Отчет

1. Добавляем контейнеры zookeeper, kafka и consumer в docker-compose.yml файл. Для брокера прописываем порты, адрес доступа, количество партиций и репликаций топика. Добавляем healthcheck для того, чтобы другие контейнеры стартовали после запуска kafka контейнера. В контейнер consumer выносим запуск предобработки данных и тренировки, поскольку именно он будет отвечать за работу с данными приходящими по api. Последним шагом запускаем сам консьюмер и создаем файл consumer-ready в качестве флага для начала работы web контейнера. Web сервис теперь просто отвечает за запуск приложения.

Листинг 1 - Код docker-compose.yml.

```
services:
    decrypt:
    image: python:3.9-slim
    container_name: decrypt_env
    volumes:
        - ./env:/app/env
    working_dir: /app/env
    command: bash -c "
        ls -la
        && source vault-pass.env
        && pip install ansible-vault
        && echo $$VAULT_PASSWORD > vault-pass.txt
        && ansible-vault decrypt secrets.env --vault-password-file vault-pass.txt --output .env
        && rm -rf vault-pass.txt"
    redis:
    image: redis:latest
    container_name: redis_db
    depends_on:
        decrypt:
            condition: service_completed_successfully
```

```
volumes:
      - ./env:/app/env
working_dir: /app/env
command: bash -c "
      ls -la
      && source .env
      && redis-server --requirepass $$REDIS_PASSWORD"
ports:
      - 6379:6379
healthcheck:
      test: ["CMD", "bash", "-c", "
            cd /app/env
            && source .env
            && redis-cli -a $$REDIS_PASSWORD ping"]
      interval: 1s
      timeout: 2s
      retries: 10
zookeeper:
image: confluentinc/cp-zookeeper:7.3.0
container_name: zookeeper
environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
healthcheck:
      test: [ "CMD", "nc", "-vz", "localhost", "2181" ]
      interval: 10s
      timeout: 3s
      retries: 3
broker:
image: confluentinc/cp-kafka:7.3.0
hostname: broker
container_name: broker
depends_on:
      zookeeper:
            condition: service_healthy
environment:
      KAFKA_BROKER_ID: 1
```

```
        KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
        KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker:9092
        KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
        KAFKA_NUM_PARTITIONS: 1
    healthcheck:
        test:       ["CMD",       "kafka-broker-api-versions",
"--bootstrap-server", "broker:9092"]
        interval: 10s
        timeout: 10s
        retries: 5
    ports:
        - 9092:9092
    consumer:
    build: .
    container_name: consumer
    volumes:
        - ./env:/app/env
    command: bash -c "
        ls -la
        && python src/preprocess.py
        && python src/train.py
        && python src/predict.py -m LOG_REG -t func
        && coverage run src/unit_tests/test_preprocess.py
        && coverage run -a src/unit_tests/test_training.py
        && coverage run -a src/unit_tests/test_database.py
        && coverage report -m
        && (python src/kafka_consumer.py --model LOG_REG &)
        && touch /tmp/consumer-ready
        && tail -f /dev/null"
    depends_on:
        decrypt:
            condition: service_completed_successfully
        redis:
            condition: service_healthy
        broker:
            condition: service_healthy
    healthcheck:
     test: ["CMD-SHELL", "test -f /tmp/consumer-ready"]
     interval: 1m
     timeout: 10s
     start_period: 5m
     retries: 30
web:
build: .
container_name: web
volumes:
     - ./env:/app/env
command: bash -c "
     ls -la
     && coverage run -a src/unit_tests/test_database.py
     && coverage run -a src/unit_tests/test_app.py
     && coverage report -m
     && (python src/app.py &)
     && sleep 30
     && curl -X GET http://localhost:8000/
     && curl -X POST http://localhost:8000/predict \
                -H 'Content-Type":" application/json' \
                --data-binary @tests/test_0.json"
depends_on:
     decrypt:
           condition: service_completed_successfully
     redis:
           condition: service_healthy
     broker:
           condition: service_healthy
     consumer:
           condition: service_healthy
ports:
     - 8000:8000
image: zarus03/ml-big-data-lab-4:latest
```

2. Выделим консьюмер в отдельный класс kafka_consumer.py. В нем запускается специальная модель, и в цикле обрабатываются приходящие с продюсера сообщения.

