**Aufgabe 5**

**1. Zielsetzung & Bewertungsbezug**

Diese Dokumentation erfüllt die Anforderungen der Aufgabe 5: **Installation & Konfiguration** eines Kafka‑Clusters (hier: Broker auf Master, ZK‑gesteuert), **Implementierung** einer Stream‑Processing‑Pipeline (Spark Structured Streaming), **horizontale Skalierbarkeit** (Partitionierung + skalierbare Spark‑Consumer), **Ergebnisse** (Parquet in HDFS) sowie **Nachvollziehbarkeit** (komplette Befehle, Code, Validierung).

**2. Datenfluss:** Binance‑Trades → Python‑Producer → Kafka‑Topic trades (2 Partitionen) → Spark‑Reader (Kafka Source) →

**3. Voraussetzungen**

* Bestehender Hadoop/HDFS + YARN‑Cluster (siehe Aufgabe 4)
* Ubuntu VMs: Master (cornelius-master) sowie Worker‑Knoten
* Java 11, Spark 3.5.1 inkl. YARN‑Integration
* Python 3 inkl. pip

**4. Kafka installieren & konfigurieren (Master)**

cd /opt

sudo curl -LO https://archive.apache.org/dist/kafka/3.4.1/kafka\_2.13-3.4.1.tgz

sudo tar -xzf kafka\_2.13-3.4.1.tgz

sudo ln -sfn /opt/kafka\_2.13-3.4.1 /opt/kafka

sudo chown -R ubuntu:ubuntu /opt/kafka\_2.13-3.4.1 /opt/kafka

# Kafka-Server für alle Interfaces, advertised Hostname, Log-Verzeichnis

sed -i 's|^#listeners=.\*|listeners=PLAINTEXT://:9092|' /opt/kafka/config/server.properties

grep -q '^advertised.listeners=' /opt/kafka/config/server.properties || \

echo 'advertised.listeners=PLAINTEXT://cornelius-master:9092' >> /opt/kafka/config/server.properties

sed -i 's|^log.dirs=.\*|log.dirs=/tmp/kafka-logs|' /opt/kafka/config/server.properties

# Startreihenfolge: ZooKeeper → Broker

/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties

sleep 3

/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties

sleep 3

# Reachability-Check

/opt/kafka/bin/kafka-broker-api-versions.sh --bootstrap-server cornelius-master:9092

# Topic 'trades' (2 Partitionen)

/opt/kafka/bin/kafka-topics.sh --bootstrap-server cornelius-master:9092 \

--create --topic trades --partitions 2 --replication-factor 1 || true

/opt/kafka/bin/kafka-topics.sh --bootstrap-server cornelius-master:9092 \

--describe --topic trades

**Prereqs prüfen:**

hdfs dfsadmin -report | sed -n '1,40p' # Live datanodes = 2

yarn node -list # NodeManagers RUNNING

**5. Datenquelle & Nachrichtenformat**

* **Quelle:** Binance WebSocket Stream btcusdt@trade (Trades in JSON)
* **Zieltopic:** trades
* **Minimal‑Schema:** { symbol: STRING, ts: LONG(ms), price: DOUBLE, qty: DOUBLE }

**6. Ingestion‑Producer (WebSocket → Kafka)**

**6.1 Abhängigkeiten**

python3 -m pip install --user kafka-python websocket-client

(Optional: python3 -m venv ~/venv && source ~/venv/bin/activate)

**6.2 Producer‑Script**

Datei: ~/project/task05/binance/binance\_ws\_to\_kafka.py

import json, os, time

from websocket import WebSocketApp

from kafka import KafkaProducer

BOOTSTRAP = os.environ.get("KAFKA\_BOOTSTRAP","cornelius-master:9092")

TOPIC = "trades"

