Big Data und Big Data Analytics

Umsetzung einer Realtime-Analyse von Reddit-Nachrichten

Alwine Schultze & Kommilitone (HS)

Juni 2023

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung (HS)

Heutzutage sind soziale Medien zu einem integralen Bestandteil unseres digitalen Lebens geworden. Sie haben die Art und Weise, wie wir kommunizieren, Informationen beschaffen und Entscheidungen treffen, grundlegend verändert. Zum einen bieten sie eine einfache und schnelle Möglichkeit, mit anderen Menschen unabhängig von Zeit und Ort in Kontakt zu treten, zum anderen dienen sie als wichtige Informationsquelle für alle möglichen Themen und liefern stets neue Nachrichten aus der ganzen Welt. Des Weiteren bieten sie eine Plattform für öffentliche Diskussionen und sind gerade für Unternehmen zu einem wichtigen Instrument für Marketing und Werbung geworden.

Die Erfassung von Daten in Echtzeit aus den sozialen Medien bietet wertvolle Einblicke in das aktuelle Geschehen. Trends und wichtige Ereignisse können frühzeitig erkannt werden und ein Eindruck über die aktuelle Stimmung und Gemütslage kann gewonnen werden. Vor allem Unternehmen würden sehr davon profitieren, agil und schnell auf Veränderungen reagieren zu können und Strategien entsprechend anpassen zu können. Durch eine genaue Analyse der Stimmung der Menschen können Unternehmen personalisierte Angebote entwickeln und maßgeschneiderte Lösungen bieten und damit die Kundenzufriedenheit erhöhen. Weiterhin lässt sich durch eine Analyse der Aktivitäten und Strategien der Wettbewerber eine Steigerung der Wettbewerbsfähigkeit erzielen. Aber auch außerhalb von Unternehmen bietet die Echtzeitdatenextraktion wertvolle Vorteile: Einerseits geben gesellschaftliche Einblicke ein besseres Verständnis für Stimmungen, Meinungen und Bedürfnisse der Gesellschaft. Dies ist besonders relevant für Forscher, Journalisten und Regierungsbehörden und hilft vor allem in Krisenzeiten, frühzeitig Informationen und Warnsignale zu erkennen, um bei der Bewältigung von Krisensituationen zu unterstützen. Andererseits ermöglicht es Nutzern über aktuelle Ereignisse und Trends auf dem Laufenden zu bleiben und Verbrauchern auf aktuelle Bewertungen, Empfehlungen und Erfahrungen zuzugreifen.

Diese Arbeit stellt die Implementierung eines Data-Pipelinesystems vor, das speziell für die Echtzeitverarbeitung von Daten der Social-Media-Plattform Reddit entwickelt wurde.

1.2 Zielsetzung und Aufbau (HS)

Ziel dieser Arbeit ist es, ein System zu entwickeln, das in der Lage ist, neue Reddit-Posts in Echtzeit abzurufen, zu verarbeiten und zu analysieren. Der Fokus liegt dabei auf dem Datenabzug, dem Datenstreaming und der Datenhaltung. Dazu werden neue Posts auf Reddit in einem ersten Schritt über eine API-Schnittstelle mittels Python und der Bibliothek PRAW abgerufen. Darauf-

hin wird Apache Kafka verwendet, um einen kontinuierlichen Fluss der Daten zu gewährleisten. Apache Spark und Hadoop sorgen für die Weiterverarbeitung und Speicherung der Daten. Schließlich werden die Daten beispielhaft analysiert, unter anderem mit einer Sentimentanalyse und Topic Modeling. Dies soll verdeutlichen, welche Möglichkeiten sich durch die Echtzeitanalyse von Daten aus Reddit ergeben.

Nach der Einleitung geht es in Kapitel 2 um die Auswahl der verwendeten Tools sowie um deren Interaktion untereinander. Es wird detailliert geschildert, wozu die verschiedenen Tools verwendet werden und wie sie miteinander interagieren. In Kapitel 3 folgt dann die Umsetzung, in der genaue Installationsanweisungen für das Aufsetzen der Anwendung vorliegen. Dafür sind für alle Schritte Kommandozeilenbefehle angegeben, die ausgeführt werden müssen, um die Anwendung zu installieren. Da das Big-Data-Cluster und Datenstreaming dann eingerichtet ist und läuft, können in einem nächsten Schritt die Reddit Daten über die API abgezogen werden. Dies ist in Kapitel 3.2 geschildert. Zusätzlich wird in Kapitel 3.3 das Datenstreaming nach Hadoop noch einmal näher erläutert, wobei ein besonderes Augenmerk auf der Bereinigung der Daten liegt. Kapitel 4 behandelt dann schließlich die exemplarische Analyse und Auswertung der Daten. In einem ersten Schritt werden in einer explorativen Datenanalyse allgemeine Informationen über die abgerufenen Daten gewonnen. Es folgt eine Sentimentanalyse und ein Topic Modeling sowie die Nutzung eines Question-Answering-Systems. Das Fazit in Kapitel 5 rundet die Arbeit ab.

1.3 Wahl der Plattform (HS)

Die Online-Plattform Reddit ist ein soziales Netzwerk, das Benutzern ermöglicht, Inhalte zu teilen, Themen zu diskutieren und sich mit anderen Mitgliedern aus der ganzen Welt zu vernetzen. Reddit besteht aus sogenannten *Subreddits*, die Posts zu einzelnen Themenbereichen und Interessensgebieten sammeln. Ein Post kann ein Textbeitrag, ein Link, ein Bild, ein Video oder eine Kombination dieser Möglichkeiten sein und andere Benutzer können darauf in Form von Kommentaren reagieren und diskutieren. Dabei kann jeder Post und jeder Kommentar mittels Upvotes (positives Feedback) und Downvotes (negatives Feedback) bewertet werden. Anhand dieser Bewertungen werden Beiträge dann entsprechend gefiltert und sortiert, sodass beispielsweise höher bewertete Posts weiter oben im Subreddit angezeigt werden und damit die Sichtbarkeit der Posts erhöht wird. Reddit zeichnet sich durch seine vielfältige Benutzerbasis aus, die aus Menschen mit unterschiedlichen Hintergründen, Interessen und Standpunkten besteht und es den Benutzern möglich macht, Wissen, Meinungen und Erfahrungen zu nahezu jedem Thema auszutauschen.

Reddit bietet viele gute Gründe für die zum Ziel gesetzte Anwendung: Zum einen besitzt die Website eine umfangreiche Benutzerbasis, die eine breite Palette von Themen und Meinungen abdeckt und damit eine Fülle von Informationen und Standpunkten zu verschiedenen Themen bietet. Zu

nahezu jedem Thema gibt es aktive Communities, die sich austauschen, Informationen teilen und diskutieren. Es werden ständig neue Inhalte erstellt und durch die große und diverse Benutzerbasis lässt sich ein realistisches Bild der öffentlichen Meinung zu allen möglichen Themen abbilden. Zum anderen sind die Daten auf Reddit strukturiert und können leicht mit der frei verfügbaren und gut dokumentierten Reddit API abgerufen werden. Durch die Kombination all dieser Faktoren bietet Reddit eine reichhaltige Datenquelle mit der die öffentliche Meinung zu einem Thema in Echzeit erfasst und analysiert werden kann, um auf diese Weise wertvolle Erkenntnisse zu sammeln.

