



FernUniversität in Hagen
Fakultät für Mathematik und Informatik
Lehrstuhl für Multimedia und Internetanwendungen

Masterarbeit

zur Erlangung
des Grades Master of Science

über das Thema:

Ein verteiltes System zur Verarbeitung genomischer Daten

von

Mike Zickfeld

Matrikelnummer:

3146820

Betreuer der Universität:

Prof. Dr.-Ing. Matthias Hemmje,
M.Sc. Thomas Krause

Abgabedatum:

01.02.2023

Kurzfassung

Die Analyse von Genomdaten ist zentraler Bestandteil der klinischen Diagnostik. Für ein solches System gibt es unterschiedliche Perspektiven zur Umsetzung. Einerseits die Perspektive des Benutzers, welcher über eine Benutzungsschnittstelle unterschiedliche Frameworks / Algorithmen zur Analyse nutzt, und andererseits die der IT-Umsetzung. Darunter zählt z. B. die Speicherung von Genomdaten und Architektur des Systems. Es gibt keine Architektur, die diese Anforderungen erfüllt und modular aufgebaut sowie erweitert werden kann. Die vorliegende Arbeit betrachtet beide Perspektiven und modelliert eine verteilte, modulare Architektur zur Analyse von Genomdaten. Dazu wurde der Stand der Wissenschaft und Technik für unterschiedliche Frameworks / Algorithmen, die Speicherung, Möglichkeiten zur Umsetzung einer Benutzungsschnittstelle, Architektur sowie Deploymentmethoden betrachtet. Darauf aufbauend ist eine verteilte, modulare Architektur mit dem Namen „Genomic Explore Framework“ modelliert worden, welche die genannten Punkte integriert. Diese Architektur beinhaltet alle notwendigen Schritte vom Hochladen der Daten über die Analyse und Generierung des Reports. Ein Teil des „Genomic Explore Frameworks“ wurde als Prototyp mit dem Namen „Genomic Insights“ umgesetzt. GenomicInsights implementiert zwei beispielhafte Workflows und ermöglicht das Hochladen und die Analyse von Daten. Zum Schluss eines Workflows wird ein beispielhafter Report generiert und dem Anwender als Download zur Verfügung gestellt. Die erstellten Modelle und der Prototyp wurden als zielführend eingestuft und ermöglichen die Erstellung eines verteilten, modularen Systems zur Analyse von Genomdaten.

Abstract

The analysis of genomic data is a central component of clinical diagnostics. There are different perspectives for the implementation of such a system. On the one hand the perspective of the user who uses different frameworks / algorithms for analysis via a user interface and on the other hand the IT implementation. This includes e.g. the storage of genome data and architecture of the system. There is no architecture that can meet these requirements and be built and extended in a modular way. This thesis considers both perspectives and models a distributed, modular architecture for genomic data analysis. For this purpose, the state of the art for different frameworks / algorithms, storage, possibilities to implement a user interface, architecture as well as deployment methods were considered. Based on this, a distributed, modular architecture called "Genomic Explore Framework" has been modeled, which integrates the mentioned points. This architecture includes all necessary steps from data upload to analysis and report generation. As part of the "Genomic Explore Framework" a prototype called "Genomic Insights" has been implemented. The prototype implements two example workflows and enables data upload and analysis. At the end of a workflow, an exemplary report is generated and made available to the user as a download. The created models and the prototype were found to be target-oriented and allow the creation of a distributed, modular system for the analysis of genomic data.

Inhaltsverzeichnis

1. Einleitung.....	11
1.1. Motivation	11
1.2. Problembeschreibung	12
1.3. Forschungsfragen	13
1.4. Methodik	15
1.5. Forschungsziele	16
1.6. Ansatz und Gliederung der Arbeit	18
2. Stand der Wissenschaft und Technik	20
2.1. Recherche zu Open-Source Big-Data Analyse Software	20
2.1.1. Apache Spark	21
2.1.2. Apache Hadoop.....	25
2.1.3. Weitere Frameworks zur Analyse von Genomdaten	28
2.1.4. Zusammenfassung und Diskussion der Analyse Software	29
2.2. Recherche zu Open-Source Speichermöglichkeiten von Genomdaten.....	31
2.2.1. SAMBA	31
2.2.2. Ceph	32
2.2.3. HDFS	33
2.2.4. Zusammenfassung und Diskussion der Speichermöglichkeiten von Genomdaten	34
2.3. Recherche zur Erstellung von Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems.....	35
2.3.1. Model View Controller	35
2.3.2. Representational State Transfer	37
2.3.3. Event Driven Architecture	38
2.3.4. Microservices	39
2.3.5. Apache Airflow.....	39
2.3.6. Zusammenfassung und Diskussion der Softwarearchitektur für eine Benutzungsschnittstelle.....	40
2.4. Recherche zu Deployment-Methoden für die gesamte Softwarelösung zur Analyse von Genomdaten.	41
2.4.1. Container	41
2.4.2. Container Orchestration	43
2.4.3. Zusammenfassung und Diskussion von Deployment-Methoden von Softwarelösungen	43
2.5. Zusammenfassung der verbleibenden Herausforderungen	44
3. Modellierung	46
3.1. Integration der Analyse Tools	51
3.1.1. Apache Spark	52
3.1.2. QIIME2	53
3.1.3. BLAST	54
3.1.4. ASaiM	55
3.1.5. Zusammenfassung der Architektur der Analyse Tools.....	56
3.2. Integration des Speichers	56

3.2.1.	Use-Cases	56
3.2.2.	CEPH	58
3.2.3.	Architektur des Storage Microservices	58
3.2.4.	Zusammenfassung der Architektur des Speichers	59
3.3.	Modellierung von Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems	60
3.3.1.	Benutzungsschnittstelle.....	60
3.3.2.	Event Driven Architecture	61
3.3.3.	Reporting.....	63
3.3.4.	Zusammenfassung von der Modellierung einer Softwarelösung für eine Benutzungsschnittstelle des IT-Systems	64
3.4.	Erstellung des Deployment-Plans für die gesamte Softwarelösung zur Analyse von Genomdaten	65
3.4.1.	Kubernetes Objekte.....	65
3.4.2.	Skalierung	67
3.4.3.	Zusammenfassung von der Erstellung des Deployment-Plans für die gesamte Softwarelösung zur Analyse von Genomdaten.....	67
3.5.	Zusammenfassung der Modellierung	68
4.	Implementierung.....	70
4.1.	Implementierung der Analyse Software im Rahmen eines POCs.....	72
4.1.1.	BLAST Microservice	73
4.1.2.	GC-Content Microservice	75
4.1.3.	Zusammenfassung der Implementierung der Open-Source Big-Data Analyse Software im Rahmen eines POCs	76
4.2.	Implementierung der Speicherlösung im Rahmen eines POCs	77
4.2.1.	Konfiguration des Dateisystems	77
4.2.2.	Storage Microservice	78
4.2.3.	Zusammenfassung der Implementierung der Speicherlösung im Rahmen eines POCs.....	79
4.3.	Implementierung einer Softwarelösung für eine Benutzungsschnittstelle des IT-Systems im Rahmen eines POCs	80
4.3.1.	Benutzungsschnittstelle.....	80
4.3.2.	Apache Kafka.....	84
4.3.3.	Apache Airflow.....	86
4.3.4.	Report Microservices	89
4.3.5.	Zusammenfassung der Implementierung einer Softwarelösung für eine intuitive Benutzungsschnittstelle des IT-Systems im Rahmen eines POCs	90
4.4.	Deployment der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs	91
4.4.1.	Deploymentmethode	92
4.4.2.	Netzwerk	92
4.4.3.	Speicher.....	93
4.4.4.	Zusammenfassung des Deployments der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs	93
4.5.	Zusammenfassung der Implementierung	93

5. Evaluation.....	96
5.1. Evaluation der Open-Source Big-Data Analyse Software mit einem Analyse Use-Case.....	97
5.1.1. BLAST Microservice.....	97
5.1.2. GC-Content Microservice.....	97
5.1.3. Zusammenfassung der Evaluation der Open-Source Big-Data Analyse Software mit einem Analyse Use-Case.....	98
5.2. Evaluation der Open-Source Speicherlösung durch einen Use-Case.....	99
5.2.1. Dateisystem.....	99
5.2.2. Storage Microservice	100
5.2.3. Zusammenfassung der Evaluation mit der Open-Source Speicherlösung durch einen Use-Case	100
5.3. Evaluation zur Benutzung einer Benutzungsschnittstelle des IT-Systems anhand eines Analyse Use-Case.....	101
5.3.1. Benutzungsschnittstelle.....	101
5.3.2. Apache Kafka.....	102
5.3.3. Apache Airflow.....	103
5.3.4. Reporting Microservice	104
5.3.5. Zusammenfassung der Evaluation zur Benutzung einer für eine intuitiven Benutzungsschnittstelle des IT-Systems anhand eines Analyse Use-Case	104
5.4. Evaluierung des Deployments der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs	105
5.4.1. Deployment der Softwarelösung.....	105
5.4.2. Netzwerk	105
5.4.3. Speicher.....	105
5.4.4. Zusammenfassung der Evaluation des Deployments der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs	106
5.5. Zusammenfassung der Evaluation des Prototyps.....	107
6. Zusammenfassung und Ausblick.....	108
6.1. Zusammenfassung	108
6.2. Beantwortung der Forschungsfragen	112
6.3. Ausblick	113
Anhang	118

Abbildungsverzeichnis

1	Forschungsansätze nach Nunamaker [7]	15
2	Apache Spark Architektur [10]	21
3	Das Spark Ökosystem [12]	22
4	Apache Spark RDDs [16]	24
5	Ökosystem von Apache Hadoop [19]	25
6	Apache Hadoop YARN Architektur [19]	26
7	MapReduce Framework [14]	27
8	Ceph Kubernetes Stack [36]	32
9	HDFS Architektur [38]	33
10	Model View Controller Pattern [39]	35
11	MVVM-Pattern am Beispiel von Vue.js [40]	36
12	Container vs VM [48]	42
13	Die vier Phasen des User Centered System Design Ansatzes [50]	46
14	Use-Context Diagramm des Genomic Explore Frameworks (M1)	48
15	Komponentendiagramm des Genomic Explore Frameworks (M2)	50
16	Architekturdiagramm Apache Spark Integration in das Genomic Explore Framework (M3)	52
17	Architekturdiagramm QIIME2 Wrapper (M4)	53
18	Architekturdiagramm BLAST Wrapper (M5)	54
19	Architekturdiagramm ASaiM Wrapper (M6)	55
20	Deployment-Plan CEPH Integration (M7)	58
21	Aktivitätsdiagramm Benutzungsschnittstelle (M9)	60
22	Aktivitätsdiagramm Beispielworkflows (M10)	61
23	Aktivitätsdiagramm Events (M11)	62
24	Aktivitätsdiagramm Reporting Microservice (M12)	63
25	Deployment-Plan GEF in Kubernetes (M13)	66
26	Komponentendiagramm von Genomic Insights	71
27	Aktivitätsdiagramm BLAST Analyse Microservice (P1)	74
28	Aktivitätsdiagramm GC-Content Analyse Microservice (P2)	75
29	Aktivitätsdiagramm Storage Microservice (P4)	78
30	Screenshot GenomicInsights Use-Case 1 Daten hochladen	80
31	Screenshot GenomicInsights Use-Case 2 Workflow Auswählen	81
32	Screenshot GenomicInsights Use-Case 2 Workflow Auswählen (Fehler)	81
33	Screenshot GenomicInsights Use-Case 2 Datei auswählen	82
34	Screenshot GenomicInsights Use-Case 2 Datenkorrektheit überprüfen	82
35	Screenshot GenomicInsights Pop-Up Detailed View	82
36	Screenshot GenomicInsights Use-Case 4 Workflow Status und Report download	83
37	Aktivitätsdiagramm GenomicInsights Workflows (P7)	87
38	Screenshot Airflow DAG BLAST Workflow (P7)	88
39	Screenshot Beispielreport GC-Content (P8)	89
40	Aktivitätsdiagramm Report Microservice (P8)	90
41	Deployment-Plan GenomicInsights auf Docker (P9)	91
42	Screenshot BLAST Docker-Compose File (P9)	92
43	Sequenzdiagramm von GenomicInsights	94

44	Screenshot Ariflow BLAST Workflow	103
45	Projektplan der Arbeit.....	118

Tabellenverzeichnis

1	Kafka Topics (P6).....	85
2	Aufbau der Kafka Events (P6).....	86
3	Übersicht der Evaluation von GenomicInisights	96
4	Funktionstest der Benutzungsschnittstelle.....	101
5	Cognitive Walkthrough weitere Software-Features (ZA1)	102

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
CIFS	Common Internet File System
CLI	Command Line Interface
DIN	Deutsches Institut für Normung
DNA	Deoxyribonucleic Acid
EDA	Event Driven Architecture
FF	Forschungsfrage
FZ	Forschungsziel
GEF	Genomic Explore Framework
GNU	General Public License
HDFS	Hadoop Distributed File System
IVDR	In Vitro Diagnostics Regulation
JAR	Java Archive
M	Modell
mRNA	Messenger ribonucleic acid
MVC	Model View Controller
MVVM	Model View ViewModel
P	Prototyp
PoC	Proof of Concept
PS	Problemstellung
RDD	Resilient Distributed Dataset
REST	Representational State Transfer
RT-qPCR	Reverse Transcription quantitative Polymerase Chain
SQL	Structured Query Language
VH	Verbleibende Herausforderung
YARN	Yet Another Resource Negotiator
ZA	Zukünftige Arbeit

1. Einleitung

Genomdaten beinhalten die genetische Zusammensetzung (DNA) eines Organismus. Eine Analyse der Genomdaten von Menschen ist in den letzten Jahren bei der frühzeitigen Erkennung oder Identifizierung von Krankheiten immer wichtiger geworden [1]. Einzelne genomische Proben können eine Datenmenge von mehreren hundert Gigabyte erzeugen. Aus diesem Grund ist die Analyse und Verarbeitung von Genomdaten durch IT-Systeme von großem Vorteil. Durch diese kann eine automatisierte und schnellere Verarbeitung der großen Datenmengen unter der Vermeidung menschlicher Fehler durchgeführt werden [2]. Beispielhafte Verfahren zur Analyse von Genomdaten sind die Genexpressionsanalyse und die Metagenomik. Die Genexpressionsanalyse hat das Ziel die Aktivität vorhandene Gene am Patienten zu identifizieren und darauffolgend Krankheiten oder pathologische Zustände zu erkennen [3]. Innerhalb der Metagenomik werden die Genome von Mikroorganismen analysiert. Viele unterschiedliche Mikroorganismen sind unter anderem im Speichel oder Darm eines Menschen zu finden. Bakterien sind ein Beispiel für Mikroorganismen. Der Lebensraum solcher Mikroorganismen bezeichnet man als Mikrobiome. Durch die Analyse der Zusammensetzung dieser Mikrobiome können auch hier Rückschlüsse auf den Gesundheitszustand eines Patienten gezogen werden [4].

1.1. Motivation

Die folgende Arbeit wird auf den Vorarbeiten an dem Thema „AI-Assisted Laboratory Diagnostics for Genomic Applications“ (GenDAI) [5] und der Bachelorarbeit mit dem Thema „Verarbeitung von RT-qPCR Daten in der Labordiagnostik“ [6], welche an diesem Lehrstuhl durchgeführt worden sind, aufbauen. GenDAI beschäftigt sich mit der konzeptionellen Architektur, welche künstliche Intelligenz und genomische Anwendungen für Analysen kombiniert. Dabei werden auch die Herausforderungen der Labordiagnostik berücksichtigt. Die Arbeit von Frau Dr. Jolkver beleuchtet im Detail, wie die Automatisierung der Genexpressionsanalyse im regulierten Bereich vorgenommen werden kann und ist damit ein Bestandteil von „GenDAI“.

Beide Arbeiten versuchen einen hohen Grad an Automatisierung in der Verarbeitung und Analyse zu erreichen, sowie menschliche Fehler wie beispielsweise eine Fehldiagnose zu verringern oder auch eine schnelle Diagnose bereitzustellen.

Menschen mithilfe der Nutzung von Technologie ein besseres Leben durch die Verabreichung richtiger Medikamente oder ein längeres Leben durch die Erkennung von Krankheiten zu ermöglichen, löst eine intrinsische Motivation aus.

1.2. Problembeschreibung

Um die im Kapitel 1 beschriebenen Vorteile der Analyse von Genomdaten in der Praxis verwendbar zu machen, können IT-Systeme bei der Verarbeitung und Analyse eingesetzt werden.

Hierbei ist die große Datenmenge eine Herausforderung, welche pro Patienten gespeichert und analysiert werden muss. Genomdaten können mehrere hundert Gigabyte an Größe erreichen.

Ebenfalls ist noch offen, welche Applikationsarchitektur und Deployment-Methode sich für diese Problemstellung am besten eignen.

Eine weitere Herausforderung ist die Erstellung einer Benutzeroberfläche zur Vermeidung von Fehlbedienungen bei gleichzeitig hoher Heterogenität der zu unterstützenden Analysen.

In dem Gebiet der Analyse von Big-Data, z. B. von Genexpressiondaten, sind bereits unterschiedliche Softwareanbieter auf dem Markt, welche ihre Software kommerziell vertreiben. Die Verwendung von Open-Source Technologien im gesamten Application-Stack sind aus der Unabhängigkeit und eines gegebenenfalls auftretenden Vendor Lock-ins wünschenswert. Die meisten existierenden Tools oder Pipelines sind nur kommerziell verfügbar.

Für die Erstellung eines verteilten Systems zur Analyse von Genomdaten sind vier Problemfelder vorhanden:

Problemfeld 1: Auswahl und Architektur einer Big-Data Softwarelösung zur Analyse von Genomdaten.

Problemfeld 2: Auswahl und Architektur einer effizienten Speichermöglichkeit von Genomdaten.

Problemfeld 3: Implementierung einer Benutzungsschnittstelle zur Interaktion mit dem IT-System.

Problemfeld 4: Auswahl einer Deployment-Methode für die gesamte Softwarelösung zur Analyse von Genomdaten.

Aufgrund der genannten Problemfelder lassen sich folgende vier „Problem Statements“ definieren:

PS 1: Es ist unklar, welche Open-Source Big-Data Softwarelösungen zur Prozessierung von Genomdaten geeignet sind.

PS 2: Es ist unklar, welche Möglichkeiten zur Speicherung von Genomdaten bestehen.

PS 3: Es ist offen, welche Kommunikationsmöglichkeiten zwischen Benutzer und verteiltem System sowie zwischen den Komponenten geeignet ist.

PS 4: Es ist unklar, welche Deployment-Methode für die Big-Data, Speichermöglichkeit und Frontend zu Analyse von Genomdaten passend ist.

1.3. Forschungsfragen

Aus den vier „Problem Statements“, die in Kapitel 1.2 vorgestellt worden sind, lassen sich folgende vier Forschungsfragen definieren:

FF 1: Welche Open-Source Big-Data Softwarelösungen eignen sich zur Analyse von Genomdaten?

Für die Analyse von Genomdaten innerhalb eines IT-Systems gibt es unterschiedliche Ansätze für eine Implementierung von Big Data Lösungen. Dort findet die eigentliche Analyse der Genomdaten statt. Um auf proprietäre Software und Lieferantenabhängigkeiten zu verzichten, werden nur Open Source Technologien zur Umsetzung verwendet.

FF 2: Wie können Genomdaten persistiert werden?

Für die Analyse müssen Genomdaten ebenfalls abgespeichert und auch für zukünftige Analysen vorgehalten werden. Für jede Art von Daten ist eine geeignete Speichertechnologie am Markt vorhanden. Aus diesen unterschiedlichen Technologien ist für den Use-Case der Persistierung von Genomdaten eine passende auszuwählen. Um auf proprietäre Software and Vendor Abhängigkeiten zu verzichten, werden nur Open Source Technologien zur Umsetzung in Betracht gezogen.

FF 3: Wie kann eine Benutzungsschnittstelle zur Interaktion zwischen Benutzer und verteiltem System, sowie die Kommunikation zwischen den Komponenten implementiert werden?

Zur benutzerfreundlichen Interaktion mit dem IT-System (Big-Data- und Speicherlösung) wird ein Frontend für die Benutzer benötigt. Es muss nicht nur der Anwender des Systems mit solchem kommunizieren, sondern auch die einzelnen Komponenten untereinander. Ebenfalls stellt der Report am Ende eines Analyseworkflows eine weitere Kommunikationsmethode zwischen dem IT-System und Anwender dar. Für solche Implementierungen gibt es unterschiedliche Technologien und Architekturen zur Realisierung.

FF 4: Wie kann eine Architektur zur Interaktion, Speicherung und Analyse von Genomdaten effizient deployed werden?

Für das Deployment des IT-Systems (Frontend, Speicherlösung und Big-Data Software) sind unterschiedliche Möglichkeiten vorhanden, welche einerseits an die örtlichen Gegebenheiten der Infrastruktur angepasst, andererseits einfach zu warten und betreibbar sein müssen.

1.4. Methodik

Die grundlegende Methodik dieser Arbeit ist die Nunamaker-Methode [7]. Diese basiert darauf, dass jede Forschungsfrage in mindestens vier Forschungsziele aufgeteilt werden kann. Die aus der Forschungsfrage definierten Forschungsziele sind in die Bereiche Observation, Theory Building, Implementation und Experimentation aufzuteilen, um zum Schluss die Forschungsfrage beantworten zu können.

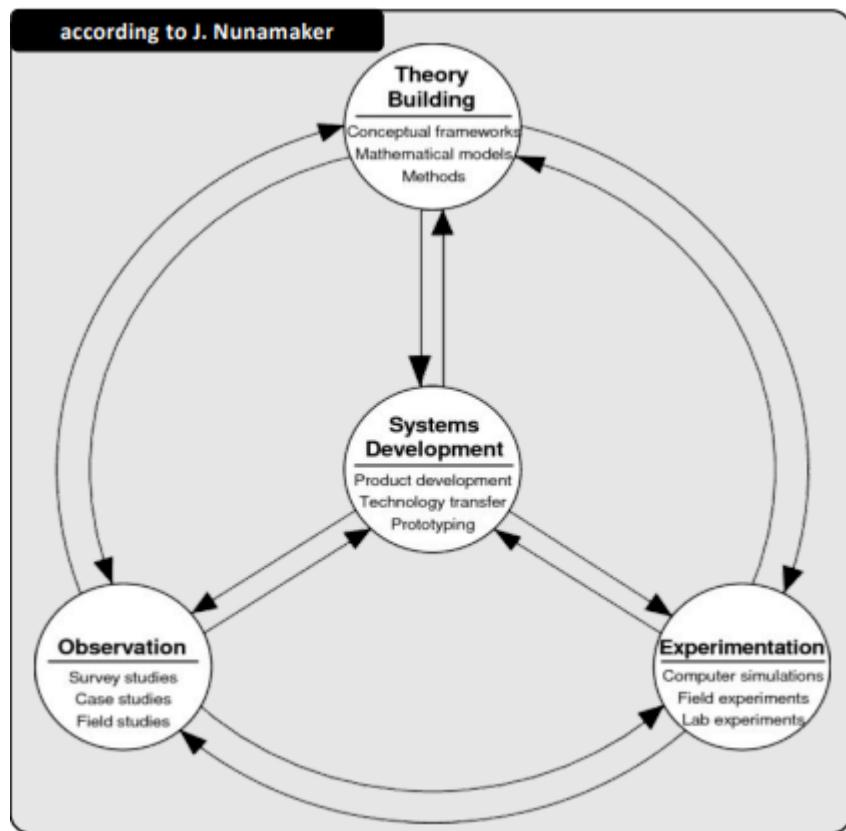


Abbildung 1: Forschungsansätze nach Nunamaker [7]

1.5. Forschungsziele

Aus jeder Forschungsfrage können vier Forschungsziele nach der Nunamaker Methode (Kapitel 1.3) abgeleitet werden. Diese repräsentieren die vier Phasen der Nunamaker Methode:

Observation = O

Theory Building = TB

Systems Development = I

Experimentation = E

FZ 1.1/O: Recherche zu Open-Source Big-Data Analyse Software.

FZ 1.2/TB: Auswahl und Architektur der Open-Source Big-Data Analyse Software.

FZ 1.3/I: Implementierung der Open-Source Big-Data Analyse Software im Rahmen eines POCs.

FZ 1.4/E: Evaluierung der Open-Source Big-Data Analyse Software mit einem Analyse Use-Case.

FZ 2.1/O: Recherche zu Open-Source Speichermöglichkeiten von Genomdaten (Big-Data).

FZ 2.2/TB: Auswahl und Architektur der Speichermöglichkeiten.

FZ 2.3/I: Implementierung der Speicherlösung im Rahmen eines POCs.

FZ 2.4/E: Evaluierung der Open-Source Speicherlösung durch einen Use-Case.

FZ 3.1/O: Recherche zur Erstellung von Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems.

FZ 3.2/TB: Modellierung von Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems.

FZ 3.3/I: Implementierung der Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems im Rahmen eines POCs.

FZ 3.4/E: Evaluierung der Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems.

FZ 4.1/O: Recherche zu Deployment-Methoden für die gesamte Softwarelösung zur Analyse von Genomdaten.

FZ 4.2/TB: Gegenüberstellung der Deployment-Methoden und Erstellung des Deployant-Plans für die gesamte Softwarelösung zur Analyse von Genomdaten.

FZ 4.3/I: Deployment der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs.

FZ 4.4/E: Evaluierung des Deployments der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs.

1.6. Ansatz und Gliederung der Arbeit

Alle aufgeteilten Forschungsziele nach den jeweiligen Bereichen Observation, Theory Building, Systems Development und Experimentation werden nach Nunamaker den folgenden Kapiteln zugeordnet [7]:

FZ Typ /O: Kapitel 2 (Stand der Wissenschaft und Technik)

FZ Typ /TB: Kapitel 3 (Modellierung)

FZ Typ /I: Kapitel 4 (Implementierung)

FZ Typ /E: Kapitel 5 (Evaluation)

Im folgenden Abschnitt sind die Forschungsziele, welche in Kapitel 1.5 nach Forschungsfrage gruppiert sind, in die verschiedenen Bereiche nach Nunamaker eingeteilt:

FZ 1.1/O: Recherche zu Open-Source Big-Data Analyse Software.

FZ 2.1/O: Recherche zu Open-Source Speichermöglichkeiten von Genomdaten.

FZ 3.1/O: Recherche zur Erstellung von Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems.

FZ 4.1/O: Recherche zu Deployment-Methoden für die gesamte Softwarelösung zur Analyse von Genomdaten.

FZ 1.2/TB: Architektur der Analyse Tools.

FZ 2.2/TB: Integration des Speichers.

FZ 3.2/TB: Modellierung von Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems.

FZ 4.2/TB: Erstellung des Deployment-Plans für die gesamte Softwarelösung zur Analyse von Genomdaten.

FZ 1.3/I: Implementierung der Open-Source Big-Data Analyse Software im Rahmen eines POCs.

FZ 2.3/I: Implementierung der Speicherlösung im Rahmen eines POCs.

FZ 3.3/I: Implementierung der Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems.

FZ 4.3/I: Deployment der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs.

FZ 1.4/E: Evaluation der Open-Source Big-Data Analyse Software mit einem Analyse Use-Case im Rahmen eines POCs.

FZ 2.4/E: Evaluation mit der Open-Source Speicherlösung durch einen Use-Case im Rahmen eines POCs.

FZ 3.4/E: Evaluation der Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems im Rahmen eines POCs.

FZ 4.4/E: Evaluation des Deployments der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs.

2. Stand der Wissenschaft und Technik

In diesem Abschnitt wird der aktuelle Stand der Wissenschaft und Technik vorgestellt, welcher den „Observation“-Bereich der in Kapitel 1.4 beschriebenen Nunamaker-Methodik darstellt. Alle Forschungsziele, welche im Kapitel 1.6 nach X.1/O sortiert wurden, haben eigene Unterkapitel. Zunächst werden unterschiedliche Ansätze für Open-Source Big-Data Analyse Software im Kapitel 2.1 beschrieben. Darauffolgend im Unterkapitel 2.2 die Speichermöglichkeiten von Genomdaten. Im Unterkapitel 2.3 werden die Möglichkeiten und Architekturen zur Erstellung einer Benutzungsschnittstelle erörtert. Die Deployment-Methoden für den gesamten Software-Stack sind im Kapitel 2.4 aufgeführt. Die Resultate werden im Kapitel 2.5 diskutiert und verbleibende Herausforderungen beschrieben.

2.1. Recherche zu Open-Source Big-Data Analyse Software

Sinngemäß übersetzt bedeutet der Begriff „Big-Data“ eine große Datenmenge oder viele Daten. Dabei spricht man nicht nur von Big-Data bei einer großen Datenmenge, sondern auch, wenn andere Gegebenheiten erfüllt sind. Diese Charakteristika zur Beschreibung von Big-Data werden als 5V's zusammengefasst:

Variety: die Vielfalt der Datenquellen und Datenstrukturen

Velocity: die Geschwindigkeit, mit welcher die Daten erstellt, verarbeitet oder verfügbar sind

Volume: die Größe des Datenbestandes

Value: wertvolle Einsichten oder Vorteile der Verarbeitung

Veracity: Unvorhersehbarkeit einiger Daten [8]

Aufgrund dieser Definition kann bei der Verarbeitung von Genomdaten von Big-Data gesprochen werden [2].

Bei der Verarbeitung von Big-Data gibt es die zwei großen Kategorien Batch- und Streamingdata. Während Batchdaten schon über einen Zeitraum gesammelt wurden und komplett zur Analyse bereitstehen, bezieht sich Streamingdata auf die Analyse von Echtzeitdaten [9].

Für die Analyse von Big-Data auf Open-Source-Basis stehen unterschiedliche Frameworks / Applikationen zur Verfügung, welche im folgenden Abschnitt vorgestellt werden.

2.1.1. Apache Spark

Spark ist ein Open-Source Projekt der Apache Software Foundation. Es besteht aus mehreren Komponenten, die zusammen eine einheitliche Plattform zur verteilten und performanten Verarbeitung von Daten bereitstellt. Der Prozess zur Verarbeitung von Daten besteht hierbei aus dem Entgegennehmen, Verarbeiten und Speichern von Daten aus einer Vielzahl von Quellen. Für die verteilte Verarbeitung kann Spark die Arbeitslast auf mehrere Computer mit einem eigenen oder mit anderen Schedulern deployen. Alle genannten Merkmale werden dem Entwickler durch eine API zur Verfügung gestellt, welche eine Abstraktion der darunterliegenden Technologien darstellt.

Die High Level Architektur von Spark besteht aus dem Cluster Manager, Driver Program und Executor. Die Architektur ist in der Abbildung 2 aufgeführt.

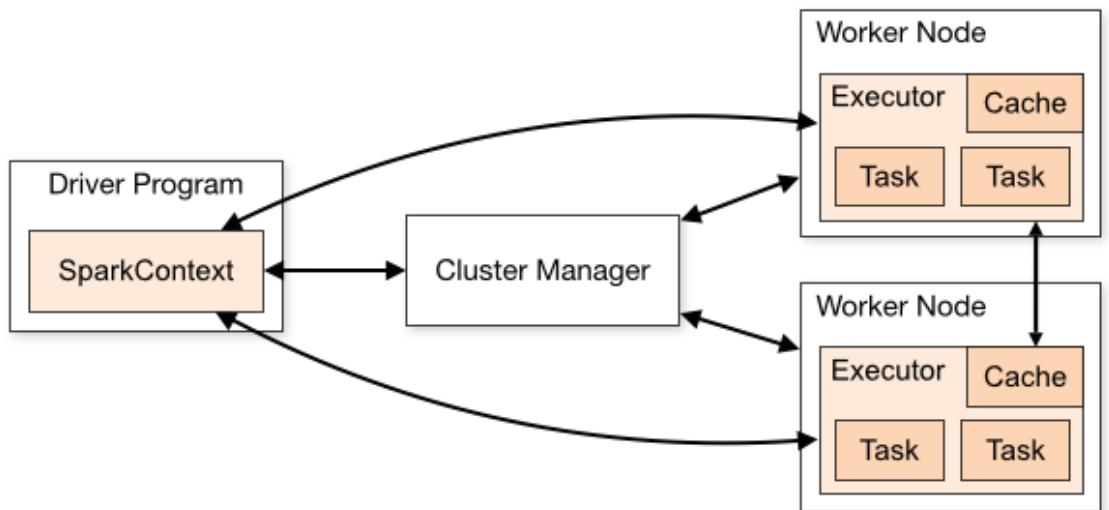


Abbildung 2: Apache Spark Architektur [10]

Der Cluster Manager ist für die in dem Cluster enthaltenen Server verantwortlich. Dort werden die Spark Applikationen ausgeführt. Spark unterstützt unterschiedliche Cluster Manager, z. B. Kubernetes, Yarn, Mesos, Docker Swarm oder den eingebauten Cluster Manager.

Die linke Komponente, welche in Abbildung 2 zu sehen ist, zeigt die Driver Program Komponente. Diese ist für die Ausführung der Spark-Anwendung verantwortlich. Einerseits kommuniziert diese mit dem Cluster Manager, um physische Ressourcen bereitgestellt zu bekommen, andererseits mit allen Exekutoren im Spark-Cluster, um deren Zustand verwalten zu können [11].

Der SparkContext zu jeder Applikation wird durch den Driver Prozess erstellt und führt die main() Methode des Programmes aus [11].

