HOCHSCHULE MITTWEIDA
University of Applied Sciences

BACHELORARBEIT

Herr
Paul Pfleger

Erstellung und Analyse einer Open-Source Cloud-Transcoding Infrastruktur

Mittweida, Januar 2023

Fakultät **Medien**

BACHELORARBEIT

Erstellung und Analyse einer Open-Source Cloud-Transcoding Infrastruktur

Autor:
Paul Pfleger

Studiengang:
Media and Acoustical Engineering

Seminargruppe:
MG19wA

Erstprüfer:
Prof. Dr. Ing. Frank Zimmer

Zweitprüfer:
M.Sc. Marcus Mathy

Einreichung:
Mittweida, 24.01.2023

Faculty of **Media Sciences**

BACHELOR THESIS

Creation and analysis of an open-source cloud transcoding infrastructure

Author:
Paul Pfleger

Course of Study:
Media and Acoustical Engineering

Seminar Group:
MG19wA

First Examiner:
Prof. Dr. Ing. Frank Zimmer

Second Examiner:
M.Sc. Marcus Mathy

Submission:
Mittweida, 24.01.2023

Bibliografische Angaben:

Pfleger, Paul:

Erstellung und Analyse einer Open-Source Cloud-Transcoding Infrastruktur

Creation and analysis of an open-source cloud transcoding infrastructure

62 Seiten, Hochschule Mittweida, University of Applied Sciences, Fakultät Medien, Bachelorarbeit, Mittweida, 2023

Abstract

Ziel der vorleigenden Bachelorarbeit ist es Eigenschaften des Cloud-Computings zu untersuchen. Konkret wurden dabei jene Vorzüge betrachtet, die bei der Anwendung im Video Transcoding von Vorteil sind. Die Untersuchung erfolgte an einer im Zuge der Arbeit erstellten Software, die, wo immer möglich, aus Open-Source Bestandteilen aufgebaut wurde. Diese ist in der Lage Videofiles, die sich in einem Objektspeicher befinden mittels einer etablierten Transcoding-Biliothek zu wandeln. Dazu kann der Nutzer Zielparameter wie Codec und Datenrate in einer Web-Nutzeroberfläche eingeben. Neben diesen funktionalen Rahmenbedingungen, werden im Zuge der Arbeit Zielsetzungen formuliert, die gerade für Cloudanwendungen typisch sind. Um diese Eigenschaften nachzuweisen wurden Messungen an der erstellten Infrastruktur vorgenommen, die eine objektive Betrachtung erleichtern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API . . . . . Application Programming Interface

AV . . . . . Audio und Video

AWS . . . . . Amazon Web Services

CPU . . . . . central processing unit

EBS . . . . . Amazon Elastic Block Store

EC2 . . . . . Amazon Elastic Compute Cloud

HDR . . . . . high dynamic range

HTTP . . . . . Hypertext Transfer Protocol

IC . . . . . intigrated circuit

ITU . . . . . international telecommunication union

JSON . . . . . JavaScript Object Notation

PQ . . . . . Perceptual Quantizer

PSNR . . . . . peak signal to noise ratio

REST . . . . . Representational State Transfer

S3 . . . . . Amazon Simple Storage Service

SDK . . . . . software development kit

SDR . . . . . standard dynamic range

SSH . . . . . Secure Shell

SSIM . . . . . structural similarity

TCO . . . . . total cost of ownership

TSL . . . . . Transport Layer Security

USD . . . . . US-Dollar

VPC . . . . . virtual private Cloud

Symbolverzeichnis

$\in$ ..... **Element von:** $n \in \mathbb{N}$ wird gesprochen als n ist ein Element von $\mathbb{N}$

$\mathbb{N}$ ..... **Menge der Natürlichen Zahlen**

$\rfloor$ ..... **Schließende Gausklammer Aufrundungsfunktion:**
Für eine reelle Zahl x ist $\lfloor x \rfloor$ die größte ganze Zahl, die kleiner oder gleich x ist.

$\rceil$ ..... **Schließende Gausklammern Abrundungsfunktion:**
Für eine reelle Zahl x ist $\lceil x \rceil$ die kleinste ganze Zahl, die größer oder gleich x ist.

$>>$ ..... **viel größer als:** Für $x >> y$ gilt x ist sehr viel größer als y.

$\lfloor$ ..... **Öffnende Gausklammer Aufrundungsfunktion:**
Für eine reelle Zahl x ist $\lfloor x \rfloor$ die größte ganze Zahl, die kleiner oder gleich x ist.

$\lceil$ ..... **Öffnende Gausklammern Abrundungsfunktion:**
Für eine reelle Zahl x ist $\lceil x \rceil$ die kleinste ganze Zahl, die größer oder gleich x ist.

1 Einleitung

1.1 Fachliche Relevanz

Die Verbreitung von Videoinhalten gestaltet sich zunehmend als ein komplexer Prozess, in dessen Verlauf eine Vielzahl von Arbeitsschritten durchlaufen wird. In besonderem Maße trägt dazu der zunehmende Konsum von Videoinhalten auf mobilen Endgeräten bei. So wurden im ersten Quartal des Jahres 2022 56 % der Videoaufrufe allein durch Smartphones getätigt [Brightcove Inc, 2022]. Dieser Umstand bringt zweierlei Herausforderungen mit sich: Zum einen ist nicht mehr, wie bei der Verbreitung von Inhalten über das lineare Fernsehen, von einer definierten Auflösung des Endgerätes auszugehen. Handys, Tablets und Laptops differieren in ihrer Bildschirmauflösung zuweilen stark. Darum sind angepasste, entsprechend skalierte Inhalte notwendig. Zum anderen kann durch die Nutzung von mobilen Zugriffspunkten auch nicht von einer definierten Verbindungsbandbreite zwischen Endgerät und bereitstellendem Server ausgegangen werden [Hong and Kim, 2002, S. 115]. Um dem Nutzer dennoch eine gleichmäßig hohe Qualität des Service zu bieten, wird eine Anpassung der Videodatenrate an die zur Verfügung stehende Bandbreite nötig [Pereira et al., 2010, S. 482ff.]. Dies geschieht durch die Wandlung von Videoinhalten mit Komprimierungsalgorithmen, die die Datenrate der Inhalte teils erheblich reduziert [Andrey et al., 2020]. Aus den beiden genannten Gründen wächst jedoch auch die Menge der Formate, in denen Inhalte angeboten werden müssen, stetig [Mariana et al., 2020]. Einen zusätzlichen Beitrag zu dieser Entwicklung leistet die zunehmende Nachfrage nach Videoinhalten mit einem erweitertem Dynamikumfang. So bieten auch Streaminganbieter zum Teil bereits Inhalte dieser Art an [Netflix Services Germany GmbH, 2022]. Videomaterial, das auf diese Weise produziert wurde, muss dennoch ohne signifikante Qualitätsverluste auch auf Geräten wiedergeben werden können, die nicht für die HDR-Wiedergabe ausgelegt sind. Die Konsequenz ist eine Reihe von teils komplexen Umwandlungsschritten, die entlang der Produktionskette nötig sind. Nur so kann auf einer Vielzahl von Endgeräten ein zufriedenstellendes Nutzererlebnis sichergestellt werden. Um dieser Herausforderung zu begegnen, kann die Cloud-Technologie eine mögliche Herangehensweise darstellen. Damit wäre die Medien-Branche nicht die erste, in der diese Technologien zu einer grundlegenden Neugestaltung von Arbeitsweisen führt. Die Bereitstellung von Ressourcen in der Cloud hat bereits in der Software- und IT-Landschaft zu einer maßgeblichen Umgestaltung von Arbeitsweisen geführt [Zhang et al., 2007, S. 701].

Die steigende Verbreitung dieser Technologie zeigt sich im stetigen Wachstum des damit verbundenen Marktes. So gaben in einer Studie der Unternehmensberatung KPMG nur 3 % der befragten Unternehmen mit mehr als 20 Mitarbeitern an, dass Cloudcomputing für sie kein relevantes Thema sei [KPMG AG, 2022]. 78 % der befragten Unternehmen haben dabei weiterhin die Absicht, mit Cloud Ressourcen Ihre Kosten zu senken (ebenda). Und tatsächlich sieht auch [Kerschbaumer, 2022] die As-a-Service-Technologie, wie die zu besprechende Art der Ressourcenbereitstellung auch genannt wird, das Potenzial, auch die Kosten für die Erstellung und Verbreitung von Video-Inhalten zu senken. Zusätzlich an Relevanz gewinnt die Fragestellung, wenn man betrachtet, dass die Nachfrage nach Videoinhalten nicht zeitlich konstant ist. Viele stark nachgefragte Inhalte sind von Interesse aufgrund ihrer Aktualität. Damit ist die Zeit, die vergeht, bis der Inhalt zu den Nutzerinnen und Nutzern gelangt, keineswegs gleichgültig. Die damit verbundene dynamische Nachfrage stellt Anbieter, vor infrastrukturelle Herausforderungen [Wang and Kim, 2019, S. 113]. Wollten sie selbst Server vorhalten, die

den beschriebenen Anforderungen gerecht werden, so würde diese, „On-Premises“ genannte, Form der Bereitstellung, mit hohen Anschaffungs- und Unterhaltskosten einhergehen [Lisdorf, 2021, S. 23 f.]. Dabei würde jedoch nur in den seltensten Fällen die Rechenleistung des Systems ausgenutzt [Lampe, 2010, S. 74]. Das Nutzen von Ressourcen eines Cloud-Anbieters kann dabei Abhilfe schaffen. Das flexible Anmieten von Rechen-, Speicher- und Netzwerkkapazitäten ermöglicht es, sich der Nachfrage von Kundenseite anzupassen. Neben der steigenden Effizienz (ebenda) kann so auch das Kundenerlebnis deutlich verbessert werden. Dies liegt unter anderem am Aspekt der Hochverfügbarkeit, der auch im Verlauf der Arbeit näher betrachtet wird. Damit ist gemeint, dass Cloud-Anbieter verschiedene Mechanismen anbieten, um ein hohes Maß an Ausfallsicherheit zu bieten [Reznik et al., 2021, S. 1].

Diese Möglichkeiten, Dienste redundant aufzubauen, werden im Bereich der Bereitstellung von Websites und anderen Onlinediensten schon länger eingesetzt, um Ausfälle zu vermeiden [Pfitzinger and Jestädt, 2016, S. 311]. So ist es nicht unüblich, durch das Aufteilen der angemieteten Ressourcen auf verschiedene Rechenzentren in unterschiedlichen Regionen, auch Situationen wie Stromausfälle oder Naturkatastrophen abzusichern [Pfitzinger and Jestädt, 2016, S. 211 f.]. Diese Eigenschaft kann sich als erfolgskritisch für die Medienbranche herausstellen. Schließlich ist es eine Anforderung, die an Live-Übertragungen gestellt wird, dass sie auch in Ausnahmesituationen funktions- und handlungsfähig bleiben. Weiterhin kann ein Ausfall im laufenden Betrieb, neben wirtschaftlichen Risiken, auch immense Folgen für die Reputation eines Übertragungsdienstleisters haben.

Damit stellt sich eine Situation dar, in der die Anforderungen an die Formatumwandlung von Medieninhalten zunehmend steigen. Neue Produktionsstandards treffen auf homogene Wiedergabebedingungen. Gleichzeitig bleiben jedoch die Anforderungen an die Verlässlichkeit der Wandlungsinfrastruktur hoch, würde doch ein Ausfall Verzögerungen im Produktionsprozess bedeuten. Damit ergibt sich die Relevanz der Frage, ob die Cloud-Technologie einen Teil zum Erfüllen dieses Anforderungsprofils beitragen kann.

1.2 Theoretische Herleitung der Forschungsfrage

Neben den dargelegten Begründungen für die Verwendung der Cloud, die aus der Praxis eines Medienunternehmens stammen, gibt es jedoch auch theoretische Überlegungen, die auf die wachsende Bedeutung von Cloudlösungen hindeuten. So stellte Gordon Moore, der spätere Gründer von Intel, schon 1975 die Therorie auf, dass sich die Anzahl der Elemente auf einem Mikrochip gleicher Größe alle zwei Jahre verdoppelt [Mack, 2011, S. 202]. Damit ergibt sich der mathematische Zusammenhang:

$$T_{\mathrm{t}} = T_0 \alpha^{\frac{t}{\tau}} \tag{1.1}$$

Wobei T_{t} und T_0 die Anzahl der Komponenten zum Zeitpunkt t bzw. zum Ausgangszeitpunkt sind [Steinicke, 2016, S. 129]. Weiterhin beschreibt α die Wachstumsrate und τ nach wie vielen Zeiteinheiten die Wachstumsrate α zu beobachten ist. Moore setzte beide Werte im Jahr 1975 auf den Wert zwei fest. Dabei zeigen Erhebungen, dass die Gesetzmäßigkeit die Entwicklung der Prozessoren bis in die Mitte der 2010er-Jahre tatsächlich recht zuverlässig vorhersagt. Aufbauend auf dieser Beobachtung und unter Einbezug der bis dato beobachteten Entwicklung der Computertechnik, stellte

David House die These auf, dass intigrated circuit (IC) ihre Leistung sogar alle 18 Monate verdoppeln. Damit trug er dem Umstand Rechnung, dass sich zum einen die Anzahl der Elemente eines IC alle zwei Jahre verdoppelt, zum anderen durch deren effektivere Platznutzung eine höhere Taktung der Elemente möglich sei (ebenda). Allerdings erreichen diese zugrundeliegenden Beobachtungen seit Beginn der 2010er-Jahre allmählich eine natürliche Grenze, wie schon in [Kurzweil, 2004, S. 379] vorhergesagt. Der Autor Ray Kurzweil bezieht jedoch noch eine weitere Dimension in die Betrachtung mit ein. Er betrachtet die verfügbare Rechenleistung, in Bezug auf die damit verbundenen Kosten. Dabei stellte er fest, dass sich in einer logarithmischen Darstellung der in der Praxis beobachteten Wertepaare, keine lineare Funktion, welche exponentielles Wachstum bedeuten würde, einstellte. Vielmehr stellte er erneut exponentielles Wachstum fest, wie sich in der Grafik, entnommen aus seinem Aufsatz, zeigt.

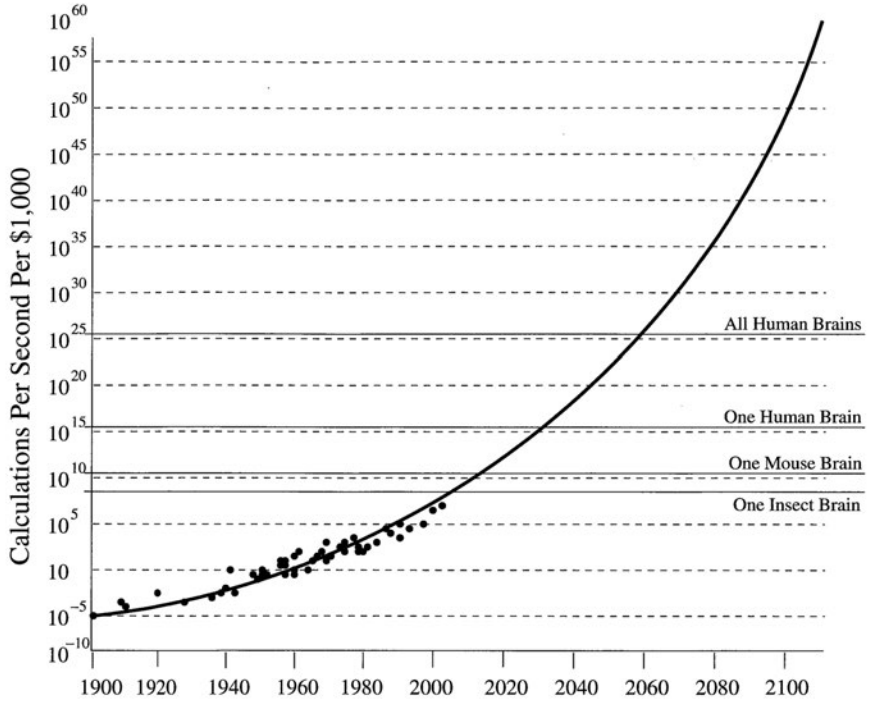

Abbildung 1.1: „The exponential growth of computing“ [Kurzweil, 2004, S. 392]

In der Darstellung wird also die Menge der Kalkulationen, die unter Einsatz von 1000 US-Dollar (USD) vorgenommen werden können, betrachtet. In der Grafik zeigt sich der exponentielle Anstieg trotz der logarithmischen Darstellung der Ordinatenachse. Daraus lässt sich ableiten, dass auch die Geschwindigkeit, mit der die Kosteneffizienz von Berechnungen zunimmt, exponentiell steigt. In Bezug auf den Gegenstand der Arbeit bedeutet das, der Kauf von Hardware wird in zunehmend kürzeren Intervallen unrentabel. Mithilfe der angeführten Gesetzmäßigkeit lässt sich dies wie folgt begründen: Der Graph impliziert in seiner Umkehrung, dass die Kosten, die nötig sind, um n Rechenoperationen in einer Zeit T durchzuführen, stetig sinken. Und mehr noch, die Zeit, die vergeht, bis es möglich ist, die gleiche Rechenoperation in derselben Zeit unter Einsatz der Hälfte des Kapitals durchzuführen, nimmt kontinuierlich ab.

An dieser Entwicklung kann jedoch nicht partizipiert werden, wenn Hardware zur längerfristigen Nutzung erworben wurde. Eine Recheneinheit, die sich in Besitz einer Unternehmung befindet, wird im Mittel, über ihre gesamte Lebenszeit hinweg, zu den gleichen Kosten n Operationen in der Zeit T durchführen. Für eine Lebenszeit der Hardware von durchschnittlich drei bis vier Jahren

[Barroso et al., 2019, S. 71] kann also von dieser Entwicklung nicht profitiert werden. Die Schlussfolgerung ist die Notwendigkeit maximaler Flexibilität in der Auswahl der genutzten Hardware. Eine Forderung, der sich mithilfe der Cloud-Technologie angenähert werden kann. So setzen viele Cloudanbieter auf selbstentwickelte Rechenchips, die in der Lage sein sollen, Rechenleistung noch effizienter bereitzustellen [Joachim and Kevin, 2022].

Bei der Betrachtung von Video-Transcoding als einen rechenaufwändigen Prozess [Akramullah, 2014, S. 169], zeigt sich, dass gerade diese Anwendung von der Kostensenkung und Effizienzsteigerung, die mit einer flexiblen Anpassung von Hardwareressourcen, einhergeht, profitieren kann. Damit zeigt sich, dass auch vor einem theoretischen Hintergrund eine Begründung für die Verwendung von Cloudressourcen, zum Zweck des Video-Transcodings gegeben werden kann.

1.3 Formulierung und Abgrenzung der Forschungsfrage

Gegenstand der Arbeit wird, aufbauend auf den vorangegangenen Betrachtungen, die Frage sein:

Welche Eigenschaften des Cloudcomputings lassen sich an einer Open-Source basierten Cloud-Transcoding-Infrastruktur nachweisen?

Dabei wird die Frage anhand einer Softwarelösung beantwortet, die im Zuge der Arbeit erstellt wurde. Folglich werden auch nur jene Eigenschaften untersucht, die für die vorliegende Transcoding-Anwendung von Relevanz sind. Die Betrachtung beschränkt sich also auf Aspekte, die unter Verwendung des zur Arbeit gehörigen Quellcodes festgestellt werden können. Die Nachweismethode ist die Messung relevanter Größen an der, im Zuge der Arbeit entwickelten, Infrastruktur. Durch die Auswertung, der so erhaltenen Daten, werden die identifizierten Eigenschaften nachgewiesen. Konkret wird die Feststellung der Eigenschaften also durch die Wandlung verschiedenartiger Dateien erfolgen. Dazu ist zu sagen, dass es sich bei der umgesetzten Software um eine modellhafte Implementierung handelt, die vorrangig für die Verwendung durch sachkundige Anwender bestimmt ist. Dabei wird von einer Zielgruppe der Anwendung ausgegangen, zu der Unternehmen gehören, deren Kernwertschöpfung die Erstellung von Bewegtbildinhalten darstellt.

2 Begriffsbestimmungen und Hintergrundwissen

Um in der folgenden Arbeit wesentliche Begriffe des zu behandelnden Themenkomplexes sicher zu verwenden, werden diese im Folgenden geklärt. Weiterhin werden relevante Gegenstände der Arbeit zuerst theoretisch erläutert, um das weitere Verständnis der Arbeit zu erleichtern.

2.1 Cloud-Computing

Der Kernbestandteil des praktischen Anteils dieser Arbeit ist die Bereitstellung einer Softwarelösung in der Cloud. Aus diesem Grund wird der Begriff nachfolgend definiert. Weiterhin ist es zweckmäßig, eine Reihe von damit verbundenen Begriffen zu klären, und anschließend in den Zusammenhang eingeordnet.

2.1.1 Definition Cloud-Computing

Zur Begriffsbestimmung des Ausdruckes Cloud können die verschiedenen Ansätze, aus der Literatur, herangezogen werden. So beschreibt [Krypczyk and Bochkor, 2020, S. 677] Cloudcomputing als eine Form der Infrastrukturbereitstellung, bei der ein Teil des IT-Leistungsspektrums ausgelagert wird. Die Auslagerung kann dabei innerhalb des Unternehmens oder an einen externen Dienstleister erfolgen. Damit kann zum einen Hardware gemeint sein, die sich nicht mehr im Eigentum dessen befindet, der sie nutzt. Vielmehr stellt der Eigentümer diese zur vorübergehenden Nutzung zur Verfügung. Weiterhin führt [Krypczyk and Bochkor, 2020, S. 677] als Merkmal an, dass in der Regel keine Anpassung der bereitgestellten Serviceleistungen an individuelle Kunden erfolgt. So kann der Kunde zwar frei aus einer Reihe von Leistungen wählen, und so bedarfsgerecht Ressourcen anmieten. Der Katalog dieser Leistungen ist dabei jedoch im Allgemeinen für eine Vielzahl von Kunden. Der metaphorische Bezug zu Wolken rührt dabei laut [Pattanayak et al., 2021, S. 14] von einer ungeordneten, großen Menge an Ressourcen her. Zwei weitere charakteristische Merkmale nennt das National Institute of Standards and Technology, eine US-Behörde für Normungen. Diese seien der Zugriff auf die Ressourcen mittels Standard Netzwerkmechanismen und die Skalierbarkeit der angemieteten Rechen- und Speicherkapazitäten [Mell and Grance, 2011]. So können Kunden bei einem Cloudanbieter bedarfsgerecht Leistungen anfordern. Damit geht auch die Möglichkeit einher, bei einer Bedarfsveränderung die in Anspruch genommenen Leistungen flexibel anzupassen. Damit ist Cloudcomputing von der sogenannten „On-Premises“-Nutzung abzugrenzen. Diese stellt die tradierte Form der Serverbereitstellung dar [Erl and Puttini, 2013, S. 36]. Dabei wird die Hardware innerhalb der Umgebung der Organistion betrieben, die sie auch nutzt (ebenda). Man kann also sagen, dass dabei Ressourcen von der gleichen Entität erworben und genutzt werden. Weniger abstrakt lässt sich sagen, dass die Unternehmenseinheit, die Bedarf an Hardwareressourcen hat, diese auch selbst betreibt [Boillat and Legner, 2013, S. 51]. In der Regel geschieht dies in Serverräumen, innerhalb der Unternehmenseinheit (ebenda).

Public Cloud als Teilmenge des Cloudbegriffs Die Bereitstellung von Ressourcen in der Cloud teilt sich im Wesentlichen in drei Modelle, die sich teils deutlich voneinander unterscheiden. So wird zwischen „private“, „hybird“ und „public“-Cloud Modellen unterteilt. Da nur letztes in der Arbeit Anwendung findet, wird dieses herangezogen. In seiner Betrachtung trägt [Frank, 2019, S. 112] zwei

bezeichnende Merkmale dieser Technologie zusammen. Zum einen benennt er die Zugänglichkeit des Dienstes für die Öffentlichkeit als ein wesentliches Merkmal. Gemeint ist, dass grundsätzlich ein breites Publikum den Dienst in Anspruch nehmen kann. Zum anderen sieht er den Zugriff der Kunden über das Internet als charakteristisch. So greift der Leistungsnehmer bei diesem Modell auf Ressourcen zu, die grundsätzlich allen Kunden des Cloudanbieters zur Verfügung stehen. Auf diese Weise nutzen zahlreich Kunden gleichzeitig die gleiche Hardware, die vom Cloudanbieter bereitgestellt wird. Die Abtrennung der Kundensysteme geschieht durch Virtualisierung. [Baun, 2022, S. 253 ff.] beschreibt diese so, dass Ressourcen in einer logischen Schicht abstrahiert werden, mit dem Ziel deren Auslastung zu optimieren. So werden auf dieser abgegrenzten Schicht virtuelle Rechner erstellt, die sich in der Handhabung wie herkömmliche Server verhalten. Die so erzeugten Ressourcen greifen jedoch nicht auf die zugrundeliegende Hardware direkt zu. Vielmehr regelt ein Host-Betriebssystem oder ein sogenannter Hypervisor den Hardwarezugriffe (ebenda). Damit können Ressourcen geteilt genutzt werden, ohne, dass die verschiedenen Nutzer mit den Daten oder Prozessen des jeweils anderen Nutzers interagieren können (ebenda).

2.1.2 Skalierbarkeit als zentrale Eigenschaft des Cloudcomputings

Ein wesentliches Versprechen, das die Cloud macht, ist es, Dienste bedarfsgerecht betreiben zu können. Entsprechend geht damit die Anforderung einher, im Falle eines steigenden Ressourcenbedarfs, die Menge an verwendeten Infrastrukturkomponenten passend zu erhöhen. Diese Eigenschaft wird gemeinhin als Skalierbarkeit bezeichnet. [Luntovskyy and Spillner, 2017] definiert diese so:

> „Die Skalierbarkeit eines vert. Systems beschreibt dessen Laufzeitverhalten bei einer Änderung verschiedener Problemgrößen.“ [Luntovskyy and Spillner, 2017, S. 356]

Die Realisation dieser Eigenschaft erfolgt üblicherweise auf zwei verschiedene Weisen: Zum einen gibt es die Möglichkeit der vertikalen Skalierbarkeit (ebenda), die im englischen Sprachraum auch als „scale-up“ bzw. „scale-down“ bezeichnet wird. Dabei werden den im System vorhandenen Bestandteilen mehr Ressourcen zugewiesen (ebenda). Eine mögliche Implementierung ist das Hinzufügen weiterer central processing unit (CPU) oder Arbeitsspeicher Einheiten zu bestehenden virtuellen Servern. Diese erreichen so eine größere Leistungsfähigkeit. Dieses Vorgehen ist vergleichbar mit dem Austauschen eines Servers im eigenen Rechenzentrum, gegen ein Modell mit mehr Rechenleistung [Michael et al., 2007, S. 2 ff.]. In der Cloud kann zu diesem Zweck vereinfacht gesagt ein anderer Server mit erweiterten Hardwarespezifikationen angemietet werden. Eine zweite Möglichkeit ist das horizontale Skalieren [Luntovskyy and Spillner, 2017, S. 356], welches auch als „scale-in“ bzw. „scale-out“ bezeichnet wird. Dabei werden mehr einzelne Einheiten zum System hinzugefügt. In der Cloud wird dieses Ziel erreicht, indem eine größere Anzahl virtueller Server gemietet wird. In einem eigenen Rechenzentrum würde dies dem Kauf neuer Server entsprechen [Michael et al., 2007, S. 2 ff.]. Beiden Arten der Skalierung ist gemein, dass sich durch die Anpassung des Systems an seine Anforderungen ein individuelles Gleichgewicht zwischen Kosten und Performance des Systems erreichen lässt.

2.2 Video-Transcoding

Ein maßgeblicher Bestandteil der Arbeit ist die Beschäftigung mit Formatumwandlung von Videoinhalten, dem Video-Transcoding. Ein wesentlicher Teil dabei ist die Videokompression. Diese lässt sich wie folgt beschreiben:

> „Video compression (video coding) is the process of converting digital video into a format suitable for transmission or storage, whilst typically reducing the number of bits." [Richardson, 2010, S. 25]

Es lässt sich also sagen, dass Videokompression die Umwandlung eines Inhaltes ist, mit dem Ziel, dessen Größe zu minimieren. Nur so wird die effiziente Speicherung und Übertragung dieser sonst sehr umfangreichen Videoinhalte möglich [Janus et al., 2012, S. 300]. Dabei wird unterschieden in eine verlustbehaftete und verlustfreie Datenreduktion. Bei verlustfreien Wandlungen wird versucht, lange Symbole durch kürzere abzubilden, was zum Beispiel durch Huffman- oder Lauf-Längen-Codierung möglich wird [Li et al., 2014, S. 203 ff.]. Bei der verlustbehafteten Kompression hingegen werden gezielt Informationen, des Ausgangsmaterials ausgelassen. Ziel ist es dabei Informationen auszuwählen, deren Fehlen durch Mechanismen der menschlichen Wahrnehmung nicht aufgelöst werden können. Um dies zu erreichen, kann das Ausgangsmaterial beispielsweise mittels eines Zusammenspiels von Quantisierung und Integraltransformationen komprimiert werden ([Li et al., 2014, S. 250 ff.]. In der Praxis werden oft beide Herangehensweisen kombiniert, um ein optimales Ergebnis zu erzielen. In Erweiterung der vorangegangenen Definition lässt sich also sagen, dass es sich bei Video-Transcoding um die Anpassung eines Videofiles an die Erfordernisse dessen weiterer Verwendung handelt.

2.2.1 Wandeln des Bild-Dynamikbereiches

Eine solche mögliche Anforderung ist beispielsweise ein bestimmter Dynamikumfang der Inhalte. Neben der Auflösung und Bitrate ist für das Qualitätsempfinden in Bezug auf ein Video, auch der Dynamikumfang eine wesentliche Einflussgröße. [Melo et al., 2016, S. 16606]. So trägt die Wiedergabe von Inhalten mit einem erhöhten Dynamikumfang zum Realismusempfinden eines szenischen Inhaltes bei und kann darüber hinaus helfen, Bildinhalte auch unter schwierigen Betrachtungsbedingungen erkennbar zu halten (ebenda). Dabei bezeichnet der Begriff Dynamikumfang das Verhältnis von dunkelst möglichem Schwarz zum hellsten möglichem Glanzlicht [Schmidt, 2021a, S. 31]. Bei einem herkömmlichen Bild mit „Standard Dynamic Range" (SDR) beträgt dieser, unter günstigen Wiedergabebedingungen etwa 1:1000 (ebenda). Untersuchungen haben jedoch gezeigt, dass die Sehpräferenzen von Zuschauern deutlich über diesen Bereich hinaus gehen würden [Daly et al., 2013, S. 563–566]. So zeigte sich, dass um 90 % der Probanden zufriedenzustellen, ein Leuchtdichtenverhältnis zwischen Schwarzwert und kleinflächigen Glanzlichtern von $1{:}10^{16}$ gewünscht wäre (ebenda). Dabei wird die Leutchtdichte angegeben in:

$$1cd/m^2 = 1nit \tag{2.1}$$

[Buser and Imbert, 1992, S. 53] Wobei letztere Einheit vorrangig im angloamerikanischen Raum verwendet wird. Würde man zur Verbesserung der Bildqualität nun aber nur die Maximalhelligkeit erhöhen, die von einem Display wiedergegeben werden kann, so würde dies zu Bildartefakten führen.

So besagt das Weber-Fechnerche Gesetz, dass für jeden Helligkeitswert L gelten muss:

$$\frac{\Delta L}{L} < 0,8 \tag{2.2}$$

Dabei bezeichnet ΔL den Helligkeitsunterschied, den zwei benachbarte Digitalwerte erzeugen. Wird diese Gesetzmäßigkeit verletzt, würden die sprunghaften Helligkeitsunterschiede zu sogenannter Posterisation (auch Banding) führen, die sich in treppenhaften Helligkeitsverläufen zeigt [Schmidt, 2021a, S. 32 ff.]. Zum Einhalten dieser Maßgabe haben sich zwei Übertragungsfunktionen etabliert. Sie haben die Aufgabe, Helligkeitswerte auf ein digitales Codewort abzubilden bzw. umgekehrt. Die Hybrid-Log-Gamma Funktion, wird aus zwei Bestandteilen zusammen gesetzt. So besteht der erste Teil der Funktion aus einer Wurzelfunktion. Der zweite Teil der Kennlinie, ab 50 % des maximalen Signalwertes, bildet Helligkeitswerte hingegen logarithmisch auf Signalwerte ab (ebenda). Damit ergibt sich laut international telecommunication union (ITU)-R Recommendation BT.2100-2 [International Telecommunication Union, 2018]:

$$E' = \begin{cases} \sqrt{3E}, & 0 \leq E \leq 1/12 \\ a\log(12E-b)+c, & 1/12 \leq E \leq 1 \end{cases} \tag{2.3}$$
$$a = 0.17883277 \quad b = 1-4 \quad c = 0.5 - a\log(4a)$$

Dabei ist E das mit dem Referenz-Weißwert normalisierte Helligkeitssignal im Wertebereich von 0 bis 1 und E' ist das resultierende nicht lineare Signal. Für die Umwandlung der digitalen Werte in Helligkeitsabstufungen wird entsprechend jeweils die Inverse Funktion verwendet (ebenda). Daneben existiert der Ansatz des Perceptual Quantizer (PQ), der das Ziel verfolgt, die Übertragungsfunktion in jedem Helligkeitsbereich an das Auflösungsvermögen der menschlichen Wahrnehmung anzupassen. Dabei wurde PQ ausgehend von der Übertragungsfunktion von digitalem Codewort zu Helligkeitswert entwickelt [Schmidt, 2021a, S. 191] und lässt sich beschreiben nach (ebenda):

$$Y = L_{\text{max}} \frac{V^{\frac{i}{m}} - a}{(b-c)^{\frac{V}{m}}}^{\frac{1}{n}} \tag{2.4}$$

$$m = 78,8438 \quad n = 0,1593 \quad a = 0,8359 \quad b = 18,8516 \quad c = 18,6875$$

Dabei ist Y der resultierende Displayhelligkeitswert, V der normalisierte Signalwert und L_{max} die maximale Displayleuchtdichte. Der Bezug auf diesen Wert macht es erforderlich, dass diese Größe in den Metadaten des Videos mit übertragen werden muss.

Häufig zusammen mit dem erweiterten Dynamikumfang findet der in der Empfehlung ITU-R BT.2020 aufgenommene erweiterte Farbraum Anwendung. Dieser ermöglicht es, Farben darzustellen, die im kleineren Farbraum Rec709, wie er in den meisten Formaten mit konventionellem Dynamikbereich Anwendung findet, nicht möglich wären. Mit diesem Umstand geht jedoch auch die Herausforderung einher, dass Inhalte, die im BT.2020 Farbraum erstellt wurden, durch Umwandlungsschritte die Kompatibilität mit dem kleineren Farbraum erhalten müssen [Kutschbach, 2019, S. 36 f.]. Um die Farben, die sich außerhalb des Farbraums befinden, abzubilden, gibt es verschiedene Vorgehensweisen, die ebenfalls sinngemäß der Darstellung von [Kutschbach, 2019, S. 37 f.] entnommen sind. Die wohl einfachste Variante ist das Clipping der betreffenden Farben. Dazu wird der unzulässige

Farbwert auf jenen Farbwert innerhalb des Zielfarbraums abgebildet, der den geringsten Abstand zum Ausgangswert hat. Da es sich dabei nicht um eine eindeutige Zuordnung handelt, gehen so Informationen verloren. Eine weitere Möglichkeit stellt das sogenannte „Colourmapping“ dar. Dabei werden unzulässige Farben auf solche abgebildet, die einen ähnlichen Farb- und Helligkeitseindruck erzeugen, sich aber innerhalb des Farbraums befinden. Zu diesem Zweck gibt es zwei wesentliche Umsetzungsmöglichkeiten. Zum einen kann der neue Farbwert mithilfe einer geschlossenen Funktion aus dem Eingangsfarbwert berechnet werden. Dieses Vorgehen setzt aber voraus, dass die Funktion hinreichend simpel sein muss, um unter den gegebenen zeitlichen Anforderungen berechenbar zu sein (ebenda). Ist dies nicht der Fall, kann der Zielwert zum anderen mithilfe eines „Lookup-Tables“ berechnet werden. Dabei handelt es sich um eine dreidimensionale Matrix, in der vorab berechnete Werte eingetragen werden [Ai et al., 2014]. So kann in der Matrix eine Kombination der Werte nachgeschlagen werden, die den Farbraum aufspannen. Auf diese Weise werden komplexere Berechnungen im Moment der Ausführung umgangen. Um die Größe der Matrix zu verringern, werden in der Praxis oft mehrere Wertekombinationen durch ein Ergebnis abgebildet [Kutschbach, 2019, S. 39 ff.]. Damit können in erhöhter Verarbeitungsgeschwindigkeit Farbraumwandlungen umgesetzt werden. Aufgrund der steigenden Relevanz der Wandlung von „high dynamic range“ (HDR) zu „standard dynamic range“ (SDR) Inhalten wird sie im Folgenden als funktionale Anforderung an die Infrastruktur mit aufgegriffen.

2.2.2 Metriken zu Beurteilung der Bildqualität

Um im weiteren Verlauf der Arbeit eine objektive Beurteilung von Transcoding-Ergebnissen vornehmen zu können, sind Metriken zu Beurteilung der Bildqualität unumgänglich. Mithilfe derartiger Größen können verschiedene Ergebnisse miteinander verglichen werden. Neben derart objektiven Beurteilungsmetriken gibt es weiterhin auch subjektive Messgrößen, deren Erstellung sich maßgeblich auf die Beurteilung der Bildqualität durch Probanden stützt. Da diese Art der Qualitätsbeurteilung allerdings sehr zeit- und kostenintensiv ist [video Clarity, Inc., 2016], wird im Folgenden nur auf die mathematische Qualitätsbeurteilung eingegangen. Deren grundlegende Idee ist es, aus einem Ursprungssignal und dem gewandelten Signal eine Differenz zu berechnen, die sich dann als Fehler der Umwandlung interpretieren lässt [Schmidt, 2021b, S. 266 ff.].