1.4 Hinweis zum Code (AS)

Im Zuge dieses Projekts wird Priorität auf Transparenz und Nachvollziehbarkeit gesetzt. Daher erfolgt eine Kennzeichnung des Codes gemäß seiner Entstehung. Dies impliziert, dass jede Sektion des Codes mit Kommentaren ausgestattet wird, welche sowohl den Entstehungsprozess als auch den Urheber des jeweiligen Abschnitts hervorhebt. Zudem wird großer Wert auf die Dokumentation der Herkunft von Code-Teilen gelegt, besonders bei der Verwendung von externen Images, Bibliotheken, Frameworks oder Code-Snippets. Diese Kennzeichnungen sind direkt im Code implementiert, was für eine sofortige Übersicht sorgt und eine eindeutige Zuordnung ermöglicht. Dieser Ansatz dient nicht nur dem Verständnis des Codes, sondern unterstützt auch die zukünftige Wartung und potenzielle Fehlerbehebung.

2 Analyse und Planung

2.1 Auswahl der Techstacks (AS)

Für die hier vorliegende Arbeit wurde sich für den Apache Big Data Tech Stack entschieden, da dieser in der Community weit verbreitet ist und die Tools der *Apache License, Version 2.0* unterliegen. Es handelt sich um eine permissive Open-Source-Lizenz, die die Nutzung, Verteilung und Modifikation der Software erlaubt, solange die Bedingungen der Lizenz eingehalten werden. Außerdem bietet der *Apache Big Data Tech Stack* vielfältige Funktionen und Vorteile für dieses Projekt. Die nachfolgende Auflistung hebt nochmals die Hauptgründe für diese Wahl hervor.

- Einer der Hauptgründe für die Auswahl des Apache Big Data Tech Stacks ist seine **Skalierbarkeit**. Apache Hadoop, das im Kern des Tech Stacks steht, ist darauf ausgelegt, mit großen Mengen von Daten umzugehen und kann problemlos auf Hunderte oder Tausende von Maschinen skaliert werden. Es bietet eine robuste und kosteneffiziente Speicherlösung für große Datenmengen.

- Ein weiterer Grund ist die hohe **Verarbeitungsgeschwindigkeit**. Apache Spark, ein weiterer Kernbestandteil des Tech Stacks, ermöglicht eine schnelle Datenverarbeitung sowohl für Batch- als auch für Streaming-Daten. Es kann die Geschwindigkeit und Effizienz der Datenverarbeitung erheblich verbessern, insbesondere wenn es darum geht, große Mengen an Reddit-Nachrichten zu analysieren und zu verarbeiten.
- Die Fähigkeit, Daten in Echtzeit zu verarbeiten, ist ein weiteres entscheidendes Merkmal des Apache Big Data Tech Stacks. Apache Kafka, eine populäre Open-Source-Plattform für Stream-Processing, ermöglicht die **Echtzeit-Verarbeitung** von Reddit-Nachrichten, wodurch die zeitnahe Auswertung dieser erst ermöglicht wird.
- Der Apache Big Data Tech Stack hat außerdem ein **reichhaltiges Ökosystem und einen starken Community-Support**. Es gibt zahlreiche Tools und Bibliotheken, die in dieses Ökosystem integriert werden können, um zusätzliche Funktionalitäten bereitzustellen. Darüber hinaus gewährleistet der umfangreiche Community-Support, dass Probleme schnell gelöst und neue Funktionen und Verbesserungen kontinuierlich entwickelt werden.

2.2 System-Architektur (AS)

Die Architektur des geplanten Systems wird in Abbildung 1 dargestellt, welche die geplante Interaktion der einzelnen Komponenten innerhalb des Tech Stacks illustriert. Im geplanten Ablauf des Datenflusses sind mehrere Schritte vorgesehen.

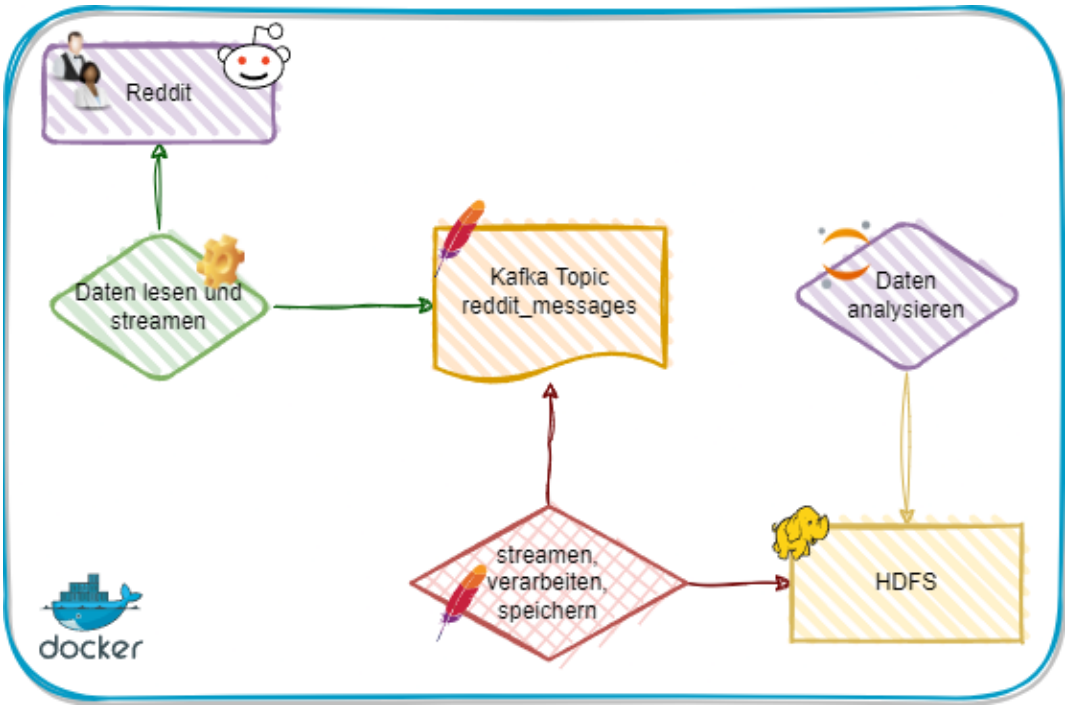

Abbildung 1: Architekturschaubild (eigene Darstellung)

In der Initialphase des vorgesehenen Prozesses ist geplant, Daten mittels der Reddit API zu extrahieren. Diese erfassten Daten werden daraufhin ununterbrochen in ein spezielles Kafka Topic gestreamt. In diesem Szenario dient Kafka als robustes und hochverfügbares System zur Konsolidierung von Streaming-Daten, um einen stetigen Informationsfluss von Reddit zu gewährleisten.

In der zweiten Phase wird Apache Spark genutzt, um auf das Kafka Topic zuzugreifen. Es nimmt die von Kafka gestreamten Daten auf, führt bei Bedarf erforderliche Anpassungen oder Transformationen aus und speichert sie anschließend in Hadoop. Apache Spark wird hier als mächtige Verarbeitungsebene fungieren, die die Effizienz bei dem Einlesen, Verarbeiten und Speichern der Daten in Hadoop erhöht. Mit diesen Daten als Grundlage sind Echtzeitanalysen der Reddit-Nachrichten möglich.