Auf der rechten Seite in der Abbildung 2 sind die Spark-Executors zu sehen. Diese sind Prozesse, welche auf den Worker-Nodes ausgeführt werden, und haben die Aufgabe, die vom Spark-Driver zugewiesenen Aufgaben zu bearbeiten. Weiterhin melden diese den Status (Erfolg oder Misserfolg) und ihre Ergebnisse dem Spark-Driver zurück. Der Executor-Prozess wird für jede Spark-Anwendung individuell gestartet und erst gestoppt, wenn die Anwendung abgeschlossen ist. Zwischen einzelnen Spark-Applikationen können keine Daten direkt ausgetauscht werden, ohne auf den darunterliegenden Speicher zuzugreifen.

Der Driver Prozess und die Executors bilden zusammen die Spark-Applikation.

Spark-Applikationen können in drei unterschiedlichen Modi deployed werden. Die Deployment-Methoden sind der Cluster mode, Client mode und Local mode.

Im Cluster Modus wird eine pre-kompilierte JAR; Python Skript oder R Skript zu dem Cluster Manager gesendet. Dieser startet den Executor- und Driver Prozess auf einem zur Verfügung stehenden Worker-Node. In diesem Fall ist der Cluster Manager für den Betrieb der Applikation innerhalb des Clusters verantwortlich.

Wird der Client Mode gewählt, bleibt der Driver Prozess auf dem Client, der die Applikation abgeschickt hat. In diesem Fall ist der Cluster-Manager nur für den Start und Betrieb des Executor Prozesses verantwortlich.

Der Local Mode ist für das Debugging oder zum Testen der Applikation da. Hierbei wird die Spark Applikation auf dem gleichen Host ausgeführt. Executor- und Driver Prozess werden auf der gleichen Maschine gestartet [10].



Abbildung 3: Das Spark Ökosystem [12]

Das Spark-Ökosystem besteht aus unterschiedlichen Komponenten, welche in der Abbildung 3 dargestellt sind. Es besteht aus dem Spark-Core, Spark Streaming, Spark SQL, GraphX, MLlib und SparkR. Neben diesen Komponenten besteht Spark aus Standard Bibliotheken für die Programmiersprachen R, Python, Scala, SQL, etc.

Der Spark-Core beinhaltet das System zur parallelen Datenverarbeitung. Dieses System wurde im vorherigen Abschnitt schon beschrieben und beinhaltet zusammengefasst den Scheduler, das Starten und Überwachen von Jobs, Interaktion mit Speichermedien, Memory Management, usw.

Alle weiteren Komponenten sind Erweiterungen des Spark-Cores. Spark Streaming erlaubt die einfache Verarbeitung von Echtzeitdaten. SparkSQL integriert die relationale Verarbeitung von Daten über die API. GraphX ist die API für Graphen und Graphen-parallele Berechnungen. MLlib ist eine Bibliothek, um Machine-Learning in Apache Spark durchzuführen. SparkR erlaubt eine verteilte Dataframe-Implementierung. Darin enthalten sind die Operationen wie z. B. Aggregation, Filterung und Auswahl [9].

Für eine performante Datenverarbeitung bietet Spark die Möglichkeit, In-Memory Berechnungen durchzuführen. Dies wird über Resilient Distributed Datasets ermöglicht. RDDs sind eine partitionierte Sammlung von Daten, auf die nur lesend zugegriffen werden kann und eine parallele Verarbeitung dieser ermöglicht. Die Partitionierung ermöglicht die Verteilung von großen Datenmengen auf mehrere Nodes. Auf RDDs kann nur lesend zugegriffen werden. Dadurch sind RDDs unveränderlich (engl. immutable). Es kann jedoch immer ein neuer RDD aus einem bestehenden erzeugt werden oder durch externe Datenquellen. Für die Erstellung von RDDs über externe Datenquellen kann jeder Speicher verwendet werden, welcher auch durch Hadoop supportet wird. Hierzu zählen Cassandra, HDFS, HBase, Amazon S3 etc. [13].

Durch die In-Memory Berechnungen und Verwendung von RDDs wird ein Performancegewinn gegenüber dem MapReduce Framework generiert. Bei der Verwendung eines MapReduce Frameworks können (Zwischen-) Ergebnisse nur durch zeitintensive Schreib- und Lesezugriffe über die Platte gespeichert werden [14]. RDDs sind weiterhin fehlertolerant. Es werden Informationen zur Datenabfolge gesammelt, um bei einem Ausfall die verlorene Partition automatisch wiederherstellen zu können [15].

Es können die zwei Operationen Transformation und Aktion durch RDDs ausgeführt werden. Die Transformation-Operationen sind Map-, Filter und Join. Durch den nur lesenden Zugriff wird bei der Ausführung dieser ein neues RDD definiert. Aktionen geben nach der durchgeföhrten Berechnung einen Wert an das Programm zurück [9].

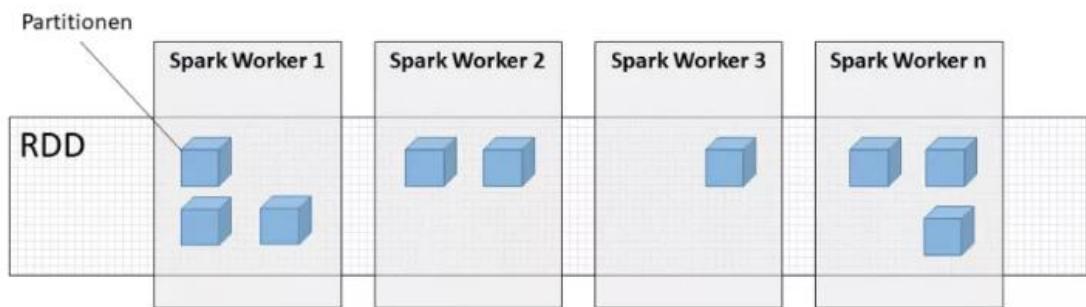


Abbildung 4: Apache Spark RDDs [16]

Spark kann von externen Datenquellen unstrukturierte Daten (z. B. TXT), semi strukturierte Daten (z. B. JSON) oder strukturierte Daten verarbeiten. Aufgrund des Open Source Ansatzes wird durch die Community eine Vielzahl von externen Datenquellen unterstützt, z. B. Cassandra, HBase, MongoDB, PostgreSQL, HDFS, MongoB uvm. [10].

Leung, Sarumi und Zhang [17] haben auf einem Apache Spark Cluster einen skalierbaren Algorithmus zur Vorhersage von Genen im Genom von eukaryotischen Organismen nach Naives Bayes erfolgreich eingesetzt.

Zhou, Li, Yuan, Liu, Yoa, Luo, Niu haben MetaSpark vorgestellt. Dieses Spark-basierte Tool ist eine Implementierung einer verteilten Verarbeitung von metagenomischen Daten [18].

2.1.2. Apache Hadoop

Hadoop ist ein Open-Source Projekt der Apache Software Foundation und besteht ähnlich wie Apache Spark aus mehreren Komponenten, welche in Verbindung das Hadoop Framework ergeben. Es wird für die Verarbeitung und Speicherung von Big Data verwendet und zeichnet sich durch Fehlertoleranz, Skalierbarkeit und Parallelverarbeitung von Berechnungen aus [9]. Strukturierte und unstrukturierte Datensätze können verarbeitet werden. Hadoop besteht aus den zwei Kernkomponenten HDFS und YARN. HDFS ist der Speicher eines Hadoop Clusters und wird im Kapitel 2.2.3 als Speichermöglichkeit von Genom Daten genauer analysiert. Dies bildet den Distributed Storage Layer. YARN ist innerhalb eines Hadoop Clusters für das Cluster Ressource Management verantwortlich und kümmert sich um das Scheduling der Arbeitslast.

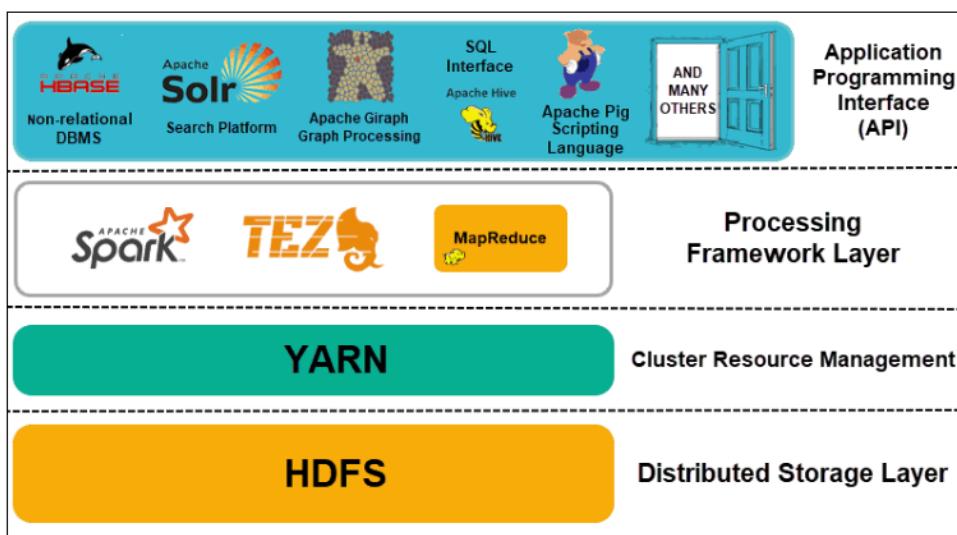


Abbildung 5: Ökosystem von Apache Hadoop [19]

Dabei besteht YARN aus dem ResourceManager und dem NodeManager. Der ResourceManager ermöglicht Anwendungen den Zugriff auf die Ressourcen innerhalb des Clusters. Als Ressourcen werden sogenannte Container verwendet, die aus CPU-Kernen, Arbeitsspeicher und System Dateien bestehen. Durch die physische Limitierung von CPU-Kernen und Arbeitsspeicher innerhalb eines Clusters kann nur eine feste Anzahl von Containern in dem Cluster gleichzeitig ausgeführt werden [20].

Für die Ausführung der Aufgaben innerhalb eines Knotens sind die NodeManager-Instanzen verantwortlich. Diese überwachen auch den Fortschritt und Status einer Aufgabe.

Der Application Manager jeder Anwendung wird in einem Container auf einem Slave Node deployed. Dabei überwacht dieser seine Applikation und sendet die gesammelten Informationen zu dem Resource Manager [21].

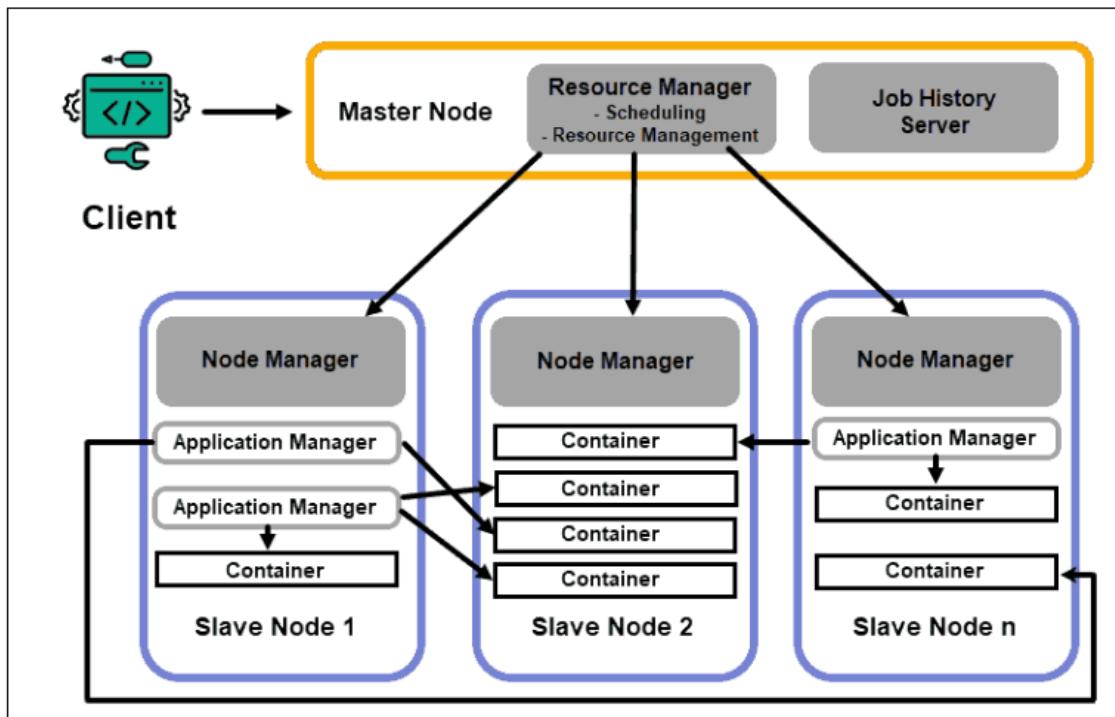


Abbildung 6: Apache Hadoop YARN Architektur [19]

Im Processing Framework Layer kann einerseits das Hadoop eigene MapReduce Framework eingesetzt werden, andererseits auch z. B. Apache Spark. Wie im Kapitel 2.1.1 beschrieben, kann Spark auch mit dem Cluster Manager YARN und dem Storage HDFS genutzt werden.

MapReduce ist Hadoops Framework zur Datenverarbeitung von großen Datensätzen auf einer Vielzahl von Computern. MapReduce besteht aus den zwei Funktionen Map und Reduce und ist darauf ausgelegt, Batchdaten zu verarbeiten. Innerhalb des Frameworks

werden Schlüssel-Paar Werte für die Verarbeitung verwendet und gibt diese dem Benutzer am Ende zurück. In der Map-Funktion wird eine Datenmenge in ein oder mehrerer Tupel (Key-Value) heruntergebrochen. Diese bilden das Ergebnis eines Algorithmus. In der Reduce-Funktion wird die Menge an Key-Value-Paaren, welche aus der Datenmenge generiert worden sind, auf eine kleine Menge mit den gleichen Keys reduziert [9].

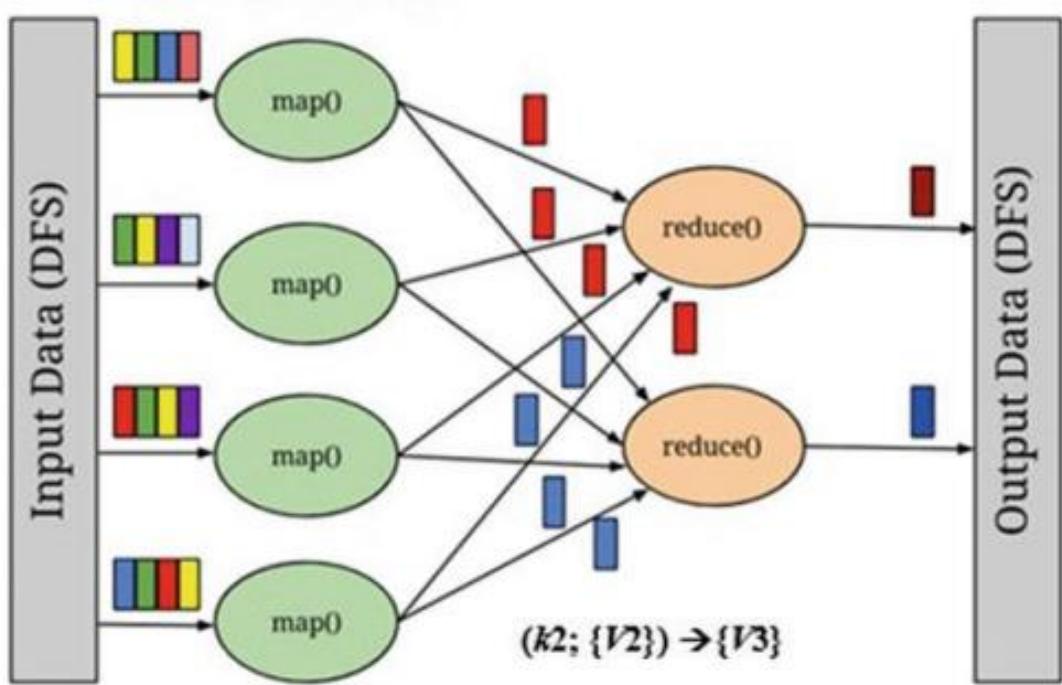


Abbildung 7: MapReduce Framework [14]

In diesem Artikel wurde Hadoop mit MPI kombiniert, um metagenomische Probleme zu lösen, die daten- und rechenleistungsintensiv sind [22]. Weiterhin haben K. Wang, Z. Wang, Bhatia, und Nordberg BioPig als analytisches Toolkit für große Sequenzdaten für Hadoop vorgestellt [23].

2.1.3. Weitere Frameworks zur Analyse von Genomdaten

QIIME2 (Quantitative Insights Into Microbial Ecology) ist eine open-source Pipeline, welche es ermöglicht, mit Hilfe von Rohdaten der DNA-Sequenzierung Mikrobiom-Analysen durchzuführen. Das Framework baut auf Python auf und kann somit in eine Python Umgebung zur Analyse integriert oder mittels CLI verwendet werden. Es steht für die Analyse eine Vielzahl von vordefinierten Skripten zur Verfügung. Ein Forscher kann, ohne selbst Algorithmen zu entwerfen, Analysen von Genomdaten durchführen [24].

BLAST (Basic Local Alignment Search Tool) ist ein Softwaretool, um nach Ähnlichkeiten innerhalb von Abschnitten bei Aminosäuresequenzen oder DNA zu suchen. Hierbei wird eine statistische Signifikanz für eine Übereinstimmung der vorhandenen Sequenzen mit einer Datenbank berechnet. Das Softwaretool muss zunächst als Package installiert werden und kann dann durch pyBlast [25] oder rBlast [26] in die Programmierumgebung für die Interaktion mit dem Tool integriert werden [27].

ASaiM (Auvergne Sequence analysis of intestinal Microbiota) ist ein open Source Framework zur Analyse von metagenomischen Daten, welches auf dem Galaxy Framework aufbaut. Dabei stellt Galaxy ein Frontend zur Verfügung, um z. B. Workflows zur Analyse aufzubauen oder auch ein Command-Line-Tool zur Interaktion mit dem Framework. Innerhalb der Galaxy Umgebung können mehr als 5500 Tools installiert werden, die der Forscher aktuell benötigt. Neben dem Frontend und der CLI kann mit Hilfe der BioBlend Library direkt aus einer Python Umgebung auf die API von Galaxy zugegriffen werden. Das ASaiM Framework enthält mehrere relevante Tools für die Analyse von Genomdaten sowie Workflows auf der Basis von Galaxy. Die gesamte Umgebung ist innerhalb eines Docker-Images verfügbar [28].

2.1.4. Zusammenfassung und Diskussion der Analyse Software

Als Open-Source Big-Data Analyse Software wurden in diesem Kapitel Apache Spark und Apache Hadoop vorgestellt. Für beide Lösungen gibt es Metagenomic Frameworks mit vorhandenen Algorithmen, jedoch werden beide Möglichkeiten nicht mehr aktiv weiterentwickelt und die letzten Commits liegen mehr als fünf Jahre zurück. In diesem Fall gilt es vor der Benutzung zu überprüfen, ob die darin enthaltenen Algorithmen immer noch einem aktuellen Stand entsprechen oder ob eigene Algorithmen auf Basis von Spark oder Hadoop entwickelt werden müssen. Neben den zwei Big-Data Analyse Tools wurden weitere Frameworks, die zur Analyse von Genomdaten genutzt werden können, vorgestellt. QIIME2 und BLAST sind Software Libraries, die zunächst in eine Python oder R Umgebung integriert werden müssen. Dadurch wird ein hoher Individualisierungsgrad der Software ermöglicht. Das ASaiM Framework auf der Basis von Galaxy ist eine Softwarelösung mit schon vorhandenem Frontend sowie REST-API und Libraries zur automatisierten Kommunikation mit dem Framework. Das Frontend bietet zwar sehr viele Einstellungsmöglichkeiten, jedoch sollen innerhalb eines Labores für einen Forscher beispielsweise vordefinierte Workflows zur Analyse bereitgestellt werden, ohne dass sich dieser in die vielfältigen Möglichkeiten von ASaiM einarbeiten muss. Für die gesamte Softwarelösung werden alle vorgestellten Analyse Softwaremöglichkeiten genutzt, um später eine vielfältige Analyse durch unterschiedliche Systeme zu ermöglichen. Aufgrund der Integration in Kubernetes und der Möglichkeit, mehr Speichermöglichkeiten außerhalb von HDFS zu benutzen, soll Spark als Big-Data Analyse Software genutzt werden. QIIME, BLAST und das ASaiM Framework werden ebenfalls genutzt.

Jedes der vier Analysetools Apache Spark, QIIME2, BLST und ASaiM muss in das zu entwickelnde System integriert werden. Außerdem müssen für jedes Tool Algorithmen ausgewählt oder auf Basis dieser entwickelt werden, mit denen die Analysen ausgeführt werden sollen. Daraus ergeben sich die folgenden acht verbleibenden Herausforderungen.

- VH1: Implementierung des Analysetools Apache Spark in die Architektur (FF1).
- VH2: Implementierung von Algorithmen zur Analyse von Genomdaten auf Basis von Apache Spark (FF1).
- VH3: Implementierung des Analysetools QIIME2 in die Architektur (FF1).
- VH4: Auswahl der Algorithmen von QIIME2 (FF1).
- VH5: Implementierung des Analysetools BLAST in die Architektur (FF1).
- VH6: Auswahl der Analysemethoden von BLAST sowie Datenbanken, mit denen die Sequenzen verglichen werden (FF1).
- VH7: Implementierung des Analysetools ASaiM in die Architektur (FF1).
- VH8: Auswahl der Analysemethoden von ASaiM (FF1).

2.2. Recherche zu Open-Source Speichermöglichkeiten von Genomdaten

Neben dem System zur Analyse und Sequenzierung von Genomdaten in Kapitel 2.1 ist das Speichern dieser Daten ebenfalls ein essenzieller Bestandteil für die Verarbeitung. Metagenomische Daten können unter anderem im FASTA [29] oder FASTQ [30] Format vorliegen. Beide sind textbasierte Dateien, welche Sequenzdaten enthalten. Diese Dateien können mehrere Megabyte bis Gigabyte groß werden. FASTQ Daten können beispielsweise im HMP [31] Portal gefunden werden. Solche Dateien in einer Datenbank zu speichern, kann je nach Datenbank möglich sein (mit oder ohne Konvertierung) oder die Datenbank unterstützt das Speichern von großen Dateien nicht (z. B. MongoDB Dokumente sind auf 16MB limitiert). Jedoch gibt es weitere Herausforderungen wie z. B. langsame Datenbank Queries, ein komplexerer Betrieb der Datenbank (große Backups, etc.) und ein größerer Arbeitsspeicherverbrauch. Alternativ können FASTA oder FASTQ Daten direkt auf einem Filesystem abgelegt und dort dem Analysetool zur Verfügung gestellt werden [32].

2.2.1. SAMBA

Das SAMBA Softwarepaket ist Teil des Projektes Software Freedom Conservancy und ist durch die GNU General Public Licence frei auf unterschiedlichen Plattformen verfügbar. SAMBA implementiert das Protokoll SMB (Server Message Block) und die spezielle Ausführung von SMB genannt CIFS (Common Internet File System) [33]. Dabei besteht das Softwarepaket aus den beiden Programmen smbd und nmbd. Diese bilden Datei- und Druckerfreigaben, Authentifizierung und Autorisierung und Namensauflösung ab. Die Dateifreigaben können direkt auf dem Host gemounted werden oder direkt als Docker oder Kubernetes Volume. Dadurch können mehreren Containern oder Pods die Dateien zugänglich gemacht werden und durch entsprechende Userberechtigungen die Daten geschützt werden [34].

2.2.2. Ceph

Ceph ist ein Open Source Tool, um Objektspeicher, Blockdevices und Dateispeicher bereitzustellen. Durch die horizontale Skalierung mittels eines Ceph Clusters ist es sehr gut skalierbar. Dabei besteht ein Ceph Cluster aus mehreren Ceph Nodes, welche untereinander die Daten replizieren und verfügbar machen [35]. Integriert werden kann der Ceph Blockspeicher in Kubernetes über den Ceph Container Storage Interface Treiber. In der Abbildung 8 ist der Technologiestack für den Kubernetes Use-Case abgebildet [36].

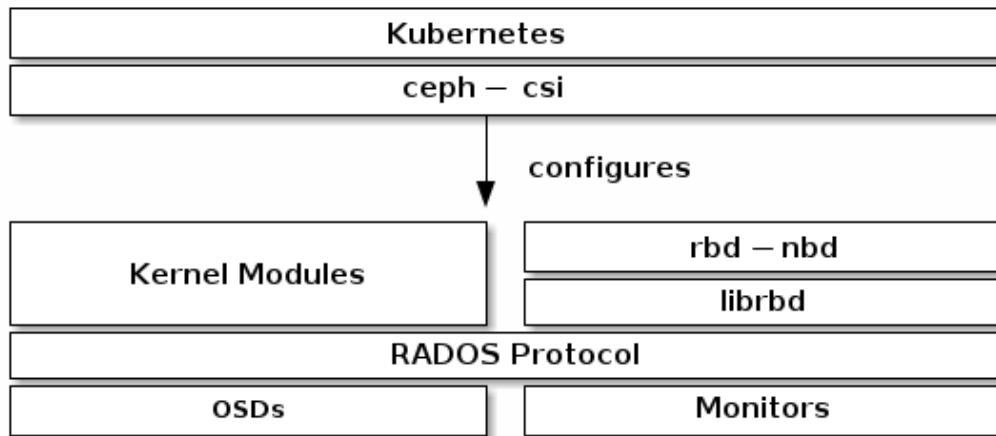


Abbildung 8: Ceph Kubernetes Stack [36]

Ceph basiert auf dem Reliable Autonomic Distributed Object Storage (RADOS) und beinhaltet die Komponenten Object Storage Device (OSD) und Monitor (MON).

Der Object Storage Daemon (OSD) ist für die Verwaltung der Daten zuständig. Darunter fällt das Speichern von Daten, die Replizierung sowie Wiederherstellung und Lastausgleich. Weiterhin überprüft dieser den Status von anderen OSDs mittels eines Heartbeats und stellt diese Informationen dem Monitor und Manager bereit. Jeder Node innerhalb eines Clusters betreibt einen OSD.

Monitors betreibt unterschiedliche Maps, welche für die Kommunikation zwischen den Nodes notwendig sind. Darunter fallen die OSD map, MDS map und CRUSH Map. Auch die Authentifizierung von Daemons und Clients wird über diesen abgewickelt.

RBD und librbd stellen in diesem Zusammenhang zwei Möglichkeiten dar, um über die Kernel Modules mit dem Objektspeicher zu interagieren. RBD ist dabei ein Client Tool und librbd eine Python Bibliothek [37].

2.2.3. HDFS

Das Hadoop Distributed File System (HDFS) ist Teil des Hadoop-Technologie-Stacks und gehört ebenfalls zur Apache Software Foundation. Es ist ein verteiltes Dateisystem, das fehlertolerant und für große Datenmengen ausgelegt ist. Die Fehlertoleranz wird durch den Einsatz von mehreren Servern erreicht. Dabei wird auf eine Master/Slave Architektur zurückgegriffen. Jedes HDFS Cluster besteht aus einem NamenNode. Dieser ist der Master-Server und verwaltet den Namensraum des Dateisystems. Weiterhin wird durch diesen der Zugriff auf Dateien durch Clients gewährt und zusätzlich das Öffnen, Schließen und die Namensänderung von Dateien oder Ordnern durchgeführt.

Neben dem NameNode gibt es mehrere DataNodes. Pro verfügbarem Node wird ein DataNode deployed. Das Bereitstellen von Lese- und Schreibzugriffen von dem Filesystem zu Clients ist eine Aufgabe der DataNodes. Von dem NameNode werden Instruktionen an die DataNodes weitergeleitet. Diese können das Erstellen und Löschen oder die Replikation von Blöcken beinhalten.

Die Daten werden in sogenannten Blöcken gespeichert. Diese sind Sequenzen aus der eigentlichen Datei und werden über das verfügbare Cluster repliziert [38].

In der Abbildung 9 ist die Architektur von HDFS abgebildet.

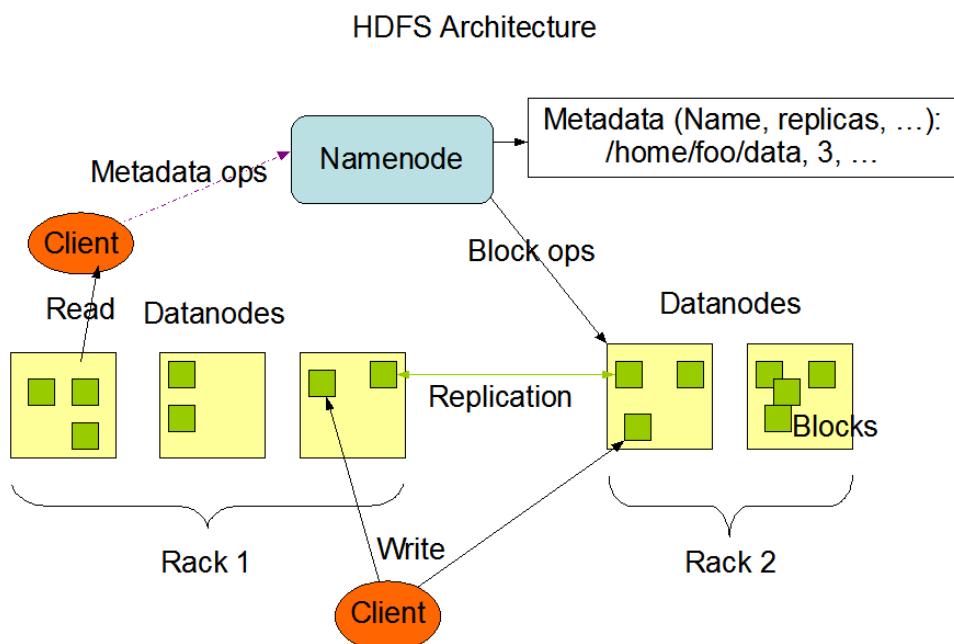


Abbildung 9: HDFS Architektur [38]

2.2.4. Zusammenfassung und Diskussion der Speichermöglichkeiten von Genomdaten

Im Kapitel 2.2 wurden die Speichermöglichkeiten von Genomdaten erläutert. Dabei wurde zunächst der Use-Case mit metagenomischen Daten erläutert und die Datenformate FASTQ und FASTA erklärt. Durch die Größe ist eine Speicherung dieser innerhalb einer Datenbank nur bedingt möglich, weshalb in diesem Kapitel unterschiedliche Möglichkeiten zur Bereitstellung eines Filesystems aufgezeigt wurden. HDFS ist das Hadoop eigene Filesystem, welches von Apache Hadoop und Apache Spark zur Ablage von BigData unterstützt wird. Durch das Ausschließen einer BigData Lösung in Kapitel 2.1.4 muss nicht zwingend auf HDFS zurückgegriffen werden. SAMBA und das darin enthaltene SMD / CIFS Protokoll stellen eine gute Möglichkeit dar, ein Dateisystem über das Netzwerk mehrerer Clients bereitzustellen. Für die Skalierung und Erweiterbarkeit ist SAMBA jedoch suboptimal und nur für kleine Use-Cases zu verwenden. Ceph und die Integration in Kubernetes ist eine sehr gute Lösung, um einen verteilten Storage unterhalb von Kubernetes zu ermöglichen. Die horizontale Erweiterbarkeit und direkte Integration in Kubernetes über den CSI-Treiber machen Ceph sehr einfach integrierbar. Bei einer Implementierung wird ein Cluster-Node des Ceph Clusters auf jedem Kubernetes Node deployed, was eine optimale Nutzung der vorhandenen Hardware darstellt. Aus diesen Gründen soll Ceph als Speicherlösung genutzt werden. Um CEPH als Speicherlösung in das verteilte System zu implementieren, bleibt folgende Herausforderung:

VH9: Implementierung der Speicherlösung CEPH in die Architektur (FF2).

Des Weiteren muss für einen einheitlichen Zugriff auf die CEPH Speicherlösung eine API als Schnittstelle implementiert werden:

VH10: Implementierung einer API, um den Zugriff auf den Speicher durch die Benutzungsschnittstelle zu abstrahieren (FF2).

2.3. Recherche zur Erstellung von Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems

In den folgenden Unterkapiteln werden Architekturen und Entwurfsmuster zur Erstellung einer Benutzungsschnittstelle sowie Möglichkeiten beschrieben, um einzelne Komponenten des verteilten Systems miteinander zu lose verbinden und Analyseworkflows zu erstellen.

2.3.1. Model View Controller

Das Model View Controller (MVC) Pattern ist ein Entwurfsmuster in der Softwareentwicklung. Dabei wird die Software in die drei Bestandteile Model, View und Controller aufgeteilt. Das Model beinhaltet die Daten, die View, wie die Daten angezeigt werden, und der Controller die Logik zur Bearbeitung der Daten und der View. In der Abbildung 10 ist der Aufbau des Patterns zu sehen [39].

