**Задание 1:**

Для выполнения первого задания был выбран следующий датасет: <https://www.kaggle.com/datasets/asaniczka/public-opinion-on-climate-change-updated-daily>.

Перед загрузкой данных из него в YDB, в ней была создана таблица (**Листинг 1**). Для загрузки данных в таблицу YDB была использована команда ***ydb import file csv -p climate\_change\_comments -i file.csv***. После загрузки данных (Рисунок 1) были созданы эндпоинты трансфера (Рисунок 2) и после был выполнен сам трансфер (Рисунок 3-4).

Листинг 1

|  |
| --- |
| CREATE TABLE climate\_change\_comments (  `comment\_id` String NOT NULL,  `score` Int32,  `self\_text` String,  `subreddit` String,  `created\_time` String,  `post\_id` String NOT NULL,  `author\_name` String,  `controversiality` String,  `ups` String,  `downs` String,  `user\_is\_verified` String,  `user\_account\_created\_time` String,  `user\_awardee\_karma` String,  `user\_awarder\_karma` String,  `user\_link\_karma` String,  `user\_comment\_karma` String,  `user\_total\_karma` String,  `post\_score` String,  `post\_self\_text` String,  `post\_title` String,  `post\_upvote\_ratio` String,  `post\_thumbs\_ups` String,  `post\_total\_awards\_received` String,  `post\_created\_time` String,  INDEX `idx\_comments` GLOBAL ON (`created\_time`),  INDEX `idx\_posts` GLOBAL ON (`post\_created\_time`),  PRIMARY KEY (`comment\_id`)  ); |

