LAPORAN PRAKTIKUM BIG DATA

Implementasi HDFS, MongoDB, Spark, Data Preprocessing, dan Data Integration

DAFTAR ISI

---

1. PENDAHULUAN

1.1 Latar Belakang

Big Data menjadi teknologi krusial dalam era digital dengan volume data yang tumbuh eksponensial. Praktikum ini mengimplementasikan tools utama ekosistem Big Data untuk memahami arsitektur, kelebihan, dan use case masing-masing platform.

1.2 Tujuan Praktikum

- Memahami sistem penyimpanan data besar (HDFS, MongoDB, Cassandra)
- Mengimplementasikan processing data dengan Spark
- Melakukan pra-pemrosesan data untuk analisis
- Mengintegrasikan data dari berbagai sumber menggunakan Sqoop, Flume, Kafka

---

2. METODOLOGI PRAKTIKUM

2.1 Environment Praktikum

- **Sistem Operasi**: Windows 11 dengan Docker Desktop
- **Tools**: Docker Container, Google Colab, MongoDB, Hadoop

- **Bahasa**: Python, PySpark, MongoDB Query, SQL

2.2 Alur Implementasi

1. Setup environment dengan Docker containers
2. Implementasi storage systems (HDFS, MongoDB)
3. Data processing dengan Apache Spark
4. Data cleaning dan transformation
5. Data integration pipeline simulation

---

3. HASIL DAN PEMBAHASAN

3.1 APACHE HDFS

Implementasi

bash

Docker Container Setup

docker run -it harisekhon/hadoop:2.8 /bin/bash

hdfs namenode -format

Kendala yang Dihadapi

```
[root@8e9845f4ae8d /]# $HADOOP_HOME/sbin/start-dfs.sh
Starting namenodes on [localhost]
localhost: ssh: connect to host localhost port 22: Connection refused
localhost: ssh: connect to host localhost port 22: Connection refused
Starting secondary namenodes [0.0.0.0]
0.0.0.0: ssh: connect to host 0.0.0.0 port 22: Connection refused
```

HDFS mengalami kendala teknis pada container Hadoop karena:

- **SSH Service** tidak berjalan dalam container
- **Network configuration** terbatas pada environment Docker
- **Java environment** compatibility issues

Analisis Teoritis HDFS

- **Architecture**: Master-Slave dengan NameNode dan DataNode
- **Data Distribution**: File dipecah menjadi blocks (default 128MB)
- **Fault Tolerance**: Replikasi data ke multiple nodes
- **Use Case**: Penyimpanan file besar untuk batch processing

3.2 MONGODB

Implementasi Berhasil

```
PS C:\Users\hbudi> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS
                     NAMES
8a0df81d3f51   mongo:latest           "docker-entrypoint.s…"   17 seconds ago   Up 16 seconds   0.0.0.0:27017->27017
/tcp, [::]:27017->27017/tcp   mongodb
8e9845f4ae8d   harisekhon/hadoop:2.8  "/bin/bash"              19 minutes ago   Up 19 minutes   0.0.0.0:50070->50070
/tcp, [::]:50070->50070/tcp   focused_williams
```

javascript

```
// Database Operations
use praktikum
db.mahasiswa.insertOne({
  nim: "12345",
  nama: "Andi",
  jurusan: "Informatika",
  alamat: {
    jalan: "Jl. Merdeka No. 1",
    kota: "Jakarta"
  }
})

// Insert banyak data
db.mahasiswa.insertMany([
  { nim: "12346", nama: "Budi", jurusan: "Sistem Informasi" },
  { nim: "12347", nama: "Citra", jurusan: "Teknik Komputer" }
])
```

Hasil Query

```
praktikum> db.mahasiswa.find().pretty()
[
  {
    _id: ObjectId('690c604a6ca196918dce5f47'),
    nim: '12345',
    nama: 'Andi',
    jurusan: 'Informatika',
    alamat: { jalan: 'Jl. Merdeka No. 1', kota: 'Jakarta', kode_pos: '12345' },
    kontak: { email: 'andi@email.com', telepon: '08123456789' }
  }
]
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('690d8d207189e69833ce5f48'),
    '1': ObjectId('690d8d207189e69833ce5f49')
  }
}
```

```
test> db.mahasiswa.createIndex({ nim: 1 })
nim_1
test> db.mahasiswa.find().sort({ nama: 1 })
    nama: 'Budi',
    jurusan: 'Sistem Informasi'
  },
  {
    _id: ObjectId('690d8d207189e69833ce5f49'),
    nim: '12347',
    nama: 'Citra',
    jurusan: 'Teknik Komputer'
  }
]
test>
```

Fitur yang Diuji:

