngshare Pod* verwendet und alle *Notebook-Server Pods* können auf ihn zugreifen.[5]

3.3.7 Moodle

Moodle ist eine Lehrplattform, welche das Erstellen von Kursen zulässt. Es wird von vielen Schulen und Hochschulen genutzt, da es die Verwaltung vereinfacht und eine Verteilung von Aufgaben und Information an die Studierenden erlaubt.
Für dieses Projekt wurde eine minimale Installation der *Moodle* Lehrplattform angelegt, diese dient in diesem Aufbau lediglich als Plattform zum Authentifizieren von *Usern* und spielt für die sonstige Funktion dieser Lösung keine große Rolle. Diese Installation läuft außerhalb des *Clusters*, es ist auch nicht notwendig, dass *Moodle* auf dem *Cluster* läuft, da die Anfragen zur Authentifizierung auch über den *Proxy* geleitet werden.

[3] vgl. Pol, S. 38.
[4] KAL20, vgl.
[5] KAL20.

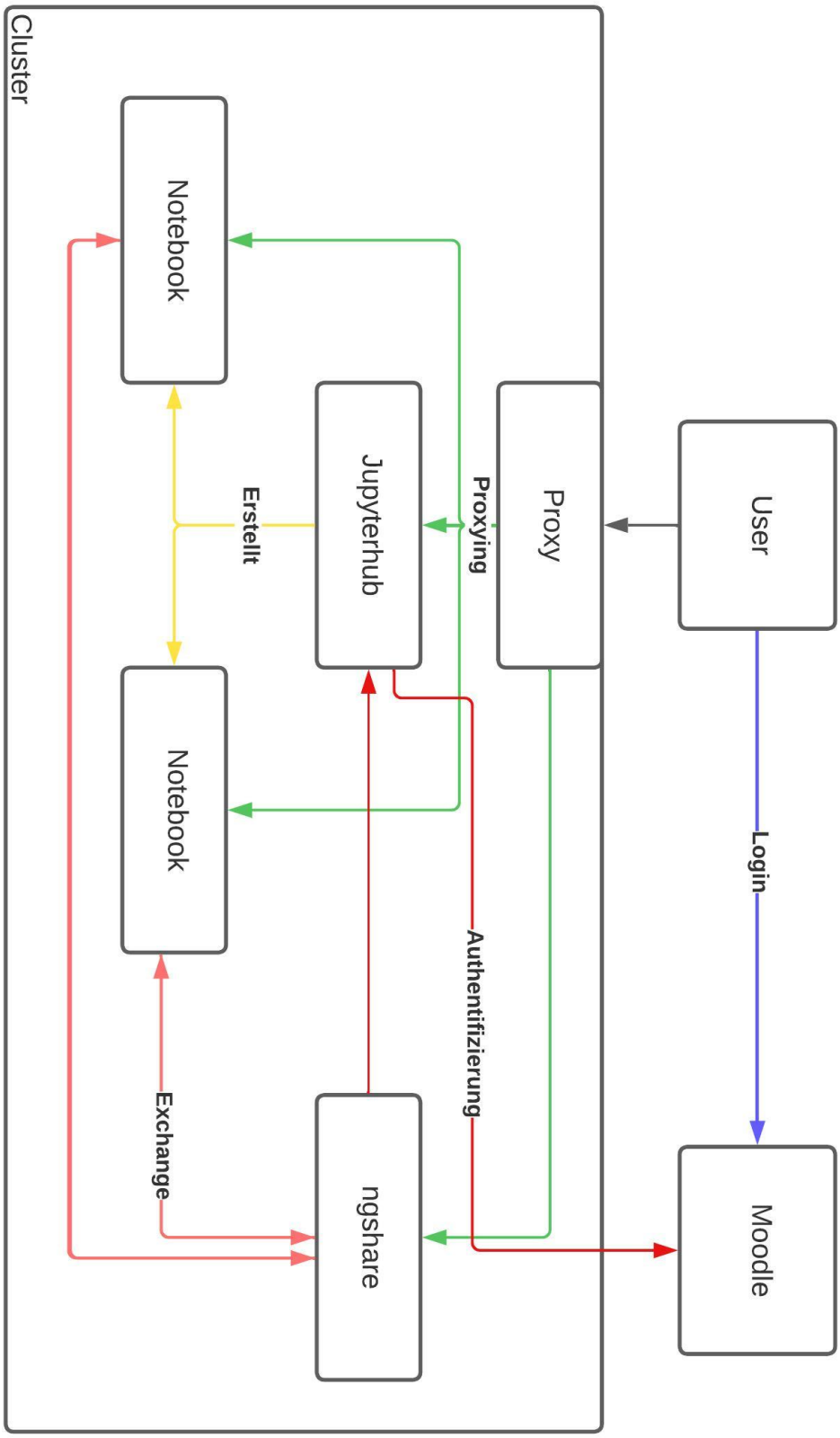

Abbildung 3.3: Darstellung der einzelnen Komponenten und ihrer Zussammenhänge, Quell: [KAL20]