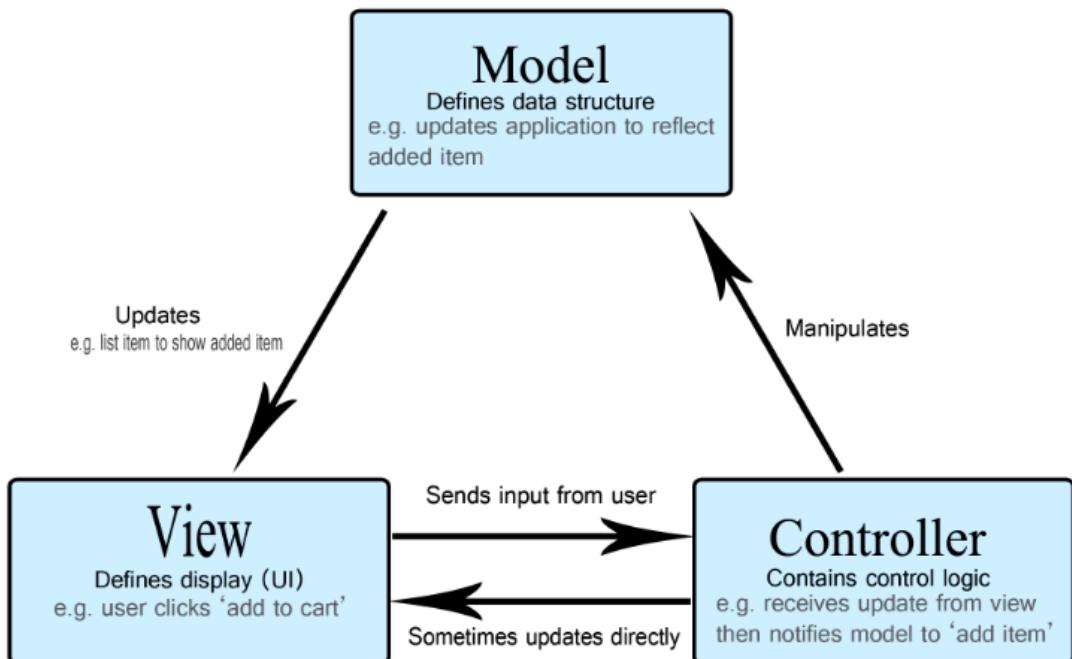


Abbildung 10: Model View Controller Pattern [39]

Eine Ableitung des MVC-Patterns ist das Model View ViewModel- Pattern. Auf diesem Pattern baut das Framework Vue.js auf. Die View ist in diesem Zusammenhang das Document Object Model (DOM). Das Model beinhaltet die Daten in Form von JavaScript Objekten und das ViewModel verbindet beide Komponenten miteinander [40].

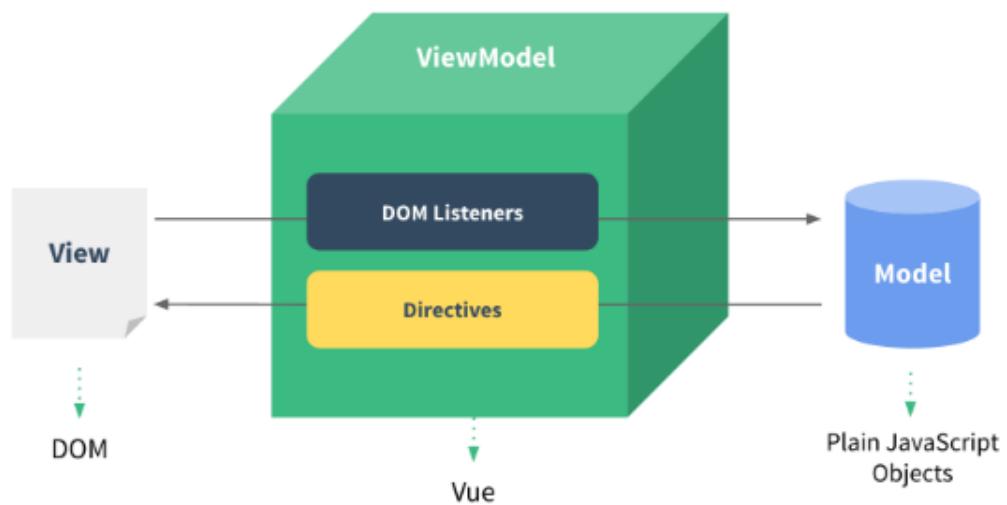


Abbildung 11: MVVM-Pattern am Beispiel von Vue.js [40]

2.3.2. Representational State Transfer

Representational State Transfer (REST) ist eine Architektur, die oft bei Webapplikationen eingesetzt wird. Dabei können Applikationen lose voneinander gekoppelt über ein Netzwerk mittels des HTTP Protokolls kommunizieren. Wenn ein Application Interface (API) die sechs Eigenschaften von REST implementiert, wird diese REST-API genannt.

Einheitliches Interface: Für die Ressourcen eines Systems, welche nach außen sichtbar sein sollen, muss ein einheitliches Application Interface vorhanden sein.

Client-Server: Eine Client / Server Architektur, welche unabhängig voneinander arbeitet.

Stateless: Es ist keine Session verfügbar, jeder HTTP Request eines Clients wird als neue Session behandelt.

Cacheable: Eine Antwort muss als Cacheable oder nicht Cacheable deklariert sein. Der Client kann eine Antwort zu einem späteren Zeitpunkt wiederverwenden, wenn diese als Cacheable deklariert wurde.

Layered System: Die Komponenten eines Systems (Datenbank, API, etc.) können auf unterschiedlichen Servern deployed sein. Der Client kann z. B. nicht einsehen, wie die Datenverarbeitung hinter der API weitergeht.

Code on demand (Optional): Neben Daten in Form von XML oder JSON kann ein ausführbarer Code zurückgegeben werden [41].

2.3.3. Event Driven Architecture

Die Event Driven Architecture (EDA) ist eine Softwarearchitektur und bildet ab, dass unterschiedliche Systeme mittels Events miteinander kommunizieren / angesteuert werden. Ein Event ist ein Wechsel eines Status oder ein Ereignis innerhalb eines Systems. Events können durch den Benutzer manuell oder durch externe Quellen automatisiert ausgelöst werden. Durch die Kommunikation mit Events wird eine lose Kopplung geschaffen, da ein Event Producer nicht weiß, was sein Event in der weiteren Verarbeitung auslöst oder wie Event Consumer beim Erreichen eines Events reagieren. Aus diesem Grund ist EDA eine Softwarearchitektur, die sich gut für verteilte Anwendungen eignet. Neben dem Event-Producer gibt es Event-Consumer. Diese warten auf eingehende Events über Event-Kanäle. Der Event-Manager stellt diese Event-Kanäle zur Verfügung und ermöglicht das Weiterleiten von Events.

Für die Event-Kanäle gibt es das Pub/Sub-Modell und das Event-Streaming-Modell. Im Pub/Sub-Modell müssen Event-Consumer einzelne Event-Kanäle abonnieren und ein Event-Producer erstellt gezielt Events für diesen Event-Kanal. Die Events innerhalb eines Kanals erreichen somit alle Abonnenten. Alle Event-Consumer, welche einen Kanal nicht abonniert haben, bekommen auch dessen Events nicht. Das ist der große Unterschied zwischen beiden Modellen. Innerhalb des Event-Streaming-Modells sind alle Events in einem Protokoll vorhanden. Es werden keine einzelnen Protokolle (Event-Streams) abonniert, sondern alle Event-Consumer haben Zugriff auf das gleiche Protokoll [42].

Ein weiterer Aspekt bei einer EDA Architektur ist Event-Sourcing. Event-Sourcing beschreibt in diesem Zusammenhang, dass alle Events einer Applikation persistiert werden. Somit ist einerseits gewährleistet, dass alle Events abgefragt und andererseits vergangene Zustände der Applikation durch das erneute Ausführen von Events wiederhergestellt werden können [43].

Als Open-Source Software für die Umsetzung einer EDA kann Apache Kafka genutzt werden. Kafka unterstützt die vorgestellten Möglichkeiten einer EDA und ermöglicht zusätzlich das Streaming von Daten [44].

2.3.4. Microservices

Microservices ermöglichen es, große und komplexe Softwarearchitekturen modular zu gestalten. Dabei sind Microservices kleine Module, welche für eine bestimmte Aufgabe innerhalb der Architektur zuständig sind. Innerhalb eines Entwicklungsprozesses und Betriebs der Softwarelandschaft sind Microservices eigenständig und können unabhängig voneinander entwickelt, betrieben und skaliert werden.

Container erlauben in diesem Zusammenhang, dass Microservices einfach gebaut, deployed und betrieben werden können. Container und Container-Orchestrierung werden im Kapitel 2.4 genauer erläutert [45].

2.3.5. Apache Airflow

Apache Airflow ist eine Open Source Software der Apache Software Foundation. Diese ermöglicht die Definition von Workflows mit Hilfe der Python Programmiersprache. Neben der Definition von Workflows können diese mittels eines Schedulers automatisch oder über eine API und Benutzungsschnittstelle manuell ausgeführt werden. Für einen Workflow können unterschiedliche Tasks definiert werden, welche dann einen Workflow abbilden [46].

2.3.6. Zusammenfassung und Diskussion der Softwarearchitektur für eine Benutzungsschnittstelle

Im Kapitel 2.3 wurden unterschiedliche Architekturen und Vorgehensweisen zur Erstellung eines Frontends beschrieben. Darunter fällt auch die Kommunikation der gesamten Komponenten der Softwarelösung. Microservices erlauben die lose Kopplung und Skalierbarkeit sowie Austauschbarkeit der Komponenten innerhalb der gesamten Softwarelösung. Deshalb werden alle Komponenten der Softwarelösung als Microservices aufgebaut.

Als Frontendtechnologie kommt Vue.js mit dem MVVM Pattern zum Einsatz. Dies ist auf die persönliche Präferenz zurückzuführen, da mit anderen Frontendtechnologien weniger Erfahrungen vorliegen.

VH11: Implementierung einer Benutzungsschnittstelle, um Daten hochzuladen, Workflows zu starten und die Ergebnisse der Workflows anzuzeigen (FF3).

Die einzelnen Komponenten sollen über die Event Driven Architecture miteinander kommunizieren, welche ebenfalls die lose Kopplung der Komponenten ermöglicht und bei einem Systemausfall über das Event-Sourcing unterschiedliche Analyseschritte wiederholt werden können. Für die Umsetzung der Event Driven Architecture wird Apache Kafka genutzt.

VH12: Modellierung der Event Driven Architecture (FF3).

Für die Steuerung von Events und das Starten von Analyseworkflows wird Apache Airflow in die Architektur integriert. Workflows werden innerhalb von Apache Airflow definiert.

VH13: Integration von Apache Airflow in die Architektur, um Workflows auszuführen (FF3).

VH14: Implementierung von Workflows in Apache Airflow (FF3).

Am Ende eines Workflows muss für den Analyst / Labormitarbeiter ein Report mit den Ergebnissen erstellt werden.

VH15: Implementierung eines Microservices, welcher den Report am Ende eines Analyseworkflows erstellt (FF3).

2.4. Recherche zu Deployment-Methoden für die gesamte Softwarelösung zur Analyse von Genomdaten.

In den folgenden Kapiteln werden Deployment-Methoden von Softwarepaketen vorgestellt.

2.4.1. Container

Ein Container ist ein isoliertes logisches Konstrukt, das die eigentliche Software und alle Abhängigkeiten dieser beinhaltet. Um einen Container starten zu können, wird eine Container-Runtime benötigt, z. B. Docker. Dabei wird das Betriebssystem virtualisiert. Wird Linux als Betriebssystem des Servers verwendet, hat jeder Container Zugriff auf den Linux-Kernel, um die CPU, Arbeitsspeicher anzufordern und Schreib- sowie Lesezugriffe auf die Festplatte durchzuführen. Eine traditionelle Virtualisierung (VM) virtualisiert die darunterliegende Hardware. Das Fehlen eines vollwertigen Betriebssystems sowie zusätzliche Software machen Container leichtgewichtig. Innerhalb einer Container-Runtime können mehrere Container gestartet werden und parallel isoliert betrieben werden. Durch die Isolation und das Beinhalten der Software sowie alle Abhängigkeiten kann ein Container auf unterschiedlichen Servern gestartet werden, ohne zusätzliche Installationen außerhalb der Container-Runtime vorzunehmen. Durch die Isolation und das Ausführen von nur einer Software lassen sich mit Containern Microservices umsetzen.

Zusammengefasst sind die Vorteile von Containern die Leichtgewichtigkeit, Plattformunabhängigkeit, sowie die Möglichkeit der Erstellung moderner Softwarearchitekturen, z. B. Microservices [47].

Eine sehr verbreitete Container-Runtime ist Docker. Im Hintergrund von Docker arbeitet containerd, ein Open Source Standard der Cloud Native Computing Foundation [48].

In der Abbildung 12 ist die Architektur im Vergleich mit einer VM abgebildet.

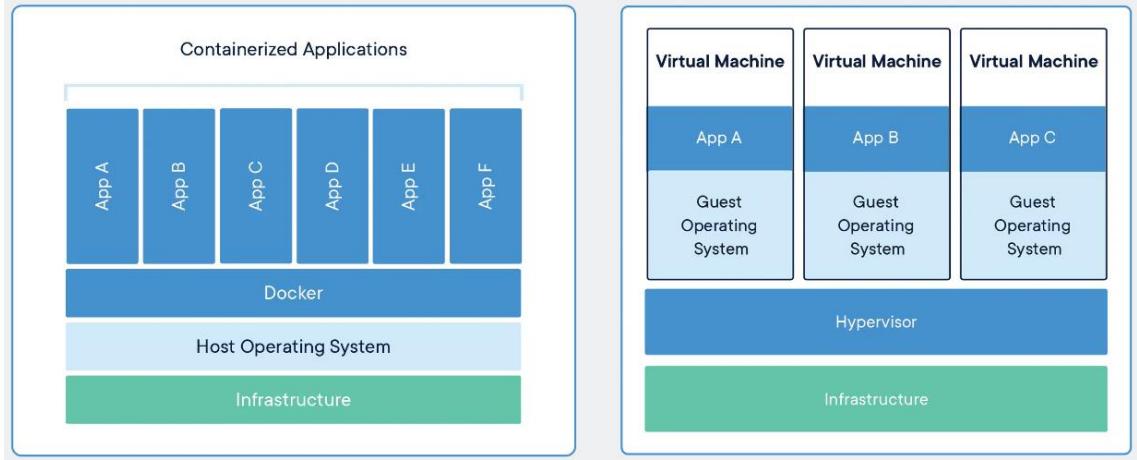


Abbildung 12: Container vs VM [48]

Um einen Docker-Container zu starten, wird ein Docker-Image benötigt. Das Image verfügt über die beschriebene Software, sowie alle Abhängigkeiten für einen Betrieb dieser. Für die Erstellung eines Images wird ein Dockerfile benötigt. Darin enthalten ist der „Bauplan“ des Images in einem Textformat. Container-Images werden in sogenannten Registries gespeichert und können so von einem zentralen Ort bezogen werden. DockerHub ist in diesem Beispiel eine große öffentliche Registry, in welcher Docker-Images von Softwareherstellern oder privaten Usern / Communities allen zur Verfügung gestellt werden [47].

2.4.2. Container Orchestration

Container Orchestration baut auf einer Container-Runtime auf und bildet mit mehreren Container-Runtimes ein Cluster. Eine Open-Source Variante von Container Orchestration ist Kubernetes. Durch die Portabilität von Containern können diese, wie in Kapitel 2.4.1 beschrieben, auf unterschiedlichen Servern mit gleicher Container Runtime deployed und betrieben werden. Um eine Anwendung auf Kubernetes zu deployen, wird nicht mehr ein Container auf einem Server gestartet, sondern die Konfiguration des Deployments innerhalb eines YAML oder JSON definiert. Dort enthalten ist z. B. die Art des Deployments, wie viele Instanzen des Containers innerhalb des Clusters betrieben werden oder über welchen Weg diese Instanzen von außen erreichbar sein sollen. Nach der Installation von Kubernetes ist die darunterliegende Hardware mit der Unterstützung der Container-Runtime abstrahiert und es wird lediglich mit der Deploymentbeschreibung gearbeitet. Kubernetes kümmert sich dann um den Betrieb des Deployments [47].

2.4.3. Zusammenfassung und Diskussion von Deployment-Methoden von Softwarelösungen

Im Kapitel 2.4 wurden die Deploymentmethoden von Containern auf Docker und Container Orchestration auf Kubernetes vorgestellt und mit dem klassischen Deployment direkt auf einem Server / VM verglichen. Für das weitere Vorgehen wird das Container Orchestration Deployment aufgrund der Skalierbarkeit verwendet und alle Services der Softwarelösung entweder durch vorhandene oder selbstgebaute Container bereitgestellt. Daraus ergibt sich folgende verbleibende Herausforderung:

VH16: Aufbau der gesamten Komponenten innerhalb eines Kubernetes Clusters (FF4).

2.5. Zusammenfassung der verbleibenden Herausforderungen

In den vorangegangenen Kapiteln wurde die Observation nach Nunamaker für die definierten Forschungsziele in Kapitel 1.5 vorgenommen. In Kapitel 2.1 wurden zwei Open-Source Big-Data Lösungen beschrieben. Für beide Lösungen gibt es Frameworks zur Analyse von metagenomischen Daten, aber es gibt keine Pipeline oder Frameworks, an denen aktiv gearbeitet wird. Apache Spark wird dem Hadoop Framework durch die Unterstützung von Kubernetes als Scheduler und die Verwendung von unterschiedlichen Speichermöglichkeiten, z. B. CEPH, vorgezogen. BLAST und QIMME sind hingegen gängige Libraries für die Analyse von metagenomischen Daten. Als komplettes Framework wurde ASaiM vorgestellt, welches auf dem Galaxy Framework aufbaut. Neben der Analyse bietet dieses auch die Möglichkeit zur Erstellung von Workflows und einer grafischen, webbasierten Benutzungsoberfläche. Das Frontend ist jedoch für den Betrieb innerhalb eines Labores aufgrund der komplexen Einstellmöglichkeiten ungeeignet. Aus diesem Grund kann das Backend mit Hilfe einer REST-API oder der BioBlend Library auch ohne das Frontend zur Analyse von Genomdaten angesteuert werden. Alle der vorgestellten Analysemöglichkeiten werden genutzt. Weiterhin wurden in Kapitel 2.2 die Speichermöglichkeiten für Genomdaten analysiert. Aufgrund der Größe der Daten werden diese auf einem Filesystem gespeichert. CEPH wird als Speicherlösung eingesetzt. Softwarearchitekturen wurde in Kapitel 2.3 erläutert. Einzelne Komponenten werden als Microservice implementiert, um so die einfache Erweiterbarkeit des verteilten Systems zu ermöglichen. Die Microservices kommunizieren mittels einer Event-Driven-Architecture miteinander. Als Tool der Event-Driven-Architecture kommt Apache Kafka zum Einsatz. Die Analyseworkflows werden innerhalb von Apache Airflow definiert, über die Apache Airflow REST-API gestartet und deren Zustand überwacht. Um weiterhin modular und unabhängig von der Infrastruktur zu sein, hat sich als Deployment-Methode für die späteren Microservices Container bzw. die Container Orchestration, welche in Kapitel 2.4 beschrieben wurde, bewährt und die gesamte Softwarelösung wird innerhalb eines Kubernetes-Clusters bereitgestellt. Folgende verbleibende Herausforderungen ergeben sich nach der „Observation“:

- VH1: Implementierung des Analysetools Apache Spark in die Architektur (FF1).
- VH2: Implementierung von Algorithmen zur Analyse von Genomdaten auf Basis von Apache Spark (FF1).
- VH3: Implementierung des Analysetools QIIME2 in die Architektur (FF1).
- VH4: Auswahl der Algorithmen von QIIME2 (FF1).
- VH5: Implementierung des Analysetools BLAST in die Architektur (FF1).
- VH6: Auswahl der Analysemethoden von BLAST sowie Datenbanken, mit denen die Sequenzen verglichen werden (FF1).
- VH7: Implementierung des Analysetools ASaiM in die Architektur (FF1).
- VH8: Auswahl der Analysemethoden von ASaiM (FF1).
- VH9: Implementierung der Speicherlösung CEPH in die Architektur (FF2).
- VH10: Implementierung einer API, um den Zugriff auf den Speicher durch die Benutzungsschnittstelle zu abstrahieren (FF2).
- VH11: Implementierung einer Benutzungsschnittstelle, um Daten hochzuladen, Workflows zu starten und die Ergebnisse der Workflows anzuzeigen (FF3).
- VH12: Modellierung der Event Driven Architecture (FF3).
- VH13: Integration von Apache Airflow in die Architektur, um Workflows auszuführen (FF3).
- VH14: Implementierung von Workflows in Apache Airflow (FF3).
- VH15: Implementierung eines Microservices, welcher den Report am Ende eines Analyseworkflows erstellt (FF3).
- VH16: Aufbau der gesamten Komponenten innerhalb eines Kubernetes Clusters (FF4).

3. Modellierung

Im vorherigen Kapitel wurde der aktuelle Stand der Wissenschaft und Technik für ein verteiltes System zur Analyse von Genomdaten vorgestellt und verbleibende Herausforderungen 1-16 wurden definiert. In diesem Kapitel geht es um das Theory Building, welches im Kapitel 1.4 der Nunamaker Methodik vorgestellt wurde. Dabei werden die Erkenntnisse aus Kapitel 2 genutzt, um ein Modell für ein verteiltes System zu entwerfen. Alle Forschungsziele, welche im Kapitel 1.6 nach X.1/TB sortiert worden sind, haben eigene Unterkapitel.

Das gesamte Kapitel basiert auf dem User Centered System Design nach Norman und Draper [49]. In der Abbildung 13 sind die vier Phasen des Design Ansatzes abgebildet.

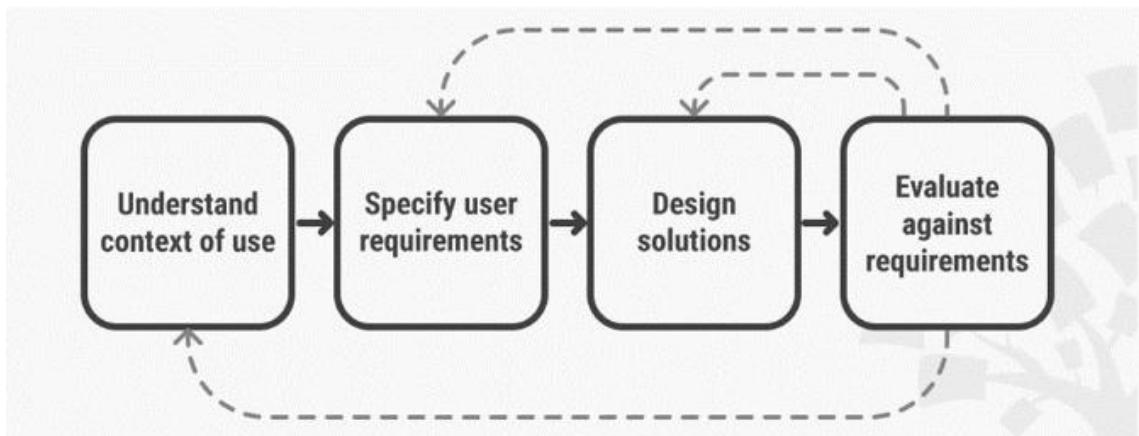


Abbildung 13: Die vier Phasen des User Centered System Design Ansatzes [50]

Ziel des Ansatzes ist es, den Benutzer in den Fokus in jeder Phase des Designs zu stellen und dadurch eine Lösung zu schaffen, welche für den Benutzer am besten ist. Die vier Phasen bilden einen iterativen Prozess. Dabei wird zunächst versucht, bestmöglich zu verstehen, in welchem Kontext die Software / Lösung in Zukunft verwendet werden soll. Danach erfolgt das Aufnehmen der Benutzeranforderungen, z. B. durch Interviews. Der nächste Schritt stellt das Hervorbringen von unterschiedlichen Designlösungen dar. Zum Schluss werden die aufgenommenen Benutzeranforderungen mit dem vorgeschlagenen Design evaluiert [50].

Im Kapitel 3.1 werden die ausgewählten Analysewerkzeuge aus Kapitel 2.1 in das verteilte System integriert (VH1, 3, 5, 7). Darauffolgend wird im Unterkapitel 3.2 gezeigt, wie die Speicherlösung CEPH aufgebaut und in dem Kubernetes-Cluster verfügbar gemacht wird (VH9, 10). Der Aufbau eines möglichen Frontends zur Interaktion mit dem System wird in Kapitel 3.3 und die im Hintergrund notwendigen Systeme Kafka, Airflow und der Reporting Microservice und mit Use-Case Diagrammen der grundsätzliche Ablauf erklärt (VH11, 12, 13, 15). Im Kapitel 3.4 wird auf Basis von Kubernetes die Gesamtarchitektur abgebildet (VH16).

Ziel ist es, die grundlegende Architektur eines verteilten Systems sowie Kommunikation der einzelnen Microservices mittels Events und grundlegende Erstellung eines Frontends aufzuzeigen. Diese resultierende Architektur wird im weiteren Verlauf der Arbeit „Genomic Explore Framework“ (GEF) genannt. Dabei wird nicht auf einzelne Algorithmen, wie z. B. Erstellung eines spezifischen Algorithmus für Spark zur Analyse von Genomdaten oder finale Erstellung eines Frontends mit User-Requirements, eingegangen. In der Abbildung 14 ist das Use-Context Diagramm des GEF abgebildet. Die Use-Cases ergeben sich automatisch anhand der Zielsetzung des Systems. Ein Benutzer des Systems, welcher Genomdaten analysieren möchte, muss diese zunächst dem verteilten System bereitstellen. Dann muss dieser die gewünschte Analyse auswählen und starten. Am Ende der Analyse benötigt der Benutzer einen Report, welcher dann für weitere Analysezwecke / Interpretationen außerhalb des verteilten Systems genutzt werden kann.

Alle resultierenden Modelle werden mit fortlaufender Nummer gekennzeichnet (M1-Mn)

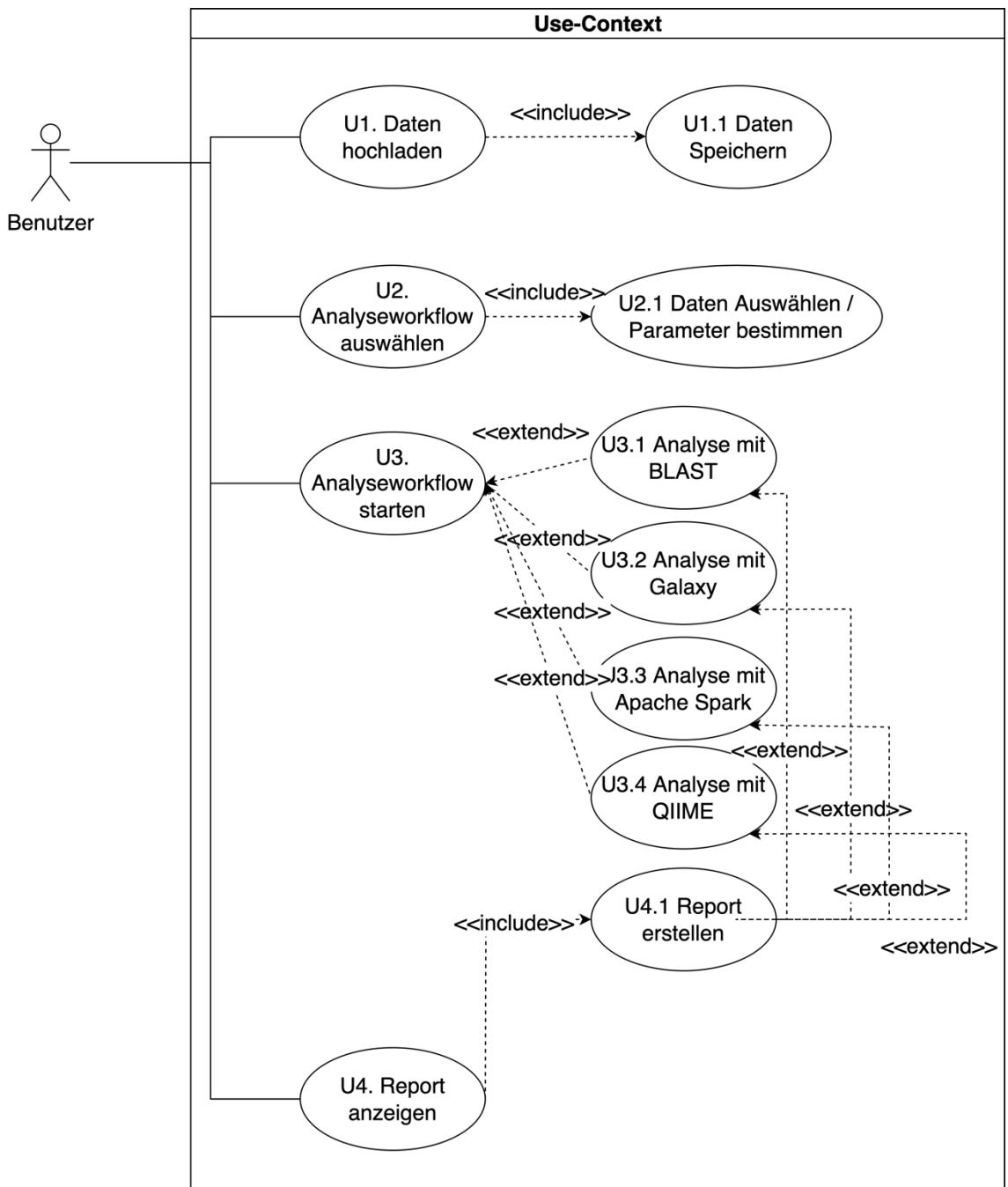


Abbildung 14: Use-Context Diagramm des Genomic Explore Frameworks (M1)

M1: Use-Context Diagramm GEF (FF1-4, löst VH11)

In der Abbildung 15 ist das Komponentendiagramm des GEFs abgebildet. Grundlegend soll dieses die Interaktion zwischen den einzelnen Komponenten mittels Events und REST-Zugriffen aufzeigen. Aufbauend auf VMs oder physischen Servern, die zu einem Kubernetes Cluster und einem CEPH Storage Cluster zusammengeschlossen sind, wird die gesamte Architektur aufgebaut. Im Mittelpunkt steht hierbei die „Event Streaming Platform“ (Apache Kafka) zur Realisierung der Event-Driven-Architecture und die dabei

resultierende lose Kopplung der Komponenten. Apache Airflow spielt ebenfalls eine zentrale Rolle in der Event-Driven-Architecture, da dort eine Event Orchestrierung (Abfolge von mehreren zusammenhängenden Events) modelliert werden kann. Links zu sehen ist das User-Frontend von dem einerseits Daten in die Speicherlösung hochgeladen werden und andererseits Analyse Workflows in Apache Airflow gestartet werden können. Rechts zu sehen sind die ausgewählten Analysetools und ein Microservice für die Ansteuerung und Adaption für die Event-Driven-Architecture. Ebenfalls abgebildet ist der Reporting Service, der am Ende eines Analyse-Workflows die Ergebnisse für den Benutzer des Systems in einen Report verpackt.

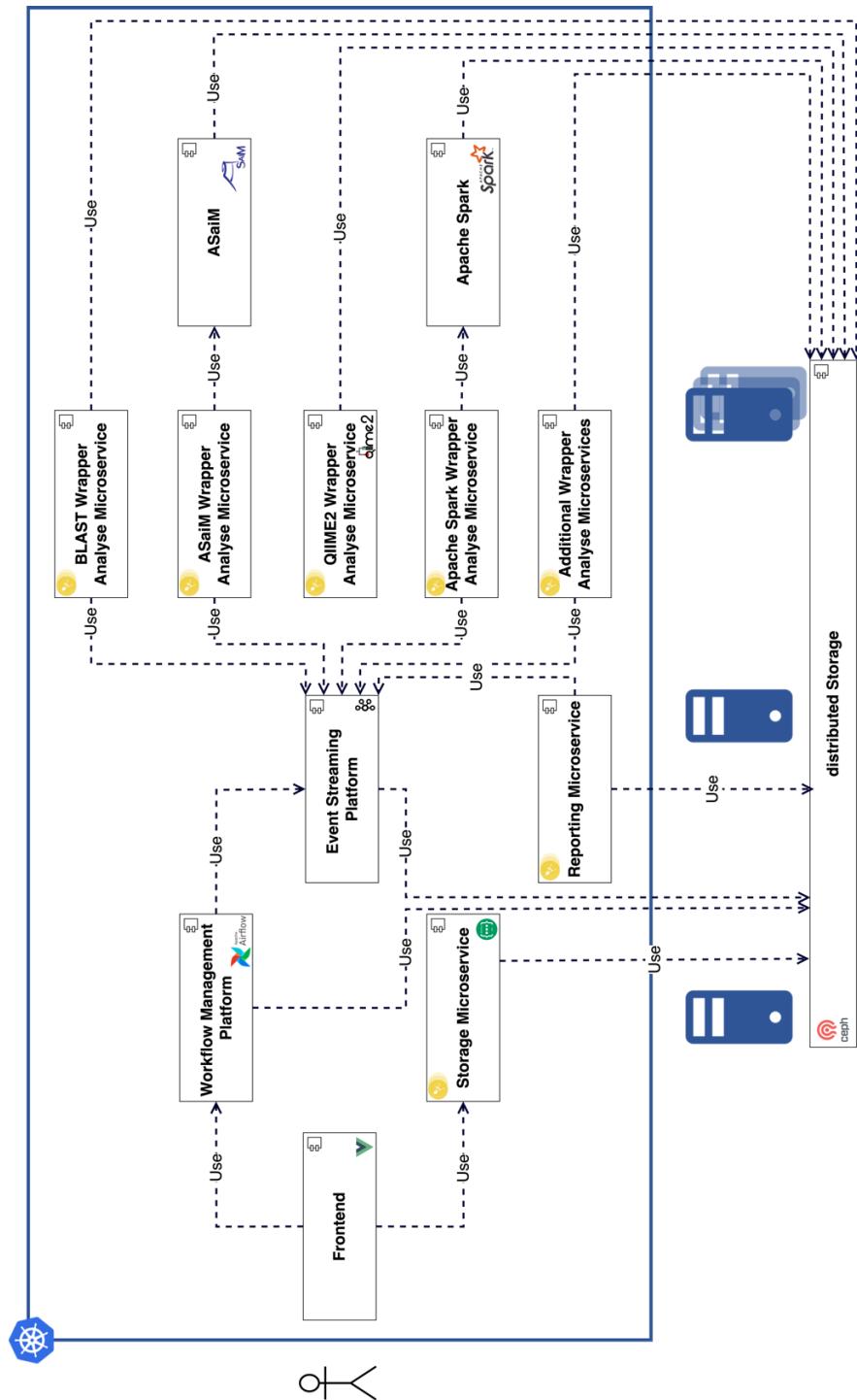


Abbildung 15: Komponentendiagramm des Genomic Explore Frameworks (M2)

M2: Komponentendiagramm GEF (FF1-4, löst VH12)

Use-Cases sowie Ablaufdiagramme und Erläuterungen der Architektur werden in den nachfolgenden Unterkapiteln beschrieben.