In der finalen Phase des Prozesses ist vorgesehen, die in Hadoop abgelegten Daten zur Untersuchung und Bewertung der Reddit-Posts zu verwenden. Ziel ist es, die gewonnenen Erkenntnisse zur Generierung wertvoller Insights einzusetzen.

3 Umsetzung

3.1 Big Data Cluster aufsetzen (AS)

3.1.1 Projektcode

Der gesamte Quellcode für dieses Projekt ist öffentlich zugänglich und auf GitHub unter dem folgenden Link zu finden: BigDataReddit. GitHub bietet eine ideale Plattform zur Unterstützung von Zusammenarbeit und fortlaufender Verbesserung in Softwareentwicklungsprojekten. Es ist wichtig zu beachten, dass für diese Arbeit keine separate Installation einer Entwicklungsumgebung erforderlich ist. Die vollständige Konfiguration und Implementierung des Clusters erfolgt innerhalb der Docker-Umgebung, was den Aufbau und die Inbetriebnahme erheblich vereinfacht.

3.1.2 Installation des Technologie-Stacks

Für die Einrichtung der technischen Infrastruktur in diesem Projekt wird auf die Ressourcen der Docker-Technologie zurückgegriffen. Docker, eine Open-Source-Plattform, hat die Landschaft der Softwareentwicklung und -bereitstellung durch die Einführung der Containerisierung revolutioniert. Die Plattform ermöglicht es, Anwendungen und ihre Abhängigkeiten in portable Container zu packen, was Konsistenz über unterschiedliche Betriebssysteme und Cloud-Plattformen hinweg sichert und zugleich Flexibilität und Skalierbarkeit fördert. Darüber hinaus unterstreicht die bereitgestellte Desktop App von Docker seine Benutzerfreundlichkeit, da sie mithilfe eines Installers aufgesetzt werden kann (siehe Docker Inc. 2023).

Docker nutzt seine fortschrittliche Technologie, um eine Vielzahl von Tools und Anwendungen mit minimalem Aufwand zu installieren und zu betreiben. Dies wird durch die Verwendung von

Docker Compose und maßgeschneiderten Konfigurationsdateien wie der *docker-compose.yml* erreicht. Diese Dateien enthalten alle notwendigen Details für die Installation und Konfiguration der jeweiligen Tools. Auch für dieses Projekt wurde eine *docker-compose.yml*-Datei verwendet (siehe Anhang A). In der vorliegenden Arbeit wurde das Docker-Image für Jupyter spezifisch angepasst, um die Installation der in der *requirements.txt* aufgeführten Abhängigkeiten zu ermöglichen. Zur Umsetzung dieser Anpassung wurde eine *Dockerfile* erstellt. Die Einzelheiten dieser Modifikation sind im Anhang B zu finden.

In der Systemkonfiguration wird auf eine Reihe von Docker-Images zurück gegriffen, die von der Community und Unternehmen bereitgestellt und auf Docker Hub öffentlich zugänglich gemacht werden. Jedes Image stellt eine speziell angepasste Umgebung für ein bestimmtes Tool oder eine Anwendung bereit, was die Systemeinrichtung vereinfacht und eine effiziente Konfiguration der erforderlichen Werkzeuge ermöglicht.

Einer der bemerkenswertesten Vorteile von Docker ist die Fähigkeit, mehrere Tools und Anwendungen in einem gemeinsamen Netzwerk auszuführen, was die Interaktion und Kommunikation zwischen diesen deutlich erleichtert. Ein solches Netzwerk kann einfach durch Ausführung des folgenden Befehls auf der Kommandozeile erstellt werden:

```
docker network create -d bridge bigdatanetwork
```

Nachdem das Netzwerk eingerichtet ist, kann die Installation der Tools innerhalb desselben initiiert werden. Man navigiert dazu in das Verzeichnis, das die *docker-compose.yml*-Datei enthält, und startet die Tools mit dem folgenden Befehl:

```
docker compose up -d
```

Mit diesem einzigen Befehl installiert und startet Docker alle in der *docker-compose.yml*-Datei spezifizierten Tools im zuvor erstellten Netzwerk. Das Flag *-d* sorgt dabei dafür, dass die Tools im Hintergrund laufen. Dank Docker kann das komplette Setup von komplexen, miteinander interagierenden Tools mit wenigen Schritten und geringen Kenntnissen über die jeweiligen Installationsspezifika durchgeführt werden.

Ob die einzelnen Docker-Container erfolgreich gestartet wurden, kann durch den Befehl *docker ps* überprüft werden (siehe Abbildung 2). Der Befehl *docker logs <Container-Name aus der Spalte NAMES>* ermöglicht die Einsicht in die einzelnen Logs der Container.

```
(DML93) C:\Users\Alwin\PycharmProjects\BigDataTwitter>docker ps
CONTAINER ID   IMAGE                                     COMMAND                  CREATED       STATUS                 PORTS                                                NAMES
b01155c61f58   jupyter/pyspark-notebook                  "tini -g -- start-no…"   24 hours ago  Up 24 hours (healthy)  4040/tcp, 0.0.0.0:8888->8888/tcp                     jupyter
967dfd732b98   bde2020/spark-worker:3.2.1-hadoop3.2      "/bin/bash /worker.sh"   24 hours ago  Up 24 hours            0.0.0.0:8081->8081/tcp                               spark-worker-1
3624965a56b8   bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8   "/entrypoint.sh /run…"   24 hours ago  Up 24 hours (healthy)  9864/tcp                                             datanode
763cc081b3b4   bde2020/spark-master:3.2.1-hadoop3.2      "/bin/bash /master.sh"   24 hours ago  Up 24 hours            0.0.0.0:7077->7077/tcp, 6066/tcp, 0.0.0.0:8080->8080/tcp   spark-master
cbd02cdd86d4   bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8   "/entrypoint.sh /run…"   24 hours ago  Up 24 hours (healthy)  0.0.0.0:9000->9000/tcp, 0.0.0.0:9870->9870/tcp       namenode
10ad17406aa   bitnami/kafka:3.4                         "/opt/bitnami/script…"   24 hours ago  Up 24 hours            0.0.0.0:9092->9092/tcp                               kafka-broker
```

Abbildung 2: Hochgefahrene Docker Container

3.1.3 Hadoop

Apache Hadoop ist ein Open-Source-Framework, das für die Verarbeitung und Speicherung großer Datensätze in verteilten Rechensystemen konzipiert wurde. Es basiert auf dem Algorithmus MapReduce von Google und dem verteilten Dateisystem von Google (GFS). Hadoop ermöglicht skalierbare und fehlertolerante Verarbeitung durch Aufteilung der Daten und Verarbeitungsaufgaben über ein Cluster von Computern. Hadoop wird oft in Big-Data-Anwendungen und in der Datenanalyse eingesetzt (vgl. Apache Software Foundation 2023a). In dem hier vorliegenden Projekt wird Hadoop hauptsächlich für die Datenhaltung verwendet.

Die für die Hadoop-Installation genutzten Docker-Images stammen von Big Data Europe (siehe Big Data Europe 2023a). Obwohl die Einrichtung bereits in einem früheren Schritt durchgeführt wurde, ist es von Bedeutung, die korrekte Funktionalität von Hadoop zu bestätigen. Diese Überprüfung lässt sich mithilfe des Health-Check Namenode durchführen, welcher durch Aufruf der entsprechenden Webseite eingesehen werden kann.