Abbildung 3.3 zeigt noch einmal alle zum Einsatz kommenden Komponenten und macht die Zusammenhänge zwischen ihnen deutlich. Zu erkennen ist, dass *Moodle* in dieser Installation nur den Login der Nutzer managt (Blauer Pfeil) und die Nutzerdaten auf Anfrage an den *Hub* weiter gibt.
Der *Proxy* leitet alle Nutzeranfragen an die entsprechenden *Pods* weiter und die Antworten wieder zurück an den Nutzer (Grüne Pfeile).
Der *JupyterHub* ist dafür zuständig, die Nutzer zu verwalten, und die *Pods* für die *Notebook-Server* zu erstellen (Gelbe Pfeile).
Ngshare übernimmt die Aufgabe, welche normalerweise dem *Exchange-Ordner* zufallen würde und verwaltet die freigegebenen *Assignments* und die eingereichten Lösungen.
Die *Notebook-Server* sind die direkte Nutzerumgebung, in welcher die Nutzer arbeiten.

3.4 Automatisierung

Für die Automatisierung des *Deployments* werden aktuell zwei *Helm-Charts* eingesetzt. Eins für den *Hub* und die dazugehörigen *Jupyter* Komponenten und eins für *ngshare*. Diese *Charts* definieren die Umgebung, welche initialisiert werden soll und erstellen alle benötigten *Pods* und *Services*. Die *Charts* werden mit dem *Helm Commandline-Tool* installiert. Bislang sind für dieses Vorgehen noch zwei Befehle notwendig, eine weitergehende Automatisierung ist jedoch möglich. So wurde zu diesem Zweck ein *Bash-Script* entwickelt, welches die notwendigen Befehle ausführt. Es gibt noch viele weitere Methoden zur Automatisierung von *Helm/Kubernetes Deployments*. Diese sind jedoch nicht ohne ein größeres Vorwissen anzuwenden und in einigen Fällen für die hier geforderte Funktion deutlich zu kompliziert. Und aufgrund ihrer Komplexität nicht für die angestrebte Zielsetzung geeignet.

3.5 Nutzung der Umgebung

3.5.1 Nutzung duch Studierende

Die Nutzung der Umgebung unterscheidet sich für die Studierenden nur geringfügig von der Nutzung einer herkömmlichen *Jupyter-Installation*.
Der Login erfolgt automatisch, sobald der Link im *Moodle-Kurs* angeklickt wird. Ein direkter Login über den *Hub* ist, wie bereits erwähnt, somit nicht mehr möglich. Was zufolge hat, dass die Studierenden sich zuerst in *Moodle* einloggen müssen, bevor sie Zugriff auf die Umgebung erhalten können.
Nach dem der Login können die *Assignments* abgerufen werden. Sobald diese synchronisiert sind, können sie ganz normal bearbeitet und eingereicht werden. Auch können die Studierenden das veröffentlichte *Feedback* zu ihren Aufgaben abrufen und einsehen.

3.5.2 Nutzung duch Lehrende

Für die Lehrenden unterscheidet sich die Nutzung insofern, dass wenn diese sich zum ersten Mal auf dem *Hub* einloggen, ein Kurs erstellte werden muss. Dieser Kurs ist notwendig, damit die Funktionalitäten von *ngshare* gewährleistet werden können. Wenn dieser Kurs erstellt ist, müssen noch die Studierenden hinzugefügt werden. Danach ist die Umgebung vollkommen einsatzbereit. Jetzt können die Lehrenden neue *Assignments* erstellen und veröffentlichen, damit diese von den Studierenden bearbeitet werden können. In der Übersicht der Aufgaben kann auch eingesehen werden, wie viele und welche Studierenden ihre Aufgaben bereits abgegeben haben. Die abgegebenen Lösungen können eingesehen und bewertet werden. Dafür können in die *Assignments*

eingebaute Tests genutzt werden, welche es erlauben, die Lösungen automatisiert auf Funktionalität zu überprüfen. Nach dieser automatischen Überprüfung können noch die Freitext-Aufgaben bewertet werden. Außerdem gibt es die Möglichkeit, *Feedback* in Form von Kommentaren zu hinterlassen. Wenn das Kontrollieren und Kommentieren abgeschlossen ist, kann das *Feedback* an die Studierenden veröffentlicht werden und ihnen somit die Möglichkeit geboten werden, diese einzusehen.

4 Vergleich mit Ähnlichen Lösungen

Im folgenden Kapitel wird der beschriebene Lösungsansatz mit einem anderen Lösungsansatz verglichen. Beide Lösungsansätze verfolgen ähnlich Ziele. Der gewählte Alternativ-Ansatz dient hier nur als Beispiel, da eine ausführliche Liste aller Möglichkeiten den Rahmen dieser Ausarbeitung überschreiten würde. Auch soll der gezogene Vergleich nicht aussagen, dass ein Ansatz besser sei als der andere. Es geht lediglich darum, einen besseren Überblick darüber zu erhalten, was an Alternativen verfügbar ist und somit eine Grundlage zu schaffen, welche die Bewertung der Ergebnisse erleichtern soll.

