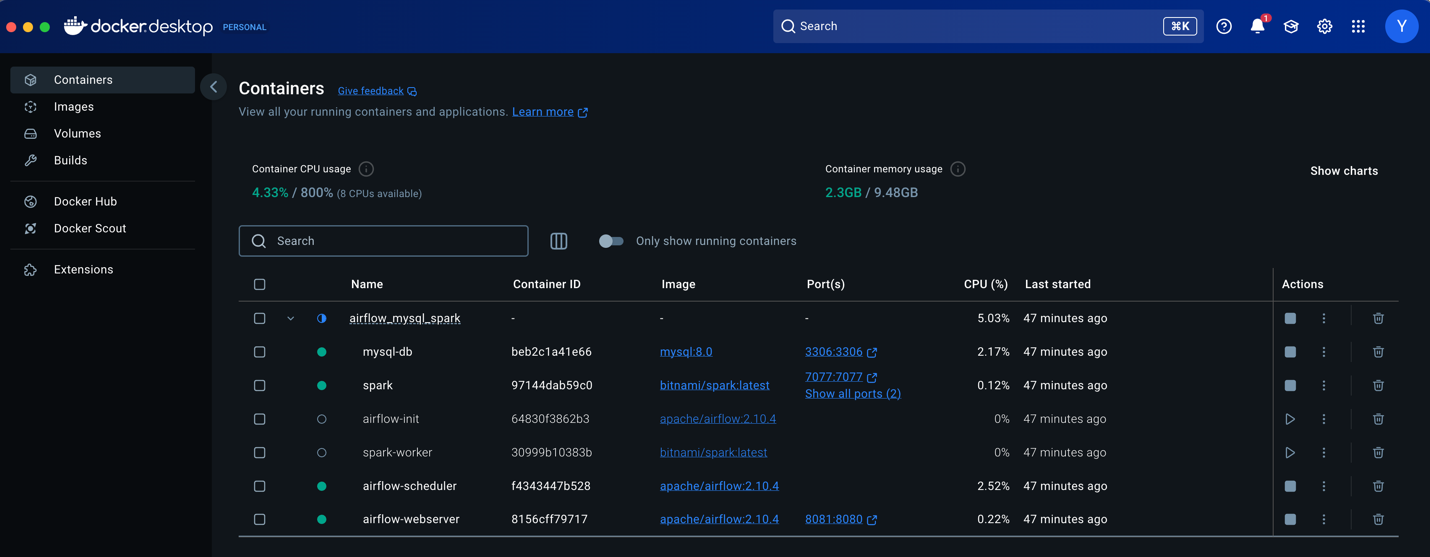
С помощью docker-compose.yaml развернуто окружения, включающее MySQL, Apache Airflow и Apache Spark. Он описывает взаимодействие между сервисами, настройку сетей, монтирование томов и переменные окружения.



**Описание файла s6dag.py**

Файл s6dag.py представляет собой **DAG (Directed Acyclic Graph)** для **Apache Airflow**, предназначенный для выполнения различных задач, связанных с **Apache Spark, Python, Scala и MySQL**. DAG запускает несколько задач, включая установку зависимостей, установку Python-библиотек, выполнение PySpark-скрипта, компиляцию Scala-кода и запуск Spark-приложения.

**1. Общая информация**

• **Название DAG**: Kostia001

• **Описание**: seminar\_6

• **Автор DAG**: Kostia

• **Дата начала работы**: 1 июня 2024

• **Периодичность запуска**: Каждый день в 06:00 (CRON: 0 6 \* \* \*)

• **Стратегия обработки пропущенных запусков**: catchup=False

• **Настройки повторных запусков**:

• Повторные попытки: 0

• Задержка перед повторной попыткой: 5 минут

• **Оповещения по почте**:

• Email: alex@alex.ru

• Ошибки: Нет

• Повторные попытки: Нет

**2. Описание задач (Tasks)**

**Task 1️ Установка системных зависимостей (task1: install\_dependencies)**

• **Используемый оператор**: BashOperator

• **Описание**: Устанавливает **Java, Scala, Curl, Telnet** в контейнере Spark.