Im HDFS werden zusätzlich noch zwei Verzeichnisse für die Datensets benötigt, dies kann nach einer Verbindung zum Container direkt in *bash* erfolgen. Dafür die nachfolgenden Kommandozeilenbefehle nacheinander ausführen.

```
docker exec -it namenode bash
cd /opt/hadoop-3.2.1/bin
hadoop dfs -mkdir /reddits
hadoop dfs -mkdir /reddits_manipulated
```

Ob die Anlage des Verzeichnises erfolgreich war, kann ebenfalls im Browser überprüft werden: Browse Directory.

3.1.4 Kafka

Kafka, ein Open-Source-Stream-Verarbeitungssystem von der Apache Software Foundation, wird für das Daten-Streaming eingesetzt. Es ermöglicht die Echtzeitverarbeitung großer Datenmengen zwischen Anwendungen durch das Konzept der *Topics*. Kafka ist hoch skalierbar, fehlertolerant und kann riesige Datenströme mit niedriger Latenz handhaben, was es zu einem Schlüsselelement in Big-Data-Architekturen macht (vgl. Apache Software Foundation 2023b). Für die Installation bzw. das Aufsetzen von Kafka werden die Images von Bitnami verwendet (vgl. Bitnami by VMware 2023).

Kafka Topics sind grundlegende Bausteine der Apache Kafka-Plattform, diese ermöglichen die Kategorisierung von Datenströmen, ähnlich den Subreddits bei Reddit. Außerdem ermöglichen sie das Senden und Empfangen von Nachrichten zwischen Produzenten und Konsumenten. Dabei fungieren sie als unveränderliche, geordnete und partitionierte Protokolle, die eine effiziente und skalierbare Echtzeit-Datenverarbeitung ermöglichen.

```
docker exec kafka-broker \
    kafka-topics.sh --create \
    --topic reddit_messages \
    --bootstrap-server localhost:9092 \
    --replication-factor 1 \
    --partitions 1
```

Mit dem oben genannten Kommandozeilenbefehl wird ein Topic mit dem Namen **reddit_messages** erstellt. Dieser kann nun von einem sogenannten Producer befüllt werden. Ein Consumer kann sich dann aus diesem Topic bedienen und die Nachrichten für sich entsprechend weiterverarbeiten. Als Konsument der Daten wird Apache Spark verwendet.

3.1.5 Apache Spark

Apache Spark hat sich in der Echtzeitdatenverarbeitung durch seine Fähigkeit zur kontinuierlichen Verarbeitung und Analyse von Echtzeitdaten etabliert. Dank seiner Skalierbarkeit und Geschwindigkeit kann Spark große Mengen an Streaming-Daten effizient verarbeiten. Mit dem *Spark Streaming*-Modul können Daten aus unterschiedlichen Quellen gelesen werden. Seine MLlib-Bibliothek ermöglicht fortschrittliche Echtzeitanalysen, einschließlich maschinellem Lernen. Dadurch wird Apache Spark zu einer umfassenden Lösung für Echtzeitdatenverarbeitungsanforderungen (vgl. Apache Software Foundation 2023c).

Für die Installation wurde auch hier auf die Images von Big Data Europe zurück gegriffen (siehe Big Data Europe 2023b). Um die Funktionalität von Spark zu bestätigen, kann der Spark Master-Knoten aufgerufen werden. In der entsprechenden Ansicht ist auch der mit dem Spark-Master verbundene Spark-Worker zu sehen (siehe Abbildung 3).

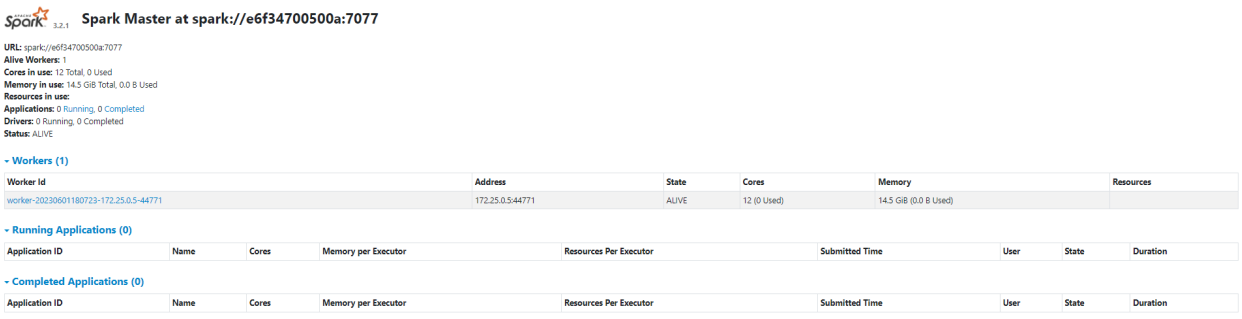

Abbildung 3: Spark Health-check

3.1.6 Jupyter Notebook

Jupyter Notebook ist eine Open-Source-Webanwendung, die es ermöglicht, Dokumente mit Live-Code, Visualisierungen und erklärendem Text zu erstellen. In diesem Projekt wird Jupyter No-

tebook genutzt, um Code direkt in dem aufgebauten Netzwerk zu entwickeln und auszuführen. Dank seiner interaktiven Umgebung können Code und Visualisierungen simultan entwickelt, ausgeführt und präsentiert werden. Dies erleichtert die Arbeit und fördert ein effizientes, interaktives Experimentieren und Debugging in Echtzeit.

```
docker logs jupyter
```

Das Jupyter Notebook kann über einen speziellen, mit einem Token versehenen Link aufgerufen werden. Dieser Link kann durch Aufrufen der Logs von Jupyter mit dem bereits genannten Befehl erhalten werden. Durch Anklicken des */lab?token*-Links öffnet sich Jupyter Notebook im Netzwerk des Tech Stacks, wodurch ein direkter Zugriff auf und Interaktion mit der Datenverarbeitungsumgebung ermöglicht wird. Das *work* Verzeichnis ist auf das *jupyter/data*-Verzeichnis des Projekts gemountet. Hier befinden sich bereits zwei Python Skripte *DataManipulation.py* und *RedditDataReader.py*.

3.2 Entwicklung der Reddit API Anbindung (HS)

Die Entwicklung der Reddit API Anbindung wird in Python durchgeführt. Für das Abrufen der Daten wird die Python-Bibliothek **PRAW** genutzt. PRAW steht für *Python Reddit API Wrapper* und ermöglicht es, Python-Anwendungen zu erstellen, die auf Reddit zugreifen können, Beiträge, Kommentare und viele weitere Informationen lesen können sowie weitere Aktionen auf der Plattform ausführen können. Der Vorteil liegt darin, dass PRAW eine einfache und intuitive Schnittstelle bietet, um auf die Reddit Daten zugreifen zu können, ohne sich um Details der HTTP-Anfragen, Authentifizierung, etc. kümmern zu müssen.

Um authentifizierten Zugriff auf die Reddit API zu bekommen, muss im Vorfeld eine eigene Applikation (Reddit App) über einen Benutzer auf der Reddit Website eingerichtet werden. Diese App muss einen eindeutigen Namen haben, da über diesen bei Zugriffen auf die App verwiesen wird. Weiterhin wird die Option *Script* verwendet, da diese Anwendung lediglich im Hintergrund läuft und weder im Browser läuft noch eine graphische Benutzeroberfläche benötigt. Neben dem Namen der App lässt sich optional noch eine Beschreibung und eine Weiterleitungs-URI (Redirect-URI) angeben. Mit der Erstellung der App wird eine Client-ID und ein Client-Geheimnis ausgegeben, welche dann benutzt werden können, um beispielsweise mittels PRAW auf die Reddit API zuzugreifen.