Der für diesen Vergleich gewählte alternativer Lösungsansatz besteht darin, auf einem *JupyterHub* mehrere Kurse zu erstellen. Dieser alternative Ansatz wurde für einen Vergleich gewählt, da dieser weit verbreitet ist. Auch wurde er für den Vergleich herangezogen, da er im Vorfeld als eine Lösung für die vorliegende Problemstellung diskutiert wurde. Er unterscheidet sich von dem in dieser Arbeit beschriebenen Ansatz dadurch, dass hier auf einem einzelnen *Hub* mehrere Kurse erstellt werden. Dieses Vorgehen würde in einigen Bereichen Vorteile mit sich bringen, in anderen aber auch deutliche Nachteile. In den kommenden Absätzen sollen die Vor- und Nachteile der beiden Lösungsansätze verglichen werden.

4.1 Mehrere Kurse auf einem Hub

Vorteile

Der größte Vorteil mehrere Kurse auf einem einzelnen *Hub* betrieben, besteht darin, dass diese Lösung nicht zwangsweise auf ein *Kubernetes Cluster* angewiesen ist und somit flexibler eingesetzt werden kann.
So sind zum Installieren einer solchen Lösung keine Kenntnisse im Bereich der *Containertechnologien* und *Kubernetes* vonnöten und somit können mehr Anwender eine solche Lösung für ihre Zwecke einsetzen.
Ein weiterer Vorteil besteht darin, dass die Verwaltung der Server und des Netzwerks einfacher ausfallen kann. So muss nur noch ein *JupyterHub Server* verwaltet werden und dieser benötigt nur noch eine IP-Adresse.
Auch wird mit dieser Methode Rechenleistung eingespart, da der gesamte Overhead, der durch *Kubernetes* und die notwendigen *Plugins* anfällt, vermieden werden kann.

Nachteile

Einer der größten Nachteile einen einzelnen *Hub* zu betreibt, besteht darin, dass die Verwaltung der Kurse sowohl für die Lehrenden als auch für die Studierenden deutlich größer ist. Ein weiterer Nachteil ist etwa die Ausfallsicherheit, da wenn der *Hub* ausfallen sollte, sind alle Kurse davon betroffen und nicht wie bei einem *Hub* pro Kurs nur eben dieser Kurs. Außerdem wird für das Aufsetzen einer solchen Umgebung deutlich mehr Zeit benötigt. Ein weiteres Problem besteht darin, dass *nbgrader* den Betrieb von mehreren Kursen zwar unterstützt, dies aber zum jetzigen Zeitpunkt noch zu einem erheblichen Mehraufwand aufseiten der Lehrenden führt.

4.2 Mehrere JupyterHubs auf Kubernetes

Vorteile

Die größten Vorteile, einen eigenen *Hub* für jeden Kurs zu betreiben, sind wie bereits in den Vorteilen von *Kubernetes* erwähnt:

- Erhöhte Ausfallsicherheit
- Starke Isolierung zwischen den einzelnen *Hubs*
- Gute Portierbarkeit
- Einfaches Aufsetzen (duch die hier erstellte Lösung)

Die oben genannten Vorteile werden dadurch ergänzt, dass die Verwaltung der Kurs auf den *Hubs* erleichtert wird, da einige Probleme, welche auftreten können, wenn mehrere Kurse auf demselben Hub betrieben werden, komplett umgangen werden. Das Aufsetzen einer neuen Umgebung ist nicht nur simpler, sondern es wird auch weniger Zeit dafür benötigt. Zusätzlich ist das Entfernen einzelner Umgebungen sehr einfach, da nur das *Helm-Deployment* deinstalliert werden muss und alle dazugehörigen Komponenten automatisch gelöscht werden. Des Weitern können auch noch andere Vorteile von Kubernetes genutzt werden, so können bei zu hoher Auslastung weitere Server automatisch gestartet werden. Als letzter Punkt ist aufzuführen, dass durch die Trennung der einzelnen Kurse in einzelne *Hubs* bei Problemen nur die betroffenen Umgebungen ausfallen.

Nachteile

Der erste zu erwähnende Nachteil dieses Lösungsansatzes bestehen darin, dass die grundlegende Infrastruktur komplizierter ist und mehr Kenntnisse aufseiten der Administratoren bedarf. Auch benötigt diese Lösung einen Rechenleistung-Overhead im Vergleich zu einem Betrieb von Jupyter direkt auf einem physischen Server. Außerdem ist das erste Aufsetzen dieser Lösung deutlich komplexer, falls hierbei nicht die *Helm Charts* genutzt werden, die im Laufe dieser Arbeit erstellt wurden, sondern direkt von Null angefangen wird.