• **Команда**:

• Обновляет пакеты.

• Устанавливает openjdk-17-jdk, scala, curl, telnet.

• Настраивает репозиторий AdoptOpenJDK.

• Дает разрешения на файлы в /opt/bitnami/spark/jars/.

**Task 2️ Установка Python-зависимостей (task2: pip\_install)**

• **Используемый оператор**: BashOperator

• **Описание**: Устанавливает Python-библиотеки внутри контейнера Spark.

• **Устанавливаемые пакеты**:

• cryptography

• pandas

• pymysql

• sqlalchemy

• pyspark

**Task 3️ Запуск PySpark-скрипта (task3: pyspark)**

• **Используемый оператор**: BashOperator

• **Описание**: Выполняет **Python-скрипт s6.py** внутри контейнера Spark.

• **Дополнительные проверки**:

• Если файл s6.py отсутствует, задача завершится с ошибкой.

**Task 4️ Компиляция Scala-кода (task4: scala)**

• **Используемый оператор**: BashOperator

• **Описание**: Компилирует **Scala-скрипт s6s1.scala** в JAR-файл.

• **Действия**:

• Проверяет наличие s6s1.scala.

• Компилирует с **опцией -J-Xmx4g (4GB памяти)**.

• Создает JAR-файл s6s1.jar.

• Если скрипт отсутствует, задача завершается с ошибкой.

**Task 5️ Запуск Spark-приложения (task5: run\_spark)**

• **Используемый оператор**: BashOperator

• **Описание**: Запускает Spark-приложение с использованием **Scala JAR-файла**.

• **Параметры Spark**:

• --driver-memory 4g

• --executor-memory 4g

• Подключает зависимости:

• com.crealytics:spark-excel\_2.12:3.5.1\_0.20.4

• mysql:mysql-connector-java:8.0.33

• Если JAR-файл отсутствует, задача завершится с ошибкой.

**3. Последовательность выполнения задач**

graph TD;