PSNR Eine der dabei am häufigsten verwendeten Metriken ist die PSNR [video Clarity, Inc., 2016]. Diese wird aus der mittleren quadratischen Abweichung, englisch „Mean Squared Error“ berechnet. Diese lässt sich je Farbkanal für eine Bild aus N Pixeln bestimmen zu:

$$MSE = \frac{1}{N} \sum_{n=1}^{N} (x_{\mathsf{n}} - y_{\mathsf{n}})^2 \tag{2.5}$$

Dabei ist x_{n} der Ausgangswert des n-ten Pixels und $\mathsf{y_n}$ der durch die Bearbeitung veränderte Wert. Anknüpfend an diese Berechnung kann die PSNR berechnet werden durch:

$$PSNR = 10 \log(\frac{Q^2}{MSE})[dB] \tag{2.6}$$

[Schmidt, 2021b, S. 267 ff.] Dabei bezeichnet Q den maximal möglichen Signalwert, der für ein Acht Bit Video beispielsweise 220 betragen würde (ebenda). Damit lässt sich die Größe als Signalrauschabstand in dB, wie er auch für andere Sachverhalte in der Übertragungstechnik angegeben wird interpretieren[Fischer, 2016, S. 366]. Typische Werte, für die PSNR eines Acht Bit Videos be-

tragen etwa 30 bis 40 db. Dabei ist zu betrachten, dass durch die logarithmische Berechnung der Größe, eine Veränderung der PSNR um 3dB bereits einer Verdopplung der Fehlerleistung entspricht [Schmidt, 2021b, S. 267 ff.].

Allerdings ist es ohnehin wichtig, die Aussagekraft von PSNR Werten kritisch einzuordnen. So führt die Art der Berechnung dazu, dass der subjektive Seheindruck und der berechnete Fehler zuweilen stark voneinander abweichen [Wang et al., 2004, S. 1]. Das rührt daher, dass hier pixel- und frameweise das Ausgangssignal mit gewandelten Signal verglichen wird. Kommt es nun beispielsweise durch einen Bearbeitungsschritt zu einem vertikalen oder zeitlichem Versatz zwischen Quell- und Vergleichsdatei, so würde dies zu miserablen Ergebnissen führen [Ohm, 2015, S. 172]. Die wahrgenommene Qualität des Videos würde hingegen darunter nur unwesentlich leiden. Selbes würde für großflächige, aber unwesentliche Helligkeitsunterschiede gelten. So hätte diese Veränderung größere Auswirkungen auf den PSNR Wert des Vergleichs, da für jedes Pixel ein relevanter Fehler zu verzeichnen wäre. Bei einem subjektiven Verfahren hingegen würde auch dieser Fehler deutlich weniger ins Gewicht fallen, da die menschliche Wahrnehmung derartige, gleichmäßige Veränderungen als wenig störend empfindet.

Zusammenfassend lässt sich sagen, dass die PSNR Metrik keine Bewertung vornimmt, wie störend eine Bildveränderung von Rezipienten wahrgenommen werden würde (ebenda). Die großen Stärken der Kenngröße liegen hingegen in ihrer einfachen Berechnung und klaren physikalischen Bedeutung der ermittelten Werte [Wang et al., 2004, S. 1]. Damit eignet sich die Metrik vor allem für den Qualitätsvergleich zwischen Codecs bei Verwendung derselben Bildfolge [Schmidt, 2021b, S. 267]. Da bei diesem Vorgehen gleichartige Fehler zu erwarten sind, fallen die negativen Eigenschaften der Metrik dabei weniger ins Gewicht.

SSIM Neben der angeführten Metrik gibt es jedoch auch Verfahren, die für eine verbesserte Korrelation zwischen objektiven Messwerten und subjektiver Bewertung sorgen. Eines davon ist das „structural similarity“ (SSIM). Bei diesem werden örtliche Pixelmuster zwischen der Ursprungsdatei und ihrer veränderten Version verglichen [Wang et al., 2004, S. 600-612]. Dazu werden zuerst beide Vergleichsdateien auf eine gemeinsame Auflösung skaliert. Der Inhalt dessen wird im Anschluss in Kontrast und Helligkeit normalisiert [Wang et al., 2004, S. 600-612]. Anschließend wird jeweils der Inhalt eines kleinen Pixelfensters verglichen [video Clarity, Inc., annt]. Dazu wird für jeden der Pixelausschnitte Original- und Ausgabe-Datei gegenübergestellt. Auf diese Weise fallen Helligkeitsunterschiede oder Kontraständerungen weniger ins Gewicht (ebenda). Daneben werden auch die Kontrast- und Helligkeitsveränderungen der Pixelraster in Betracht gezogen. Da sie von der menschlichen Wahrnehmung als deutlich weniger störend empfunden werden, werde diese mit einem niedrigeren Faktor in der Ähnlichkeitsberechnung berücksichtigt als strukturelle Veränderungen [Rao et al., 2014, S. 273-275]. So werden die drei Vergleichsgrößen unabhängig voneinander betrachtet und ihre Ähnlichkeit zum Ausgangsbild berechnet. Der Gesamtwert für ein Bild wird anschließend durch Mittelung aus diesen Werten berechnet [video Clarity, Inc., annt]. Dabei steht der Wert eins für eine besonders hohe Qualität, bei der Original und Ausgabedatei deckungsgleich sind. Allgemein spricht man ab einem Wert von etwa 0,99 davon, dass die beiden Dateien visuell nicht mehr unterscheidbar sind [Flynn et al., 2013, S. 27 f.]. Dieses Vorgehen macht die Metrik deutlich robuster gegen großflächige Helligkeits- oder Kontrastveränderungen (ebenda).

Zusammenfassung Um die Unterschiede zwischen den beiden erläuterten Metriken zu illustrieren, können die Bilder unter Abbildung A herangezogen werden. Es ist offensichtlich, dass diese in Bezug auf die menschliche Wahrnehmung von signifikant unterschiedlicher Qualität sind. Dennoch weisen sie nahezu gleiche MSE und damit auch PSNR Werte, in Bezug auf das Originalbild (a), auf. Ihre SSIM Werte hingegen differieren zum Teil stark, wie die Tabelle 2.1 zeigt.

Teilbild	Modifikation	MSE	SSIM
b	Helligkeitsänderung	144	0,988
c	Kontrastveränderung	144	0,913
d	Impuls Rauschen	144	0,840
e	Weichzeichnung	144	0,694
f	Kompressionsartefakte	142	0,662

Tabelle 2.1: Vergleich MSE und SSIM der Bildveränderungen in Abbildung A [Rao et al., 2014, S. 275]

Damit lässt sich zusammenfassend sagen, dass die Aussagekraft von SSIM gerade in Hinblick auf verschiedenartige Fehler, größer ist, als die von PSNR. Die Metrik hat ihre Berechtigung vor allem im Vergleich von Codequalitäten. Da die zu erwartenden Fehler entlang einer Messreihe hier gleichartig sind, ist die Aussagekraft auch für PSNR Werte durchaus gegeben. Hinzu kommt die leichte physikalische Interpretierbarkeit des Signalrauschabstands. Weiterhin begünstigt der geringe mathematische Aufwand die Implementierung dieser Messgröße.

2.3 Open-Source

Nicht zufällig beschäftigt sich die Arbeit mit einer Infrastruktur, die auf Open-Source Technologien aufbaut. Diese Art der Softwarelizenzierung spielt eine tragende Rolle in der Erstellung und Gestaltung von Softwareprojekten. So setzten schon 2021 laut einer Studie des IT-Branchenverbandes BITKOM 70 % der befragten Unternehmen mit mehr als 20 Mitarbeitern Open-Source-Software in Ihrem Unternehmen ein [Gentemann et al., 2021, [S. 21].

2.3.1 Definition des Open-Source-Begriffs

Eine mögliche Definition des Open-Source Begriffs stammt von der Open Source Initiative. Dabei handelt es sich um eine gemeinnützige Organisation, deren Ziel die Förderung und Standardisierung des Open-Source-Gedankens ist [Initiative, 2022a]. Sie spezifiziert Open Source anhand von zehn Punkten, die im Folgenden zusammen gefasst werden.

Ein Open-Source-Programm zeichnet sich dadurch aus, dass es möglich ist, den Quellcode einzusehen, und dieser in einer für Menschen les- und nachvollziehbaren Art und Weise aufbereitet ist. Damit ein Programm als Open-Source bezeichnet werden kann, muss seine Lizenz weiterhin die Veränderung der Software erlauben. Damit ist zum einen die Möglichkeit zur aktiven Mitarbeit am Softwareprojekt, zum anderen aber auch die Möglichkeit der Einbindung in ein eigenes Softwareprojekt gemeint. So muss es auch möglich sein, Teile oder den gesamten Quellcode in eigener Software zu verwenden, auch mit dem Ziel, diese zu verkaufen. Weiterhin muss es möglich sein, die Software auch auszuführen, ohne, dass dafür ein zusätzlicher Lizenzschlüssel notwendig wird. Eine weitere Forderung ist es, dass die Software technologieneutral zu halten ist. Damit ist gemeint, dass es keiner spezifischen Hardware, Schnittstelle oder Technologie bedarf, um die Software zu verwenden [Initiative, 2022b].

Open-Source Lizenzen Neben dieser allgemeinen Definition gibt es verschiedene Lizenzmodelle, zur konkreten Ausgestaltung dieser Forderungen. Obschon sie sich alle unter dem Begriff Open-Source sammeln, unterscheidet sich ihr Inhalt deutlich. Das wohl am häufigsten verwendete Lizenzmodell ist die „MIT-License“ [Rathee and Chobe, 2022, S. 46]. Sie räumt das Recht zum Kopieren, Nutzen und Anpassen der Software ein. Dabei wird auch das Recht zur kommerziellen Weiterverwendung eingeräumt. Die einzige Restriktion, die vorgenommen wird ist, dass die Software wiederum mit einem Copyright Hinweis versehen werden muss. Dieser muss auch den Hinweis enthalten, dass vom Ersteller der Software keine Haftung für die Verwendung seiner Software übernommen wird.

Ähnlich verhält sich auch die „Berkeley Software Distribution License“. In dieser werden jedoch mehr Vorgaben gemacht zu Copyright, Anmerkungen und Disclaimern, die bei einer Weiterverwendung der Software nötig werden. Ein Beispiel von auf diese Art lizenzierter Software ist der Nginx Proxy (ebenda), der auch im weiteren Verlauf der Arbeit Anwendung findet.

Anders gestaltet ist hingegen die „GNU General Public License“. Hier ist die Rede von einer reziproken Lizenz. Damit ist gemeint, dass alle Rechte, die von der Lizenz eingeräumt werden, auch von jeder Software, die Teile der Ausgangssoftware beinhaltet, gewährleistet werden müssen. Wird also so lizenzierter Programmcode in einer Anwendung angewendet, ist es untersagt, diesen nicht öffentlich zugänglich zu machen. Diese Eigenschaft wird auch als „Copyleft“ bezeichnet [Free Software Foundation, Inc., 2007]. Damit ist zwar das kommerzielle Ausführen des lizensierten Programms gestattet, nicht aber der Weitervertrieb, einer Software, die Teiles des Ursprungsprogramms verwendet, ohne den gesamten Quellcode offenzulegen (ebenda). Ein Beispiel für auf diese Weise lizenzierte Software ist der Linuxkernel, wie er einer Vielzahl von Betriebssystemen als Vorlage dient [Rathee and Chobe, 2022, S. 45].

Alle Open-Source-Lizenzen eint eine ganze Reihe von Vorteilen gegenüber anderen Lizenztypen. So ist ein Open-Source-Projekt unabhängig von einzelnen Firmen oder Entwicklern. Wird eine Änderung oder das Beheben eines Fehlers von einem Nutzer gewünscht, so kann dies selbständig oder durch Dritte passieren [Cassia, 2007]. Bei der Software eines kommerziellen Anbieters hingegen müssten Anwender auf dessen Unterstützung hoffen. Läuft diese Unterstützung aus, da keine Wartung oder Weiterentwicklung mehr angeboten wird, erschwert dies den weiteren Betrieb der Software sehr (ebenda). Ebenfalls ins Gewicht fällt, dass Open-Source-Software für ein hohes Maß an Transparenz sorgt. So sorgt der Umstand, dass der Quellcode von jedem eingesehen werden kann dafür, dass jeder sich von der Qualität des Quellcodes überzeugen kann. Auf diese Weise können auch bekannte Sicherheitslücken erkannt und beseitigt werden [Kashiwa et al., 2019, S. 12]. Dabei ist diese Eigenschaft vor dem Hintergrund von folgenschweren Sicherheitslücken immer wieder Gegenstand von Untersuchungen und Diskussionen [Rogowsky, 2014, S. 59 ff.]. So kommt [Rogowsky, 2014, S. 165] zu dem Schluss, dass das Verwenden von Open-Source Bestandteilen allein weder ein unkalkulierbares Risiko noch ein Garant für sichere Software sind. Vielmehr seien die Überprüfungsmechanismen, die zur Qualitätssicherung angewendet werden, entscheidend.

Zusammenfassend lässt sich sagen, dass es gerade die Aspekte der maximalen Transparenz und verbesserten Nachvollziehbarkeit sind, die OpenSource-Lösungen im Rahmen der vorliegenden Arbeit von besonderem Interesse erscheinen lassen.

2.3.2 FFmpeg als Beispiel für Open-Source Software

Im Zuge der weiteren Arbeit wird besonders eine quelloffene Anwendung eine tragende Rolle einnehmen. Die Rede ist von FFmpeg, einer plattformunabhängigen Open-Source-Software. Der Kernbestandteil des Projektes ermöglicht es, über ein Commandline-Interface Audio und Video (AV)-Formate zu wandeln [FFmpeg.org, 2022a]. Daneben besteht die Software aus dem AV-Player FFplay und dem AV-Analysewerkzeug FFprobe (ebenda). Gestartet ist das Projekt im Jahr 2000 und weißt heute mehr als 109.000 Änderungen von mehr als 2000 Mitbearbeitern auf [Synopsys, Inc, 2022]. Die hauptsächlich in C programmierte Anwendung erfreut sich heute großer Beliebtheit (ebenda). So greifen bekannte Softwares, wie der verbreitete VLC Player, auf FFmpeg zurück (ebenda). Mit einigen Ausnahmen, die jedoch manuell aktiviert werden müssen, fällt die Software unter die „GNU Lesser General Public License“ in der Version 2.1+ [FFmpeg Community, 2022]. Anders als die im vorangegangenen beschriebene „GNU GPL“ erlaubt diese unter bestimmten Bedingungen auch die Verwendung der Bibliothek, in Anwendungen, die anschließend nicht öffentlich einsehbar sind [Free Software Foundation, Inc., 2022].

Für die Software, die im Zuge der Arbeit erstellt wird, lässt sich sagen, dass es sich im Sinne der Lizenz um eine von der Bibliothek abgeleitete Software handelt [Free Software Foundation, Inc., 2000]. So ist sie zwar auf die Funktion mit FFmpeg ausgelegt, enthält diese aber nicht direkt, womit die erstellte Software nicht von der FFmpeg Lizenz berührt wird. In der Software-Architektur, die im Zuge der Arbeit erstellt wurde, fungiert FFmpeg als zentrale Arbeitseinheit. Die Software stellt dazu eine Vielzahl von AV-Encodern zur Verfügung. Auch Funktionen für bildverändernde Effekte bietet FFmpeg an, die auf diese Weise leicht genutzt werden können. Mithilfe einer Konsoleneingabe kann es dazu aufgerufen werden und die Datei, die bearbeitet werden soll, wird als Parameter übergeben. Auch Codecs und deren Einzelheiten, wie Bitrate und Codec-Profil können so ausgewählt werden. Weitere Parameter können für die notwendigen Videofilter übergeben werden. Kommen dazu mehrere Filter zum Einsatz, so funktionieren diese in der Art einer Kette. Die Ausgabe eines ersten Filters dient dem folgenden Effekt als Eingabedatei, womit sich komplexe Bearbeitungsketten zusammen stellen lassen.

Einzig die Bedienung der Software kann zuweilen als umständlich empfunden werden, da es nötig ist, für jede Funktion die nötigen Parameter und deren zulässige Werte zu kennen. Ein Umstand, der im weiteren Verlauf der Arbeit mit der Erstellung eines Nutzerinterfaces adressiert wird.

3 Definition von Anforderungen an die Infrastruktur

Zur Beantwortung der Forschungsfrage wird, als wesentlicher Bestandteil der Abhandlung, eine Softwarelösung entwickelt. An diese wird eine Reihe von Anforderungen formuliert, die von der erarbeiteten Lösung erfüllt werden müssen. In Messungen und ergänzenden Betrachtungen werden diese schließlich auf nachvollziehbare Weise überprüft. Dies geschieht abschließend in Messungen und deren Auswertung. So können Eigenschaften nachgewiesen werden, die Antwort auf die Forschungsfrage geben.

3.1 Funktionale Anforderungen

Zuerst werden dazu Anforderungen definiert, die sich unmittelbar auf den Funktionsumfang der Anwendung beziehen. Dies betrifft zuallererst die Kernfunktion der Softwarelösung, das Wandeln von Videodateien mithilfe verschiedener Codecs. Damit die Praxisrelevanz des Systems als gegeben gesehen werden kann, müssen sowohl Produktionscodecs als auch solche, die für das Ausspiel an Endnutzer relevant sind, ineinander umgewandelt werden können. Dabei sind mit Produktionscodecs jene, auch als „Intermediate-Standards“ bezeichnete, Formate gemeint, die in der Mitte einer Produktionskette Anwendung finden [Schmidt, 2021b, S. 270]. Aus der Auswahl dieser sind mindestens die beiden Hauptvertreter Apple ProRes und DNxHD/HR (ebenda) zu implementieren. Bei der Auswahl der relevanten Konsumenten-Codecs wurde sich dabei vorrangig an deren Relevanz für die Nutzung im Web orientiert. Dazu zählt H.264 als Konsumentenformat für die kompatible Auslieferung von HD Inhalten [Richardson, 2010, S. 4 f.]. Weiterhin soll der H.265/HEVC Kompressionsalgorithmus unterstützt werden. Als Weiterentwicklung des H.264 Codecs ist dieser in einer Vielzahl von Anwendungsfällen stark verbreitet. Damit liegt es auf der Hand, diese Anforderung zu definieren [Schmidt, 2021b, S. 270]. Als Alternative dazu soll auch der VP9 Codec unterstützt werden. Dabei handelt es sich um ein Format, das von der „Aliance for Open Media“ entwickelt wurde (ebenda) und seine Relevanz vor allem aus der Anwendung auf der Plattform YouTube bezieht [Iza Paredes et al., 2017, S. 52]. Weiterhin sollen die transcodierten Videoinhalte in eine Auswahl relevanter Containerformate gepackt werden können. Bei der Auswahl dieser wurde sich auf eine Zusammenstellung aus [Adobe Inc., 2022], und Teile der Auflistung von [Maayan, 2020] gestützt. Als Ergebnis sollen, für eine praktische Einsetzbarkeit der Infrastruktur, mindestens folgende Container anwendbar sein:

- mp4
- mov
- mkv
- mxf

Eine weitere grundlegende funktionale Anforderung ist, dass die Auflösung der Inhalte mithilfe der Anwendung skaliert werden kann. Wie im Theorieteil bereits angeklungen, stellt dies eine wesentliche Anforderung dar, da die Varianz der Displayauflösungen, auf denen Inhalte konsumiert werden, stetig zunimmt. Wie zuvor schon beschrieben, stellt durch die zunehmende Bedeutung von HDR Inhalten, auch deren Konvertierung eine zunehmend relevante Anforderung an einen Videoencoder

dar. Aus diesem Grund soll die Anwendung auch das Wandeln von HDR in SDR Inhalte ermöglichen. Ziel ist es also in HDR produzierte Inhalte in den SDR-Dymanikbereich zu wandeln. Dabei soll es dem Benutzer möglich sein, für die Wandlung weitere Parameter anpassen zu können. So soll spezifiziert werden können, mit welchem Algorithmus Farben, die sich außerhalb des Farbraumes befinden, in den neuen Farbraum abgebildet werden. Dabei sollen die Testreihen zum Prüfen dieser Eigenschaft auf eine nähere Auswertung des HDR Materials verzichten, damit alle Testreihen ohne gesonderte Hardware, wie beispielsweise ein HDR-fähiges Display, nachvollzogen werden können. Mit der Möglichkeit der HDR zu SDR Wandlung soll sichergestellt werden, dass die Relevanz der Anwendung auch perspektivisch gegeben bleibt. Eine weitere Anforderung, die von der Anwendung zu erfüllen ist, stellt die Qualität der Wandlungsergebnisse dar. Damit der sinnhafte Einsatz der Anwendung möglich ist, soll die Anwendung bei gleichen Eingabeparametern eine vergleichbare Qualität des Ausgabematerials liefern, wie es eine lokal ausgeführte Transcoding-Software täte. Dies soll sich im Vergleich des erstellten Cloud-Encoders mit einer kommerziellen Video-Transcoding-Software, anhand eines „Full-Reference“ Verfahrens nachweisen lassen.

Diese rein funktionalen Anforderungen dienen dazu, grundlegend die Anwendbarkeit und Vergleichbarkeit der Infrastruktur mit lokal ausgeführten Anwendungen sicherzustellen. Daneben wird die Anforderung an die Infrastruktur gestellt, dass Ihr Aufbau maßgeblich aus Open Source-Komponenten besteht. Konkret wird diese Anforderung auf alle Softwarebestandteile bezogen, die von Dritten bezogen wurden und im Cloud-Encoder Anwendung finden. An diesen Stellen sollen ausschließlich Programme verwendet werden, die unter einer Lizenz veröffentlicht wurden, die der Definition in Abschnitt 2.3 genügen. Davon ausgenommen sind lediglich jene Ressourcen, die vom Cloudanbieter als Serverinfrastruktur bezogen werden. Durch den quelloffenenen Aufbau soll die Wiederholbarkeit des Aufbaus gewährleistet werden. So kann zu jeder Zeit die Erstellung der Infrastruktur nachvollzogen werden, ohne, dass dafür Lizenzen erworben werden müssen. Sind unter diesen Gesichtspunkten wesentliche Eigenschaften der Infrastruktur sichergestellt, können spezielle Eigenschaften herausgearbeitet werden.

3.2 Cloudspezifische Anforderungen

Einen engeren Bezug zur Forschungsfrage haben jene Anforderungen, die sich auf Besonderheiten der Cloud beziehen. So geht es dabei darum, Eigenschaften zu formulieren, die in der Entwicklung der Infrastruktur angestrebt werden sollen. Ziel ist es so Vorteile aufzudecken, die in einer On-Premises Umsetzung nicht oder nur schwer möglich wären. Das ist zum Teil auch dem Umstand geschuldet, dass sich in der herkömmlichen Bereitstellung Zielkonflikte zwischen den Dimensionen des Angestrebten ergeben. So ist es beispielsweise leicht möglich, auch im eigenem Rechenzentrum Kapazitäten für eine Skalierung vorzuhalten. Allerdings gehen damit hohe Kosten einher, was die Erfüllung des Ziels der Kosteneffizienz gefährdet. Um zu analysieren, ob die Dimensionen von der umgesetzten Infrastruktur erreicht werden, sollen diese nun präzisiert werden.

Skalierbarkeit So ist ein wesentliches Versprechen, dass die Cloud macht, die Skalierbarkeit von Anwendungen. Hintergrund ist dabei, dass bei einem höheren Aufkommen an Anfragen, die Softwarelösung daran angepasst werden kann. Für den Cloud-Encoder bedeutet das, dass durch ein Zuweisen neuer Rechenkapazitäten die Zeit, die benötigt wird, um eine bestimmte Transcoding-Aufgabe zu erfüllen, minimiert werden kann. Die erarbeitete Cloud-Lösung soll also beide unter

Unterabschnitt 2.1.2 beschriebenen Arten der Skalierung ermöglichen. Dabei soll die vertikale Skalierung die Durchlaufzeit einer Datei durch den Cloud-Encoder signifikant verkürzen. Als signifikant soll dabei eine Herabsetzung der Durchlaufzeit um mindestens 30% Prozent betrachtet werden. Diese Reduktion soll sich bereits ab einer Verdopplung der Server-Kosten einstellen. Diese Aufwendungen beziehen sich dabei nur auf die anfallenden Mietkosten, die durch die Verwendung des jeweiligen Servers mit mehr Hardwareressourcen anfallen. Die verstrichene Durchlaufzeit soll in beiden Fällen vom Versenden der ersten relevanten Anfrage bis zum Eintreffen der Erfolgsmeldung der letzten Wandlung gemessen werden.

Relevant ist das besonders, da für die horizontale Skalierung die Durchlaufzeit einer Reihe von Anfragen gemessen werden soll. So soll die horizontale Skalierung der Infrastruktur dafür sorgen, dass die Zeit, die eine Menge m von Anfragen benötigt, verringert wird. Als signifikant soll im Zuge der Betrachtung eine Veränderung der Gesamtdurchlaufzeit T um t_1 angesehen werden. Dabei ist t_1 die Zeit, die eine Anfrage unter Verwendung eines Servers benötigt. Diese Veränderung soll sich jeweils bereits durch das Hinzufügen eines Servers beobachten lassen. Weiterhin soll auch die Gültigkeitsbedingung dieser Zielsetzung untersucht werden. Es soll also betrachtet werden, ob diese Forderung auch für größere Serverzahlen oder ein hohes Aufkommen an Anfragen haltbar ist.

Als Rahmenbedingungen für die Skalierungsversuche soll formuliert werden, dass alle gestellten Anfragen je Versuch gleichartig sein sollen, um die Vergleichbarkeit sicherzustellen.

Hochverfügbarkeit Eine weitere Anforderung an das System ist dessen Hochverfügbarkeit. So muss das System über eine Testperiode von 48 Stunden mit einer Verfügbarkeit von mindestens 99,99 % erreichbar sein. Die Erreichbarkeit wird dabei definiert durch die Bereitschaft des Systems Anfragen, die mittels Hypertext Transfer Protocol (HTTP) gestellt werden, zu beantworten. Wird die gestellte Anfrage also mit einer positiven Antwort erwidert, so soll davon ausgegangen werden, dass sich das System in einem betriebsbereiten Zustand befindet. Als positiv sollen dabei Antworten mit einem HTTP-Statuscode zwischen 200 und dem theoretischen Wert 299 betrachtet werden. Die Antwort muss für jede Anfrage maximal einer Sekunde nach Versenden dieser vorliegen. Für den Test soll das System in der kleinsten Konfiguration, die noch alle Redundanzfunktionen zulässt, ausgerollt werden. Auf diese Weise wird geprüft, ob die Infrastruktur über längere Zeit hinweg zuverlässig arbeitet. Weiterhin sollen scheinbar zufällig auftretende Fehler, welche die Verwendbarkeit beeinträchtigen, ausgeschlossen werden. Nur so kann das Zusammenspiel der Komponenten auch für den Dauerbetrieb als geeignet betrachtet werden.

Ein weiterer Unteraspekt der Systemverfügbarkeit ist dessen Erreichbarkeit im Falle eines Fehlers. Dabei meint der Begriff Fehlertoleranz im Zusammenhang mit einem Rechnersystem, dass bei einem Ausfall einer Recheneinheit die Funktion des Gesamtsystems erhalten bleibt [Bengel, 2014, S. 6]. Ziel ist es also, Teile des Systems so aufzubauen, dass die Gefahr von Serviceausfällen minimiert wird [Pattanayak et al., 2021, S. 14]. Um dieses Ziel zu erreichen, müssen Redundanzen geschaffen werden, die im Fehlerfall den Systembetrieb sicherstellen. Für den Cloud-Encoder bedeutet das, Anfragen können weiterhin beantwortet werden, nachdem ein Server, der für die Wandlung der Videodateien zuständig ist, ausgefallen ist. Das heißt für den Test, dass nach Abschalten einer Rechnereinheit, die zum Wandeln von Dateien verwendet wird, geprüft werden soll, ob die geschaffenen Redundanzen dafür sorgen, dass das System weiter funktionsfähig bleibt. Als funktionsfähig soll das System betrachtet werden, wenn weiterhin erfolgreich Wandlungsanfragen vom System bearbeitet werden können

Kosteneffizienz Eine letzte zentrale Anforderung, die eng mit den beiden vorangegangenen Eigenschaften verknüpft ist, ist die Kosteneffizienz. Es wäre schließlich durchaus denkbar, eine hohe Verfügbarkeit oder die Skalierung eines Systems in traditionellen IT-Systemen zu erreichen. Nur geht damit oft ein hoher Aufwand vonseiten des Betreibers des Systems einher. Daher muss es auch eine Anforderung sein, dass das System, sowohl die beiden vorangegangenen Anforderungen erfüllt, als auch finanziell effizient betrieben werden kann. Als kosteneffizient wird das Cloud-System im Zuge der Betrachtung gesehen, wenn es Kostenvorteile im Vergleich zu einem „On-Premises“-betriebenem System bietet.

Auf Seiten der „On-Premises“-Anwendung wird dabei ein Vergleichssystem herangezogen, das in Hinblick auf Videotranscoding vergleichbare Ergebnisse liefern könnte. Es soll dazu von einem geeignetem Desktop- oder Workstation-System ausgegangen werden. Für den Cloud-Encoder soll eine Konfiguration betrachtet werden, die zum einen alle zuvor getesteten Redundanzfunktionen bietet. Zum anderen sollen in Bezug auf die Servergrößen solche gewählt werden, die sich bei gleichzeitiger Betrachtung von Durchlaufzeiten und der damit verbundenen Kosten als besonders effizient gezeigt haben. Eine entsprechende Konfiguration muss zuvor durch den Vergleich verschiedener Servergrößen ermittelt werden. Da kein konkretes Nutzungsprofil vorliegt, soll für beide Systeme von einer Dauernutzung ausgegangen werden. Dabei werden in die Betrachtungen alle wesentlichen Betriebs- und Anschaffungskosten, die bei beiden Systemen anfallen würden, beachtet. Für die Anschaffung des inhouse Systems werden dabei die Kosten, die nach einer Abschreibungsrechnung auf den jeweiligen Betriebszeitraum entfallen würden, wertmindernd berücksichtigt. Durch Gegenüberstellung der Kosten des Cloud-Encoders und einer lokal betriebenen Anwendung wird eine Aussage möglich, ob der Betrieb des Cloud-Encoders auch unter finanziellen Gesichtspunkten als attraktiv gesehen werden kann.

4 Beschreibung der umgesetzten Infrastruktur

Um die zuvor definierten Anforderungen überprüfen zu können, wurde ein Softwareprototyp erstellt. Dessen Funktionsweise wird in den folgenden Kapiteln näher erklärt. Dazu wurde eine Unterteilung in die beiden grundlegenden Bestandteile des Programms vorgenommen. So wird zuerst auf jene Teile der Lösung eingegangen, die zur Geschäftslogik und Nutzerinteraktion zählen. Im darauffolgenden Abschnitt werden solche Bestandteile beschrieben, die für das Erstellen und Konfigurieren der Netzwerk- und Laufzeitumgebung zuständig sind. Der Quellcode für die Software lässt sich, zusätzlich zu den Anlagen, in den Git Repositorys zu dieser Arbeit einsehen. So kann die Webanwendung unter diesem Link[1] eingesehen werden. Der Code zur Erstellung der Cloud-Infrastruktur kann unter folgendem Link[2] erreicht werden.

4.1 Nutzerinterface und Geschäftslogik

Ein wesentlicher Ansatz bei der Planung der Softwarearchitektur war eine strikte Trennung zwischen Nutzeroberfläche und Video-Transcoding-Einheit. Auf diese Weise wird eine unabhängige Skalierung der beiden Bestandteile ermöglicht. Diese Idee wurde unter anderem durch das Aufsetzten zweier unabhängiger Webserver umgesetzt. In der weiteren Beschreibung wird zur Vereinfachung jener Server, dessen Aufgabe unter anderem ist, Benutzereingaben entgegenzunehmen, als „Frontend“ bezeichnet. Jener Bestandteil, der das Video-Transcoding unmittelbar steuert und schlussendlich durchführt, wird als „Backend“ bezeichet.

4.1.1 Node.js als Anwendungsplattform

Konkret handelt es sich bei den Servern um zwei Node.js Anwendungen. Dies ist ein Framework, um Javascript-Code zum Erstellen von Webserver-Backends zu verwenden [Zammetti, 2013, S. 120 f.]. Die Lösung zeichnet sich durch einen guten Kompromiss aus Ausführungsgeschwindigkeit und einfacher Handhabbarkeit aus (ebenda). Von Vorteil ist weiterhin das Verwenden der gleichen Programmiersprache im Front- und Backend der Webanwendung. Auf diese Weise konnten Recherchezeiten für unbekannte Syntaxbestandteile minimiert werden. Ein weiterer Aspekt, der sich als vorteilhaft erwiesen hat, ist die komfortable Umsetzung von asynchronen Funktionsaufrufen durch JavaScript. So lässt sich eine Vielzahl von umfangreicheren Prozessen umsetzen, ohne dass die Anwendung für weitere Anfragen blockiert wird [Zammetti, 2020, S. 128 ff.]. Gerade bei Prozessen mit erhöhter Laufzeit wie Videodecodierung erwies sich dies als erfolgskritisch.
Die asynchrone Ausführungsweise ist für das Framework zwingend, auch weil es im Einzel-Thread-Betrieb arbeitet [Ihrig, 2013, S. 28]. Das bedeutet, dass nebenläufiges Abarbeiten von verschiedenen Anfragen nicht vorgesehen ist. Anstatt, wie „Apache HTTP Server“, eine Reihe von Threads zu erstellen, die nebeneinander ausgeführt werden, handelt Node.js alle Anfragen nacheinander ab (ebenda). Entgegen der intuitiven Annahme, hat dieses Vorgehen durchaus Skalierungsvorteile, da kein Overhead der einzelnen Prozesse entsteht (ebenda). Es liegt jedoch auch auf der Hand, dass es problematische Folgen hat, diesen Thread durch Ein- oder Ausgaben zu blockieren. Ein einzelner

[1] https://github.com/paulpfleg/deploy
[2] https://github.com/paulpfleg/aws_terraform_for_ba

synchroner Aufruf könnte so die Reaktionsfähigkeit der Anwendung für alle Nutzer beeinträchtigen. Um dies zu verhindern, werden die asynchronen Aufrufe mithilfe des sogenannten Eventloops abgearbeitet [Shute, 2019, S. 82]. Dabei wird ein asynchroner Prozess nur gestartet und beim nächsten Durchlauf der Ausführungsschleife dessen Erfüllung überprüft. Damit ist es für die umgesetzte Lösung möglich, auch bei einer Vielzahl bestehender Anfragen, noch weitere entgegenzunehmen.

4.1.2 Aufbau der API und Schnittstellengestaltung

Um eine lose Kopplung zwischen Front- und Backend zu erreichen, wurde sich für deren Interaktion über ein Application Programming Interface (API) entschieden. Hinter dieser Bezeichnung verbirgt sich grundlegend eine Schnittstelle, die Teile einer Anwendung für andere Rechner zugreifbar macht [De, 2017, S. 1]. Konkret bedeutet das, dass nicht alle Funktionen, die einem Programm zur Verfügung stehen sollen, auch auf dem gleichen Rechner ausgeführt werden müssen. Vielmehr können Anwendungen verteilt über mehrere Rechner aufgebaut werden. Die Informationen, zum Aufruf der Funktionen, beziehungsweise die Ergebnisse des Aufrufs werden über ein Rechnernetz ausgetauscht (ebenda).

Eine erste Inspiration für das verwendete Vorgehen bildete [Salkosuo, 2020]. In seinem Quellcode implementierte der Autor GET- und POST-Endpoints, mit denen FFMPEG Wandlungen für Dateien, die auf einem lokalen Rechner abliegen, ausgelöst werden können. So konnten im Vorfeld der Arbeit erste Versuche mit dieser Software vorgenommen werden. Dabei zeigte sich das Ansprechen von FFmpeg als belastbarer Ansatz. In ganz wesentlichen Punkten geht die Quelle jedoch ganz andere Wege, als die umgesetzte Lösung. So verwendet die Quelle das Modul „fluent-ffmpeg“. Dieses bietet einen Konstruktor für FFmpeg-Befehle an, mit dessen Hilfe sich dessen Parameter als Attribute übergeben lassen [The fluent-ffmpeg contributors, 2021]. In der Umsetzung wurde sich jedoch gegen diese Variante entschieden, da beispielsweise für das Erstellen von Filterketten, wie sie zum Einsatz kommen, um HDR Inhalte zu wandeln, komplexe Aufrufe nötig sind. Dazu erschien es zweckdienlicher, die gut dokumentierte FFmpeg eigene Syntax beizubehalten.

Für das Erstellen besonders skalierbarer Anwendungen wurden die Representational State Transfer (REST) Prinzipien definiert. Sie legen Gestaltungsmuster fest, mit denen es erleichtert wird, eine große Anzahl von API Anfragen zu bedienen [De, 2017, S. 29 f.]. Es sollen nur kurz wesentliche, in der Cloud-Infrastruktur angewendete, Prinzipien thematisiert werden. So wird von den REST-Prinzipien beispielsweise gefordert, dass eine Server-Client-Beziehung umzusetzen ist (ebenda). Im umgesetzten Beispiel fragt das Frontend, in der Rolle des Clients, einen Dienst beim Server, in Gestalt des Backends, an. Wie die Bezeichnung Server bereits andeutet, stellt dieser einen Dienst zur Verfügung. Im konkreten Fall ist dies der Dienst der Formatumwandlung von Videodateien. Ein weiteres umgesetztes Entwurfsmuster ist das Prinzip der Zustandslosigkeit. Es besagt, dass alle Informationen, die zum Erfüllen der Anfrage benötigt werden, im API-Aufruf enthalten sein müssen. Damit ist es nicht nötig, dass der Server den Zustand eines Clients speichert (ebenda). Das heißt, alle Parameter, die vom Backend benötigt werden, sind in dem jeweiligem Aufruf enthalten. Entsprechend werden auf dem Backend keine Informationen über vorangegangene Aufrufe oder Client-Zustände gespeichert.