3.1. Integration der Analyse Tools

In Kapitel 2.1 wurden die Analyse-Tools vorgestellt und für die weitere Arbeit Apache Spark, ASaIM, BLAST und QIIME ausgewählt. Herausfordernd ist hierbei einerseits die Intergration in das Kubernetes-Cluster und andererseits die Einbettung in die Event-Driven-Architecture. In der Zielarchitektur werden die Workflows mittels Events gestartet. Keines der vorgestellten Analysetools unterstützt diese Art von Architektur direkt. Aus diesem Grund muss ein selbstentwickelter Microservice vor jedes Analysetool vorgeschaltet werden. Dieser übernimmt die Funktion eines Adapters, der die Events von Apache Kafka interpretieren kann und darauffolgend die Analysen mittels der verfügbaren APIs startet. Adressiert wird in diesen Unterkapiteln die FF1 und die VH 1, 3, 5, und 7.

3.1.1. Apache Spark

Apache Spark ist ein mögliches Analysetool, welches in der zukünftigen Architektur zur Analyse von Genomdaten genutzt werden soll. In der Abbildung 16 ist die Architektur von Apache Spark mit beiden Microservices dargestellt.

Apache Spark muss in diesem Fall nicht direkt in das Cluster deployed werden, da Kubernetes die Funktion als Scheduler von Jobs übernimmt. Für die Integration in die Event-Driven-Architecture werden zwei Microservices benötigt. Ein Microservice nimmt die Events für den Start einer Analyse mit Apache Spark entgegen und übermittelt den Request mit dem dazu vorgesehenen Algorithmus an die Kubernetes API. Diese deployed dann die notwendigen Pods zur Analyse. Der zweite Microservice überwacht die laufenden Jobs und erstellt ein Event, wenn ein Job beendet worden ist (erfolgreich oder nicht erfolgreich).

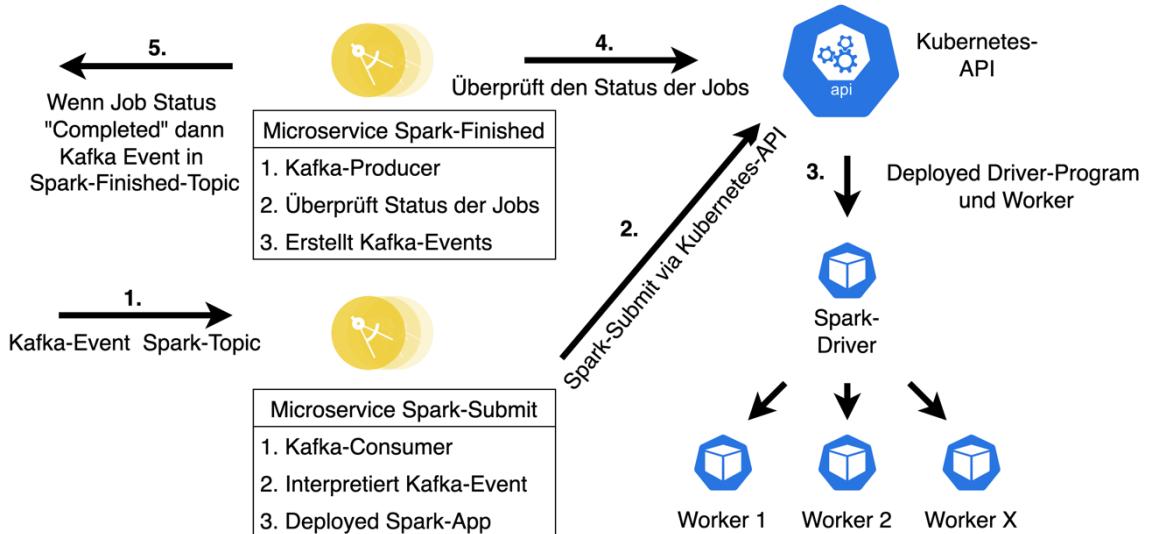


Abbildung 16: Architekturdiagramm Apache Spark Integration in das Genomic Explore Framework (M3)

M3: Architekturdiagramm Apache Spark Integration (FF1, löst VH1)

3.1.2. QIIME2

Neben der Interpretation / Erstellung von Kafka-Events wird QIIME2 innerhalb des Microservices installiert und kann dort entweder als CLI oder via API, z. B. in einem Python Script, verwendet werden. In der Abbildung 17 ist die Integration von QIIME2 inklusive des Microservices zu sehen.

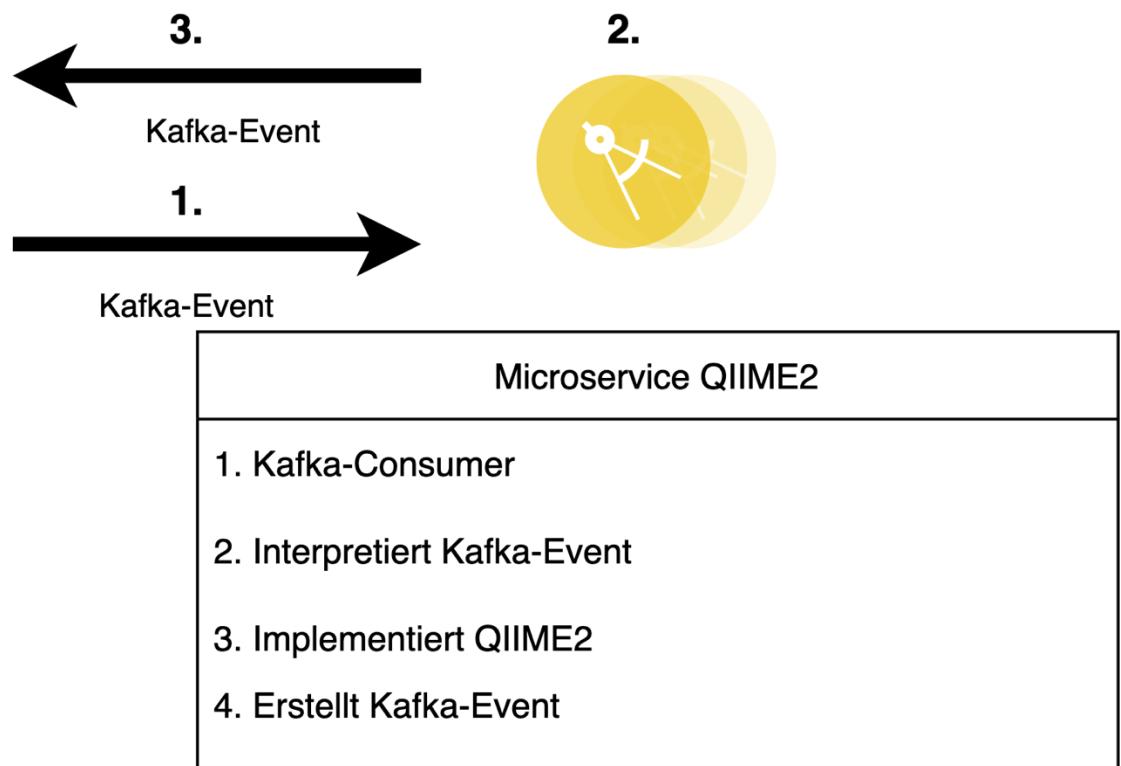


Abbildung 17: Architekturdiagramm QIIME2 Wrapper (M4)

M4: Architekturdiagramm QIIME2 Wrapper (FF1, löst VH3)

3.1.3. BLAST

Wie im Kapitel 2 beschrieben, gibt es für eine Analyse mit BLAST zwei unterschiedliche Möglichkeiten. Einerseits die lokale Analyse und andererseits die Analyse mit dem Public verfügbaren Service von BLAST. Für eine zuverlässige lokale Analyse müssen die Datenbanken von BLAST aktualisiert werden. Durch das Benutzen der Public API ist dies nicht erforderlich. Wenn der Public Service mehr als 100x benutzt wird, werden die zukünftigen Analysen durch langsamere Pipelines durchgeführt. Der Public Service priorisiert interaktive Benutzer (Webinterface) und je nach Anzahl der Analysen von anderen Benutzern kann es zu Laufzeitunterschieden kommen [51]. Aus diesen Gründen wird die Analyse mit BLAST lokal durchgeführt. Dabei wird innerhalb des Microservices die BLAST installiert und die Interpretation / Erstellung der Kafka-Events vorgenommen. In der Abbildung 18 ist die Architektur von BLAST abgebildet.

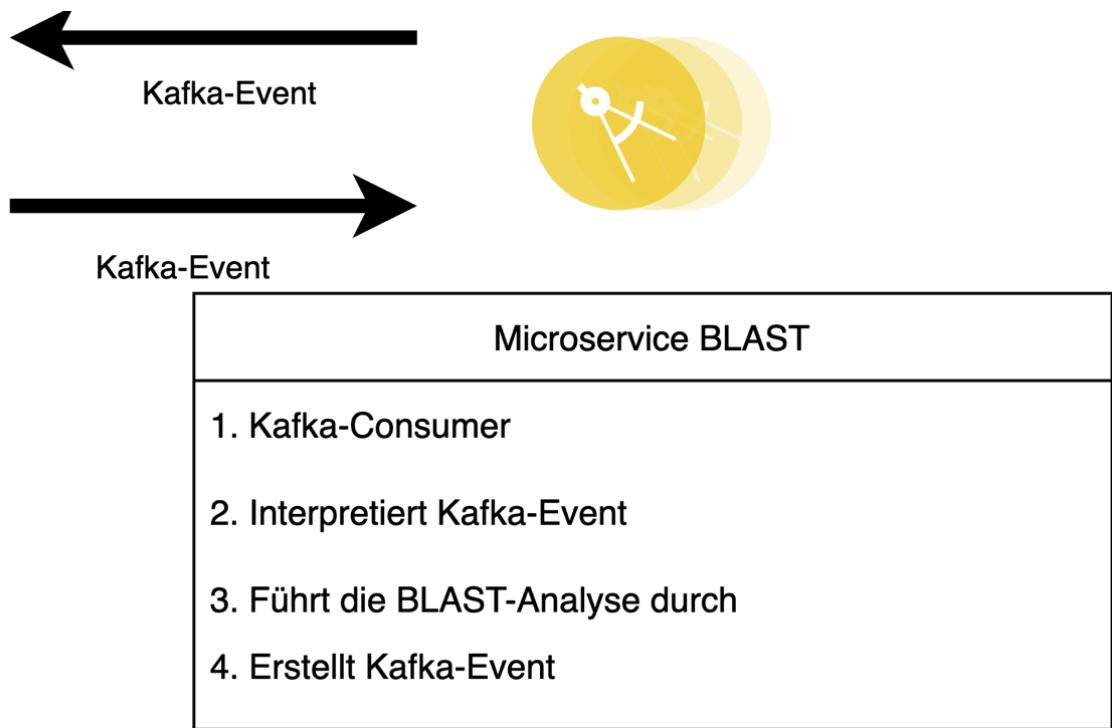


Abbildung 18: Architekturdiagramm BLAST Wrapper (M5)

M5: Architekturdiagramm BLAST Wrapper (FF1, löst VH5)

3.1.4. ASaiM

Für die Integration von ASaiM in die Event-Driven-Architecture wird nur ein Microservice benötigt. Dieser interpretiert die eingehenden Kafka Events für die Analyse mit ASaiM und startet die Analyse. Für die Kommunikation mit ASaiM wird die BioBlend-API des Galaxy Frameworks genutzt. Ist die Analyse abgeschlossen, wird dies durch ein erstelltes Event der Event-Driven-Architecture mitgeteilt. In der Abbildung 19 ist die Integration von ASaiM dargestellt.

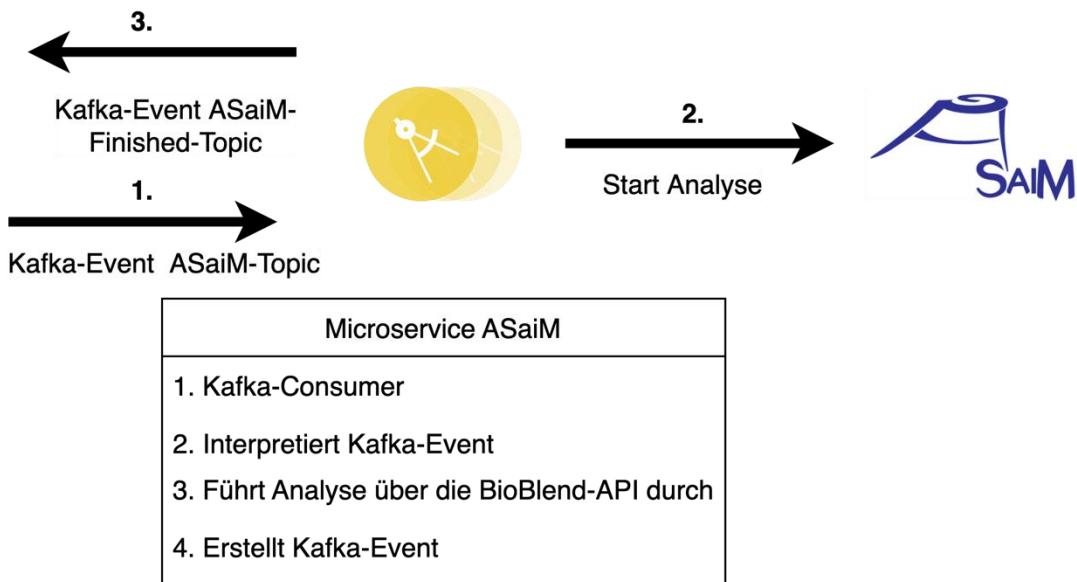


Abbildung 19: Architekturdiagramm ASaiM Wrapper (M6)

M6: Architekturdiagramm ASaiM Wrapper (FF1, löst VH7)

3.1.5. Zusammenfassung der Architektur der Analyse Tools

In dem Kapitel 3.1 wurde aufgezeigt, wie die Integration der Analysetools in die Event-Driven-Architecture vorgenommen werden kann. Dabei wurde festgestellt, dass für eine Integration dieser Tools ein Microservice implementiert werden muss, welcher Events entgegennehmen, interpretieren und erstellen kann. Die Analysen in Apache Spark, ASaIM, BLAST und QIIME2 werden lokal innerhalb des Clusters ausgeführt. Die notwendigen Microservices für die Integration in die Event-Driven-Architecture wurden konzeptionell vorgestellt. Die Konzepte zur Integration der vier Analysetools werden durch folgende vier Modelle beschrieben:

M3: Architekturdiagramm Apache Spark Integration (FF1, löst VH1)

M4: Architekturdiagramm QIIME2 Wrapper (FF1, löst VH3)

M5: Architekturdiagramm BLAST Wrapper (FF1, löst VH5)

M6: Architekturdiagramm ASaIM Wrapper (FF1, löst VH7)

3.2. Integration des Speichers

Der Storage-Cluster ermöglicht es, dass unabhängig davon, auf welchem Kubernetes-Knoten die Analyse durchgeführt wird, die Daten für das Analysetool verfügbar sind. Weiterhin muss aber auch der Benutzer des Systems die Daten in das Cluster hochladen können und der Report des Workflows dem Anwender als Download zur Verfügung gestellt werden. Wird eine Analyse gestartet, müssen auch die Softwarekomponenten auf die Speicher lesend für die Rohdaten und schreibend für die Ergebnisse auf den Speicher zugreifen. In den beiden nachfolgenden Unterkapiteln werden die Use-Cases und die Architektur erläutert. Diese beziehen sich auf die FF2 und die VH9 und 10.

3.2.1. Use-Cases

Für die Analyse Workflows gibt es folgende Use-Cases zur Interaktion mit dem Speicher des IT-Systems:

Der Benutzer kann direkt auf das Filesystem zugreifen, um z. B. mittels SCP (Secure Copy) Daten in den Speicher zu schreiben.

Der Benutzer hat die Möglichkeit, über das Frontend Daten in den Speicher zu laden.

Das Frontend kann auf die verfügbaren Daten zugreifen.

Der Anwender kann einen erstellten Report am Ende eines Workflows downloaden.

Der erste Use-Case ist für Benutzer wichtig, welche auf das verteilte System nur durch das Frontend zugreifen und nicht mit größeren Datenmengen arbeiten.

Der zweite Use-Case ist einerseits für Benutzer mit großen Datenmengen geeignet und andererseits, um in Zukunft auch automatisiert Daten in das Storage-Cluster zu schreiben. Gerade bei größeren Datenmengen kann ein Hochladen über ein Frontend zeitintensiv sein.

Der dritte Use-Case ermöglicht dem Frontend, die verfügbaren Daten dem Anwender zu präsentieren, wonach er diese beim Start eines Workflows diese auswählen kann.

Der vierte Use-Case bezieht sich auf den erfolgreichen Durchlauf eines Workflows. Dort soll der Benutzer den Report über das Frontend als Download zur Verfügung gestellt bekommen.

Die Use-Cases eins, drei und vier werden durch einen Storage Microservice umgesetzt, welcher im Unterkapitel 3.2.3 erläutert wird.

3.2.2. CEPH

Das CEPH-Cluster wird wie Kubernetes auf den vorhandenen Knoten innerhalb des Clusters deployed und der abrufbare Speicherplatz hängt maßgeblich mit dem verfügbaren Speicher auf den Nodes zusammen. In der Abbildung 20 ist die Architektur des CEPH Storages abgebildet. Auf Volumes innerhalb des Storage Cluster kann über die S3- und der Container-Storage-Interface-Schnittstelle zugegriffen werden. Weiterhin können Daten von außen direkt auf das gemountete Filesystem geschrieben werden.

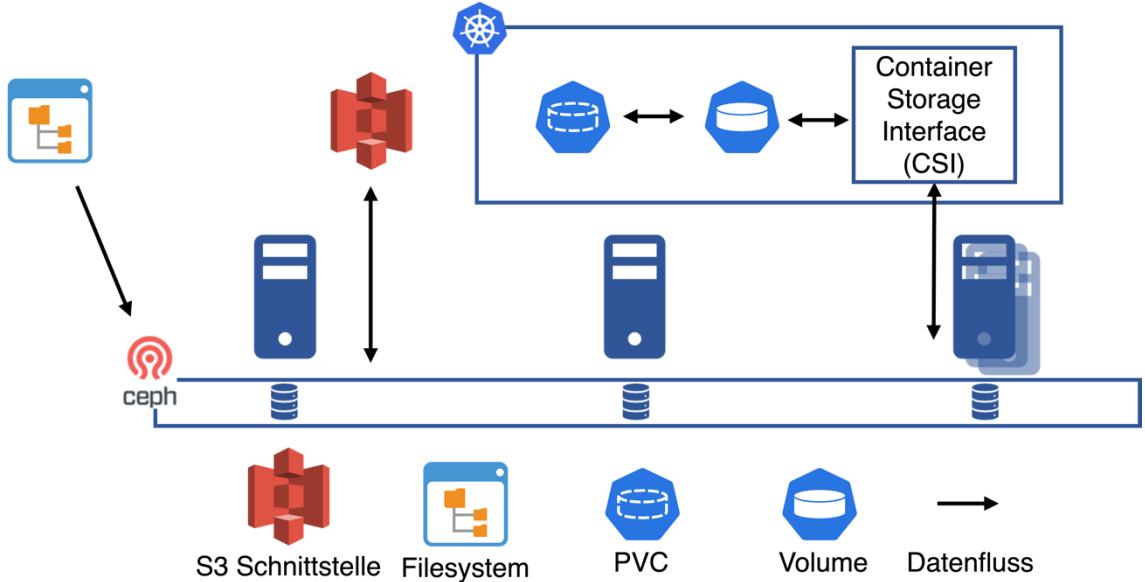


Abbildung 20: Deployment-Plan CEPH Integration (M7)

M7: Deployment-Plan CEPH Integration (FF2, löst VH9)

3.2.3. Architektur des Storage Microservices

Wie im Kapitel 3.2.1 erwähnt, werden innerhalb des Storage Microservices die Use-Cases eins, drei und vier umgesetzt. Der Zugriff auf den Storage Microservice erfolgt über die Benutzungsschnittstelle, welcher im Kapitel 3.3.1 modelliert wird. Um diesen Zugriff zu ermöglichen, wird der Storage Microservice als REST-API die beschriebenen Use-Cases (2 – 4) als Endpunkt der Benutzungsschnittstelle zur Verfügung stellen.

M8: Use-Cases Storage Microservice (FF2, löst VH10)

3.2.4. Zusammenfassung der Architektur des Speichers

In dem Kapitel 3.2 wurde beschrieben, wie ein Benutzer Daten in das verteilte System hochladen und wie die deployten Komponenten auf das Storage-Cluster zugreifen können. Dabei kann ein Hochladen über das Frontend oder direkt auf das Filesystem, z. B. mittels SCP, erfolgen. Die Komponenten können auf die Daten mittels Kubernetes Volumes, welche in Kapitel 3.4.1 beschrieben werden, oder mittels S3 Schnittstelle zugreifen. Die Benutzungsschnittstelle interagiert über einen Storage Microservice mit der Speicherlösung, welcher die beschrieben Use-Cases des Speichers 2-4 bereitstellt. Aus dieser Modellierung entstanden folgende zwei Modelle:

M7: Architekturdiagramm CEPH Integration (FF2, löst VH9)

M8: Use-Cases Storage Microservice (FF2, löst VH10)

3.3. Modellierung von Softwarelösungen für eine Benutzungsschnittstelle und Kommunikation der Komponenten des IT-Systems

Damit Analysen innerhalb des verteilten Systems durchgeführt werden können, wird eine Benutzungsschnittstelle (Frontend) benötigt. Weiterhin wird Kafka als Event-Queue, Airflow als Event-Orchestrator, und ein Reporting Microservice für die Kommunikation von der Benutzungsschnittstelle und zwischen den Komponenten innerhalb des verteilten Systems benötigt. Diese Bausteine werden in den nachfolgenden Unterkapiteln erörtert und beziehen sich auf die FF3 sowie der VH 11, 12, 13, 14 und 15.

3.3.1. Benutzungsschnittstelle

Die Benutzungsschnittstelle (Frontend) ist die zentrale Schnittstelle eines Forschers mit dem gesamten System. Hier können die Dateien hochgeladen, Analysen gestartet und Reports angezeigt werden. Diese soll nur die notwendigen Funktionen zum Benutzen des verteilten Systems beinhalten, um etwaige Fehlbedienungen zu vermeiden. In der Abbildung 21 ist das Ablaufdiagramm der Benutzungsschnittstelle dargestellt.

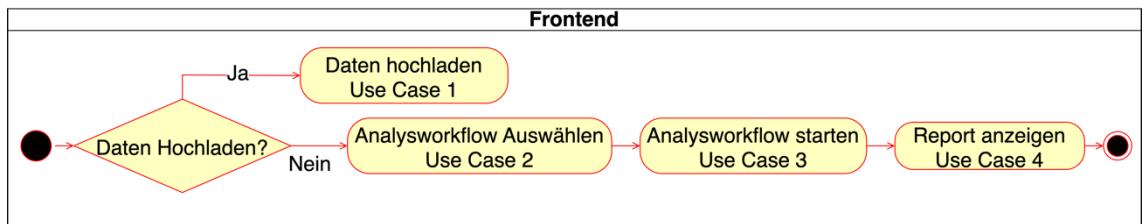


Abbildung 21: Aktivitätsdiagramm Benutzungsschnittstelle (M9)

M9: Aktivitätsdiagramm Benutzungsschnittstelle (FF3, löst VH11)

Der Benutzer muss im ersten Schritt Daten über das Frontend hochladen oder bereits vorhandene Daten auswählen. Im nächsten Schritt hat der Benutzer die Auswahl der vorhandenen Workflows. Dort muss der passende Workflow für den Analyse Use-Case ausgewählt und gestartet werden. Während der Analyse wird dem Benutzer der Status des Workflows angezeigt. Im letzten Schritt bekommt der Benutzer den Report als Download zur Verfügung gestellt. Das Frontend kommuniziert einerseits mit Apache Airflow für das Starten von Workflows und andererseits mit einer REST API (Microservice Storage), um z. B. hochgeladene Daten in den Speicher zu schreiben.

3.3.2. Event Driven Architecture

Die Event Driven Architecture ist der zentrale Baustein des verteilten Systems. Hier findet die lose Kopplung der einzelnen Komponenten statt. Apache Kafka ist hierbei die Event-Queue, auf die alle Komponenten der Event-Driven Architecture zugreifen, um so Analysen zu starten, den Status mitzuteilen oder den Report zu erstellen. Lediglich mit Kafka lassen sich jedoch keine Workflows definieren. In diesem Zusammenhang ist die Verschaltung von mehreren Events gemeint. Ein Analyseworkflow besteht immer aus mindestens zwei Schritten:

Der Analyse selbst

Dem Erstellen des Reports

Dabei kann auch der Fall auftreten, dass innerhalb eines Analyseworkflows mehrere Analysen mit dem gleichen Datensatz durchgeführt und diese am Ende in einem gemeinsamen Report zusammengefasst werden. Zwei mögliche Beispielabläufe sind in der Abbildung 22: dargestellt.

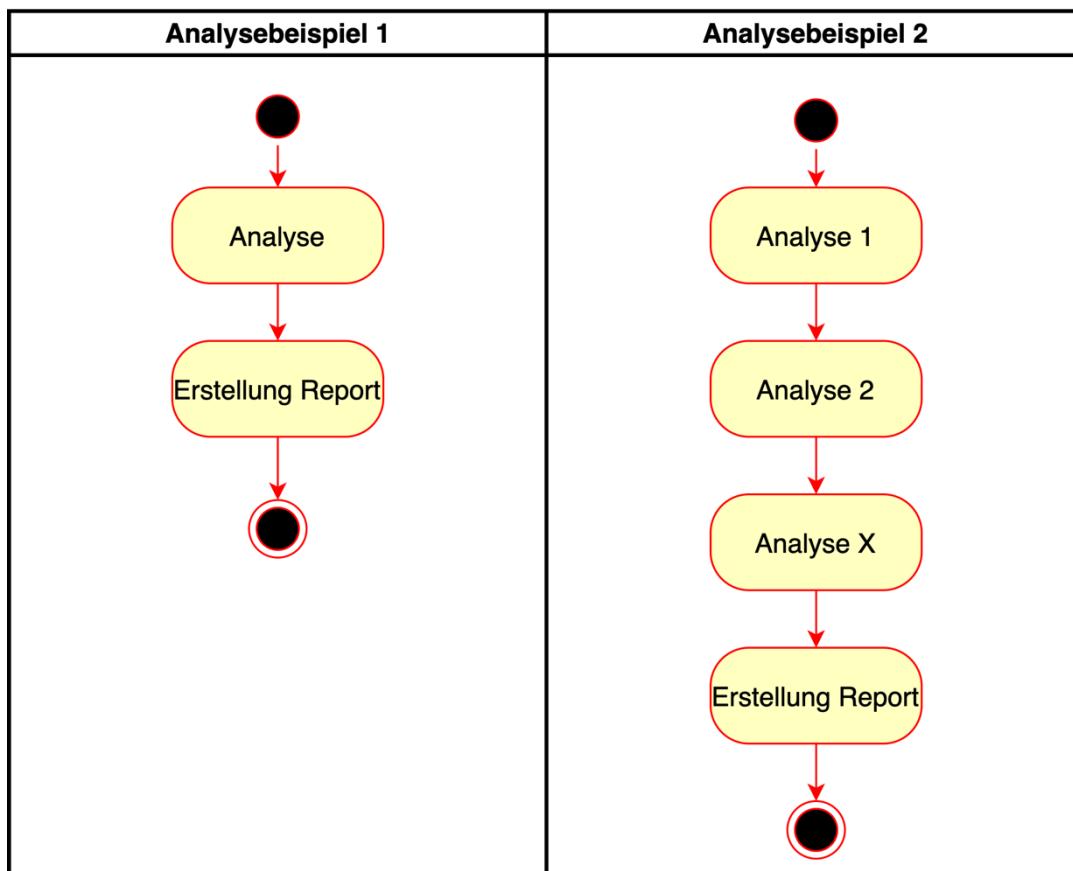


Abbildung 22: Aktivitätsdiagramm Beispielworkflows (M10)

M10: Aktivitätsdiagramm Beispielworkflows (FF3, löst VH13)

Damit diese Abhängigkeiten nicht innerhalb des Frontends implementiert werden müssen und somit die lose Kopplung auch für das Frontend gewahrt werden kann, wird die Orchestrierung von Events und das Erstellen von Workflows in Apache Airflow ausgelagert. Innerhalb von Airflow können so vordefinierte Workflows erstellt werden, die z. B. hintereinander oder parallele Analysen mit einem Datensatz durchführen und am Ende einen Report für den Benutzer erstellen.

Neben der Event-Queue und der Orchestrierung haben die eigentlichen Events ebenfalls eine große Bedeutung. Durch diese müssen alle notwendigen Parameter, welche z. B. für die Analyse, Erstellung eines Reports und Statusmitteilung übertragen werden. Eine beispielhafte Darstellung ist in Abbildung 23 zu finden.

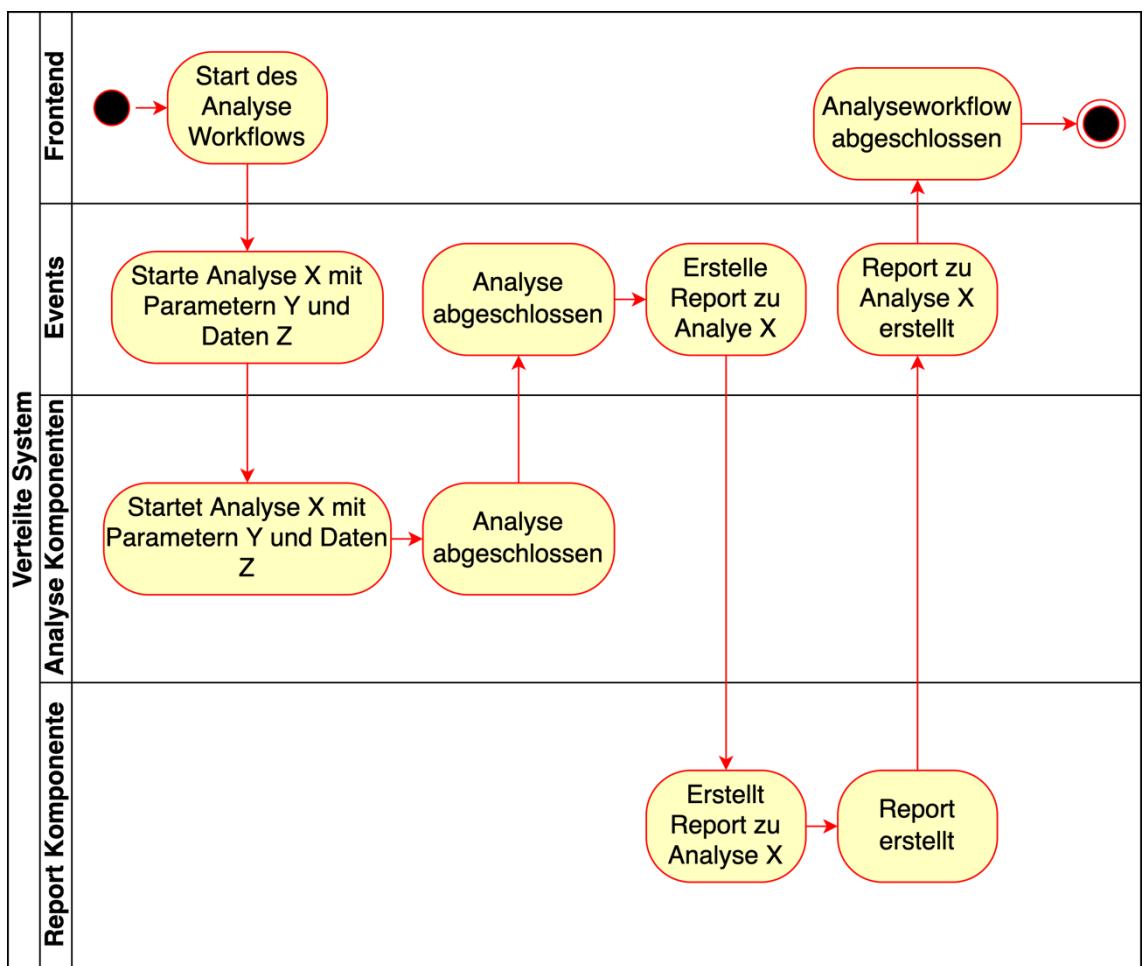


Abbildung 23: Aktivitätsdiagramm Events (M11)

M11: Aktivitätsdiagramm Events (FF3, löst VH14)

3.3.3. Reporting

Am Ende eines Analyse Workflows wird ein Report als Zusammenfassung erstellt. In der Abbildung 24 ist das Aktivitätsdiagramm des Reporting Microservice dargestellt. Die Erstellung des Reports wird ebenfalls über ein Event gestartet und dem Benutzer am Ende als Download zur Verfügung gestellt.