Nach Eingabe eines Subreddits (standardmäßig 'all' für das Abrufen von Posts aus allen Subreddits) werden mittels PRAW laufend neue Posts abgerufen und gespeichert. Folgende Informationen werden dabei pro Post abgerufen:

- *Titel:* Der Titel des Posts.
- *Zeitpunkt der Erstellung*

- *Subreddit:* Das Subreddit, in dem der Post verfasst wurde.
- *Inhalt:* Der Inhalt des Posts.

Jeder Post besitzt als einzigartige Kennzeichnung eine ID. Um Dopplungen beim Abrufen der Posts zu vermeiden, werden die IDs der abgerufenen Posts in einer Liste gespeichert. Bevor ein Post gespeichert wird, wird die ID des Posts mit den IDs in der Liste verglichen. Nur wenn sich die ID noch nicht in der Liste befindet, wird der Post gespeichert (siehe Anhang D).

3.3 Datenstreaming nach Hadoop (AS)

Die Python-Datei *DataManipulation.py* beinhaltet den Spark-Code, der zur Abholung der Daten aus Kafka, zur anschließenden Datenmanipulation und zur Speicherung der Daten in den Hadoop Datanode genutzt wird (siehe Anhang C). Dieses Skript ist kontinuierlich aktiv und liest den Kafka-Stream. Damit wird sichergestellt, dass die Daten in Realtime von Kafka abgeholt, verarbeitet und in hadoop zur weiteren Analyse bereit gestellt sind. Dabei werden zum einen die Daten in ihrer rohen Form in das Verzeichnis */reddits* geschrieben (vgl. Kapitel 3.1.3). Außerdem werden die Daten auch in Verarbeiteter Form in das Hadoop-Verzeichnis */reddits_manipulated* geschrieben. Dabei werden folgende Aktionen auf den *Titel* und den *Inhalt* der Reddits ausgeführt:

- **remove_whitespace** ist darauf ausgelegt, überflüssige Leerzeichen, Tabulatorzeichen und Zeilenumbrüche zu eliminieren. Durch die Entfernung dieser nichtdruckbaren Zeichen wird der Text gestrafft und auf seine wesentlichen Bestandteile reduziert. Dies erleichtert die Weiterverarbeitung und Analyse des Textes, indem es Klarheit schafft und mögliche Unordnung oder inkonsistente Formatierungen beseitigt.
- **remove_special_characters** dient dazu, Sonderzeichen aus den Texten zu entfernen. Der Verzicht auf solche Zeichen ist in vielen Fällen für die Weiterverarbeitung des Textes und die Durchführung von Textanalysen von Vorteil. Sie ermöglicht es, den Text auf seine wesentlichen sprachlichen Bestandteile zu reduzieren und potenzielle Störungen oder Verzerrungen, die durch nicht-alphabetische Zeichen verursacht werden könnten, zu beseitigen.
- **lower_case** konvertiert alle Großbuchstaben in den zugehörigen Kleinbuchstaben. Dadurch wird eine einheitliche Behandlung von Wörtern ermöglicht, unabhängig von ihrer ursprünglichen Schreibweise. Dies ist insbesondere wichtig für die effektive Erkennung von Stoppwörtern, da die Stoppwortliste in *nltk* ausschließlich Kleinbuchstaben enthält.
- **remove_links** identifiziert und entfernt alle Internet-Links innerhalb der Reddit-Posts. Durch die Entfernung der Links wird der Text auf das wesentliche sprachliche Material reduziert,

wodurch eine sauberere und konzentriertere Textanalyse ermöglicht wird. Dies ist insbesondere hilfreich, um irrelevante Informationen zu entfernen und den Fokus auf den tatsächlichen Inhalt und Kontext des Textes zu legen.

- **tokenize** teilt den gegebenen Text in einzelne Wörter (Token) auf. Durch die Tokenisierung wird der Text in handhabbare Einheiten zerlegt, die dann einzeln analysiert und verarbeitet werden können. Diese Art der Textzerlegung ist ein grundlegender und unverzichtbarer Schritt in vielen Bereichen der Textanalyse und des maschinellen Lernens. Sie ermöglicht es, die Struktur des Textes zu verstehen, relevante Informationen zu extrahieren und die Textdaten für weitere Verarbeitungsschritte, wie die Stoppwortentfernung oder die Feature-Extraktion, vorzubereiten.

- **remove_stopwords** zielt darauf ab, alle sogenannten Stoppwörter aus dem Text zu entfernen. Stoppwörter sind häufig vorkommende Wörter wie *und*, *der*, *ist*, die für die semantische Bedeutung eines Textes oft wenig aussagekräftig sind. Durch ihre Entfernung wird der Text auf die wesentlichen, informationsreichen Wörter reduziert, was eine effektivere und fokussiertere Textanalyse ermöglicht. Dies ist insbesondere wichtig in Bereichen wie Informationsrückgewinnung oder maschinelles Lernen, wo die Reduktion von Dimensionalität und Rauschen einen entscheidenden Vorteil darstellt.

Das Skript wird in einem Terminalfenster im JupyterLab Launcher mit dem Befehl *python3 work/DataManipulation.py* gestartet. Beim ersten Start des Skripts leitet Spark den Download der notwendigen Kafka-Komponenten ein.

In der Zwischenzeit kann das separate Skript zur Erfassung von Daten aus Reddit gestartet werden. Sobald Daten in Kafka eintreffen, verarbeitet das Spark-Skript diese sofort und speichert die verarbeiteten Daten in Hadoop.

3.4 Senden der Daten an das Kafka Topic (HS)

Auch dieses Skript liegt bereits im *work*-Verzeichnis von Jupyter. Dieses kann ähnlich dem Spark-Skript gestartet werden mit dem Befehl *python3 work/RedditDataReader.py*. Das Skript ruft, wie bereits in Kapitel 3.2 beschrieben, die Daten von Reddit ab und schreibt sie mittels eines Kafka Producers in das Topic *reddit_messages*.

4 Analyse und Auswertung der Daten

4.1 Explorative Datenanalyse (HS)

Die gesamte Analyse und Auswertung der Daten befindet sich in dem Jupyter Notebook *DataAnalysis.ipynb* (siehe Anhang E). Für die Analyse werden die gespeicherten Daten abgerufen und

in einem Pandas-Dataframe abgelegt. In der ersten Auswertung werden die Anzahl neuer Posts pro Subreddit abgerufen. Dies liefert eine guten Überblick über die beliebtesten und aktivsten Subreddits und die am meisten diskutierten Themenbereiche. Daneben lässt sich ein beliebiges Subreddit angeben, aus dem die neuesten Posts angezeigt werden. Zusätzlich wird eine Wordcloud erstellt, die sich über alle Subreddits erstreckt, um einen generellen Eindruck von aktuell diskutierten Themen zu bekommen. In der Analyse wird sowohl auf die Rohdaten in *reddits* als auch auf die bereinigten Daten in *reddits_manipulated* zugegriffen. Die Rohdaten eignen sich gut, um beispielsweise den Titel der Posts anzuzeigen, während die bereinigten Daten für Analysen wie die Wordcloud nötig sind, da anderenfalls das Ergebnis zum Beispiel durch Stoppwörter beeinflusst werden würde.

