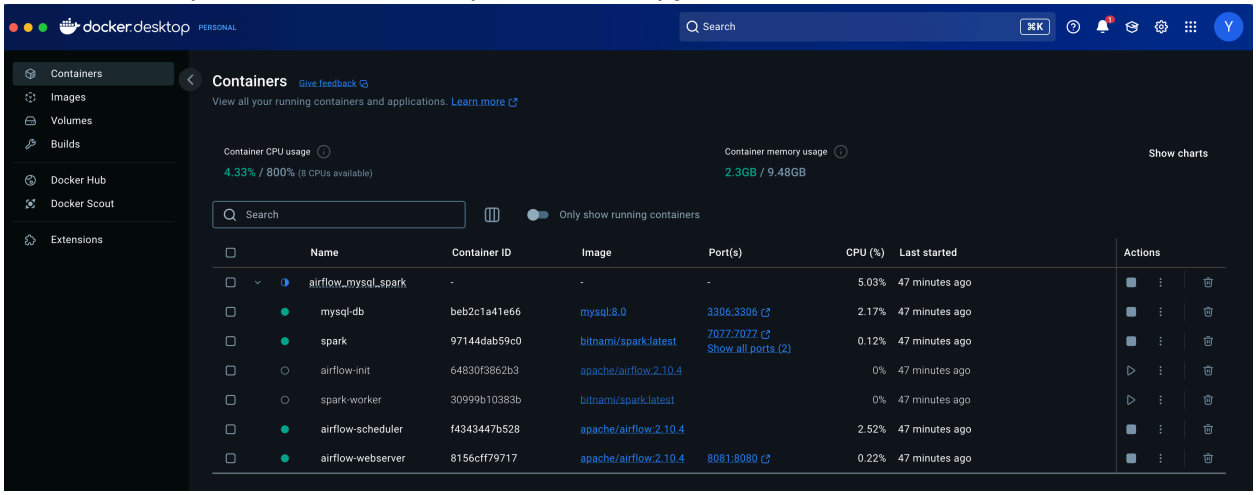
С помощью docker-compose.yaml развернуто окружения, включающее MySQL, Apache Airflow и Apache Spark. Он описывает взаимодействие между сервисами, настройку сетей, монтирование томов и переменные окружения.



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 1 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 2 Установка Python-зависимостей (task2: pip_install)

• **Используемый оператор**: BashOperator
• **Описание**: Устанавливает Python-библиотеки внутри контейнера Spark.
• **Устанавливаемые пакеты**:

• cryptography
• pandas
• pymysql
• sqlalchemy
• pyspark

Task 3 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.

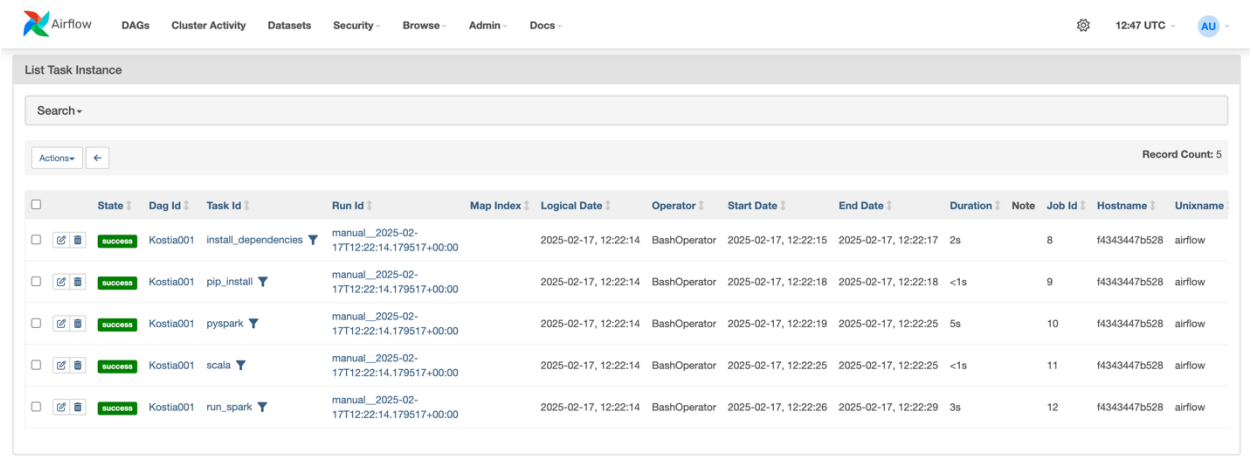
Task 4 Компиляция Scala-кода (task4: scala)
• **Используемый оператор**: BashOperator
• **Описание**: Компилирует **Scala-скрипт s6s1.scala** в JAR-файл.