In der konsequenten Umsetzung gipfeln diese Prinzipien im Architekturmodell der Microservices. Dabei wird eine größere Anzahl individuell unabhängiger Programmteile ausgerollt. Auf diese Weise ist jeder der „kleinen Dienste“ ein vollwertiges eigenes Programm inklusive Netzwerkanbindung und Möglichkeiten zur Datenhaltung [Familiar, 2015, S. 6]. Die Orientierung an diesem Modell bietet für den Cloud-Encoder die Möglichkeit, das Backend und das Frontend getrennt voneinander anzupassen, weiterzuentwickeln, und zu warten. So können in der Backend-Anwendung neue Funktionen implementiert werden, ohne, dass Anpassungen am Frontend nötig werden. Die einzige Bedingung besteht darin, dass die API-Schnittstellen in Art und Funktion erhalten bleiben (ebenda). Eine zusätzliche Ebene an Komplexität wird in der umgesetzten Softwarelösung durch die Interaktion mit dem Amazon Simple Storage Service (S3)-Speicher hinzugefügt. Dabei handelt es sich um einen hochverfügbaren, skalierbaren Objektspeicher, der durch den Cloud-Anbieter Amazon Web Services (AWS) zur Verfügung gestellt wird [Gulabani, 2017, S. 223]. Dieses Angebot ermöglicht es, Dateien in Form von Objekten, hoch- bzw. herunterzuladen und diese zu verwalten (ebenda). Die Art, wie dieser Speicher in die Anwendung integriert wurde, kann der Grafik Abbildung 4.6 entnommen werden. So kann auf diesen nicht, wie auf ein direkt mit einem Rechner verbundenes Speichermedium, ohne Umschweife zugegriffen werden. Vielmehr erfolgt auch hier die Interaktion über eine API. Der Ablauf der API-Aufrufe, um eine Anfrage zum Wandeln von Videodateien zu stellen, soll nachfolgend noch einmal erläutert werden.

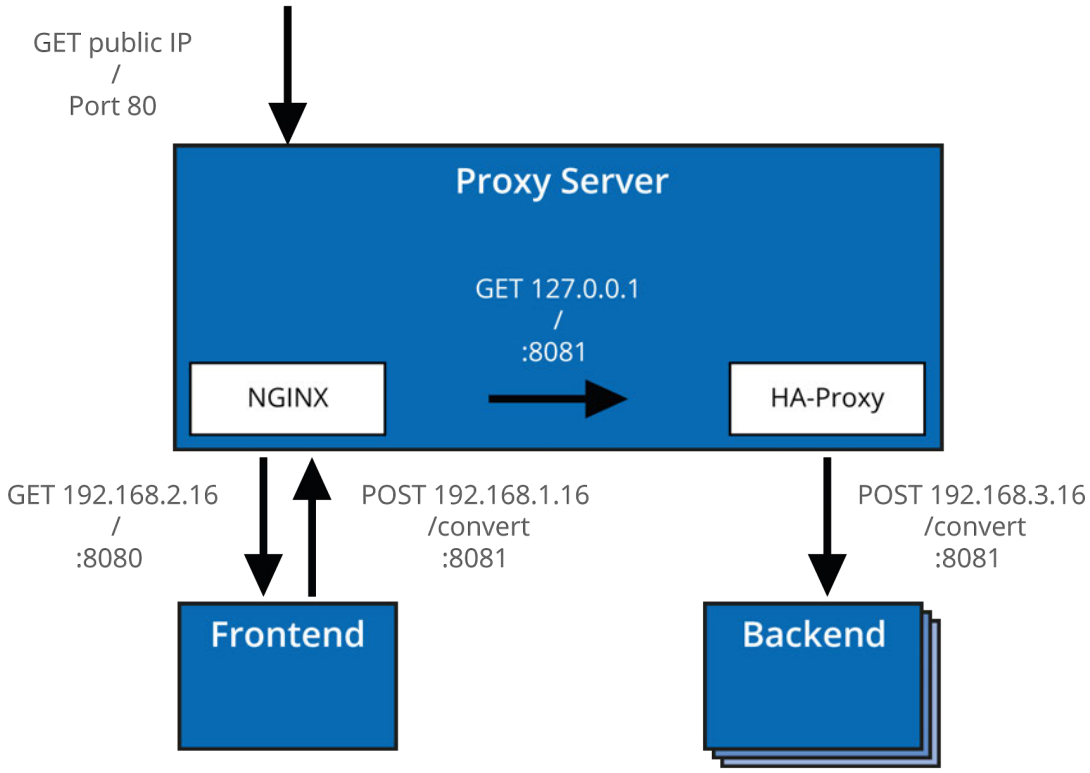

Abbildung 4.1: Ablaufs einer Anfrage und Netzwerk Adressierung (eigene Darstellung)

Wie in Abbildung 4.1 angedeutet, sendet die Frontend-Anwendung einen HTTP-POST-Request an das Backend. Auf dem dort laufenden Node.js Server ist ein entsprechender Endpoint implementiert. Dabei handelt es sich um eine Funktion, die durch HTTP-Anfragen aufgerufen werden kann. Diese spezifiziert dann die weitere Behandlung des Aufrufs. Neben der URL, unter welcher der

Ziel-Endpunkt zu erreichen ist und einer IP-Adresse, an die eine Antwort gesendet wird, enthält die API-Anfrage auch entsprechende Nutzdaten. Diese sind in der JavaScript Object Notation (JSON) codiert und beinhalten die Parameter, mit denen die Videodatei umgewandelt wird. Für eine Beispielanfrage mit Farbraumwandlung könnten die Nutzdaten einer entsprechenden Anfrage, wie folgt aussehen:

```
{
  bitrate:          "10",
  outputName:       "filename.mp4",
  outputFormat:     "mp4",
  url:              "https://nodename.s3.region.amazonaws.com/file.mov?ID",
  filename:         "file.mov",
  codec:            "h264",
  width:            "720",
  height:           "480",
  profile:          "main",
  peaklum:          "100",
  tonemap:          "linear",
  primaries:        "709",
  matrix:           "709",
  transfer:         "709",
  advanced_colour:  "advanced_colour"
}
```

Abbildung 4.2: Nutzdaten des JSON-Inhalts einer Beispielanfrage

Im Anschluss an die Umwandlung sendet das Backend eine HTTP-Antwort an die in der Anfrage spezifizierte Ressource. Diese enthält in Ihren Nutzdaten, wiederum JSON codiert, eine Erfolgsmeldung oder einen Fehlercode inklusive kurzer Fehlerbeschreibung. So kann diese Information zur weiteren Fehlerbehandlung genutzt werden.

4.1.3 Geschäftlogik

In Anlehnung an das Schichtenmodell der Softwarearchitektur wird jener Teil der Anwendung, der für das Durchführen des eigentlichen Zielprozesses zuständig ist, als Geschäftslogik bezeichnet [Schatten et al., 2010, S. 215]. Im Beispiel des Cloud-Encoders ist damit jener Teil der Anwendung, der den eigentlichen Transcoding-Prozess durchführt, gemeint. Er wird auf einem der Backend-Server realisiert. Dazu werden zu Beginn die Teile der JSON-Anfrage in lokale Variablen übernommen. Anschließend wird ihr Inhalt einer Zulässigkeits- und Sicherheitsüberprüfung unterzogen, die unter Abschnitt 4.4 näher beschrieben wird. Danach wird basierend auf den jeweiligen Werten ein String zusammengesetzt, der an FFmpeg übergeben wird. Wurde vom Nutzer angegeben, dass er die erweiterte Farboptionen nutzen möchte, wird dafür ein gesonderter Sub-String eingefügt. So wird in diesem Fall der Befehl um folgende Filterkette erweitert:

```
-vf zscale=t=linear:npl=${peaklum},
    format=gbrpf32le,zscale=p=${primary},tonemap=tonemap=${tonemap}:desat=0,
    zscale=t=${transfer}:m=${matrix}:r=tv,format=yuv420p
```

Diese besteht im Wesentlichen aus vier Teilen. Im ersten Teil wird die Maximalhelligkeit linear auf die Zielhelligkeit abgebildet. Als Zwischenschritt wird anschließend das Pixelformat mittels „format“-Funktion in ein RGB- Format gewandelt. Im dritten Schritt werden die in Anhang C dargestellten Optionen an die Funktion übergeben. Abschließend wird das Pixelformat zurück in ein YUV-Format gewandelt. Die Bestandteile, die mit ${...} umgeben sind stellen dabei die Parameter dar, die aus dem JSON-Objekt stammen.

Der so zuammengesetzte Befehlsstring wird in einem asynchronen „ChildProcess“ ausgeführt. Dieser hat den Vorteil, dass der Befehl ausgeführt werden kann, ohne, dass die Anwendung für weitere eingehende Anfragen blockiert wird [Ihrig, 2013, S. 28]. Damit können parallel mehrere Transcoding-Prozesse auf einem Backend-Server stattfinden. Voraussetzung für deren effiziente Abarbeitung ist dabei natürlich immer, dass genügend Hardwareressourcen zur Verfügung stehen. Um eben diese möglichst effizient nutzen zu können, gliedert das „ChildProcess“-Modul rechenintensive Aufgaben aus dem Eventloop aus (ebenda). Die Single-Thread-Architektur von Node.js würde sonst die optimale Nutzung der CPU-Ressourcen verhindern. Stattdessen ermöglicht es die verwendete „exec“-Methode Bash-Befehle aus Node.js in einer neuen Shell aufzurufen. Auf diese Weise wird der zuvor zusammen gesetzte FFmpeg Befehl an die Shell übergeben. Wie dabei die Ausgaben der Shell gehandhabt werden, kann gesondert spezifiziert werden. Damit eventuelle Fehler beim Formatwandeln festzustellen sind, werden Sie in der umgesetzten Infrastruktur an die Konsole des Node.js-Servers zurückgegeben.

4.1.4 Nutzerinterkation

Neben der auf dem Backend Server ausgeführten Geschäftslogik ist das Nutzerinterface wesentlicher Bestandteil des Cloud-Encoders. Eine elementare zu erfüllende Aufgabe dabei ist das Rendering der HTML-Seiten, die das Nutzerinterface bilden. So können Nutzerinnen und Nutzer die gewünschten Parameter der gewandelten Videodatei in ein HTML-Formular eintragen. Einen umfassenden Überblick über das Nutzerinterface findet sich in Anhang C.

Prüfung der Eingaben Die eingegebenen Daten werden im Anschluss in mehreren Schritten bereits an dieser Stelle auf Plausibilität geprüft. So sind die möglichen Eingabedatentypen bereits auf der HTML-Ebene restriktiert. Im Anschluss folgt eine Prüfung mittels einer JavaScript-Funktion, die diese Parameter auf das Einhalten eines sinnvollen Wertebereiches prüft. Ein weiterer Mechanismus zu Überprüfung der Nutzereingaben ist die Funktion „visibility()“. Hat der Nutzer einen Videocontainer ausgewählt, so sorgt diese dafür, dass nur Codecs angezeigt werden, die tatsächlich mit den entsprechenden Containern verwendet werden können. Nach der Wahl eines Codecs sorgt sie weiterhin dafür, dass nur gültige Transcoding-Profile ausgewählt werden können.

Einbindung der Dateien des Objektspeichers Im unteren Teil der mit „Convert Files“ überschriebenen Seite finden sich die Dateien, die sich aktuell im AWS S3-Objektspeicher befinden. Diese werden mithilfe der auf der Seite implementierten Funktion „getBucketObjectList()“ geladen. Dazu ist zu sagen, dass Teile der Einbindung des Objektspeichers von der Anwendung von [Ankit, 2022] inspiriert sind. In seinem Beispielcode zeigt er die Einbindung eines AWS S3-Buckets in eine Node.js Anwendung. Die Funktionen, die zum Aufruf und Ändern von Dateien im Objektspeicher verwendet werden, sind daher an diese Lösung angelehnt. Schon beim Laden der Seite wird auf diese Weise ein GET-Request an den „/all-files“ Endpoint gesendet. Dieser wurde in der Node.js Anwendung

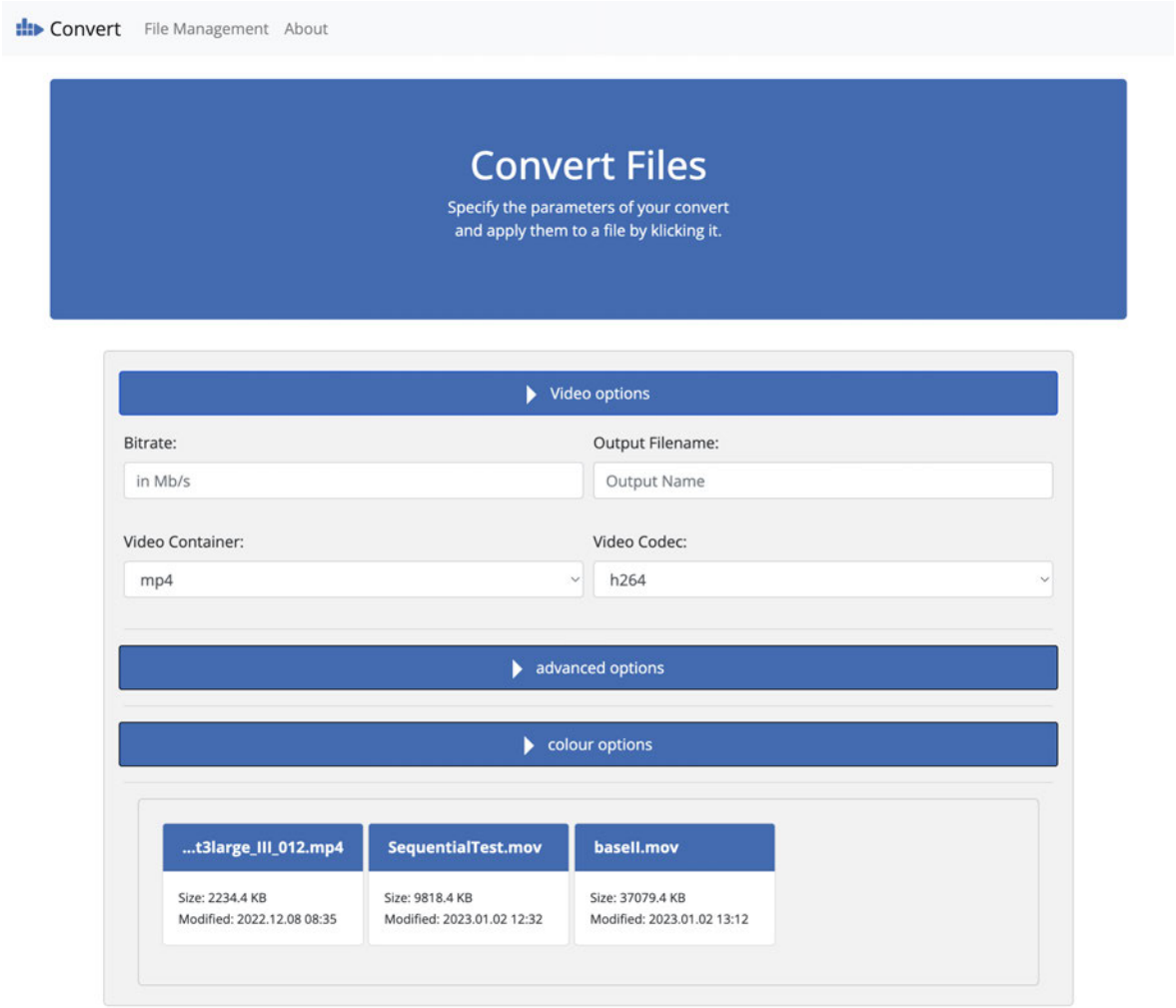

Abbildung 4.3: Darstellung des Nutzerinterfaces zur Parametereingabe

spezifiziert und ruft die Funktion „S3Get“ im S3-Controller Paket auf. Diese hat im Wesentlichen zwei Aufgaben. Zuerst wird so die „getBucketListFromS3()“-Methode aufgerufen. Darüber hinaus, werden Logging- und Fehlerbehandlungs-Funktionalitäten zu dieser Funktion hinzugefügt. Die „getBucketListFromS3()“-Methode selbst instanziiert zuerst ein Objekt der vordefinierten Klasse „S3“. Die Beschreibung dieser erfolgt im AWS-software development kit (SDK), einer Sammlung von JavaScript Klassen und Methoden. Zur Interaktion mit dem Objektspeicher werden diese vom Clouddanbieter zur Verfügung gestellt. In diesem Schritt werden die Zugangsdaten des Objektspeichers als entsprechende Parameter des neuen „S3“ Objektes übergeben. Diese beiden Schlüssel werden beim Start der Anwendung als Umgebungsvariablen eingegeben. Die Methode „.listObjects“ des S3 Elements liefert ein Objekt zurück, das unter anderem ein Array der im Objektspeicher befindlichen Dateien, sowie deren Größe und Änderungsdatum enthält. Die darin enthaltenen Daten werden anschließend zur besseren Lesbarkeit aufbereitet. Anschließend werden sie in eine HTTP-Antwort eingebettet und zurück an die aufrufende Funktion gesendet. In einem Callback dieser Funktion werden dann die Bestandteile des erhaltenen Arrays in den HTML-Text gerendert. Dazu wird die JQuery Methode „.getElementById“ verwendet. Diese erlaubt es, den Inhalt eines HTML-Elements zu ändern. Das Ergebnis sind die in Abbildung 4.4 dargestellten Karten, die je eine Datei im Objektspeicher repräsentieren. Wie bereits unter Abschnitt 4.1 erwähnt, macht Node.js es möglich, diese Aufrufe asynchron durchzuführen. Die Bilder unterhalb zeigen auch, dass Dateinamen, die länger

als 19 Zeichen sind, automatisch gekürzt werden. Damit wird sichergestellt, dass auch Dateien mit einem langen Namen übersichtlich im Nutzerinterface dargestellt werden können. So wird in diesem Fall der Anfang des Dateinamens durch drei Punkte ersetzt.

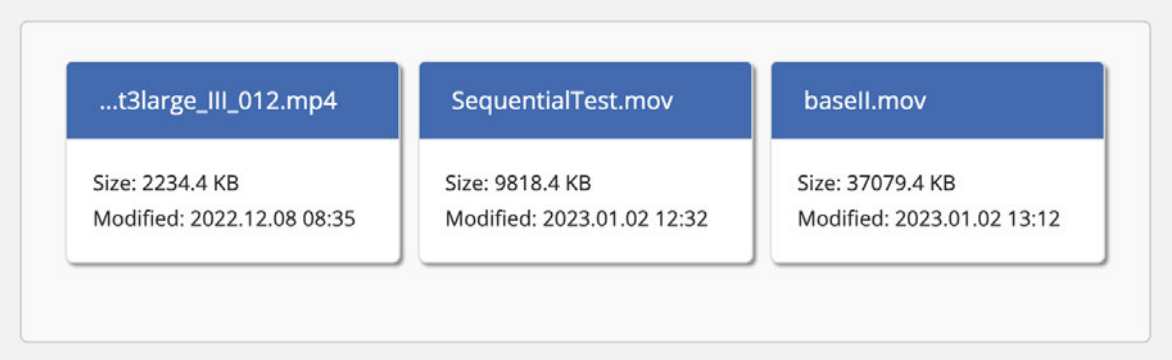

Abbildung 4.4: Karten zur Repräsentation der Dateien im Objektspeicher

```
function shorten (item)
{
    item.lastModified=
        item.lastModified.slice(0,-8).replace(/-/g, ".").replace(/T/g, " ");
    var len = item.key.length;
    item.shortkey=item.key;
    if (len>19) {
      item.shortkey="..."+item.key.slice(-19);
    }
    return(item);
}
```

Abbildung 4.5: Funktion zum Formatieren des Karteninhalts Quellcodeauszug

Dem Quellcode ist neben der Funktion, zum Kürzen des Namens, auch zu entnehmen, dass auch eine Anpassung des „.lastModified“-Attributs des „item“-Objekts vorgenommen wird. Da die vom Objektspeicher zurückgegebenen Daten recht unübersichtlich formatiert sind, wird dies nötig. So werden die letzten acht Zeichen, die Sekunden und Zeitzone enthalten, entfernt und eine leichter lesbare Formatierung hergestellt.

Versenden der Formulardaten Wird nun auf eine der so erzeugten Karten geklickt, wird die Funktion „sendForm()“ aufgerufen. Neben der bereits beschriebenen Datenvalidierung ergänzt diese auch zwei Parameter des Formulars, bevor die Anfrage versendet wird. So wird zum einen der Name, der ausgewählten Datei im Objektspeicher hinzugefügt. Zum anderen wird eine sogenannte „presigned URL“ zu den Formulardaten hinzugefügt. Um diese zu generieren, wird der API-Endpoint „/get-object-url“ zusammen mit dem Parameter des ausgewählten Dateinamens aufgerufen. Ähnlich wie beim Abruf der Elemente des Objektspeichers wird damit der Aufruf einer Methode des AWS-SDK ausgelöst. Wie der Name der „getPresignedURL“-Methode schon andeutet, wird damit ein Link erstellt, mit dem es möglich wird, eine bestimmte Ressource aus dem Objektspeicher herunterzuladen [Amazon Web Services, Inc, 2022e]. Dieser Link ermöglicht es dann eine spezifizierbare Zeit lang, die Datei, für die er erstellt wurde, herunterzuladen. Daraufhin werden die Formulardaten an den POST-Endpoint „/parameters“ des Frontend-Servers gesendet. Im damit verbundenen Codeabschnitt

wird der URL-Encoded Request in das JSON-Format überführt. Mit dieser deutlich besser lesbaren Form der übermittelten Daten werden dann verschiedene Logging Schritte durchlaufen. Ziel dieser ist es, eventuelle Fehler des Programms nachvollziehbar zu machen. Im Anschluss wird ein neuer Request im JSON Format an den Backend-Server versendet. Entsprechend der Antwort, die nach dem Wandeln der Datei zurückkommt, wird eine HTML-Seite gerendert, die den Nutzer über den Status des Prozesses informiert.

Dateienverwaltung Neben der Eingabe von Parametern zum Wandeln von Videodateien kann auch der Inhalt des Objektspeichers über das Nutzerinterface verwaltet werden. Die entsprechende Seite des Nutzerinterfaces kann in Abbildung C.3 eingesehen werden. Auf dieser können Dateien heruntergeladen oder gelöscht werden.. Ähnlich wie zuvor beschrieben, geschieht dies wiederum durch Klicken der entsprechenden Datei-Karte. Diese beinhalten jeweils eine Funktion, die über einen HTML-Endpoint auf dem Frontend-Server die entsprechende Funktionalität der AWS SDK auslöst. Da beim Testen der Anwendung wiederholt Dateien versehentlich gelöscht wurden, wurde das Nutzerinterface um eine Warnung vor dem Löschen von Dateien erweitert. Auf der gleichen Seite können auch neue Dateien zum Objektspeicher hinzugefügt werden. Dies geschieht durch das Klicken der entsprechenden Schaltfläche im oberen Drittel der Seite. Das HTML-File-Input-Element löst, wenn es verändert wird, wiederum einen API-Aufruf aus. Damit bietet das Nutzerinterface die Möglichkeit Dateien zu verwalten und diese anschließend zu wandeln.

4.2 Beschreibung der Infrastruktur und Netzwerkelemente

Neben dem reinen Anwendungscode, wird die Funktionsweise des Cloud-Encoders maßgeblich durch die Rechen-, Speicher- und Netzwerkkomponenten bestimmt, mit deren Hilfe die Anwendung betrieben wird. Es wäre hier nicht ganz richtig von Hardware zu sprechen, da es sich aufgrund der Virtualisierung nicht zwingend um solche handeln muss [Luntovskyy and Spillner, 2017, S. 97]. Dennoch werden die wesentlichen Bestandteile, aus denen sich die Infrastruktur zusammensetzt, im Folgenden beschrieben.

4.2.1 Recheneinheiten

Das Herzstück der Infrastruktur bilden die Front- bzw. Backend-Server. Bei diesen handelt es sich mit AWS-Amazon Elastic Compute Cloud (EC2)-Instanzen jeweils um virtuelle Server, die einer Anwendung als Recheneinheiten zur Verfügung stehen [Nadon, 2017, S. 107]. Cloudtypisch ist deren Rechenleistung flexibel wählbar und bestimmt maßgeblich den stündlich abgerechneten Preis mit (ebenda). In der umgesetzten Anwendung kamen zumeist Servereinheiten mit sehr beschränkten Ressourcen zum Einsatz, um die Anwendung bei minimalen Kosten testen zu können. So wurden für die Proxy- und Frontendinstanz, deren Aufgaben weniger rechenintensiv sind, „t2.micro“ Instanzen verwendet. Sie verfügen über einen CPU-Kern und einen Arbeitspeicher von einem GiB [Amazon Web Services Inc, 2022]. In den durchgeführten Tests haben sich diese Spezifikationen für die weniger belasteten Anwendungsteile als zweckmäßig gezeigt. Mehr Ressourcen wurden hingegen den Backend-Servern zugewiesen. So kamen für das Video-Transcoding, als rechenintensivem Prozess, beispielsweise Instanzen des Typs „t3.medium“ oder „t3.large“ zum Einsatz. Diese verfügen über jeweils zwei Prozessorkerne mit einer Taktfrequenz von bis zu 3,1GHz und 4 bzw. 8 GiB Arbeitsspeicher (ebenda). Eine genauere Betrachtung zweckmäßiger Servergrößen erfolgt in Unterabschnitt 5.2.3.

Neben der Prozessorkapazität wurde auch dessen Architektur und das Betriebssystem der EC2 Server angepasst, um möglichst zweckmäßig für den Einsatz im Cloud-Encoder zu sein. Dies ist über das Auswählen eines „Amazon Machine Images“, kurz AMI möglich [Amazon Web Services, Inc, 2022b]. Diese ermöglichen es, vorgefertigte Konfigurationen der genannten Parameter auf dem Server zu laden (ebenda). So wurden im Verlauf der Arbeit sowohl Prozessoren der x86 als auch Arm Architektur getestet. Dabei handelt es sich um die beiden größten Architekturen für Serverprozessoren derzeit. Obwohl sich deren Aufbau grundlegend unterscheidet, ist es für das Verständnis der Arbeit nur wichtig, deren grundlegendste Eigenschaften in die Betrachtung einzubeziehen. So hat die Verwendung von x86-Prozessoren in Serversystemen eine lange Historie [Ledin, 2022, S. 288]. Der verwendete Cloudanbieter bietet jedoch auch Arm-basierte Prozessoren an, die teils selbst vom Cloudanbieter entwickelt werden. Aufgrund ihrer genauen Abstimmung auf den Anwendungsfall sind sie oft effizienter und damit günstiger als vergleichbare x-86 Modelle [Amazon Web Services, Inc, 2023]. Als Betriebssystem für alle Server wurde sich für Ubuntu 22.04 LTS Linux entschieden. Maßgeblich dafür war die große Anzahl an bereits vorkompilierten Paketen und zahlreiche syntaktische Vereinfachungen für das Verwalten des Servers per Shell-Befehlen. Mit der Wahl einer LTS-Version wird weiterhin sichergestellt, dass diese auch mittelfristig unterstützt wird, was den Wartungsaufwand minimiert.

4.2.2 Speicherressourcen

Neben den Recheneinheiten bildet auch der Speicher einen wesentlichen Bestandteil der erstellen Cloudumgebung. Ein unverzichtbarer Teil dessen sind die Speichereinheiten, die den Instanzen direkt zugewiesen sind. Sie fungieren ähnlich den Festplatten eines Computers. Für den Cloud-Encoder speichern sie neben Anwendungs- und Konfigurationsdateien, im Falle der Backend-Instanzen, auch die heruntergeladenen Videodateien. Ähnlich wie die CPU-Kapazität kann auch die Größe dieser virtuellen Festplatten fast beliebig skaliert werden. Im Falle des Frontends wird von einer Speichergröße von 30 GB ausgegangen, um das Betriebssystem, die Anwendungen und die jeweiligen Konfigurations-Dateien zu hinterlegen. Wesentlich größer muss die Festplatte der Backendinstanzen gewählt werden. Beispielhaft wurden hier eine Festplatte von 1 TB gewählt. In einer praktischen Anwendung, in der auch längere Videodateien in Produktionscodecs abgelegt werden, kann jedoch durchaus die Verwendung eines größeren Speichers nötig werden.

Eine zweite verwendete Speicherart ist der AWS S3 Speicher, der als geteilter Speicher für Videodateien dient. Dabei handelt es sich um einen skalierbaren Objektspeicher in der Cloud [Gulabani, 2018, S. 147 f.]. Dieser ermöglicht es, Dateien unter anderem über die REST-API, die er zur Verfügung stellt, zu verwalten. Dafür werden, wie bereits angeklungen, HTTP-Anfragen an den Objektspeicher gestellt, um mit dessen Inhalt zu interagieren. Die Anfragen können dabei entweder über private Netze oder auch das Internet an den Objektspeicher gesendet werden. Diese Art des Zugriffes macht es möglich, dass sowohl Frontend- als auch Backend Instanzen auf den Speicher zugreifen können. So können Dateien, die vom Nutzer hochgeladen werden, von der Frontend-Instanz aus in den Objektspeicher gelangen. Von dort können sie zu Beginn des Wandlungsprozesses in den Speicher der jeweiligen Recheninstanz geladen werden. Die gewandelte Datei wird im Anschluss wiederum vom Speicher des Servers in den S3 Speicher hochgeladen. Schließlich werden automatisch sowohl die Ausgangsdatei, als auch das gewandelte Video vom lokalen Speicher gelöscht. Damit wird der Speicherbedarf des Servers auf ein nötiges Minimum reduziert.

4.2.3 Netzwerkkomponenten

Um den Austausch von Daten zwischen den zuvor beschriebenen Rechnern und Speichern zu ermöglichen, werden Netzwerkkomponenten genutzt. Ähnlich wie bei den CPU Einheiten werden jedoch auch diese virtualisiert, sodass keine zwingende Beziehung zwischen Hardware und Netzwerkfunktion besteht. Ein Ziel der geschaffenen Netzwerkinfrastruktur ist es dabei, dass Netzwerkverkehr zwischen den Instanzen nicht über das Internet abgewickelt werden muss. Dies ermöglicht die virtual private Cloud (VPC). Die VPC ermöglicht es, ein virtuelles, logisch isoliertes Netzwerk aufzuspannen [Amazon Web Services, Inc, 2022c]. Vergleichbar ist diese Eigenschaft mit einem Heimnetzwerk, in dem genau wie im praktischen Beispiel Netzwerkadressen nach RFC1918 vergeben werden [Rekhter et al., 2022]. Zusätzlich erhalten Cloudserver jedoch eine öffentliche IP-Adresse, wenn sie aus dem Internet erreichbar sein sollen.

Subnets In Analogie zu V-Lans können VPCs in weitere Subnetze unterteilt werden. Dies spiegelt sich in der Untergliederung der Infrastruktur in die Subnetzte „Proxy“, „Frontend“, „Backend A“ und „Backend B“ wider. Die Unterteilung der Backend-Server hat ihren Hintergrund in der Verteilung dieser auf zwei Verfügbarkeitszonen. Mehr Informationen dazu folgen in Unterabschnitt 5.2.1. Abbildung 4.6 greift diese Unterteilung mit den durch Strichlinien unterteilten Bereichen auf. So steht jede der Boxen für eines der Subnets. Die blauen Boxen in der Infrastruktur stehen hingegen für die jeweiligen virtuellen Server. Die Darstellung als Stapel, wie bei „Backend A“ geschehen, verweist dabei auf die Möglichkeit, dass an dieser Stelle eine Vielzahl von Servern existiert. Ebenfalls in dieser Darstellung sind die Adressbereiche der Subnetze zu sehen. Dabei wird auch deutlich, dass nur der Reverse Proxy in der Lage ist, zwischen den Subnetzen zu „routen“. Auf diese Weise kann der Netzwerkverkehr genauer reglementiert werden.

Security Groups So kann jedem Netzwerk eine „Security Group“ zugewiesen werden. Diese fungieren ähnlich einer Firewall, die nur auf ein bestimmtes Sub-Netz angewendet wird. So können anhand der Parameter Netzwerkprotokoll, Port und IP-Adresse von Verbindungen spezifiziert werden, die zugelassen oder abgelehnt werden. Zu diesem Zweck wurden in der umgesetzten Lösung zwei Security Groups erstellt. Server, die der restriktiveren der beiden Gruppen zugewiesen sind, können nur von Adressen innerhalb der Infrastruktur erreicht werden. Allein der Proxyserver, dessen Aufgabe noch näher beschrieben wird, ist nicht dieser Gruppe zugewiesen. In Abbildung Abbildung 4.6 wird dies durch das Label „Public Subnet“ ausgedrückt. Als Teil dieser Gruppe ist er aus dem Internet erreichbar. In Abbildung 4.6 ist dies durch die Pfeile am oberen Bildrand dargestellt. So ist erkennbar, dass die Instanzen in den privaten Subnetzen nicht direkt nach außen kommunizieren. Eine Verbindung zum Internet kann allein der Proxyserver aufbauen. Damit das gelingt, wird zusätzlich ein „Routingtable“ erstellt. Er hat die Aufgabe, jedes Netzwerkpaket, das nicht direkt an eine IP-Adresse innerhalb des privaten Netzwerks adressiert ist, an das Internet Gateway weiterzuleiten. Nicht über dieses Gateway laufen die Anfragen an den Objektspeicher ab. Sie erfolgen über einen „VPC-Endpoint“. Diese Einrichtung des Cloudanbieters ermöglicht es, dass die entsprechenden Anfragen nicht über das Internet, sondern über interne Netze des Betreibers gesendet werden. Neben Sicherheitsaspekten wirkt sich dies auch kostensenkend aus, da für Daten, die in das Internet gesendet werden, eine Gebühr anfällt.

Reverse Proxy Damit beispielsweise der Frontend-Server, der keinen direkten Internetzugang hat, zur Servicebereitstellung genutzt werden kann, kommt ein „Reverse Proxy“ zum Einsatz. Dabei handelt es sich um eine Anwendung, die, stark vereinfacht, HTTP-Anfragen entgegennimmt

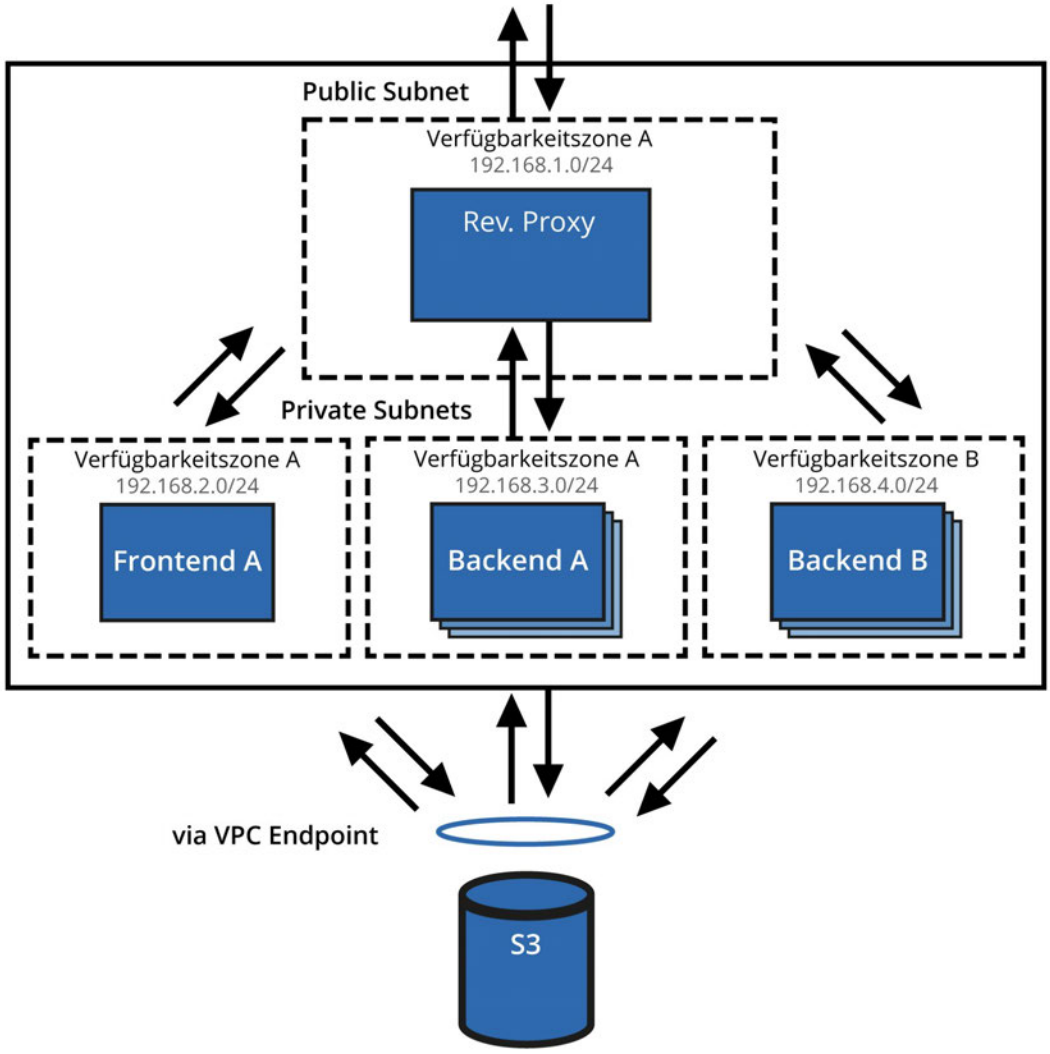

Abbildung 4.6: Vereinfachte Darstellung der Netzwerkinfrastruktur (eigene Darstellung)

und weiterleitet [Christudas, 2019]. Damit erfüllt sie, neben der angesprochenen Erreichbarkeit von sonst unzugänglichen Servern, verschiedene Aufgaben. Zum einen wird so der interne Aufbau der Infrastruktur für Entitäten aus dem Internet verborgen [Du and Nie, 2011]. Der Sicherheitsaspekt dessen wird in Abschnitt 4.4 näher erklärt. Daneben übernimmt der „Reverse Proxy“ auch die Aufgabe, Subdomains und URLs für den Nutzer komfortabel zugreifbar zu machen. Dieser, für eine reibungslose Funktion der Infrastruktur unumgängliche Dienst, wird wie zuvor schon angedeutet auf einem gesonderten, virtuellen Server ausgeführt, der als einziger uneingeschränkt aus dem Internet erreichbar ist. Am Beispiel des Cloud-Encoders hat er damit die Aufgabe, dass dessen Startseite ohne zusätzliche URL-Bestandteile aufrufbar wird. Da aus Kostengründen keine Domain für die Infrastruktur erworben wurde, heißt das, dass beim Eingeben der öffentlichen IP-Adresse des Proxyservers direkt die Startseite des Encoder geöffnet wird. Dies geschieht, da „nginx“, eine Open Source Anwendung, die zu diesem Zweck zum Einsatz kommt, die Anfrage an die entsprechende Ressource auf dem Frontend Server weiterleitet. Die anschließende Antwort wird dann zuerst an den Proxyserver gesendet. Dieser leitet die empfangenen Pakete dann an die anfragende Entität weiter. Auch Anfragen des Frontend-Servers an das Backend werden zuerst an den Reverse Proxy gesendet. Von dort aus werden diese jedoch nicht direkt an einen Server gesendet, sondern an einen Dienst, der ebenfalls auf dem Proxyserver ausgeführt wird.