URL = "wss://stream.binance.com:9443/stream?streams=btcusdt@trade"

producer = KafkaProducer(bootstrap\_servers=[BOOTSTRAP],

value\_serializer=lambda v: json.dumps(v).encode("utf-8"))

def on\_msg(\_, message):

m = json.loads(message).get("data", {})

out = {

"symbol": m.get("s"),

"ts": m.get("T"), # ms

"price": float(m["p"]) if m.get("p") else None,

"qty": float(m["q"]) if m.get("q") else None,

}

if out["symbol"] and out["ts"]:

producer.send(TOPIC, out)

def on\_err(\_, err): print("WS error:", err)

def on\_close(\*\_): print("WS closed")

def run():

while True:

try:

WebSocketApp(URL, on\_message=on\_msg, on\_error=on\_err, on\_close=on\_close).run\_forever()

except Exception as e:

print("Reconnect after error:", e)

time.sleep(2)

if \_\_name\_\_ == "\_\_main\_\_":

run()

**Start:**

KAFKA\_BOOTSTRAP="cornelius-master:9092" \

python3 ~/project/task05/binance/binance\_ws\_to\_kafka.py

**Schnellkontrolle:**

/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server cornelius-master:9092 \

--topic trades --from-beginning --max-messages 5

**7. Stream‑Processing (Kafka → Spark → HDFS)**

**7.1 Script anlegen**

Datei: ~/project/task05/spark-streaming/stream\_trades\_v2.py

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, from\_json, to\_timestamp, from\_unixtime, window, sum as F\_sum

from pyspark.sql.types import StructType, StructField, StringType, LongType, DoubleType

spark = SparkSession.builder.appName("stream\_trades\_v2").getOrCreate()

spark.sparkContext.setLogLevel("WARN")

schema = StructType([

StructField("symbol", StringType(), True),

StructField("ts", LongType(), True), # ms

StructField("price", DoubleType(), True),

StructField("qty", DoubleType(), True)

])

kafka\_df = (spark.readStream

.format("kafka")

.option("kafka.bootstrap.servers", "cornelius-master:9092")

.option("subscribe", "trades")

.option("startingOffsets", "latest")

.load())

rows = (kafka\_df.selectExpr("CAST(value AS STRING) AS json")

.select(from\_json(col("json"), schema).alias("r"))

.select("r.\*")

.withColumn("event\_time", to\_timestamp(from\_unixtime(col("ts")/1000.0))))

# 1) Console-Sink (Sichtkontrolle)

console\_q = (rows.writeStream

.format("console")

.outputMode("append")

.option("truncate","false")

.trigger(processingTime="10 seconds")

.start())

# 2) Parquet + Checkpoint (voll qualifizierte HDFS-URIs)

file\_q = (rows.writeStream

.format("parquet")

.option("path", "hdfs://cornelius-master:9000/out/stream\_trades")

.option("checkpointLocation", "hdfs://cornelius-master:9000/chk/stream\_trades")

.outputMode("append")

.trigger(processingTime="10 seconds")

.start())

# 3) Optional: 1-min VWAP pro Symbol

agg = (rows

.withWatermark("event\_time","2 minutes")

.groupBy(window(col("event\_time"), "1 minute"), col("symbol"))

.agg((F\_sum(col("price")\*col("qty"))/F\_sum("qty")).alias("vwap"),

F\_sum("qty").alias("sum\_qty")))

agg\_q = (agg.writeStream

.format("console")

.outputMode("update")

.option("numRows","50")

.trigger(processingTime="30 seconds")

.start())

spark.streams.awaitAnyTermination()

**7.2 Ausführung**

spark-submit \

--master yarn --deploy-mode client \

--conf spark.executor.instances=2 \

--conf spark.executor.cores=1 \

--conf spark.executor.memory=512m \

--conf spark.driver.memory=512m \

--conf spark.sql.shuffle.partitions=2 \

--packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.1 \

~/project/task05/spark-streaming/stream\_trades\_v2.py

**Erklärung:**

* **Kafka‑Connector** wird per --packages geladen.
* **Trigger** (10s) steuert Micro‑Batch‑Takt; Checkpointing ermöglicht Recovery & Exactly‑once‑Verarbeitung soweit der Sink es unterstützt.