• **Действия**:
• Проверяет наличие s6s1.scala.
• Компилирует с **опцией -J-Xmx4g (4GB памяти)**.
• Создает JAR-файл s6s1.jar.
• Если скрипт отсутствует, задача завершается с ошибкой.

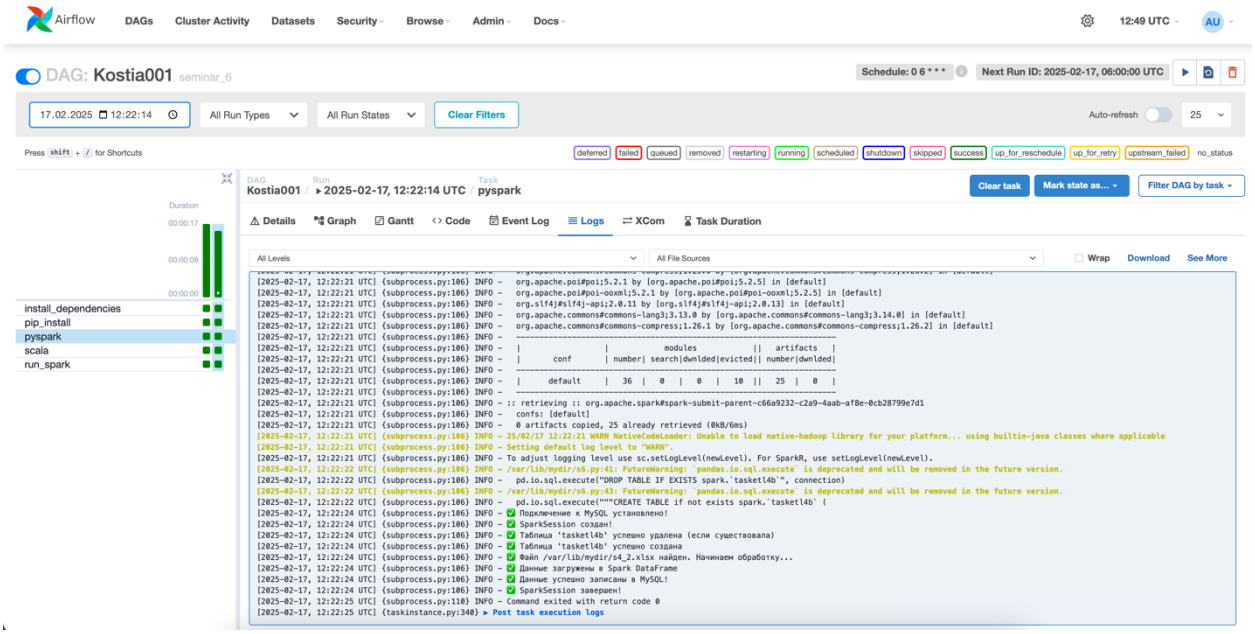
Task 5 Запуск Spark-приложения (task5: run_spark)
• **Используемый оператор**: BashOperator
• **Описание**: Запускает Spark-приложение с использованием **Scala JAR-файла**.
