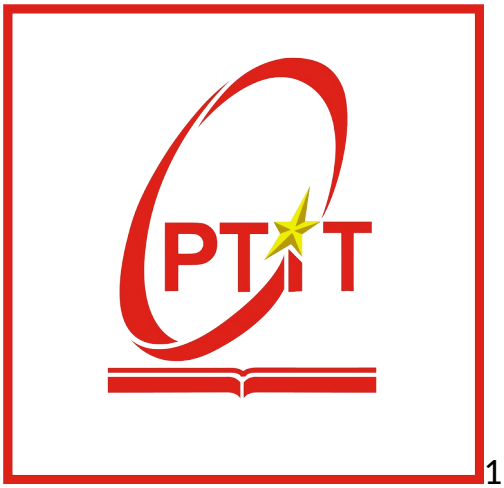
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



1

Báo cáo hàng tuần

Môn học: Thực tập cơ sở

Giảng viên: Kim Ngọc Bách

Họ và tên: Nguyễn Hữu Phúc

Mã SV: B22DCAT224

Lớp: E22CQCN04-B

Báo cáo tuần 9

Trong tuần này, em đã tập trung vào việc tối ưu hóa và chỉnh sửa code, đồng thời triển khai thành công việc thu nhận dữ liệu từ Kafka sang Apache Spark. Các nhiệm vụ chính bao gồm:

- **Cải thiện mã nguồn: Em đã xem xét và nâng cấp mã nguồn hiện tại, chú trọng vào việc tăng tính dễ đọc, tính mô-đun và hiệu suất. Quá trình này bao gồm tái cấu trúc các thành phần quan trọng để đảm bảo khả năng bảo trì và mở rộng.**
- **Tích hợp Kafka và Spark: Em đã phát triển và tích hợp một pipeline ETL mạnh mẽ bằng Spark để xử lý dữ liệu thời gian thực từ topic Kafka 'product_views'. Pipeline này phân tích dữ liệu JSON, thực hiện các biến đổi cần thiết và chuẩn bị dữ liệu cho các bước xử lý tiếp theo, tạo ra một luồng dữ liệu liền mạch giữa Kafka và Spark.**

Cấu trúc file thư mục

```
(base) nguyenphuc@phuc2108:~/Documents/GlamiraUserFlowInsightsProject/GlamiraUserFlowInsights/MyGlamira$ tree
.
├── config
│   ├── database_config.py
│   └── __pycache__
│       └── database_config.cpython-312.pyc
├── database
│   ├── __init__.py
│   ├── kafka_producer.py
│   ├── mongodb_connect.py
│   ├── __pycache__
│   │   ├── __init__.cpython-312.pyc
│   │   ├── kafka_producer.cpython-312.pyc
│   │   ├── mongodb_connect.cpython-312.pyc
│   │   └── schema_manager.cpython-312.pyc
│   └── schema_manager.py
└── src
    ├── main.py
    ├── __pycache__
    │   └── main.cpython-312.pyc
    └── spark_kafka_consumer.py
```

File main.py

```python
from database.mongodb_connect import MongoDBConnect,
query_product_views
from database.schema_manager import create_mongodb_schema,
validate_mongodb_schema
from config.database_config import get_database_config
from database.kafka_producer import KafkaMessageProducer

def main():
    configMongo = get_database_config()

    kafka_producer = KafkaMessageProducer(
        bootstrap_servers='localhost:9092',
        topic='product_views'
    )
```

```
   with MongoDBConnect(configMongo["mongodb"].uri,
configMongo["mongodb"].db_name) as mongo_client:
        db = mongo_client.db

        # Nếu có logic schema
        create_mongodb_schema(db)
        validate_mongodb_schema(db)

        # Gửi dữ liệu sang Kafka
        total_records = query_product_views(db, kafka_producer)
        print(f"Finished processing {total_records} records")



if __name__ == "__main__":
  main()
```

File spark_kafka_consumer.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql.streaming import StreamingQueryListener

class BatchListener(StreamingQueryListener):
   def __init__(self, max_empty_batches=3):
        super().__init__()
        self.empty_batches = 0
        self.max_empty_batches = max_empty_batches

   def onQueryStarted(self, event):
        print(f"Query started: {event.id}")

   def onQueryProgress(self, event):
        num_input_rows = event.progress['numInputRows']
        if num_input_rows == 0:
            self.empty_batches += 1
            print(f"⚠️ Empty batch count:
{self.empty_batches}")
        else:
            self.empty_batches = 0  # reset if data comes

        if self.empty_batches >= self.max_empty_batches:
```

```
            print("Đã đọc hết dữ liệu từ Kafka (hoặc không còn
dữ liệu mới).")
            # Stop the application gracefully
            event.sparkSession.stop()

    def onQueryTerminated(self, event):
        print(f"Query terminated: {event.id}")

def main():
    spark = SparkSession.builder \
        .appName("SparkKafkaConsumer") \
        .getOrCreate()

    spark.sparkContext.setLogLevel("WARN")

    # Đăng ký listener
    listener = BatchListener(max_empty_batches=3)
    spark.streams.addListener(listener)

    # Đọc từ Kafka
    df = spark.readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", "localhost:9092") \
        .option("subscribe", "product_views") \
        .option("startingOffsets", "earliest") \
        .load()

    # Chuyển value → string
    messages = df.selectExpr("CAST(value AS STRING) as
message")

    # Ghi ra console
    query = messages.writeStream \
        .outputMode("append") \
        .format("console") \
        .option("truncate", False) \
        .start()

    query.awaitTermination()

if __name__ == "__main__":
    main()
```

File database_config.py

```
import os
from dataclasses import dataclass
from dotenv import load_dotenv
from typing import Dict


class DatabaseConfig:
  def validate(self) -> None:
      for key, value in self.__dict__.items():
          if value is None:
              raise ValueError(f"Missing config for {key}")