4.3 Zusammenfassung

Zusammengefasst lässt sich sagen, dass beide Lösungsansätze ihre deutlichen Vor- und Nachteile haben, sodass eine pauschale Empfehlung zur Nutzung der einen oder anderen Variante nicht möglich ist. So haben beide Lösungen für ihre eigenen Anwendungsfälle ihre Daseinsberechtigung und es muss aufgrund der jeweiligen Anforderungen entschieden werden.
Dennoch ist auch zuerkennen, dass der in der Arbeit gewählte Ansatz für die im Vorfeld definierten Ziele dieses Projektes deutlich besser geeignet ist. Ausschlaggebend für diese Beurteilung ist vor allem die deutliche Erleichterung aufseiten der Lehrenden, was die Verwaltung der Kurse anbelangt.
Ein weiterer Grund, welcher die Lösung mit mehreren *Hubs* zur besseren Wahl unter Berücksichtigung der Ziele macht, ist die bessere Automatisierung der Erstellung von neuen *Hubs* und die damit einhergehende Vereinfachung, sodass weniger Fachwissen vonnöten ist, um einen neuen *Hub* zu erstellen.

5 Beurteilung der Ergebnisse

In diesem Kapitel wird sich mit den Ergebnissen des Projektes auseinandergesetzt und beurteilt, wie gut die formulierten Ziele erreicht wurden, welche Ziele nicht erreicht wurden und welche Schlussfolgerungen daraus gezogen wurden. Des Weiteren wird auf Probleme eingegangen, welche im Verlauf des Projektes aufgetreten sind, wie diese gelöst wurden und welche nicht gelöst werden konnten.
Im Anschluss wird ein Blick in die Zukunft gewagt und einige Verbesserungsmöglichkeiten und Erweiterungen der erstellten Lösung aufgezeigt, welche im zeitlichen Rahmen des Projektes nicht mehr umgesetzt werden konnten.

5.1 Erreichte Ziele

Zur Erinnerung, die Ziele dieses Projektes waren:
Must-haves:

- Ein *JupyterHub* mit *nbgrader* als *Kubernetes Deployment*
- Mehrere *Deployments* für mehrere Kurse auf demselben *Cluster*
- Authentifizieren über *Moodle*
- Der *Hub* für einen Kurs soll möglichst leicht konfigurierbar sein
- Integratioin der Lehrmaterialien z.B. über *git* und *nbgitpuller*

Nice-to-haves:

- Webanwendung zum Erstellen von *Hubs*

Wenn man die definierten Ziele mit der erstellten Lösung vergleicht, ist zu erkennen, dass einige dieser erfolgreich erreicht werden konnten.
So wurde ein *Helm-Chart* erstellt, welches *JupyterHub* und alle anderen notwendigen Komponenten definiert, sodass diese als *Kubernetes Deployment* installiert werden können.
Auch die Erstellung von mehreren *Deployments* mit dem Ziel, für jeden Kurs eine eigene Umgebung zur Verfügung zu stellen, ist problemlos möglich. Um eine bessere Übersichtlichkeit zu gewährleisten, für jeden Kurs ein eigene *Kubernetes Namespace* erstellt, dies bietet den Vorteil, dass anhand der *Namespaces* sofort zuerkennen ist, welche Komponenten zu welchem Kurs gehören.
Die Konfiguration der *Hubs* für einzelne Kurse ist auch möglich, allerdings ist dies in der aktuellen Version nicht erreichbar, ohne das *Helm-Chart* zu modifizieren. An diesem Punkt können in der Zukunft noch Verbesserungen vorgenommen werden.
Die Authentifizierung über *Moodle* wurde erfolgreich eingerichtet, diese passiert wie bereits beschrieben über den *LTIauthentificator*. Dafür muss im entsprechenden Moodle Kurs eine Aktivität angelegt werden, die einen Link zum *Hub* und einige Konfigurationen enthält.

Die Integration der Lehrmaterialien ist in der final erstellten Lösung mit in *ngshare* integriert. Hier können von den Lehrende Aufgaben veröffentlicht werden und die Studierenden müssen

beim Start ihrer *Notebook-Server* nur mit den veröffentlichten Materialien synchronisieren, somit fällt die Notwendigkeit eines zusätzlichen *Git Repositorys* weg und der Verwaltungsaufwand wird etwas reduziert.