**7.3 Validierung der Ergebnisse**

hdfs dfs -ls -R /out/stream\_trades | head

hdfs dfs -ls -R /chk/stream\_trades | head

# Ad-hoc Analyse

spark-sql <<'SQL'

CREATE OR REPLACE TEMP VIEW stream\_trades

USING parquet OPTIONS (path 'hdfs://cornelius-master:9000/out/stream\_trades');

SELECT symbol, COUNT(\*) rows, SUM(qty) total\_qty, AVG(price) avg\_price

FROM stream\_trades GROUP BY symbol;

SQL

**8. Horizontale Skalierbarkeit**

**8.1 Skalierung über Kafka‑Partitionen**

* **Ist:** trades mit **2** Partitionen → bis zu 2 parallele Consumer‑Tasks (pro Kafka‑Gruppe).
* **Skalierung:**

/opt/kafka/bin/kafka-topics.sh --bootstrap-server cornelius-master:9092 \

--alter --topic trades --partitions 4

/opt/kafka/bin/kafka-topics.sh --bootstrap-server cornelius-master:9092 --describe --topic trades

Hinweis: Partitionen lassen sich **erhöhen**, nicht verringern. Verteilung greift für neue Nachrichten.

**8.2 Skalierung über Spark‑Ressourcen**

* Erhöhe parallele Consumer/Tasks:

spark-submit \

--master yarn --deploy-mode client \

--conf spark.executor.instances=4 \

--conf spark.executor.cores=1 \

--conf spark.sql.shuffle.partitions=4 \

--packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.1 \

~/project/task05/spark-streaming/stream\_trades\_v2.py

* Beobachtung in **YARN UI (8088)**/**Spark UI (History Server)**: mehr aktive Tasks, geringere Batch‑Latenz, sinkender Consumer‑Lag (§9.2).

**9. Betrieb, Monitoring & Semantik**

**9.1 UIs & Logs**

* **YARN RM UI:** http://<MASTER>:8088 – App‑Status, Container‑Logs
* **Spark History:** (falls aktiviert) http://<MASTER>:18080 – Batch‑Laufzeiten, Wasserstände
* **Kafka Tools:**

# Topic/ISR/Leader

/opt/kafka/bin/kafka-topics.sh --bootstrap-server cornelius-master:9092 --describe --topic trades

# Consumer-Lag pro Gruppe (Connector: spark-structured-streaming-...)

/opt/kafka/bin/kafka-consumer-groups.sh --bootstrap-server cornelius-master:9092 \

--describe --group spark-kafka-source-... # tatsächlichen Gruppennamen aus den Logs/UI entnehmen

**9.2 Metriken & Nachweis der Skalierbarkeit**

* **Batch‑Dauer** (Spark UI): sinkt bei mehr Executor/Partitionen.
* **Lag** (Consumer‑Groups): geht gegen 0 bei nachhaltiger Verarbeitung.
* **HDFS Throughput**: Anzahl/Größe der ausgelieferten Parquet‑Dateien pro Minute.

**9.3 Verarbeitungssemantik**

* Kafka + Structured Streaming bieten **mindestens einmal**‑Zustellung mit idempotenter/transactional‑Sinks.
* **Parquet/HDFS (append)** ist typischerweise **at‑least‑once** → potenzielle Duplikate bei Restarts.  
  **Gegenmaßnahme:** dropDuplicates(["symbol","ts"]).withWatermark("event\_time","2 minutes") vor dem Sinken (Trade‑ID wäre ideal).

**10. Ergebnisse & Nutzen**

* Live‑Ingestion aus externer Quelle in Kafka.
* Streaming‑ETL mit Spark: Parsing, Zeitstempelung, (optionale) **VWAP‑Aggregation**.
* Persistenz als **Parquet** inkl. Checkpointing in HDFS.
* **Skalierbarkeit** demonstriert über Partitionen und Executor‑Skalierung.