task1[Установка системных зависимостей] -->

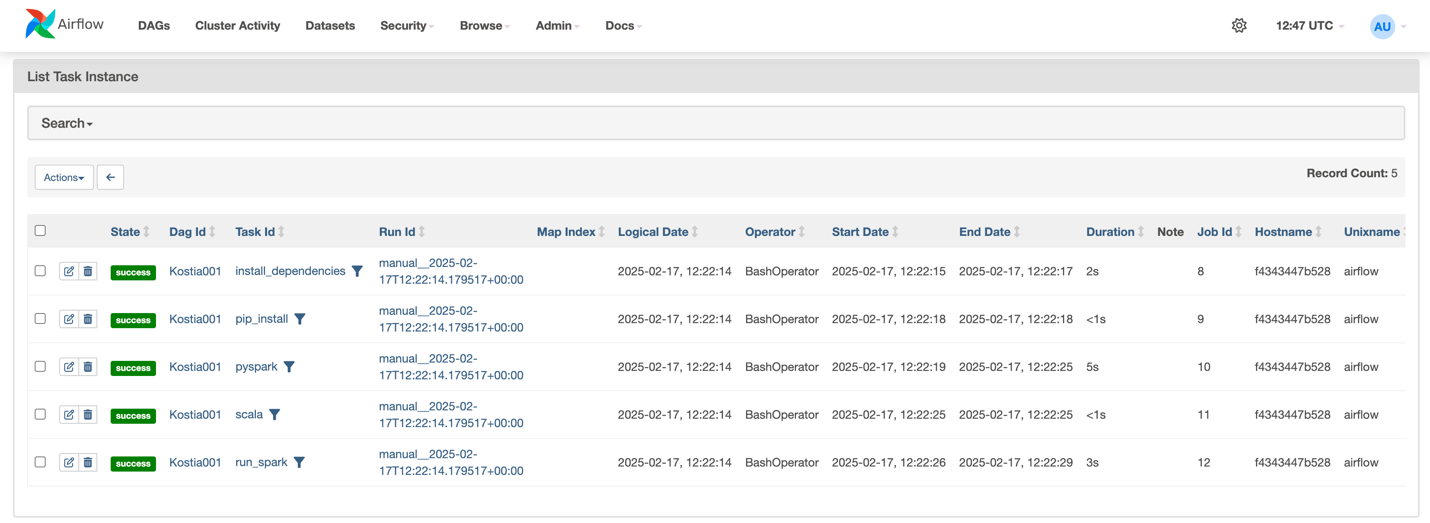
task2[Установка Python-библиотек] -->

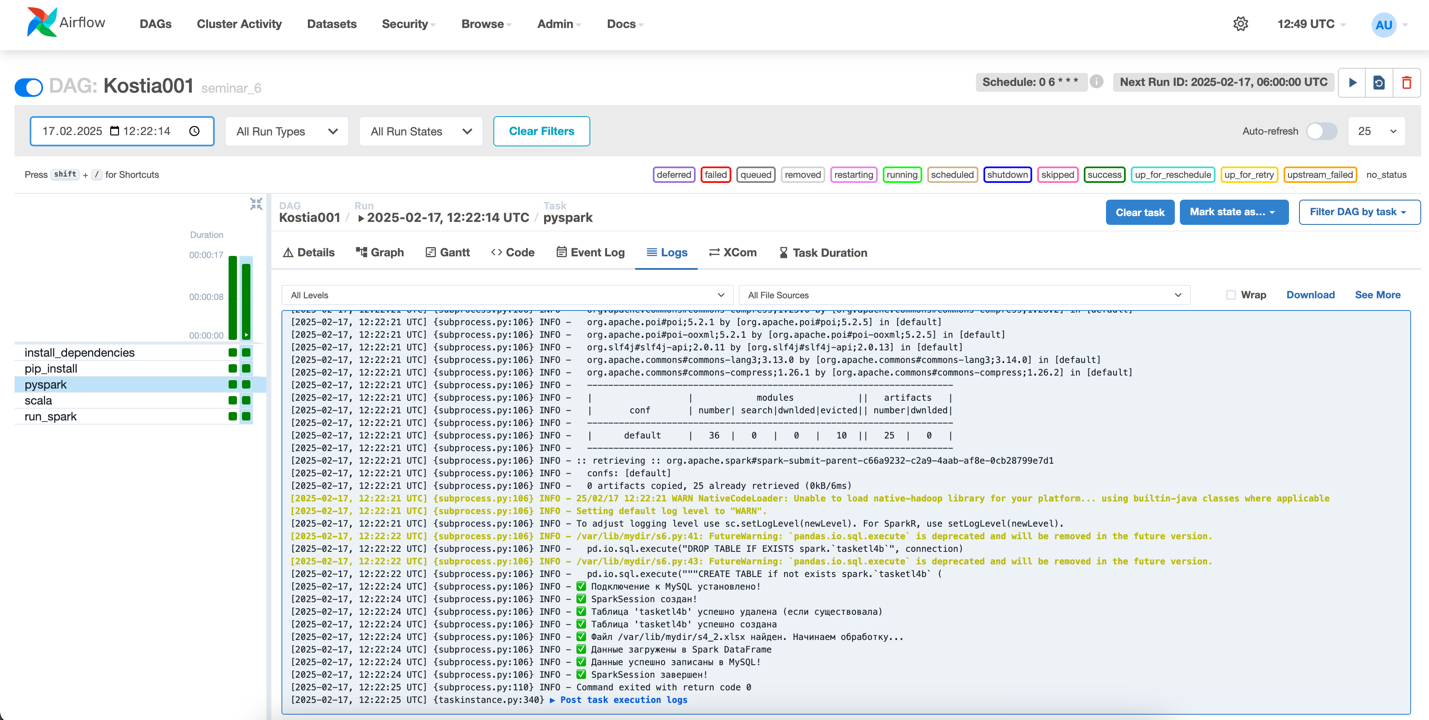
task3[Запуск PySpark-скрипта] -->

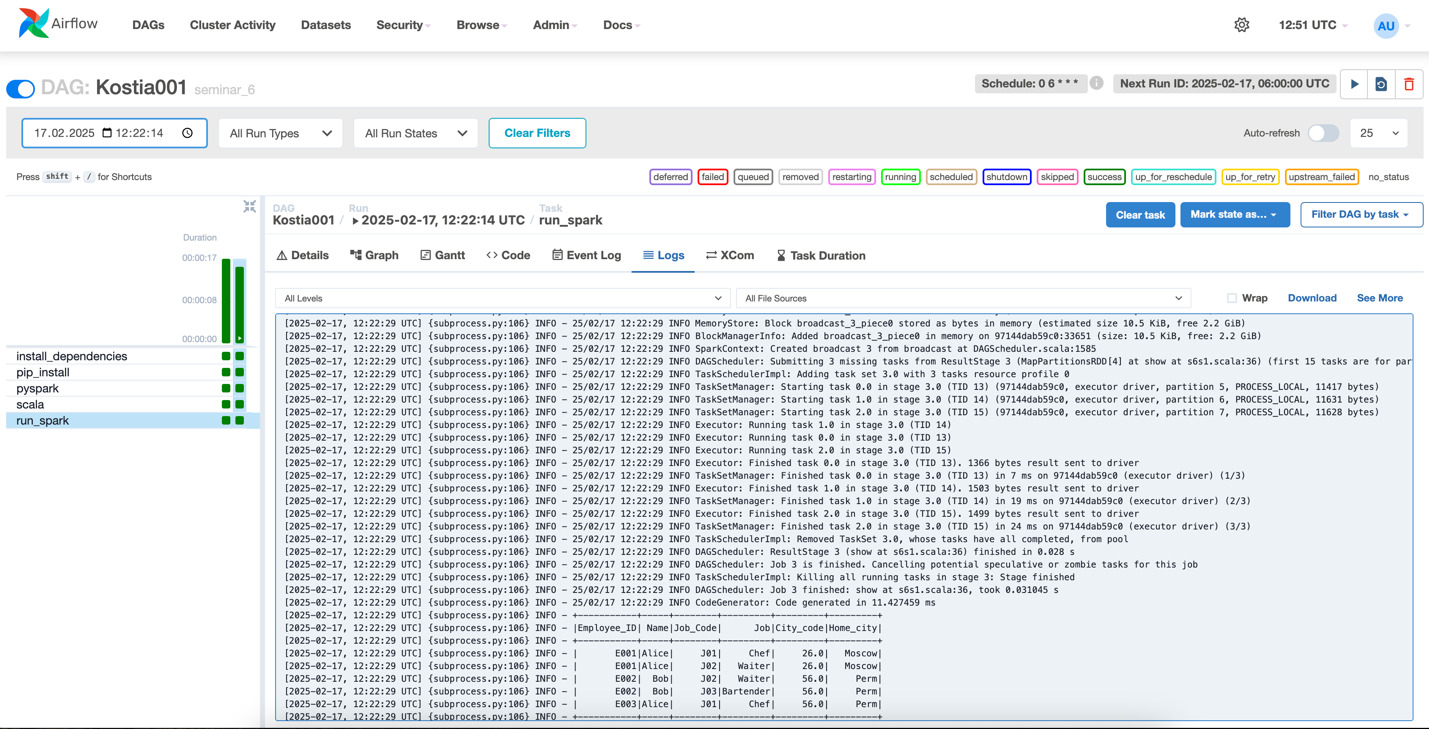
task4[Компиляция Scala-кода] -->

task5[Запуск Spark-приложения]

Этот DAG автоматизирует процесс установки зависимостей, выполнения PySpark-скрипта, компиляции Scala-кода и запуска Spark-приложения в контейнере. Он предназначен для **ETL-задач, анализа данных и работы с MySQL в Apache Spark**.