• **Параметры Spark**:
• --driver-memory 4g
• --executor-memory 4g
• Подключает зависимости:
• com.crealytics:spark-excel_2.12:3.5.1_0.20.4
• mysql:mysql-connector-java:8.0.33
• Если JAR-файл отсутствует, задача завершится с ошибкой.

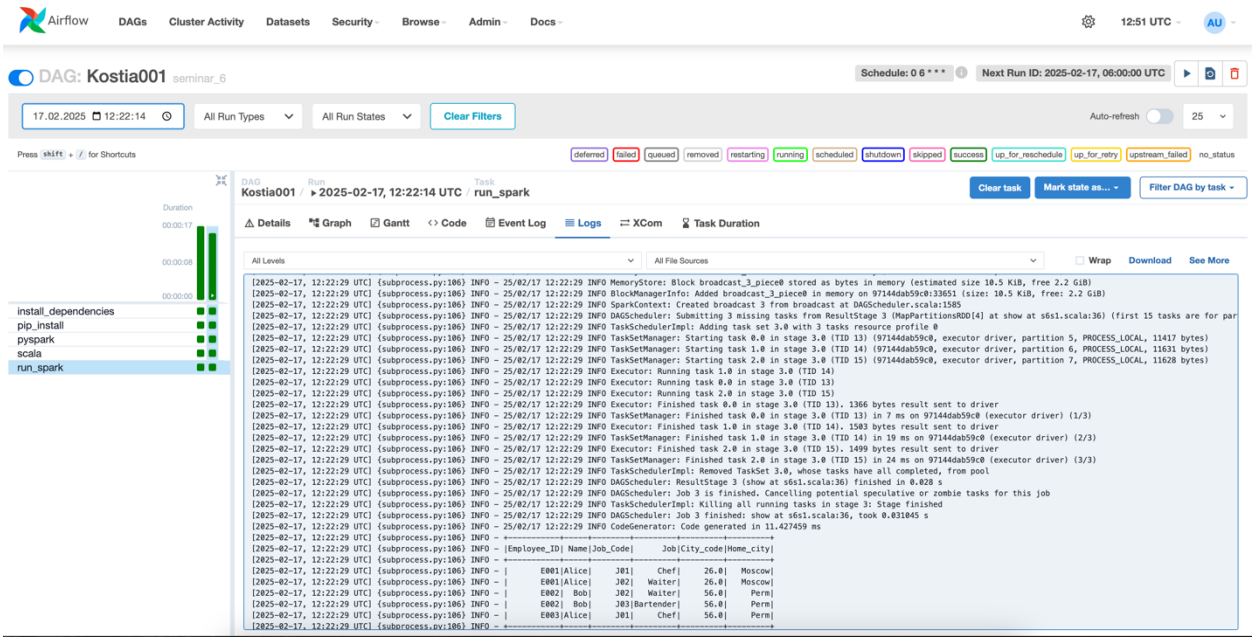
3. Последовательность выполнения задач

```
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**.


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
12:47 UTC
AU
List Task Instance
Search
Actions
Record Count: 5
State
Dag Id
Task Id
Run Id
Map Index
Logical Date
Operator
Start Date
End Date
Duration
Note
Job Id
Hostname
Unixname
success Kostia001 install_dependencies manual__2025-02-17T12:22:14.179517+00:00 2025-02-17, 12:22:14 BashOperator 2025-02-17, 12:22:15 2025-02-17, 12:22:17 2s 8 f4343447b528 airflow
success Kostia001 pip_install manual__2025-02-17T12:22:14.179517+00:00 2025-02-17, 12:22:14 BashOperator 2025-02-17, 12:22:18 2025-02-17, 12:22:18 <1s 9 f4343447b528 airflow
success Kostia001 pyspark manual__2025-02-17T12:22:14.179517+00:00 2025-02-17, 12:22:14 BashOperator 2025-02-17, 12:22:19 2025-02-17, 12:22:25 5s 10 f4343447b528 airflow
success Kostia001 scala manual__2025-02-17T12:22:14.179517+00:00 2025-02-17, 12:22:14 BashOperator 2025-02-17, 12:22:25 2025-02-17, 12:22:25 <1s 11 f4343447b528 airflow
success Kostia001 run_spark manual__2025-02-17T12:22:14.179517+00:00 2025-02-17, 12:22:14 BashOperator 2025-02-17, 12:22:26 2025-02-17, 12:22:29 3s 12 f4343447b528 airflow


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
12:49 UTC
AU
DAG: Kostia001 seminar_6
Schedule: 0 6 * * *
Next Run ID: 2025-02-17, 06:00:00 UTC
17.02.2025 12:22:14
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status
DAG Kostia001 / Run 2025-02-17, 12:22:14 UTC / Task pyspark
Clear task
Mark state as...