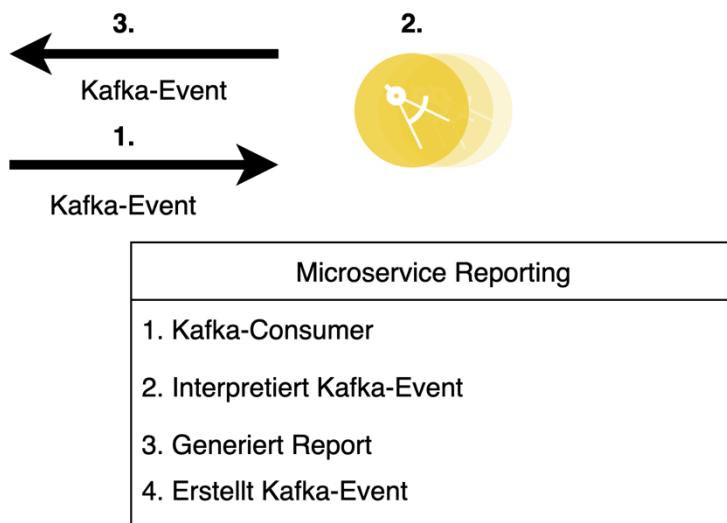


Abbildung 24: Aktivitätsdiagramm Reporting Microservice (M12)

M12: Aktivitätsdiagramm Reporting Microservices (FF3, löst VH15)

3.3.4. Zusammenfassung von der Modellierung einer Softwarelösung für eine Benutzungsschnittstelle des IT-Systems

In den vorangegangenen Unterkapiteln wurde der grundlegende Ablauf der Benutzungsschnittstelle sowie deren Interaktionen mit der REST API sowie der Event-Driven-Architecture vorgestellt. Weiterhin wurde die Event-Queue mit Kafka sowie die Notwendigkeit der Event-Orchestrierung mit Apache Airflow konzeptionell beschrieben und die Erstellung des Reports am Ende eines Analyse Workflows erläutert. Aus diesem Unterkapitel entstanden folgende vier Modelle:

M9: Aktivitätsdiagramm Benutzungsschnittstelle (FF3, löst VH11)

M10: Aktivitätsdiagramm Beispielworkflows (FF3, löst VH13)

M11: Aktivitätsdiagramm Events (FF3, löst VH14)

M12: Aktivitätsdiagramm Reporting Microservices (FF3, löst VH15)

3.4. Erstellung des Deployment-Plans für die gesamte Softwarelösung zur Analyse von Genomdaten

Neben den in den vorherigen Unterkapiteln vorgestellten Architekturen werden für ein Deployment unterschiedliche Kubernetes Objekte benötigt. Ein solches Architekturbild mit dem Fokus darauf wird in Kapitel 3.4.1 beschrieben. Weiterhin muss das gesamte Deployment skalierbar sein. In Kapitel 3.4.2 werden mögliche Skalierungsmaßnahmen beschrieben und adressiert die FF4 und VH 14.

3.4.1. Kubernetes Objekte

Damit außerhalb des Kubernetes Clusters auf das Frontend zugegriffen werden kann, wird ein Ingress Controller benötigt, welcher den Datenverkehr von außen an den Frontend Pod weiterleitet. Das Frontend teilt sich in das eigentliche Frontend und einer REST API auf. Der Kubernetes Service vor manchen Komponenten dient dazu, die einzelnen Services mittels eines Namens und nicht der internen IP-Adresse innerhalb des Kubernetes Cluster erreichen zu können und ermöglicht diesen Service bei einer Skalierung das Loadbalancing zwischen mehreren Deployten Komponenten. Dieser wird nur benötigt, wenn mittels eines Netzwerkes darauf zugegriffen wird. Im Falle des Frontends übernimmt diese Funktion der schon beschriebenen Ingress-Controller. Damit die Daten von dem CEPH-Cluster den einzelnen Microservices zur Verfügung stehen, werden Kubernetes Volumes angelegt und diese in die Microservices gemounted. Eine beispielhafte Architektur ist in der Abbildung 25 zu sehen.

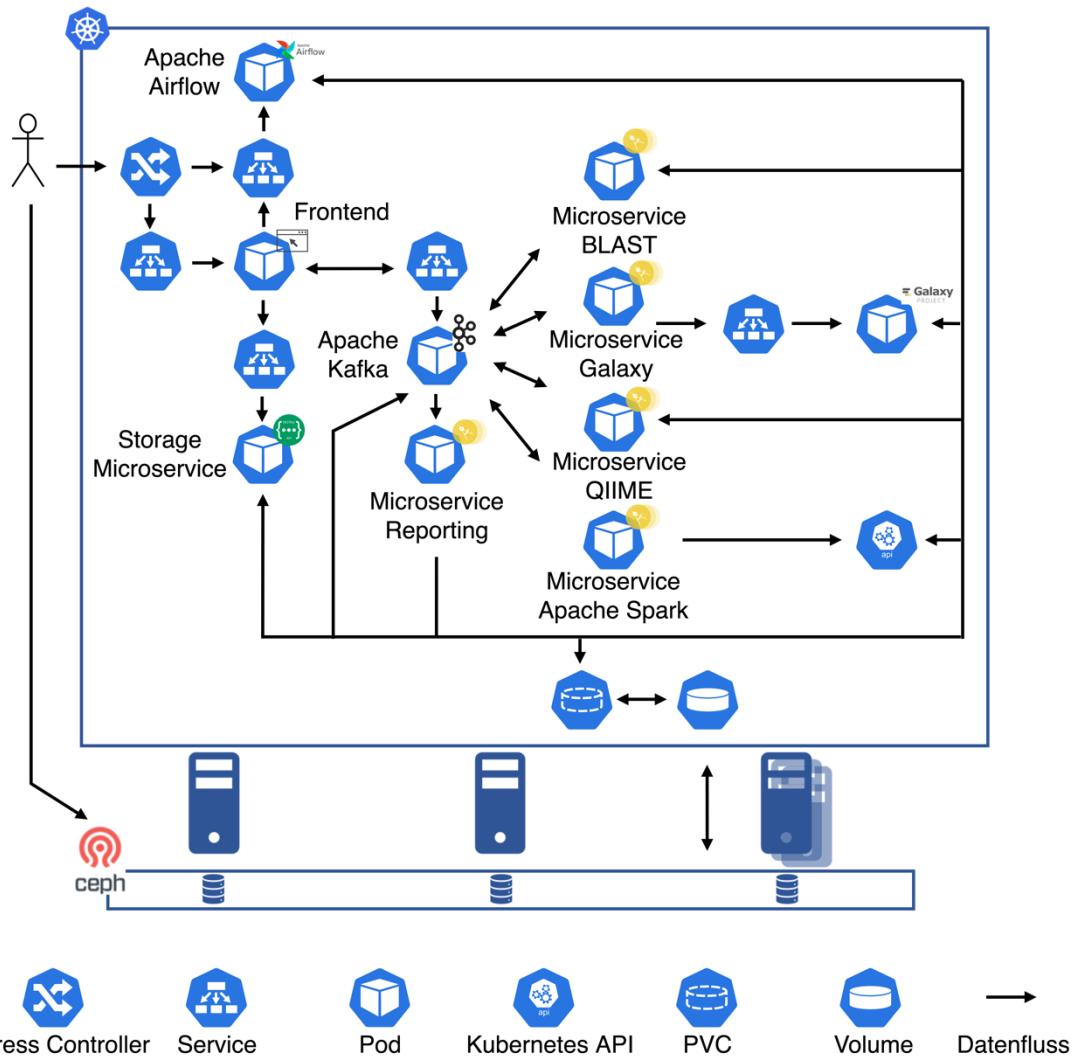


Abbildung 25: Deployment-Plan GEF in Kubernetes (M13)

M13: Deployment-Plan GEF in Kubernetes (FF4, löst VH16)

3.4.2. Skalierung

Es gibt drei unterschiedliche Möglichkeiten, eine Skalierung des vorgestellten verteilten Systems zu ermöglichen:

Horizontale oder vertikale Erweiterung des Clusters.

Vertikale Erweiterung des Speichers.

Skalierung von Komponenten innerhalb des verteilten Systems.

Die horizontale oder vertikale Erweiterung des Clusters ermöglicht es, dem Administrator das ausgelastete Cluster durch performantere Hardware oder durch mehr Compute Nodes zu erweitern.

Ähnlich zu der Skalierung der Hardware lässt sich der Speicher des Clusters auf jedem verfügbaren Node erweitern, um so eine vertikale Skalierung zu schaffen.

Alle Komponenten innerhalb des Clusters lassen sich ebenfalls bei Bedarf manuell oder automatisch skalieren. So wird entweder über den deployten Kubernetes Service / Ingress Controller ein Load-Balancing zwischen den mehrfach deployten Komponenten vorgenommen oder Events werden an mehrere Microservices verteilt.

3.4.3. Zusammenfassung von der Erstellung des Deployment-Plans für die gesamte Softwarelösung zur Analyse von Genomdaten

Die Kubernetes Objekte, welche für ein Deployment der Komponenten notwendig sind, sowie die Möglichkeiten der Skalierung des verteilten Systems wurden in den Unterkapiteln vorgestellt. Als Kubernetes Objekte werden zusätzlich der Ingress-Controller für das Frontend, Kubernetes Services für die Erreichbarkeit der Komponenten innerhalb des Clusters und Kubernetes Volumes als Zugriff auf den Speicher beschrieben. Die Skalierung kann über die Hardware (Compute-Nodes und Speicher) oder über die Anzahl an deployten Komponenten innerhalb des Clusters erfolgen. Der Deployment-Plan ist das Modell 13:

M13: Deployment-Plan GEF in Kubernetes (FF4, löst VH16)

3.5. Zusammenfassung der Modellierung

In diesem Kapitel wurden aufbauend auf Kapitel 2 alle Forschungsziele, welche im Kapitel 1.6 nach X.1/TB vorgestellt wurden, und die Remaining Challenges 1-5 aus Kapitel 2 adressiert. Dabei wurden zu Beginn die grundlegenden Use-Cases des verteilten Systems und die Zielarchitektur aufgezeigt. In den Unterkapiteln wurden einzelne Teile der Zielarchitektur genauer beschrieben. Für das Deployment in ein Kubernetes Cluster wurde ebenfalls eine Zielarchitektur mit Kubernetes Objekten erstellt und Skalierungsmöglichkeiten des verteilten Systems aufgezeigt.

Folgende Modelle sind innerhalb des Kapitels für das Genomic Explore Framework erstellt worden

- M1: Use-Context Diagramm GEF (FF1-4, löst VH11)
- M2: Komponentendiagramm GEF (FF1-4, löst VH12)
- M3: Architekturdiagramm Apache Spark Integration (FF1, löst VH1)
- M4: Architekturdiagramm QIIME2 Wrapper (FF1, löst VH3)
- M5: Architekturdiagramm BLAST Wrapper (FF1, löst VH5)
- M6: Architekturdiagramm ASaiM Wrapper (FF1, löst VH7)
- M7: Architekturdiagramm CEPH Integration (FF2, löst VH9)
- M8: Use-Cases Storage Microservice (FF2, löst VH10)
- M9: Aktivitätsdiagramm Benutzungsschnittstelle (FF3, löst VH11)
- M10: Aktivitätsdiagramm Beispielworkflows (FF3, löst VH13)
- M11: Aktivitätsdiagramm Events (FF3, löst VH14)
- M12: Aktivitätsdiagramm Reporting Microservices (FF3, löst VH15)
- M13: Deployment-Plan GEF in Kubernetes (FF4, löst VH16)

Im nächsten Kapitel wird ein Prototyp aufgebaut, welcher die vorgestellten Techniken implementiert. Dabei reduziert sich der Prototyp auf die Implementierung des Frontends auf Basis des Aktivitätsdiagramm (M9), des Storage Microservices (M8), zwei Beispielworkflows innerhalb von Apache Airflow (M10), der Events (M11), des Reporting Microservices (M12) und der Integration des BLAST Wrappers (M5). Des Weiteren wird ein Python Script implementiert, um den GC-Content einer Gensequenz zu berechnen. Dies würde in der GEF innerhalb eines Spark-Clusters berechnet werden, wird aber aufgrund der Komplexität innerhalb eines Wrapper Microservices als Beispiel implementiert.

Deployed wird der Prototyp auf einem Single-Host innerhalb von Docker und als Filesystem wird das Host-Filesystem genutzt. Der Prototyp hat das Ziel, die Architektur des verteilten Systems und die Kommunikation der einzelnen Komponenten mit Events zu implementieren.

4. Implementierung

Aufbauend auf dem Kapitel 3, der Modellierung und des resultierenden GEF, erfolgt in diesem Kapitel die Implementierung eines Prototyps. Dabei stellt die Implementierung einen weiteren Schritt der vorgestellten Nunamaker Methodik von Kapitel 1.4 dar. Der Prototyp enthält Bestandteile, welche in der Modellierung erstellt wurden. Alle Forschungsziele, welche im Kapitel 1.6 nach X.1/I sortiert wurden, haben eigene Unterkapitel.

Der Name des Prototyps ist GenomicInsights und ist eine Teilimplementierung des Genomic Explore Frameworks. Die Bestandteile des Prototyps sind mit P1-Pn durchnummeriert.

GenomicInsights beinhaltet zwei Analysemethoden. Einerseits ist BLAST (M5, P1) integriert und andererseits ein Microservice zur Generierung des GC-Content (P2) aus einer Gensequenz. Diese Analysemethoden sollen die Funktionsweise des GEF belegen.

Weitere Bestandteile und Konfigurationen von GenomicInsights sind die Speicherkonfiguration (P3), Storage Microservice (M8, P4), Benutzungsschnittstelle (M9, P5), Apache Kafka (M10 und M11, P6), Apache Airflow (M10 und M11, P7), Reporting Microservice (M12, P8) und das Deployment (P9). Um die Komplexität von GenomicInsights zu minimieren, ist die in Kapitel 3 verteilte Architektur für die Ausführung auf einem einzelnen Knoten implementiert.

Aus diesem Grund werden die Container innerhalb einer Docker-Umgebung gestartet und als Storage kein verteilter Storage, sondern das lokale Filesystem genutzt.

Die Microservices werden in Python und das Vue-Frontend mit HTML, CSS und Javascript implementiert.

In der Abbildung 26 sind die Komponenten von GenomicInsights zu sehen.

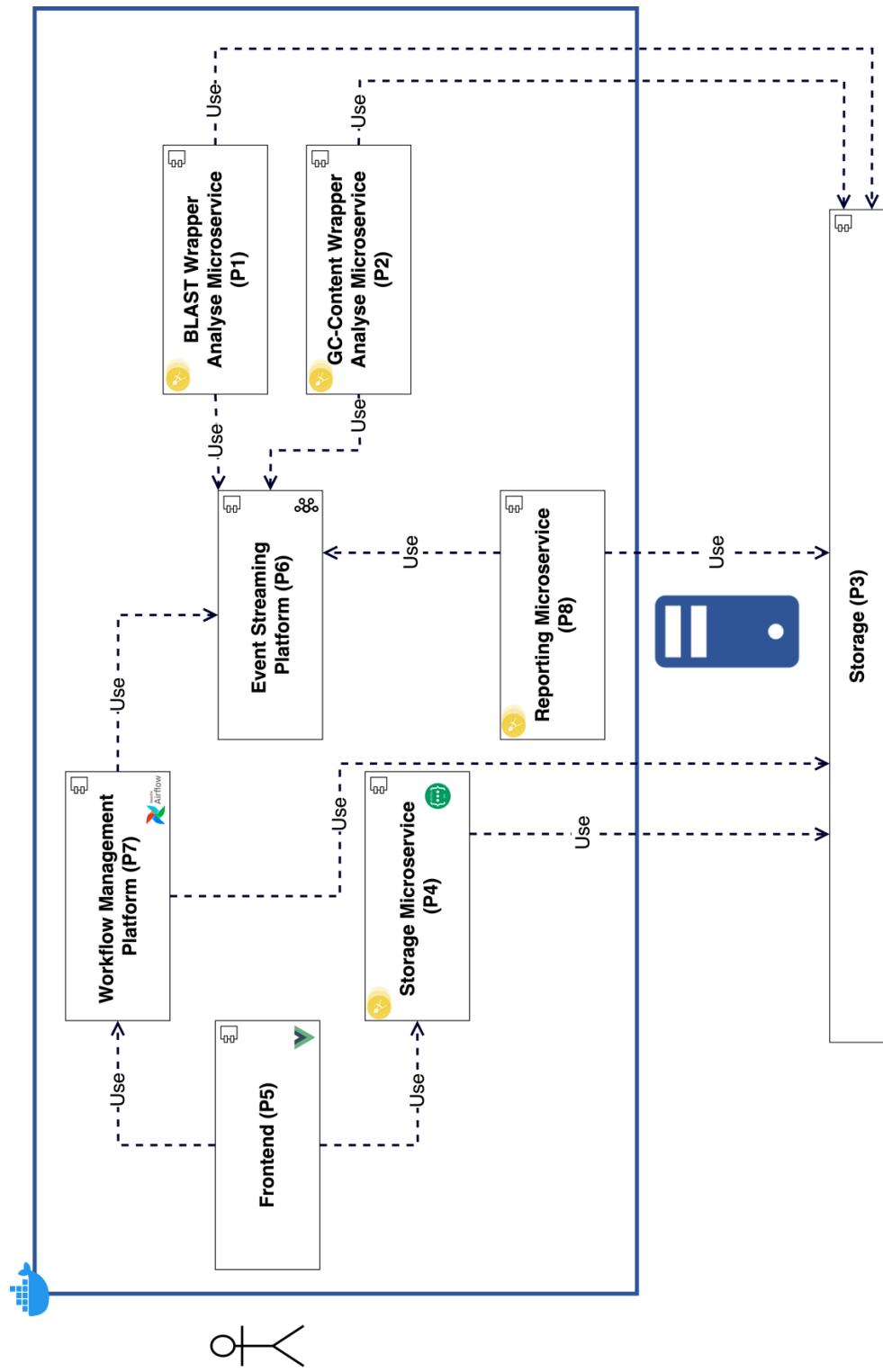


Abbildung 26: Komponentendiagramm von Genomic Insights

4.1. Implementierung der Analyse Software im Rahmen eines POCs

Die Analysebestandteile von GenomicInsights sind der im Kapitel 3.1 beschriebene BLAST Microservice und GC-Content Microservice. Letzterer ermöglicht die Berechnung des prozentualen Anteils von Guanin und Cytosin innerhalb einer DNA oder RNA. Die prozentuale Berechnung des GC-Content Microservices könnte innerhalb eines Spark Clusters erfolgen. In diesem Fall würden unterschiedliche Worker einen Ausschnitt der DNA- oder RNA-Sequenz erhalten und für diesen Ausschnitt den Wert berechnen. Für die Implementierung im Rahmen von GenomicInsights wird auf ein Spark-Cluster verzichtet und diese Berechnungen ebenfalls in einem Microservice durchgeführt.

In den folgenden Unterkapiteln wird die Implementierung der beiden Microservices beschrieben. Beide Microservices kommunizieren mittels Events mit dem Kafka Broker und greifen auf den RAW sowie RESULT Storage zu. Die Architektur des Speichers sowie der Inhalt der Events werden in Kapitel 2 bzw. Kapitel 3 genauer erörtert.

4.1.1. BLAST Microservice

Für BLAST gibt es auf Dockerhub ein Dockerimage, auf dem der Microservice von BLAST aufbaut. Die Standalone Version von BLAST beinhaltet keine API, sondern nur eine CLI (Command Line Interface) zum Starten von Analysen. Aus diesem Grund muss der Analyseaufruf direkt in dem Microservice, welcher die Events als Kafka-Consumer entgegennimmt und als Kafka-Producer erstellt, erfolgen. Die Kafka Konfiguration der Topics wird genauer im Kapitel 4.3 beschrieben. Das bestehende Image von BLAST wird um die benötigte Funktionalität erweitert. Darunter zählt z. B. die Installation von dem Python Modul „Kafka-Python“ für Kafka Funktionalitäten. Innerhalb eines Python Scripts werden die BLAST Befehle zur Analyse ausgeführt. Wird der Container gestartet, startet das Python Script und wartet dauerhaft auf Events in der „blast_analyzer“ Topic des Kafka-Brokers. Kommt dort ein Event an, wird ein neuer Thread erstellt und die Daten des Bodys übergeben. Dort enthalten ist der Filename der Rohdatei und die eindeutige Workflow ID von Apache Kafka. Passend zum Analyse Use-Case müssen Datenbanken zum Vergleich der Genomsequenz vorgehalten werden. Diese werden auf dem Filesystem unter dem Speichernamen „BLAST_DATABASES“ gespeichert. Da diese von allen BLAST Microservices benötigt werden und je nach Use-Case mehrere Gigabyte groß sein können, werden diese nicht direkt in den Container integriert, sondern von außen injiziert. Dadurch wird die Containergröße optimiert und verhindert, dass Daten redundant auf dem System gespeichert werden. Innerhalb des Containers sind die zwei Ordner für die Roh- und Resultdaten verfügbar. Weiterhin werden die benötigten Kafka Libraries für das auszuführende Python Script mit Hilfe von PIP installiert. Innerhalb des gestarteten Threads wird das BLAST-Kommando mit Hilfe des Moduls OS.SYSTEMS ausgeführt.

Dort wird mit „blastn“ die Identifikation von Nukleotidsequenzen mit der Rohdatei der zu durchsuchenden Datenbank und der Name der Ergebnisdatei dynamisch angegeben. Dieser Use-Case wird in Kapitel 5.1 genauer beschrieben. Für die Implementierung anderer Use-Cases müssen weitere Kommandos dem Microservice hinzugefügt werden.

Ist die Analyse fertig, wird durch das Python-Script ein „Success“-Event in der „blast_report“ Topic abgeschickt. Der Zusammenhang von diesen Events wird in der Workflowbeschreibung unter Apache Airflow genauer beschrieben.

In der Abbildung 27 ist das Aktivitätsdiagramm des BLAST Microservices abgebildet.

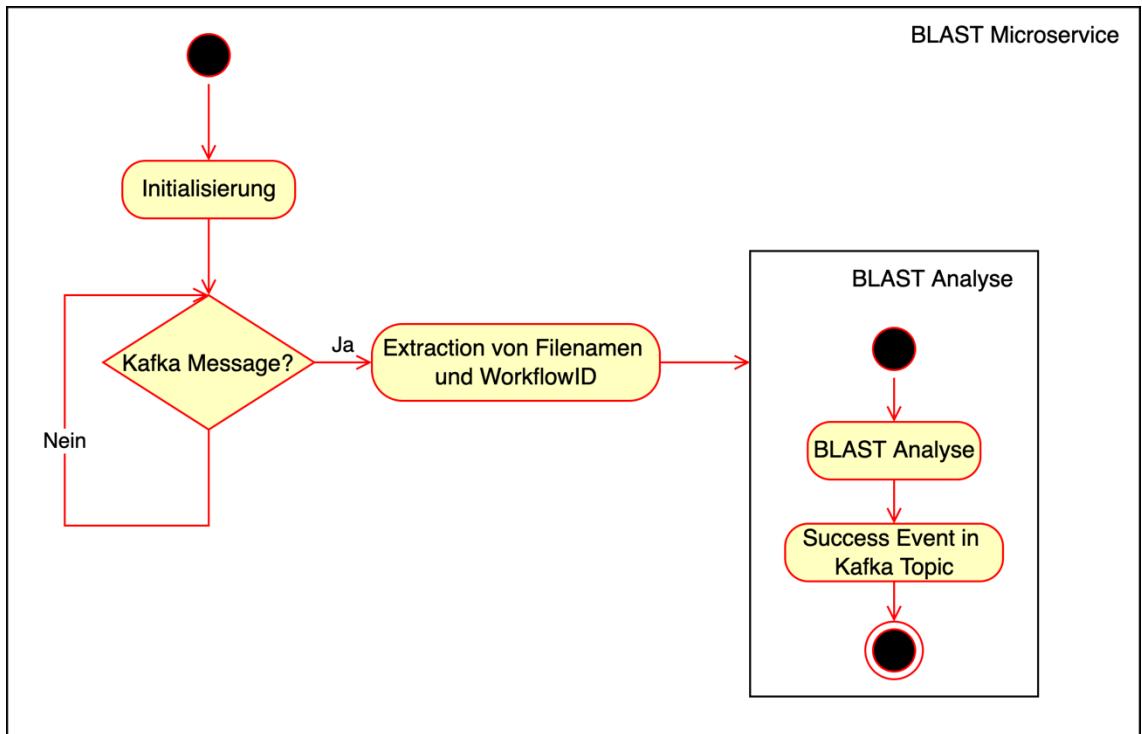


Abbildung 27: Aktivitätsdiagramm BLAST Analyse Microservice (P1)

P1: Wrapper BLAST Analyse Microservice (FF1, VH5, M5)

4.1.2. GC-Content Microservice

Der GC-Content Microservice hat die Aufgabe, innerhalb von GenomicInsights den GC-Wert aus einer oder mehreren Sequenzen zu errechnen. Der Container baut auf dem Python Docker Image von Dockerhub auf. Das Image wird ebenfalls um die notwendige Funktionalität erweitert, z. B. mit der Installation des Python Moduls „Kafka-Python“. Ähnlich wie bei dem vorgestellten BLAST Microservice führt auch dieser die Berechnung innerhalb eines neuen Threads durch und die grundlegende Logik ist bei beiden Microservices gleich. Der GC-Content Microservice schreibt sich als Kafka-Consumer in die „gc_analyzer“ Topic ein. Kommt dort ein Event an, wird mit Hilfe der BioPython Library die FASTQ Datei eingelesen, der GC-Content Wert berechnet und die Ergebnisse mit einer Namenskonvention im Ergebnisordner gespeichert. Die Namenskonvention wird in Kapitel 4.2 genauer beschrieben. Ist die Analyse fertig, wird durch das Python-Script in der „gc_report“ Topic ein „Success“-Event abgeschickt. Die Events werden in der Workflowbeschreibung unter Apache Airflow genauer beschrieben. In der Abbildung 28 ist das Ablaufdiagramm des GC-Content Microservices abgebildet.

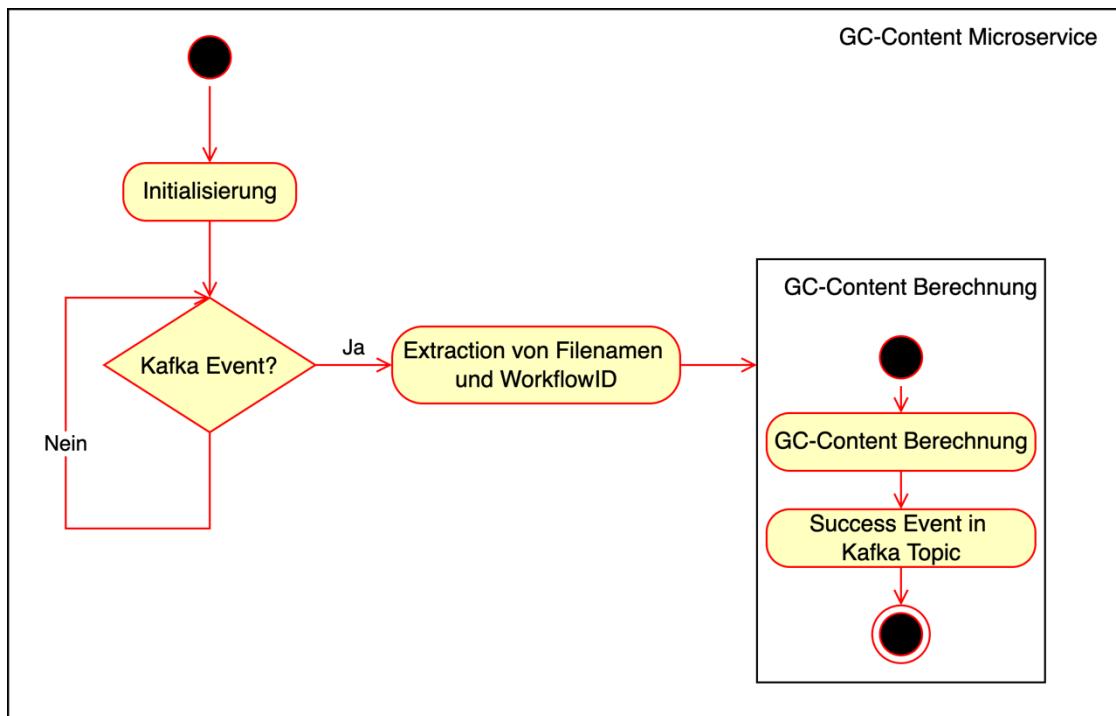


Abbildung 28: Aktivitätsdiagramm GC-Content Analyse Microservice (P2)

P2: Wrapper GC-Content Analyse Microservice (FF1)

4.1.3. Zusammenfassung der Implementierung der Open-Source Big-Data Analyse Software im Rahmen eines POCs

Im Kapitel 4.1 ist die Implementierung der Microservices zur Analyse der Genomdaten für GenomicInsights beschrieben. Beide Microservices sind in die Event-Driven Architecture integriert und können mit Hilfe von Docker auf dem System gestartet werden. Folgende zwei Prototypen wurde während der Implementierung der Analysetools entwickelt:

P1: Wrapper BLAST Analyse Microservice (FF1, VH5, M5)

P2: Wrapper GC-Content Analyse Microservice (FF1)

4.2. Implementierung der Speicherlösung im Rahmen eines POCs

Wie in der Einleitung beschrieben, ist für GenomicInsights kein verteiltes Dateisystem zur Speicherung der Daten notwendig. Aufgrund des Single Hosts wird das Filesystem des Hosts verwendet. Weiterhin ist in dem GEF ein Storage Microservice vorhanden, welcher die Daten, die ein Benutzer über das Frontend hochlädt, auf dem Dateisystem speichert und Reports für den Benutzer abrufbar macht. Die Konfiguration des Dateisystems und des Storage Microservices werden in den folgenden Unterkapiteln beschrieben.

4.2.1. Konfiguration des Dateisystems

Da auf dem Single Host das Dateisystem zur Speicherung der Daten verwendet wird, ist keine gesonderte Konfiguration notwendig. Auf dem Host müssen folgende Ordner angelegt werden (P3):

RAW: beinhaltet die Rohdaten

RESULTS: beinhaltet die Ergebnisdaten der Analysen

REPORT: beinhaltet die Reportdaten der Workflows

BLAST_DATABASES: beinhaltet die BLAST Datenbanken

Die Bereitstellung der Ordner in den Containern wird von Docker bei Start des Containers vorgenommen. Mehr Details dazu sind in Kapitel 4.4 zu finden.

Für das Speichern der Daten kommt eine Namenskonvention zum Einsatz:

Rohdaten = Dateiname

Resultdaten = result_Dateiname_WorkflowID

Reportdaten = report_Dateiname_WorkflowID

Die WorkflowID ist eine eindeutige ID innerhalb von Apache Airflow, um den ausgeführten Workflow zu identifizieren.

P3: Konfiguration des Speichers (FF2)

4.2.2. Storage Microservice

Der Storage Microservice ist ein Application Interface (API) und ist mit FastAPI implementiert worden. Dieser wird benötigt, damit der Benutzer über das Frontend Rohdaten hochladen, diese innerhalb eines Workflows ausgewählen und der Report am Ende eines Workflows dem Benutzer zurückgegeben werden kann. FastAPI wird wie die anderen Microservices in Python implementiert und in einem Python Docker Image aufgebaut. Diese drei Use-Cases sind als Endpoint in der API definiert. Der Aufruf erfolgt über eine HTTP GET oder POST-Methode (REST-Schnittstelle):

/upload: POST-Methode, welche im HTTP-Body die Rohdatei des Benutzers enthält. Diese wird in dem RAW-Ordner gespeichert.

/getFiles: GET-Methode, welche die vorhandenen Dateinamen im RAW-Folder im JSON-Format zurückgibt.

/downloadReport: GET-Methode, welche innerhalb der URL den Dateinamen der Reportdatei enthält. Diese Datei wird im Reportordner gesucht und als BLOB zurückgegeben.

In der Abbildung 29 ist das Ablaufdiagramm des Storage Microservices abgebildet.