**s6dag.py**

from airflow import DAG  
from airflow.operators.bash import BashOperator  
from datetime import datetime, timedelta  
  
default\_args = {  
 'owner': 'Kostia',  
 'depends\_on\_past': False,  
 'start\_date': datetime(2024, 6, 1),  
 'email': ['alex@alex.ru'],  
 'email\_on\_failure': False,  
 'email\_on\_retry': False,  
 'retries': 0,  
 'retry\_delay': timedelta(minutes=5)  
}  
  
dag1 = DAG('Kostia001',  
 default\_args=default\_args,  
 description="seminar\_6",  
 catchup=False,  
 schedule\_interval='0 6 \* \* \*')  
  
# Устанавливаем все системные зависимости  
task1 = BashOperator(  
 task\_id='install\_dependencies',  
 bash\_command="""  
 docker exec -u root spark sh -c "  
 apt-get update && \  
 apt-get install -y gnupg curl && \  
 curl -fsSL https://packages.adoptium.net/artifactory/api/gpg/key/public | gpg --dearmor > /usr/share/keyrings/adoptium-keyring.gpg && \  
 echo 'deb [signed-by=/usr/share/keyrings/adoptium-keyring.gpg] https://packages.adoptium.net/artifactory/deb bookworm main' > /etc/apt/sources.list.d/adoptium.list && \  
 apt-get update && \  
 DEBIAN\_FRONTEND=noninteractive apt-get install -y openjdk-17-jdk scala curl telnet && \  
 chmod -R 755 /opt/bitnami/spark/jars/  
 "  
 """,  
 dag=dag1  
)  
  
# Устанавливаем Python-библиотеки  
task2 = BashOperator(  
 task\_id='pip\_install',  
 bash\_command="""  
 docker exec spark sh -c "  
 pip install --no-cache-dir cryptography pandas pymysql sqlalchemy pyspark  
 "  
 """,  
 dag=dag1  
)  
  
# Запускаем Python-скрипт PySpark  
task3 = BashOperator(  
 task\_id='pyspark',  
 bash\_command="""  
 docker exec spark sh -c "  
 if test -f /var/lib/mydir/s6.py; then \  
 python /var/lib/mydir/s6.py; \  
 else \  
 echo 's6.py not found!' && exit 1; \  
 fi  
 "  
 """,  
 dag=dag1  
)  
  
task4 = BashOperator(  
 task\_id='scala',  
 bash\_command="""  
 docker exec spark sh -c "  
 # export JAVA\_HOME=/opt/bitnami/java && \  
 # export PATH=$JAVA\_HOME/bin:$PATH && \  
 echo 'Using JAVA\_HOME='$JAVA\_HOME && \  
 javac -version && \  
 cd /var/lib/mydir && \  
 if test -f s6s1.scala; then \  
 scalac -encoding UTF-8 -J-Xmx4g -J-Xms2g -classpath '/opt/bitnami/spark/jars/\*' -d s6s1.jar s6s1.scala && \  
 jar cf s6s1.jar \*.class && \  
 echo '✅ Scala script compiled successfully.'; \  
 else \  
 echo '❌ Scala script not found!' && exit 1; \  
 fi  
 "  
 """,  
 dag=dag1  
)  
  
task5 = BashOperator(  
 task\_id='run\_spark',  
 bash\_command="""  
 docker exec spark sh -c "  
 cd /var/lib/mydir && \  
 if test -f s6s1.jar; then \  
 /opt/bitnami/spark/bin/spark-submit --driver-memory 4g --executor-memory 4g \  
 --class MySQLExample \  
 --packages com.crealytics:spark-excel\_2.12:3.5.1\_0.20.4,mysql:mysql-connector-java:8.0.33 \  
 /var/lib/mydir/s6s1.jar; \  
 else \  
 echo 'Scala JAR file not found!' && exit 1; \  
 fi  
 "  
 """,  
 dag=dag1  
)  
  
task1 >> task2 >> task3 >> task4 >> task5

------------------------------------------------

**s6.py**

import sys  
import os  
import pandas as pd  
from pandas.io import sql  
from pyspark.sql.functions import col, lit, to\_date  
from pyspark.sql.session import SparkSession  
from sqlalchemy import create\_engine  
from pyspark.sql.window import Window  
from pyspark.sql.functions import sum as sum1  
  
# Настройки подключения к MySQL  
MYSQL\_USER = "airflow"  
MYSQL\_PASSWORD = "airflow"  
MYSQL\_HOST = "mysql-db" # Имя контейнера MySQL  
MYSQL\_DB = "spark"  
MYSQL\_JDBC\_DRIVER = "com.mysql.cj.jdbc.Driver"  
  