4.2 Sentimentanalyse (HS)

Für die Sentimentanalyse wird der *SentimentIntensityAnalyzer* aus dem NLTK-Paket verwendet. Damit wird dem Dataframe eine neue Spalte **Sentiment** hinzugefügt, in der die Bewertung der Stimmung aus dem Zusammenschluss von Titel und Inhalt eines Posts vorliegt. Beispielhaft werden die ersten 20 Ergebnisse inklusive des Sentiments ausgegeben. Zusätzlich wird das durchschnittliche Sentiment aller Posts angegeben, was einen Eindruck über die aktuelle Stimmung vermittelt. Außerdem wird eine Statistik mittels der *describe()*-Methode angegeben. Mit Hilfe des Sentiments kann nun analysiert werden, welche Subreddits eher positiv und welche eher negativ gestimmt sind. Dazu werden zuerst nur Subreddits gefiltert, die seit Beginn des Datenabzugs mehr als 10 neue Posts bekommen haben. Dadurch wird sichergestellt, dass nicht ein einziger Post, der sehr positiv oder sehr negativ ist, das Ergebnis verfälscht. Als Nächstes wird der Durchschnittswert des Sentiments pro Subreddit berechnet, um die Subreddits dementsprechend zu sortieren. Schließlich werden die 10 besten und die 10 schlechtesten Subreddits ausgegeben.

4.3 Topic Modeling (HS)

Für das Topic Modeling wird die Gensim-Bibliothek verwendet, um ein *Latent Dirichlet Allocation (LDA)-Modell* für die Themenmodellierung zu trainieren. Dazu werden separate Listen angelegt, die den Inhalt der Reddit-Posts enthalten. Dann wird ein Dictionary und ein Corpus erstellt, der die Texte in eine Bag-of-Words-Darstellung umwandelt. Schließlich wird das LDA-Modell mit dem Corpus und dem Dictionary trainiert und die Ergebnisse werden ausgegeben.

4.4 Question-Answering-System (HS)

Zum Schluss wird beispielhaft ein Question-Answering-System implementiert, welches Fragen zu den ausgelesenen Reddit-Posts beantworten soll. Hierfür werden die *Pipelines* aus der Transformers-

Bibliothek von Huggingface benutzt. Nach optionaler Wahl eines Subreddits werden Titel und Inhalt der Posts zusammengefügt, um beides als Kontext für das Question-Answering-System nutzen zu können. Nach Angabe der Task **question-answering** und eines Modells können dann Fragen gestellt werden, die das System beantwortet.

5 Fazit (AS & HS)

Diese Arbeit präsentiert die Implementierung eines Data-Pipelinesystems, das Daten in Echtzeit von Reddit abruft, diese streamt, verarbeitet und schlussendlich analysiert. Trotz des erfolgreichen Aufbaus und der Anwendung der Pipeline konnten im Laufe des Projekts mehrere Verbesserungspotenziale identifiziert werden, die in folgenden Abschnitten diskutiert werden.

Ein kritischer Punkt in der Architektur ist die Single-Node-Konfiguration von Kafka. Obwohl sie für dieses Projekt ausreichend war, bietet sie keine ausreichende Ausfallsicherheit für größere, produktive Umgebungen. Zukünftige Arbeiten könnten sich darauf konzentrieren, Kafka in einer Multi-Node-Konfiguration aufzusetzen, bei der mehrere Instanzen des Kafka-Brokers parallel laufen. Dies würde die Ausfallsicherheit des Systems signifikant erhöhen und seine Skalierbarkeit verbessern.

Ein weiterer Bereich, für den eine Überarbeitung sinnvoll erscheint, ist die Art und Weise, wie Spark von Kafka Daten erhält. Die aktuelle Methode kann zu Unterbrechungen führen, wenn der Spark-Thread nicht als **UninterruptibleThread** läuft. Eine mögliche Lösung könnte die Verwendung von asynchronem Code oder die Verarbeitung von Nachrichten in kleineren Chargen sein, um die Anfälligkeit des Systems für Unterbrechungen zu reduzieren.

Ebenfalls nicht ideal ist das Question-Answering-System im Bereich der Auswertung der Daten. Da die Reddit-Daten durch die vielen verschiedenen Themenbereiche und Formate (beispielsweise bestehen die Titel in bestimmten Subreddits wie *AskReddit* nur aus Fragen) sehr verschieden sind und das Question-Answering-System sehr konkrete Fragen braucht, waren die meisten erhaltenen Antworten nicht korrekt oder ergaben keinen Sinn. Obwohl die Implementierung durchaus ohne Probleme funktioniert, sind die Ergebnisse meistens nicht verwertbar. Eventuell wäre das System anwendbar auf bereits ausgewertete Daten zu einem konkreten Themengebiet, hätte dann jedoch nur einen sehr speziellen Nutzen.

Ungeachtet der identifizierten Bereiche für Verbesserungen, demonstriert das Projekt, wie ein überschaubarer Technologiestack effektiv genutzt werden kann, um ein funktionales und robustes System für Echtzeit-Datenverarbeitung und -analyse zu etablieren. Die vorliegende Arbeit legt damit einen stabilen Grundstein für künftige Projekte im Bereich der Echtzeitdatenverarbeitung und -analyse. Sie ebnet den Pfad für fortlaufende Optimierungen und Erweiterungen, beispielsweise in Form von Echtzeit-Datenvisualisierungen von Social Media Nachrichten.

Anhang

A Docker Konfiguration *docker-compose.yml*

```
# Autor: AS
# Dieses docker-compose.xml File wurde aus den von den jeweiligen Community-
# membern zur Verfügung gestellten docker-compose.yml Files zusammen gestellt
# und entsprechend um projektspezifische Konfigurationen erweitert.

version: "3"

services:
# Der Kafka-Broker speichert, empfängt und sendet Nachrichten, und stellt die
# robuste und effiziente Kommunikation zwischen Produzenten und Konsumenten sicher.
  kafka:
    image: docker.io/bitnami/kafka:3.4
    container_name: kafka-broker
    ports:
      - 9092:9092
    volumes:
      - "kafka_data:/bitnami"
    environment:
      - ALLOW_PLAINTEXT_LISTENER=yes

# Die Hadoop Namenode verwaltet die Dateisystem-Metadaten für HDFS, hält die
# Verzeichnisstruktur aller Dateien im System aufrecht und reguliert den Zugriff
# auf diese Dateien durch die Clients.
  namenode:
    image: bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8
    container_name: namenode
    restart: always
    ports:
      - 9870:9870
      - 9000:9000
    volumes:
      - hadoop_namenode:/hadoop/dfs/name
    environment:
```

```
      - CLUSTER_NAME=bigdata_cluster
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
    env_file:
      - ./hadoop/hadoop.env

# Die Hadoop Datanode speichert Daten in HDFS (Hadoop Distributed File System)
# und verarbeitet die Anfragen vom Namenode.
  datanode:
    image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
    container_name: datanode
    restart: always
    volumes:
      - hadoop_datanode:/hadoop/dfs/data
    environment:
      - CORE_CONF_fs_defaultFS=hdfs://namenode:9000
    env_file:
      - ./hadoop/hadoop.env
    depends_on:
      - namenode

# Spark Master Instanz, sie koordiniert und verteilt Aufgaben an die Spark
# Worker und ist für das Ressourcenmanagement des Clusters verantwortlich.
  spark-master:
    image: bde2020/spark-master:3.2.1-hadoop3.2
    container_name: spark-master
    ports:
      - "8080:8080"
      - "7077:7077"
    environment:
      - INIT_DAEMON_STEP=setup_spark
      - HADOOP_CONF_DIR=/etc/hadoop
      - SPARK_HOME=/spark
    depends_on:
      - namenode

# Spark Worker Instanz, sie führt die ihm vom Master zugewiesenen Aufgaben
# aus und verwaltet die Daten- und Rechenressourcen seiner lokalen Maschine.
```

```
  spark-worker-1:
    image: bde2020/spark-worker:3.2.1-hadoop3.2
    container_name: spark-worker-1
    depends_on:
      - spark-master
    ports:
      - "8081:8081"
    environment:
      - "SPARK_MASTER=spark://spark-master:7077"

# Jupyter Notebook als Analysetool für die Daten
  jupyter-notebook:
    build: .
    container_name: jupyter
    ports:
      - "8888:8888"
    depends_on:
      - spark-master
    volumes:
      - ./jupyter/data:/home/jovyan/work
    environment:
      - "SPARK_MASTER=spark://spark-master:7077"

volumes:
  hadoop_namenode:
  hadoop_datanode:
  kafka_data:
    driver: local

# Die Tools werden alle in dem Netzwerk "bigdatanetwork" betrieben,
# damit wird der gegenseitige Zugriff erleichtert.
networks:
  default:
    external:
      name: bigdatanetwork
```