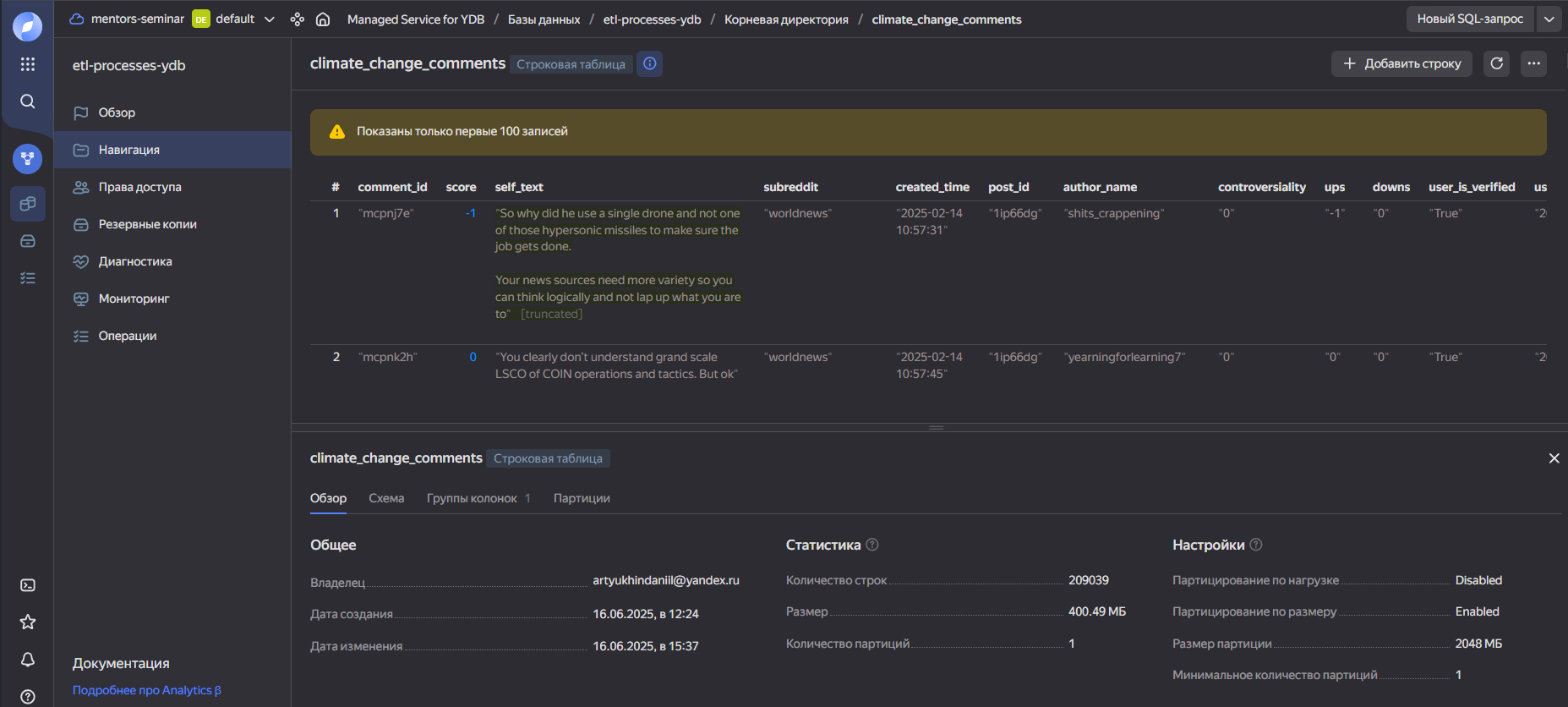


Рисунок 1

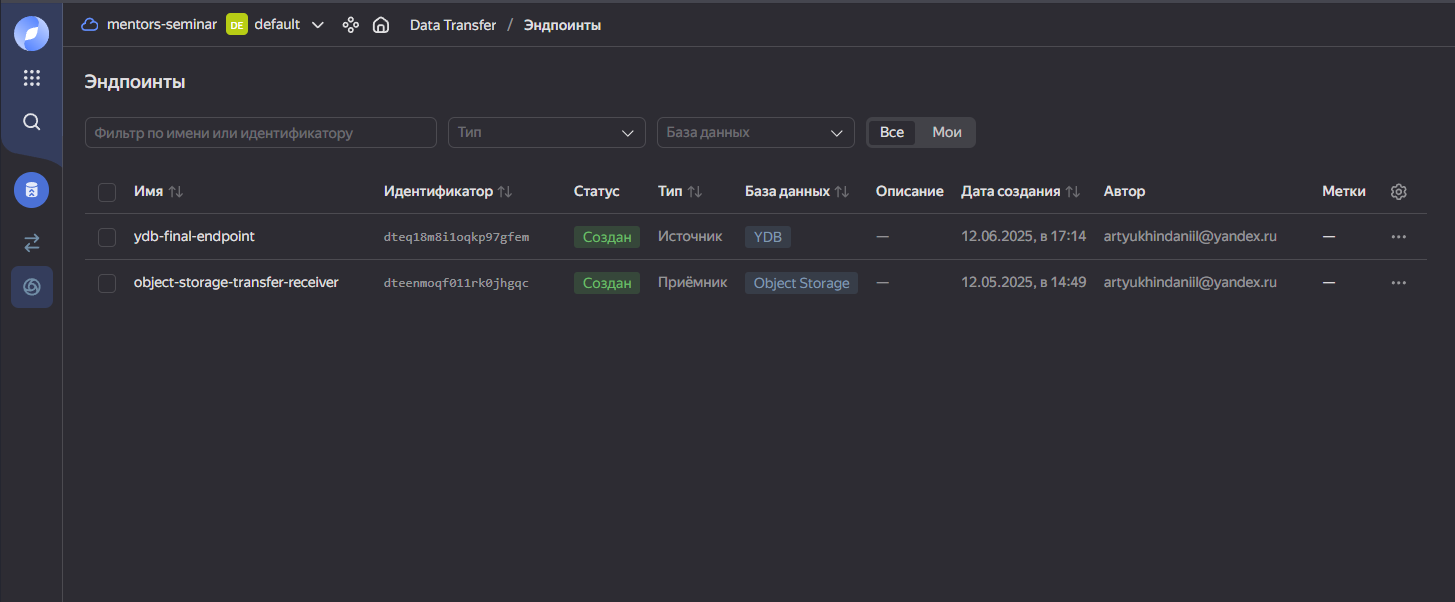


Рисунок 2

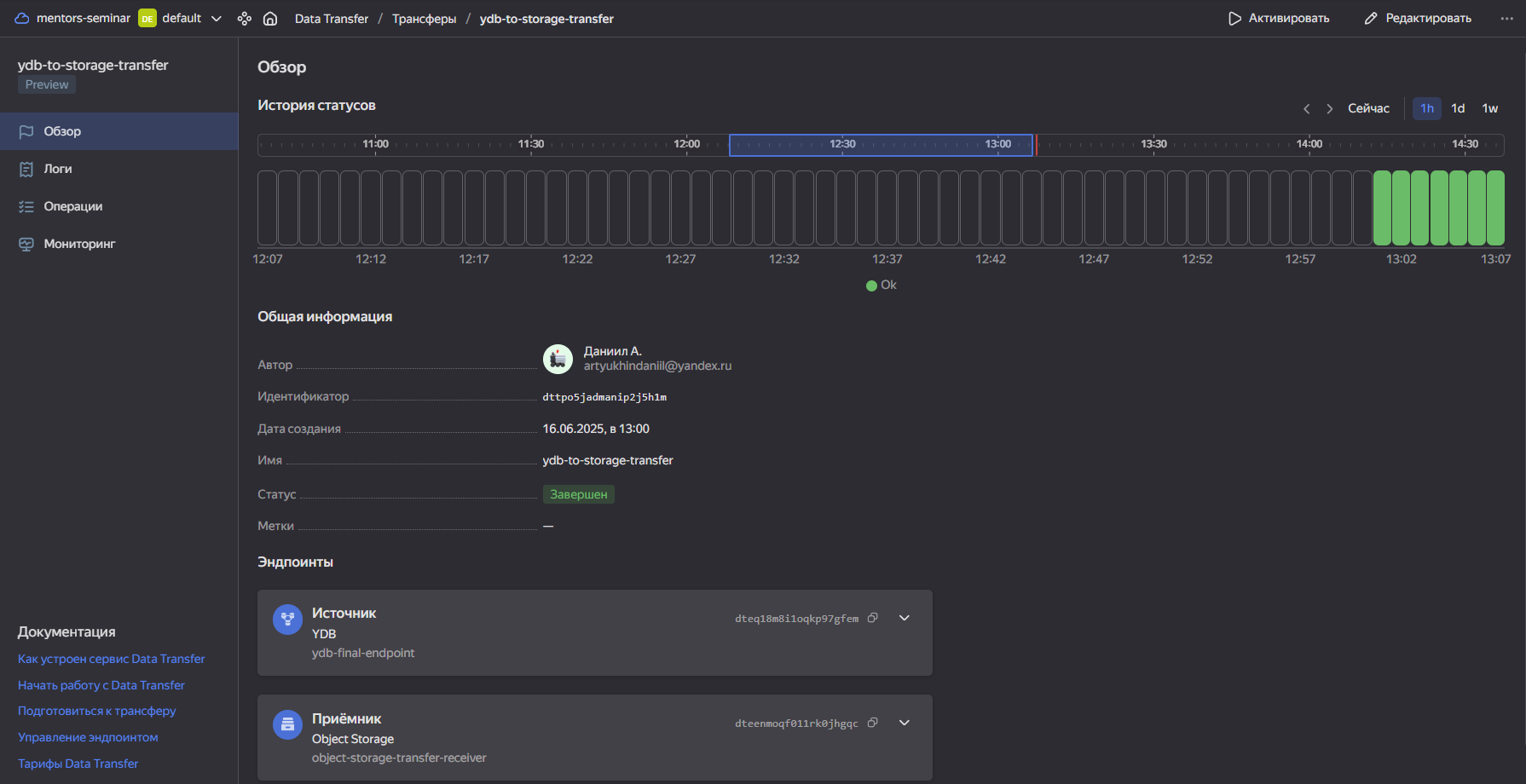


Рисунок 3

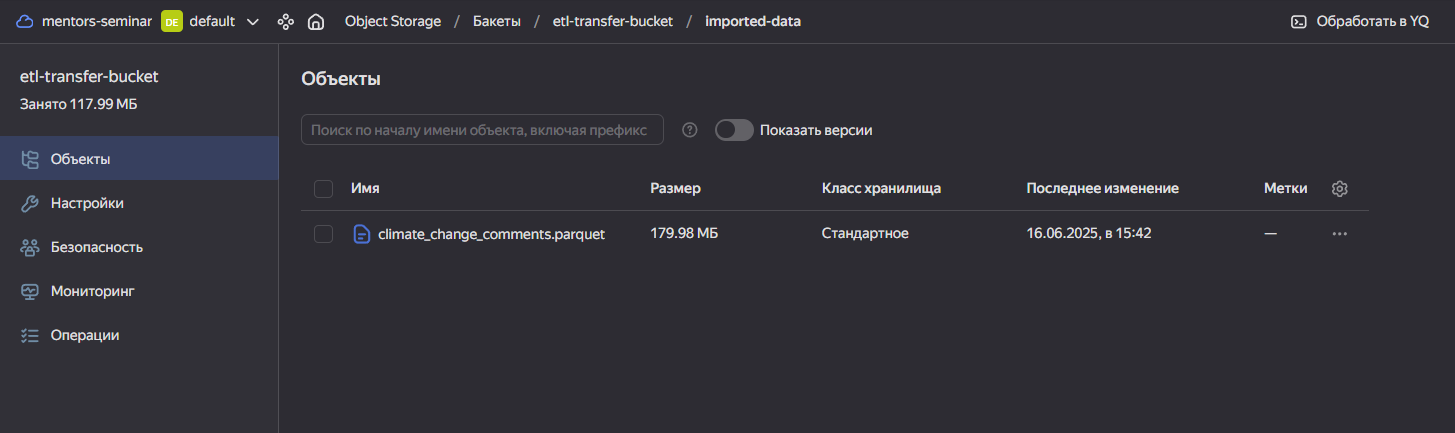


Рисунок 4

**Задание 2:**

Для выполнения данного задания нам необходимо создать Airflow DAG (**Листинг 2**), который будет создавать временный кластер Yandex Cloud Dataproc, создавать в нем задание PySpark (**Листинг 3**) с написанным нами скриптом, и удалять кластер после выполнения этого задания Pyspark. После, мы запускаем данный DAG в Managed Service For Airflow и проверяем результат (Рисунок 5-6).