5.2 Nicht erreichte Ziele

Bei den nicht erreichten Zielen fällt als erstes die Webanwendung zum Erstellen von *Hubs* ins Auge. Auch wenn diese nur als Nice-to-have verzeichnet ist, würde diese eine deutliche Erleichterung bieten, da damit die Notwendigkeit des direkten Zugriffs auf das *Cluster* umgangen werden würde. Dies hat zur Folge, dass in der aktuellen Version alle Lehrenden, welche einen *Hub* für ihren Kurs erstellen wollen, Zugriff auf das *Cluster* brauchen oder es einen Administrator geben muss, welcher die angefragten *Hubs* erstellt.
In der zur Verfügung stehenden Zeit war es des weiteren nicht möglich, eine Möglichkeit zu entwickeln, die innerhalb der im *Hub* für Aufgaben vergebenen Punkte automatisch auf *Moodle* zurückzuspielen, sodass dieses aktuell noch von Hand erfolgen muss.

5.3 Probleme im Verlauf des Projektes

Signifikante Probleme gab es bei der Bearbeitung des Projektes nicht. Das größte Hindernis, das am meisten Zeit in Anspruch genommen hat, war die richtige Konfiguration des Netzwerkes innerhalb des Clusters. Dieses Hindernis entstand jedoch nicht aus technischen Gründen, sondern aus mangelnder Erfahrung mit dem Thema und einer übereilten Arbeitsweise. Dadurch ist ein Fehler erst im späteren Verlauf aufgefallen, wodurch mehr Zeit zur Behebung benötigte als eigentlich notwendig gewesen wäre.
Eine weitere Schwierigkeit bestand darin, dass die einzelnen *Pods* aufgrund der Funktion von Dateisystemen in *Kubernetes*, kein Dateisystem teilen und somit eine Lösung für den Austausch der Daten für *nbgrader* gefunden werden musste. Die Lösung für dieses Problem wurde wie bereits erwähnt mithilfe von *ngshare* umgesetzt, welches den sonst benötigten *Exchange-Ordner* durch einen *Backend-Server* ersetzt.
Sonstige Herausforderungen bestanden vor allem darin, die einzelnen Komponente so zu konfigurieren, dass sie wie gewünscht zusammen arbeiten und alle notwendigen Funktionen zur Verfügung stehen. So musste beispielsweise sichergestellt werden, dass die verschiedenen Services richtig konfiguriert sind, sodass alle eine entsprechende IP-Adresse zugewiesen bekommen und die *Pods* richtig untereinander kommunizieren können.

5.4 Einhaltung der Zeitplanung

Wie in Abschnitt 3.2 beschreiben, wurde für die Bearbeitung des Projektes ein grober Zeitplan erstellt. Dieser Zeitplan diente als Richtlinie, um den Verlauf des Projektes im Zeitlichen Rahmen beurteilen zu können und den Überblick zu behalten, welche Teile des Projekts bereits abgeschlossen sind und welche im Zeitplan als abgeschlossen geplant waren.
Dieser Abschnitt soll darüber reflektieren, wie gut die Einhaltung dieser Planung funktioniert hat und welche Probleme es dabei gab.

In den ersten Wochen der Bearbeitungszeit verlief das Projekt nach Zeitplan. Erst als die in Abschnitt 5.3 genannten Schwierigkeiten auftraten, begann der tatsächliche Stand etwas hinter der Planung zurückzufallen. Zwar konnten die meisten der aufkommenden Hindernisse kurzfristig gelöst werden, allerdings summierten sich auch diese im weiteren Verlauf der Arbeit weiter auf. Das größte Problem, das bei der Zeitplanung gemacht wurde, war, dass der zeitliche Aufwand, der für die Erstellung der schriftlichen Ausarbeitung notwendig war, deutlich unterschätzt wurde.

Dies führte dazu, dass in den letzten Wochen der Bearbeitungsfrist deutlich mehr Fokus auf das Schreiben gelegt werden musste und somit einige Optionale Ziele oder Verbesserungen der Erstellten Lösung nicht mehr umgesetzt werden konnten.

5.5 Ausblick in die Zukunft

Vergleicht man die erreichten Ziele mit denen, die nicht erreicht wurden, kann man bereits ein paar Verbesserungsmöglichkeiten erkennen, welche in Zukunft umgesetzt werden könnten, um die erstellte Lösung noch umfangreicher zu machen und noch mehr Features zur Verfügung zustellen. Auf diese und auch nicht in den Zielen definierten Erweiterungen wird in diesem Absatz eingegangen, um aufzuzeigen, was in Zukunft mit etwas mehr Entwicklungsarbeit möglich sein könnte.