# Подключение к MySQL через SQLAlchemy  
try:  
 engine = create\_engine(f"mysql+pymysql://{MYSQL\_USER}:{MYSQL\_PASSWORD}@{MYSQL\_HOST}/{MYSQL\_DB}")  
 connection = engine.connect()  
 print("✅ Подключение к MySQL установлено!")  
except Exception as e:  
 print(f"❌ Ошибка подключения к MySQL: {e}")  
 sys.exit(1)  
  
# Создание SparkSession с загрузкой необходимых JAR-пакетов  
spark = SparkSession.builder \  
 .appName("Excel Processing") \  
 .config("spark.jars.packages",  
 "com.crealytics:spark-excel\_2.12:3.5.1\_0.20.4,mysql:mysql-connector-java:8.0.33") \  
 .config("spark.driver.extraClassPath", "/opt/bitnami/spark/jars/mysql-connector-java-8.0.33.jar") \  
 .config("spark.executor.extraClassPath", "/opt/bitnami/spark/jars/mysql-connector-java-8.0.33.jar") \  
 .getOrCreate()  
  
print("✅ SparkSession создан!")  
  
# Пример SQL-запроса  
try:  
 with engine.connect() as connection:  
 pd.io.sql.execute("DROP TABLE IF EXISTS spark.`tasketl4b`", connection)  
 print("✅ Таблица 'tasketl4b' успешно удалена (если существовала)")  
 pd.io.sql.execute("""CREATE TABLE if not exists spark.`tasketl4b` (  
 `№` INT(10) NULL DEFAULT NULL,  
 `Месяц` DATE NULL DEFAULT NULL,  
 `Сумма платежа` FLOAT NULL DEFAULT NULL,  
 `Платеж по основному долгу` FLOAT NULL DEFAULT NULL,  
 `Платеж по процентам` FLOAT NULL DEFAULT NULL,  
 `Остаток долга` FLOAT NULL DEFAULT NULL,  
 `проценты` FLOAT NULL DEFAULT NULL,  
 `долг` FLOAT NULL DEFAULT NULL  
 )  
 COLLATE='utf8mb4\_0900\_ai\_ci'  
 ENGINE=InnoDB""", connection)  
 print("✅ Таблица 'tasketl4b' успешно создана")  
except Exception as e:  
 print(f"❌ Ошибка при создании таблицы: {e}")  
  
# Окно для вычисления суммарных значений  
w = Window.partitionBy(lit(1)).orderBy("№").rowsBetween(Window.unboundedPreceding, Window.currentRow)  
  
# Проверка наличия Excel-файла  
excel\_path = "/var/lib/mydir/s4\_2.xlsx"  
  
if os.path.exists(excel\_path):  
 print(f"✅ Файл {excel\_path} найден. Начинаем обработку...")  
 try:  
 df1 = spark.read.format("com.crealytics.spark.excel") \  
 .option("sheetName", "Sheet1") \  
 .option("useHeader", "false") \  
 .option("treatEmptyValuesAsNulls", "false") \  
 .option("inferSchema", "true") \  
 .option("addColorColumns", "true") \  
 .option("usePlainNumberFormat", "true") \  
 .option("startColumn", 0) \  
 .option("endColumn", 99) \  
 .option("timestampFormat", "MM-dd-yyyy HH:mm:ss") \  
 .option("maxRowsInMemory", 20) \  
 .option("excerptSize", 10) \  
 .option("header", "true") \  
 .format("excel") \  
 .load(excel\_path).limit(1000) \  
 .withColumn("проценты", sum1(col("Платеж по процентам")).over(w)) \  
 .withColumn("долг", sum1(col("Платеж по основному долгу")).over(w))  
  
 df1 = df1.withColumn("Месяц", to\_date(col("Месяц"))) # Приведение типа к `DATE`  
 print("✅ Данные загружены в Spark DataFrame")  
  
 except Exception as e:  
 print(f"❌ Ошибка при загрузке Excel файла: {e}")  
else:  
 print(f"❌ Файл {excel\_path} не найден!")  
 sys.exit(1)  
  