Листинг 2

|  |
| --- |
| import datetime  from airflow import DAG  from airflow.utils.trigger\_rule import TriggerRule  from airflow.providers.yandex.operators.dataproc import (  DataprocCreateClusterOperator,  DataprocCreatePysparkJobOperator,  DataprocDeleteClusterOperator,  )  YC\_DP\_AZ = 'ru-central1-a'  YC\_DP\_SSH\_PUBLIC\_KEY = 'ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIF32WuDT0XtoS23R8tuj3Dc2T0ko+wlYLH50zwUl1JcC Yandex Cloud Data Proc Key'  YC\_DP\_SUBNET\_ID = 'e9bii53l9tcfd082p8m0'  YC\_DP\_SA\_ID = 'aje2cd52q35tf41qohb9'  YC\_BUCKET = 'data-proc-main-bucket'  with DAG(  dag\_id='dataproc\_dag',  schedule\_interval='@daily',  tags=['dataproc'],  start\_date=datetime.datetime.now(),  max\_active\_runs=1,  catchup=False  ) as ingest\_dag:  create\_dataproc\_cluster = DataprocCreateClusterOperator(  task\_id='create-cluster-task',  cluster\_name=f'dataproc-etl-final-cluster',  cluster\_description='',  ssh\_public\_keys=YC\_DP\_SSH\_PUBLIC\_KEY,  service\_account\_id=YC\_DP\_SA\_ID,  subnet\_id=YC\_DP\_SUBNET\_ID,  s3\_bucket=YC\_BUCKET,  zone=YC\_DP\_AZ,  cluster\_image\_version='2.1',  masternode\_resource\_preset='s2.small',  masternode\_disk\_type='network-hdd',  masternode\_disk\_size=32,  computenode\_resource\_preset='s2.small',  computenode\_disk\_type='network-hdd',  computenode\_disk\_size=32,  computenode\_count=1,  computenode\_max\_hosts\_count=3,  services=['YARN', 'SPARK'],  datanode\_count=0,  )  create\_pyspark\_job = DataprocCreatePysparkJobOperator(  task\_id='create-pyspark-job-task',  main\_python\_file\_uri=f's3a://{YC\_BUCKET}/pyspark-scripts/spark-task-script.py',  )  delete\_dataproc\_cluster = DataprocDeleteClusterOperator(  task\_id='delete-cluster-task',  trigger\_rule=TriggerRule.ALL\_DONE,  )  create\_dataproc\_cluster >> create\_pyspark\_job >> delete\_dataproc\_cluster |

Листинг 3

|  |
| --- |
| from pyspark.sql import SparkSession  from pyspark.sql.functions import col, to\_timestamp  from pyspark.sql.types import IntegerType, StringType, BooleanType, FloatType  spark = SparkSession.builder.appName("CSV to Parquet ETL process").getOrCreate()  source\_path = "s3a://etl-transfer-bucket/raw-data/reddit\_opinion\_climate\_change.csv"  target\_path = "s3a://data-proc-main-bucket/processed-data/reddit\_opinion\_climate\_change.parquet"  try:  df = spark.read.option("header", "true").option("inferSchema", "true").csv(source\_path)  df = df.withColumn("comment\_id", col("comment\_id").cast(StringType())) \  .withColumn("score", col("score").cast(FloatType())) \  .withColumn("self\_text", col("self\_text").cast(StringType())) \  .withColumn("subreddit", col("subreddit").cast(StringType())) \  .withColumn("created\_time", to\_timestamp(col("created\_time").cast("string"), "yyyy-MM-dd HH:mm:ss")) \  .withColumn("post\_id", col("post\_id").cast(StringType())) \  .withColumn("author\_name", col("author\_name").cast(StringType())) \  .withColumn("controversiality", col("controversiality").cast(FloatType())) \  .withColumn("ups", col("ups").cast(IntegerType())) \  .withColumn("downs", col("downs").cast(IntegerType())) \  .withColumn("user\_is\_verified", col("user\_is\_verified").cast(BooleanType())) \  .withColumn("user\_account\_created\_time",  to\_timestamp(col("user\_account\_created\_time").cast("string"), "yyyy-MM-dd HH:mm:ss")) \  .withColumn("user\_awardee\_karma", col("user\_awardee\_karma").cast(FloatType())) \  .withColumn("user\_awarder\_karma", col("user\_awarder\_karma").cast(FloatType())) \  .withColumn("user\_link\_karma", col("user\_link\_karma").cast(FloatType())) \  .withColumn("user\_comment\_karma", col("user\_comment\_karma").cast(FloatType())) \  .withColumn("user\_total\_karma", col("user\_total\_karma").cast(FloatType())) \  .withColumn("post\_score", col("post\_score").cast(IntegerType())) \  .withColumn("post\_self\_text", col("post\_self\_text").cast(StringType())) \  .withColumn("post\_title", col("post\_title").cast(StringType())) \  .withColumn("post\_upvote\_ratio", col("post\_upvote\_ratio").cast(FloatType())) \  .withColumn("post\_thumbs\_ups", col("post\_thumbs\_ups").cast(IntegerType())) \  .withColumn("post\_total\_awards\_received", col("post\_total\_awards\_received").cast(IntegerType())) \  .withColumn("post\_created\_time",  to\_timestamp(col("post\_created\_time").cast("string"), "yyyy-MM-dd HH:mm:ss"))  df = df.dropna()  df = df.dropDuplicates()  df.write.mode("overwrite").parquet(target\_path)  print("Task completed")  except Exception as e:  print("Exception thrown:", e)  spark.stop() |