@dataclass
class MongoDBConfig(DatabaseConfig):
  uri: str
  db_name: str


def get_database_config() -> Dict[str, DatabaseConfig]:
  load_dotenv()
  config = {
      "mongodb": MongoDBConfig(
          uri=os.getenv("MONGO_URI"),
          db_name=os.getenv("MONGO_DB_NAME")
      )
  }
  for db, setting in config.items():
      setting.validate()
  return config


db_config = get_database_config()
```

File kafka_producer.py

```
from kafka import KafkaProducer
import json

class KafkaMessageProducer:
   def __init__(self, bootstrap_servers='localhost:9092',
topic='product_views'):
       self.topic = topic
       self.producer = KafkaProducer(
           bootstrap_servers=bootstrap_servers,
```

```
            value_serializer=lambda v:
json.dumps(v).encode('utf-8')
        )

    def send(self, message: dict):
        self.producer.send(self.topic, message)

    def flush(self):
        self.producer.flush()
```

File mongodb_connect.py

```
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure
from config.database_config import get_database_config
from bson import ObjectId

def clean_document(doc):
    # Convert ObjectId và các kiểu không JSON-serializable về
chuỗi
    for key, value in doc.items():
        if isinstance(value, ObjectId):
            doc[key] = str(value)
    return doc

class MongoDBConnect:
  def __init__(self, mongo_uri, db_name):
        self.mongo_uri = mongo_uri
        self.db_name = db_name
        self.client = None
        self.db = None


  def connect(self):
        try:
            self.client = MongoClient(self.mongo_uri)
            self.client.server_info()  # Test connection
            self.db = self.client[self.db_name]
            print(f"Connected to MongoDB: {self.db_name}")
            return self.db
        except ConnectionFailure as e:
            raise Exception(f"Failed to connect to MongoDB:
{e}") from e
```

```
    def close(self):
        if self.client:
            self.client.close()
            print("MongoDB connection closed")


    def __enter__(self):
        self.connect()
        return self


    def __exit__(self, exc_type, exc_val, exc_tb):
        self.close()

def query_product_views(db, kafka_producer=None):
    projection = {
        "_id": 1,
        "time_stamp": 1,
        "current_url": 1,
        "referrer_url": 1,
        "collection": 1,
        "product_id": 1,
        "option": 1
    }
    cursor = db.summary.find({}, projection)

    records_processed = 0

    try:
        for doc in cursor:
            records_processed += 1

            # Send to Kafka if producer is provided
            if kafka_producer:
                cleaned_doc = clean_document(doc)
                kafka_producer.send(cleaned_doc)

            if records_processed % 500000 == 0:
                print(f"Processed {records_processed} records")
    finally:
        cursor.close()

    print(f"Total records processed: {records_processed}")
    if kafka_producer:
```

```
      kafka_producer.flush()

  return records_processed
```

File schema_manager.py

```
from pymongo import MongoClient
from pymongo.errors import CollectionInvalid

def create_mongodb_schema(db):
   if "summary" not in db.list_collection_names():
       try:
           db.create_collection("summary", validator={
               "$jsonSchema": {
                   "bsonType": "object",
                   "required": ["_id", "time_stamp",
"current_url", "collection", "product_id"],
                   "properties": {
                       "_id": {"bsonType": ["objectId",
"string"]},
                       "time_stamp": {"bsonType": ["int",
"long"]},
                       "ip": {"bsonType": ["string", "null"]},
                       "user_agent": {"bsonType": ["string",
"null"]},
                       "resolution": {"bsonType": ["string",
"null"]},
                       "user_id_db": {"bsonType": ["string",
"null"]},
                       "device_id": {"bsonType": ["string",
"null"]},
                       "api_version": {"bsonType": ["string",
"null"]},
                       "store_id": {"bsonType": ["string",
"null"]},
                       "local_time": {"bsonType": ["string",
"null"]},
                       "show_recommendation": {"bsonType":
["string", "bool", "null"]},
                       "current_url": {"bsonType": "string"},
                       "referrer_url": {"bsonType": ["string",
"null"]},
```

```
                        "email_address": {"bsonType":
["string", "null"]},
                        "recommendation": {"bsonType": ["bool",
"null"]},
                        "utm_source": {"bsonType": ["bool",
"string", "null"]},
                        "utm_medium": {"bsonType": ["bool",
"string", "null"]},
                        "collection": {"bsonType": "string"},
                        "product_id": {"bsonType": "string"},
                        "option": {
                            "bsonType": ["array", "null"],
                            "items": {
                                "bsonType": "object",
                                "properties": {
                                    "option_label":
{"bsonType": ["string", "null"]},
                                    "option_id": {"bsonType":
["string", "null"]},
                                    "value_label": {"bsonType":
["string", "null"]},
                                    "value_id": {"bsonType":
["string", "null"]}
                                }
                            }
                        }
                    }
                })
            db.summary.create_index("product_id")
            print("Created summary collection with schema")
        except CollectionInvalid:
            print("summary collection already exists")
    else:
        print("summary collection already exists, skipping
creation")

def validate_mongodb_schema(db):
    collections = db.list_collection_names()
    print("Collections:", collections)
    if "summary" not in collections:
        raise ValueError("Missing summary collection in
MongoDB")
    if db.summary.find_one() is None:
```

```
        print("Warning: summary collection is empty")
    else:
        print("summary collection contains documents")
    print("Validated schema for summary collection")
```