Die erste mögliche Verbesserung ist es, das erstellte *Shellscript* zu erweitern, sodass dieses Variablen akzeptiert und somit die individuelle Konfiguration der *Hubs* erweitern würde, ohne dass Änderungen am erstellten *Helm-Chart* notwendig sind. Diese Verbesserung spielt sehr nah mit dem optionalen Ziel einer *Webanwendung* zusammen. Welche für eine minimal funktionierende Version einfach nur aus einer Seite mit Eingabemöglichkeiten für verschiedene Variablen bestehen müsste, die dann im Backend an das erstellte *Script* weiter gegeben werden könnten. So könnten über eine solche Seite verschiedene Einstellungen getroffen werden, wie etwa die Begrenzung oder das Zusichern von Rechenleistung für einzelne Nutzer. Dies ist insbesondere bei einer große Datenverarbeitung oder der Nutzung für KI-Kurse vorteilhaft.
Auch die Möglichkeit, erzielte Punkte direkt auf *Moodle* zurückzuspielen, ist eine Erweiterung, die für viele Anwender sehr sinnvoll sein kann. Da dies den Lehrenden weiteren Verwaltungsaufwand ersparen würde und den Studierenden dabei helfen könnte einen besseren Überblick zu behalten, da sie nicht mehr an zwei verschiedenen Orten nachsehen müssten, wie ihr aktueller Leistungsstand ist. Wie schon erwähnt wurde für diese Erweiterung, innerhalb dieser Arbeit, noch keine technische Möglichkeit entwickelt. Ein möglicher Ansatz hierfür könnte eventuell über eine Datenbank realisiert werden.
Die letzte Verbesserung, auf welche hier eingegangen werden wird, ist es, die Nutzerdaten und alle anderen *Persistant Volumens* nicht auf den *Nodes* des *Clusters* selbst zu speichern, sondern hier für eine andere Lösung zu finden. So könnte der Cloudspeicher verschiedener Anbieter genutzt werden, oder man nutzt ein eigenes *Storage-Area-Network* (SAN). Die Trennung der Daten von den *Nodes* bringt eine erhöhte Sicherheit mit sich. Da, wenn es zu Schäden an den *Nodes* oder deren Speichermedien kommen sollte, kann dies zu einem Verlust der Nutzerdaten führen. Mit der Nutzung eine *SANs* oder dem Speicher eines Cloudanbieters kann dieses Risiko reduzieren. Dies liegt daran, dass die großen Cloudspeicher Anbieter und auch lokale selbst betriebene SANs in der Regel Mechanismen haben, welche vor Datenverlust schützen und verlorene Daten wieder herstellen können.

5.6 Schlussfolgerung

Im Ganzen betrachtet, kann man sagen, dass der größte Teil der im Vorfeld definierten Ziele erreicht wurden. Die erstellte Lösung ist des Weiteren so gestaltet, dass es möglich ist, weitere Module hinzuzufügen oder nicht benötigte zu entfernen. Dies führt in Verbindung mit der Tatsache, dass sie auf Kubernetes und Helm basiert bzw. mit diesen installiert wird dazu, dass sie auf vielen verschiedenen Plattformen mit verschiedenen Voraussetzungen angewandt werden kann. So sind verschiedene Kubernetes Konfigurationen denkbar, auch Moodle als Lehrplattform ist nicht die einzige Möglichkeit, da der LTIauthentificator mehrere verschiedene Plattformen unterstützt. Es wurde also eine funktionelle, wenn auch noch erweiterbare Lösung für die Problemstellung entwickelt, welche auf verschiedenen Umgebungen und für verschiedene Anwendungsfälle eingesetzt werden kann.
Die beschriebenen Verbesserungsvorschläge betreffen nicht die grundlegenden Funktionen, sondern sind lediglich um die Benutzung noch weiter zu erleichtern und eine bessere Nutzererfahrung aufseiten der Lehrenden zu gewährleisten.

6 Zusammenfassung

Dieses Kapitel dient als Zusammenfassung für die in dieser Ausarbeitung beschriebenen Ergebnisse des Projektes und soll noch einmal die wichtigsten Eckpunkte in Erinnerung rufen.

Zu Beginn wurden die grundlegenden Technologien und Softwareprodukte, welche zum Realisieren der Ziele eingesetzt wurden, beschrieben und in einem Umfang erklärt, welcher für das Verständnis und die Beurteilung der Ergebnisse notwendig ist.
Im Anschluss wurden die definierten Ziele beschrieben und die Arbeit in mehrere Pakete aufgeteilt.

Das übergeordnete Ziel des Projektes bestand darin, eine Möglichkeit zu entwickeln, die es ohne einen großen Aufwand erlaubt, *JupyterHub* Umgebungen auf *Kubernetes* zu erstellen. Diese *JupyterHub* Umgebungen sollten über *nbgrader* verfügen, um das Austeilen und Bewerten von *Assignments* zu ermöglichen und es sollte die Möglichkeit bestehen, sich über die Lehrplattform *Moodle* einzuloggen, sodass keine weitere Registrierung notwendig ist.