# Запись данных в MySQL  
try:  
 df1.write \  
 .format("jdbc") \  
 .option("url", f"jdbc:mysql://{MYSQL\_HOST}/{MYSQL\_DB}") \  
 .option("dbtable", "tasketl4b") \  
 .option("user", MYSQL\_USER) \  
 .option("password", MYSQL\_PASSWORD) \  
 .option("driver", MYSQL\_JDBC\_DRIVER) \  
 .mode("overwrite") \  
 .save()  
  
 print("✅ Данные успешно записаны в MySQL!")  
  
except Exception as e:  
 print(f"❌ Ошибка при записи в MySQL: {e}")  
  
# Завершаем сессию  
spark.stop()  
print("✅ SparkSession завершен!")

------------------------------------------------

**s6s1.scala**

/\*  
spark-submit --packages com.crealytics:spark-excel\_2.12:3.5.1\_0.20.4,mysql:mysql-connector-java:8.0.33 /var/lib/mydir/d1.scala  
  
Конфигурация докер "/Users/kostia/Programs:/var/lib/mydir:rw"  
\*/  
  
import org.apache.spark.sql.{DataFrame, SparkSession}  
import org.apache.spark.sql.functions.\_  
import java.util.Properties  
  
object MySQLExample {  
 def main(args: Array[String]): Unit = {  
 // Создаем SparkSession  
 val spark = SparkSession.builder()  
 .appName("MySQLExample")  
 .config("spark.driver.extraJavaOptions", "-Dfile.encoding=UTF-8")  
 .getOrCreate()  
  
 // Параметры подключения к MySQL  
 val dbUrl = "jdbc:mysql://mysql-db:3306/spark?serverTimezone=UTC"  
 val dbUser = "airflow"  
 val dbPassword = "airflow"  
 val dbDriver = "com.mysql.cj.jdbc.Driver"  
  
 // Читаем Excel-файл  
 val df1 = spark.read  
 .format("com.crealytics.spark.excel")  
 .option("sheetName", "Лист1")  
 .option("useHeader", "true")  
 .option("header", "true")  
 .option("treatEmptyValuesAsNulls", "false")  
 .option("inferSchema", "true")  
 .option("startColumn", 0)  
 .option("endColumn", 99)  
 .option("timestampFormat", "MM-dd-yyyy HH:mm:ss")  
 .load("/var/lib/mydir/d1.xlsx")  
  
 df1.show()  
  
 // Настройки для соединения с MySQL  
 val connectionProperties = new Properties()  
 connectionProperties.put("user", dbUser)  
 connectionProperties.put("password", dbPassword)  
 connectionProperties.put("driver", dbDriver)  
  
 // Функция для записи DataFrame в MySQL  
 def writeToMySQL(df: DataFrame, tableName: String): Unit = {  
 try {  
 df.write  
 .mode("append")  
 .jdbc(dbUrl, tableName, connectionProperties)  
 println(s"✅ Данные успешно записаны в таблицу $tableName.")  
 } catch {  
 case e: Exception =>  
 println(s"❌ Ошибка при записи в таблицу $tableName: ${e.getMessage}")  
 }  
 }  
  
 // Запись данных в таблицы MySQL  
 try {  
 println("📊 Загружаем данные в MySQL...")  
  
 writeToMySQL(df1.select("City\_code", "Home\_city").distinct(), "tabr1")  
 writeToMySQL(df1.select("Job\_Code", "Job").distinct(), "tabr2")  
 writeToMySQL(df1.select("Employee\_ID", "Name", "Job\_Code").distinct(), "tabr3")  
 writeToMySQL(df1.select("Employee\_ID", "Name", "City\_code").distinct(), "tabr4")  
  
 println("✅ Данные успешно сохранены в MySQL.")  
  
 } catch {  
 case e: Exception =>  
 println(s"❌ Ошибка при работе с MySQL: ${e.getMessage}")  
 } finally {  
 // Закрываем SparkSession  
 println("🛑 Остановка SparkSession...")  
 spark.stop()  
 }  
 }  
}