Filter DAG by task
Details Graph Gantt Code Event Log Logs XCom Task Duration
All Levels
All File Sources
Wrap
Download
See More
Duration
install_dependencies
pip_install
pyspark
scala
run_spark
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - org.apache.poi#poi;5.2.1 by [org.apache.poi#poi;5.2.5] in [default]
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - org.apache.poi#poi-ooxml;5.2.1 by [org.apache.poi#poi-ooxml;5.2.5] in [default]
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - org.slf4j#slf4j-api;2.0.11 by [org.slf4j#slf4j-api;2.0.13] in [default]
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - org.apache.commons#commons-lang3;3.13.0 by [org.apache.commons#commons-lang3;3.14.0] in [default]
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - org.apache.commons#commons-compress;1.26.1 by [org.apache.commons#commons-compress;1.26.2] in [default]
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - ---
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - | | modules || artifacts |
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - | conf | number| search|dwnlded|evicted|| number|dwnlded|
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - ---
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - | default | 36 | 0 | 0 | 10 || 25 | 0 |
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - ---
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - :: retrieving :: org.apache.spark#spark-submit-parent-c66a9232-c2a9-4aab-af8e-0cb28799e7d1
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - confs: [default]
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - 0 artifacts copied, 25 already retrieved (0kB/6ms)
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - Setting default log level to "WARN".
[2025-02-17, 12:22:21 UTC] {subprocess.py:106} INFO - To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
[2025-02-17, 12:22:22 UTC] {subprocess.py:106} INFO - /var/lib/mydir/s6.py:41: FutureWarning: `pandas.io.sql.execute` is deprecated and will be removed in the future version.
[2025-02-17, 12:22:22 UTC] {subprocess.py:106} INFO - pd.io.sql.execute("DROP TABLE IF EXISTS spark.`tasketl4b`", connection)
[2025-02-17, 12:22:22 UTC] {subprocess.py:106} INFO - /var/lib/mydir/s6.py:43: FutureWarning: `pandas.io.sql.execute` is deprecated and will be removed in the future version.
[2025-02-17, 12:22:22 UTC] {subprocess.py:106} INFO - pd.io.sql.execute("""CREATE TABLE if not exists spark.`tasketl4b` (
[2025-02-17, 12:22:24 UTC] {subprocess.py:106} INFO - ✅ Подключение к MySQL установлено!
[2025-02-17, 12:22:24 UTC] {subprocess.py:106} INFO - ✅ SparkSession создан!
[2025-02-17, 12:22:24 UTC] {subprocess.py:106} INFO - ✅ Таблица 'tasketl4b' успешно удалена (если существовала)
[2025-02-17, 12:22:24 UTC] {subprocess.py:106} INFO - ✅ Таблица 'tasketl4b' успешно создана
[2025-02-17, 12:22:24 UTC] {subprocess.py:106} INFO - ✅ Файл /var/lib/mydir/s4_2.xlsx найден. Начинаем обработку...
[2025-02-17, 12:22:24 UTC] {subprocess.py:106} INFO - ✅ Данные загружены в Spark DataFrame
[2025-02-17, 12:22:24 UTC] {subprocess.py:106} INFO - ✅ Данные успешно записаны в MySQL!