Im Vorfeld und während der Bearbeitung des Projektes wurden verschiedene theoretische Möglichkeiten zur Umsetzung dieser Ziele in Betracht gezogen. Die ausgewählten Komponenten wurden installiert und so konfiguriert, dass diese zusammen arbeiten.
Es wurde ein *Helm-Chart* erstellt, welches alle Komponenten der *JupyterHub* Umgebung konfiguriert und es ermöglicht, diese als eine Einheit zu installieren. Ein weiteres *Helm-Chart* für die Installation von *ngshare* wurde bereitgestellt. Es wurde ein *Bash Skript* erstellt, welches die beiden *Helm-Charts* und alle weiteren *Kubernetes* Ressourcen wie etwa eine *Storageclass* erstellt und deployed. Dieses *Shell-Script* stellt eine große Vereinfachung beim Erstellen von neuen Umgebungen dar, da es weniger Fachwissen für seine Anwendung voraussetzt als die Installation der Komponenten direkt über *kubectl* und *Helm*.
Die meisten der gesetzten Ziele konnten im Verlauf der Arbeit erreicht werden, so ist der Betrieb von *JupyterHub* Umgebungen auf dem erstelltem *Cluster* möglich, auch die Funktionalitäten von *nbgrader* können genutzt werden, wozu die Nutzung von *ngshare* notwendig war.
Wie beschrieben können auch einige der nicht erreichten Ziele in der Zukunft noch implementiert werden. So kann zum Beispiel eine Webanwendung entwickelt werden, welche es ermöglicht, neue Umgebungen zu konfigurieren und zu erstellen, ohne dass die Nutzer Zugriff auf *Cluster* benötigen.
Ein Paar der im Verlauf auftretenden Probleme wurden beschrieben und es wurde aufgezeigt, wie diese gelöst wurden. Es wurden weitere Verbesserungen aufgezeigt, welche in Zukunft umgesetzt werden können, um die erstellte Lösung noch umfangreicher oder noch robuster zu machen.

7 Persönliche Erkenntnisse

In diesem Kapitel werde ich auf meine persönlichen Erfahrungen und gewonnenen Kenntnisse eingehen und die Ergebnisse des Projektes aus einem persönlichen Blickwinkel betrachten und beurteilen.

7.1 Ausgangssituation

Vor Beginn des Projektes hatte ich bereits grundlegende Erfahrungen mit *Kubernetes* und *JupyterHub*. Diese stammen sowohl aus dem Studium als auch aus einem bereits vorher bearbeiteten Projekt. Die bereits vorhandenen Kenntnisse waren jedoch grade in Bezug auf *Kubernetes* auf einem eher rudimentären Stand, da ich bisher kein so großes Projekt mit *Kubernetes* umgesetzt hatte.

7.2 Erhaltene Kenntnisse

In der Einarbeitung und während der Bearbeitung dieses Projektes konnte ich meine Kenntnisse deutlich erweitern. So konnte ich erste Erfahrungen mit dem Betrieb von *Kubernetes* auf mehreren *Nodes* sammeln. Im Vorfeld hatte ich *Kubernetes* nur auf einzelnen VMs oder in Form von *Minikube*, ein Tool zur Simulation von kleinen *Clustern*, genutzt hatte.
Auch konnte ich meine Kenntnisse im Bereich von *Helm* erweitern und lernen, wie ein *Helm-Chart* aufgebaut ist und welche Konfigurationsmöglichkeiten es bietet. Des Weiteren habe ich gelernt, wie man ein erstelltes *Helm-Chart* installiert und die so erzeugten Komponenten verwaltet.
Die größten Erkenntnisse habe ich jedoch im Bereich des *Kubernetes-Networking* gesammelt. Da ich hier mit vorher nur minimale Berührungspunkte hatte. Ich habe mich mit verschiedenen *Netzwerk-Plugins* beschäftigt, mich mit ihren einzelnen Vor- und Nachteilen auseinandergesetzt und gelernt, wie sie zu installieren und anzuwenden sind.
Ich konnte Kenntnisse sammeln, welche mir die Umsetzung des Projektes erlaubten und mich auf zukünftige Projekte im Themenbereich *Kubernetes* vorbereitet haben.

7.3 Betrachtung der Ergebnisse

Aus persönlicher Sicht bin ich sehr zufrieden mit den Ergebnissen des Projektes, es wurde eine Lösung erarbeitet, welche die grundlegenden Ziele erfüllt und es ermöglicht, für einzelne Kurse eine *JupyterHub* Umgebung zu erstellen. Mit dem Netzwerk innerhalb des *Clusters* hatte ich, aufgrund meiner Unerfahrenheit in diesem Bereich, einige Schwierigkeiten, die ich aber nach einer Einarbeitungsphase lösen konnte.
Auch meine Vorgehensweise würde ich als positiv auffassen, dadurch, dass ich mit der minimalsten möglichen Jupyterumgebung angefangen habe, konnte ich mich nach und nach mit den benötigten Komponenten beschäftigen und in der immer komplizierter werdenden Umgebung somit einen guten Überblick behalten.
Auch wenn nicht alle Ziele erreicht werden konnten, bin ich doch der Meinung, dass die erstellte Lösung ein weites Einsatzgebiet hat und gegenüber den bisher existierenden Lösungen einige Vorteile bietet.