B Dockerfile (Jupyter)

```
# Autor AS

# Basis-Image
FROM jupyter/pyspark-notebook

# Setze das Arbeitsverzeichnis im Container
WORKDIR /home/jovyan/work

# Kopiere alle Dateien im aktuellen Verzeichnis zum
# Arbeitsverzeichnis im Container
COPY . .

# Führe pip install durch, falls du eine requirements.txt hast
RUN pip install --no-cache-dir -r requirements.txt
```

C Python Code (Spark) *DataManipulation.py*

```
# Autor AS
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.types import StringType
from pyspark.sql.functions import udf

import datetime
import json
import re
import nltk
from nltk.corpus import stopwords

# Laden der Ressourcen
nltk.download('punkt')
nltk.download('stopwords')

# Konvertieren der Stoppwörter in eine Liste
stopWords = list(stopwords.words('english'))

# Konstanten
TOPIC = "reddit_messages"
APP_NAME = "KafkaDataFetch"
DATA_ATTRIBUTES_FOR_CLEANUP = ["Titel", "Inhalt"]

# Spark Session aufbauen (Netzwerk: local)
spark = SparkSession.builder \
    .master("local[*]") \
    .appName(APP_NAME) \
    .config("spark.jars.packages",
            "org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.1") \
    .getOrCreate()



# Datenmanipulationen vor dem Speichern
def data_manipulation(df, epoch_id):
    new_df = df
```

```
    # neues Dataframe mit den Werten erstellen und zurück liefern
    # für die Persistierung
    manipulate_text_udf = udf(manipulate_text, StringType())
    new_df = new_df.withColumn("value", manipulate_text_udf(col("value")))

    return new_df


# Textmanipulation für die Werte "Titel" und "Inhalt"
def manipulate_text(value):
    data = json.loads(value)

    for data_attr in DATA_ATTRIBUTES_FOR_CLEANUP:
        text = data[data_attr]
        text = remove_whitespace(text)
        text = remove_special_characters(text)
        text = lower_case(text)
        text = remove_links(text)
        text = tokenize(text)
        data[data_attr] = remove_stopwords(text)
    return json.dumps(data)


# Entfernen unnötiger Leerzeichen
def remove_whitespace(text):
    text = text.replace('\r', ' ')
    return text.replace('\n', ' ')


# Entfernen von Sonderzeichen
def remove_special_characters(text):
    return re.sub('[^a-zA-z0-9\s]', '', text)


# Entfernen der Links
def remove_links(text):
```

```
    pattern = r"http\S+|www\S+"
    return re.sub(pattern, "", text)


# Alles in Kleinschreibung umwandeln
def lower_case(text):
    return text.lower()


# ggf. tokenizen, aber nur wenn topic modelling etc. gemacht wird,
# nicht wenn Q&A bestehen bleibt
def tokenize(text):
    wordTokens = nltk.word_tokenize(text)
    return " ".join([token for token in wordTokens])


# Stoppwörter entferenn
def remove_stopwords(text):
    wordTokens = nltk.word_tokenize(text)
    return " ".join([word for word in wordTokens if word not in stopWords])


# Methode zum Schreiben der gelesenen Daten in die Hadoop-Verzeichnisse
# "redidits" und "reddits_manipulated"
def write_to_hadoop(df, epoch_id):
    df_raw = df
    df_mani = data_manipulation(df, epoch_id)

    # Datumsstring für heutiges Datum (Start Datum des Jobs)
    date_str = datetime.datetime.now().strftime("%Y-%m-%d")
    # Speicherorte für raw Data und manipulierte Daten
    fn_raw = f"hdfs://namenode:9000/reddits/data_{date_str}.csv"
    fn_mani = f"hdfs://namenode:9000/reddits_manipulated/data_{date_str}.csv"

    # Speichern der DataFrame als CSV in Hadoop.
    df_raw.write.csv(fn_raw, mode="append")
    df_mani.write.csv(fn_mani, mode="append")
```

```
if __name__ == "__main__":
    # Verbindung zum Kafka Topic für das Lesen der Daten:
    df = spark \
        .readStream \
        .format("kafka") \
        .option("kafkaConsumer.pollTimeoutMs", 60*60*1000) \
        .option("kafka.bootstrap.servers", "kafka-broker:9092") \
        .option("startingOffsets", "earliest") \
        .option("subscribe", TOPIC) \
        .load()

    # Konvertierung der Ausagabe aus Kafka in einen String
    df = df.selectExpr("CAST(value AS STRING)")
    # Verarbeitungsquery aufbauen und für jeden eingelesenen "batch"
    # Kafka-Messages die Methode "write_to_hadoop" aufrufen
    query = df.writeStream.foreachBatch(write_to_hadoop).start()
    # Solange auf Kafka hören, bis der Abbruch kommt.
    query.awaitTermination()
```

D Python Code *RedditDataReader.py*

```
# Autor: HS

# use praw library to access Reddit API
import praw
import datetime
import json
from confluent_kafka import Producer
import time

p = Producer({'bootstrap.servers': 'kafka-broker:9092'})

# set access id and secret to reddit app
reddit = praw.Reddit(
    client_id='QK6TtiVAAfZ5XVp3lNHQLw',
    client_secret='Z9fjb3B87YDkZHi8TXtBC4GipN6bcw',
    user_agent='akad_bigdata',
    redirect_uri='http://www.example.com/unused/redirect/uri'
)

# choose subreddit ('all' for all subreddits)
subreddit = reddit.subreddit('all')

# get posts
processed_ids = []
processed = False

while True:
    posts = list(subreddit.new())

    for submission in posts:
        if submission.id not in processed_ids:
            print(submission.title)
            post_dict = {
                'Titel': submission.title,
                'Erstellt_UTC': submission.created_utc,
                'Erstellt': datetime.datetime.fromtimestamp(submission.created_utc).is
```

```
            'Subreddit': submission.subreddit.display_name,
            'Inhalt': submission.selftext
        }
        processed_ids.append(submission.id)
        processed = True
    else:
        processed = False

    byte_like = json.dumps(post_dict).encode('utf-8')
    p.produce('reddit_messages', byte_like)

#only flush if new posts were submitted
if processed:
    p.flush()
```

