**Aufgabe 4 – Data Lake & Big Data Processing**

**1. Zielsetzung & Bewertungsbezug**

Mit dieser Dokumentation erfülle ich die Kernanforderungen der Aufgabe 4: **Installation** des Data Lakes (Hadoop/HDFS + YARN + Spark), **Datenhaltung** sowie **Verarbeitungsschritte** (Batch‑Jobs mit Spark, Ein-/Ausgabe in HDFS) und **Ergebnispräsentation**.

**2. Architekturüberblick**

**2.1 Cluster-Topologie**

+-----------------------+

| cornelius-master |

| NN (NameNode), RM |

| 141.72.13.55 |

+-----------+-----------+

| \

| \

+---------------------+ +---------------------+

| |

+----v-------------------+ +---------v------------------+

| cornelius-worker-1 | | cornelius-worker-2 |

| DN (DataNode), NM | | DN (DataNode), NM |

| 141.72.13.58 | | 141.72.12.68 |

+------------------------+ +----------------------------+

**Rollen:**

* **HDFS:** NameNode (Master), DataNodes (Worker‑1, Worker‑2), Replikation = 2
* **YARN:** ResourceManager (Master), NodeManagers (Worker‑1, Worker‑2)
* **Spark:** Client/Cluster‑Modus auf **YARN**

**2.2 Wichtige Ports / Web‑UIs**

* HDFS NameNode UI: http://141.72.13.55:9870
* YARN ResourceManager UI: http://141.72.13.55:8088

**3. Systemvoraussetzungen**

* Ubuntu (VMs): 1× Master, 2× Worker
* **Java 11 (OpenJDK)**, **Hadoop 3.3.6**, **Spark 3.5.1 (Hadoop3 build)**
* SSH‑Zugang per Key, ausgehender Internetzugang für Downloads
* Ausreichender Plattenplatz für Roh‑ und Kurationsdaten

**4. Installation (Reproduzierbar, schrittweise)**

**Hinweis:** Sofern nicht anders angegeben, gelten die Schritte auf **allen drei Hosts** (Master, Worker‑1, Worker‑2). Befehle sind **idempotent** gestaltet, wo möglich.

**4.1 Login per SSH (vom lokalen Rechner)**

ssh -i ~/.ssh/corne\_key.pem ubuntu@141.72.13.55

ssh -i ~/.ssh/corne\_key.pem ubuntu@141.72.13.58

ssh -i ~/.ssh/corne\_key.pem [ubuntu@141.72.12.68](mailto:ubuntu@141.72.12.68)

**4.2 Basis‑Pakete & Java**

sudo apt-get update -y

sudo apt-get install -y openjdk-11-jdk curl unzip net-tools procps

**4.3 Hostnamen auflösen**

echo "141.72.13.55 cornelius-master" | sudo tee -a /etc/hosts

echo "141.72.13.58 cornelius-worker-1" | sudo tee -a /etc/hosts

echo "141.72.12.68 cornelius-worker-2" | sudo tee -a /etc/hosts

**4.4 Umgebungsvariablen**

# Java

sudo tee /etc/profile.d/java.sh >/dev/null <<'SH'

export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64

export PATH="$JAVA\_HOME/bin:$PATH"

SH

# Hadoop

sudo tee /etc/profile.d/hadoop.sh >/dev/null <<'SH'

export HADOOP\_HOME=/opt/hadoop

export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop

export PATH="$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin:$PATH"

SH

# Spark

sudo tee /etc/profile.d/spark.sh >/dev/null <<'SH'

export SPARK\_HOME=/opt/spark-3.5.1-bin-hadoop3

export PATH="$SPARK\_HOME/bin:$PATH"

SH

# aktivieren

source /etc/profile.d/java.sh; source /etc/profile.d/hadoop.sh; source /etc/profile.d/spark.sh

java -version

**4.5 Hadoop & Spark installieren**

# Hadoop 3.3.6

curl -L https://archive.apache.org/dist/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz \

| sudo tar -xz -C /opt

sudo ln -sfn /opt/hadoop-3.3.6 /opt/hadoop

# Spark 3.5.1 (Hadoop3 build)

curl -L https://archive.apache.org/dist/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tgz \

| sudo tar -xz -C /opt

**5. Hadoop/HDFS + YARN einrichten**

**5.1 Master konfigurieren (nur auf cornelius-master)**

**Verzeichnisse**

sudo mkdir -p /var/lib/hadoop/hdfs/{namenode,datanode}

sudo chown -R ubuntu:ubuntu /var/lib/hadoop

**core-site.xml**

<?xml version="1.0"?>

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://cornelius-master:9000</value>