So konnte der Aufwand für die Lehrenden deutlich gesenkt werden, es wurden Möglichkeiten eingebunden, sich über verschiedene Plattformen zu authentifizieren und die Verteilung von Lehrmaterialien konnte realisiert werden. Die aufgezeigten Verbesserungsmöglichkeiten sind meiner Meinung nach nicht essenziell für die gewünschten Funktionen, sondern sind nur Erweiterungen, um den Aufwand in der Erstellung und Verwaltung der Umgebungen noch weiter zu senken.

7.4 Betrachtung der Zeitplanung

Aus persönlicher Sicht war die Zeitplanung sehr hilfreich, auch wenn diese nur als eine grobe Richtlinie genutzt wurde und nicht als eine strickte Vorgabe. Sie half dabei, die Zeit nicht aus den Augen zu verlieren und den Fortschritt zu kontrollieren. Die eher grobe Planung hing vor allem damit zusammen, dass ich bisher wenig Erfahrung im Umsetzen solcher großen Projekte hatte und mir im Vorfeld nicht bewusst war, welche Teile der Implementierung wie viel Zeit in Anspruch nehmen würden. So konnte die Planung für den Großteil des Projektes eingehalten werde. Der größte Fehler, der bei der Zeitplanung begangen wurde, war jedoch, dass ich unterschätzt hatte, wie viel Zeit ich zum Verfassen der schriftlichen Ausarbeitung tatsächlich brauchen würde. Diese Fehlplanung führte am Ende dazu, dass ich nicht alle Ziele, die ich mir vorgenommen hatte, umsetzen konnte. Ich habe aber daraus gelernt, dass ich bei zukünftigen Projekten dieser Art deutlich mehr Zeit für das Schreiben einer Ausarbeitung bzw. einer Dokumentation einplanen werde.

Literatur

[Aut21] The Kubernetes Authors. *Kubeadm*. 2021. URL: https://kubernetes.io/docs/reference/setup-tools/kubeadm/ (besucht am 06. 07. 2023).

[Aut23] The Kubernetes Authors. 2023. URL: https://kubernetes.io/docs/concepts/overview/ (besucht am 03. 07. 2023).

[Bai17] Jonathan Baier. *Getting Started with Kubernetes*. Second Edition. Bd. 1. Birmingham, UK: Packet Publishing Ltd., 2017.

[BBH21] Brendan Burns, Joe Beda und Kelsey Hightower. *Kubernetes - Eine kompakte Einführung*. 2., aktualiesierte und erweiterte Auflage. Bd. 2. Heidelberg, Deutschland: dpunkt.verlag GmbH, 2021.

[BD20] Andrew Block und Austin Dewey. *Learn Helm*. Hrsg. von 1. Pack Publishing Ltd., 2020.

[Bla+18] Douglas Blank u. a. *nbgrader: A Tool for Creating and Grading Assignments in the Jupyter Notebook*. 2018.

[Con23] Project Jupyter Contributors. 2023. URL: https://jupyterhub.readthedocs.io/en/latest/ (besucht am 04. 07. 2023).

[Don21] Tom Donohue. *Die Unterschiede zwischen Docker, containerd, CRI-O und runc*. 2021. URL: https://www.kreyman.de/index.php/others/linux-kubernetes/232-unterschiede-zwischen-docker-containerd-cri-o-und-runc (besucht am 07. 07. 2023).

[IH23] Bilgin Ibryam und Roland Huß. *Kubernetes Paterns. Reusable Elements for Designing Cloud Native Applications*. Bd. 2. Sebastopol, CA 95472: O'Reilly Media, Inc., 2023.

[KAL20] Team KALE. 2020. URL: https://ngshare.readthedocs.io/en/latest/index.html# (besucht am 11. 07. 2023).

[Min+21] Francesco Minna u. a. „Understanding the Security Implications of Kubernetes Networking". In: *Copublished by the IEEE Computer and Reliability Societies* September/October (2021).

[Nat21] Cloud Native. *Helm Project Journey Report*. 2021. URL: https://www.cncf.io/reports/helm-project-journey-report/ (besucht am 05. 07. 2023).

[Pol] Alex Pollitt. *Introduction to Kubernetes Networking and Security*. Tigera.

[Ran23] SUSE Rancher. 2023. URL: https://ranchermanager.docs.rancher.com/v2.5/how-to-guides/advanced-user-guides/manage-clusters/create-kubernetes-persistent-storage/manage-persistent-storage/about-persistent-storage (besucht am 19. 07. 2023).

[Vel22] Nived Velayudhan. 2022. URL: https://opensource.com/article/22/6/kubernetes-networking-fundamentals (besucht am 21. 07. 2023).

[Wat22] Stephen Watts. 2022. URL: https://www.splunk.com/en_us/blog/learn/state-of-kubernetes.html#:~:text=A%20record%20high%20of%2096,in%20the%20prior%20year%27s%20survey). (besucht am 23. 07. 2023).

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Werken anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

27. Juli 2023

Jonas Matthias Kloos