- Create database and collection
- Insert data dengan nested documents
- Query dengan filter condition
- Index creation untuk optimisasi
- Aggregation pipeline
- Sorting dan projection

Kelebihan MongoDB

- **Flexible Schema**: Document-based structure
- **Horizontal Scaling**: Sharding capabilities
- **Rich Query Language**: Powerful aggregation framework
- **Use Case**: JSON data, real-time applications, content management

Praktikum Cassandra

```
cqlsh:praktikum> CREATE TABLE mahasiswa (
            ...     nim text PRIMARY KEY,
            ...     nama text,
            ...     jurusan text
            ... );
cqlsh:praktikum>
cqlsh:praktikum> INSERT INTO mahasiswa (nim, nama, jurusan) VALUES ('12345', 'Budi', 'Informatika');
cqlsh:praktikum> SELECT * FROM mahasiswa;INSERT INTO mahasiswa (nim, nama, jurusan) VALUES ('12346', 'Citra', 'Sistem Informasi');

cqlsh:praktikum> SELECT * FROM mahasiswa WHERE jurusan='Informatika' ALLOW FILTERING;

 nim   | jurusan     | nama
-------+-------------+------
 12345 | Informatika | Budi

(1 rows)
cqlsh:praktikum> SELECT * FROM mahasiswa WHERE jurusan='Sistem Informasi' ALLOW FILTERING;

 nim   | jurusan          | nama
-------+------------------+-------
 12346 | Sistem Informasi | Citra

(1 rows)
```

3.3 APACHE SPARK

Implementasi di Google Colab

```
from pyspark.sql import SparkSession

# Membuat Spark session
spark = SparkSession.builder \
    .appName("SimpleSpark") \
    .getOrCreate()

print("✅ Spark berhasil dijalankan!")
```

```
✅ Spark berhasil dijalankan!
```

python

Word Count dengan RDD

data = ["Hello Spark", "Hello World", "Spark is awesome"]

rdd = spark.sparkContext.parallelize(data)

word_counts = rdd.flatMap(lambda line: line.split(" ")) \

.map(lambda word: (word, 1)) \

.reduceByKey(lambda a, b: a + b)

Perbandingan RDD vs DataFrame

```
=== SIMULASI SPARK WORD COUNT ===

1. SIMULASI RDD:
'Hello': 2
'Spark': 2
'World': 1
'is': 1
'awesome': 1
'Big': 1
'Data': 1
'praktikum': 1

2. SIMULASI DATAFRAME:
Hasil DataFrame-style:
       words  count
3      Spark      2
2      Hello      2
1       Data      1
0        Big      1
4      World      1
5    awesome      1
6         is      1
7  praktikum      1

✅ PRAKTIKUM SPARK SIMULASI SELESAI!
```

RDD (Resilient Distributed Dataset):

- Lower-level API
- Functional programming style
- More control over execution

DataFrame:

- Higher-level abstraction

- SQL-like operations
- Catalyst optimizer for performance
- Better performance untuk structured data

3.4 DATA PREPROCESSING

Data Cleaning Implementasi

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.12/dist-packages (3.5.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.12/dist-packages (from pyspark) (0.10.9.7)
DATA AWAL:
   id           nama  usia      gaji jk        tgl      kota  \
0   1    Budi Susanto  25.0   5500000  L  2022-01-15   Jakarta
1   2     Ani Lestari   NaN   8000000  P  2022-02-20   Bandung
2   3   Candra Wijaya  35.0  12000000  L  2022-01-18  Surabaya
3   4  Dewi Anggraini  22.0   4800000  P  2022-03-10       JKT
4   5    Budi Susanto  25.0   5500000  L  2022-01-15   Jakarta

              ulasan
0  Transaksi berhasil
1    Pengiriman cepat
2         Sangat puas
3     Barang diterima
4  Transaksi berhasil

=== DATA CLEANING ===
DATA SETELAH CLEANING:
   id           nama   usia      gaji jk        tgl      kota  \
0   1    Budi Susanto  25.00   5500000  L  2022-01-15   Jakarta
1   2     Ani Lestari  26.75   8000000  P  2022-02-20   Bandung
2   3   Candra Wijaya  35.00  12000000  L  2022-01-18  Surabaya
3   4  Dewi Anggraini  22.00   4800000  P  2022-03-10   Jakarta
4   5    Budi Susanto  25.00   5500000  L  2022-01-15   Jakarta

              ulasan
0  Transaksi berhasil
1    Pengiriman cepat
2         Sangat puas
3     Barang diterima
4  Transaksi berhasil
```

python

```
# Handling missing values
df_clean = df.na.fill({"usia": 30, "ulasan": "Tidak ada ulasan"})