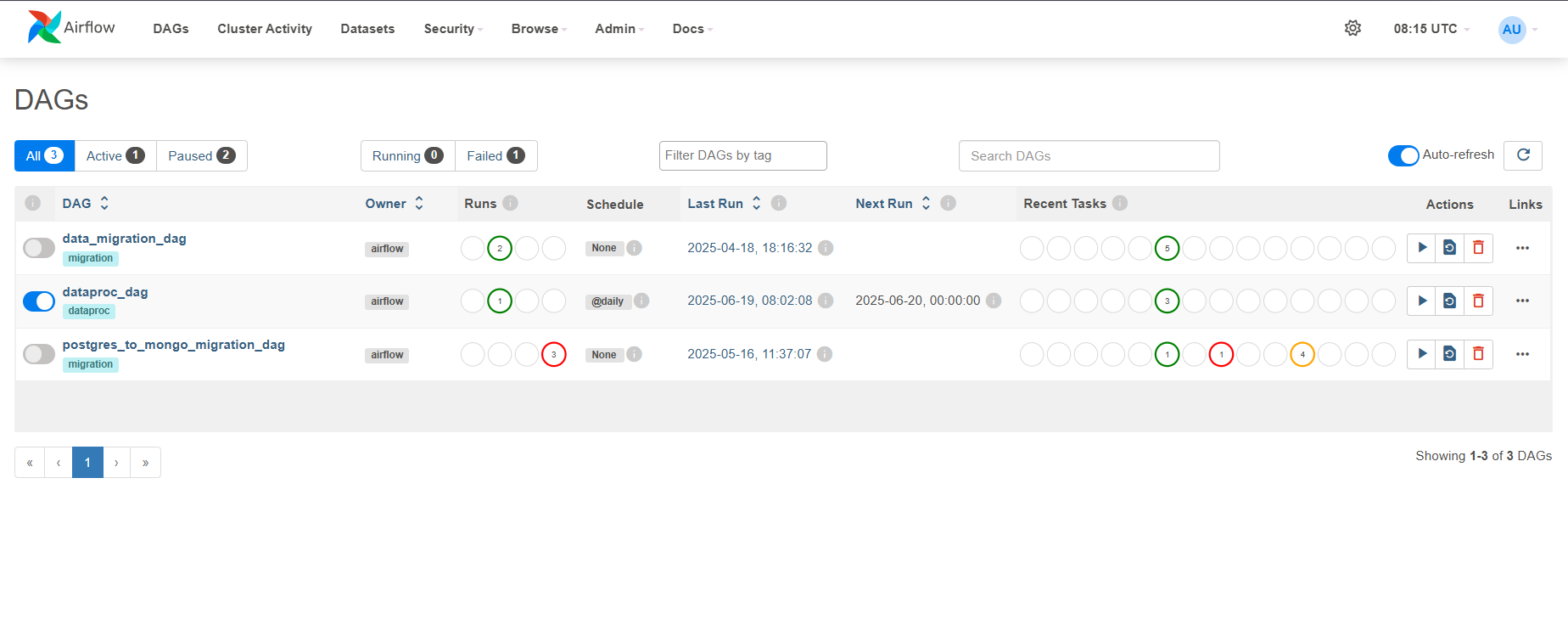


Рисунок 5

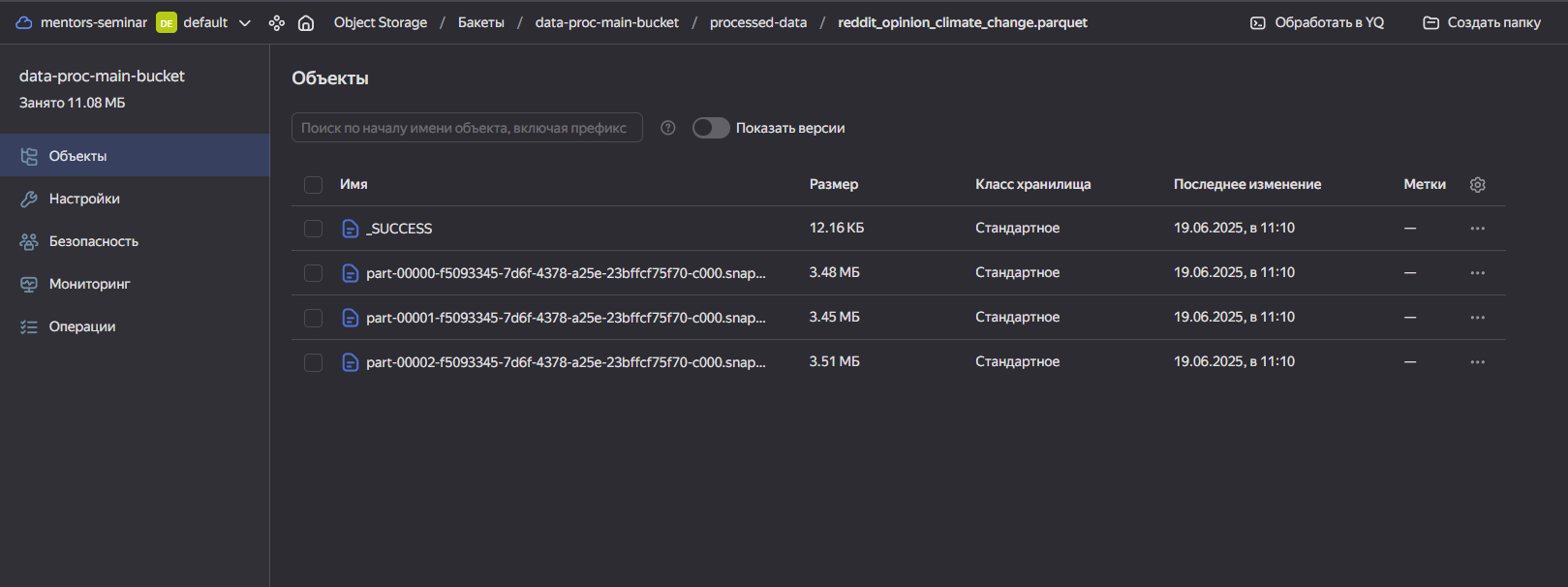


Рисунок 6

**Задание 3:**

Для выполнения третьего задания мы напишем два скрипта: первый будет читать данные из файла и записывать их в топик Kafka (**Листинг 4**), а второй будет считывать эти сообщения в потоке, и записывать их в таблицу Clickhouse (**Листинг 5**). Далее мы запустим их как PySpark-задания в нашем кластере Yandex Data Processing.

Листинг 4

|  |
| --- |
| from pyspark.sql import SparkSession  from pyspark.sql.functions import to\_json, col, struct  def send\_batch\_to\_kafka(iterator):  for row in iterator:  row.write.format('kafka') \  .option('kafka.bootstrap.servers', 'rc1a-bmibv8b7h7v6d78n.mdb.yandexcloud.net:9091') \  .option('topic', 'dataproc-topic') \  .option('kafka.security.protocol', 'SASL\_SSL') \  .option('kafka.sasl.mechanism', 'SCRAM-SHA-512') \  .option('kafka.sasl.jaas.config',  "org.apache.kafka.common.security.scram.ScramLoginModule required "  "username=\"user1\" "  "password=\"password1\";") \  .save()  yield "Partition Processed"  def main():  spark = SparkSession.builder \  .appName("parquet-to-kafka-spark") \  .getOrCreate()  df = spark.read.parquet("s3a://data-proc-main-bucket/processed-data/reddit\_opinion\_climate\_change.parquet").cache()  df = df.select(to\_json(struct([col(c).alias(c) for c in df.columns])).alias('value'))  df.rdd.mapPartitions(send\_batch\_to\_kafka).collect()  if \_\_name\_\_ == 'main':  main() |