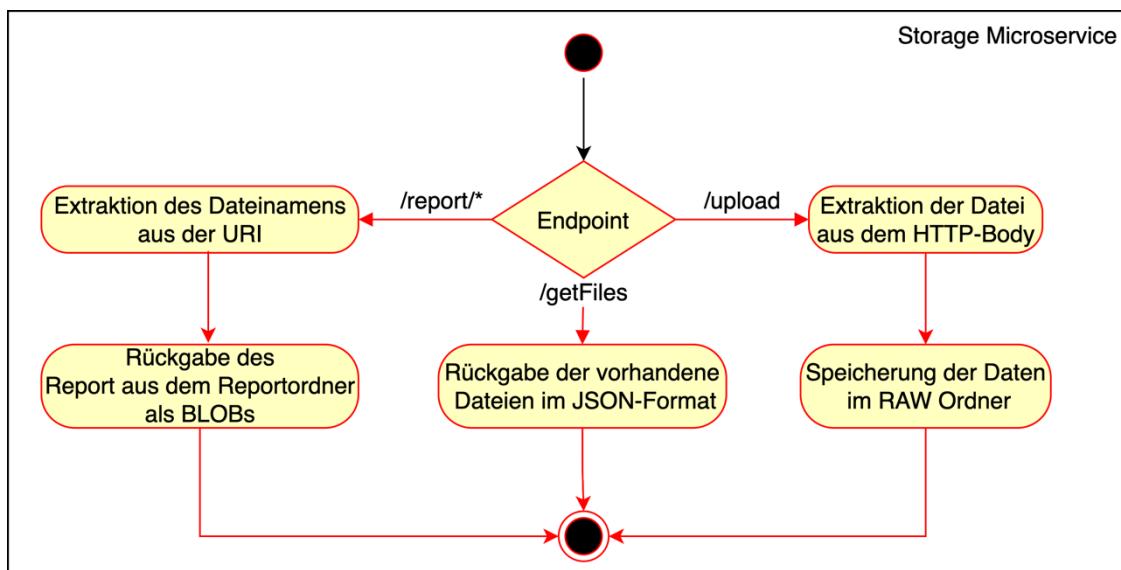


Abbildung 29: Aktivitätsdiagramm Storage Microservice (P4)

P4: Storage Microservice (FF2, VH10, M8)

4.2.3. Zusammenfassung der Implementierung der Speicherlösung im Rahmen eines POCs

Im Kapitel 4.2 wurde die Konfiguration des Speichers sowie des Microservices, welcher die API zur Interaktion mit dem Speicher bereitstellt, beschrieben. Aufgrund der Implementierung auf einem Single-Node, werden die Daten direkt auf dem Dateisystem des Hosts gespeichert. Dort werden die beschriebenen Ordner erstellt. Der Microservice stellt eine API mit drei Endpunkten zur Verfügung. Dieser ermöglicht es, Daten auf das Dateisystem zu schreiben und abzufragen, sowie den Report dem Benutzer zur Verfügung zu stellen. Die Konfiguration des Speichers ist der P3 und der Storage Microservice ist P4:

P3: Konfiguration des Speichers (FF2)

P4: Storage Microservice (FF2, VH10, M8)

4.3. Implementierung einer Softwarelösung für eine Benutzungsschnittstelle des IT-Systems im Rahmen eines POCs

Aufbauend auf dem GEF werden in dem nächsten Abschnitt die Implementierung des Frontends, der Event-Driven Architecture und des Reporting Microservices vorgestellt. Die Event-Driven Architecture teilt sich in die zwei Kapitel Konfiguration von Apache Kafka sowie der Konfiguration von Apache Airflow auf. Innerhalb des Frontends werden die vier beschrieben Use-Cases des GEF in GenomicInsights berücksichtigt und durch den Reporting Microservice die Möglichkeit gegeben, einen Report am Ende eines Workflows aus den generierten Daten zu erstellen. Die Event-Driven Architektur enthält zwei Workflows, die jeweils den BLAST und GC-Content Microservice beinhalten und am Ende einen Report erstellen. Dafür werden in Apache Airflow beispielhaft zwei Workflows angelegt und in Kafka unterschiedliche Topics zur Kommunikation konfiguriert.

4.3.1. Benutzungsschnittstelle

Die Benutzungsschnittstelle (P5) hat aufbauend auf Kapitel 3.3.1 vier unterschiedliche Use-Cases in dem GEF, welche ebenfalls in GenomicInsights umgesetzt sind:

Use-Case 1: Daten hochladen

Use-Case 2: Analyseworkflow auswählen

Use-Case 3: Analyseworkflow starten

Use-Case 4: Report anzeigen / downloaden

Das Frontend ist in Vue.js umgesetzt und wird durch die Frontend Library „ElementPlus“, welche Buttons, Menüpunkte, Icons etc. zur Verfügung stellt, ergänzt.

Use-Case 1 hat in der Benutzungsschnittstelle von GenomicInsights einen eigenen Menüpunkt („Upload Data“). Dahinter hat ein Benutzer einerseits die Möglichkeit mittels „Drag and Drop“ eine Datei hochzuladen und andererseits den Filebrowser zu nutzen. Im Hintergrund wird der Storage Microservice angesprochen und der Endpoint /upload genutzt. Ein Screenshot ist in der Abbildung 30 zu finden.

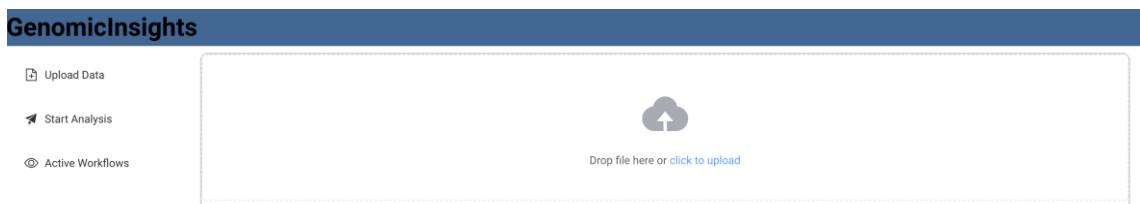


Abbildung 30: Screenshot GenomicInsights Use-Case 1 Daten hochladen

Use-Case 2 und 3 sind innerhalb eines Menüpunktes („Start Analyse“) zusammengefasst. Im oberen Bereich der Benutzungsschnittstelle sind die unterschiedlichen Schritte, um eine Analyse zu starten, abgebildet. Im ersten Schritt muss ein Workflow ausgewählt werden. Hierbei wird beim Laden die Airflow API angesprochen und alle hinterlegten Workflows mit dazugehöriger Beschreibung angezeigt (siehe Abbildung 31). Sind viele Workflows im System hinterlegt, kann das Finden eines spezifischen Workflows Zeit in Anspruch nehmen. Aus diesem Grund ist eine Suchfunktion eingebaut, die es erlaubt, die Einträge zu filtern. Gefiltert wird nach dem Namen des Workflows.

The screenshot shows the 'GenomicInsights' application interface. At the top, there's a blue header bar with the title 'GenomicInsights'. Below it is a navigation bar with links for 'Upload Data', 'Start Analysis', and 'Active Workflows'. The main area is titled 'Select Workflow' and contains a search bar labeled 'Search for Workflows...'. Below the search bar is a table with two columns: 'Name' and 'Description'. The table lists two workflows: 'BLAST_NUKLEOTID' (Description: 'Analyse Nukleotidsequences with BLAST') and 'GC_CONTENT' (Description: 'GC-Content calculation'). A red box highlights the 'Select' column. At the bottom of the workflow selection area is a blue 'Next step' button.

Abbildung 31: Screenshot GenomicInsights Use-Case 2 Workflow Auswählen

Der Benutzer muss einen Workflow auswählen. Klickt dieser trotzdem auf den Button „Next step“, wird dieser durch einen roten Rahmen und einen Text darauf aufmerksam gemacht, einen Workflow auszuwählen (siehe Abbildung 32).

This screenshot is identical to Abbildung 31, but it includes a red error message 'Please select an entry!' displayed prominently above the 'Next step' button. The rest of the interface, including the workflow list and the 'Select' column highlight, remains the same.

Abbildung 32: Screenshot GenomicInsights Use-Case 2 Workflow Auswählen (Fehler)

Im nächsten Schritt werden, nach erfolgreicher Auswahl des Workflows, die verfügbaren Dateien angezeigt. Diese aufgelisteten Dateinamen werden durch den Storage Microservice zurückgegeben (siehe Abbildung 33). Sind viele Dateien im System hinterlegt, kann das Finden einer spezifischen Datei lange dauern. Aus diesem Grund ist eine Suchfunktion eingebaut, die es erlaubt, die Einträge zu filtern. Gefiltert wird nach dem Namen der Datei.

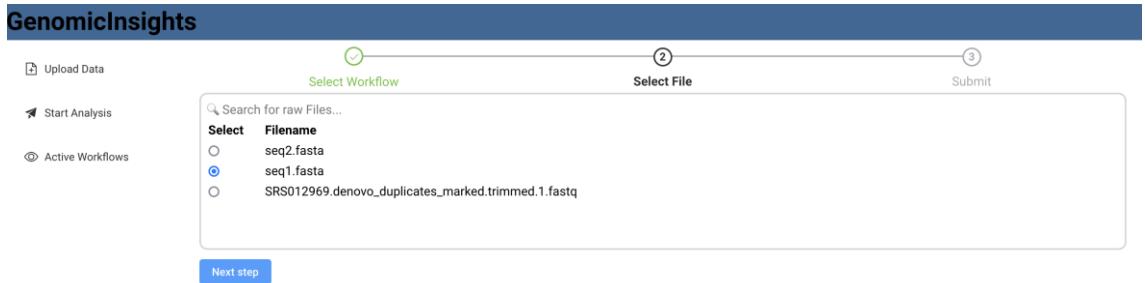


Abbildung 33: Screenshot GenomicInsights Use-Case 2 Datei auswählen

Auch hier muss der Benutzer eine Datei auswählen. Klickt er trotzdem auf den Button „Next step“, wird der Benutzer durch einen roten Rahmen und einem Text darauf aufmerksam gemacht, eine Datei auszuwählen.

Im letzten Schritt bekommt der Benutzer die getätigten Eingaben (Workflow und Dateiname) angezeigt und kann daraufhin auf den Button „Submit“ drücken (siehe Abbildung 34). Im Hintergrund wird über die Airflow API der ausgewählte Workflow gestartet und der entsprechende Dateiname übergeben.

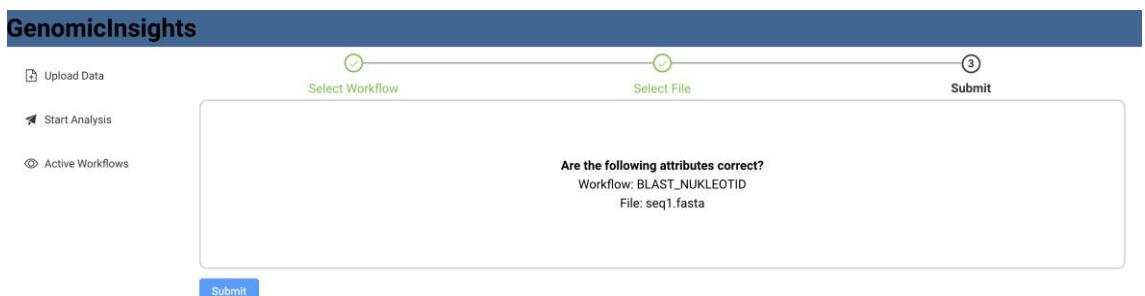


Abbildung 34: Screenshot GenomicInsights Use-Case 2 Datenkorrektheit überprüfen

Es erscheint ein Pop-Up, welcher den Benutzer fragt, ob er in die detaillierte View wechseln möchte, um seinen Workflow zu verfolgen. Klickt dieser auf „Yes“, wird er in eine andere View weitergeleitet. Diese ist im Rahmen von GenomicInsights nur über diesen Schritt erreichbar. Das Pop-Up ist in der Abbildung 35 zu sehen.

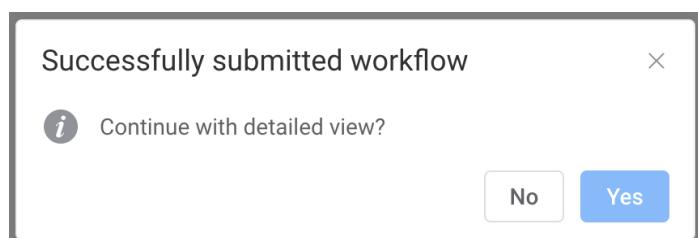


Abbildung 35: Screenshot GenomicInsights Pop-Up Detailed View

In der detaillierten View ist der aktuelle Stand des Workflows (welche Steps sind schon durchlaufen und welcher Step wird gerade bearbeitet) zu sehen. Dies geschieht über eine Abfrage (alle fünf Sekunden) der Airflow API. Das automatische Abfragen kann der Benutzer über einen Schalter unten rechts aktivieren oder deaktivieren. Ist der Workflow erfolgreich bis zum letzten Step durchgelaufen, kann der Benutzer den „Download Report“ Button drücken und erhält den Report als PDF. Im Hintergrund wird der Storage Microservice für den Download des Reports angesprochen (siehe Abbildung 36).

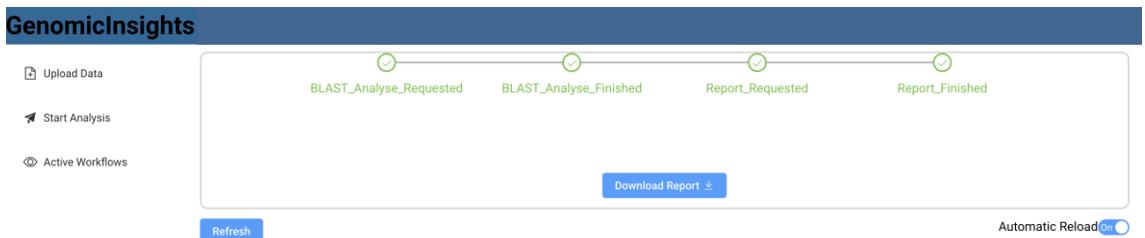


Abbildung 36: Screenshot GenomicInsights Use-Case 4 Workflow Status und Report download

P5: Benutzungsschnittstelle (FF3, VH11, M9)

4.3.2. Apache Kafka

Für das Deployment von Apache Kafka (P6) wird ein vorgebautes Dockerimage verwendet, das von Confluent unter der Apache 2 Lizenz zur Verfügung gestellt wird. Ebenfalls ist ein vordefiniertes docker-compose File zum Start der Apache Kafka Umgebung auf einem Single Host vorhanden [52]. Die Umgebung besteht aus den zwei Containern Apache Zookeeper und Apache Kafka Broker. Zum Zeitpunkt der Erstellung von GenomicInsights wird der Kafka-Broker in der Version 3.2.2 und der Zookeeper Version 3.7.1 verwendet. Der Broker ist unter dem Port 9092 innerhalb des Docker Netzwerkes verfügbar. Für das Benutzen der Kafka Umgebung ist für GenomicInsights keine weitere Konfiguration notwendig. Für GenomicInsights ist ein Speichermapping nicht notwendig. Im Falle eines neuen Deployments ist die Konfiguration gelöscht. In einem produktiven Betrieb müssen die Topics und die darin enthaltenen Events auch nach einem neuen Deployment vorgehalten werden.

Innerhalb des Brokers müssen unterschiedliche Kafka Topics für die Events erstellt werden. GenomicInsights beinhaltet zwei Vorgehensweisen, um Topics zu erstellen, welche Vorteile und Nachteile aufweisen. Diese werden am Ende des Unterkapitels erörtert.

Für die „Start“ Analyse-Events werden jeweils zwei Topics für die zwei Microservices BLAST und GC-Content erstellt. Damit eine Skalierung der Microservices vollzogen werden kann, müssen diese mit mehreren Partitionen erstellt werden, da ein Kafka-Consumer immer mindestens einer Partition fest zugewiesen wird. Ist eine Topic mit nur einer Partition erstellt, erhält innerhalb einer Consumer-Group nur ein Kafka-Consumer die Events, auch wenn zwei oder mehrere zur Verfügung stünden [53]. Aus diesem Grund muss schon beim Erstellen einer Topic die maximale Anzahl an Kafka-Consumern innerhalb einer Consumer-Group feststehen. Dies impliziert die maximale Skalierung der Microservices für eine Aufgabe. Um die Möglichkeit der Skalierung von GenomicInsights zu ermöglichen, wird eine Partitionsanzahl von zwei gewählt. Eine Skalierung des Report Microservices ist im Rahmen von GenomicInsights nicht vorgesehen, da die Erstellung eines Reports einerseits nur die Transformation der Ergebnisdaten erfordert. Auch die „*_report“ Topics werden mit nur einer Partition erstellt, da hier die Consumer in unterschiedlichen Consumer-Groups sind, um alle Events zu empfangen und so den Workflow in Apache Airflow weiterzuführen. Details werden in dem Unterkapitel zu Apache Airflow beschrieben.

Tabelle 1: Kafka Topics (P6)

Name	Partitionsanzahl	Events
blast_analyzer	2	Start der BLAST Analyse
blast_report	1	Erfolgsmeldung der BLAST Analyse
gc_analyzer	2	Start der GC-Content Berechnung
gc_report	1	Erfolgsmeldung der GC-Content Berechnung
report	1	Start / Erfolgsmeldung der Reporterstellung

Wie der Tabelle 1 entnommen werden kann, ist die Report-Topic die einzige, bei der die Start- und Erfolgsmeldung über eine Topic geschickt werden. In diesem Fall erhalten alle Kafka-Consumer innerhalb der unterschiedlichen Apache Airflow Workflows die Nachricht, welche auf eine Successnachricht warten sowie der Consumer des Report Microservices. Dies kann bei einer hohen Skalierung zu viel Netzwerkverkehr durch das Senden von Events führen. Im Falle des Report Microservices wird dieser aufgrund der schon beschriebenen Gründe nicht skaliert. Weiterhin muss bei dem Report-Microservice unter den zwei unterschiedlichen Events (Start Erstellung Report und Erfolgsmeldung Report) unterschieden werden. Vorteil von einer Topic ist die reduzierte Komplexität (Anzahl der Topics). Eine Topic kann wie folgt im Kafka-Broker angelegt werden:

```
kafka-topics --create --topic gc_analyzer --replication-factor 1 --partitions 2 --bootstrap-server localhost:9092
```

In der Tabelle 2 sind die unterschiedlichen Kafka Events abgebildet. Events, welche eine Analyse starten, haben keinen „Key“. Die Kafka Producer entscheiden, in welche Partition innerhalb einer Topic die Nachricht gesendet wird. Beeinflussend ist dabei der Key. Ist ein Key gesetzt, schickt ein Producer ein Event mit dem gleichen Key immer in die gleiche Partition. Dies hat zur Folge, dass immer der gleiche Consumer das Event erhält, was eine Skalierung der Analyse Microservices nicht möglich macht.

Tabelle 2: Aufbau der Kafka Events (P6)

Events	Key	Value
Start der BLAST Analyse	none	WorkflowID, Dateiname
Erfolgsmeldung der BLAST Analyse	“Report_BLAST”	WorkflowID, “Successful”
Start der GC-Content Berechnung	none	WorkflowID, Dateiname
Erfolgsmeldung der GC-Content Berechnung	“Report_GC”	WorkflowID, “Successful”
Start Reporterstellung	“Report”	WorkflowID, Resultdateiname
Erfolgsmeldung der Reporterstellung	“Report”	WorkflowID, “Successful”, Workflow

P6: Event Streaming Platform (FF3, VH12, M10 und M11)

4.3.3. Apache Airflow

Für Apache Airflow (P7) ist ein offizielles Dockerimage der Apache Foundation auf Dockerhub verfügbar. Dies kommt innerhalb von GenomicInsights zum Einsatz und wird mittels einem vordefinierten docker-compose File, auf welches in der Apache Dokumentation verwiesen wird, deployed [54]. In das Dockerimage müssen weitere Python Module nachinstalliert werden, z. B. das Python-Kafka Modul. Neben Apache Airflow müssen weitere Komponenten, wie z. B. eine PostgreSQL Datenbank, gestartet werden. Diese sind ebenfalls im verfügbaren docker-compose File definiert. Apache Airflow wird in der zum Zeitpunkt der Erstellung von GenomicInsights verfügbaren Version 2.4.0 benutzt.

Apache Airflow wird zur Workflowdefinition innerhalb des GEF und in GenomicInsights verwendet. Dort wird für beide vorgestellten Use-Cases jeweils ein Workflow definiert, welcher die Abfolge von Events beschreibt. Die Workflows verwenden für die einzelnen Schritte einen PythonOperator, welcher unterschiedliche Methoden aufruft, um Kafka Events zu senden oder zu empfangen. Für den Start beider Workflows muss ein Dateiname der Rohdatei dem Workflow übergeben werden. Danach wird das entsprechende Analyseevent in die richtige Topic geschrieben. Ist dies geschehen, wird

ein Kafka-Consumer in die jeweilige Report Topic mit der eindeutigen Consumer-Group (der WorkflowID) eingeschrieben. Dadurch erhält dieser jedes Event innerhalb dieser Topic. Der Task innerhalb des Workflows bleibt so lange aktiv, bis die entsprechende Successmeldung mit der passenden WorkflowID des Workflows eintrifft. Dann wird im nächsten Task die Erstellung des Reports durch ein Event in die „report“ Topic gestartet. Zum Schluss wird wieder ein Kafka-Consumer-Task mit der eindeutigen Consumer Group der WorkflowID gestartet, bis eine Successmeldung mit der passenden WorkflowID des Workflows eintrifft. Die Pipeline wurde ab diesem Punkt erfolgreich durchlaufen und die Reportdatei erstellt. In der Abbildung 37 sind die Sequenzdiagramme beider Workflows abgebildet. Der Eventaufbau der unterschiedlichen Events ist im Kapitel 4.3.2 genauer erläutert.

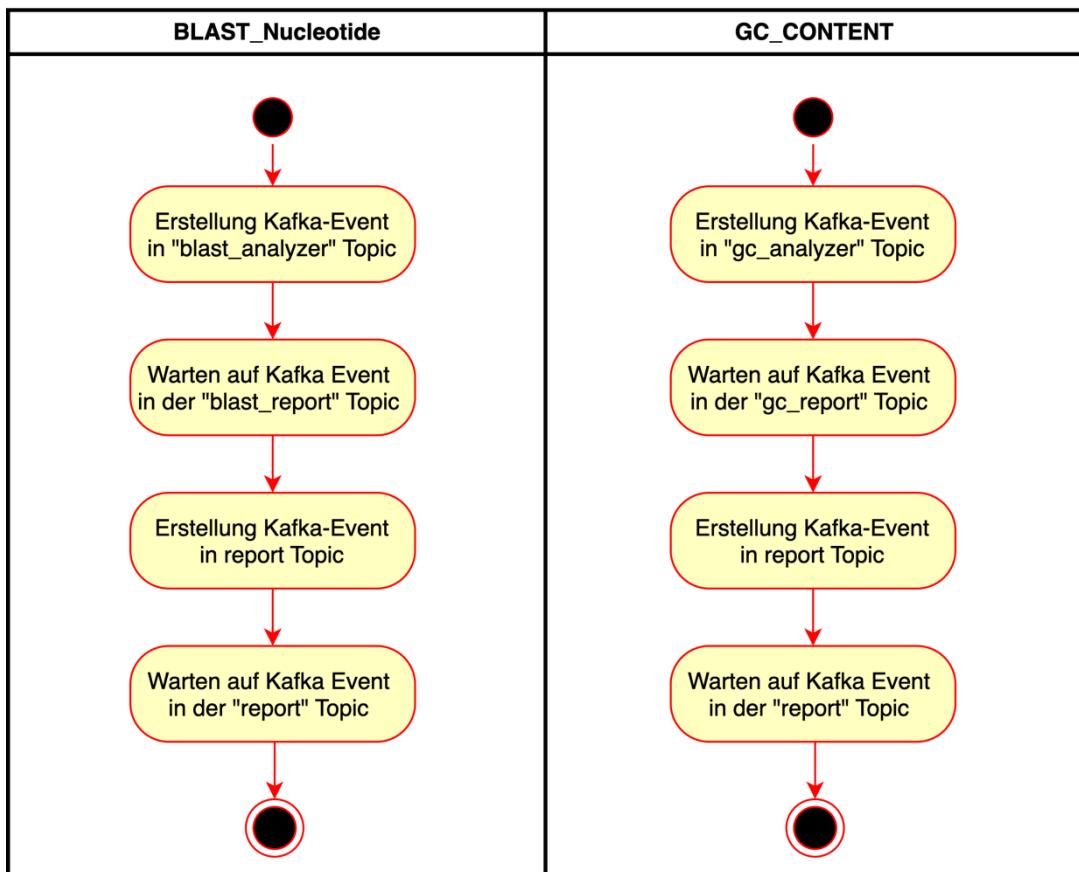


Abbildung 37: Aktivitätsdiagramm GenomicInsights Workflows (P7)

Ein Workflow wird in einem Python-Script definiert, welches durch Apache Airflow vordefiniert ist. Im folgenden Abschnitt ist der Pythoncode zur Workflowdefinition zu sehen.

```

72     with DAG(
73         "BLAST_NUKLEOTID",
74         default_args=default_args,
75         description="Analyse Nukleotidsequences with BLAST",
76         schedule_interval=None,
77         catchup=False,
78         tags=["BLAST"],
79     ) as dag:
80
81         t1 = PythonOperator(
82             task_id="BLAST_Analyse_Requested",
83             python_callable=produce_blast_request,
84             op_kwargs={"run_id": "{{ run_id }}", "file": "{{ dag_run.conf['file'] }}"},
85         )
86
87
88         t2 = PythonSensor(
89             task_id="BLAST_Analyse_Finished",
90             python_callable=analyzer_sensor,
91             op_kwargs={"run_id": "{{ run_id }}"},
92         )
93
94         t3 = PythonOperator(
95             task_id="Report_Requested",
96             python_callable=produce_report_request,
97             op_kwargs={"run_id": "{{ run_id }}", "file": "{{ dag_run.conf['file'] }}"},
98         )
99
100        t4 = PythonSensor(
101            task_id="Report_Finished",
102            python_callable=report_sensor,
103            op_kwargs={"run_id": "{{ run_id }}"},
104        )
105        t1 >> t2 >> t3 >> t4

```

Abbildung 38: Screenshot Airflow DAG BLAST Workflow (P7)

P7: Workflow Management Platform und Implementierung der Workflows aus P1 und P2 (FF3, VH13 und VH14, M10 und M11)

4.3.4. Report Microservices

Der Report Microservice (P8) baut wie die anderen selbstimplementierten Microservices im Rahmen von GenomicInsights auf einem Python Dockerimage auf. Nachinstalliert werden Python Module, wie z. B. Kafka-Python, FPDF2 oder Matplotlib.

Der Microservice schreibt sich als Kafka-Consumer in die „report“ Topic ein. Kommt dort ein Event an, wird zunächst überprüft, ob eine „Successmeldung“ einer Reporterstellung angekommen ist. Wenn dem so ist, werden keine weiteren Aktionen durchgeführt. Kommt ein Event zur Reporterstellung an, wird mit Hilfe des gesetzten „Workflow“ Values in dem Event unterschieden, wie der Report je nach Workflow erstellt wird. Dafür gibt es zwei unterschiedliche Funktionen, welche die Reporterstellung je nach Workflow durchführen. Für die Reporterstellung benötigt der Microservice Zugriff auf den Resultordner und den Reportordner. Im Falle von BLAST ist der Output schon ein PDF, welches im Resultordner liegt. Hier wird nur eine Überschrift „Report“ hinzugefügt. Der GC-Content Microservice schreibt in den Resultordner eine Liste der GC-Contents für jede Sequenz im FASTQ File. Hier wird der Mittelwert aller Werte berechnet und mit Hilfe von Matplotlib eine Grafik erstellt. Diese zeigt die einzelnen GC-Content Werte als Graph.

In der Abbildung 39 ist ein Beispielreport zum Workflow der Berechnung des GC-Contents zu sehen.

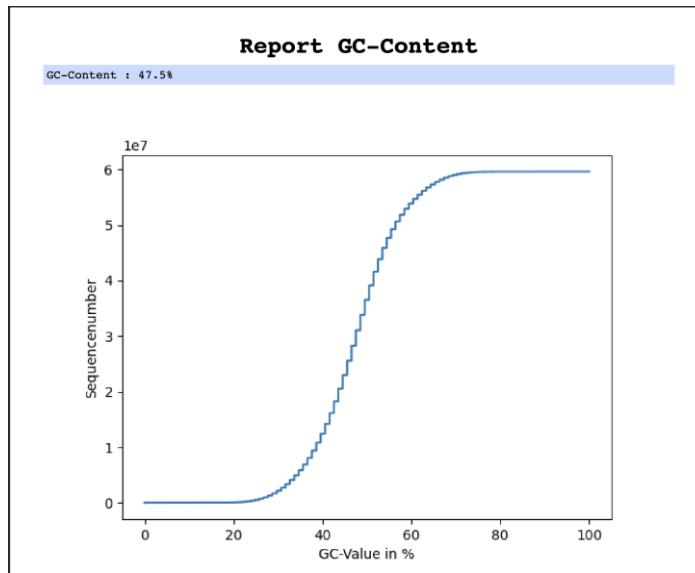


Abbildung 39: Screenshot Beispielreport GC-Content (P8)

Als Report wird immer ein PDF in der Namenskonvention „report_filename_workflowid.pdf“ im Reportordner erstellt. Nach erfolgreicher Erstellung wird ein Successevent in die „report“ Topic geschrieben.

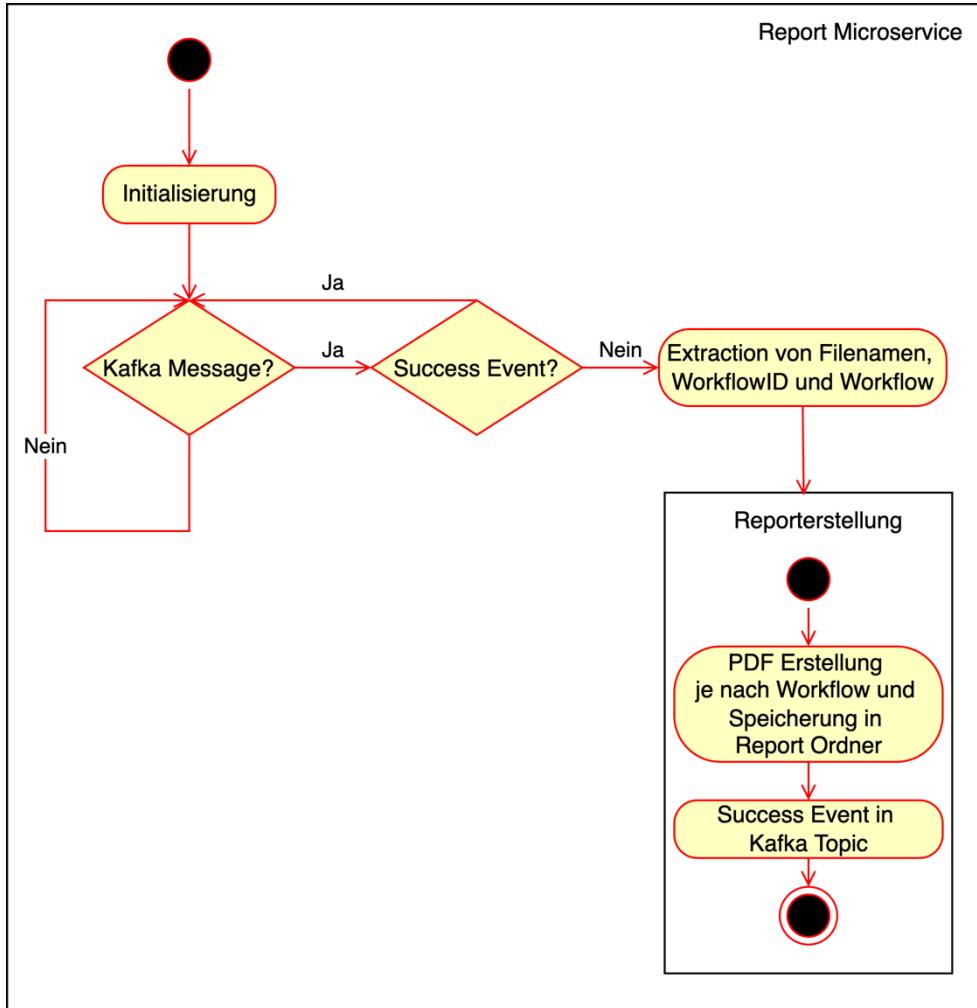


Abbildung 40: Aktivitätsdiagramm Report Microservice (P8)

P8: Report Microservice (FF3, VH15, M12)

4.3.5. Zusammenfassung der Implementierung einer Softwarelösung für eine intuitive Benutzungsschnittstelle des IT-Systems im Rahmen eines POCS

In den Unterkapiteln wurden das Frontend von GenomicInsights, die Konfiguration von Airflow und Kafka sowie die Implementierung des Report Microservices vorgestellt. Für einzelne Komponenten ist ein Ablaufdiagramm zu sehen und Konfigurationsdetails von Apache Kafka und Apache Airflow sind erörtert.

4.4. Deployment der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs

Alle Bestandteile von GenomicInsights sind als Docker Container auf einem Single Host deployed. In den folgenden Unterkapiteln werden Details zum Deployment von GenomicInsights vorgestellt. In der Abbildung 41 ist ein Architekturdiagramm des gesamten Prototyps aus der Sicht von Docker zu sehen.