# Remove duplicates
df_clean = df_clean.dropDuplicates()


# Standardize values
```

```
df_clean = df_clean.withColumn("kota",
    when(df_clean["kota"] == "JKT", "Jakarta")
    .otherwise(df_clean["kota"]))
```

Data Transformation

- **Standardization**: Z-score normalization
- **Normalization**: Min-Max scaling
- **Binning**: Age group categorization
- **Feature Engineering**: Date feature extraction

Feature Engineering

python

```
# Date feature extraction
df_engineered = df_clean.withColumn("tahun_reg", year("timestamp_reg"))
df_engineered = df_engineered.withColumn("bulan_reg", month("timestamp_reg"))

# One-Hot Encoding
ohe = OneHotEncoder(inputCols=["jk_index", "kota_index"],
                    outputCols=["jk_ohe", "kota_ohe"])
```

3.5 DATA INTEGRATION

Sqoop Simulation

```
================================================
PRAKTIKUM 1: APACHE SQOOP
Transfer data dari MySQL ke HDFS
================================================
📊 STEP 1: Data di MySQL Database 'company'
Tabel 'employees' di MySQL:
   id   nama   department     gaji
0   1   Andi  Engineering  7500000
1   2   Budi    Marketing  6500000
2   3  Citra  Engineering  8000000
3   4   Dewi           HR  6000000
4   5    Eka  Engineering  9000000

📥 STEP 2: Sqoop Import ke HDFS
Perintah: sqoop import --connect jdbc:mysql://localhost/company --table employees --target-dir /user/hadoop/employees -m 1
✅ Data berhasil diimport ke HDFS: /user/hadoop/employees/
Isi HDFS setelah import:
   id   nama   department     gaji
0   1   Andi  Engineering  7500000
1   2   Budi    Marketing  6500000
2   3  Citra  Engineering  8000000
3   4   Dewi           HR  6000000
4   5    Eka  Engineering  9000000

📤 STEP 3: Sqoop Export dari HDFS ke MySQL
Buat data baru di HDFS:
   id   nama department     gaji
0   6  Fajar      Sales  7000000
Perintah: sqoop export --connect jdbc:mysql://localhost/company --table employees_export --export-dir /user/hadoop/new_employees
✅ Data berhasil diexport ke MySQL tabel 'employees export'
```

Workflow:

1. MySQL → Sqoop → HDFS (Import)
2. HDFS → Sqoop → MySQL (Export)

Use Case: Periodic data sync antara RDBMS dan Hadoop ecosystem

Flume Simulation

```
==================================================
PRAKTIKUM 2: APACHE FLUME
Koleksi data log real-time
==================================================
🔄 STEP 1: Konfigurasi Flume Agent
File: netcat-logger.conf

# Agent components
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# NetCat Source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

# Logger Sink
a1.sinks.k1.type = logger

# Memory Channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000

# Binding
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1


🌊 STEP 2: Simulasi Data Streaming ke Flume
Flume Agent menerima data log:
  [1] 2024-01-15 10:30:15 INFO: User login successful - user_id: 12345
  [2] 2024-01-15 10:31:22 ERROR: Database connection timeout - retrying...
  [3] 2024-01-15 10:32:45 INFO: Data processing completed - records: 1000
  [4] 2024-01-15 10:33:10 WARN: High memory usage detected - 85%
  [5] 2024-01-15 10:34:30 INFO: Backup job started - size: 2.5GB

✅ Flume berhasil mengumpulkan 5 log events
```

Architecture:

- **Source**: NetCat (port 44444)
- **Channel**: Memory (1000 events capacity)
- **Sink**: Logger (console output)

Use Case: Real-time log collection dan monitoring

Kafka Simulation

```
==================================================
PRAKTIKUM 3: APACHE KAFKA
Message publishing & subscribing
==================================================
📝 STEP 1: Buat Kafka Topic
Perintah: kafka-topics.sh --create --topic praktikum-bigdata --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
✅ Topic 'praktikum-bigdata' berhasil dibuat

📤 STEP 2: Kafka Producer - Mengirim Pesan
Producer mengirim pesan ke topic 'praktikum-bigdata':
  PRODUCER → [1] Pesan pertama: Hello Kafka!
  PRODUCER → [2] Data sensor: temperature=25.6C, humidity=60%
  PRODUCER → [3] User action: button_clicked, item_id=789
  PRODUCER → [4] System alert: CPU usage 75%
  PRODUCER → [5] Transaction: order_id=456, amount=250000