Loadbalancer Weiterhin kommt in der Anwendung der Dienst HAProxy zum Einsatz, der ebenfalls als Open-Source-Software verfügbar ist. Er übernimmt im Cloud-Encoder die Aufgabe eines Loadbalancers. Allgemein bezeichnet man damit eine Einheit in einem Netzwerk von Rechnern, mit der Aufgabe Rechenlast auf eine Gruppe von Servern zu verteilen, um die Leistung des Gesamtsystems zu verbessern [Membrey et al., 2012]. Konkret bedeutet das, dass der Loadbalancer die Anfragen, die an das Backend gestellt werden, auf die verschiedenen Server verteilt. Dabei geht er nach dem „least connected“ Prinzip vor. Das heißt, aus der Gruppe der verfügbaren Rechner wählt er immer den aus, zu dem gerade die wenigsten Verbindungen bestehen. Eine weitere Funktion, die durch HAProxy umgesetzt wird, sind als „Health Checks“ bezeichnete Statusabfragen der verwalteten Server. Dabei sendet die Einheit in wählbaren Intervallen einen HTTP-GET-Request an einen bestimmten Endpoint auf dem zu überwachenden Server. Antwortet dieser mit einem HTTP-Status Code zwischen 200 und 299 wird davon ausgegangen, dass der Server sich in einem funktionsbereiten Zustand befindet. Wird hingegen ein anderer, oder überhaupt kein Statuscode zurückerhalten, wird der Server als defekt betrachtet. Der Loadbalancer wird dann eine variable Zeit lang keine Anfrage an diesen Backend-Server mehr weiterleiten, bevor er erneut versucht diesen zu erreichen. Im Falle des umgesetzten Beispiels geschieht dies alle 2s. Damit wird gewährleistet, dass ein Ausfall eines Backend-Servers maximal 2s lang zu nicht beantworteten Anfragen führt. Dies gilt natürlich nur, solange noch mindestens ein funktionsfähiger Server verfügbar ist. Auf diese Weise verbessert der Loadbalancer nicht nur die Leistung, sondern auch die Zuverlässigkeit der Transcoding-Infrastruktur. Das Zusammenspiel von Reverse Proxy und Loadbalancer kann auch in Abbildung 4.7 nachvollzogen werden. So ist zu sehen, wie zuerst alle Anfragen an den Proxy gelangen. Der darunter dargestellte Loadbalancer verteilt die Anfragen dann auf die Anwendungsserver, welche links und rechts abgebildet sind.

Zusammenfassend lässt sich sagen, dass durch die logische Teilung der Netzwerkbereiche und die Implementierung von Security Groups bereits auf der Netzwerkebene erste Sicherheitsaspekte umgesetzt worden sind. Weiterhin legt die strikte Trennung von Zuständigkeiten und die Implementierung eines Loadbalancers den Grundstein für eine nahtlose Skalierbarkeit der Infrastruktur.

4.3 Beschreibung der Servicebereitstellung

Neben dem Aufbau der erstellten Infrastruktur zeichnet diese sich durch die Art aus, wie das gesamte System gestartet wird. So liegt es auf der Hand, dass die zuvor beschriebenen Komponenten in vielen denkbaren Anwendungen nicht ununterbrochen in der Cloud ausgeführt werden. Vielmehr erscheint ein bedarfsgerechtes Starten und Stoppen des Systems als zweckmäßig. Um dies mit möglichst geringem Aufwand zu erreichen, wurde der sogenannte „Ausroll-Prozess“ weitestgehend automatisiert. Im folgenden Abschnitt werden die dafür nötigen Mechanismen grob umrissen.

Grundsätzlich könnten alle zuvor beschriebenen Konfigurationen der Cloud-Ressourcen in der Web-Oberfläche des Cloudanbieters umgesetzt werden. Allerdings ist anhand der vorausgegangenen Beschreibungen auch deutlich geworden, dass es sich dabei um einen langwierigen und fehleranfälligen Prozess handeln würde. Aus diesem Grund wurde sich für die Bereitstellung der Infrastruktur für eine „Infrastructure as a Code“ Lösung entschieden. Deren Grundgedanke ist es, mithilfe einer deklarativen Programmiersprache alle Komponenten und Konfigurationen zu beschreiben, die sonst in einer grafischen Nutzeroberfläche durchgeführt werden müssten [Krief and Hashimoto, 2020].

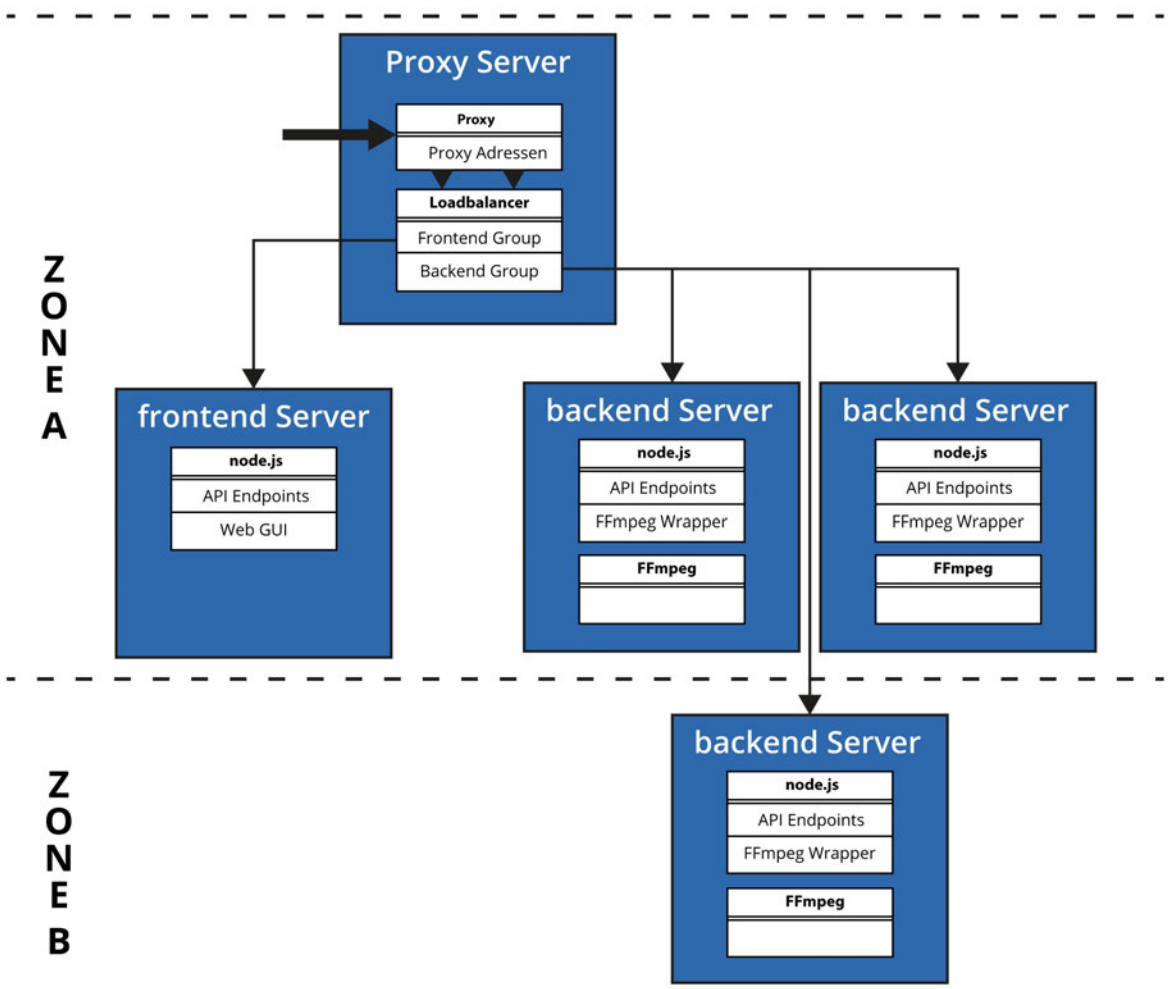

Abbildung 4.7: Zusammenspiel von Reverse Proxy und Loadbalancer (eigene Darstellung)

Terraform Konkret handelt es sich beim verwendeten Werkzeug um „Terraform“, einer von HashiCorp entwickelten Open Source Lösung, um genau dieser Anforderung gerecht zu werden [Zadka, 2022, S. 225 f.]. So werden im Code die zuvor beschriebenen Bestandteile der Infrastruktur erstellt und deren Eigenschaften spezifiziert. Dabei können, wie in vielen anderen Programmiersprachen auch, Variablen gesetzt werden, die dann an mehreren Stellen im Code wiederverwendet werden können. Damit können Fehlerquellen durch uneinheitliche Ausgangsparameter minimiert werden. Die verwendeten Variablen können dann entweder in einer sonders dafür angelegten Datei als Konstanten zugewiesen, oder beim Ausführen des Codes vom Nutzer eingegeben werden. So wird beim Ausrollen des Cloud Encoders beispielsweise nach der Anzahl der Backend-Server gefragt, die erstellt werden sollen. Zusätzlich stellt Terraform auch Sprachbestandteile zur Verfügung, mit deren Hilfe Schleifen oder Bedingungsabfragen realisiert werden können. So wird mit einer solchen Bedingung die vom Nutzer reingegebene Anzahl an Backend-Servern geprüft. Nur wenn diese größer als eins ist, wird ein zweites Subnetz für die zusätzlichen Rechner erstellt. Auf diese Weise wird sich sichergestellt, dass immer nur die effizientest mögliche Infrastruktur ausgerollt wird. Anschließend besteht die Herausforderung, die Server gleichmäßig auf die beiden Subnetze zu verteilen. Auch diese Anforderung lässt sich mit den von Terraform implementierten Funktionen umsetzten. So kann spezifiziert werden, wie viele Server dem jeweiligen Subnetz zugeordnet werden sollen.

Diese beiden Werte lassen sich leicht aus der Gesamtanzahl der Backend-Server bestimmen, zu $n_{\text{ZoneA}} = \lceil \frac{n_{\text{ges}}}{2} \rceil$ beziehungsweise $n_{\text{ZoneB}} = \lfloor \frac{n_{\text{ges}}}{2} \rfloor$. Für die, mit den Gaußklammern angezeigten, Rundungsfunktionen stellt Terraform bereits Funktionen bereit.

Das Erstellen der Ressourcen erfolgt dann durch die Eingabe des „apply“-Befehls in die Shell des Betriebssystems auf dem lokalen Rechner eines Anwenders. Daraufhin gibt die Anwendung eine Übersicht der Ressourcen, die entsprechend des vorliegenden Codes erstellt werden sollen, aus. Wird dieser Plan vom Nutzer bestätigt, beginnt Terraform mit dem Übersetzen des Codes in Befehle, die über das jeweilige Kommandozeileninterface des Cloudanbieters an diesen gesendet werden. Umgekehrt wird beim Herunterfahren der Infrastruktur vorgegangen. Durch das Ausführen des Terraform „destroy“-Befehls werden die beim Cloudanbieter in Anspruch genommenen Ressourcen heruntergefahren. Diese Vorgehensweise erhöht zusätzlich die Portierbarkeit von Infrastrukturen zwischen Cloudanbietern, da die verwendeten Codebestandteile jeweils ähnlich ausfallen.

Server Konfiguration Eine weitere wichtige Aufgabe, die Terraform in der vorliegenden Anwendung übernimmt, ist die Konfiguration der Server, nach deren Erstellung. So können in einem ersten Schritt Konfigurationsdateien, deren genaue Zusammenstellung von Nutzereingaben abhängen, angepasst werden. Dies ist nötig, da die Einstellungen, die an Proxy und Loadbalancer getroffen werden, von der Anzahl der Backend-Server abhängen. Um effizient arbeiten zu können ist es nötig, dass der Loadbalancer die IP-Adressen der erstellten Front- und Backendserver kennt. Nur so ist sichergestellt, dass er nur Server abfragt, die grundsätzlich auch bereit wären Anfragen entgegenzunehmen. Um dies zu erreichen, bietet Terraform zwei Funktionen an. Zum einen können mit der „templatefile()“ Funktion Terraform Variablen in ein Bash Script eingefügt werden. Zum anderen können mithilfe der Anwendung Bash oder Powershell Befehlsfolgen auf dem lokalen Rechner ausgeführt werden, die wiederum genutzt werden können, um bestimmte Werte in Konfigurationsdateien zu ändern oder zu ergänzen. So können die tatsächlich nötigen IP-Adressen der Backend-Server in die Konfigurationsdateien eingefügt werden. Damit werden nur Server überwacht, die auch tatsächlich existieren. Zuerst geschieht dies im Programmteil „local_exec.tf“. Dabei ist .tf jeweils die Endung einer Terraform Datei, die Beschreibungen der Infrastruktur enthält. Der zeitliche Ablauf der Ausführung dieser Dateien ist auch der Abbildung 4.8 zu entnehmen. Dabei stellen die schwarzen Pfeile von „x.tf“ zu „y.tf“ eine Abhängigkeit der Form „X muss nach Y ausgeführt werden“ dar. In den Kästen unter den .tf Files sind jeweils Konfigurationsschritte aufgeführt, die neben dem Erstellen der betreffenden Bestandteile in der Cloud-Umgebung ausgeführt werden. Die auf diese Weise angepassten Konfigurationsdateien werden anschließend über eine Secure Shell (SSH) auf die entsprechenden Server gesendet. Auch zur Automatisierung dessen bietet Terraform ein Modul an. In der Abbildung wird dieser Schritt durch die gestrichelten Pfeile dargestellt. Neben dem Hochladen der beschriebenen Konfigurationsdateien wird diese Funktion auch verwendet, um die ebenfalls individuell angepassten „.service“ Dateien auf die Front- und Backendinstanzen hochzuladen. Diese dienen als Konfiguration für den jeweiligen System-Service. Dabei handelt es sich um ein Werkzeug, das in vielen Linux Distributionen enthalten ist, um Hintergrundprozesse zu verwalten [Kirkbride, 2020]. Diese Funktion wird genutzt, um die Node.js Anwendungen bei einem Neustart oder im Fehlerfall automatisch neu zu starten. Damit wird sichergestellt, dass die Serveranwendung ohne, dass es dafür einen manuellen Eingriff braucht, zuverlässig bereitgestellt wird. Außerdem werden mit ihrer Hilfe Umgebungsvariablen an die Node.js Anwendungen übergeben. Diese sind wie bereits erwähnt unter anderem nötig, um eine Authentifikation beim API Zugriff auf den Objektspeicher zu gewährleisten. Durch die dynamische Anpassung, der so übergeben Werte mithilfe des Terraform-Codes können diese leicht angepasst werden, sollte es zu Änderungen kommen. Daneben wird mit diesem

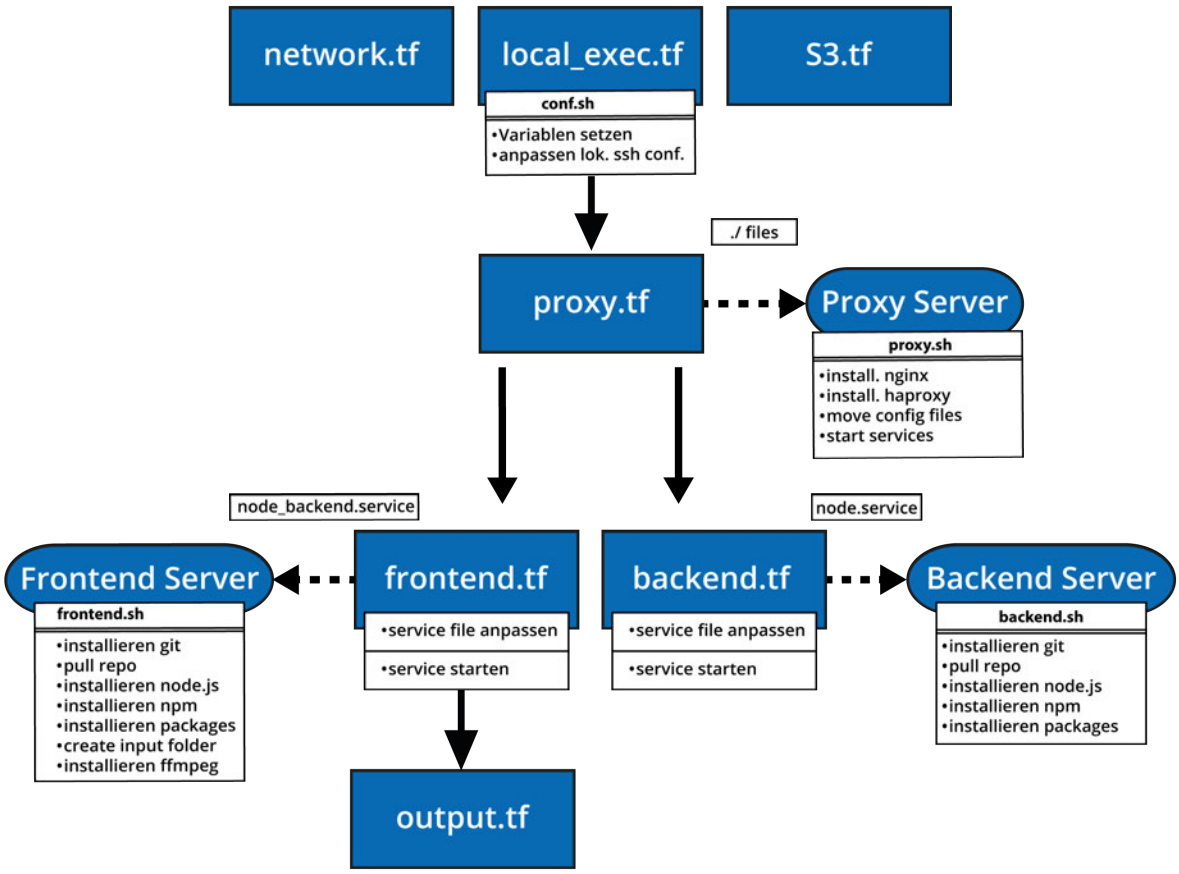

Abbildung 4.8: Ablauf der Server Konfiguration (eigene Darstellung)

Verfahren auch die IP-Adresse übergeben, unter welcher der Loadbalancer zu erreichen ist. So wird sichergestellt, dass Anpassungen in der Terraform Konfiguration, die zu anderen IP-Adressen führen, keine Anpassung der Node.js Anwendung erforderlich machen, da immer die passenden Werte der Variablen gesetzt werden können.

Das gleiche Modul, das auch für das Verteilen der Konfigurationsdateien verantwortlich ist, lädt auch je ein Bash-Script auf die Server hoch. Dessen Aufgabe ist es, alle weiteren Konfigurationen am Server vorzunehmen, die für das Ausführen der jeweiligen Anwendungen notwendig sind. So werden beispielsweise die zum Teil zuvor erwähnten Anwendungen nginx, HAproxy und Git installiert. Bei letzterem handelt es sich um ein Quellcode-Versionierungswerkzeug, das verwendet wird, um den jeweils aktuellen Stand der Node.js Anwendungen auf die Server zu laden. Damit ist sichergestellt, dass sich neue Funktionen oder Verbesserungen jeweils in der ausgerollten Infrastruktur widerspiegeln. Dies geschieht, da der letzte veröffentlichte Stand des Quellcodes aus einem Repository genannten, Onlinespeicher abgerufen wird. Im Falle des Backend-Servers wird auch FFmpeg auf diese Weise installiert. Als letzter Schritt in der Bereitstellung werden durch die „output.tf“ Datei die Werte der öffentlichen IP-Adresse und des automatisch erstellten Domain-Names des Proxys ausgegeben. Auf diese Weise ist schnell erkennbar, wie die Infrastruktur aufrufbar ist.

4.4 Beschreibung und Reflexion von Sicherheitsaspekten

Da die Sicherheit von Cloud-Anwendungen gegenüber Angriffen, unberechtigten Zugriffen und Datendiebstahl eine wesentliche Dimension darstellt [Hsu, 2018], wird diese im Folgenden umrissen. Dazu ist vorab anzumerken, dass die Anwendung in ihrem beschriebenen Zustand nicht zur Verwendung als öffentliche Website gedacht ist. Dennoch wurden bereits auf der Architekturebene zahlreiche sicherheitsrelevante Entscheidungen getroffen. Diese werden im Folgenden reflektiert.

Netzwerksicherheit Eine dabei berücksichtigte Dimension sind die Entscheidungen, die in der Netzwerkarchitektur getroffen wurden. Von Relevanz sind dabei beispielsweise die Security Groups, mit ihrer Restriktion der Erreichbarkeit der Node.js Server. So werden grundsätzlich nur Verbindungen der Protokolle HTTP und SSH zugelassen. Andere Verbindungen, die in Verdacht stehen, einen schadhaften Hintergrund zu haben, werden kategorisch blockiert. Weiterhin fügt auch die Unterteilung des Netzwerks in verschiedene Subnetze für eine zusätzliche Sicherheitsschicht. Es wird damit für Netzwerkentitäten außerhalb der Cloudumgebung unmöglich, direkt mit den node.js Servern zu interagieren. Damit wird es in jedem Fall nötig, zuerst den Reverse-Proxy anzusprechen. Damit erschwert dieser auch, dass etwaige Angreifer Informationen über den internen Aufbau der Infrastruktur erhalten. So geschieht die Interkation aus dem Internet, unabhängig vom angefragten Dienst, immer über den Reverse Proxy. Ein Aufschluss darüber, welcher Server im Hintergrund die Anfrage bedient, oder welche Technologie er verwendet, wird damit erschwert. Damit werden weniger Informationen über mögliche Schwachstellen der Infrastruktur an den potenziellen Angreifer gegeben. Weiterhin führt das Zwischenschalten des Reverse-Proxys zum Minimieren der etwaigen Angriffsfläche. So gilt der NGINX-Proxy aufgrund der langen Erfahrung seiner Entwickler mit der Software [F5, Inc., 2023] als sehr sicher. Die laufenden Verbesserungen der letzten Jahre haben dazu einen wichtigen Beitrag geleistet.

Ein Sicherheitsaspekt, der aktuell durch den Proxy noch nicht adressiert wird, ist die Transport Layer Security (TSL) Verschlüsselung. Zwar ist es grundsätzlich leicht möglich, mit Nginx-Verbindungen zu verschlüsseln und die dafür nötigen Zertifikate zu verwalten [Sysoev, 2023]. Allerdings ist dies nicht für die Domain möglich, die vom Cloudanbieter automatisch zur Verfügung gestellt wird, da die Nutzer dieser häufig wechseln. Aus diesem Grund erfolgt im Beispieleinsatz keine Verschlüsselung auf der Transportebene. Da jedoch über das Frontend keine sensitiven Daten wie Passwörter übertragen werden, kann dieser Umstand als weniger kritisch gesehen werden. In einer produktiven Anwendung wäre es jedoch leicht möglich, ein Zertifikat für die verwendete Domain zu erhalten. Im Zuge dessen wird es damit auch erleichtert, künftig weitere Sicherheitsmaßnahmen zu treffen, da der Proxyserver als Kontaktpunkt zum Internet zentral gehärtet werden könnte.

Sicherheitsaspekte auf Anwendungsebene Im Laufe der Entwicklung wurden bereits erste sicherheitskritische Herausforderungen identifiziert. Eine solche stellt die Wandlung von Nutzereingaben in FFmpeg Befehle dar. Wie bereits beschrieben, können Nutzer über das Webinterface Parameter eingeben, die dann zu Bestandteilen des Wandlungs-Befehls werden. Dieser wird in einer separaten Shell ausgeführt. Dieses Vorgehen bietet Angriffsfläche für „Command-injection-Angriffe“, bei denen durch übergebenen Freitext Zeichenkombinationen insertiert werden, die dann vom System als Befehle aufgefasst werden. Da das System von sich aus nicht in der Lage ist, so eingesetzten Code zu erkennen, werden die Befehle entsprechend ausgeführt [Eilers, 2019, 407]. So könnte durch das Verwenden bestimmter Sonderzeichen ein Teil der Eingabe als gesonderter

Shell-Befehl interpretiert werden. Durch solche Befehle könnte es dann möglich werden, den Server in unzulässiger Weise zu manipulieren. Um dies zu verhindern, wurden verschiedene Gegenmaßnahmen ergriffen. Zum einen wurde der Datentyp, der in die Formularfelder eingegeben werden kann, bereits auf HTML-Ebene eingeschränkt. Damit verbleibt nur das Feld für den Dateinamen der ausgegebenen Datei, in das ohne weiteres schadhafte Zeichenkombinationen eingegeben werden könnten. Um auch diese abzufangen, wird nach dessen Zusammensetzten der FFmpeg-Befehl noch einmal auf Sonderzeichen überprüft, was eine Vielzahl von derartigen Angriffen verhindern kann (ebenda).

Ein weiterer sicherheitsrelevanter Punkt ist der Zugriff der Server per SSH. So ist es auf diese Weise möglich, die Server als Administratornutzer zu verwalten. Um das davon ausgehende Sicherheitsrisiko zu minimieren, wurde eine Reihe von Maßnahmen ergriffen. So wurde zum einen die Anmeldung mittels Passwort deaktiviert. Diese Art der Anmeldung birgt das Risiko, dass durch häufige, automatisierte Anmeldeversuche das Passwort erraten werden kann. Dementsprechend werden in der Konfiguration nur SSH-Verbindungen mittels eines Schlüsselpaars aus öffentlichem und privatem Schlüssel zugelassen. Um den Zugriff zu ermöglichen, wird der öffentliche Schlüssel des Rechners, von dem aus die Infrastruktur erstellt wird, auf dem Proxyserver hinterlegt. Der private Schlüssel wird dann mittels „SSH-Agent“-Schlüsselweiterleitung zur Verfügung gestellt. So wird der Schlüssel nur temporär über den Proxyserver weitergeleitet. Damit wird verhindert, dass der private SSH-Schlüssel auf dem Proxyserver zurückbleibt und eventuell schadhaft eingesetzt werden kann. Auf diese Weise wurde das Vorgehen, nur einen zentralen Kontaktpunkt zu bieten, auch für SSH-Verbindungen angewendet. Die restriktiven Sicherheitsregeln der privaten Netzwerke lassen nur SSH-Verbindungen von innerhalb der Infrastruktur, nicht aber aus dem Internet zu. So müssen auch alle SSH-Verbindungen über den Proxyserver weitergeleitet werden. Durch die Anwendung dieser Maßnahmen konnte ein erfolgreicher Testbetrieb gewährleistet werden. Weitere notwendige Anpassungen am Sicherheitskonzept der Anwendungen, für deren Produktionsreife, werden in Abschnitt 6.2 angedeutet.

5 Messung und Auswertung der gesetzten Anforderungen

Im folgenden Abschnitt werden die in Kapitel 3 aufgestellten Anforderungen überprüft. Dazu wird für Anforderungen, bei denen dies möglich ist, jeweils ein Versuch umrissen. Anhand der Versuchsergebnisse wird so die Quantifizierbarkeit der Anforderungen erreicht. Insgesamt wird für alle gestellten Anforderungen ein Nachweis erbracht und so die Beantwortung der Forschungsfrage gestützt.

5.1 Nachweis der funktionalen Anforderungen

Wie bereits beim Aufstellen der Anforderungen erwähnt, geht es bei den funktionalen Anforderungen darum zu prüfen, ob der angestrebte Funktionsumfang der Software gegeben ist. Dies kann mithilfe verschiedener Wandlungen getestet werden. Betrachtet man anschließend das Ergebnis dieser, so lässt sich durch deren Analyse feststellen, ob die Erwartungen erfüllt werden konnten. Die entsprechenden Wandlungsergebnisse finden sich jeweils in den digitalen Anlagen zu dieser Arbeit.

5.1.1 Nachweis des Transcoding von praxisrelevanten Containern und Codecs

Der Nachweis der verschiedenen Video-Codec- und Containerformate ist leicht möglich, indem die Möglichkeiten für entsprechende Eingaben und korrekte Ausgabe-Dateien aufgezeigt werden. Auf der Präsentationsebene wird die Anforderung durch die entsprechenden Eingabeoptionen der Nutzeroberfläche erfüllt. So kann, wie in Abbildung 5.1 gezeigt, in einem Dropdown das gewünschte Containerformat gewählt werden. Wurde die Auswahl eines Containers vorgenommen, nimmt eine JavaScript-Funktion automatisch eine Filterung der auswählbaren Codecs vor. So können jeweils nur Codecs ausgewählt werden, die mit dem entsprechendem Containerformat kombinierbar sind. Damit kann aus einer entsprechenden Teilmenge der folgenden Codecs gewählt werden.

- H.264
- HEVC
- VP9
- DNxHD
- ProRes

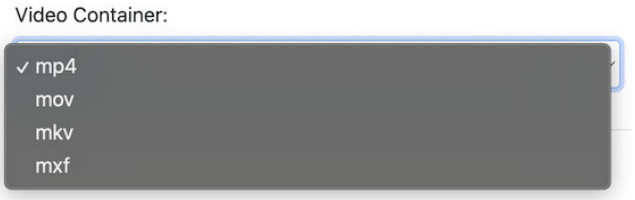

Abbildung 5.1: Dropdown Auswahl Codecs

Weiterhin konnte überprüft werden, dass auch das Wandeln mit einer definierten Bitrate mit der Software umsetzbar ist. Dabei wurde zum Auslesen der Parameter, wie auch bei allen folgenden Umwandlungen, die Software „MediaInfo“ verwendet. Es handelt sich dabei um ein Open-Source-Programm zur Analyse von AV-Dateien [MediaArea.net, 2022]. So konnte geprüft werden, dass die vom Nutzer eingegebene Bitrate sich bei der Ausgabedatei nachweisen lässt. Der vom Nutzer eingegebene Wert bezieht sich dabei nur auf die Videodatenrate. Weiterhin ist zu erwähnen, dass es sich dabei keineswegs um eine konstante Datenrate handelt. Je nach Aufbau der Bildinhalte und abhängig von der Art der Codierung kann die Datenrate innerhalb einer Datei durchaus stark variieren. Zusammenfassend kann für die

unter Kapitel 3 definierten Anforderungen an Wandlungsoptionen gesagt werden, dass diesen durchaus entsprochen wurde. Die entsprechend transcodierten Dateien können in den digitalen Anlagen zur Arbeit im Unterordner „Codecs_Container“ eingesehen werden.

5.1.2 Nachweis der Wandlung von Videoauflösungen

Als eine weitere, leicht zu überprüfende Anforderung an die Software wurde formuliert, dass sie es ermöglichen soll, Videoinhalte zu skalieren. Damit wird die Anpassung an verschiedenartige Wiedergabebedingungen erreicht. Überprüft wurde die Anforderung, indem eine Datei mit der anfänglichen Auflösung von 1920 zu 1080 Pixeln in die Auflösungen 1280×720, 720×480 und 480×360 Pixeln gewandelt wurde. Die Einhaltung dieser Parameter wurde jeweils wiederum mit dem Werkzeug „MediaInfo“ überprüft. Dabei konnten die jeweiligen Parameter bestätigt werden. Die Einhaltung dieser Anforderung kann damit als gegeben gesehen werden. Auch zu diesem Versuch finden sich die entsprechenden in den digitalen Anlagen zur Arbeit. Im Unterordner „Skalierung_Auflösung“ findet sich sowohl die Ausgangsdatei als auch die entsprechenden Ergebnisse.

5.1.3 Nachweis der HDR zu SDR Wandlung

Wie bereits in Unterabschnitt 2.2.1 angedeutet, handelt es sich bei der Wandlung von HDR- zu SDR Inhalten gleichermaßen um ein wegweisendes, wie auch komplexes Thema. Damit sind weder Umsetzung noch Nachweis der Anforderungserfüllung trivial. Im folgenden Beispiel wurde von einer freien, von LG zur Verfügung gestellten Datei ausgegangen, welche unter diesem Link[3] heruntergeladen werden kann. Um die zu übertragenden Datenmenge während des Transcodings zu reduzieren, wurde das Video auf eine Länge von 5s gekürzt. Alle anderen Parameter der Ausgangsdatei blieben dabei unverändert. Anschließend wurde mithilfe des Cloud-Encoders die Wandlung durchgeführt. Die Ein- beziehungsweise Ausgabeparameter können der folgenden Tabelle entnommen werden.

Parameter	Eingabewert	Ausgabewert
Container	mov	mov
Codec	HEVC	HEVC
Profil	Main 10	Main
Video Bitrate	53,5 Mb/s	15,0 Mb/s
Bittiefe	10 bit	8 bit
Farbraum	BT.2020	BT.709
Transferfunktion	PQ	Gammafkt. 2.2
Maximale Leutdichte	1200 nit	100 nit

Tabelle 5.1: Parameter HDR zu SDR Wandlung

Die so gewonnenen Werte zeigen, dass sowohl die Transferfunktion, als auch der Farbraum bei der Wandlung angepasst wurde. Der Parameter der Ausgangsfunktion wurden von einer PQ Funktion in die SDR kompatible Gammafunktion gewandelt, hier mit BT.709 bezeichnet. Wie in Unterabschnitt 2.2.1 bereits beschrieben, wird für eine PQ Funktion die maximale Leuchtdichte des Mastering Displays als ein Eintrag in den Metadaten übertragen. Dieser Wert findet sich in der Tabelle unter dem Eintrag „Maximale Leutdichte“. Der Cloud-Encoder auf der Gegenseite implementiert

[3] https://4kmedia.org/lg-new-york-hdr-uhd-4k-demo/

ebenfalls die Möglichkeit, diesen Wert zu spezifizieren. Dieser wird an den FFmpeg-Videofilter übergeben, der für die Wandlung zuständig ist. Hier wurde als Zielparameter der SDR-typische Wert von 100 Nits gewählt. Wie bereits erwähnt, wurde bei der Wandlung weiterhin der Farbraum der Datei angepasst. Die Werte des erweiterten Farbraums BT.2020 wurden in den SDR kompatiblen Farbraum BT.709 gewandelt. Wie zuvor erwähnt, wird bei diesem Vorgang „Colour Mapping“ notwendig, um Werte außerhalb des kleineren Farbraums abzubilden. Als dritter Parameter wurde auch die Bittiefe der datei auf 8 bit reduziert. Damit soll dem Umstand Rechnung getragen werden, dass dies für die weitere Verwendung des Resultats in Streaming Anwendungen als sinnvoller Parameter erscheint. Die Waveform-Darstellung der Ausgangsdatei kann in Abbildung B.1 eingesehen werden. Die gewandelten Dateien können auch in den digitalen Anlangen zur Arbeit im Unterordner „HDR“ eingesehen werden. Die folgende Testreihe wird die Eigenschaften der von FFmpeg zu diesem Zweck angebotenen Algorithmen zeigen.

Versuch zu „Tone Mapping“ Algorithmen Im Vorfeld der Testreihe wurde die Anforderung gestellt, dass deren Ergebnisse auch ohne HDR Wiedergabegerät nachvollziehbar sein sollen. So ist es eine Besondere Herausforderung, dass davon auszugehen ist, dass nicht in jeder Umgebung, in der die Ergebnisse der Arbeit nachvollzogen werden sollen ein HDR fähiger Monitor zur Verfügung steht. Aus diesem Grund wurde auf eine nähere Auswertung der HDR Ausgangsdatei verzichtet. Um dem zu begegnen werden die Wandlungsergebnisse ausschließlich relativ zueinander vergleichen. Auf diese Weise sind die Ergebnisse in jeder Umgebung nachvollziehbar. Um die Eigenschaften der verschiedenen Algorithmen vergleichbar zu machen, wurden zuerst unter Beibehaltung der spezifizierten Parameter je eine Datei mit jeder der Methoden umgewandelt. Anschließend wurde jeweils der Frame Nummer 50 als PNG Datei exportiert, um sicherzustellen, dass der gleiche Zeitpunkt über alle Dateien hinweg betrachtet wird. Die so erzeugten Standbilder wurden anschließend mithilfe einer Waveform-Monitor-Anwendung untersucht. Dabei handelt es sich um ein Werkzeug, dass ein Bild auf seine Helligkeitswerte untersucht. Dazu wird ein Bild spaltenweise analysiert. Die Helligkeitswerte der n-ten Spalte werden an der entsprechenden horizontalen Position dargestellt. Dazu werden diese auf der y-Achse der Darstellung abgetragen. Die Darstellung der n+1-ten Spalte erfolgt entsprechend rechts daneben. Das zur Anwendung gekommene Werkzeug unterteilt weiterhin die Helligkeitswerte nach den Grundfarben der additiven Farbmischung. Da Helligkeitswerte so entsprechend farbig abgetragen werden, können auch Farbgebungsunterschiede so erkannt werden. Im Versuch wurde das Waveform-Monitor-Werkzeug aus Adobe Premiere Pro 2023 verwendet. Die auf diese Weise entstandenen Gegenüberstellungen der Standbilder und ihrer Waveform-Darstellungen können unter Anhang B eingesehen werden. Die wohl einfachste Methode zur Abbildung der Farben ist das Clipping. Diese Herangehensweise wurde bereits in Unterabschnitt 2.2.1 beschrieben. Die dort angedeuteten Eigenschaften zeigen sich deutlich in der Testdatei unter Abbildung B.2. Hier werden helle Bildbestandteile übersteuert. In der Konsequenz verlieren beispielsweise Wolken oder helle Dächer ihre Zeichnung und erscheinen einfarbig. Dies zeigt sich auch in der Waveform-Darstellung durch die große Menge an dargestellten Maximalwerten im oberen Drittel des Bildes. Aus dem gestiegenem Kontrast resultiert ein übersättigter Bildeindruck. Daneben sorgt die Verschiebung großer Bildteile auf 70% bis 100% der Gesamthelligkeit für eine hohe Gesamthelligkeit. In der Konsequenz wirkt das Bild stark unnatürlich und kann als unbrauchbar bezeichnet werden.