E Analyse und Auswertung *DataAnalysis.ipynb*

```
# # Data Analysis
# Autor: HS

# ### Read the data

from pyspark.sql import SparkSession

# create sparkSession
spark = SparkSession.builder.appName("HadoopReadExample").getOrCreate()

# path to the data
path = "hdfs://namenode:9000/reddits/"
path_manip = "hdfs://namenode:9000/reddits_manipulated/"

# read data
dframe = spark.read.csv(path + "*.csv", inferSchema=True, header=True)
dframe_manip = spark.read.csv(path_manip + "*.csv", inferSchema=True, header=True)

rows = dframe.collect()
rows_manip = dframe_manip.collect()



# ### Create a dataframe with title, content and subreddit as columns

# original data
import json
import pandas as pd

data_rows = []

# extract data for title, content and subreddit
for row in rows:
    data = json.loads(row[0])
    title = data["Titel"]
    content = data["Inhalt"]
    subreddit = data["Subreddit"]
```

```
    data_rows.append({"title": title, "content": content, "subreddit": subreddit})

#add data to dataframe
df = pd.DataFrame(data_rows)

#drop duplicates
df = df.drop_duplicates()

# print dataframe
print(df.head(10))



# manipulated data
import json
import pandas as pd

data_rows = []

# extract data for title, content and subreddit
for row in rows_manip:
    data = json.loads(row[0])
    title = data["Titel"]
    content = data["Inhalt"]
    subreddit = data["Subreddit"]

    data_rows.append({"title": title, "content": content, "subreddit": subreddit})

#add data to dataframe
df_manip = pd.DataFrame(data_rows)

#drop duplicates
df_manip = df_manip.drop_duplicates()

# print dataframe
print(df_manip.head(10))
```

```
# ### Show most active subreddits

#show most active subreddits
df['subreddit'].value_counts().head(20)


# ### Choose a subreddit and show posts

#choose subreddit
subreddit = "sportsnewstoday"

#show new posts
df[df["subreddit"]==subreddit]


# ### Wordcloud from chosen subreddit


from wordcloud import WordCloud
import matplotlib.pyplot as plt

#choose subreddit or take all subreddits
subdf = df_manip[df_manip["subreddit"]==subreddit] # specific subreddit
#subdf = df_manip # all subreddits

# combine text from title and content
text = ' '.join(subdf['title']) + ' ' + ' '.join(subdf['content'])

# create wordcloud
wordcloud = WordCloud(width=800, height=400).generate(text)

# show wordcloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

```
# ### Sentiment Analysis


import pandas as pd
import nltk
nltk.download('vader_lexicon')  # Lade den Sentiment-Analyse-Lexikon aus NLTK

from nltk.sentiment import SentimentIntensityAnalyzer

# create SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

# load dataframe
subdf = df[df["subreddit"]==subreddit]

# add new column for sentiment
subdf['Sentiment'] = subdf[['title', 'content']].apply(lambda row: sia.polarity_scores

#show top 20 results
subdf.head(20)



#calculate average sentiment
average_sentiment = subdf['Sentiment'].mean()

#print result
print("Durchschnittliches Sentiment:", average_sentiment)



#get statistics about sentiment column
sentiment_stats = subdf['Sentiment'].describe()

#print results
print("Sentiment-Statistik:")
print(sentiment_stats)
```

```
# ### Top 10 and Bottom 10 Subreddits per Sentiment


#count subreddits and select only those with more than 10 new posts
subreddit_counts = df['subreddit'].value_counts()
subreddits = subreddit_counts[subreddit_counts > 10]
subdf = df_manip[df_manip['subreddit'].isin(subreddits.index)]

# add column for sentiment
subdf['Sentiment'] = subdf[['title', 'content']].apply(lambda row: sia.polarity_scores

#calculate average sentiment
subreddit_sentiment_avg = subdf.groupby('subreddit')['Sentiment'].mean()

# sort subreddits
sorted_subreddit_sentiment = subreddit_sentiment_avg.sort_values(ascending=False)

# top 10 subreddits
top_10_subreddits = sorted_subreddit_sentiment.head(10)

# bottom 10 subreddits
bottom_10_subreddits = sorted_subreddit_sentiment.tail(10)

# show results
print("Beste 10 Subreddits:")
print(top_10_subreddits)
print("")
print("Schlechteste 10 Subreddits:")
print(bottom_10_subreddits)


# ### Topic Modeling

import gensim
from gensim import corpora
```

```
#choose subreddit or select all
subdf = df_manip[df_manip["subreddit"]==subreddit]
#subdf = df_manip

#create lists
title_list = subdf['title'].tolist()
content_list = subdf['content'].tolist()
texts = title_list
texts.extend(content_list)
texts = [[word for word in doc.split()] for doc in texts]

# create dictionary and corpus
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

# train model
lda_model = gensim.models.LdaModel(corpus=corpus, id2word=dictionary, num_topics=10, p

# extract topics and words
topics = lda_model.print_topics(num_words=10)

#print results
for topic_id, topic in topics:
    topic_words = topic.split("+")
    topic_words = [word.split("*")[1].replace('"', '').strip() for word in topic_words
    topic_words_str = ", ".join(topic_words)
    print(f"Topic {topic_id + 1}: {topic_words_str}")


# ### Question-Answering-System


from transformers import pipeline

subdf = df[df["subreddit"]==subreddit]
```

```
text = ' '.join([f"{title} {content}" for title, content in zip(subdf['title'], subdf
qa = pipeline(task="question-answering", model="distilbert-base-uncased-distilled-squa

answer = qa(question="Who is trainer of Liverpool football club?", context=text)

print(answer)


#show context
range_of_context_lower = answer['start']-200 if answer['start']-200 >= 0 else 0
range_of_context_upper = answer['end']+300
context = text[range_of_context_lower:range_of_context_upper]
print(context)
```

Online-Quellen

Apache Software Foundation (2023a). *Apache Hadoop*. en. Zugriff am 24.05.2023. URL: https://hadoop.apache.org/.

— (2023b). *Apache Kafka*. en. Zugriff am 24.05.2023. URL: https://kafka.apache.org/.

— (2023c). *Apache Spark™ - Unified Engine for large-scale data analytics*. en. Zugriff am 29.05.2023. URL: https://spark.apache.org/.

Big Data Europe (2023a). *GitHub - big-data-europe/docker-hadoop: Apache Hadoop docker image*. en. Zugriff am 25.05.2023. URL: https://github.com/big-data-europe/docker-hadoop.

— (2023b). *GitHub - big-data-europe/docker-spark: Apache Spark docker image*. en. Zugriff am 31.05.2023. URL: https://github.com/big-data-europe/docker-spark.

Bitnami by VMware (2023). *bitnami/kafka - Docker Image | Docker Hub*. en. Zugriff am 31.05.2023. URL: https://hub.docker.com/r/bitnami/kafka.

Docker Inc. (2023). *Docker: Accelerated, Containerized Application Development*. de. Zugriff am 24. Mai 2023. URL: https://www.docker.com/.