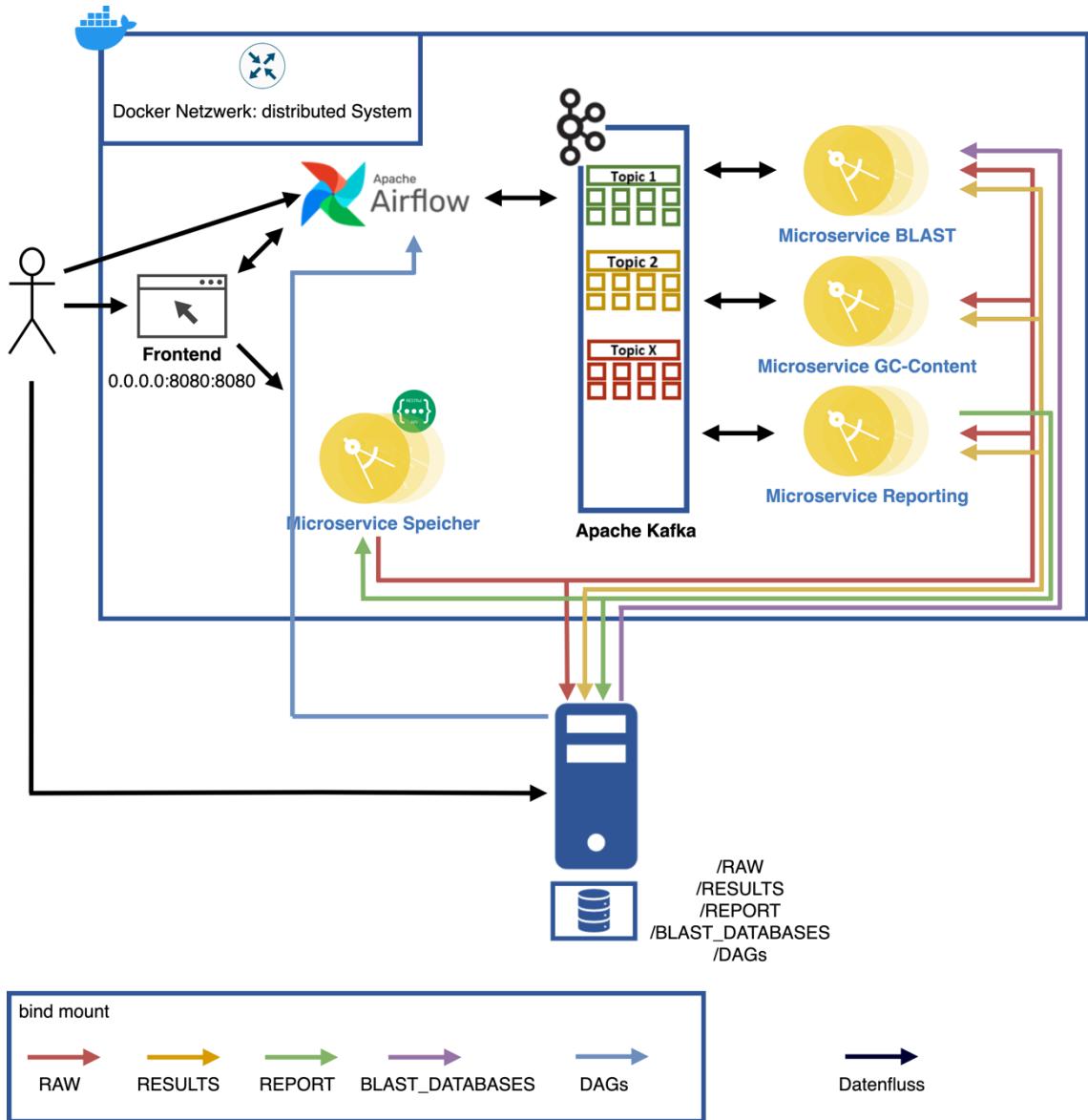


Abbildung 41: Deployment-Plan GenomicInsights auf Docker (P9)

P9: Deployment GenomicInsights innerhalb von Docker (FF4)

4.4.1. Deploymentmethode

Als Deploymentmethode sind für alle Microservices einzelne Docker-Compose Files erstellt oder schon vorhandene angepasst worden. Durch die Verwendung von Docker-Compose Files ist der Bau der Container und das Deployment dokumentiert.

Mit einem „docker-compose build“ wird das entsprechende Image gebaut und mit „docker-compose up -d“ die Applikation gestartet. In der Abbildung 42 ist das Docker-Compose File von BLAST zu sehen.

```
1  ---
2  version: '3'
3
4  services:
5    blast_microservice:
6      build: .
7      image: blast_microservice:latest
8      volumes:
9        - ./BLAST_DATABASES/db/:/blast/db/
10       - ../../data/results/:/blast/results/
11       - ../../data/raw/:/blast/raw/
12     restart: always
13
14
15   networks:
16     default:
17       name: distributed_system_network
18       external: true
```

Abbildung 42: Screenshot BLAST Docker-Compose File (P9)

4.4.2. Netzwerk

Damit die einzelnen Services miteinander kommunizieren können und eine DNS-Auflösung mittels Containernamen möglich ist, muss ein eigenes Docker-Netzwerk erstellt werden. In der Abbildung 41 ist unter „networks“ das Netzwerk definiert, in dem sich alle Services befinden. Alle Docker-Compose Files haben diese Anpassung bezüglich des Netzwerks. Beim Start wird automatisch überprüft, ob das Netzwerk schon angelegt wurde und fügt dem Netzwerk den Container hinzu.

4.4.3. Speicher

Der RAW-, RESULTS- und Reportordner werden als sogenannter Bind-Mount in dem Container zur Verfügung gestellt. Dabei werden für jeden Microservice die richtigen Ordner im Docker-Compose File hinzugefügt.

In der Abbildung 41 ist unter „Volumes“ die Bind-Mounts des BLAST Microservices zu sehen.

4.4.4. Zusammenfassung des Deployments der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs

Alle Komponenten von GenomicInsights sind als Dockercontainer innerhalb eines Hosts mit Hilfe von „docker-compose“ deployed. Dabei befinden sich alle Container in demselben Netzwerk, um eine reibungslose Kommunikation und Namensauflösung zu ermöglichen. Die Speicherorte sind als „Bind-Mounts“ in den Containern zur Verfügung gestellt.

4.5. Zusammenfassung der Implementierung

In diesem Kapitel wurden aufbauend auf Kapitel 3 alle Forschungsziele, welche im Kapitel 1.6 nach X.1/I vorgestellt wurden, adressiert. Dabei wurden zu Beginn die Bestandteile und Zielarchitektur von GenomicInisghts aufgezeigt. In den Unterkapiteln wurden einzelne Teile von GenomicInisghts genauer beschrieben und Implementierungsdetails erörtert. Im nächsten Kapitel werden Experimente zu allen Forschungszielen anhand von GenomicInisghts durchgeführt. In der Abbildung 43 ist das Sequenzdiagramm von GenomicInisghts abgebildet. Dort zu entnehmen ist der Nachrichtenaustausch zwischen den einzelnen Komponenten am Beispiel des BLAST Workflows.

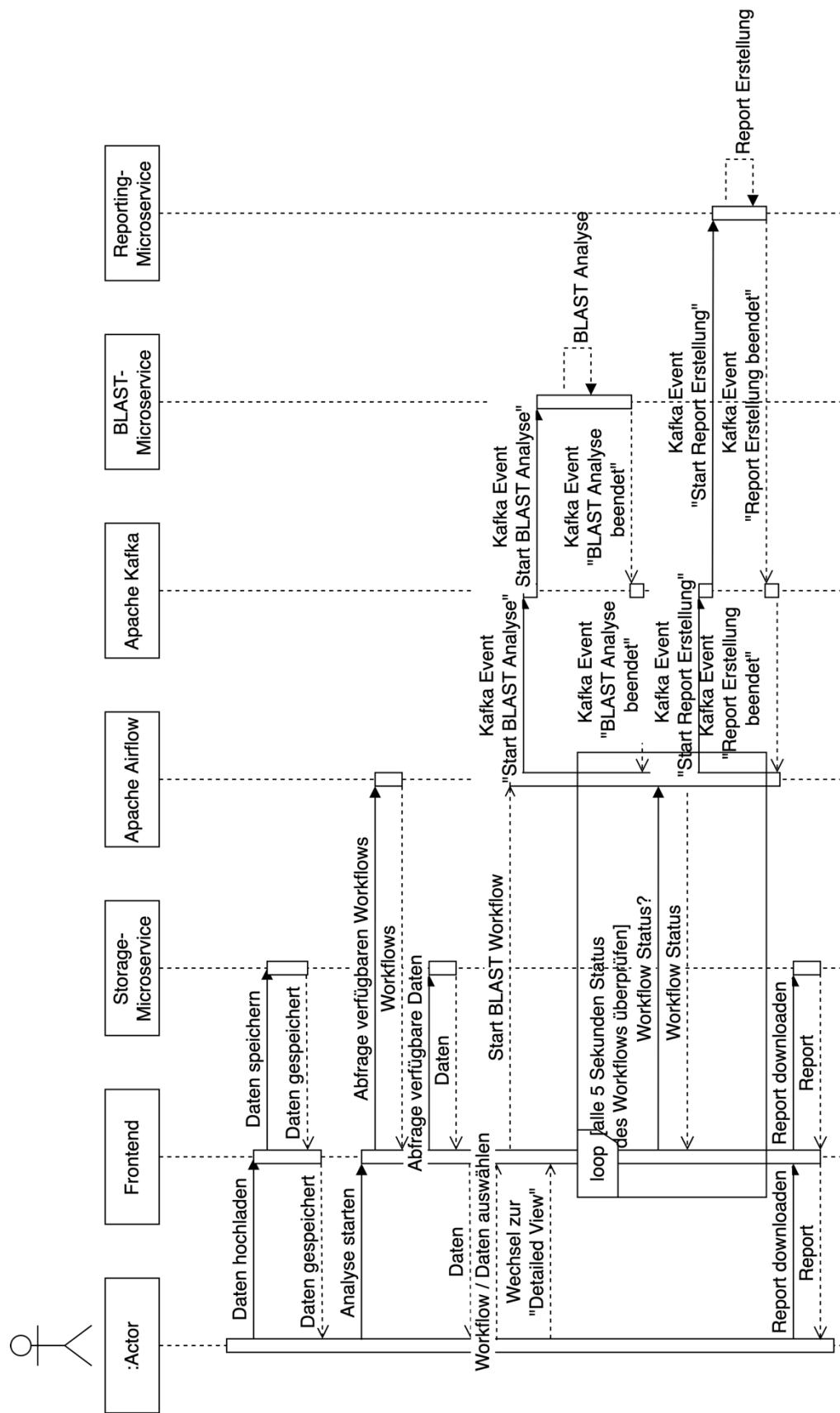


Abbildung 43: Sequenzdiagramm von GenomicInsights

Folgende Prototypen wurden in der Implementierung entwickelt und den Kategorien Analyse, Infrastruktur und Anwendung eingeteilt:

Analyse:

P1: Wrapper BLAST Analyse Microservice (FF1, VH5, M5)

P2: Wrapper GC-Content Analyse Microservice (FF1)

Infrastruktur:

P3: Konfiguration des Speichers (FF2)

P4: Storage Microservice (FF2, VH10, M8)

P6: Event Streaming Platform (FF3, VH12, M10 und M11)

P7: Workflow Management Platform und Implementierung der Workflows aus P1 und P2 (FF3, VH13 und VH14, M10 und M11)

P8: Report Microservice (FF3, VH15, M12)

P9: Deployment GenomicInisights innerhalb von Docker (FF4)

Anwendung:

P5: Benutzungsschnittstelle (FF3, VH11, M9)

Im nächsten Kapitel werden alle genannten Prototypen evaluiert.

5. Evaluation

In den folgenden Unterkapiteln wird eine Evaluation von GenomicInsights aus Kapitel 4 durchgeführt. Dabei ist die Evaluation der letzte Schritt der in Kapitel 1.4 vorgestellten Nunamaker Methodik. Alle Forschungsziele, welche im Kapitel 1.6 nach X.1/E sortiert wurden, haben eigene Unterkapitel. Zur Evaluation von GenomicInisghts kommen die Methodik der Testcases und Cognitive Walkthroughs zum Einsatz. Innerhalb der Testcases wird das zu erwartende Ergebnis vor dem Test definiert und am Ende mit diesem verglichen (input-process-output-objectives perspective) [55]. Die Cognitive Walkthroughs sind eine Form der Qualitativen Bewertung. Dabei werden z. B. die Use-Cases der Benutzungsschnittstelle mit einem Benutzer der Schnittstelle verwendet und am Ende Verbesserungsvorschläge für weitere Entwicklungen durch den Benutzer beschrieben [56]. In der Tabelle 3 ist eine Übersicht abgebildet, welche Komponenten von GenomicInisghts mit welcher Evaluationsmethode getestet werden.

Tabelle 3: Übersicht der Evaluation von GenomicInisghts

Komponente von GenomicInisghts	Evaluationsmethode
Analysesoftware (P1, P2)	Testcases
Speicherlösung (P3, P4)	Testcases
Benutzungsschnittstelle (P5)	Cognitive Walkthrough, Testcases
Reporting (P8)	Testcases
Apache Kafka (P6)	Testcases
Apache Airflow (P7)	Testcases
Deployment (P9)	Testcases

Innerhalb der Testcases kommen zwei Beispieldatensätze in unterschiedlichen Formaten zum Einsatz. Der erste Beispieldatensatz ist ein Analysebeispiel der National Library of Medicine [57]. Dort sind Beispieldaten zur Analyse im FASTA-Format hinterlegt und ebenfalls das zu dem Analysebeispiel passende Ergebnis. Der zweite Beispieldatensatz stammt von dem Human Microbiome Project Data Portals und hat die ID „SRS012969“ [58]. Dieser Datensatz ist im FASTQ Format.

Wenn die Evaluation nicht das gewünschte Ergebnis liefert und aus diesem Grund in der Zukunft noch etwas weiterentwickelt werden muss, wird dies als „zukünftige Arbeit“ (ZA1 -ZAn) gekennzeichnet.

5.1. Evaluation der Open-Source Big-Data Analyse Software mit einem Analyse Use-Case

Innerhalb der Experimente mit dem BLAST und GC-Content Microservice geht es um die korrekte Verarbeitung des Kafka-Events und das Erstellen des „Success“-Events, den korrekten Zugriff auf den Speicher, die Analyse bzw. Berechnung auf Basis der in der Einleitung von Kapitel 5 erwähnten Daten und die korrekte Erstellung der Result-Datei, welche die Vorbereitung für den Report-Microservice darstellt.

5.1.1. BLAST Microservice

Der BLAST Microservice wird mit dem ersten Beispieldatensatz im FASTA-Format getestet. Um eine Analyse mit dem BLAST Microservice durchzuführen, wird neben den Daten ebenfalls eine Datenbank zur Analyse benötigt. Diese Datenbank muss zum Analyse Use-Case passend ausgewählt werden. Für die implementierte Analyse in Kapitel 4.1 des BLAST Microservices und die Beispieldaten ist die Datenbank mit dem Namen „NT.00“ für das Experiment heruntergeladen und innerhalb des BLAST Database Ordners entpackt worden. Diese und weitere Datenbanken können auf dem FTP-Server von NCBI heruntergeladen werden [59]. Innerhalb des Beispieldatensatzes sind zwei Analysen beschrieben, welche im Rahmen der Evaluation des Microservices ebenfalls zum Einsatz kommen. Um die Ergebnisse nach der Analyse zu evaluieren, wird der standardmäßige Report von BLAST mit den beschriebenen Ergebnissen auf der Website verglichen. Dieser wird in den Result Ordner geschrieben. Nach dem Eintreffen des Kafka Events zum Start der Analyse wurden beide Datensätze korrekt analysiert und die Ergebnisse stimmen mit denen des Analysebeispiels der National Library of Medicine überein. Zum Schluss wurde ein korrektes „Success“-Event erstellt.

5.1.2. GC-Content Microservice

Für das Experiment des GC-Content Microservices wird der zweite Datensatz, welcher in der Einleitung vorgestellt wurde, verwendet. Das gewünschte Ergebnis stellt eine Liste mit dem GC-Content der jeweiligen Sequenzen aus dem Datensatz dar. Nach dem Eintreffen des Kafka Events zum Start der Berechnung ist diese erfolgreich durchgeführt worden. Im Resultordner befindet sich eine Liste mit den prozentualen Werten für jede Sequenz. Zum Schluss ist ein korrektes „Success“-Event erstellt worden.

5.1.3. Zusammenfassung der Evaluation der Open-Source Big-Data Analyse Software mit einem Analyse Use-Case

Beide Microservices arbeiten korrekt mit den Kafka-Events zum Start der Analyse bzw. der Berechnung. Während der Analyse bzw. Berechnung werden die erwarteten Daten in den Resultordner mit entsprechender Namenskonvention geschrieben. Die drei Analyse Microservice Wrapper für Apache Spark (M3), QIIME2 (M4) und ASaiM (M6), welche im Prototyp nicht implementiert worden sind, können als zukünftige Arbeit identifiziert werden. Des Weiteren können die ungelösten verbleibenden Herausforderungen (VH 2, 4, 6, und 8) zur Erstellung und / oder Auswahl von Analysealgorithmen ebenfalls als ZA definiert werden.

ZA1: Implementierung Microservice Wrapper Apache Spark (M3).

ZA2: Implementierung Microservice Wrapper QIIME2 (M4).

ZA3: Implementierung Microservice Wrapper ASaiM (M6).

ZA4: Implementierung von Algorithmen zur Analyse von Genomdaten auf Basis von Apache Spark (VH2).

ZA5: Auswahl der Algorithmen von QIIME2 (VH4).

ZA6: Auswahl der Analysemethoden von BLAST sowie Datenbanken, mit denen die Sequenzen verglichen werden (VH6).

ZA7: Auswahl der Analysemethoden von ASaiM (VH8).

5.2. Evaluation der Open-Source Speicherlösung durch einen Use-Case

Um die Implementierung der Speicherlösung zu testen, werden einerseits Experimente mit dem Dateisystem durchgeführt und andererseits mit dem Storage Microservice. Diese werden mit den in der Einleitung erwähnten Beispieldatensätzen oder mit den erstellten Daten der jeweiligen Microservices getestet.

5.2.1. Dateisystem

Das Experiment mit dem Dateisystem bezieht sich auf das korrekte Lesen und Schreiben der jeweiligen Ordner. Diese werden durch die implementierten Microservices von GenomicInsights benutzt. In diesem Fall wird eine Beobachtung durchgeführt, ob Daten innerhalb der Ordner korrekt geschrieben und gelesen werden können. Während der Durchführung der unterschiedlichen Microservices wurde diese Beobachtung durchgeführt. Das Ergebnis der Beobachtung ist, dass auf alle Ordner korrekt geschrieben und gelesen werden kann und das Dateisystem aus diesem Grund funktioniert.

5.2.2. Storage Microservice

Innerhalb des Storage Microservices sind drei Use-Cases implementiert, die durch ein Experiment getestet werden können. Für das Experiment werden die beschrieben HTTP-Anfragen an den Microservice gestellt. Der Erwartungswert der Use-Cases ist wie folgt:

/upload, POST-Methode: Die Daten werden mit dem richtigen Namen in den RAW-Ordner geschrieben.

/getFiles, GET-Methode: Die Dateinamen der verfügbaren Daten innerhalb des RAW-Ordners werden zurückgegeben.

/downloadReport, GET-Methode: Der richtige Report wird im Report-Ordner als BLOB zum Download zur Verfügung gestellt.

Alle drei Use-Cases wurden erfolgreich durchgeführt und alle Erwartungswerte sind erfüllt.

5.2.3. Zusammenfassung der Evaluation mit der Open-Source Speicherlösung durch einen Use-Case

Für die Speicherlösung wurden die Konfiguration des Dateisystems und die Funktion des Storage Microservices überprüft. Daten können in den jeweiligen Ordner geschrieben und gelesen werden und alle drei Use-Cases des Storage Microservices haben die Erwartungswerte erfüllt. Als zukünftige Arbeit kann die Implementierung von CEPH als Speicherlösung definiert werden (M7):

ZA8: Implementierung von CEPH als Speicherlösung (M7)

5.3. Evaluation zur Benutzung einer Benutzungsschnittstelle des IT-Systems anhand eines Analyse Use-Case

In den folgenden Unterkapiteln sind Experimente zur Benutzungsschnittstelle, Apache Kafka, Apache Airflow und dem Reporting Microservice beschrieben. Die Experimente der Benutzungsschnittstelle, Apache Kafka und Apache Airflow beziehen sich auf die Durchführung beider Analyse-Workflows (BLAST und GC-Content). Benutzungsschnittstelle

5.3.1. Benutzungsschnittstelle

Das Experiment der Benutzungsschnittstelle bezieht sich auf die vier definierten Use-Cases der Benutzungsschnittstelle und weiteren Softwarefeatures. Die Benutzungsschnittstelle wurde durch einen Funktionstest (mit Hilfe der beschriebenen Use-Cases des Frontends) und durch einen Cognitive Walkthrough getestet. Dieser wurde mit dem wissenschaftlichen Betreuer der Masterarbeit, Herrn Krause, durchgeführt und stellt eine qualitative Evaluation des Frontends dar. Die Ergebnisse der vier definierten Use-Cases sind in der Tabelle 4 zu sehen.

Tabelle 4: Funktionstest der Benutzungsschnittstelle

Use-Case	Beschreibung	Funktionstest
U1	Daten Hochladen	✓
U2	Analyseworkflow Auswählen	✓
U3	Analyseworkflow Starten	✓
U4	Report anzeigen / downloaden	✓

Die Ergebnisse des Cognitive Walkthroughs sind in der Tabelle 5 aufgelistet.

Tabelle 5: Cognitive Walkthrough weitere Software-Features (ZA1)

Weitere Software Features	Beschreibung
Navigation Highlighting	Aufzeigen anhand des Menüs (auf der linken Seite des Frontends), in welchem Dialog sich der Benutzer befindet (farblich hinterlegt)
About-Dialog	Zusätzlicher About-Dialog, in welchem die Version der Software, sowie der Komponenten und Analysealgorithmen aufgelistet sind
Formatvalidierung	Überprüfung / Einschränkung der verfügbaren Daten anhand der unterstützten Dateitypen eines Workflows
Active Workflows	Ausimplementierung der aktiven Workflows (gestartete Workflows der letzten 24h)
Fehlerhandling	Fehler, die innerhalb der Durchführung des Workflows, z. B. innerhalb der Microservices, auftreten dem Benutzer sichtbar machen
Zugriff auf Logs	Zugriff auf Logs des Analyseworkflows

ZA9: Erweiterung der Benutzungsschnittstelle

5.3.2. Apache Kafka

Das Experiment zu Apache Kafka erfolgt indirekt durch Beobachtung während des Durchlaufens des Analyse-Workflows. Während des Durchlaufs können die Kafka-Events von den Producern erstellt und über die Topic einem Consumer zur Verfügung gestellt werden. Insgesamt müssen für einen Workflow vier Events über Apache Kafka geschickt werden. Am Beispiel der BLAST Analyse sind dies folgende:

- BLAST Analyse starten
- BLAST Analyse beendet
- Report erstellen
- Report erstellt

Alle vier Nachrichten sind erfolgreich in den in Kapitel 4.3.2 beschriebenen Topics vorhanden. Sind mehrere BLAST-Microservices deployed und werden mehrere Analyseworkflows gestartet, werden die Events an unterschiedliche Microservices

geschickt, um eine horizontale Skalierung zu ermöglichen. Für die Evaluation sind zwei BLAST-Microservices zum Einsatz gekommen, die durch die Konfiguration in Kapitel 4.3.2 beide Events bei mehrmaligem Starten von Analysen erhalten haben. Dies konnte über debugging Logs sichtbar gemacht werden.

5.3.3. Apache Airflow

Die hinterlegten Tasks in Apache Airflow beziehen sich auf die Erstellung und den Empfang von Kafka-Events. Für die Durchführung eines beschriebenen Workflows werden die vier Kafka-Events aus Kapitel 5.3.2 erstellt oder auf diese gewartet. Die Erwartungswerte für das Experiment sind, dass die Kafka-Events korrekt erstellt werden und das Warten auf Events funktioniert. Als Beispiel ist hier zu erwähnen, dass der Report nicht vor der Fertigstellung der Analyse oder Berechnung gestartet wird. Beide Erwartungswerte sind erfüllt und in der Abbildung 44 ist das Warten auf die Fertigstellung der Analyse des BLAST Microservices zu sehen.

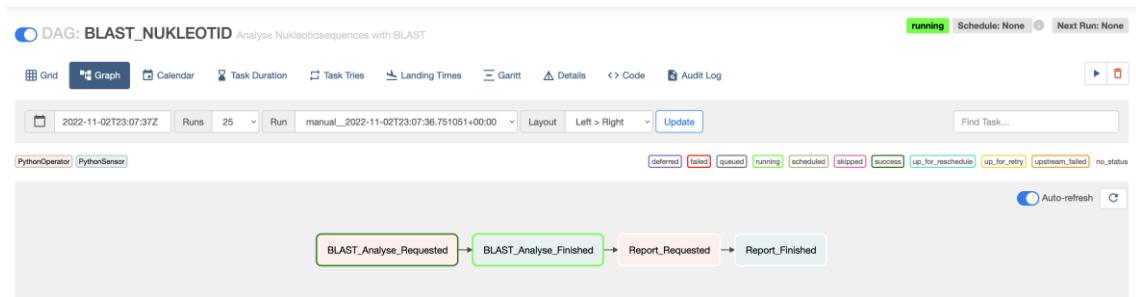


Abbildung 44: Screenshot Ariflow BLAST Workflow

Die Messung der Dauer, bis der BLAST und GC-Content Workflow durchlaufen sind, (von Start bis Ende des Workflows in Apache Airflow) hat folgende Ergebnisse hervorgebracht:

BLAST: 00:45 Minuten

GC-Content: 40:13 Minuten

Die Durchlaufzeiten hängen stark mit der verwendeten Hardware zusammen. Für die Evaluation von GenomicInisights ist ein Macbook Pro 2021 mit M1 Pro CPU, 32GB Arbeitsspeicher und 2TB Festplatte zum Einsatz gekommen.

5.3.4. Reporting Microservice

Der Erwartungswert für die Experimente mit dem Reporting Microservice ist, dass am Ende beider implementierten Analyse Workflows auf Basis deren Ergebnisse ein Report im Report-Ordner erstellt wird. Dafür muss der Microservice Kafka-Events erhalten und diese für beide Workflows korrekt verarbeiten und am Ende eine „Successmeldung“ zurückgemeldet werden. Weiterhin muss dann die Logik für die Erstellung des Reports für den gestarteten Workflow korrekt durchgeführt werden. Die Kafka-Events wurden während der Experimente korrekt verarbeitet und die „Successmeldung“ zurückgeschickt. Dies konnte einerseits durch die korrekte Erstellung eines Reports für den ausgewählten Workflow und der Analyseergebnisse und andererseits durch den wartenden Task von Apache Airflow validiert werden. Für beide implementierten Analyseworkflows wurden die beschriebenen Experimente durchgeführt und funktionieren.

5.3.5. Zusammenfassung der Evaluation zur Benutzung einer für eine intuitiven Benutzungsschnittstelle des IT-Systems anhand eines Analyse Use-Case

Die definierten und durchgeführten Experimente für die Benutzungsschnittstelle, Apache Kafka, Apache Airflow und dem Reporting Microservice konnten alle als erfolgreich evaluiert werden. Neben den Use-Cases der Benutzungsschnittstelle konnten während dem Cognitive Walkthrough für die Zukunft Software-Features identifiziert werden, welche die Benutzerfreundlichkeit der Benutzungsschnittstelle verbessern. Neben der Benutzungsschnittstelle können in der Zukunft innerhalb von Apache Airflow komplexere Analyseworkflows definiert werden und die Reports des Reporting Microservices können an die Bedürfnisse des Labors angepasst werden. Daraus ergibt sich folgende Arbeit für die Zukunft:

ZA9: Erweiterung der Benutzungsschnittstelle

ZA10: Implementierung komplexerer Workflows

ZA11: Anpassung der Reports an die Bedürfnisse des Labors

5.4. Evaluierung des Deployments der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs

Die Experimente zum Deployment der gesamten Softwarelösung sind in den folgenden Unterkapiteln beschrieben. Diese beziehen sich auf das Deployment, das Netzwerk und den Speicher. Alle drei Experimente bilden die Basis, um die Software zu starten und einem Benutzer Analysen zu ermöglichen.

5.4.1. Deployment der Softwarelösung

Alle Softwarekomponenten können erfolgreich mit Docker-Compose auf einem Single-Node deployed und gestartet werden. Die dort enthaltene Konfiguration der Container (z. B. Bind-Mounts) wurde erfolgreich durchgeführt. Neben dem Deployen können die Container mit der hinterlegten Konfiguration ebenfalls mit Hilfe von Docker-Compose gebaut und alle benötigten Libraries korrekt installiert werden.

5.4.2. Netzwerk

Das Experiment zum Netzwerk beinhaltet die Beobachtung, ob Softwarekomponenten über das Docker-Netzwerk miteinander kommunizieren können. Dies ist z. B. das Starten von Workflows über das Frontend oder die Kommunikation des Frontends mit dem Storage Microservice. Alle Komponenten können über die definierten Wege (Kafka-Events oder REST) miteinander kommunizieren. Aus diesem Grund kann das Experiment als erfolgreich angesehen werden.

5.4.3. Speicher

Der Erwartungswert des Experiments zum Speicher ist, dass die Microservices auf die richtigen Ordner lesend sowie schreibend zugreifen können. Dies ist mit Hilfe der Durchführung beider Workflows geschehen. Dabei greifen unterschiedliche Microservices auf alle Ordner zu und es kann am Ende des Workflows validiert werden, ob die richtigen Dateien gelesen und geschrieben wurden. Für alle Microservices, welche auf den Speicher zugreifen, ist der Erwartungswert eingetroffen.

5.4.4. Zusammenfassung der Evaluation des Deployments der gesamten Softwarelösung zur Analyse von Genomdaten im Rahmen eines POCs

In den vorherigen Unterkapiteln wurden die Experimente zum Deployment erörtert. Dabei beziehen sich die Experimente auf den Bau und das Deployment der Softwarekomponenten, die Kommunikation einzelner Komponenten über das Docker-Netzwerk und den Zugriff auf den Speicher. Alle Experimente sind erfolgreich. In der Zukunft kann die Architektur auf Kubernetes aufgebaut werden und eine regulatorische Begutachtung muss stattfinden, welche Anforderungen für die Verarbeitung von metagenomischen Daten existieren (z. B. Verschlüsselung des Datenverkehrs). Daraus ergeben sich folgende zwei Arbeiten für die Zukunft:

ZA12: Implementierung der Architektur auf Basis von Kubernetes (M13)

ZA13: Identifizierung von regulatorischen Anforderungen zur Verarbeitung von metagenomischen Daten

5.5. Zusammenfassung der Evaluation des Prototyps

Im Kapitel 5 wurden aufbauend auf der Implementierung von GenomicInsights aus Kapitel 4 Experimente durchgeführt. Dabei sind alle Forschungsziele, welche im Kapitel 1.6 nach X.1/E vorgestellt wurden, adressiert. Zu Beginn eines Experiments wurde die Durchführung und der Erwartungswert beschrieben. Ferner wurde erläutert, ob das Ergebnis der Experimente positiv oder negativ ausfällt. Alle durchgeführten Experimente zu GenomicInsights waren erfolgreich. Folgende zukünftigen Arbeiten konnten während der Evaluierung identifiziert werden:

- ZA1: Implementierung Microservice Wrapper Apache Spark (M3)
- ZA2: Implementierung Microservice Wrapper QIIME2 (M4)
- ZA3: Implementierung Microservice Wrapper ASaIM (M6)
- ZA4: Implementierung von Algorithmen zur Analyse von Genomdaten auf Basis von Apache Spark (VH2).
- ZA5: Auswahl der Algorithmen von QIIME2 (VH4).
- ZA6: Auswahl der Analysemethoden von BLAST sowie Datenbanken, mit denen die Sequenzen verglichen werden (VH6).
- ZA7: Auswahl der Analysemethoden von ASaIM (VH8).
- ZA8: Implementierung von CEPH als Speicherlösung (M7)
- ZA9: Erweiterung der Benutzungsschnittstelle
- ZA10: Implementierung komplexerer Workflows
- ZA11: Anpassung der Reports an die Bedürfnisse des Labors
- ZA12: Implementierung der Architektur auf Basis von Kubernetes (M13)
- ZA13: Identifizierung von regulatorischen Anforderungen zur Verarbeitung von metagenomischen Daten

6. Zusammenfassung und Ausblick

In dem Gebiet der Metagenomik gibt es viele unterschiedliche Softwareanbieter oder Verfahren zur Analyse dieser Daten (z. B. QIIME2, BLAST, Apache Spark, etc.). Es konnte kein IT-System am Markt gefunden werden, welches es ermöglicht, unterschiedliche Analysesoftwares innerhalb des IT-Systems durch eine Benutzungsschnittstelle, Speichermöglichkeit der Daten und eine Reporterstellung zusammenzufassen. Des Weiteren sind die Modularität, Erweiterbarkeit und Open Source des verteilten Systems zentraler Bestandteil der Anforderungen gewesen. In den folgenden Unterkapiteln wird eine Zusammenfassung der vorliegenden Arbeit erstellt, die Realisierung der definierten Forschungsziele analysiert und ein Ausblick für weitere Forschungstätigkeiten gegeben.

6.1. Zusammenfassung

Im Kapitel 1 wurde die Motivation der Analyse von genomischen Daten und die Probleme der Erstellung eines verteilten Systems für eine solche Analyse skizziert. Weiterhin wurden die Forschungsfragen und darauf aufbauend mit welcher Methodik die Arbeit gestaltet wurde, vorgestellt. Im letzten Schritt von Kapitel 1 wurden die vier Forschungsfragen mit Hilfe der Methode auf die Teilstudien heruntergebrochen. Diese Forschungsziele ergeben aus der Nunamaker Methodik die Gliederung der Arbeit:

FF 1: Welche Open-Source Big-Data Softwarelösungen eignen sich zur Analyse von Genomdaten?

FF 2: Wie können Genomdaten persistiert werden?

FF 3: Wie kann eine Benutzungsschnittstelle zur Interaktion zwischen Benutzer und verteiltem System, sowie die Kommunikation zwischen den Komponenten implementiert werden?

FF 4: Wie kann eine Architektur zur Interaktion, Speicherung und Analyse von Genomdaten effizient deployed werden?