Weniger helle Teile des Bildes zeigt hingegen die Darstellung der Datei, die mittels des „linear“ Algorithmus erstellt wurde. Bei dieser Methode werden alle Werte relativ zum maximalen Helligkeitswert abgebildet [FFmpeg.org, 2022b]. In der Analyse von Abbildung B.3 zeigt sich dabei, dass es nur wenige Werte gibt, die im oberen Drittel der Werteskala abgebildet werden. Durch dieses Vorgehen

verlieren besonders dunkle Bildbereiche an Zeichnung. Weiterhin kommt es zu einer Verschiebung jedes Helligkeitswertes, sodass auch Werte innerhalb des zulässigen Bereiches nicht mehr farbgetreu abgebildet werden.

Um die Farbtreue, im Bereich der zulässigen Farben zu erhöhen, verwendet der „gamma“ Algorithmus eine logarithmische Funktion, um Werte im oberen Helligkeitsbereich so zu stauchen, dass sie im kleineren Farbraum abgebildet werden können [FFmpeg.org, 2022b]. Daraus folgt, dass Mitteltöne weitestgehend farbecht wiedergegeben werden können. Dennoch zeigt sich in Abbildung B.4, dass es zu einer Kumulation von Werten in Richtung des Maximalwerts kommt. Das zeigt sich beispielsweise in der Vollaussteuerung der Blauwerte. Insgesamt lässt sich sagen, dass bei dieser Art der Umwandlung ein deutlich natürlicherer Bildeindruck entsteht, als bei den beiden voran gegangenen Methoden.

Bei der mit „Reinhard“ bezeichneten Methode, welche unter Abbildung B.5 eingesehen werden kann, soll vorrangig die mittlere Bildhelligkeit beibehalten werden [FFmpeg.org, 2022b]. Dafür geht die Methode Kompromisse bei der Farbtreue ein (ebenda). Insgesamt zeigt sich im Waveform-Monitor jedoch auch, dass es zu einer starken Verdichtung in Richtung der helleren Bildbereiche kommt. So zeigt sich im Vergleich zur Gamma-Transferfunktion, dass deutlich mehr Bildbestandteile im Bereich zwischen 80 und 90 % der Maximalhelligkeit angesiedelt sind. Im Ergebnis führt dies zwar zu einem kontrastreichen Bildeindruck, lässt aber helle Details an Zeichnung, also feinen Farb- und Helligkeitsabstufungen verlieren. Dies zeigt sich auch im Waveform-Monitor an den Maximalwerten des linken Bilddrittels.

Als vorletzte Option implementiert der Cloud-Encoder den „Hable“ Tonemapping Algorithmus. Laut der Dokumentation des Filters ist es das erklärte Ziel dieses Filters besonders die hellen und dunklen Details des Bildes beizubehalten. Zu diesem Zweck wird in Kauf genommen, dass die Durchschnittshelligkeit des Bildes unter der eines mit dem „Reinhard“ Algorithmus gewandelten Bildes liegt. In der Dokumentation des verwendeten Filters heißt es dazu weiter, dass größere Kompromisse in Bezug auf die Farbtreue eingegangen werden, als bei der vorherigen Methode [FFmpeg.org, 2022b]. Mithilfe der vorliegenden Messung kann diese Aussage leicht bestätigt werden. So zeigt sich, dass das Bild unter Abbildung B.6 in der Waveform-Darstellung deutlich weniger Bildanteile mit Helligkeitswerten zwischen 90 und 100 % des maximalen Helligkeitswertes aufweist. Weiterhin liegen beispielsweise die Blauwerte, die im Bild den Himmel darstellen, bei 85 bis 90 %. Bei der Umwandlung mithilfe des Reinhard-Algorithmus lagen diese zwischen 95 und 100 % der maximalen Helligkeit.

Der letzte Algorithmus, der aus FFmpeg in die Bedienung des Cloud-Encoders implementiert wurde, wird mit „Mobius“ bezeichnet. Laut der Dokumentation solle dieser besonders farbecht arbeiten. Dazu soll er Farben, die außerhalb des zulässigen Farbraums liegen, unauffällig im Farbraum abbilden [FFmpeg.org, 2022b]. Der Eindruck der farbechten Wiedergabe kann im Beispiel unter Abbildung B.7 nicht bestätigt werden. So resultiert aus der Wandlung ein eher kontrastloses, ungesättigtes Bild. Es zeigt sich, dass zum einen ein maximaler Helligkeitswert von etwa 85 % des Maximalwertes nicht überschritten wird. Zum anderen scheint die Funktion den Schwarzwert stark anzuheben, was dazu führt, dass dieser in nur zwei minimalen Bildbereichen erreicht wird. Ob der Algorithmus nur für das einzelne Testmaterial ungeeignet ist, oder ob es möglich ist, diesen mit weiteren Parametern auf das Material anzupassen, müsste in weiteren Untersuchungen betrachtet werden.

Zusammenfassend kann jedoch gesagt werden, dass die Verwendung unterschiedlicher Tonemapping-Funktionen zu sechs signifikant unterschiedlichen Ergebnissen geführt hat. Welche der Funktionen für den konkreten Anwendungsfall am besten geeignet ist, hängt, neben den Eigenschaften des zu wandelnden Materials, auch von persönlichen Präferenzen ab. Für den durchgeführten Test lässt sich jedoch sagen, dass die Clipping-Methode sich als ungeeignet erwiesen hat. Zu groß waren im Testbild die Bestandteile außerhalb des zulässigen Wertebereichs, die im Ergebnis zum Verlust von Bildinformationen führten. Weiterhin hat sich am Beispiel auch der „Mobius“ Algorithmus als unpassend herausgestellt, da hier das Bild für ein direktes Weiterverwenden über zu wenig Kontrast und Farbsättigung verfügt hätte.

Letztlich ist zu sagen, dass der Test gezeigt hat, dass die Infrastruktur der Anforderung einer HDR zu SDR Wandlung gerecht wird. Mit ihren zahlreichen Anpassungsmöglichkeiten bietet sie weiterhin dem Benutzer die Möglichkeit, die Ausgabe bedarfsgerecht anzupassen. Die erarbeitete Lösung zeigt sich damit auch vor dem Hintergrund dieser Aufgabe mit steigender Relevanz, als zukunftssicher.

5.1.4 Nachweis der Transcoding-Qualität

Um den Nachweis über die Qualität der gewandelten Videodateien zu führen, werden diese den Ergebnissen eines kommerziellen Softwareencoders gegenübergestellt. Als Vergleichslösung wird dazu die Software „Adobe Media Encoder“ in der Version 23.0 verwendet. Zur Gegenüberstellung gebracht werden in diesem Versuch drei verschiedene Umwandlungen. Als Ausgangsmaterial wird für jeden dieser drei Versuchsdurchgänge der von Sony zur Verfügung gestellte Clip „Swordsmith HDR[4]“ verwendet. Dieser wurde zuerst, um klar definierte Ausgangsbedingungen zu schaffen, mithilfe des Adobe Media Encoders auf folgende Ausgangswerte gewandelt:

Parameter	Ausgangswert	Ergebnis I	Ergebnis II	Ergebnis III
Container	mov	mp4	mp4	mp4
Codec	HEVC	H.264	HEVC	ProRes
Profil	Main	Main	Main	Proxy 422
Bitrate	12 Mb/s	5 Mb/s	5 Mb/s	(ca. 56 Mb/s)
Farbraum	Rec709	Rec709	Rec709	Rec709

Tabelle 5.2: Parameter der durchgeführten Umwandlungen

Bei den durchgeführten Wandlungsprozessen wurde der Adobe Media Encoder jeweils auf einem lokalen Rechner ausgeführt. Der Cloud-Encoder wurde in beschriebener Art und Weise mit einem Backend-Server der Größe „t3.large“ betrieben. Auf diese Weise wurde die Ausgangsdatei jeweils mit beiden Transcoding-Lösungen unter Verwendung der in Tabelle 5.2 gezeigten drei Sätzen von Parametern umgewandelt. Im Anschluss wurde das entsprechende Ergebnis mit der Ausgangsdatei verglichen. Zu diesem Zweck konnten die beiden Metriken PSNR und SSIM herangezogen werden. Beide Werte wurden mithilfe eines entsprechenden FFmpeg Filters ermittelt. Die Ergebnisse der mit diesen Parametern durchgeführten Umwandlungen werden im Folgenden gegenübergestellt. Aus Gründen der besseren Übersicht werden dabei jeweils nur die Ergebnisse für die kompletten Frames betrachtet. Die komponentenweise Aufstellung der PSNR- bzw. SSIM-Werte ist Anhang D zu entnehmen. Weiterhin können die gewandelten Dateien in den digitalen Anlagen zur Arbeit im Unterorderner „Qualität“ eingesehen werden. In einer ersten Betrachtung der Werte zeigt sich, dass die

[4] https://4kmedia.org/sony-swordsmith-hdr-uhd-4k-demo/

PSNR	H.264	HEVC	ProRes
FFmpeg Cloud-Encoder	41,87	44,10	48,15
Adobe Media Encoder	40,79	41,85	47,92
Relative Abweichung	2,58%	5,10%	0,48%

Tabelle 5.3: PSNR Werte zwischen der Ausgangsdatei und den gewandelten Videodateien

Durchgänge untereinander die zu erwartende Staffelung aufweisen. Den höchsten Signalrauschabstand weist damit die ProRes transcodierte Datei auf, was sich jedoch auch in einer um den Faktor zwölf größeren Datei niederschlägt. Bei den beiden Dateien, die jeweils mit einer Bitrate von 5Mb/s transcodiert wurden, zeigt sich das gleiche, erwartbare Bild. Jene Datei, die mittels HEVC Codec komprimiert wurde, weist einen etwas höheren Signalrauschabstand auf, als die H.264 komprimierte Datei. Bevor allerdings Vergleiche zum eigentlichen Untersuchungsgegenstand angestellt werden, müssen zusätzlich die Ergebnisse der SSIM Berechnung herangezogen werden.

SSIM	H.264	HEVC	ProRes
FFmpeg Cloud-Encoder	0,9769	0,9840	0,9943
Adobe Media Encoder	0,9733	0,9799	0,9954
Relative Abweichung	0,37%	0,42%	-0,11%

Tabelle 5.4: SSIM Werte zwischen der Ausgangsdatei und den gewandelten Videodateien

So lassen besonders diese Werte Rückschlüsse auf die Qualität der transcodierten Dateien zu [Schmidt, 2021b, S. 268]. In der gemeinsamen Betrachtung der Ergebnisse fällt auf, dass für die Codecs H.264 und HEVC der FFmpeg basierte Encoder leicht bessere Ergebnisse liefert als der Adobe Media Encoder. Ein etwas anderes Bild zeigt sich hingegen beim Rendering von ProRes Dateien. Laut PSNR Wert liegt auch dort der Cloud-Encoder minimal vor dem Adobe Media Encoder. Zieht man jedoch die SSIM Ergebnisse in Betracht, so sprechen diese leicht für letzteren. Dieser Eindruck bestätigt sich, betrachtet man die PSNR Ergebnisse für die einzelnen Komponenten der Datei (siehe Tabelle D.1). So liegt der Signalrauschabstand der Helligkeitskomponente Y bei der ProRes Datei des kommerziellen Anbieters deutlich über dem Vergleichswert der Open Source Lösung. Eine weitere Besonderheit, die bei dieser näheren Betrachtung auffällt ist, dass sowohl bei den PSNR- als auch SSIM-Werten für die V Komponente der HEVC und ProRes Datei der Media Encoder leicht bessere Werte liefert. Da die U- und V- Komponenten die Farbinformationen der Datei einhalten könnte dies als ein Indiz für eine höhere Farbtreue des Adobe Media Encoders gewertet werden. Ein Überblick, über die Relationen der einzelnen SSIM beziehungsweise PSNR Werte kann auch mittels der Tabelle D.3 bzw. Tabelle D.4 gewonnen werden.

In diesem Versuch nicht berücksichtigt wurde die Zeit, die die jeweiligen Encoder für den entsprechenden Wandlungsprozess benötigten. Hier wäre auch keine Vergleichbarkeit gegeben, da der Encoder aus dem Hause Adobe auf einem lokalen Rechner ausgeführt wurde, während FFmpeg nur die recht minimalen Ressourcen einer EC2 t3.large Instanz zur Verfügung standen. Ein weiterer interessanter Umstand ist, dass die Transcoding-Qualität des Cloud-Encoders in geringem Maß vom verwendeten Prozessor des Backend-Servers abhängt. So zeigte sich, dass Prozessoren der t2 Familie minimal bessere Werte liefern, als solche der „t3“ Familie. Um diesen Umstand weiter zu untersuchen, wurde der Versuch mit fünf weiteren Servertypen wiederholt. Zwar wurde bei dieser Untersuchung die Beobachtung, dass verschiedene Servertypen sich auf die Wandlungsqualität auswirken bestätigt, der Betrag der Abweichung blieb jedoch gering. So zeigte sich, dass sowohl für die PSNR als auch die SSIM Metrik eine relative Abweichung von 0,03 % nicht überstiegen wurde.

Aufgrund der sehr geringen Abweichungen ist davon auszugehen, dass dieser Umstand keine Auswirkungen auf die wahrnehmbare Qualität der Dateien hat. Aus diesem Grund wird der Parameter der Transcoding-Qualität auch im Unterabschnitt 5.2.3 nicht in die Überlegung zur Kosteneffizienz einbezogen.

Zusammenfassend lässt sich sagen, dass die Anforderung einer vergleichbaren Qualität der transcodierten Files zwischen dem Cloud-Encoder und der kommerziellen Software Adobe Media Encoder als erfüllt gesehen werden kann. So übertraf die FFmpeg Lösung den lokal ausgeführten Encoder sogar in zwei von drei betrachteten Codecs. Ein wertender Vergleich der beiden Softwaresysteme wird jedoch an dieser Stelle nicht vorgenommen, da dieser nicht wesentlicher Bestandteil der Arbeit ist. Für eine abschließende Bewertung dessen wären weitere Messungen mit einer größeren Spanne von Bitraten notwendig.

5.1.5 Nachweis der Verwendung von Open-Source-Bestandteilen

Als weitere Anforderung an die erstellte Software-Lösung wurde formuliert, dass, mit Ausnahme der durch den Cloudanbieter bereitgestellten Serverinfrastruktur, ausschließlich Open-Source Lösungen zum Einsatz kommen sollen. Um dies zu prüfen, werden im Folgenden die Lizenzen der verwendeten Bestandteile aufgeführt.

Anwendung	Verwendungszweck	Lizenz	Quelle des Codes
FFmpeg	Videoumwandlung	GNU LGPL [FFmpeg Community, 2022]	GitHub[5]
HAProxy	Loadbalancer	GNU GPL [Tarreau, 2006]	GitHub [6]
NGINX	Reverse Proxy	BSD-3-Clause-Lizenz [Rathee and Chobe, 2022]	GitHub[7]
Terraform	Infrastructure as a Code	Mozilla Public License 2.0 [HashiCorp, 2014]	GitHub[8]
Node.js	Web Server	MIT License [Node.js Community, 2022]	GitHub[9]

Tabelle 5.5: Verwendete Open-Source-Bestandteile und deren Lizenzen

Wie der Tabelle zu entnehmen ist, wurden zur Erstellung der Infrastruktur nur Programm-Bestandteile Dritter verwendet, die unter einer der Open-Source-Lizenzen veröffentlicht wurden. Damit ist zum einen gesichert, dass die Verwendung der Software im Zuge der Arbeit zulässig ist, und zum anderen, dass ihr Quellcode einsehbar ist. Zu diesem Zweck wurden die entsprechenden Repositorys in der Tabelle verlinkt.

Ein weiterer Bestandteil der Software sind die Javascript-Pakete, die innerhalb der Node.js Anwendungen Verwendung finden. Auch diese wurden von Dritten erstellt und zur Nutzung überlassen. Im Zuge dessen ist auch ihre Lizenz zu prüfen. Dies geschah mittels eines weiteren Javascript Programms, dass alle Pakete, die in der Anwendung implementiert wurden, auf ihre Lizenzen prüft.

[5] https://github.com/FFmpeg/FFmpeg
[6] https://github.com/haproxy/haproxy
[7] https://github.com/nginx/nginx
[8] https://github.com/hashicorp/terraform
[9] https://github.com/nodejs/node

Zu diesem Zweck werden die in den Paketen enthaltenen Lizenzdokumente analysiert. Die dabei vorgefundenen Lizenzen werden anschließend in zusammengefasster Form ausgegeben. Auf diese Weise wurden neben den in der Tabelle erwähnten, folgende Lizenzen vorgefunden:

- ISC
- Apache - 2.0
- AFL-2.1
- Unlicense

Allen diesen Lizenzen ist gemein, dass sie die Nutzung der Pakete erlauben und sogar für kommerzielle Nutzung zulassen. Auch die verwendeten Java-Script Pakete genügen also den Anforderungen, die im Vorfeld definiert wurden.

Zusammenfassend lässt sich damit sagen, dass die genannten Pakete und Fremdprogramme den Anspruch „Open-Source“ erfüllen. Damit kann die in Kapitel 3 gesetzte Anforderung, in Bezug auf die Verwendung quelloffener-Software, als erfüllt betrachtet werden.

5.2 Nachweis der cloudspezifischen Anforderungen

Nachdem wesentliche funktionale Anforderungen der Infrastruktur überprüft wurden, werden in den folgenden Abschnitten jene aufgestellten Erfordernisse überprüft, die sich unmittelbar auf die Umsetzung des Encoders als Cloud-Infrastruktur beziehen.

5.2.1 Hochverfügbarkeit

Um die Hochverfügbarkeit nachzuweisen, wurden zwei Versuche durchgeführt. Zum einen wurde über eine Laufzeit von 48 Stunden die Verfügbarkeit der Infrastruktur überprüft. Dazu wurde das Uptime-Monitoring Tool „Uptime Kuma“ verwendet. Dabei handelt es sich um ein Open-Source Werkzeug, dass in regelmäßigen Abständen prüft, ob die angegebenen Websites erreichbar sind [Lam, 2021]. Dazu lassen sich mithilfe des Werkzeugs HTTP-Anfragen spezifizieren, die in bestimmten Abständen an IP-Adressen oder Domains gesendet werden (ebenda). Anschließend wird anhand der erhaltenen Antwort festgestellt, ob die Seite verfügbar ist. Weiterhin wird auch die Zeit, die vergeht bis die Anfrage beantwortet wird, aufgezeichnet. Diese Metrik kann als erstes Indiz der Auslastung des Servers gewertet werden.

Langzeittest Um die Verfügbarkeit der Infrastruktur über einen längeren Zeitraum hinweg überprüfen zu können, wurde ein entsprechender Versuch entworfen. Dazu wurde die Infrastruktur, wie in Kapitel 4 dargestellt, ausgerollt. Dabei wurden zwei Backend-Server parametrisiert, da dies die minimale Infrastrukturgröße ist, bei der alle Redundanzfunktionen ausgeschöpft werden. Die laufende Infrastruktur wurde, wie zuvor erwähnt, durch die Überwachungssoftware überprüft. Zu diesem Zweck wurde der Loadbalancer, der Anfragen auf das Backend verteilt, alle 60 Sekunden abgefragt. Mit dieser Abfrage wurde damit indirekt auch die Verfügbarkeit der beiden Backend-Server überprüft. Um in der Versuchsauswertung auch eventuelle Ausfälle eines einzelnen Backend-Servers nachvollziehen zu können, wurden diese zusätzlich einzeln überwacht. Ebenfalls in einem Intervall von 60 Sekunden wurden der Proxy- und Frontend-Server überwacht. Auf den Servern wurden dazu HTTP-Endpoints implementiert, die entsprechende Statuscodes an die Überwachungssoftware lieferten.

Im Falle des Proxys wurden hingegen die Endpoints der dahinterliegenden Server abgefragt. Da dies nicht möglich ist, wenn der Proxy nicht verfügbar wäre, kann auf diese Weise dessen Verfügbarkeit geprüft werden. Damit werden auch Aussagen zur Verfügbarkeit dieser, für den Betrieb der Anwendung essenziellen, Server möglich. In der beschriebenen Art und Weise wurde die Infrastruktur über 48 Stunden hinweg überprüft. Sie wurde in dieser Zeit nur durch die Anfragen der Testsoftware und die Statusabfragen des Loadbalancers belastet. Weitere Umwandlungen fanden in diesem Zeitfenster nicht statt. Diese Testbedingungen wurden gewählt, da sie am leichtesten jederzeit replizierbar sind. So würden bei einem Langzeittest unter Last große Datenmengen anfallen. Diese würden unweigerlich auch erhöhte Kosten mit sich bringen, da der Cloudanbieter für das Speichern und Versenden von Daten Kosten erhebt. Weiterhin wurden, wie bereits angedeutet, jeweils im gleichen Intervall die Antwortzeiten der Server aufgezeichnet.

In der Auswertung des Tests zeigt sich, dass alle überwachten Server über die gesamten 48 Stunden hinweg durchgehend verfügbar waren. So konnten keine Ausfälle dokumentiert werden. In Bezug auf die erfassten Antwortzeiten gab es jedoch zuweilen größere Schwankungen. Der folgenden Tabelle ist, für jeden Server einzeln, dessen maximale Antwortzeit in ms zu entnehmen. Weiterhin wurde auch der Zeitpunkt des Auftretens des Ereignisses festgehalten. Dieser wurde jeweils in Stunden und Minuten nach dem Versuchsbeginn ausgedrückt.

Servername	$\overline{T}$	T_{max}	$t(T_{\text{max}})$
Proxy	5ms	169ms	28h 00min
Frontend	5ms	652ms	5h 48min
Backend Loadbalanced	5ms	50ms	19h 00min
Backend A	4ms	83ms	27h 00min
Backend B	3ms	29ms	2h 39min

Tabelle 5.6: Gemessene Antwortzeiten der verschiedenen Infrastrukturbestandteile

Der Tabelle ist zu entnehmen, dass die erhöhten Antwortzeiten der einzelnen Server zeitlich unabhängig voneinander aufgetreten sind. So traten die Ereignisse über die gesamte Versuchslaufzeit hinweg verteilt auf. Auffällig ist, dass die erhöhten Antwortzeiten vermehrt zu vollen Stunden aufgetreten sind. Eine mögliche Begründung dafür könnten regelmäßig durchgeführte Wartungsaufgaben des Betriebssystems sein. So wäre es denkbar, dass stündlich ausgeführte Hintergrundaufgaben die Server zeitweise belasten, sodass es zu erhöhten Antwortzeiten kommt. Für eine Aussage, ob es sich dabei um eine zufällige Häufung, oder einen systematischen Umstand handelt, ist die Datengrundlage des Versuchs jedoch zu gering. Ein Verhalten, das sich jedoch mit dem durchgeführten Versuch nachvollziehen lässt, ist das erfolgte Loadbalancing. So kam es 27 Stunden nach Versuchsbeginn zu einer erhöhten Antwortzeit des Backend-Servers A. Diese führte jedoch nicht zu einer erhöhten Antwortwortzeit des Loadbalancers. Damit zeigt sich, dass die Anfrage, die an die HA-Proxy-Anwendung gestellt wurde, an den Server gelangte, der keine erhöhte Antwortzeit aufwies. Anderenfalls hätte sich dies in einem neuen Maximalwert der Antwortzeit des Loadbalancers niedergeschlagen. Damit diese Aussage stichhaltig ist, müsste davon ausgegangen werden, dass die Anfragen nahezu zur gleichen Zeit an den Loadbalancer und den Backend-Server gestellt wurde. Davon ist auszugehen, da das Monitoring synchron für alle Server mit dem Start der entsprechenden Anwendung begann und von dort an exakt alle 60s durchgeführt wurde. Es ist jedoch auch zu sagen, dass die Verbesserung der Antwortzeit für den Endpoint durch den vorgesetzten Loadbalancer nicht systematischer Natur ist. Auf die Gründe dafür wird im folgenden Abschnitt näher eingegangen.

Abschließend lässt sich zum Versuch sagen, dass für eine umfängliche Betrachtung der Verfügbarkeit der Infrastruktur weitere Tests mit größerem zeitlichen Umfang sinnvoll wären. Weiterhin wären auch Tests unter verschiedenen Lastszenarien eine mögliche Ergänzung zum durchgeführten Test. Es lässt sich jedoch auch feststellen, dass mit einer Verfügbarkeit von 100% im Testzeitraum der formulierten Anforderung entsprochen wurde. Damit ist ein erster Anhaltspunkt für die Eignung der Infrastruktur zum Dauerbetrieb gegeben.

Fehlertoleranz Eine weitere Dimension der Hochverfügbarkeit ist die Fehlertoleranz. Dieser Begriff wurde bereits in Absatz 3.2 erklärt. Für den Fall der umgesetzten Infrastruktur wurde diese Anforderung schon im Vorfeld nur für die Backend-Server gestellt. Um diese Eigenschaft des Systems zu überprüfen, wurde das System mit zwei Backend-Servern ausgerollt. Anschließend wurde mithilfe des zuvor beschriebenen Uptime-Monitoringtools in Schritten von 20 Sekunden die Verfügbarkeit des Systems geprüft. Antwortete das System binnen maximal 250ms mit einem Statuscode zwischen 200 und 299 so wurde es als verfügbar betrachtet. Konkret wurde jeweils ein HTTP-GET-Endpunkt abgefragt, der vom Reverse Proxy auf den Loadbalancer weitergeleitet wurde. Das detaillierte Vorgehen dazu ist Unterabschnitt 4.2.3 zu entnehmen. Nach einer Zeit von $t_1 = 5min$ wurde einer der beiden Backend-Server abgeschaltet. Anschließend wurde weiter in gleicher Weise die Verfügbarkeit des Systems geprüft. Bei der Versuchsdurchführung konnten die in Abbildung 5.2 gezeigten Antwortzeiten des Systems gemessen werden. Bei der Betrachtung der Antwortzeiten für die beiden

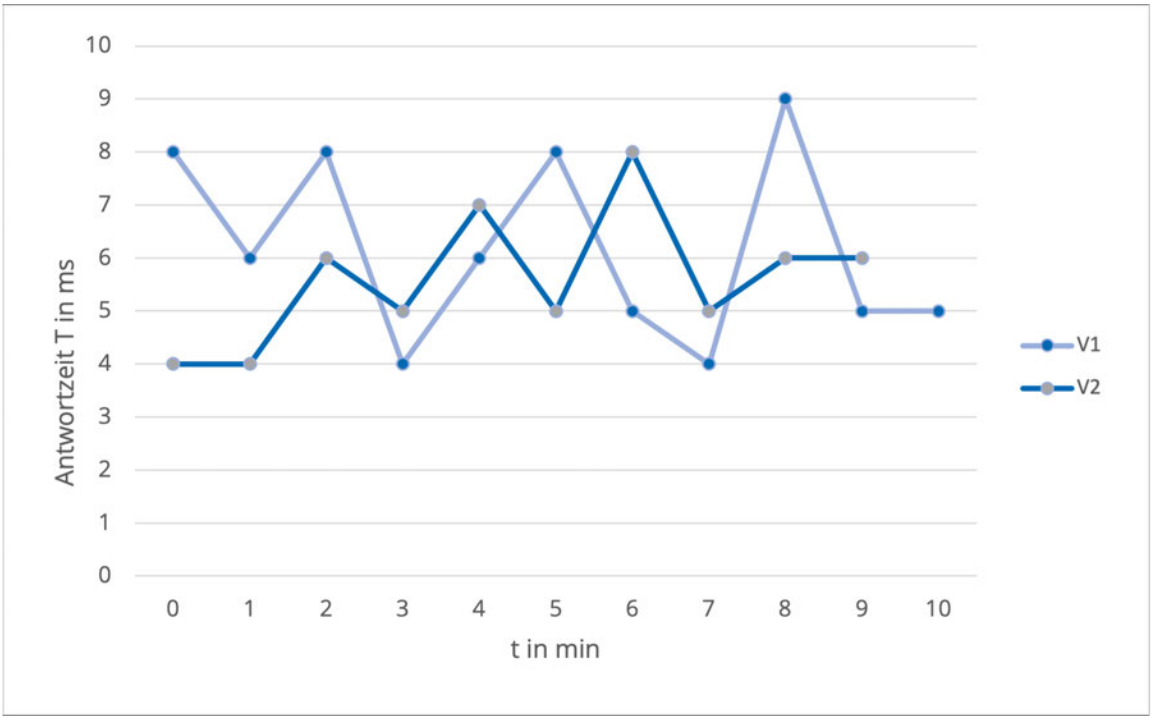

Abbildung 5.2: Gemessene Antwortzeiten beim Abschalten einer Instanz bei $t = 5min$

durchgeführten Messreihen zeigt sich, dass sich am Zeitpunkt t_1 keine signifikanten Änderungen in den Antwortzeiten der Infrastruktur feststellen lassen. Für den gesamten Messzeitraum verhält sich die Antwortzeit als zufallsverteilte Größe.

Die Infrastruktur bleibt also auch beim Abschalten eines Backend-Servers verfügbar. Dies wird wie in Unterabschnitt 4.2.3 angedeutet durch den „Health-Check“ Mechanismus des Loadbalancers erreicht. Zusätzlich werden Backend-Server, wie ebenfalls in Unterabschnitt 4.2.3 erwähnt, beim Hochfahren

der Infrastruktur automatisch abwechselnd auf zwei Verfügbarkeitsbereiche verteilt. Wird also eine Systemkonfiguration mit drei Backend-Servern gestartet, so laufen der erste und dritte Backend-Server in Verfügbarkeitszone A, währende der zweite Server in der Zone B ausgerollt wird. Bei den „Availability Zones“ handelt es sich um jeweils mindestens ein diskretes Rechenzentrum, das über eine redundante Strom- und Netzwerkanbindung verfügt [Amazon Web Services, Inc, 2022d]. Damit kann neben dem Ausfall eines einzelnen virtuellen Servers, auch der Ausfall eines gesamten Rechenzentrums abgefedert werden. Da die Rechenzentren der verschiedenen Verfügbarkeitsbereiche über eine große Entfernung zueinander verfügen, schützt diese Aufteilung auch vor großflächigen Stromausfällen oder Naturkatastrophen.

Unter dem Gesichtspunkt der Hochverfügbarkeit ist zu berücksichtigen, dass bei der umgesetzten Lösung nicht alle Servereinheiten redundant ausgeführt sind. So wurde für die Frontendeinheit und den Proxy auf deren hochverfügbaren Aufbau verzichtet. Dies ist vorrangig dem Ziel geschuldet, die Infrastruktur bei minimalen Kosten testen zu können. So würde die Hochverfügbarkeit aller Bestandteile mindesten zwei weitere virtuelle Server erfordern. Da sich bei ersten Tests jedoch vorrangig die Backend-Server als fehleranfällig erwiesen haben, wurde sich auf diese beschränkt. Weiterhin bietet der redundante Aufbau auch Potenziale in Bezug auf die horizontale Skalierbarkeit, wie der folgende Abschnitt zeigt.

Zusammenfassend kann In Bezug auf die Hochverfügbarkeit des Systems die Anforderungen erfüllt wurden. So zeige sich die Anwendung im Testzeitraum als durchgehendverfügbar. Auch die Fehlertoleranz der reduntant aufgebauten Systembestandteile konnte in diesem Kapitel überptüft werden.

5.2.2 Skalierbarkeit

Eine weitere Eigenschaft von Cloudsystemen, die häufig als wesentliches Charakteristikum genannt wird, ist die Skalierbarkeit. Wie in Kapitel 3 festgelegt, ist das Nachweiskriterium für diese Eigenschaft die Durchlaufzeit bestimmter Files durch den Cloud-Encoder. Um diese festzustellen, wird die Zeit, die vergeht vom Auslösen einer Konvertierungsanfrage bis zum Eingang der Antwort des Backend-Servers gemessen. Diese wird erst dann ausgelöst, wenn die Datei erfolgreich geändert und in den Objektspeicher hochgeladen wurde. Die Messung erfolgt dabei von einer Funktion auf dem Frontend-Server, welche die verstrichene Zeit in Millisekunden misst und inklusive eines Zeitstempels in eine Log-Datei einträgt. Einerseits ist damit ein effektives Mittel zur Quantifizierung von Durchlaufzeiten geschaffen. Andererseits ist in den folgenden Betrachtungen zu berücksichtigen, dass auch die Zeit, welche von der Anfrage und Antwort benötigt wird, um über das Netzwerk zum jeweiligen Zielserver zu gelangen, mit in der festgehaltenen Zeit beinhaltet ist. Daneben beeinflussen auch die Zeiten für Up- und Download der Videodatei aus dem Objektspeicher die Messwerte. Genauere Betrachtungen zur Vorgehensweise und den so erzielten Ergebnissen werden nun in den Unterkapiteln vorgenommen.

Vertikale Skalierbarkeit Eine Teilanforderung an die Infrastruktur stellt die vertikale Skalierbarkeit dar. Um diese zu überprüfen, werden den Backend-Servern verschiedene Mengen an Ressourcen zugewiesen. Dies geschieht über die Verwendung von unterschiedlichen Instanzgrößen. Dabei handelt es sich um eine diskrete Hardwarekonfiguration, die mit einer bestimmten Menge an CPU-Kernen und Arbeitsspeicher eine virtuelle Hardwareumgebung definiert. So wurde der erste Testdurchgang

mittels einer „t2.micro“-Instanz durchgeführt. Deren Hardwarespezifikationen wurden bereits im Unterabschnitt 4.2.1 umrissen. Im zweiten Testdurchgang wurde an selber Stelle eine „t3.large“ Instanz verwendet, die deutlich mehr Ressourcen für den Backend-Server zur Verfügung stellt. Auch diese wurde im Unterabschnitt 4.2.1 bereits beschrieben. Mit beiden Ressourcen wurde der wie folgt spezifizierte Wandlungsprozess durchgeführt. So wurde, wie in den zuvor beschriebenen Messungen zu

Parameter	Ausgangswert	Zielwert
Container	mov	mp4
Codec	HEVC	H.264
Profil	Main	Main
Bitrate	12 Mb/s	5 Mb/s
Farbraum	Rec709	Rec709

Tabelle 5.7: Parameter der durchgeführten Umwandlungen

PSNR und SSIM, als Ausgangsdatei die von Sony zur Verfügung gestellte Datei „Swordsmith HDR“ verwendet. Diese wurde wiederum wie beschrieben zuerst auf eine Bitrate von 12Mb/s und den Rec709 Farbraum gewandelt. Anschließend fand Wandlung der Datei mit beiden Instanzen und den beschriebenen Zielparametern statt. Auf diese Weise wurden folgende Durchlaufzeiten gemessen. Es zeigt sich, dass sich die Durchlaufzeit der Datei mittels vertikaler Skalierung des Backend-Servers

Instanzgröße	Durchgang 1	Durchgang 2	Durchgang 3
t2.micro	302s	331s	297s
t3.large	240s	238s	238s

Tabelle 5.8: Durchlaufzeiten der Datei

tatsächlich verkürzen lässt. So ließ sich die mittlere Durchlaufzeit durch die Skalierung von einer t2.micro auf eine „t3.large“ Instanz von 310s auf 239s verkürzen. Das entspricht einer Reduktion der Bearbeitungszeit um 23%. Die Skalierung würde jedoch auch für eine Steigerung der Kosten für den Backend-Server von 1,34¢ auf 9,6¢ je Stunde sorgen. Die angegebenen Preise beziehen sich dabei jeweils auf die Miete je einer Instanz in einem Rechenzentrum in Zentraleuropa und ohne dass eventuelle Rabatte oder Freikontingente berücksichtigt wurden. Das entspräche einer Steigerung von 616%. Dazu ist zu sagen, dass es sich bei den gewählten Instanzgrößen noch um recht kleine Ressourcenzuordnungen handelt. So würde der absolute Kostenunterschied zwischen den gewählten Größen, bei einem durchgehenden Betrieb des jeweiligen Backend-Servers über einen Monat hinweg, sich ergeben zu: $73,65USD - 13,35USD = 60,30USD$ [Amazon Web Services, Inc, 2022]. Weitere Betrachtungen zur Abwägung der entstehenden Kosten sowie zur vertikalen Skalierung mit weiteren Instanzgrößen sind in Unterabschnitt 5.2.3 angestellt.

Horizontale Skalierbarkeit Als ein Ergebnis zur vertikalen Skalierbarkeit lässt sich schon an dieser Stelle festhalten, dass diese nicht überaus effizient ist. Aus diesem Grund ist es sinnvoll, auch Untersuchungen zur horizontalen Skalierbarkeit des Backends durchzuführen. Um die Durchlaufzeiten auch bei kleinen Instanzgrößen überschaubar zu halten, wurden die in Tabelle 5.9 dargestellten Parameter gewählt:

Parameter	Ausgangswert	Zielwert
Container	mov	mp4
Codec	HEVC	H.264
Profil	Main	Main
Bitrate	4 Mb/s	1 Mb/s
Farbraum	Rec709	Rec709

Tabelle 5.9: Parameter der durchgeführten Umwandlungen

So zeigte sich in vorangegangenen Vorversuchen, dass ressourcenarme Instanzen weniger oft bei Wandlungen mit geringen Bitraten abstürzten. Weiterhin wurden die Bitraten gesenkt, um die Anfallenden-Datenmenge für Speicherung und Transport möglichst gering zu halten. Da diese Aspekte wesentlich die Kosten, die für den Test anfallen, bestimmen, konnte so kostengünstiger getestet werden. Allerdings wird bei diesem Versuch der Anfall einer Reihe von Anfragen an die Infrastruktur simuliert. Zu diesem Zweck wurde der Code des Frontend-Servers leicht angepasst. Die Änderungen sind im Repository zu dieser Arbeit unter dem Branch „serialised Requests“ nachvollziehbar. So wurde im Code eine Funktion hinzugefügt, die beim Eingang einer Anfrage diese in gleicher Weise n-Mal an das Backend weiterleitet. Dabei ist n eine Konstante, die für diesen Versuch als n=3 gewählt wurde. Auf diese Weise ist es möglich, im Benutzerinterface nur eine Anfrage unter Verwendung der zuvor angegebenen Parameter zu stellen, während daraus drei gleichartige Umwandlungen resultieren. Die ebenfalls in Absatz 5.2.2 beschriebene Funktion wurde im Zuge dessen dahingehend angepasst, dass erst beim Erhalt der n-ten Erfolgsmeldung die Durchlaufzeit gestoppt und in das Protokoll eingetragen wird.