Листинг 5

|  |
| --- |
| from pyspark.sql import SparkSession  from pyspark.sql.functions import from\_json, col, to\_timestamp  from pyspark.sql.types import StructType, StringType, IntegerType, BooleanType, FloatType  def write\_batch(batch, id):  batch.write \  .format("jdbc") \  .option("url", "jdbc:clickhouse://rc1a-4lv719lifmpcj20q.mdb.yandexcloud.net:8443/etl-final") \  .option("dbtable", "processed\_kafka\_comments") \  .option("user", "db\_user") \  .option("password", "p4ssw0rd") \  .option("driver", "org.clickhouse.Driver") \  .mode("append") \  .save()  def main():  spark = SparkSession.builder \  .appName("dataproc-kafka-read-stream") \  .getOrCreate()  schema = StructType() \  .add("comment\_id", StringType()) \  .add("score", FloatType()) \  .add("self\_text", StringType()) \  .add("subreddit", StringType()) \  .add("created\_time", StringType()) \  .add("post\_id", StringType()) \  .add("author\_name", StringType()) \  .add("controversiality", FloatType()) \  .add("ups", IntegerType()) \  .add("downs", IntegerType()) \  .add("user\_is\_verified", BooleanType()) \  .add("user\_account\_created\_time", StringType()) \  .add("user\_awardee\_karma", FloatType()) \  .add("user\_awarder\_karma", FloatType()) \  .add("user\_link\_karma", FloatType()) \  .add("user\_comment\_karma", FloatType()) \  .add("user\_total\_karma", FloatType()) \  .add("post\_score", IntegerType()) \  .add("post\_self\_text", StringType()) \  .add("post\_title", StringType()) \  .add("post\_upvote\_ratio", FloatType()) \  .add("post\_thumbs\_ups", IntegerType()) \  .add("post\_total\_awards\_received", IntegerType()) \  .add("post\_created\_time", StringType())  kafka\_df = spark.readStream.format("kafka") \  .option('kafka.bootstrap.servers', 'rc1a-bmibv8b7h7v6d78n.mdb.yandexcloud.net:9091') \  .option("subscribe", "dataproc-topic") \  .option("kafka.security.protocol", "SASL\_SSL") \  .option("kafka.sasl.mechanism", "SCRAM-SHA-512") \  .option("kafka.sasl.jaas.config",  "org.apache.kafka.common.security.scram.ScramLoginModule required "  "username=\"user1\" "  "password=\"password1\";") \  .option("startingOffsets", "latest") \  .load()  df = kafka\_df.selectExpr("CAST(value AS STRING) as json\_str") \  .select(from\_json(col("json\_str"), schema).alias("data")) \  .select("data.\*") \  .withColumn("created\_time", to\_timestamp(col("created\_time").cast("string"), "yyyy-MM-dd HH:mm:ss")) \  .withColumn("user\_account\_created\_time",  to\_timestamp(col("user\_account\_created\_time").cast("string"), "yyyy-MM-dd HH:mm:ss")) \  .withColumn("post\_created\_time",  to\_timestamp(col("post\_created\_time").cast("string"), "yyyy-MM-dd HH:mm:ss"))  query = df.writeStream \  .foreachBatch(write\_batch) \  .option("checkpointLocation", "s3a://dataproc-main-bucket/kafka-postgres-checkpoint") \  .trigger(processingTime="10 seconds") \  .start()  query.awaitTermination()  if \_\_name\_\_ == 'main':  main() |