📥 STEP 3: Kafka Consumer - Menerima Pesan
Perintah: kafka-console-consumer.sh --topic praktikum-bigdata --from-beginning --bootstrap-server localhost:9092
Consumer membaca pesan:
  CONSUMER ← [1] Pesan pertama: Hello Kafka!
  CONSUMER ← [2] Data sensor: temperature=25.6C, humidity=60%
  CONSUMER ← [3] User action: button_clicked, item_id=789
  CONSUMER ← [4] System alert: CPU usage 75%
  CONSUMER ← [5] Transaction: order_id=456, amount=250000

🔄 STEP 4: Real-time Streaming Simulation
Producer mengirim pesan baru...
  LIVE → Real-time update: stock_price=15200
  LIVE → New user registered: user_id=999
```

Components:

- **Topic**: praktikum-bigdata

- **Producer**: Message publisher
- **Consumer**: Message subscriber
- **Zookeeper**: Coordination service

Use Case: Real-time data streaming dan event processing

Integrasi Semua Data :

```
============================================================
INTEGRASI: DATA PIPELINE LENGKAP
MySQL → Sqoop → HDFS → Kafka → Flume → Spark
============================================================
🚀 ALUR DATA INTEGRATION:
   1. 📊 Data tersimpan di MySQL (transaksi, user data)
   2. 📥 Sqoop import data ke HDFS untuk processing
   3. 🗂️ Data di HDFS diproses oleh Spark/MapReduce
   4. 📤 Hasil processing dikirim ke Kafka topic
   5. 📨 Kafka stream data ke berbagai consumers
   6. 📝 Flume mengumpulkan application logs
   7. 🔄 Real-time monitoring dan analytics

🎯 USE CASE: E-COMMERCE DATA PLATFORM
   • MySQL: Produk, user, transaksi data
   • Sqoop: Periodic sync ke data warehouse
   • Kafka: Real-time user activity streaming
   • Flume: Application log collection
   • Spark: Analytics & machine learning

✅ DATA INTEGRATION PRAKTIKUM SELESAI!
📊 Total komponen yang dipraktikumkan:
   • Sqoop: Import/Export MySQL-HDFS
   • Flume: Log collection dengan NetCat source
   • Kafka: Producer/Consumer messaging
   • Integrasi: Full data pipeline simulation
```

---

4. ANALISIS PERBANDINGAN TOOLS

4.1 Storage Systems

Tool	Data Model	Scalability	Use Case
HDFS	File-based	Horizontal	Batch processing, Large files
MongoDB	Document	Horizontal	Real-time apps, JSON data
Cassandra	Column-family	Horizontal	Time-series, High write throughput

4.2 Processing Frameworks

Framework	Processing Model	Performance	Use Case
MapReduce	Batch	Disk-based	Legacy Hadoop jobs
Spark RDD	In-memory	High	Complex transformations
Spark DataFrame	In-memory	Very High	Structured data analytics

4.3 Data Integration

Tool	Pattern	Latency	Use Case
Sqoop	Batch	High	Database-HDFS integration
Flume	Streaming	Low	Log collection
Kafka	Streaming	Very Low	Real-time messaging

---

5. KENDALA DAN SOLUSI

5.1 Technical Challenges

HDFS Docker Issues

- **Problem**: SSH service tidak tersedia dalam container
- **Solution**: Dokumentasi kendala dan analisis teoritis

Spark Java Configuration

- **Problem**: Java gateway process exited in Colab
- **Solution**: Use simplified Spark configuration dan simulation

Time Constraints

- **Problem**: Waktu praktikum terbatas dengan jadwal padat
- **Solution**: Fokus pada tools yang feasible dan simulation untuk complex setup

5.2 Learning Outcomes

- Understanding trade-offs antara different Big Data tools
- Hands-on experience dengan real deployment challenges
- Ability to choose appropriate tools untuk specific use cases

---

6. KESIMPULAN

6.1 Key Findings

1. **MongoDB** excelled untuk flexible schema dan rapid development
2. **Spark** provided superior performance untuk data processing dengan in-memory computation
3. **screeData Integration** tools memiliki specialized use cases dalam data pipeline
4. **Containerization** mempermudah environment setup tetapi memiliki limitations untuk distributed systems

6.2 Recommendations

1. **Untuk real-time applications**: MongoDB + Kafka
2. **Untuk batch analytics**: HDFS + Spark
3. **Untuk data integration**: Kombinasi Sqoop, Flume, Kafka berdasarkan latency requirements
4. **Untuk development**: Gunakan containerized environments untuk konsistensi

6.3 Future Work

- Implementasi multi-node cluster setup
- Performance benchmarking pada dataset besar
- Integration dengan cloud platforms (AWS, GCP, Azure)
- Machine learning pipeline dengan Spark MLlib

Link Repository :

https://github.com/Budskyman/BigData-Praktikum-2025?tab=readme-ov-file#bigdata-praktikum-2025