Der Methodik folgend ist in Kapitel 2 die Recherche zu allen Recherche-Forschungszielen durchgeführt worden. Dabei wurden unterschiedliche Analysemethoden und Speichermöglichkeiten sowie Möglichkeiten zur Erstellung einer Benutzerschnittstelle und Softwarearchitekturen und Deploymethoden von Software vorgestellt. Aus Kapitel zwei sind 16 verbleibende Herausforderungen definiert worden, welche in Kapitel 3 adressiert wurden:

- VH1: Implementierung des Analysetools Apache Spark in die Architektur (FF1).
- VH2: Implementierung von Algorithmen zur Analyse von Genomdaten auf Basis von Apache Spark (FF1).
- VH3: Implementierung des Analysetools QIIME2 in die Architektur (FF1).
- VH4: Auswahl der Algorithmen von QIIME2 (FF1).
- VH5: Implementierung des Analysetools BLAST in die Architektur (FF1).
- VH6: Auswahl der Analysemethoden von BLAST sowie Datenbanken, mit denen die Sequenzen verglichen werden (FF1).
- VH7: Implementierung des Analysetools ASaiM in die Architektur (FF1).
- VH8: Auswahl der Analysemethoden von ASaiM (FF1).
- VH9: Implementierung der Speicherlösung CEPH in die Architektur (FF2).
- VH10: Implementierung einer API, um den Zugriff auf den Speicher durch die Benutzungsschnittstelle zu abstrahieren (FF2).
- VH11: Implementierung einer Benutzungsschnittstelle, um Daten hochzuladen, Workflows zu starten und die Ergebnisse der Workflows anzuzeigen (FF3).
- VH12: Modellierung der Event Driven Architecture (FF3).
- VH13: Integration von Apache Airflow in die Architektur, um Workflows auszuführen (FF3).
- VH14: Implementierung von Workflows in Apache Airflow (FF3).
- VH15: Implementierung eines Microservices, welcher den Report am Ende eines Analyseworkflows erstellt (FF3).
- VH16: Aufbau der gesamten Komponenten innerhalb eines Kubernetes Clusters (FF4).

Innerhalb des Modellierungskapitels (Kapitel 3) sind die gewonnenen Erkenntnisse aus Kapitel 2 zu einem verteilten System zur Analyse von Genomdaten zusammengeflossen. Dabei wurde ein verteiltes System designt, welches über eine Event-Driven-Architektur modular mit Hilfe von Microservices aufgebaut wurde. Für alle vorgestellten Komponenten des IT-Systems von Kapitel 3 wurde eine Logik zur Integration in die Event-Driven Architecture vorgestellt oder Use-Cases skizziert. Die daraus resultierende Architektur heißt „Genomic Explore Framework“. Aus Kapitel 3 sind 13 Modelle entstanden:

- M1: Use-Context Diagramm GEF (FF1-4, löst VH11)
- M2: Komponentendiagramm GEF (FF1-4, löst VH12)
- M3: Architekturdiagramm Apache Spark Integration (FF1, löst VH1)
- M4: Architekturdiagramm QIIME2 Wrapper (FF1, löst VH3)
- M5: Architekturdiagramm BLAST Wrapper (FF1, löst VH5)
- M6: Architekturdiagramm ASaiM Wrapper (FF1, löst VH7)
- M7: Architekturdiagramm CEPH Integration (FF2, löst VH9)
- M8: Use-Cases Storage Microservice (FF2, löst VH10)
- M9: Aktivitätsdiagramm Benutzungsschnittstelle (FF3, löst VH11)
- M10: Aktivitätsdiagramm Beispielworkflows (FF3, löst VH13)
- M11: Aktivitätsdiagramm Events (FF3, löst VH14)
- M12: Aktivitätsdiagramm Reporting Microservices (FF3, löst VH15)
- M13: Deployment-Plan GEF in Kubernetes (FF4, löst VH16)

Kapitel 4 beschäftigt sich mit der Implementierung des Prototyps „GenomicInsights“, welcher Teile des „Genomic Explore Frameworks“ implementiert (M5, M8, M9, M10; M11 und M12). Innerhalb von GenomicInsights wurden aufgrund der Komplexität nicht alle der im Kapitel 3 definierten Analysemethoden umgesetzt. Kapitel 4 sollte die grundsätzliche Umsetzbarkeit des „Genomic Explore Frameworks“ zeigen. Dort entstanden neun Prototypen:

P1: Wrapper BLAST Analyse Microservice (FF1, VH5, M6)

P2: Wrapper GC-Content Analyse Microservice (FF1)

P3: Konfiguration des Speichers (FF2)

P4: Storage Microservice (FF2, VH10, M8)

P5: Benutzungsschnittstelle (FF3, VH11, M9)

P6: Event Streaming Platform (FF3, VH15, M10 und M11)

P7: Workflow Management Platform und Implementierung der Workflows aus P1 und P2 (FF3, VH13 und VH14, M10 und M11)

P8: Report Microservice (FF3, VH15, M12)

P9: Deployment GenomicInsights innerhalb von Docker (FF4)

Aufbauend auf GenomicInsights wurden in Kapitel 5 Experimente mit diesem durchgeführt, um z. B. die Umsetzung der definierten Use-Cases aus Kapitel 3 zu überprüfen. Die Experimente haben ergeben, dass die Umsetzung des „Genomic Explore Frameworks“ innerhalb von „GenomicInsights“ erfolgreich war. Aus der Evaluierung sind Arbeiten für die Zukunft identifiziert worden, welche im Ausblick aufgelistet sind.

6.2. Beantwortung der Forschungsfragen

FF 1: Welche Open-Source Big-Data Softwarelösungen eignen sich zur Analyse von Genomdaten?

Unterschiedliche Analysemethoden für die Analyse von Genomdaten wurden vorgestellt und in das verteilte System als Microservice integriert. Diese sind in den Abbildungen 16-19 zu sehen.

FF 2: Wie können Genomdaten persistiert werden?

Unterschiedliche Möglichkeiten zur Speicherung von Genomdaten wurden vorgestellt und aufgrund der Charaktereigenschaften dieser Daten kommt ein verteiltes Dateisystem (CEPH) zum Einsatz. Die Architektur ist in der Abbildung 20 zu sehen.

FF 3: Wie kann ein Frontend zur Interaktion zwischen Benutzer und verteiltem System, sowie die Kommunikation zwischen den Komponenten implementiert werden?

Für das Frontend wurden neben der Technologie zur Umsetzung die Use-Cases für die Interaktion zwischen Anwender und IT-System definiert. Weiterhin ist zentraler Bestandteil der Kommunikation zwischen den Analysekomponenten und der Reporterstellung eine Event-Driven-Architektur. Diese baut auf Apache Kafka auf. Analyseworkflows können über Apache Airflow definiert und gestartet werden. Für die Reporterstellung wurde ähnlich zu den Analysemethoden ein Microservices designt. Die Ergebnisse sind in den Abbildungen 21-24 zu sehen.

FF 4: Wie kann eine Architektur zur Interaktion, Speicherung und Analyse von Genomdaten effizient deployed werden?

Für ein Deployment der gesamten Softwarelösung wurden Docker und Kubernetes vorgestellt und aufgrund der Vorteile wurde Kubernetes ausgewählt. Für alle Komponenten wurde eine Zielarchitektur mit Kubernetes Objekten in der Abbildung 25 erstellt.

6.3. Ausblick

Für die Analysekomponenten, welche schon in das GEF integriert worden sind, müssen die verbleibenden Komponenten implementiert (ZA1-3) und die passenden Algorithmen ausgewählt oder erstellt werden (ZA4-7). Weiterhin kann CEPH als Speicherlösung implementiert werden (ZA8). Für die implementierte Benutzungsschnittstelle können die identifizierten Verbesserungen des Cognitive Walkthrougs umgesetzt werden (ZA9). Weiterhin können komplexere Workflows definiert und ausgeführt (ZA10) und die Reports mit Hilfe von Interviews an die Bedürfnisse des Labors angepasst werden (ZA11). Die gesamte Architektur kann ebenfalls auf Kubernetes deployed werden (ZA12). Eine regulatorische Begutachtung, welche Anforderungen es bei der Verarbeitung solcher Daten gibt, kann ebenfalls durchgeführt und die Architektur auf diese angepasst werden (ZA13).

- ZA1: Implementierung Microservice Wrapper Apache Spark (M3)
- ZA2: Implementierung Microservice Wrapper QIIME2 (M4)
- ZA3: Implementierung Microservice Wrapper ASaIM (M6)
- ZA4: Implementierung von Algorithmen zur Analyse von Genomdaten auf Basis von Apache Spark (VH2).
- ZA5: Auswahl der Algorithmen von QIIME2 (VH4).
- ZA6: Auswahl der Analysemethoden von BLAST sowie Datenbanken, mit denen die Sequenzen verglichen werden (VH6).
- ZA7: Auswahl der Analysemethoden von ASaIM (VH8).
- ZA8: Implementierung von CEPH als Speicherlösung (M7)
- ZA9: Erweiterung der Benutzungsschnittstelle
- ZA10: Implementierung komplexerer Workflows
- ZA11: Anpassung der Reports an die Bedürfnisse des Labors
- ZA12: Implementierung der Architektur auf Basis von Kubernetes (M13)
- ZA13: Identifizierung von regulatorischen Anforderungen zur Verarbeitung von metagenomischen Daten

Literaturverzeichnis

Wenn kein Autor oder Erstellungsdatum innerhalb der Quelle aufgeführt worden ist, wird dies innerhalb des Literaturverzeichnisses als o.A. (ohne Autor) und o.J. (ohne Jahr) gekennzeichnet.

- [1] S. Pawel, O. ChuangKee, L. M. H. T., P. Y. Ming, K. A. M. und O. H. San, „Advancing Personalized Medicine Through the Application of Whole Exome Sequencing and Big Data Analytics,“ *Frontiers in Genetics*, Bd. 10, p. 49, 2019.
- [2] K. Y. He, D. Ge und M. M. He, „Big Data Analytics for Genomic Medicine,“ *International Journal of Molecular Sciences*, Bd. 18, Nr. 2, 2017.
- [3] A. A. Alizadeh, D. T. Ross, C. M. Perou und R. v. d. Matt, „Towards a novel classification of human malignancies based on gene expression patterns,“ *The Journal of Pathology*, Bd. 195 (1), pp. 41-52, 2001.
- [4] J. A. Gilbert, R. A. Quinn, J. Debelius, Z. Z. Xu, J. Morton, N. Garg, J. K. Jansson, P. C. Dorrestein und R. Knight, „Microbiome-wide association studies link dynamic microbial consortia to disease,“ *Nature*, Nr. 535, pp. 94-103, 2016.
- [5] T. Krause, E. Jolkver, S. Bruchhaus, M. Kramer und M. Hemmje, *GenDAI - AI-Assisted Laboratory Diagnostics for Genomic Applications*, 2021.
- [6] E. Jolkver, Verarbeitung von RT-qPCR Daten in der Labordiagnostik, Fernuniversität in Hagen: Fakultät für Mathematik und Informatik, 2022.
- [7] J. F. Nunamaker, M. Chen und T. D. Purdin, „Systems Development in Information Systems Research,“ *Journal of Management Information Systems*, Nr. 7, pp. 89-106, 1990.
- [8] S. Fosso Wamba, S. Akter, A. Edwards, G. Chopin und D. Gnanzou, „How ‘big data’ can make big impact: Findings from a systematic review and a longitudinal case study,“ *International Journal of Production Economics*, pp. 4-6, 1 Januar 2014.
- [9] P. Sewal und H. Singh, „A Critical Analysis of Apache Hadoop and Spark for Big Data Processing,“ *2021 6th International Conference on Signal Processing, Computing and Control (ISPCC)*, pp. 308-313, 2021.
- [10] S. Sonkar, „Data Engineering for Beginners – Get Acquainted with the Spark Architecture,“ 06 November 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/11/data-engineering-for-beginners-get-acquainted-with-the-spark-architecture/#h2_7. [Zugriff am 05 August 2022].
- [11] S. Hussain, „Understanding the working of Spark Driver and Executor,“ 27 Dezember 2019. [Online]. Available: <https://blog.knoldus.com/understanding-the-working-of-spark-driver-and-executor/>. [Zugriff am 05 August 2022].
- [12] J. L. Sonnenburg und F. Bäckhed, „Diet–microbiota interactions as moderators of human metabolism,“ *Nature*, Bd. 535, pp. 56-64, 06 07 2016.
- [13] o. A., „RDD Programming Guide,“ o. J.. [Online]. Available: <https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds>. [Zugriff am 08 August 2022].

- [14] Y. Li, „Performance Analysis of Scheduling Algorithms in Apache Hadoop,“ in *2020 16th International Conference on Computational Intelligence and Security (CIS)*, 2020, pp. 149-154.
- [15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. a. M. M. Ma, M. J. Franklin, S. Shenker und I. Stoica, „Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing,“ *In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12)*, pp. 2-2, April 2012.
- [16] L. Wuttke, „Einführung in Apache Spark: Komponenten, Vorteile und Anwendungsbereiche,“ o. J.. [Online]. Available: <https://datasolut.com/was-ist-spark/>. [Zugriff am 08 August 2022].
- [17] C. K. Leung, O. A. Sarumi und C. Y. Zhang, „Predictive Analytics on Genomic Data with High-Performance Computing,“ *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 2187-2194, 2020.
- [18] W. Zhou, R. Li, S. Yuan, C. Liu, S. Yao, J. Luo und B. Niu, „MetaSpark: a spark-based distributed processing tool to recruit metagenomic reads to reference genomes,“ *Bioinformatics*, Bd. 33, Nr. 7, pp. 1090-1092, 2017.
- [19] V. Kaplarevic, „Apache Hadoop Architecture Explained (with Diagrams),“ 25 Mai 2020. [Online]. Available: <https://phoenixnap.com/kb/apache-hadoop-architecture-explained>. [Zugriff am 09 August 2022].
- [20] Microsoft, „Apache Hadoop-Architektur in HDInsight,“ 21 Juni 2022. [Online]. Available: <https://docs.microsoft.com/de-de/azure/hdinsight/hdinsight-hadoop-architecture>. [Zugriff am 08 August 2022].
- [21] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed und E. Baldeschwieler, „Apache Hadoop YARN: Yet Another Resource Negotiator,“ *SOCC '13: Proceedings of the 4th annual Symposium on Cloud Computing*, pp. 1-16, Oktober 2013.
- [22] H. Lin, Z. Su, X. Meng, X. Jin, Z. Wang, W. Han, H. An, M. Chi und Z. Wu, „Combining Hadoop with MPI to Solve Metagenomics Problems that are both Data- and Compute-intensive,“ *International Journal of Parallel Programming*, Bd. 46, Nr. 4, pp. 762-775, 2018.
- [23] H. Nordberg, K. Bhatia, K. Wang und Z. Wang, „BioPig: a Hadoop-based analytic toolkit for large-scale sequence data,“ *Bioinformatics*, Bd. 29, Nr. 23, pp. 3014-3019, 2013.
- [24] QIIME2, „What is QIIME2?,“ 2021. [Online]. Available: <https://docs.qiime2.org/2022.8/about/>. [Zugriff am 13 August 2022].
- [25] o.A., „PyBlast,“ o. J.. [Online]. Available: <https://github.com/jsgounot/PyBlast>. [Zugriff am 13 August 2022].
- [26] o.A., „rBLAST - Interface for BLAST search (R-Package),“ o.J.. [Online]. Available: <https://github.com/mhahsler/rBLAST>. [Zugriff am 13 August 2022].
- [27] N. H. e. Bergman, Comparative Genomics: Volumes 1 and 2, Totwa (NJ): Humana Press, 2007.

- [28] B. Batut, K. Gravouil, C. Defois, S. Hiltemann, J.-F. Brugère, E. Peyretaillade und P. Peyret, „ASaiM: A Galaxy-based framework to analyze microbiota data,“ *GigaScience*, Nr. 7, 22 Mai 2018.
- [29] Ghent University, „FASTA file format,“ 2015. [Online]. Available: <http://bioinformatics.intec.ugent.be/MotifSuite/fastformat.php>. [Zugriff am 10 August 2022].
- [30] Illumina, „FASTQ files explained,“ 26 Oktober 2021. [Online]. Available: <https://emea.support.illumina.com/bulletins/2016/04/fastq-files-explained.html>. [Zugriff am 10 August 2022].
- [31] HMP, „HMP Portal,“ o.J.. [Online]. Available: <https://portal.hmpdacc.org>. [Zugriff am 10 August 2022].
- [32] M. Orlov, „Why Storing Files in the Database Is Considered Bad Practice,“ o.J.. [Online]. Available: <https://maximorlov.com/why-storing-files-database-bad-practice/>. [Zugriff am 11 August 2022].
- [33] SAMBA, „SAMBA Wiki,“ 27 Juli 2022. [Online]. Available: https://wiki.samba.org/index.php/Main_Page. [Zugriff am 11 August 2022].
- [34] C. Hertel, „Samba: An Introduction,“ 27 November 2001. [Online]. Available: <https://www.samba.org/samba/docs/SambaIntro.html>. [Zugriff am 11 August 2022].
- [35] Ceph, „Intro to Ceph,“ o.J.. [Online]. Available: <https://docs.ceph.com/en/latest/start/intro/>. [Zugriff am 12 August 2022].
- [36] Ceph, „BLOCK DEVICES AND KUBERNETES,“ o.J.. [Online]. Available: <https://docs.ceph.com/en/latest/rbd/rbd-kubernetes/>. [Zugriff am 12 August 2022].
- [37] Ceph, „Ceph Documentation,“ o.J.. [Online]. Available: <https://docs.ceph.com/en/latest/>. [Zugriff am 12 August 2022].
- [38] D. Borthakur, „HDFS Architecture Guide,“ 18 Mai 2022. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Introduction. [Zugriff am 11 August 2022].
- [39] Mozilla Developer Network, „MVC,“ 18 Februar 2022. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Glossary/MVC>. [Zugriff am 15 August 2022].
- [40] Vue.js, „Vue.js Getting Started,“ o.J.. [Online]. Available: <https://012.vuejs.org/guide/>. [Zugriff am 15 August 2022].
- [41] L. Gupta, „REST Architectural Constraints,“ 09 März 2022. [Online]. Available: <https://restfulapi.net/rest-architectural-constraints/>. [Zugriff am 15 August 2022].
- [42] Red Hat, „Was ist EDA (Event Driven Architecture)?,“ 19 September 2019. [Online]. Available: <https://www.redhat.com/de/topics/integration/what-is-event-driven-architecture#funktionsweise>. [Zugriff am 14 August 2022].
- [43] M. Fowler, „Event Sourcing,“ 12 Dezember 2005. [Online]. Available: <https://martinfowler.com/eaaDev/EventSourcing.html>. [Zugriff am 14 August 2022].
- [44] Apache Kafka, „Apache Kafka,“ 2022. [Online]. Available: <https://kafka.apache.org/>. [Zugriff am 15 August 2022].

- [45] P. Jamshidi, C. Pahl, N. Mendonça, J. Lewis und S. Tilkov, „Microservices: The Journey So Far and Challenges Ahead,“ *IEEE Software*, Bd. 35, pp. 24-35, 2018.
- [46] Apache Software Foundation, „Apache Airflow,“ 2022. [Online]. Available: <https://airflow.apache.org>. [Zugriff am 15 August 2022].
- [47] IBM Cloud Education, „Containers,“ 23 Juni 2021. [Online]. Available: <https://www.ibm.com/cloud/learn/containers>. [Zugriff am 17 August 2022].
- [48] Docker, „Use containers to Build, Share and Run your applications,“ o.J.. [Online]. Available: <https://www.docker.com/resources/what-container/>. [Zugriff am 17 August 2022].
- [49] D. A. Norman und S. W. Draper, *User Centered System Design: New Perspectives on Human-computer Interaction*, 1986.
- [50] o.A., „User Centered Design,“ o.J.. [Online]. Available: <https://www.interaction-design.org/literature/topics/user-centered-design>. [Zugriff am 20 August 2022].
- [51] National Library of Medicine, „BLAST documentation,“ o.J.. [Online]. Available: https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs. [Zugriff am 20 Oktober 2022].
- [52] T. Avasthi , „Guide to Setting Up Apache Kafka Using Docker,“ 03 August 2022. [Online]. Available: <https://www.baeldung.com/ops/kafka-docker-setup>. [Zugriff am 05 Oktober 2022].
- [53] N. Narkhede, G. Shapira und T. Palino, *Kafka: The Definitive Guide*, O'Reilly Media, Inc., 2017.
- [54] Apache Foundation, „Running Airflow in Docker,“ 2022. [Online]. Available: <https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#docker-compose-env-variables>. [Zugriff am 05 Oktober 2022].
- [55] D. Almog und T. Heart, „What Is a Test Case? Revisiting the Software Test Case Concept,“ *Communications in Computer and Information Science*, Bd. 42, pp. 13-31, September 2009.
- [56] J. Rieman, M. Franzke und D. Redmiles, „Usability Evaluation with the Cognitive Walkthrough,“ in *Conference Companion on Human Factors in Computing Systems*, New York, Association for Computing Machinery, 1995, pp. 387-388.
- [57] National Library of Medicine, „NCBI BLAST: Extra Exercises Part 1: Identifying Sequences,“ o.J.. [Online]. Available: <https://guides.ncbi.nlm.nih.gov/tutorial/ncbi-blast-identify-and-compare-sequences-v2/single-page>. [Zugriff am 05 Oktober 2022].
- [58] o.A., „Testdatei SRS012969,“ o.J.. [Online]. Available: <https://portal.hmpdacc.org/files/596fc2de57601ec08a01fdee59b509b1>. [Zugriff am 17 Oktober 2022].
- [59] National Library of Medicine, „Index of /blast/db,“ o.J.. [Online]. Available: <https://ftp.ncbi.nlm.nih.gov/blast/db/>. [Zugriff am 20 Oktober 2022].

Anhang

Anhang 1: Projektplan der Arbeit

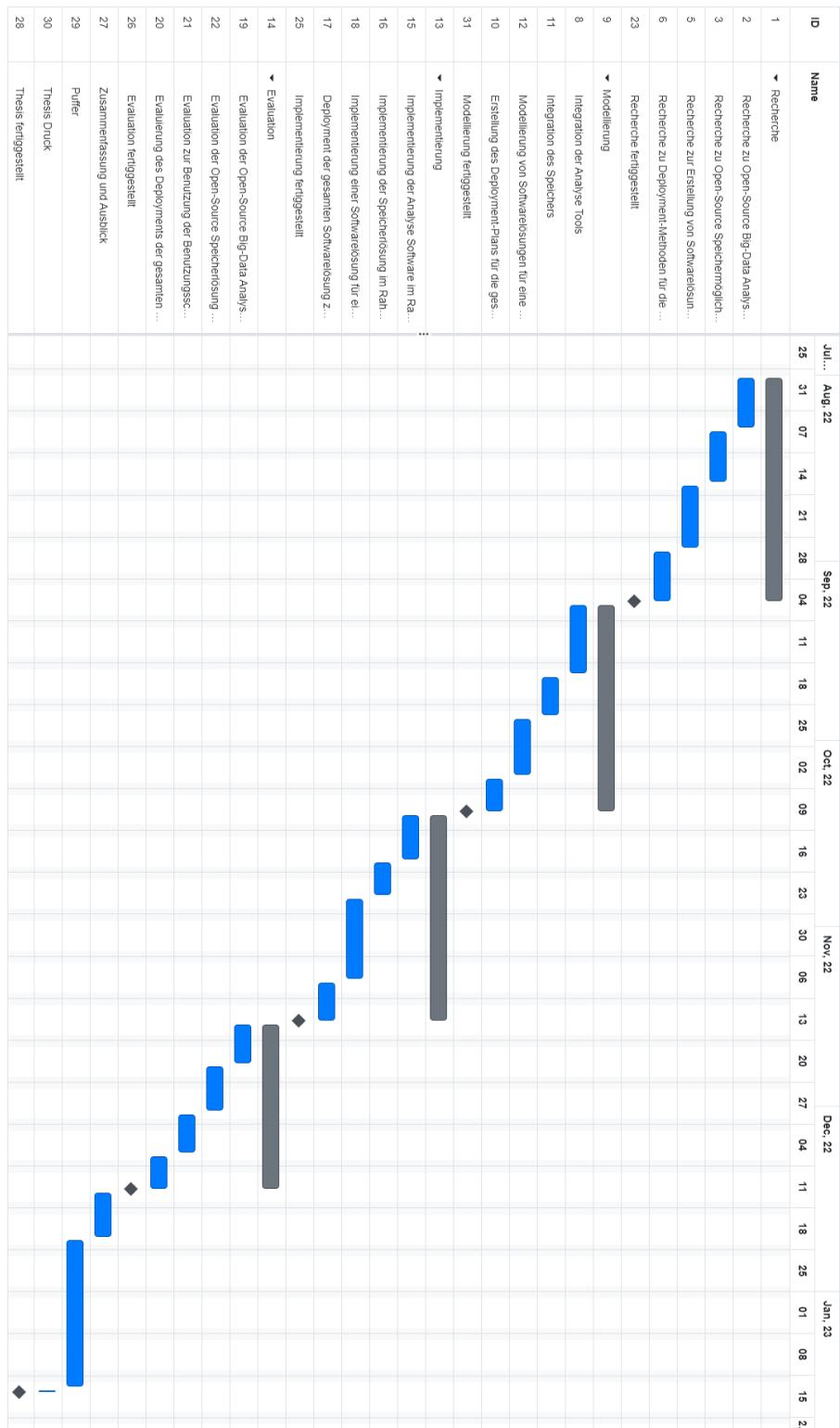


Abbildung 45: Projektplan der Arbeit

Anhang 2 Benutzerhandbuch GenomicInsights

Konfiguration der Softwarekomponenten

Alle Softwarekomponenten werden innerhalb des docker-compose Files konfiguriert. In diesen YML-Files können die Version und Speicherort der Daten festgelegt werden. Damit die Komponenten untereinander kommunizieren können, muss das gleiche Netzwerk konfiguriert sein.

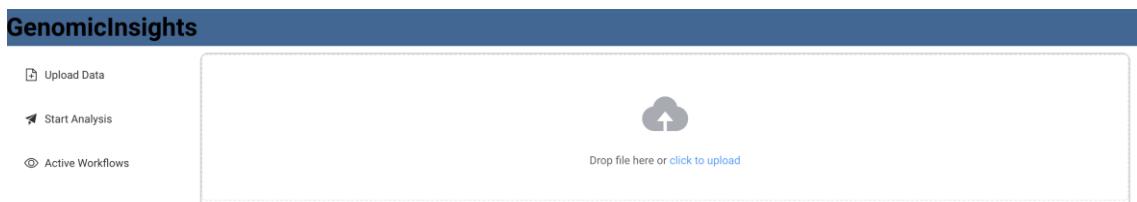
```
1  ---
2  version: '3'
3
4  services:
5    gc_microservice:
6      build: .
7      image: gc_microservice:latest
8      volumes:
9        - ../../data/raw/:/raw/
10       - ../../data/results/:/results/
11      restart: always
12
13
14  networks:
15    default:
16      name: distributed_system_network
17      external: true
18
```

Deployment / Start der Komponenten

Die Komponenten werden nicht nur durch die docker-compose Files konfiguriert, sondern die Container können ebenfalls gebaut und gestartet werden. Für einen Bau oder Start wird in das entsprechende Verzeichnis des docker-compose Files gewechselt und ein „docker-compose build“ oder „docker-compose up -d“ abgesetzt.

Hochladen von Daten

Um Daten innerhalb des Systems verfügbar zu machen, gibt es innerhalb der Benutzungsschnittstelle von GenomicInsights einen eigenen Menüpunkt („Upload Data“). Dahinter hat ein Benutzer einerseits die Möglichkeit mittels „Drag and Drop“ eine Datei hochzuladen und andererseits den Filebrowser zu nutzen. Im Hintergrund wird der Storage Microservice angesprochen und der Endpoint /upload genutzt.



Analyse von Daten

Über den Menüpunkt „Start Analyse“ kann ein Benutzer Analysen starten. Im oberen Bereich der Benutzungsschnittstelle sind die unterschiedlichen Schritte, welche durchzuführen sind, um eine Analyse zu starten, abgebildet. Im ersten Schritt muss ein Workflow ausgewählt werden. Hierbei wird beim Laden die Airflow API angesprochen und alle hinterlegten Workflows mit dazugehöriger Beschreibung angezeigt. Sind viele Workflows im System hinterlegt, kann das Finden eines spezifischen Workflows lange dauern. Aus diesem Grund ist eine Suchfunktion eingebaut, die es erlaubt, die Einträge zu filtern. Gefiltert wird nach dem Namen des Workflows.

The screenshot shows a user interface titled "GenomicInsights". At the top, there are three numbered steps: ① Select Workflow, ② Select File, and ③ Submit. Step ① is highlighted. On the left, there are buttons for "Upload Data", "Start Analysis", and "Active Workflows". The main area contains a search bar labeled "Search for Workflows..." and a list of two workflows:

Select	Name	Description
<input type="radio"/>	BLAST_NUKLEOTID	Analyse Nukleotidsequences with BLAST
<input type="radio"/>	GC_CONTENT	GC-Content calculation

At the bottom, there is a blue "Next step" button.

Der Benutzer muss einen Workflow auswählen, klickt dieser trotzdem auf den Button „Next step“, wird dieser durch einen roten Rahmen und einem Text darauf aufmerksam gemacht, einen Workflow auszuwählen.

The screenshot shows the same user interface as the previous one, but with a red border around the "Select Workflow" section and a red text message "Please select an entry!" above the "Next step" button.

Im nächsten Schritt werden, nach erfolgreicher Auswahl des Workflows, die verfügbaren Dateien angezeigt. Diese aufgelisteten Dateinamen wurden durch den Storage Microservice zurückgegeben. Sind viele Dateien im System hinterlegt, kann das Finden einer spezifischen Datei lange dauern. Aus diesem Grund ist eine Suchfunktion eingebaut, die es erlaubt, die Einträge zu filtern. Gefiltert wird nach dem Namen der Datei.

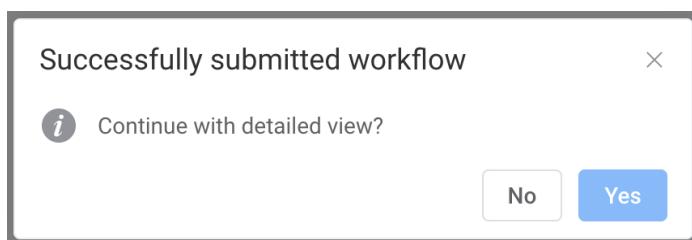
The screenshot shows the GenomicInsights interface. At the top, there's a blue header bar with the title 'GenomicInsights'. Below it is a navigation bar with links for 'Upload Data', 'Start Analysis', and 'Active Workflows'. The main area has three steps: 'Select Workflow' (with a green checkmark), 'Select File' (with a red circle containing a question mark), and 'Submit' (with a blue circle containing a question mark). Step 1 has a sub-section for 'Select Filename' with options: seq2.fasta (radio button), seq1.fasta (radio button, selected), and SRS012969.denovo_duplicates_marked.trimmed.1.fastq (radio button). A 'Next step' button is at the bottom.

Auch hier muss der Benutzer eine Datei auswählen. Klickt er trotzdem auf den Button „Next step“, wird der Benutzer durch einen roten Rahmen und einen Text darauf aufmerksam gemacht, eine Datei auszuwählen.

Im letzten Schritt bekommt der Benutzer die getätigten Eingaben (Workflow und Dateiname) nochmal angezeigt und kann daraufhin auf den Button „Submit“ drücken. Im Hintergrund wird über die Airflow API der ausgewählte Workflow gestartet und der entsprechende Dateiname übergeben.

The screenshot shows the GenomicInsights interface. The 'Select File' step is highlighted with a red box. A pop-up message in the center says: 'Are the following attributes correct? Workflow: BLAST_NUKLEOTID File: seq1.fasta'. At the bottom, there's a 'Submit' button.

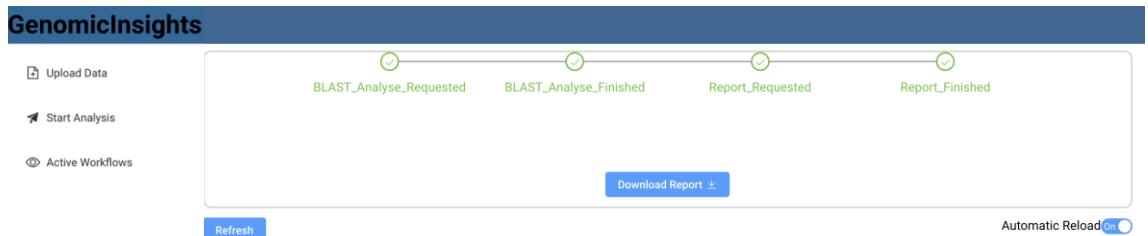
Es erscheint ein Pop-Up, welcher den Benutzer fragt, ob er in die detaillierte View wechseln möchte, um seinen Workflow zu verfolgen. Klickt dieser auf „Yes“, wird er weitergeleitet in eine andere View. Diese ist im Rahmen von GenomicInsights nur über diesen Schritt erreichbar.



In der detaillierten View ist der aktuelle Stand des Workflows (welche Steps sind schon durchlaufen und welcher Step wird gerade bearbeitet) zu sehen. Dies geschieht über eine Abfrage (alle fünf Sekunden) der Airflow API. Das automatische Abfragen kann der Benutzer über einen Schalter unten rechts aktivieren oder deaktivieren.

Download des Reports

Ist der Workflow Erfolgreich bis zum letzten Step durchgelaufen, kann der Benutzer den „Download Report“ Button drücken und erhält den Report als PDF. Im Hintergrund wird der Storage Microservice für den Download des Reports angesprochen.



Selbstständigkeitserklärung

Name: Mike Zickfeld

Matrikel-Nr.: 3146820

Fach: Masterstudiengang Wirtschaftsinformatik

Modul: Masterarbeit

Thema: Ein verteiltes System zur Verarbeitung genomicscher Daten

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

Datum: _____ Unterschrift: _____