</property>

</configuration>

**hdfs-site.xml**

<?xml version="1.0"?>

<configuration>

<property><name>dfs.replication</name><value>2</value></property>

<property><name>dfs.namenode.name.dir</name><value>file:///var/lib/hadoop/hdfs/namenode</value></property>

<property><name>dfs.datanode.data.dir</name><value>file:///var/lib/hadoop/hdfs/datanode</value></property>

</configuration>

**yarn-site.xml** (Master/RM)

<?xml version="1.0"?>

<configuration>

<property><name>yarn.resourcemanager.hostname</name><value>cornelius-master</value></property>

<property><name>yarn.resourcemanager.address</name><value>cornelius-master:8032</value></property>

<property><name>yarn.resourcemanager.scheduler.address</name><value>cornelius-master:8030</value></property>

<property><name>yarn.resourcemanager.resource-tracker.address</name><value>cornelius-master:8031</value></property>

<property><name>yarn.resourcemanager.webapp.address</name><value>0.0.0.0:8088</value></property>

<property><name>yarn.nodemanager.aux-services</name><value>mapreduce\_shuffle</value></property>

<property><name>yarn.nodemanager.vmem-check-enabled</name><value>false</value></property>

<property><name>yarn.scheduler.minimum-allocation-mb</name><value>256</value></property>

<property><name>yarn.scheduler.maximum-allocation-mb</name><value>4096</value></property>

</configuration>

**Worker-Liste**

printf "cornelius-worker-1\ncornelius-worker-2\n" | sudo tee /opt/hadoop/etc/hadoop/workers

**Namenode initialisieren & Dienste starten**

hdfs namenode -format

hdfs --daemon start namenode

yarn --daemon start resourcemanager

jps

ss -ltnp | grep -E '9000|9870|8088' # UIs: 9870(HDFS), 8088(YARN)

**Konfigs auf Worker verteilen**

# lokal:

ssh-add -D

ssh-add ~/.ssh/corne\_key.pem

ssh -A -i ~/.ssh/corne\_key.pem ubuntu@141.72.13.55

# auf master:

scp /opt/hadoop/etc/hadoop/{core-site.xml,hdfs-site.xml} ubuntu@141.72.13.58:/tmp/

scp /opt/hadoop/etc/hadoop/{core-site.xml,hdfs-site.xml} [ubuntu@141.72.12.68:/tmp/](mailto:ubuntu@141.72.12.68:/tmp/)

**5.2 Worker konfigurieren (auf beiden Workern)**

sudo mv /tmp/core-site.xml /opt/hadoop/etc/hadoop/

sudo mv /tmp/hdfs-site.xml /opt/hadoop/etc/hadoop/

sudo tee /opt/hadoop/etc/hadoop/yarn-site.xml >/dev/null <<'XML'

<?xml version="1.0"?>

<configuration>

<property><name>yarn.resourcemanager.hostname</name><value>cornelius-master</value></property>

<property><name>yarn.resourcemanager.address</name><value>cornelius-master:8032</value></property>

<property><name>yarn.resourcemanager.scheduler.address</name><value>cornelius-master:8030</value></property>

<property><name>yarn.resourcemanager.resource-tracker.address</name><value>cornelius-master:8031</value></property>

<property><name>yarn.nodemanager.resource.memory-mb</name><value>2048</value></property>

<property><name>yarn.nodemanager.resource.cpu-vcores</name><value>1</value></property>

<property><name>yarn.nodemanager.aux-services</name><value>mapreduce\_shuffle</value></property>

<property><name>yarn.nodemanager.vmem-check-enabled</name><value>false</value></property>

</configuration>

XML

sudo mkdir -p /var/lib/hadoop/hdfs/datanode

sudo chown -R ubuntu:ubuntu /var/lib/hadoop

# Dienste starten

hdfs --daemon start datanode

yarn --daemon start nodemanager

jps

**Clusterstatus prüfen**

hdfs dfsadmin -report | sed -n '1,120p'

yarn node -list

**6. Datenhaltung**

**6.1 HDFS‑Namensraum & Zonen**

Wir strukturieren den Data Lake in **Zonen** zur sauberen Trennung von Roh‑, Kurations- und Ausgabelayern:

/datalake

/raw # unveränderte Rohdaten, wie angeliefert (CSV/JSON/Parquet)

/curated # bereinigte, validierte, angereicherte Daten (Parquet)

/sandbox # Ad‑hoc Analysen/Experimente

/results # finale Analyse-/Berichtsergebnisse (Parquet/CSV)

/tmp # temporäre Stagingbereiche

**Anlegen:**

hdfs dfs -mkdir -p /datalake/{raw,curated,sandbox,results,tmp}

**6.2 Replikation, Blockgröße, Quotas**

* **dfs.replication = 2** (s.o.), schützt gegen Node‑Ausfälle bei 2 Worker‑Knoten.
* **Quotas** optional zur Kostenkontrolle: hdfs dfsadmin -setSpaceQuota / -setQuota.