Bei allen Versuchen zur horizontalen Skalierbarkeit wurde die Instanzgröße „t3.medium“ verwendet, deren Spezifikationen in Unterabschnitt 4.2.1 bereits beschrieben wurden. Den Anfang der Versuchsreihe bildete das Ausrollen der Infrastruktur mit nur einer Backendinstanz. Da in diesem Fall keine weiteren Instanzen zur Verfügung standen, wurden alle drei Anfragen an den einzigen Backend-Server weitergeleitet, der dann die Anfragen bearbeitete. Im Anschluss wurde das Vorgehen unter Verwendung von zwei beziehungsweise drei Backend-Servern wiederholt. Auf diese Weise konnten die folgenden Durchlaufzeiten gemessen werden:

Instanzanzahl	1x t3.medium	2x t3.medium	3x t3.medium
Durchgang 1	162,1s	109,7s	54,8s
Durchgang 2	161,0s	110,7s	57,0s
Durchgang 3	162,4s	111,4s	56,5s
Mittelwert	161,8s	110,6s	56,1s

Tabelle 5.10: Durchlaufzeiten der Datei

Bei der Betrachtung der Ergebnisse zeigt sich, dass durch das Hinzufügen weiterer Instanzen die Durchlaufzeit einer Gruppe von Anfragen deutlich verringert werden kann. So lässt sich im Versuch, durch das Hinzufügen einer weiteren Instanz jeweils eine Verringerung der Durchlaufzeit um 51,2s bzw. 54,5s beobachten. Damit lässt sich die Durchlaufzeit in erster Näherung durch folgende

Gleichung beschreiben:

$$t_n \approx \begin{cases} t_1 \lceil \frac{m}{n} \rceil, & 1 \leq n \leq m \\ t_1, & n > m \end{cases} \tag{5.1}$$

$$n \in \mathbb{N} \wedge n \neq 0; \quad m \in \mathbb{N} \wedge m \neq 0;$$

Dabei bezeichnet n die Anzahl der verfügbaren Backend-Server und m die Anzahl der gleichzeitig von der gesamten Menge der Backend-Server zu bearbeitenden Anfragen. Weiterhin handelt es sich bei t_1 um die Durchlaufzeit einer einzelnen Anfrage bei $n = 1$ Backend-Server. Die Gültigkeit der Gleichung ist weiterhin nur unter der Bedingung gegeben, dass alle Anfragen und Server gleichartig sind.

Das Verhalten der Infrastruktur ist zu erklären durch die Art, wie Anfragen vom Loadbalancer auf das Backend verteilt werden. So wird eine Anfrage jeweils an den Server verteilt, zu dem die wenigsten Verbindungen bestehen. Unter der Bedingung das gilt $n \geq m$ wird die m-te Anfrage immer an den n-ten Server weitergeleitet, da zu diesem noch keine Verbindung besteht. Gilt nun jedoch $n < m$ so wird die m-te Anfrage wieder an einen Server geleitet werden, der bereits zuvor eine Anfrage erhalten hat. Dies ist der Fall, da alle Server reihum bereits mindestens eine Anfrage erhalten haben. Da insbesondere bei kleineren Instanzen aber schon eine Anfrage ausreicht, um einen Großteil der Hardwareressourcen des Servers zu binden, wird es zwangsläufig z u e iner größeren Durchlaufzeit der Anfragen bei diesem Server kommen. Da für alle $n \in$ die Grenzkosten eines zusätzlichen Servers der gleichen Instanzgröße konstant sind, zeigt sich horizontale Skalierung in diesem Fall als besonders kosteneffizient. W enn a lso g ilt: $m >> n$ k ann m it j eder V erdopplung der Serveranzahl in erster Näherung auch die mittlere Durchlaufzeit halbiert werden. Dabei steigen die Kosten direkt proportional zur Anzahl der Instanzen. Daher rührt die indirekte Proportionalität zwischen der Anzahl von Backend-Servern und der Durchlaufzeit der Anfragen. Auf der anderen Seite zeigt sich, dass wenn $n > m$ gilt, durch das Hinzufügen zusätzlicher Server keine weitere Optimierung der Durchlaufzeit erreicht werden kann. Auch dieser Umstand ist unter Kenntnis der internen Beschaffenheit der Anwendung leicht nachzuvollziehen. So wird eine Anfrage immer nur von genau einer Instanz bearbeitet. Eine Teilung einer Videodatei, um von mehreren Backend-Servern bearbeitet zu werden, erfolgt zum Zeitpunkt der Arbeit nicht.

Zusammenfassung Skalierbarkeit Wie in den vorangegangenen beiden Abschnitten deutlich geworden, erfüllt die Infrastruktur die Anforderungen an beide Arten der Skalierbarkeit. So konnte in den vorangegangenen Versuchen gezeigt werden, dass sich die Durchlaufzeit zum einen durch das Hinzufügen neuer Instanzen, zum anderen aber auch das Erweitern bestehender Instanzen verringern lässt. Dabei ist auch deutlich geworden, dass für eine optimale Verwendung der Infrastruktur vor allem die bedarfsgerechte Kombination beider Herangehensweisen zielführend ist. So lässt sich die horizontale Skalierung in vielen Fällen zwar kostengünstiger als die vertikale Skalierung umsetzen. Allerdings kann auf diese Weise keine Verbesserung mehr erzielt werden, sobald der erste hinzugefügte Server nicht ausgelastet werden kann. Um hier eine weitere Optimierung der Durchlaufzeiten auf diese Weise möglich zu machen, müsste eine Funktion zum Verteilen einer Anfrage auf mehrere Server hinzugefügt werden. In Bezug auf Gleichung 5.1 kann abschließend gesagt werden, dass die vertikale Skalierung maßgeblich den unteren Grenzwert der Durchlaufzeiten einer Gruppe von Anfragen bestimmt. Weiterhin hat sich in den durchgeführten Tests gezeigt, dass die Wahl von zu kleinen Instanzgrößen, also ein gänzlicher Verzicht auf die vertikale Skalierungsebene, erheblich zur Gefahr

von Ausfällen des Systems beiträgt. So ließ sich beobachten, dass der Versuch zur horizontalen Skalierung mit „t2.nano“ Instanzen nicht durchführbar war. Hier kam es zu Abstürzen jener Server, an die zwei oder mehr Anfragen gestellt wurden.

Entsprechend wichtig ist die Analyse des Anforderungsprofils an die Anwendung. So müssten für komplexere Anfragen, die mit hohen Auflösungen oder komplexen Farbraumwandlungen arbeiten, größere Instanzen gewählt werden. Ist hingegen eine große Anzahl weniger anspruchsvoller Umwandlungen charakteristisch für das Anforderungsprofil, ist eine größere Anzahl kleinerer Instanzen zweckmäßig. Abschließend kann jedoch gesagt werden, dass es gerade die Möglichkeit ist, die Anwendung derartig flexibel anzupassen und zu skalieren, die sich im Einsatz als Vorteil erweisen kann.

5.2.3 Kosteneffizienz

Um die Kosten, die das System während des Betriebs verursacht, sinnvoll einordnen zu können, ist es notwendig festzulegen, welche Spezifikation die angemieteten Server haben sollen. Diese bestimmen wesentlich die Gebühren, die vom Cloud-Anbieter erhoben werden. Hier manifestiert sich wieder, dass diese Größe einer genauen Analyse der anfallenden Arbeitslasten bedarf. Problematisch ist dieser Umstand, da zum gegenwärtigen Zeitpunkt verschiedenartige Einsatzmöglichkeiten der Anwendung denkbar sind. Aus diesem Grund wird im Folgenden eine allgemeine Betrachtung verschiedener Servergrößen vorgenommen. Ziel dessen ist es, möglichst allgemeine Aussagen zur Effizienz verschiedener Server zu treffen.

Versuchsbeschreibung Zu diesem Zweck wurde eine gesonderte Versuchsreihe entworfen. Wie in den Versuchen zuvor wurden dabei drei gleichartige Umwandlungen von der Infrastruktur durchgeführt. Wiederum wurde die Zeit gemessen, die vom Zeitpunkt des Einreichens der ersten Anfrage, bis zur vollständigen Erledigung der letzten verging. Dabei wurden für die Eingangs- und Zielparameter die gleichen verwendet, wie sie in Tabelle 5.9 aufgeführt worden sind. Im Verlauf der Versuchsreihe, wurden mit jeder Servergröße je drei Messungen dieser Art durchgeführt. Wie schon in den vorangegangenen Versuchen wurden die Durchlaufzeiten mit einer Timerfunktion auf dem Frontend-Server festgestellt. Die gemessenen Durchlaufzeiten wurden anschließend je Servertyp gemittelt festgehalten. Insgesamt wurden sieben verschiedene Servergrößen untersucht. Dabei wurden zum einen mit den „t3“ Instanzen Prozessoren der x86-Architektur getestet. Zum anderen kamen mit den Instanzen der „c6“ und „t4“ Reihen auch Prozessoren der Arm-Architektur zum Einsatz. Die Messung der verschiedenen Server erfolgte nacheinander, jeweils getrennt durch einen Neustart der Infrastruktur.

Versuchsauswertung Um Rückschlüsse auf die Kosteneffizienz zu ziehen, wurden neben den Durchlaufzeiten auch die Kosten, die für den jeweiligen Servertyp anfallen, festgehalten. Wie schon im vorangegangen Versuch, wurden dabei die stündlichen Kosten für einen Server in der Region „Zentraleuropa“ herangezogen. Rabatte oder Freikontingente fanden keine Berücksichtigung. Neben diesen beiden grundlegenden Metriken ist außerdem eine zusammengesetzte Größe geschaffen worden, die sowohl die Dauer der Berechnung als auch die Kosten des Servers, auf dem sie ausgeführt wurde, berücksichtigt. Mithilfe dieser Vergleichsgröße lassen sich so die verschiedenen Server

gegenüberstellen. Die zusammengesetzte Metrik wird wie folgt berechnet:

$$R = \frac{1}{\frac{T}{T_0}x} \tag{5.2}$$

$$T_0 = 22s$$

Dabei ist T die gemessene Durchlaufzeit und T_0 die mittlere Zeit, die für die Übertragung der Anfragen und die Up- bzw. Downloads der Dateien nötig ist. Die Variable x beschreibt die stündlichen Kosten in US-Dollar, die für den Betrieb eines Servers der jeweiligen Größe anfallen. Das Reziproke des Terms wurde gewählt, da es zur intuitiven Interpretation der Größe beiträgt, wenn höhere Werte einem positiv zu bewertenden Ergebnis entsprechen. Die Messgröße trägt mit Ihrer Gestaltung auch dem Umstand Rechnung, dass sich die Kosten einer Umwandlungsaufgabe aus zwei Faktoren zusammensetzen. Das sind die Kosten, die je Zeiteinheit für den Betrieb des Servers anfallen, und die Zeitspanne, die eine Aufgabe auf dem jeweiligen Server benötigt. So wäre es nicht zielführend, zwar die Kosten des Servers durch kleinere Instanzgrößen zu minimieren, damit aber in Kauf zu nehmen, dass der Server für ein Vielfaches der Zeit in Anspruch genommen werden muss. Die Metrik bildet diese Extremwertüberlegung ab. So zielt eine Optimierung der Durchlaufzeit, indirekt auch auf eine Kostenoptimierung ab. Die Messgröße bildet damit aber noch nicht den Umstand ab, dass bei zeitkritischen Aufgaben eine Übergewichtung des Faktors Zeit durchaus sinnvoll sein kann. Derart zeitgebundene Aufgaben sind dabei auch in der Anwendung durch ein Medienunternehmen leicht denkbar. Aus diesem Grund wurde eine weitere Metrik ausgewertet, die auf eine Betonung des Faktors Zeit setzt. Die Metrik wird wie folgt berechnet:

$$R_{\mathrm{q}} = \frac{1}{(\frac{T}{T_0})^2 x} \tag{5.3}$$

$$T_0 = 22s$$

Da der Faktor Zeit hier quadratisch in die Größe einfließt, wird seiner erhöhten Relevanz bei zeitkritischen Aufgaben Rechnung getragen. Die auf diese Weise berechneten Größen können der Tabelle Tabelle 5.11 entnommen werden. Weiterhin können die gewandelten Dateien in den digitalen Anlagen zur Arbeit im Unterordner „Instanzgrößen" nachgewiesen werden. Es zeigt sich, dass in erster

Instanzgröße	m in $/h	T in s	R in 1/$	R $_{\mathrm{q}}$ in 1/$
t3.medium	0,024	161,84	5,664	0,770
c6g.large	0,078	122,45	2,315	0,416
c6g.xlarge	0,155	61,61	2,301	0,821
t4g.large	0,077	128,18	2,235	0,384
t4g.xlarge	0,154	64,45	2,222	0,759
t3.large	0,096	163,81	1,399	0,188
t3.xlarge	0,192	86,04	1,332	0,341

Tabelle 5.11: Vergleich der Effizienz verschiedener Servergrößen

Näherung die nächst größere Instanz innerhalb einer Prozessorfamilie mit einem verdoppeltem Kostenfaktor und einer halbierten Durchlaufzeit einhergeht. Logische Konsequenz ist, dass die Werte des R Parameters innerhalb der Prozessorfamilien eng beisammen liegen.

Fehlerdiskussion Überraschenderweise zeigt sich in der Tabelle jedoch die „t3.medium“ Instanz laut des Parameters R als am effektivsten. Dieser Umstand überrascht besonders, da die größere „t3.large“ Instanz trotz mehr verfügbarer Ressourcen, länger für die Bearbeitung der gleichen Anfrage benötigt. Dieser Umstand kann mit der Eigenschaft der „t3“ Instanzen als Spitzenlast CPUs erklärt werden. Diese ermöglichen es für einen begrenzten Zeitraum höhere CPU Leistungen abzurufen [Amazon Web Services, Inc, 2022f]. Ist diese Möglichkeit hingegen erschöpft, wird die Leistung der Instanz für einen begrenzten Zeitraum limitiert (ebenda). Das beobachtete Verhalten könnte also darauf zurückgeführt werden, dass die „t3.large“ Instanz zum Zeitpunkt der Anfrage nicht über die Möglichkeit verfügte, Spitzenleistung abzurufen. Der Start der „t3.medium“ bzw. „large“-Instanzen verlief allerdings identisch, sodass es keinen beobachtbaren Grund für verschiedene Spitzenlastfähigkeiten gegeben hat. Auch eine Beobachtung der Systemauslastung des „t3.large“ Servers ergab, dass dieser durchaus bis in den hohen zweistelligen Prozentbereich ausgelastet wird, was gegen eine Leistungsminderung spricht. Analoges gilt für die Schreib- und Lesegeschwindigkeiten der verwendeten Instanz-Hauptspeicher. Auch für diese kann nur eine begrenzte Zeit lang der maximale Datendurchsatz realisiert werden, bevor dieser gedrosselt wird [Amazon Web Services Inc, 2022]. Jedoch waren auch diese für die beiden fraglichen Messungen identisch konfiguriert. Der gleiche Mechanismus greift auch für die Netzwerkdatenrate, mit der die Instanzen angebunden sind. Bei derart kleinen Dateigrößen, wie es im Test jedoch der Fall war, kann dieser Aspekt nicht als Erklärung für das beobachtete Verhalten dienen.

Um zufällige Fehler zu minimieren, wurde die Messung beider Instanzen wiederholt. So wurde der beschriebene Versuch mit einem Abstand von 24 Stunden erneut durchgeführt. Dabei wurde das Ergebnis jedoch bestätigt. Damit kann nicht abschließend geklärt werden, warum dieses Verhalten wiederholt beobachtet wurde. Eine weitere mögliche Erklärung für das beobachtete Verhalten könnte die Eigenschaft der verwendeten Server als virtuelle Maschinen sein. So werden, wie unter Absatz 2.1.1 bereits erwähnt, die einzelnen virtuellen Maschinen nur mittels Virtualisierung auf physischen Maschinen zusammengefasst [Modi, 2019, S. 19]. Aus diesem Grund kann nicht garantiert werden, dass zum Zeitpunkt, in dem eine größere Menge an Hardwareressourcen benötigt wird, diese auch zur Verfügung stehen. Damit können im Vergleich zu dedizierter Hardware durchaus Leistungsbeeinträchtigungen einhergehen [Abidi and Singh, 2013, S. 5]. So kann die Last der Anfragen, die insgesamt an den Cloudprovider gestellt werden, einen Einfluss auf dessen Antwortzeiten haben [Hofmann and Woods, 2010, S. 92]. Üblicherweise sollten derartige Beeinträchtigungen aber nicht über längere Zeiten hinweg bestehen. Aus diesem Grund verwundert es, dass die Beobachtung auch bei einer neuerlichen Versuchsdurchführung Bestand hatte. Es ist also nicht möglich, das Ergebnis vollständig zu erklären.

Ergebnisse Neben den „t3“-Instanzen zeigen sich weiterhin auch die „c6g“ Instanzen als sehr effektiv. Laut R Parameter liegen sie, wenn auch mit deutlichem Abstand, auf dem Rang hinter der „t3.medium“ Instanz. Wird jedoch eine Übergewichtung des zeitlichen Faktors in Betracht bezogen, so erweist sich die „c6.xlarge“ Instanz als besonders effizient. Hinzu kommt, dass es bei diesem Ressourcentyp keine Einschränkung bezüglich der Dauer von hohen Auslastungen gibt. Damit ist es besonders für Anwendungen, in denen häufig aufwändige Aufgaben bearbeitet werden, sinnvoll diese Instanz zu verwenden. Insgesamt lässt sich für den Versuch resümieren, dass die Unterschiede zwischen den einzelnen Prozessoren teils erheblich sind. So weißt die R Metrik eine relative Differenz zwischen ungünstigster und maximal effizienter Ressourcenwahl von 325% auf. Damit zeigt sich die Relevanz der durchgeführten Betrachtung deutlich. Damit jedoch die Effizienzsteigerung durch die Infrastruktur maximiert werden kann, müsste ihr konkreter Anwendungsfall analysiert werden.

So entscheidet darüber, ob eine „t3.medium“, „cg6.large“ oder „cg6.xlarge“ sinnvoll ist, allein die Menge der Anfrage und deren Dringlichkeit. Zu diesem Versuch muss allerdings gesagt werden, dass es sich nicht um eine vollständige Auflistung aller Instanzgrößen handelt. Das Abprüfen aller verfügbaren Servereinheiten würde den Rahmen dieser Arbeit deutlich übersteigen. Die gewonnenen Erkenntnisse können jedoch als Anhaltspunkt, für eine Berechnung der laufenden Kosten der Cloud-Infrastruktur herangezogen werden.

Berechnung der Kosten des Cloud-Encoders Wie schon unter Kapitel 3 beschrieben, hängen die Kosten, die der Cloud-Anbieter für den Betrieb der Infrastruktur erhebt, maßgeblich vom Nutzungsprofil der Anwendung ab. Es ist allerdings sehr schwer, dieses zu verallgemeinern. Aus diesem Grund wurden zum Bestimmen der Kosten folgende Annahmen getroffen: Die Anwendung wird durchgängig betrieben, das heißt nicht nur bei Bedarf gestartet, sondern sie steht dauerhaft zur Verfügung. Für diese Annahme wurde sich entschieden, da es sich bei Unkenntniss der tatsächlichen Ausführungszeit, um die ungünstigst mögliche handelt. Daraus folgt, dass im Folgenden, bei beibehaltung aller sonstigen Parameter, der obere Grenzwert der laufenden Kosten bestimmt wird. Auch Rabatte, die sich ergeben würden, wenn man Recheneinheiten vorab für einen längeren Zeitraum mietet wurden nicht berücksichtigt. Damit soll verhindert werden, dass etwaige Annahmen zu den Kosten sich in der Praxis als unhaltbar erweisen. Weiterhin wird davon ausgegangen, dass die Software von einem begrenzten Nutzerkreis in Anspruch genommen. Das heißt, es wird von der unternehmensinternen Nutzung ausgegangen, die eine Zahl von 200 täglich gewandelten Dateien nicht übersteigt. Aus diesem Grund werden für den Frontend- und Proxyserver die minimalen T2.micro Instanzen gewählt, die sich in den durchgeführten Tests als ausreichend erwiesen haben. Für das Backend wird von 2 Servern der Größe „c6g.large“ ausgegangen. Diese wurden im vorangegangenen Test als praktikabel, auch für den Einsatz unter länger andauernder Belastung, befunden.

Um die Kosten für den S3-Objektspeicher festzulegen, wurde von einem Speicherbedarf von 5 TB ausgegangen. Nach folgendem Überschlag:

$$\frac{5TB * 1024^2}{\frac{220Mb/s}{8}} \hat{=} 53h \tag{5.4}$$

würde der Speicherplatz ausreichen, um 53h Videoinhalte im Produktionscodec ProRes422HQ zu speichern. Weiterhin wird davon ausgegangen, dass monatlich 5000 POST- bzw. LIST-Anfragen an den Speicher gestellt werden. Diese werden verwendet, um neue Objekte zum Objektspeicher hinzuzufügen bzw. die gespeicherten Objekte anzuzeigen. Das entspräche täglich etwas mehr als 160 derartiger Anfragen. Weiterhin wird auch von 5000 GET-Anfragen zum Herunterladen von Objekten aus dem Speicher ausgegangen. Der mit Abstand größte Kostenfaktor des Speichers ist jedoch die Datenmenge, die aus dem Speicher heruntergeladen wird. Anders als für Transaktionen zwischen den Instanzen und dem Objektspeicher fallen für Daten, die von den Instanzen in das Internet gesendet werden, Kosten an. Diese betragen für die ersten 10 TB versendeter Daten 0,09USD pro übertragenem Gigabyte. Danach fallen die kosten leicht degressiv. Für die Berechnung wird angenommen, dass monatlich 2 TB komprimierter Videodateien heruntergeladen werden. Dem Überschlag in Gleichung 5.4 nach würde das bei 8 Mb/s Videodatenrate 582h Videomaterial entsprechen. Damit wäre auch dem Umstand Rechnung getragen, dass aus einer Quelldatei eine Vielzahl von Ausgangsformaten gewandelt wird, die anschließend heruntergeladen werden müssten.

Insgesamt ergeben sich für eine Infrastruktur mit diesen Eigenschaften Kosten von 703,41 USD. Auf ein Jahr gerechnet würde dies Aufwendungen in Höhe von 8.415,96 USD entsprechen. Dabei setzen sich die Kosten wie in Tabelle 5.12 gezeigt zusammen.

Posten	monatliche Kosten
2x „t2.micro" Frontend & Proxy Server	19,56 USD
2x 30 GB Amazon Elastic Block Store (EBS) Instanzspeicher	7,14 USD
2x „c6g.large" Backend Server	113,30 USD
2x 1 TB EBS Instanzspeicher	238,00 USD
S3 Objektspeicher	325,41 USD
Summe	**703,41 USD**

Tabelle 5.12: Kosten der Cloud-Infrastruktur (Quelle: AWS-Preisrechner abgerufen am 8.12.2022)

Für die weiteren verwendeten AWS Dienste fallen keine zusätzlichen Kosten an. Dazu gehören auch die unter Kapitel 4 erwähnten Netzwerkressourcen, deren Nutzung bei Verwendung mit je mindestens einem virtuellen Server kostenlos bleibt. Die genaue Aufstellung der Kosten kann auch unter dierser Adresse[10] abgerufen werden. Dazu muss gesagt werden, dass der tatsächliche Preis, der für den Betrieb der Infrastruktur anfallen würde, mitunter deutlich unter dem oben berechnetem liegen könnte. Ein maßgeblicher Grund dafür ist, dass Cloudanbieter signifikante Rabatte unter bestimmten Bedingungen anbieten [Amazon Web Services, Inc, 2022a]. So können Instanzen, die über einen längeren Zeitraum hinweg angemietet werden, mit Rabatten von bis zu 68% gegenüber dem „On-Demand"-Preis versehen werden (ebenda). Im Gegensatz zum „On-Demand"-Preismodell, wie es für die Beispielrechnung verwendet wurde, bindet sich der Leistungsnehmer dabei jedoch auf einer Vertragslaufzeit von bis zu drei Jahren (ebenda). Preise für den Betrieb der Infrastruktur können damit passend zu den jeweiligen Anforderungen gestaltet werden. So können für Recheneinheiten, die dauerhaft betrieben werden, die vorteilhaften Preise des „Rerserved-Modells" verwendet werden. Instanzen, die jedoch bedarfsgerecht gestartet und gestoppt werden können hingegen von der Flexibilität der „On-Demand"-Preise profitieren. Diese verschiedenen Gestaltungsmöglichkeiten führen jedoch auch dazu, dass die berechneten Preise nur als ein Anhaltspunkt und grober Vergleichswert zu verstehen sind.

Aufstellung der Kosten der „On-Premises"-Umgebung Um einen Vergleich mit einem „vor Ort" betriebenem System zu ermöglichen, wird der vom Cloud-Anbieter Azure angebotene Kostenrechner verwendet. Dieser zieht die total cost of ownership (TCO) Kennzahl heran, um die Kosten, die beim Betreiben einer vergleichbaren Softwarelösung in einer „On-Premises"-Umgebung entstehen würden, zu überschlagen. Das Ziel dieser Methode ist es, die Gesamtkosten, die über den kompletten Lebenszyklus eines IT-Systems entstehen, zu erfassen [Gadatsch, 2021, S. 86 ff.]. Auf diese Weise sollen Aufwendungen für Anschaffung, Betrieb und Entsorgung der Anwendungs-Umgebung berücksichtigt werden (ebenda). Der Azure TCO-Kalkulator vereinfacht diesen Ansatz ein Stück weit. Er führt zuerst die Systemanschaffungskosten an. Diese werden zusammengesetzt aus den Hardwarekosten und dem Installationsaufwand. Weiterhin finden die Kosten für den Betrieb, wie Energie und Wartung einschließlich der dafür benötigten Teile und Arbeitsleistung Berücksichtigung.

[10] https://calculator.aws/#/estimate?id=07510af435be3d225d56e30e6bef8970fd75fd37

Im Zuge der Vergleichbarkeit erschien es nicht als zweckmäßig, Infrastruktur einfach ohne Anpassungen mithilfe des Rechners heranzuziehen. So ist davon auszugehen, dass lokal ausgeführte Videoumwandlungs-Programme in der Regel enger an die Architektur eines PCs oder einer Workstation angelehnt sind. Darum wurde auch die verteilte Systemarchitektur dieser Betrachtung nicht zugrunde gelegt. Stattdessen wurde in diesem Beispiel von folgenden Eckdaten ausgegangen: Es wird ein Hardwareserver mit 8 CPU Kernen und 24 GB RAM angenommen, auf dem ein Linux Betriebssystem ausgeführt wird. Weiterhin wird von einem 5 TB großen SSD Speicher für die Datenhaltung ausgegangen. In Ermangelung von kleineren wählbaren Größen wird von einer Anbindung des Servers mit einer 1 GB Datenleitung ausgegangen. In Bezug auf die Kosten, die für Arbeitsleistungen und weitere Rechencenter Kosten angenommen werden, wurde sich an den Empfehlungen des Anbieters orientiert. Lediglich für die Elektrizitäts-Kosten wurde mit 0,46 USD/kWh ein Wert gewählt, der zum Zeitpunkt der Erstellung der Arbeit realistisch für Energiepreise in Deutschland erscheint. Als Ergebnis gibt die Berechnung einen Wert von 15.934 USD je Jahr an, was monatlichen Kosten von 1202,42 USD entspricht. Dazu ist allerdings zu sagen, dass diese Berechnung auch die Anschaffung der Hardware in voller Höhe auf das erste Betriebsjahr anrechnet, was das Ergebnis verfälscht. Laut [Barroso et al., 2019] beträgt die mittlere Lebensdauer eines Servers in einem Datencenter jedoch drei Jahre. Da man davon ausgehen kann, dass sich der Wertverlust eines Servers gleichmäßig über dessen Lebenszeit verteilt, erscheint es zweckmäßig jährlich nur ein Drittel des Server-Anschaffungspreises von 8.356,80 USD abzuschreiben. Ähnliches gilt für Speichermedien, im Falle der Betrachtung SSD Einheiten, deren Laufzeit in Rechenzentren bis zu 6 Jahre betragen kann [Schroeder et al., 2016]. Damit entfällt nur ein Sechstel der Speicherkosten von 2.048 USD auf jedes Betriebsjahr. Auf diese Weise ergibt sich folgender Wert:

$$15.934\,USD - \frac{2}{3}\,6.964\,USD - \frac{5}{6}\,2.048\,USD = 9584,67\,USD. \tag{5.5}$$

Das würde mittleren monatlichen Kosten von 798 USD entsprechen. Allerdings kann davon ausgegangen werden, dass auch dieser Wert noch keiner vollständigen Betrachtung des Sachverhaltes gerecht wird. So würde das Modell davon ausgehen, dass nach der Lebenszeit des Servers alle Bestandteile dessen entsorgt werden. In der Praxis ist dies natürlich nicht der Fall, da die Hardware auch nach Ablauf ihrer Lebenszeit noch einen Restwert besitzt [Das et al., 2022, S. 351 ff.]. Entsprechend müsste dieser vom Wertverfall subtrahiert werden. Da jedoch schwer vorherzusagen ist, wie hoch dieser Wert ist und ob er sich für den Betreiber des Rechenzentrums tatsächlich realisieren lässt (ebenda), kann dieser Wert schwer beziffert werden. Weiterhin ist die Lebenszeit des Servers eher eine theoretische Annahme, da die Nutzungsdauer einzelner Hardwarebestandteile teils stark differiert. So kann davon ausgegangen werden, dass das Gehäuse oder die Netzteile weniger schnell an Wert verlieren, als Rechen- oder Speichereinheiten. Eine genaue Festschreibung der Kosten, die für den inhouse Betrieb anfallen, ist also nur schwer möglich.

Weiterhin muss gesagt werden, dass Microsoft mit ihrem Dienst Azure, von dem die Berechnungen stammen, natürlich ein Eigeninteresse an den Berechnungsergebnissen hat. Als einer der führenden Anbieter für public Cloud Dienste, kann Microsoft durchaus unterstellt werden, die Kosten für den „On-Premises“-Betrieb eher hoch anzusetzen. Allgemein wird davon ausgegangen, dass derartige Kostenrechner immer nur als Anhaltspunkt für Schätzungen verwendet werden können [Hurwitz, 2020, S. 34]. So wirken die jährlichen Wartungskosten von 20% sicher eher hochgegriffen. Dieser erhöhte Faktor könnte jedoch den Umstand verborgener Kosten in Betracht ziehen. [Gadatsch, 2021, S. 88] spricht so davon, dass Kosten, die nicht direkt kalkulierbar sind, in konservativeren Kostenschätzungen oft nicht beinhaltet sind und Unterhaltskosten damit oft unterschätzt werden. So geht die

Quelle davon aus, dass indirekte Kosten, wie Reputations- oder Kundenzufriedenheitsdefizite durch Systemausfälle, häufig eine Korrektur von „On-Premises“-Kosten nach oben nötig machen. Auch die Arbeitszeit, die verloren gehe, da Mitarbeiter im Umgang mit der Hardware geschult werden müssen, sei laut [Gadatsch, 2021, S. 87 f.] ein Faktor, der häufig unterschätzt werde. Diese Umstände lassen die veranschlagten 20 % Unterhaltskosten zumindest diskutabel erscheinen.

Zusammenfassend lässt sich damit sagen, dass die Infrastruktur durchaus unter Kosten, die mit denen einer „On-Premises“-Umgebung vergleichbar sind, betrieben werden kann. Unter den ausgewählten Parametern ergibt sich bei Verwendung der entwickelten Lösung sogar ein Einsparungspotenzial von 12 %. Diese Kennzahl ist jedoch nicht überzubewerten, da viele Größen, deren tatsächlicher Wert sich nur schwer beziffern lässt, maßgeblich Einfluss nehmen. Eine aussagekräftige Bestimmung von Kostenvorteilen wäre so nur für eng abgegrenzte Anwendungsfälle möglich. Dennoch kann gesagt werden, dass der Anforderung, zu vergleichbaren Kosten den Cloud-Betrieb zu ermöglichen, im dargestellten Beispiel entsprochen wurde. So konnte die Kosteneffizienz der Infrastruktur in der durchgeführten Rechnung dargelegt werden.

6 Zusammenfassung

Nachdem im Vorangegangenen ausführlich die Eigenschaften der Infrastruktur analysiert wurden, werden diese nun zusammengefasst. Dabei ist es zweckmäßig sowohl die funktionalen Anforderungen als auch die cloudspezifischen Anforderungen betrachten, um eine Antwort auf die gestellte Forschungsfrage zu geben. Des Weiteren wird ein Ausblick zu den Möglichkeiten, die Infrastruktur unabhängig von der vorliegenden Arbeit weiterzuentwickeln, erarbeitet.

6.1 Zusammenfassung der Ergebnisse

In Hinblick auf die funktionalen Anforderungen lässt sich sagen, dass diese vollständig erfüllt worden. So ermöglicht es die Infrastruktur, die zuvor festgelegten Codecs zum Wandeln von Video Files zu verwenden. Auch die spezifizierten Containerformate, die unter Abschnitt 3.1 festgelegt wurden, konnten in Testaufgaben verwendet werden. Daneben konnte auch die Auflösungs-Skalierung von Inhalten an Beispielen gezeigt werden. Zusätzlich wurden auch die Möglichkeiten, mit der Software HDR- in SDR-Inhalte umzuwandeln, aufgezeigt. Mit dem Nachweis der Transcoding-Qualität konnte damit abschließend die Anwendbarkeit der Infrastruktur aufgezeigt werden. Auf diese Weise wurde dargelegt, dass die erarbeitete Softwarelösung unter funktionalen Gesichtspunkten mit vorhandenen Video-Umrechnungsprogrammen vergleichbar ist.

Daneben wurde gezeigt, dass der Forschungsgegenstand, welcher die Verwendung von Open-Source-Software Bestandteilen verlangt, erfüllt ist. Zu diesem Zweck wurden die Lizenzen der Drittanbieter-Pakete und Programme analysiert, mit dem Ergebnis, dass die verwendeten Bestandteile der Anforderung in Hinblick auf deren Quelloffenheit entsprechen.

In einem dritten Schritt konnten besondere Eigenschaften, die durch die Ausführung der Infrastruktur in der Cloud ermöglicht werden, aufgezeigt werden. Diese Aspekte liefern die Grundlage für die Beantwortung der gestellten Forschungsfrage. So ließen sich in einem ersten Versuch zwei Dimensionen der Hochverfügbarkeit nachweisen. In einem ersten Langzeitversuch konnte gezeigt werden, dass die Infrastruktur auch über ausgedehntere Zeiträume hinweg fehlerfrei für Anfragen zur Verfügung steht. Damit wurde die Möglichkeit nachgewiesen, die Infrastruktur auch über Tage, durchgängig in Betrieb zu halten und bei Bedarf Dateien mit Ihrer Hilfe zu wandeln. In einem zweiten Versuch konnte die Fehlertoleranz der Infrastruktur nachgewiesen werden. Auf diese Weise wurde dargelegt, dass die Infrastruktur auch im Falle des Ausfalls eines Backend-Servers weiterhin in der Lage ist, Anfragen zu bearbeiten. Im Zuge dessen wurde auch erörtert, dass die Anwendung die Möglichkeit der Verteilung über mehrere Rechenzentren bietet. Es war damit möglich einen Vorteil der Infrastruktur aufzuzeigen, der bei Umsetzung in einem diskreten Rechenzentrum nur mit immensem Aufwand erreichbar wäre. So würde der konventionelle „On-Premises“-Betrieb nur schwerlich die Verteilung der Anwendung über mehrere Regionen ermöglichen. Die Cloud-Architektur der vorliegenden Anwendung hingegen ermöglicht leicht das Verteilen der Backend-Server auf mehrere Verfügbarkeitszonen. Als zusammenfassendes Ergebnis der beiden Tests kann die hohe Verfügbarkeit, auch in Fehlersituationen, als eine nachweisbare Eigenschaft des Cloud-Computings festgehalten werden.

Eine weitere Eigenschaft, die an der Infrastruktur abgeprüft wurde, ist die Skalierbarkeit. So ließ sich in einem ersten Test feststellen, dass die Infrastruktur über die Möglichkeit einer vertikalen Skalierung verfügt. Das Hinzufügen von zusätzlichen Rechenressourcen zu den Backend-Servern zeigte dementsprechend eine deutliche Verkürzung der Durchlaufzeiten. Damit konnte nachgewiesen werden, dass mithilfe der Anpassung von Servergrößen die Zeit, die eine Datei braucht, um von der Infrastruktur gewandelt zu werden, erheblich verkürzt werden kann. In einem zweiten Versuch konnte auch die horizontale Skalierbarkeit der Infrastruktur nachgewiesen werden. Wieder zeigte sich im Versuch eine verkürzte Durchlaufzeit der Dateien nach dem Erweitern der Infrastruktur. Auch eine erste Abwägung zur Anwendung der beiden Skalierungsvarianten konnte so durchgeführt werden. Damit kann die Skalierbarkeit der Infrastruktur als gegeben betrachtet werden. Diese Eigenschaft könnte sich im praktischen Einsatz der Anwendung als relevant zeigen, da auf diese Weise die Abwägung zwischen Kosten und zeitlicher Effizienz der Infrastruktur flexibel gestaltet werden kann. Dieser Aspekt ist von besonderem Interesse, da diese Flexibilität in einer „On-Premises“-Umgebung nicht zu erreichen wäre. Zur Beantwortung der Forschungsfrage kann damit auch die Skalierbarkeit der Anwendung als nachgewiesen betrachtet werden. Damit ist es möglich, den Cloud-Encoder eng an die Anforderungen anzupassen, die ein konkreter Anwendungsfall an seine Verwendung stellt.

Die beiden vorangegangenen Eigenschaften gewinnen besonders an Relevanz, da im Versuch zur Kosteneffizienz der Infrastruktur gezeigt werden konnte, dass die beschriebenen Vorteile bei moderaten Kosten erreicht werden können. So lieferte die Untersuchung unter Absatz 5.2.3 einen Ausblick auf die Kosten, die für den Betrieb der entwickelten Softwarelösung anfallen würden. Es zeigte sich dabei, dass die erarbeitete Infrastruktur auch unter Effizienzgesichtspunkten konkurrenzfähig zu einer vergleichbaren „On-Premises“-Architektur ist. Weiterhin zeigte die Untersuchung, den großen Einfluss der Auswahl effizienter Servergrößen. Die gewonnenen Messwerte legen außerdem nahe, dass bestimmte Prozessorreihen deutlich besser für den getesteten Anwendungsfall geeignet sind. Damit zeigten sich signifikante Einsparpotenziale. Die im Zuge des Versuches verwendete Metrik liefert weiterhin ein Werkzeug zur Überprüfung weiterer Prozessorreihen, um eine weitere Effizienzsteigerung zu ermöglichen. So lässt sich als dritte nachweisbare Eigenschaft die Kosteneffizienz ergänzen. Diese konnte dabei unter Einhaltung der beiden zuvor benannten Eigenschaften aufgezeigt werden, was von besonderem Interesse sein sollte.