[2025-02-17, 12:22:24 UTC] {subprocess.py:106} INFO - ✅ SparkSession завершен!
[2025-02-17, 12:22:25 UTC] {subprocess.py:110} INFO - Command exited with return code 0
[2025-02-17, 12:22:25 UTC] {taskinstance.py:340} ▶ Post task execution logs


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
12:51 UTC
AU
DAG: Kostia001 seminar_6
Schedule: 0 6 * * *
Next Run ID: 2025-02-17, 06:00:00 UTC
17.02.2025 12:22:14
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status
DAG Kostia001 / Run 2025-02-17, 12:22:14 UTC / Task run_spark
Clear task
Mark state as...
Filter DAG by task
Details Graph Gantt Code Event Log Logs XCom Task Duration
All Levels
All File Sources
Wrap
Download
See More
Duration
install_dependencies
pip_install
pyspark
scala
run_spark
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO MemoryStore: Block broadcast_3_piece0 stored as bytes in memory (estimated size 10.5 KiB, free 2.2 GiB)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 97144dab59c0:33651 (size: 10.5 KiB, free: 2.2 GiB)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO SparkContext: Created broadcast 3 from broadcast at DAGScheduler.scala:1585
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO DAGScheduler: Submitting 3 missing tasks from ResultStage 3 (MapPartitionsRDD[4] at show at s6s1.scala:36) (first 15 tasks are for par
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSchedulerImpl: Adding task set 3.0 with 3 tasks resource profile 0
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSetManager: Starting task 0.0 in stage 3.0 (TID 13) (97144dab59c0, executor driver, partition 5, PROCESS_LOCAL, 11417 bytes)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSetManager: Starting task 1.0 in stage 3.0 (TID 14) (97144dab59c0, executor driver, partition 6, PROCESS_LOCAL, 11631 bytes)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSetManager: Starting task 2.0 in stage 3.0 (TID 15) (97144dab59c0, executor driver, partition 7, PROCESS_LOCAL, 11628 bytes)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO Executor: Running task 1.0 in stage 3.0 (TID 14)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO Executor: Running task 0.0 in stage 3.0 (TID 13)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO Executor: Running task 2.0 in stage 3.0 (TID 15)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO Executor: Finished task 0.0 in stage 3.0 (TID 13). 1366 bytes result sent to driver
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 13) in 7 ms on 97144dab59c0 (executor driver) (1/3)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO Executor: Finished task 1.0 in stage 3.0 (TID 14). 1503 bytes result sent to driver
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSetManager: Finished task 1.0 in stage 3.0 (TID 14) in 19 ms on 97144dab59c0 (executor driver) (2/3)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO Executor: Finished task 2.0 in stage 3.0 (TID 15). 1499 bytes result sent to driver
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSetManager: Finished task 2.0 in stage 3.0 (TID 15) in 24 ms on 97144dab59c0 (executor driver) (3/3)
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO DAGScheduler: ResultStage 3 (show at s6s1.scala:36) finished in 0.028 s
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO TaskSchedulerImpl: Killing all running tasks in stage 3: Stage finished
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO DAGScheduler: Job 3 finished: show at s6s1.scala:36, took 0.031045 s
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - 25/02/17 12:22:29 INFO CodeGenerator: Code generated in 11.427459 ms
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - +-----------+-----+--------+----------+---------+
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - |Employee_ID| Name|Job_Code| Job|City_code|Home_city|
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - +-----------+-----+--------+----------+---------+
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - | E001|Alice| J01| Chef| 26.0| Moscow|
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - | E001|Alice| J02| Waiter| 26.0| Moscow|
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - | E002| Bob| J02| Waiter| 56.0| Perm|
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - | E002| Bob| J03|Bartender| 56.0| Perm|
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - | E003|Alice| J01| Chef| 56.0| Perm|
[2025-02-17, 12:22:29 UTC] {subprocess.py:106} INFO - +-----------+-----+--------+----------+---------+

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}/{M
YSQL_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()
    }
  }
}
```