**6.3 Dateiformate & Kompression**

* **Eingang:** häufig **CSV/JSON**.
* **Kuratiert:** **Parquet + Snappy** (spaltenorientiert, komprimiert, Prädikat‑Pushdown, Schema‑Evolution).
* **Partitionierung:** nach natürlicher Dimension (z. B. year=2024/month=01), verbessert Prädikatfilter.

**6.4 Namenskonventionen**

/datalake/raw/<domain>/<dataset>/ingest\_date=YYYY-MM-DD/part-\*.csv  
/datalake/curated/<domain>/<dataset>/year=YYYY/month=MM/part-\*.parquet

**6.5 Zugriffsrechte**

* POSIX‑ACLs/HDFS‑Rechte je Zone (hdfs dfs -chmod, -chown), **write‑once, read‑many** für /raw.

**7. Verarbeitung mit Spark auf YARN (Batch‑Pipeline)**

**7.1 Spark‑Defaults (empfohlen)**

sudo tee /opt/spark-3.5.1-bin-hadoop3/conf/spark-defaults.conf >/dev/null <<'CONF'

spark.eventLog.enabled true

spark.eventLog.dir hdfs:///spark-logs

spark.history.fs.logDirectory hdfs:///spark-logs

CONF

hdfs dfs -mkdir -p /spark-logs

**History‑Server starten (optional):**

$SPARK\_HOME/sbin/start-history-server.sh

**7.2 Beispiel‑Job (ETL von CSV → Parquet, Partitionierung, Aggregation)**

**Datei:** etl\_nyc\_taxi.py

#!/usr/bin/env python3

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, year, month, to\_timestamp, count, avg

spark = (SparkSession.builder

.appName("etl-nyc-taxi")

.getOrCreate())

raw = "/datalake/raw/mobility/nyc\_taxi/ingest\_date=2025-08-31"

cur = "/datalake/curated/mobility/nyc\_taxi"

res = "/datalake/results/mobility/nyc\_taxi"

# 1) Laden & Typisierung

df = (spark.read

.option("header", True)

.option("inferSchema", True)

.csv(raw))

# 2) Bereinigung/Typen

df = (df

.withColumn("pickup\_ts", to\_timestamp(col("tpep\_pickup\_datetime")))

.withColumn("dropoff\_ts", to\_timestamp(col("tpep\_dropoff\_datetime")))

.dropna(subset=["pickup\_ts", "dropoff\_ts", "passenger\_count", "total\_amount"]))

# 3) Kuratiert schreiben (Parquet, partitioniert)

(df.write

.mode("overwrite")

.partitionBy(year(col("pickup\_ts")).alias("year"), month(col("pickup\_ts")).alias("month"))

.parquet(cur))

# 4) Aggregation (Beispielmetriken)

agg = (df

.withColumn("year", year(col("pickup\_ts")))

.withColumn("month", month(col("pickup\_ts")))

.groupBy("year", "month")

.agg(count("\*").alias("trips"), avg("total\_amount").alias("avg\_fare")))

# 5) Ergebnisse schreiben

(agg.write

.mode("overwrite")

.partitionBy("year", "month")

.parquet(res))

spark.stop()

**Hinweis:** Partitionierung über year, month wird sowohl im Kurations‑Layer als auch im Ergebnis genutzt.

**Ausführung auf YARN (Cluster‑Modus):**

spark-submit \

--master yarn \

--deploy-mode cluster \

--conf spark.yarn.maxAppAttempts=1 \

--conf spark.executor.memory=1g \

--conf spark.executor.cores=1 \

--conf spark.executor.instances=2 \

etl\_nyc\_taxi.py

**7.3 Validierung & Sichtprüfung**

# Ergebnisse ansehen

hdfs dfs -ls -R /datalake/curated/mobility/nyc\_taxi | head -n 50

hdfs dfs -ls -R /datalake/results/mobility/nyc\_taxi | head -n 50

# Stichprobe lesen (Spark Shell)

spark-shell --master yarn --deploy-mode client <<'SCALA'

val df = spark.read.parquet("/datalake/results/mobility/nyc\_taxi")

df.orderBy($"year".desc, $"month".desc).show(12, false)

SCALA

**8. Ergebnisse & Nutzen**

* Funktionsfähiger **Data Lake** (HDFS) mit **Replikation 2** über zwei Worker.
* **Spark‑ETL** wandelt Roh‑CSV in **Parquet** (kuratiert), erzeugt **partitionierte Aggregationen** (Results).
* Pipeline ist wiederholbar, nachvollziehbar (optionale **History Logs**).
* Strukturierte **Datenzonen** erleichtern Governance, Performance und spätere Erweiterungen.