6.2 Ausblick auf Möglichkeiten des Ausbaus der Infrastruktur

Um den Gedanken eines iterativen Softwareentwicklungsmodells aufzugreifen, werden im Folgenden Möglichkeiten zur weiteren Verbesserung der Infrastrukturangedeutet. Damit wird zum einen ein Ausblick auf weitere eigene Entwicklungstätigkeit gegeben. Zum anderen wird damit konsequent der Open Source Gedanke aufgegriffen. So steht der Quellcode, der der Arbeit zugrunde liegt, frei zugänglich zur Verfügung. Auf diese Weise ist es auch Dritten möglich, die Software an ihre Bedürfnisse anzupassen. Weiterhin ist es auch denkbar, dass Funktionen, die im Zuge der Arbeit noch nicht implementiert wurden, zu einem späteren Zeitpunkt von Dritten ergänzt werden.

Ein Kritikpunkt, der in Bezug auf die Infrastruktur vorgebracht werden könnte, ist die fehlende horizontale Skalierbarkeit von Frontend– und Proxy-Server. So ist das Hinzufügen weiterer Server dieser Art zwar in der Netzwerkinfrastruktur vorgesehen und auch im Quellcode leicht möglich, wurde jedoch im Zuge der Arbeit nicht umgesetzt. Das ist der Tatsache geschuldet, dass das Testen größerer

Konfigurationen, mit einer größeren Anzahl von Servern, auch die entsprechenden Kosten gesteigert hätte. Durch die entsprechende Konfiguration des bereits vorhandenen Loadbalancers ließe sich jedoch mit geringem Aufwand auch die horizontale Skalierbarkeit des Frontends herstellen.

Ein weiterer Gegenstand eventueller zukünftiger Entwicklungsarbeit sollte die Auflösung der eins zu n Beziehung zwischen Backend-Server und Nutzeranfrage sein. So können aktuell Anfragen nur von genau einem Server bearbeitet werden. In der Umkehrung ist es nicht möglich, dass eine Anfrage auf mehrere Backend-Server verteilt wird. Im Hinblick auf umfangreichere Videodateien kann dies allerdings zum Nachteil werden: Während ein Server mit einer umfangreichen Videoumrechnung beschäftigt ist, die den Server voll auslastet, kann es bei den verbleibenden Backend-Servern zu Leerlauf kommen. In dieser Situation würden die zur Verfügung gestellten Rechenkapazitäten nicht optimal genutzt. Dieser Umstand lässt sich verbessern, indem eine Videodatei in mehrere Teile zerlegt und anschließend auf die Backend-Server verteilt würde. Jeder der qualifizierten Server würde dann für den ihm zugeteilten Bestandteil der Videodatei die Umwandlung vornehmen. Anschließend müsste die Videodatei wieder zusammengefügt und die entsprechenden Container–Header ergänzt werden. Von der Umsetzung dieses Konzeptes wurde im Zuge der Arbeit, aufgrund seiner Komplexität, Abstand genommen. So müsste zur Bearbeitung dieser Anforderung auch eine weiterführende Monitorring-Funktionalität der Server implementiert werden. Dies ist nötig, da die Auslastung der Backend-Server engmaschig überwacht werden müsste, um sicherzustellen, dass nur an Server mit freier Kapazität Teile der zu wandelnden Datei gesendet werden. Der Algorithmus, der aktuell zur Anwendung kommt und auf der Anzahl der Verbindungen beruht, wäre dann nicht mehr funktional. Durch das Verteilen der Anfragen auf mehrere Server würde jeder Server etwa die gleiche Anzahl von aktiven Verbindungen aufweisen. Aufgrund der Komplexität der Anforderung würde deren Umsetzung den Rahmen der Arbeit übersteigen.

Eine Funktion, die ebenfalls Potenzial zum Steigern der Effizienz der Anwendung hätte, wäre das Hinzufügen von zweierlei Caching. So könnte zum einen der Inhalt des Objektspeichers auf dem Frontend-Server zwischengespeichert werden. Damit müsste dieser weniger oft abgefragt werden, was auch Kostenvorteile brächte. Ähnlich könnte auch mit heruntergeladenen Dateien auf dem Backendserver verfahren werden. So könnten diese eine Zeit lang auf dem Server verbleiben, für den Fall, dass weitere Wandlungen mit derselben Datei durchgeführt werden. In diesem Fall würde so das Herunterladen der Datei entfallen. Entsprechend könnte der Datenaustausch zwischen Objektspeicher und Backend-Server reduziert werden.

Eine weitere, ebenfalls umfangreiche Weiterentwicklungsmöglichkeit wäre das Hinzufügen einer Nutzerverwaltung. So würde die Verwendung der vorliegenden Infrastruktur als öffentliche Anwendung das Anlegen von Nutzern notwendig machen. Es wäre dann nicht mehr zweckmäßig, dass alle Benutzer der Anwendung auf die gleichen Dateien zugreifen. Jeder Nutzer sollte in diesem Fall nur die von ihm hochgeladenen Dateien einsehen können. So würde das Umsetzen dieser Gestaltungsidee zwar die Möglichkeit eröffnen, die Infrastruktur auch für die Nutzung über das öffentliche Internet zu öffnen. Andererseits würden damit umfangreiche Sicherheitsanforderungen einhergehen. Weiterhin würde die Verwaltung einer Vielzahl von Nutzern hohe Anforderungen an die Datenhaltung und Skalierbarkeit der Anwendung hinzufügen. Damit würde auch diese Anforderung den Umfang der Arbeit deutlich übersteigen.

6.3 Fazit

Es zeigt sich insgesamt, dass wesentliche Eigenschaften der Cloud, wie sie zum Teil auch theoretisch im Zuge der Arbeit hergeleitet wurden, sich an der Infrastruktur nachweisen lassen. Damit eröffnet die Arbeit einen konkreten Anwendungsfall dieser Art der Infrastrukturbereitstellung für ein Medienunternehmen. Es konnte dargelegt werden, dass auch unter Verwendung von Open-Source Bestandteilen eine Software-Lösung erarbeitet werden konnte, die eine Reihe positiver Eigenschaften nachweisbar macht. Es wird jedoch auch deutlich, dass es noch umfangreiche Möglichkeiten zur Weiterentwicklung der Infrastruktur gibt. Diese bergen das Potenzial, die Eigenschaften der Anwendung voll auszuschöpfen. Zusammenfassend kann die Forschungsfrage damit dahingehend beantwortet werden, dass die Eigenschaften Hochverfügbarkeit, Skalierbarkeit und Kosteneffizienz herausgearbeitet und nachgewiesen werden konnten.

Literaturverzeichnis

[Abidi and Singh, 2013] Abidi, F. and Singh, V. (2013). Cloud servers vs. dedicated servers — a survey. In *2013 IEEE International Conference in MOOC, Innovation and Technology in Education (MITE)*, pages 1–5.

[Adobe Inc., 2022] Adobe Inc. (2022). Choosing the right video format. `https://www.adobe.com/creativecloud/video/discover/best-video-format.html`. Aufgerufen am 21.12.2022.

[Ai et al., 2014] Ai, T., Ali, M. A., Steffan, G., Ovtcharov, K., Zulfiqar, S., and Mann, S. (2014). Real-time hdr video imaging on fpga with compressed comparametric lookup tables. In *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–6.

[Akramullah, 2014] Akramullah, S. (2014). *Video Coding Performance*, pages 161–208. Apress, Berkeley, CA.

[Amazon Web Services, Inc, 2022] Amazon Web Services, Inc (2022). `https://calculator.aws/#/addService/EC2`. Aufgerufen am 30.11.2022.

[Amazon Web Services Inc, 2022] Amazon Web Services Inc (2022). Amazon-ec2-instance-typen. `https://aws.amazon.com/de/ec2/instance-types/`. Aufgerufen am 23.11.2022.

[Amazon Web Services, Inc, 2022a] Amazon Web Services, Inc (2022a). Amazon ec2 reserved instances – preise. `https://aws.amazon.com/de/ec2/pricing/reserved-instances/pricing/`. Aufgerufen am 17.12.2022.

[Amazon Web Services, Inc, 2022b] Amazon Web Services, Inc (2022b). Amazon machine images (ami). `https://docs.aws.amazon.com/de_de/AWSEC2/latest/UserGuide/AMIs.html`. Aufgerufen am 23.11.2022.

[Amazon Web Services, Inc, 2022c] Amazon Web Services, Inc (2022c). Amazon virtual private cloud (amazon vpc). `https://aws.amazon.com/de/vpc/`. Aufgerufen am 23.11.2022.

[Amazon Web Services, Inc, 2022d] Amazon Web Services, Inc (2022d). Regionen und availability zones. `https://aws.amazon.com/de/about-aws/global-infrastructure/regions_az/`. Aufgerufen am 29.11.2022.

[Amazon Web Services, Inc, 2022e] Amazon Web Services, Inc (2022e). Sharing objects using presigned urls. `https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html`. Aufgerufen am 20.11.2022.

[Amazon Web Services, Inc, 2022f] Amazon Web Services, Inc (2022f). Standard mode examples. `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/standard-mode-examples.html#t3_standard_example`. Aufgerufen am 08.12.2022.

[Amazon Web Services, Inc, 2023] Amazon Web Services, Inc (2023). Aws graviton-prozessor. https://aws.amazon.com/de/ec2/graviton/. Aufgerufen am 10.01.2023.

[Andrey et al., 2020] Andrey, N., Jan De, C., Aditya, M., and Anne, A. (2020). More efficient mobile encodes for netflix downloads. https://netflixtechblog.com/more-efficient-mobile-encodes-for-netflix-downloads-625d7b082909. Aufgerufen am 06.01.2023.

[Ankit, 2022] Ankit, S. (2022). Aws-s3-nodejs. https://github.com/voidVic/AWS-S3-NodeJS. Aufgerufen am 22.11.2022.

[Barroso et al., 2019] Barroso, L. A., Hölzle, U., and Ranganathan, P. (2019). *The Datacenter as a Computer*. Springer International Publishing.

[Baun, 2022] Baun, C. (2022). *Betriebssysteme kompakt Grundlagen, Hardware, Speicher, Daten und Dateien, Prozesse und Kommunikation, Virtualisierung*. IT kompakt. Springer Vieweg, 3rd ed. 2022. edition.

[Bengel, 2014] Bengel, G. (2014). *Einführung und Grundlagen*, pages 1–19. Springer Fachmedien Wiesbaden, Wiesbaden.

[Boillat and Legner, 2013] Boillat, T. and Legner, C. (2013). From On-Premise Software to Cloud Services: The Impact of Cloud Computing on Enterprise Software Vendors' Business Models. *Journal of theoretical and applied electronic commerce research*, 8:39 – 58.

[Brightcove Inc, 2022] Brightcove Inc (2022). Q1 2022 global video index. *Global Video Index*. Aufgerufen am 24.11.2022.

[Buser and Imbert, 1992] Buser, P. and Imbert, M. (1992). *Vision*. Bradford Books. MIT Press, London, England.

[Cassia, 2007] Cassia, F. (2007). Open source, the only weapon against "planned obsolescence". https://web.archive.org/web/20110120192512/http://www.theinquirer.net/inquirer/news/1001739/open-source-weapon-planned-obsolescence. Aufgerufen am 05.12.2022.

[Christudas, 2019] Christudas, B. (2019). *Install, Configure, and Run Nginx Reverse Proxy*, pages 843–846. Apress, Berkeley, CA.

[Daly et al., 2013] Daly, S., Kunkel, T., Sun, X., Farrell, S., and Crum, P. (2013). Viewer preferences for shadow, diffuse, specular, and emissive luminance limits of high dynamic range displays. *SID Symposium Digest of Technical Papers*, 44(1):563–566.

[Das et al., 2022] Das, K., Saha, S., Chowdhury, S., Reza, A. W., Paul, S., and Arefin, M. S. (2022). A sustainable e-waste management system and recycling trade for bangladesh in green it. In Vasant, P., Weber, G.-W., Marmolejo-Saucedo, J. A., Munapo, E., and Thomas, J. J., editors, *Intelligent Computing & Optimization*, pages 351–360, Cham. Springer International Publishing.

[De, 2017] De, B. (2017). *Designing a RESTful API Interface*. Apress, Berkeley, CA.

[Du and Nie, 2011] Du, J. and Nie, G. (2011). Design and implementation of security reverse data proxy server based on ssl. In Zhu, M., editor, *Information and Management Engineering*, pages 523–528, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Eilers, 2019] Eilers, C. (2019). *You've been hacked! alles über Exploits gegen Webanwendungen*. Rheinwerk Computing, 1. auflage edition.

[Erl and Puttini, 2013] Erl, T. and Puttini, Ricardoand Mahmood, Z. (2013). *Cloud computing: Concepts, Technology & Architecture*. Prentice Hall, Philadelphia, PA.

[F5, Inc., 2023] F5, Inc. (2023). What is nginx? `http://nginx.org/en/docs/http/configuring_https_servers.html`. Aufgerufen am 15.01.2023.

[Familiar, 2015] Familiar, B. (2015). *What Is a Microservice?*, pages 9–19. Apress, Berkeley, CA.

[FFmpeg Community, 2022] FFmpeg Community (2022). Ffmpeg/license.md. `https://github.com/FFmpeg/FFmpeg/blob/master/LICENSE.md`. Aufgerufen am 06.12.2022.

[FFmpeg.org, 2022a] FFmpeg.org (2022a). Fabout ffmpeg. `https://ffmpeg.org/about.html`. Aufgerufen am 05.12.2022.

[FFmpeg.org, 2022b] FFmpeg.org (2022b). Ffmpeg filters documentation. `https://ffmpeg.org/ffmpeg-filters.html#tonemap-1`. Aufgerufen am 01.12.2022.

[Fischer, 2016] Fischer, W. (2016). *DVB-S/S2 - Messtechnik*, pages 359–374. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Flynn et al., 2013] Flynn, J. R., Ward, S., Abich, J., and Poole, D. (2013). Image quality assessment using the ssim and the just noticeable difference paradigm. In Harris, D., editor, *Engineering Psychology and Cognitive Ergonomics. Understanding Human Cognition*, pages 23–30, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Frank, 2019] Frank, R. (2019). *Cloud-Transformation wie die Public Cloud Unternehmen verändert*. Springer eBooks Business and Economics. Springer Gabler.

[Free Software Foundation, Inc., 2000] Free Software Foundation, Inc. (2000). Gnu lesser general public license. `https://www.gnu.de/documents/lgpl-2.1.de.html`. Aufgerufen am 06.12.2022.

[Free Software Foundation, Inc., 2007] Free Software Foundation, Inc. (2007). Gnu general public license 3. `https://www.gnu.org/licenses/gpl-3.0`. Aufgerufen am 05.12.2022.

[Free Software Foundation, Inc., 2022] Free Software Foundation, Inc. (2022). Warum man die lesser gpl nicht für die nächste bibliothek verwenden sollte. https://www.gnu.org/licenses/why-not-lgpl.de.html#:~:text=Die%20Wahl%20der%20Lizenz%20macht,ausschlie%C3%9Flich%20f%C3%BCr%20freie%20Programme%20nutzbar. Aufgerufen am 06.12.2022.

[Gadatsch, 2021] Gadatsch, A. (2021). *IT-Investitionsrechnung und Total Cost of Ownership-Analyse*, pages 75–93. Springer Fachmedien Wiesbaden, Wiesbaden.

[Gentemann et al., 2021] Gentemann, L., Termer, D. F., and Weber, D. A. (2021). Open-Source-Monitor Studienbericht 2021. *Open-Source-Monitor*, 2.

[Gulabani, 2017] Gulabani, S. (2017). *Hands-on Simple Storage Service (S3)*, pages 223–307. Apress, Berkeley, CA.

[Gulabani, 2018] Gulabani, S. (2018). *Amazon Web Services Bootcamp Develop a scalable, reliable, and highly available cloud environment with AWS*. Packt Publishing, Birmingham, UK.

[HashiCorp, 2014] HashiCorp, I. (2014). terraform/license. https://github.com/hashicorp/terraform/blob/main/LICENSE. Aufgerufen am 06.12.2022.

[Hofmann and Woods, 2010] Hofmann, P. and Woods, D. (2010). Cloud computing: The limits of public clouds for business applications. *IEEE Internet Computing*, 14(6):90–93.

[Hong and Kim, 2002] Hong, T. and Kim, Y. (2002). Optimal buffering strategy for streaming service in time varying wireless environment. In Chang, W., editor, *Advanced Internet Services and Applications*, pages 115–123, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Hsu, 2018] Hsu, T. (2018). *Hands-On Security in DevOps*. Packt Publishing, Birmingham, UK.

[Hurwitz, 2020] Hurwitz, J. S. (2020). *Cloud computing for dummies, second edition*. John Wiley & Sons, Nashville, TN, 2 edition.

[Ihrig, 2013] Ihrig, C. J. (2013). *Executing Code*, pages 129–145. Apress, Berkeley, CA.

[Initiative, 2022a] Initiative, O. S. (2022a). About the open source initiative. https://opensource.org/about. Aufgerufen am 05.12.2022.

[Initiative, 2022b] Initiative, O. S. (2022b). The open source definition. https://opensource.org/docs/osd. Aufgerufen am 05.12.2022.

[International Telecommunication Union, 2018] International Telecommunication Union (2018). Recommendation itu-r bt.2100-2: Image parameter values for high dynamic range television for use in production and international programme exchange. https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.2100-2-201807-I!!PDF-Ew.pdf. Aufgerufen am 25.11.2022.

[Iza Paredes et al., 2017] Iza Paredes, C., Mezher, A. M., and Aguilar Igartua, M. (2017). Performance comparison of h.265/hevc, h.264/avc and vp9 encoders in video dissemination over vanets. In Gaggi, O., Manzoni, P., Palazzi, C., Bujari, A., and Marquez-Barja, J. M., editors, *Smart Objects and Technologies for Social Good*, pages 51–60, Cham. Springer International Publishing.

[Janus et al., 2012] Janus, S., Chen, J., Cranton, W., and Fihn, M. (2012). *Video Compression*, pages 287–300. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Joachim and Kevin, 2022] Joachim, H. and Kevin, K. (2022). Amazon, google, microsoft: Cloudriesen machen den chipherstellern konkurrenz. `https://www.handelsblatt.com/technik/it-internet/technologie-amazon-google-microsoft-cloud-riesen-machen-den-chipherstellern-konkurrenz/27976160.html`. Aufgerufen am 04.01.2023.

[Kashiwa et al., 2019] Kashiwa, Y., Ihara, A., and Ohira, M. (2019). What are the perception gaps between floss developers and se researchers? In Bordeleau, F., Sillitti, A., Meirelles, P., and Lenarduzzi, V., editors, *Open Source Systems*, pages 44–57, Cham. Springer International Publishing.

[Kerschbaumer, 2022] Kerschbaumer, K. (2022). Beijing 2022: Virtualized production of curling competition hints at future. `https://www.sportsvideo.org/2022/02/09/beijing-2022-virtualized-production-of-curling-competition-hints-at-future/`. Aufgerufen am 15.01.2023.

[Kirkbride, 2020] Kirkbride, P. (2020). *systemd*, pages 221–234. Apress, Berkeley, CA.

[KPMG AG, 2022] KPMG AG (2022). Cloud-monitor 2022. *KPMG Cloud-Monitor*.

[Krief and Hashimoto, 2020] Krief, M. and Hashimoto, M. (2020). *Terraform Cookbook*. Packt Publishing, Birmingham, UK.

[Krypczyk and Bochkor, 2020] Krypczyk, V. and Bochkor, O. (2020). *Handbuch für Softwareentwickler*. Rheinwerk Computing. Rheinwerk Verlag, Bonn, 1. auflage, 1., korrigierter nachdruck edition.

[Kurzweil, 2004] Kurzweil, R. (2004). *The Law of Accelerating Returns*, pages 381–416. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Kutschbach, 2019] Kutschbach, P. (2019). HDR Farbraumtransformationen Transformationsalgorithmus. *FKT-Fernseh-und Kinotechnik*, 2019.

[Lam, 2021] Lam, L. (2021). Uptime kuma. `https://github.com/louislam/uptime-kuma`. Aufgerufen am 29.11.2022.

[Lampe, 2010] Lampe, F., editor (2010). *Green-IT, Virtualisierung und Thin Clients: Mit neuen IT-Technologien Energieeffizienz erreichen, die Umwelt schonen und Kosten sparen ; mit 32 Tabellen*. Praxis IT-Management und -Anwendung. Vieweg + Teubner, Wiesbaden, 1. aufl. edition.

[Ledin, 2022] Ledin, J. (2022). *Modern Computer Architecture and Organization Learn x86, ARM, and RISC-V architectures and the design of smartphones, PCs, and cloud servers*. Packt Publishing, 2. edition.

[Li et al., 2014] Li, Z.-N., Drew, M. S., and Liu, J. (2014). *Lossless Compression Algorithms*, pages 185–224. Springer International Publishing, Cham.

[Lisdorf, 2021] Lisdorf, A. (2021). *Cloud Computing Basics*. Apress, Berkeley, CA.

[Luntovskyy and Spillner, 2017] Luntovskyy, A. and Spillner, J. (2017). *Cloud Computing, Virtualisation, Storage and Networking*, pages 77–133. Springer Fachmedien Wiesbaden, Wiesbaden.

[Maayan, 2020] Maayan, G. (2020). 8 best video file formats for 2020. `https://www.computer.org/publications/tech-news/trends/8-best-video-file-formats-for-2020`. Aufgerufen am 28.11.2022.

[Mack, 2011] Mack, C. A. (2011). Fifty years of moore's law. *IEEE Transactions on Semiconductor Manufacturing*, 24(2):202–207.

[Mariana et al., 2020] Mariana, A., Anush, M., Liwei, G., Lishan, Z., and Anne, A. (2020). Improving our video encodes for legacy devices. `https://netflixtechblog.com/improving-our-video-encodes-for-legacy-devices-2b6b56eec5c9`. Aufgerufen am 06.01.2023.

[MediaArea.net, 2022] MediaArea.net (2022). Mediainfo. `https://mediaarea.net/de/MediaInfo`. Aufgerufen am 23.11.2022.

[Mell and Grance, 2011] Mell, P. and Grance, T. (2011). Sp 800-145: The nist definition of cloud computing. `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf`. Aufgerufen am 04.01.2023.

[Melo et al., 2016] Melo, M., Barbosa, L., Bessa, M., Debattista, K., and Chalmers, A. (2016). Context-aware HDR video distribution for mobile devices. *Multimedia Tools and Applications*, 76(15):16605–16623.

[Membrey et al., 2012] Membrey, P., Hows, D., and Plugge, E. (2012). *Network Load Balancing*, pages 153–174. Apress, Berkeley, CA.

[Michael et al., 2007] Michael, M., Moreira, J. E., Shiloach, D., and Wisniewski, R. W. (2007). Scale-up x scale-out: A case study using nutch/lucene. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8.

[Modi, 2019] Modi, R. (2019). *Azure for Architects Implementing cloud design, DevOps, containers, IoT, and serverless solutions on your public cloud*. Packt Publishing, 2. edition.

[Nadon, 2017] Nadon, J. (2017). *EC2 Resources at Scale*, pages 189–202. Apress, Berkeley, CA.

[Netflix Services Germany GmbH, 2022] Netflix Services Germany GmbH (2022). Netflix in dolby vision oder hdr10 streamen. https://help.netflix.com/de/node/42384. Aufgerufen am 05.12.2022.

[Node.js Community, 2022] Node.js Community (2022). node/license. https://github.com/nodejs/node/blob/main/LICENSE. Aufgerufen am 06.12.2022.

[Ohm, 2015] Ohm, J.-R. (2015). *Multimedia signal coding and transmission*. Signals and communication technology. Springer Berlin, Heidelberg.

[Pattanayak et al., 2021] Pattanayak, B. K., Hota, N., and Mishra, J. P. (2021). *A Systematic Overview of Fault Tolerance in Cloud Computing*. Smart Innovation, Systems and Technologies 153. Springer, Singapore, 1st ed. 2021. edition.

[Pereira et al., 2010] Pereira, R., Azambuja, M., Breitman, K., and Endler, M. (2010). An architecture for distributed high performance video processing in the cloud. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 482–489. IEEE.

[Pfitzinger and Jestädt, 2016] Pfitzinger, B. and Jestädt, T. (2016). *IT-Betrieb*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Rao et al., 2014] Rao, K., Kim, D. N., and Hwang, J. J. (2014). *Video coding standards AVS China, H.264/MPEG-4 PART 10, HEVC, VP6, DIRAC and VC-1*. Signals and Communication Technology. Springer, Dordrecht, 1st ed. 2014. edition.

[Rathee and Chobe, 2022] Rathee, S. and Chobe, A. (2022). *Open Source Licenses*, pages 37–56. Apress, Berkeley, CA.

[Rekhter et al., 2022] Rekhter, Y., Moskowitz, R. G., Karrenberg, D., Jan de Groot, G., and Lear, E. (2022). Rfc.1996: Address allocation for private internets.

[Reznik et al., 2021] Reznik, Y., Cenzano, J., and Zhang, B. (2021). Transitioning broadcast to cloud. *Applied Sciences*, 11(2):503.

[Richardson, 2010] Richardson, I. E. G. (2010). *The H.264 advanced video compression standard*. John Wiley and Sons, Ltd, second edition edition.

[Rogowsky, 2014] Rogowsky, M. (2014). Analyse zur sicherheit von open-source software. *Lehrstuhl Netzarchitekturen und Netzdienste*.

[Salkosuo, 2020] Salkosuo, S. (2020). Ffmpeg api. https://github.com/samisalkosuo/ffmpeg-api/tree/master. Aufgerufen am 16.12.2022.

[Schatten et al., 2010] Schatten, A., Demolsky, M., Winkler, D., Biffl, Stefan andGostischa-Franta, E., and Östreicher, T. (2010). *Best Practice Software-Engineering*. Spektrum Akademischer Verlag, Heidelberg.

[Schmidt, 2021a] Schmidt, U. (2021a). *Professionelle Videotechnik*. Springer Berlin Heidelberg.

[Schmidt, 2021b] Schmidt, U. (2021b). *Professionelle Videotechnik*. Springer Berlin Heidelberg.

[Schroeder et al., 2016] Schroeder, B., Lagisetty, R., and Merchant, A. (2016). Flash reliability in the field: The expected and the unexpected. In *Usenix FAST 2016*.

[Shute, 2019] Shute, Z. (2019). *Speed up web development with the powerful features and benefits of JavaScript*. Packt Publishing, Birmingham, UK.

[Steinicke, 2016] Steinicke, F. (2016). *More and More, and More than Moore's Law*, pages 127–143. Springer International Publishing, Cham.

[Synopsys, Inc, 2022] Synopsys, Inc (2022). Fffmpeg. `https://www.openhub.net/p/ffmpeg`. Aufgerufen am 06.12.2022.

[Sysoev, 2023] Sysoev, I. (2023). Configuring https servers. `https://www.nginx.com/resources/glossary/nginx/`. Aufgerufen am 15.01.2023.

[Tarreau, 2006] Tarreau, W. (2006). Haproxy's license. `https://www.haproxy.org/download/1.3/doc/LICENSE`. Aufgerufen am 06.12.2022.

[The fluent-ffmpeg contributors, 2021] The fluent-ffmpeg contributors (2021). node-fluent-ffmpeg. `https://github.com/fluent-ffmpeg/node-fluent-ffmpeg/commits/master`. Aufgerufen am 16.12.2022.

[video Clarity, Inc., 2016] video Clarity, Inc. (2016). Understanding mos, jnd, and psnr. *A Video Clarity White Paper*.

[video Clarity, Inc., annt] video Clarity, Inc. (Veröffentlichungsjahr unbekannt). White paper – advancing to multi-scale ssim. *A Video Clarity White Paper*.

[Wang and Kim, 2019] Wang, C. and Kim, H. (2019). Touchdown on the cloud: The impact of the super bowl on cloud. In *2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 107–113. IEEE.

[Wang et al., 2004] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

[Zadka, 2022] Zadka, M. (2022). *Terraform*, pages 225–230. Apress, Berkeley, CA.

[Zammetti, 2013] Zammetti, F. (2013). *Pro iOS and Android Apps for Business with jQuery Mobile, node.js, and MongoDB*. Apress, Berkeley, CA.

[Zammetti, 2020] Zammetti, F. (2020). *Modern Full-Stack Development*. Apress, Pottstown, PA, USA.

[Zhang et al., 2007] Zhang, L.-J., van der Aalst, W., and Hung, P. C. K. (2007). *2007 IEEE International Conference on Services Computing: (SCC 2007) proceedings Salt Lake City, Utah, USA, July 9-13, 2007*. IEEE Computer Society, Las Alamitos Calif.

Anhang A: Anhang zum Theorieteil

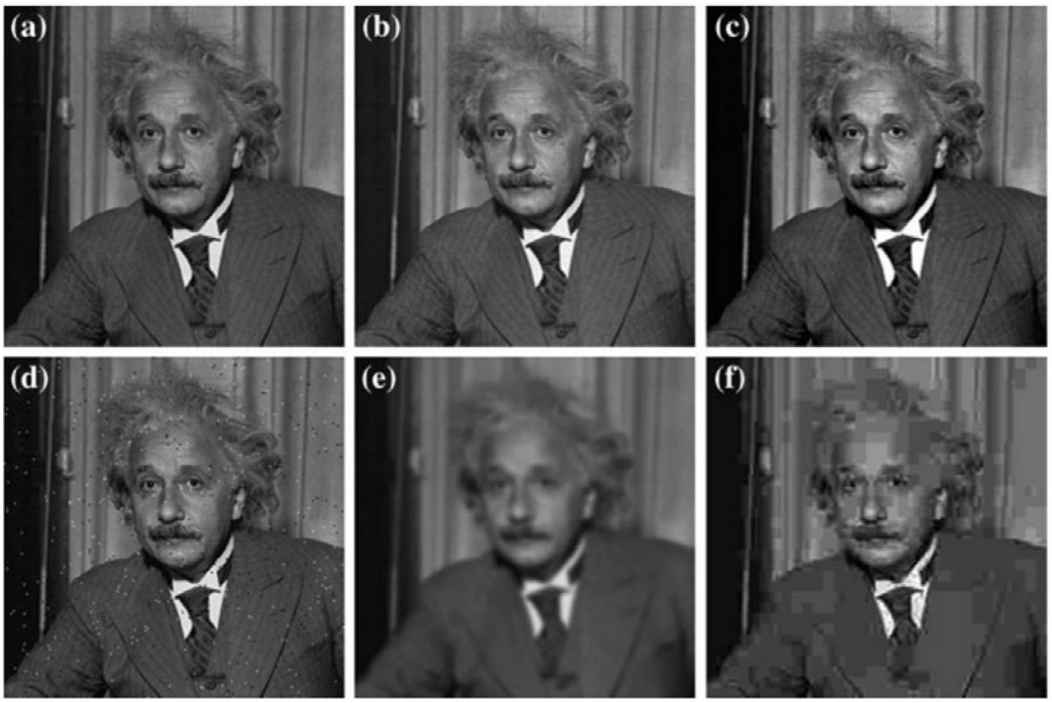

Abbildung A.1: Verschiedene Bildfehler, die den gleichen MSE Wert erzeugen [Rao et al., 2014, S.275]

