Erläuterung zur Umsetzung der Dateninfrastruktur

In der praktischen Implementierung dieser Dateninfrastruktur wird eine Microservice-Architektur basierend auf Docker(-Compose) implementiert, die aus mehreren wesentlichen Komponenten besteht: einem Kafka-Producer für die Datenaufnahme, einem Spark-Service inklusive Kafka-Consumer zur Datenverarbeitung und einem HDFS zur Datenspeicherung.

Der im Rahmen des Projekts erstellte Code wird in einem Git-Repository versioniert. Dies gewährleistet eine lückenlose Nachverfolgung der Änderungen und erleichtert die Wartung sowie die Weiterentwicklung des Systems. In der dort vorliegenden README finden sich weitere Informationen sowie eine Installationsanleitung für das Projekt. Die exakten Konfigurationen der Komponenten können der "docker-compose.yml" aus dem GitHub-Repository entnommen werden.

Die Implementierung beginnt mit dem ***Data Ingestion Service***. Die Eingangsdaten für diesen Dienst liegen im Verzeichnis „/input“, von dem aus sie durch den Kafka-Producer eingelesen werden. Dieser wird über die Datei „kafka_producer.py“ eingebettet. Der Producer transformiert zunächst, mithilfe der „melt“-Funktion aus der Pandas Bibliothek, die Daten in ein verwertbares Format. Anschließend werden die Datensätze bereinigt, indem alle NaN-Werte herausgefiltert werden. Zuletzt werden die bereinigten und transformierten Daten in Batches von maximal 1 MB in das Kafka-Topic "Temperature" geschrieben. Für die Konnektivität und Verwaltung der Komponenten baut dieser Dienst eine Verbindung zu einem Kafka-Broker und einem Zookeeper auf.

Während der Kafka-Producer Batches verarbeitet und veröffentlicht, überprüft der ***Data Processing Service***, welcher über eine PySpark-Komponente (zu finden in der „temperature_processor.py“ Datei) realisiert wird, in einem Intervall von 10 Sekunden und durch Unterstützung eines Kafka-Consumers, ob neue Daten im Kafka-Topic „Temperature“ vorhanden sind. Ist dies der Fall, werden diese verarbeitet. Die aggregierten Daten umfassen unter anderem die monatliche Durchschnittstemperatur und den Modus sowie weitere Metriken. Dieser Prozess wird durch das „Absetzen“ eines Tasks mittels des „Spark-Submit“-Containers umgesetzt. Dieser übergibt die Ausführung der „temperature_processor.py“ an den „Spark-Master“-Container, welcher wiederum die Last auf die „Spark-Worker“ verteilt. In diesem Fall existiert nur ein Worker-Container. Dieser ist jedoch beliebig durch Hinzufügen weiterer Worker-Container erweiterbar.

Nach Abschluss der Verarbeitung eines Batches werden die Daten mittels des ***Data Persistence Service*** in einem Hadoop Distributed File System (HDFS) gespeichert. Die Verwaltung wird hierbei vom Namenode übernommen, welcher Metadaten speichert und die Verteilung der Daten an die Datanodes organisiert.

Sind alle Datensätze erfolgreich verarbeitet, bleibt der Spark-Service für mindestens 120 Sekunden aktiv. Während dieser Zeit prüft er in einem Intervall von 10 Sekunden, ob neue Daten im Kafka-Topic zur Verfügung gestellt wurden. Ist dies nicht der Fall, wird eine Funktion zur Visualisierung der

aggregierten Daten ausgeführt. Die resultierenden Visualisierungen werden in einer Ordnerstruktur im Ordner „Output“ gespeichert. Diese Struktur gliedert sich wie folgt:

→ Wetterstationsnummer (z.B. 164)

→ Jahr (z.B. 1996)

→ temperature_metrics.png

Nach dem jeweiligen Abschluss der Verarbeitung werden der Kafka-Producer und der Spark-Service inklusive des Kafka-Consumers gestoppt. Parallel wird in einem weiteren Docker-Container (hdfs-backup) ein Cron-Job erzeugt, welcher jeden Tag um 03:00 Uhr ein Backup der HDFS-Daten vornimmt, um die Datensicherheit zu gewährleisten. In diesem Backup können anschließend die Integrität und Vollständigkeit der gesicherten Daten überprüft werden, um den Aspekten der Data Governance gerecht zu werden.

Insgesamt stellt diese Microservice-Architektur eine hohe Skalierbarkeit, Verlässlichkeit und Wartbarkeit sicher, wobei auch Aspekte wie Datensicherheit, Data Governance und Datenschutz durch geeignete Maßnahmen berücksichtigt wurden. Weitere Informationen dazu finden Sie in der README.