Anhang B: HDR SDR Wandlungsergebnisse

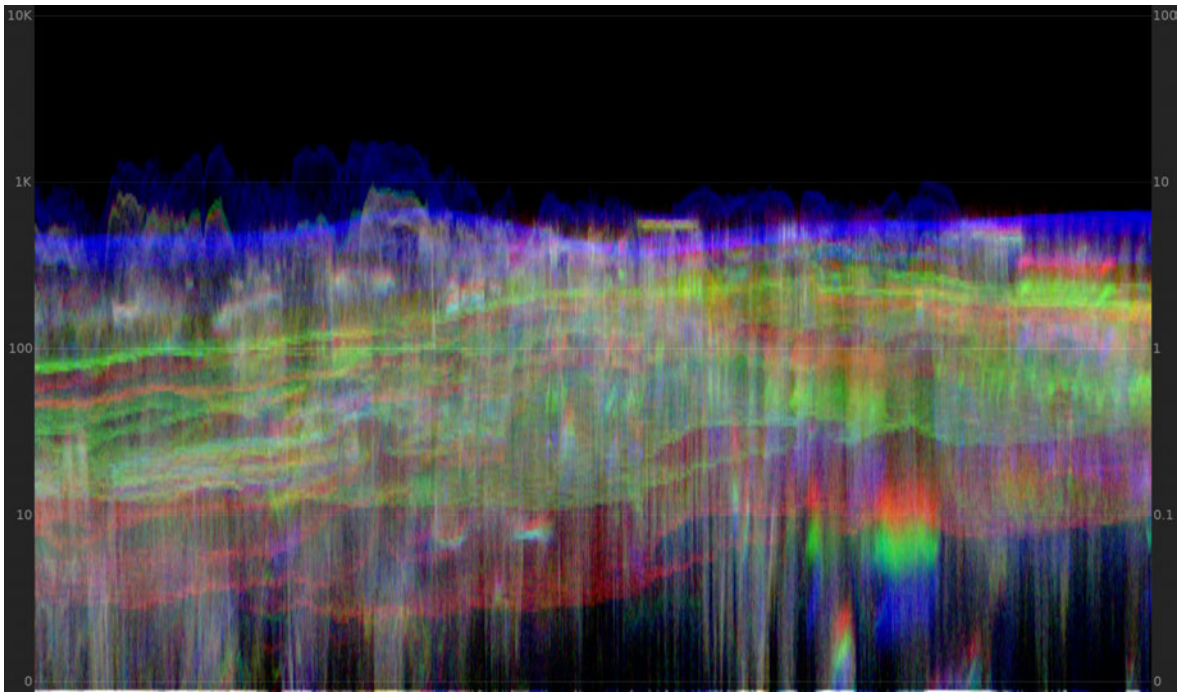

Abbildung B.1: Waveform-Monitor des Frames 50 der Ausgangsdatei

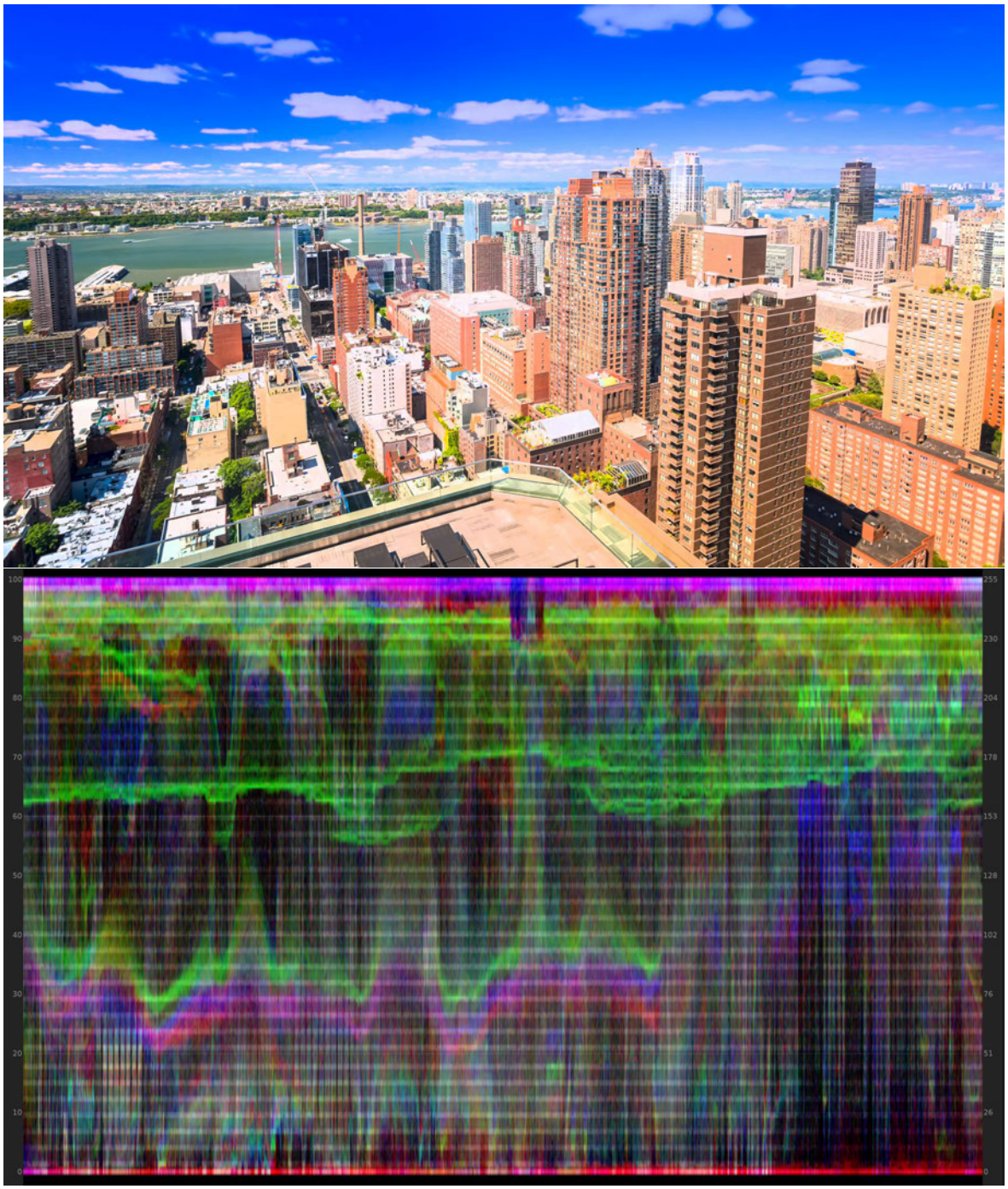

Abbildung B.2: Tonemapping „Clip“ Frame 50 mit entsprechendem Waveformmonitor

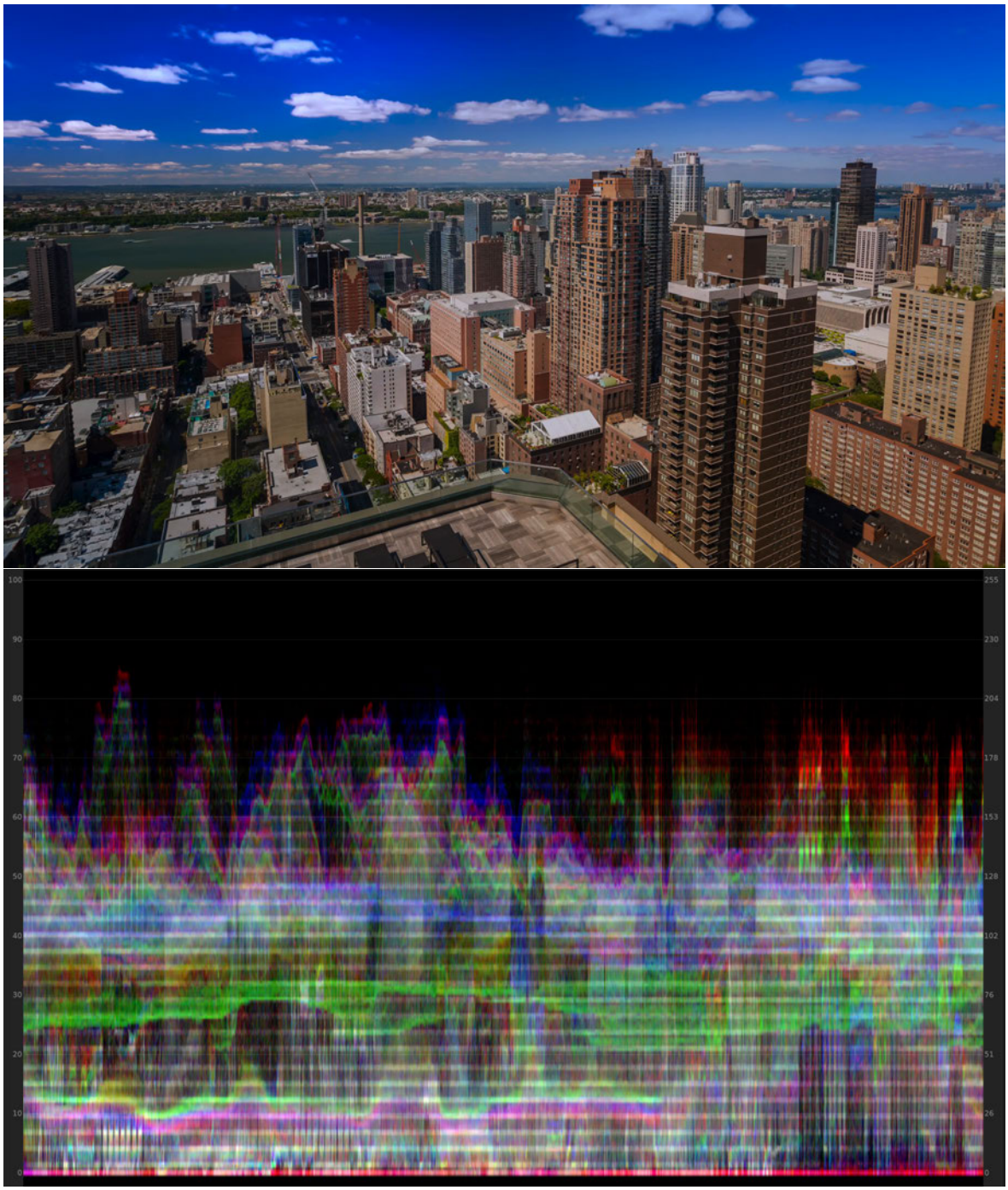

Abbildung B.3: Tonemapping „Linear“ Frame 50 mit entsprechendem Waveformmonitor

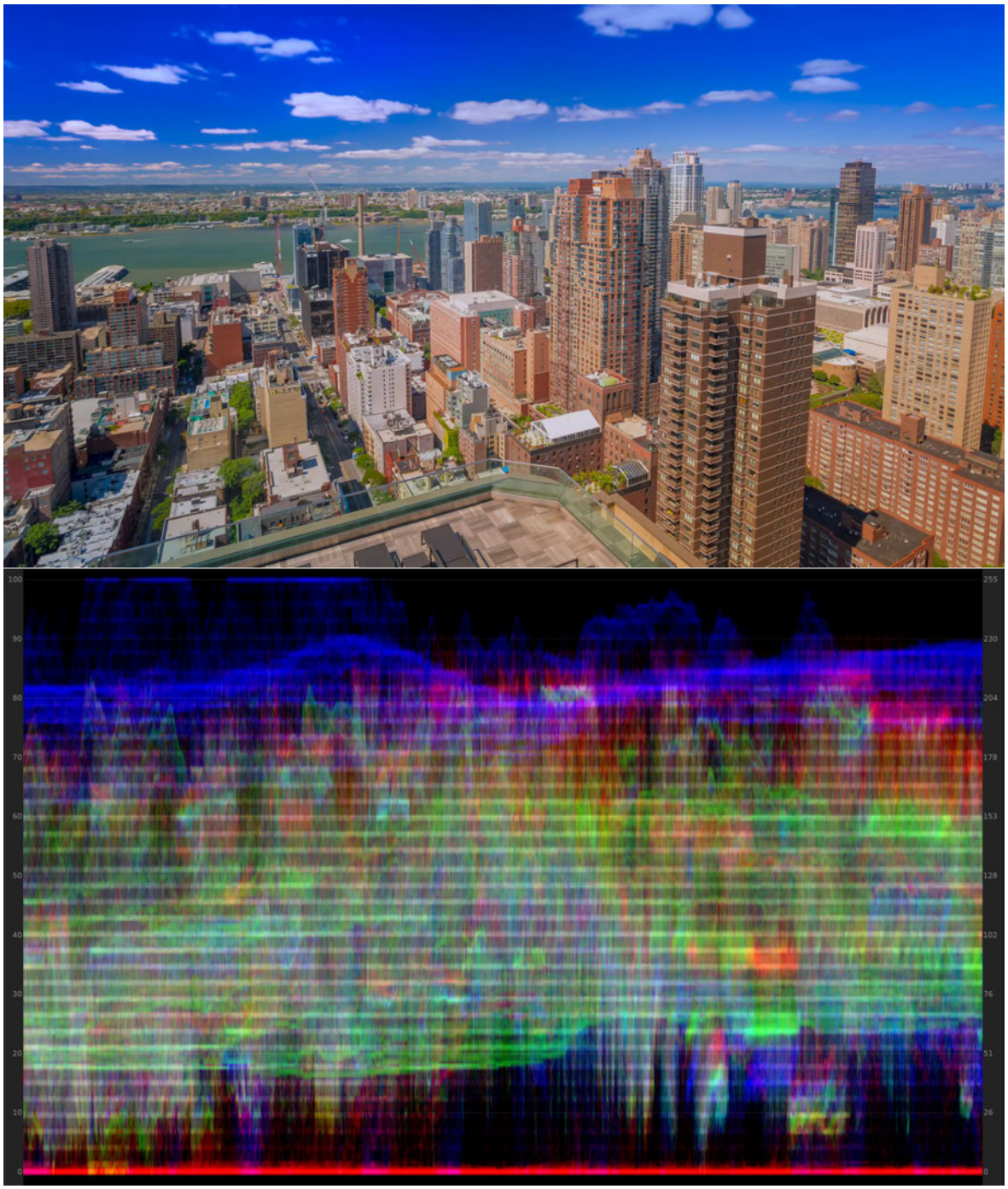

Abbildung B.4: Tonemapping „Gamma“ Frame 50 mit entsprechendem Waveformmonitor

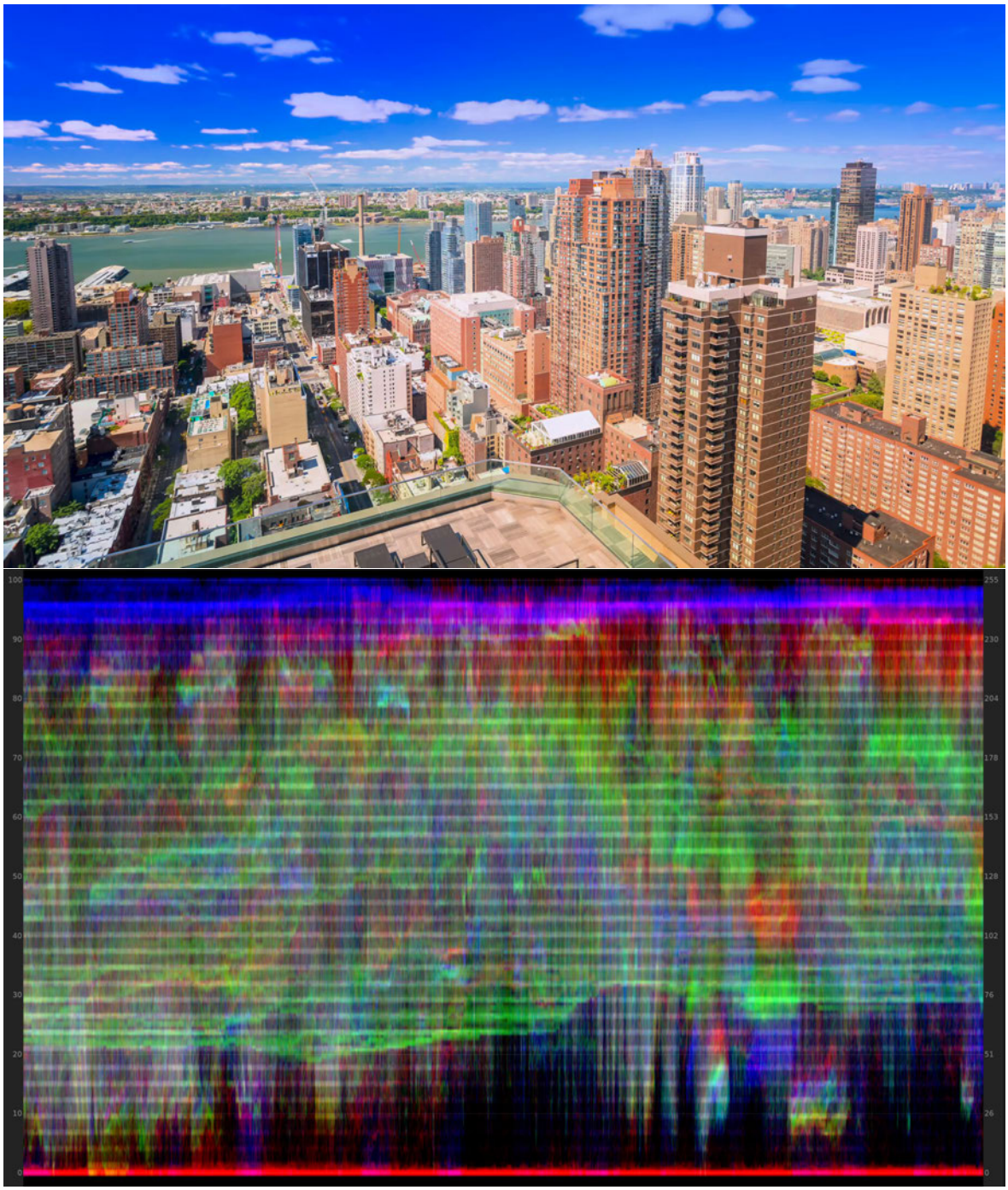

Abbildung B.5: Tonemapping „Reinhard“ Frame 50 mit entsprechendem Waveformmonitor

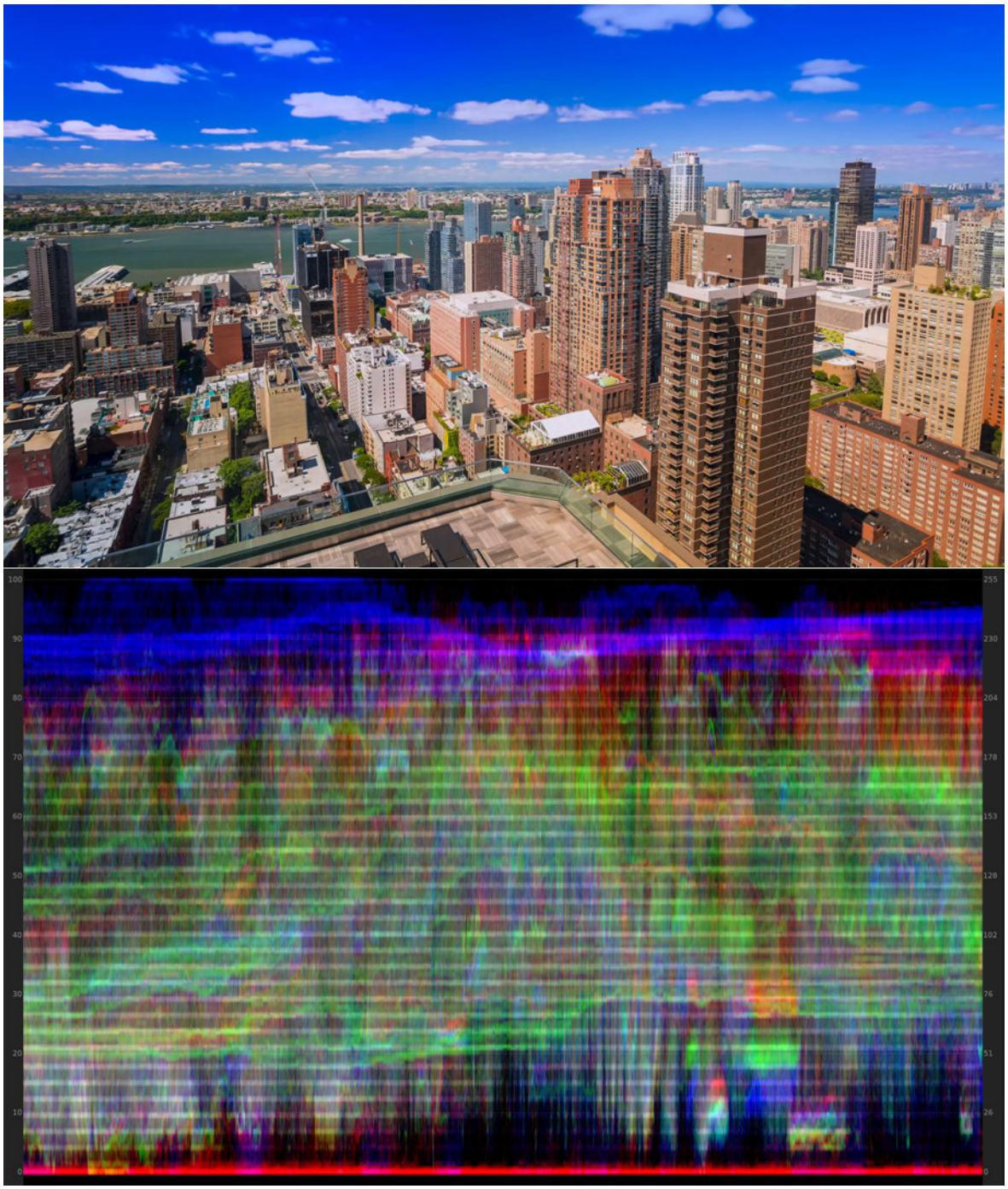

Abbildung B.6: Tonemapping „Hable“ Frame 50 mit entsprechendem Waveformmonitor

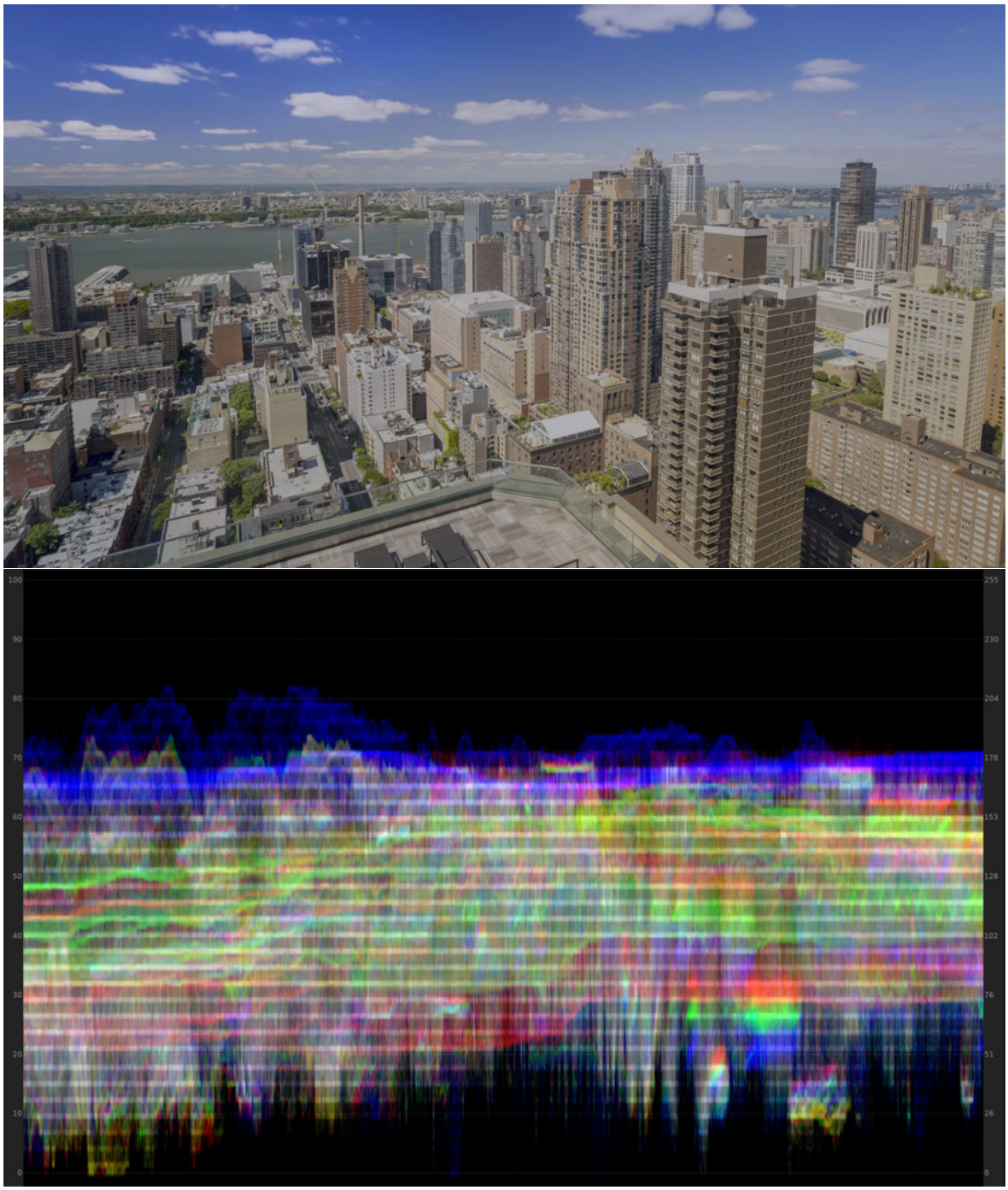

Abbildung B.7: Tonemapping „Mobius“ Frame 50 mit entsprechendem Waveformmonitor

Anhang C: Nutzerinterface

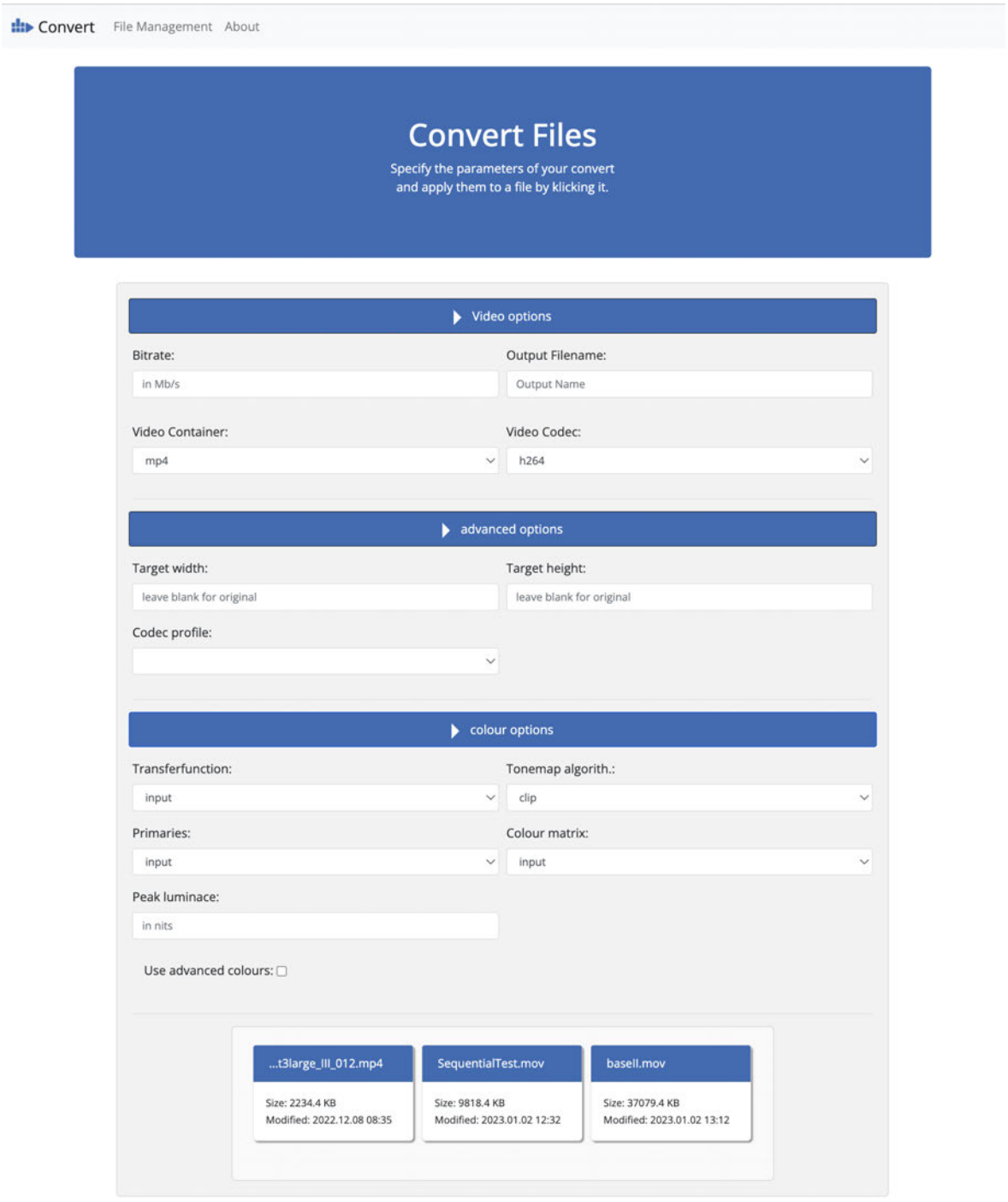

Abbildung C.1: Einstellungsoptionen des Nutzerinterfaces

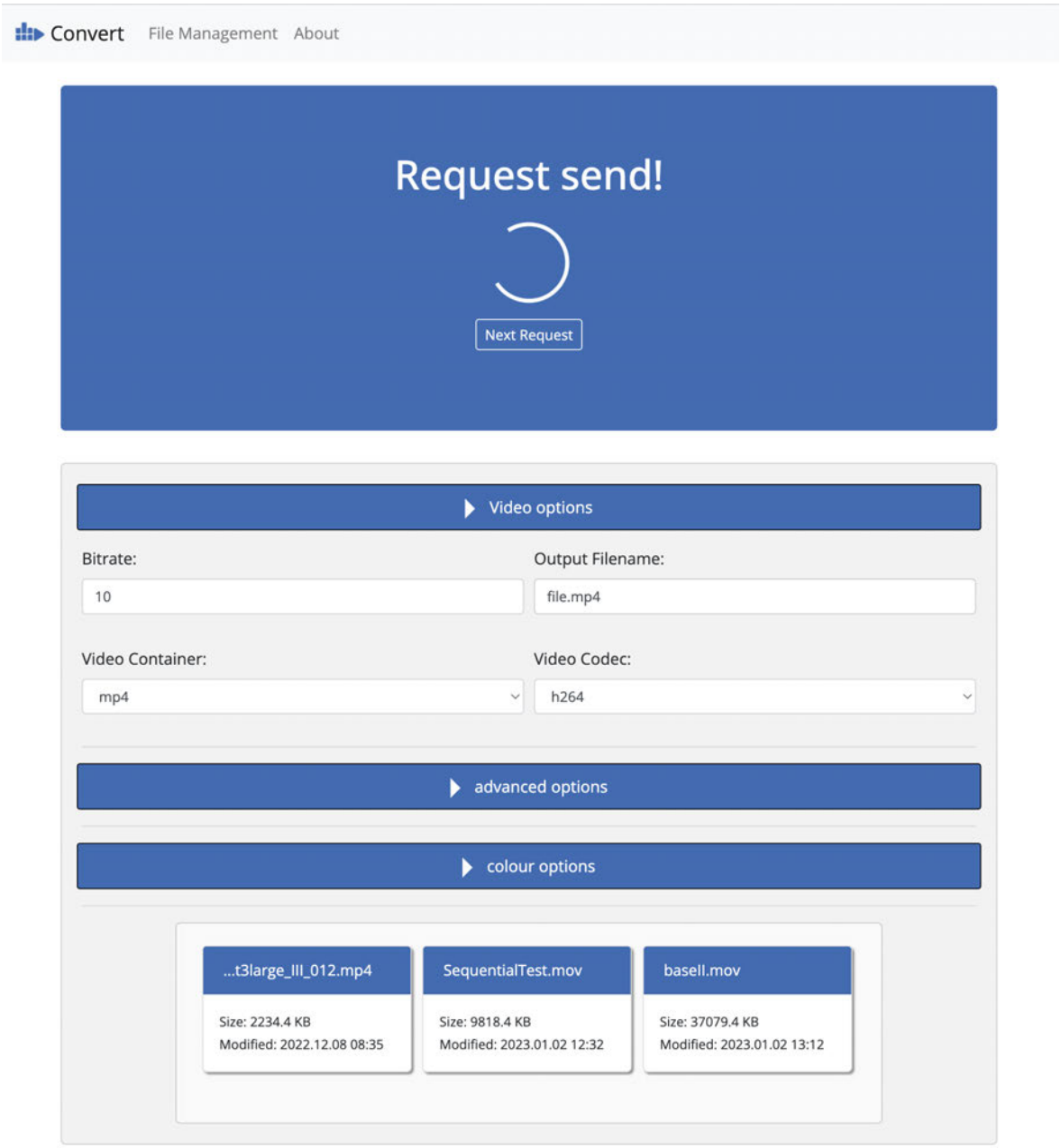

Abbildung C.2: Userinterface nach Einreihung einer Anfrage

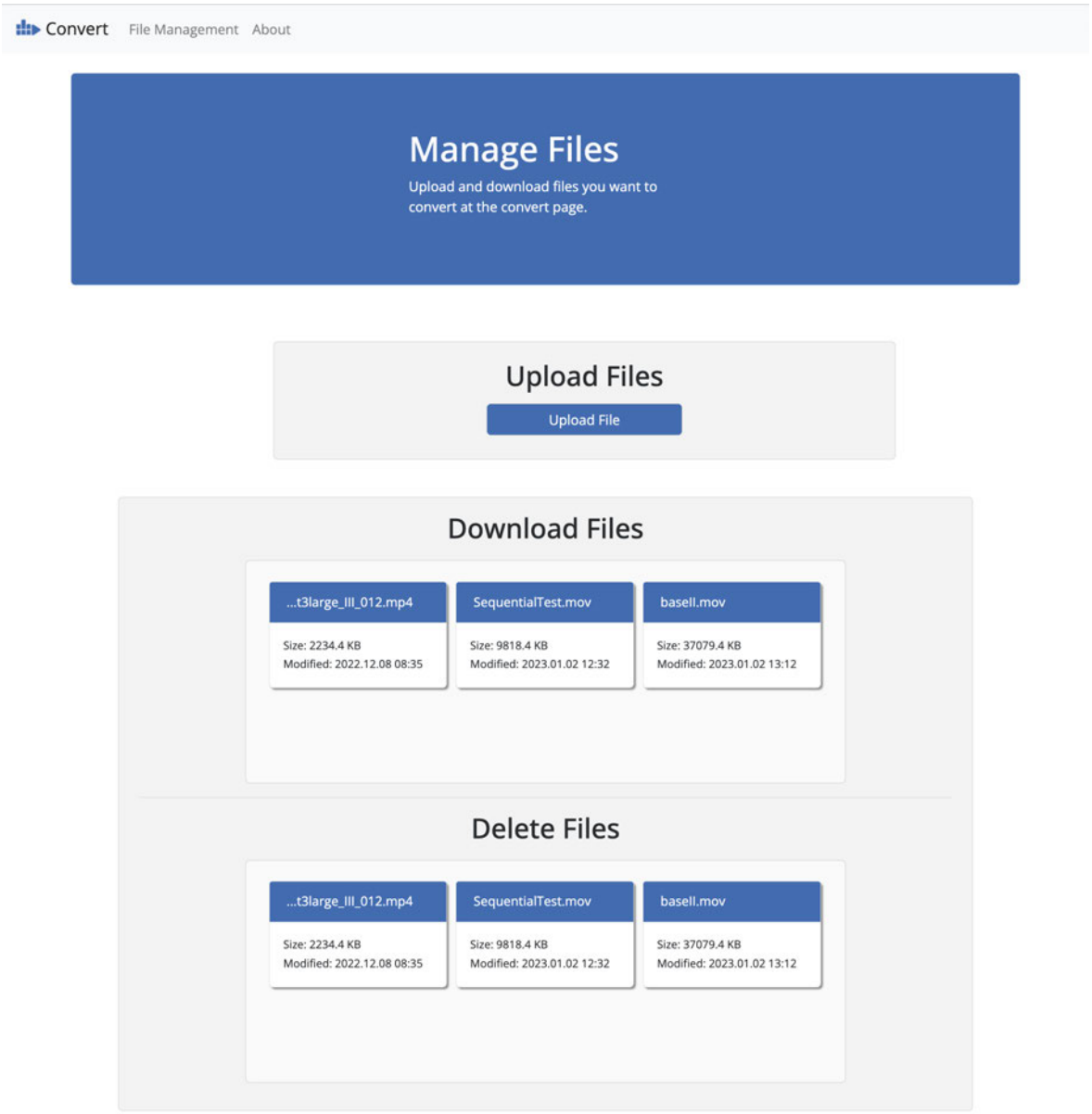

Abbildung C.3: Bedienoberfläche zur Verwaltung von Dateien

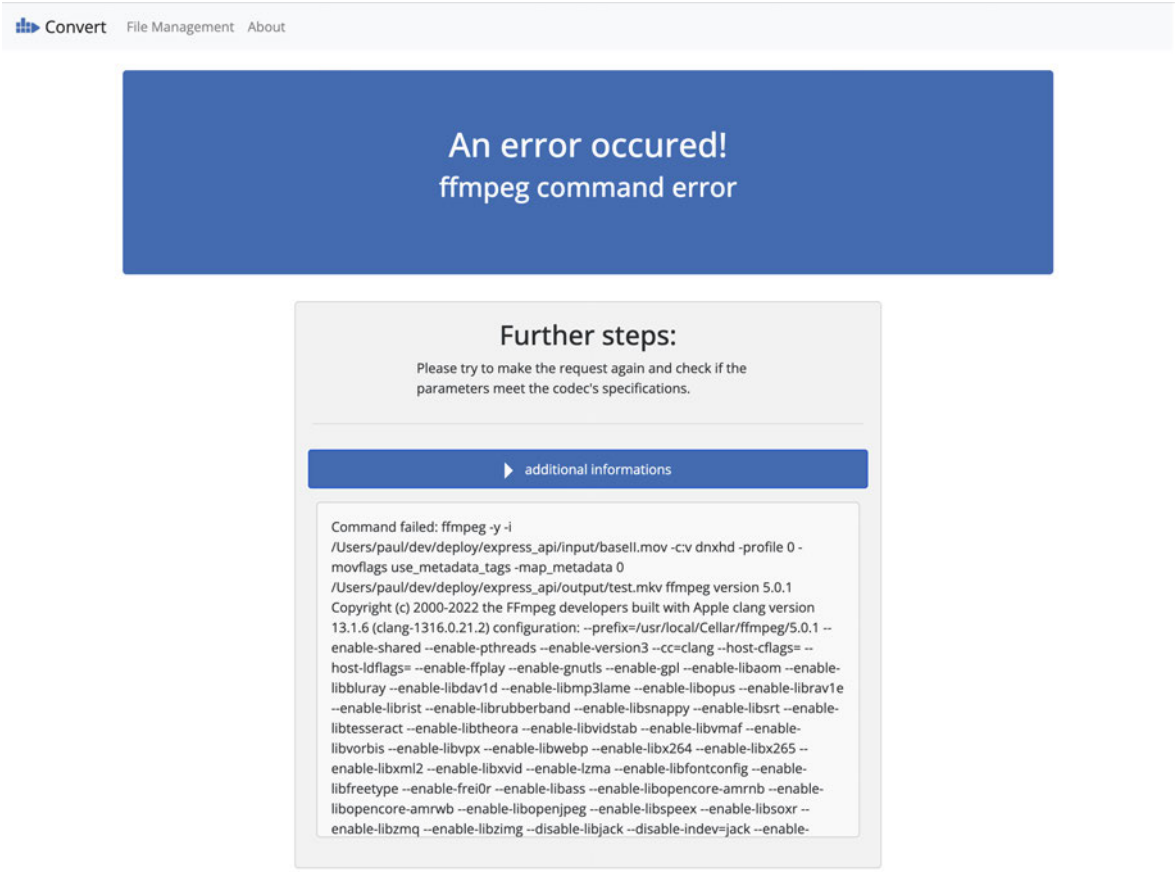

Abbildung C.4: Datenausgabe im Falle eines Fehlers

Anhang D: Qualitätsmessungen

Encoder	Ausgabe	PSNR Y	PSNR U	PSNR V
Media Encoder	5Mb/s H.264 Main mp4	39,94	41,68	45,63
FFmpeg Cloud		40,67	45,39	47,52
Media Encoder	5Mb/s H.265 Main mp4	40,98	42,5	47,89
FFmpeg Cloud		43,24	45,64	47,80
Media Encoder	ProRes Proxy 422.mxf	48,62	44,43	52,28
FFmpeg Cloud		47,13	50,85	52,06

Tabelle D.1: PSNR - komponentenweise Aufstellung

Encoder	Ausgabe	SSIM Y	SSIM U	SSIM V
Media Encoder	5Mb/s H.264 Main mp4	0,973	0,977	0,986
FFmpeg Cloud		0,973	0,982	0,989
Media Encoder	5Mb/s H.265 Main mp4	0,977	0,983	0,991
FFmpeg Cloud		0,983	0,984	0,990
Media Encoder	ProRes Proxy 422.mxf	0,995	0,994	0,998
FFmpeg Cloud		0,992	0,996	0,997

Tabelle D.2: SSIM - komponentenweise Aufstellung

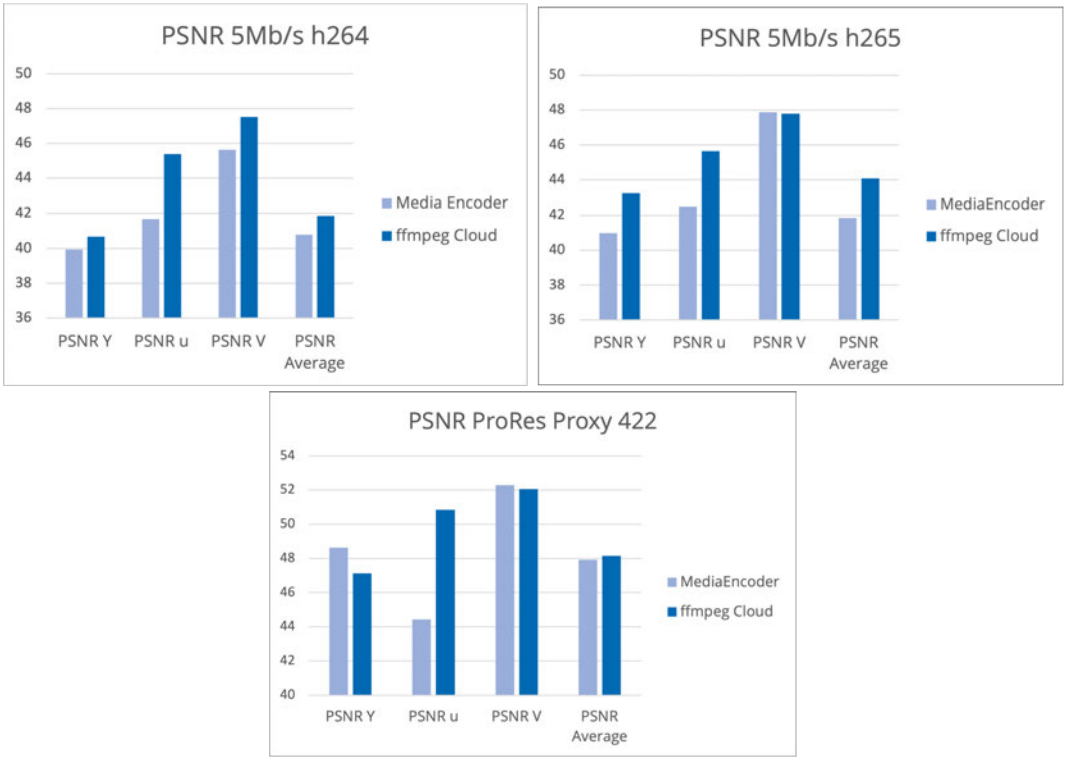

Tabelle D.3: Balkendiagramme komponentenweise Aufstellung der PSNR Ergebnisse

Tabelle D.4: Balkendiagramme komponentenweise Aufstellung der SSIM Ergebnisse

Eidesstattliche Erklärung

Hiermit versichere ich – Paul Pfleger – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.
Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.
Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 20. Januar 2023

Ort, Datum