Листинг 2 - Реализация kafka_consumer.py.

```
import json
import pandas as pd
import pickle
import traceback
import sys
import argparse
import configparser
from logger import Logger
from database import RedisClient
from confluent_kafka import Consumer, KafkaError

SHOW_LOG = True



class KafkaConsumer:
     def __init__(
     self,
     args: argparse.Namespace,
     broker: str,
     topic: str,
     group: str
     ):
     logger = Logger(SHOW_LOG)
     self.log = logger.get_logger(__name__)

     self.config = configparser.ConfigParser()
     self.config.read("config.ini")

     self.args = args

     self.model, self.scaler = self._load_model()
```

```
    self.log.info('Kafka Consumer Model initialized')

    self.database_client = RedisClient()

    conf = {
        'bootstrap.servers': broker,
        'group.id': group,
        'session.timeout.ms': 6000,
        'auto.offset.reset': 'earliest',
        'enable.auto.offset.store': False}

    self.topic = topic

    self.consumer = Consumer(conf)
    self.consumer.subscribe(topics=[self.topic])

    def _load_model(self):
    """
    Loads the machine learning model and scaler from disk.

    Returns:
        tuple: Loaded model and scaler.
    """
    try:
        with open(self.config[self.args.model]["path"], "rb") as
model_file:
            model = pickle.load(model_file)

        with  open(self.config["STD_SCALER"]["path"],  "rb")  as
scaler_file:
            scaler = pickle.load(scaler_file)

        return model, scaler
    except FileNotFoundError:
        self.log.error(traceback.format_exc())
        sys.exit(1)
    def process(self, data: dict):
    prediction_id = data['prediction_id']
    input_data = json.loads(data['input_data'])

    X = self.scaler.transform(pd.json_normalize(input_data['X']))
    pred = self.model.predict(X).tolist()

    prediction_data = {'prediction': pred}

    self.database_client.set(prediction_id,
json.dumps(prediction_data))

    def consume_messages(self, timeout: float = 1.0):
    try:
        self.log.info(f"Starting consumer for topic
{self.topic}")
        while True:
            msg = self.consumer.poll(timeout)

            if msg is None:
                self.log.debug("No message received")
                continue

            if msg.error():
                if msg.error().code() ==
KafkaError._PARTITION_EOF:
                self.log.error(f'Reached end of partition:
{msg.topic()}[{msg.partition()}]')
                else:
                self.log.error(f'Error while consuming
messages: {msg.error()}')
                continue

            self.log.info(f"Received message on consumer one:
{msg.value().decode('utf-8')}")
```

```
self.process(json.loads(msg.value().decode('utf-8')))

      except KeyboardInterrupt:
            self.log.info("Stopping consumer...")

      finally:
            self.consumer.close()
            self.log.info("Consumer closed")


def main(args):
      consumer = KafkaConsumer(
      args,
      broker="broker:9092",
      topic="predictions",
      group="ml_app_group"
      )
      consumer.consume_messages()


if __name__ == "__main__":
      parser       =       argparse.ArgumentParser(description="Kafka
Consumer")
      parser.add_argument("-m", "--model",
                        type=str,
                        help="Select model",
                        required=True,
                        default="LOG_REG",
                        const="LOG_REG",
                        nargs="?",
                        choices=["LOG_REG"])
      args = parser.parse_args()
      main(args)
```

3. Выделим продюсер тоже в отдельный класс kafka_producer.py. В нем реализован специальный метод по отправке сообщения в брокер.

Листинг 3 - Реализация kafka_consumer.py.

```
import json
from logger import Logger
from confluent_kafka import Producer

SHOW_LOG = True



class KafkaProducer:
    def __init__(
    self,
    broker: str,
    topic: str
    ):
    logger = Logger(SHOW_LOG)
    self.log = logger.get_logger(__name__)

    producer_config = {
        'bootstrap.servers': broker,
    }
    self.topic = topic

    self.producer = Producer(producer_config)

    def delivery_report(self, err, msg):
    if err is not None:
        self.log.error(f'Message delivery failed: {err}')
    else:
        self.log.info(f'Message delivered to {msg.topic()}
[{msg.partition()}]')

    def send_message(self, message: dict):
```

```
    try:
        data = json.dumps(message).encode('utf-8')
        self.producer.produce(
            self.topic,
            value=data,
            callback=self.delivery_report
        )
        self.producer.flush()

    except Exception as e:
        self.log.error(f'Failed to send message: {e}')
        raise
```

4. Сам продюсер запускается в коде приложения. Отправки сообщения в брокер происходит в момент получения запроса на предсказание по ручке /predict.

Листинг 4 - Реализация app.py.

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List, Dict
from logger import Logger
from database import RedisClient
from kafka_producer import KafkaProducer
import traceback
import uvicorn
import json
import uuid

SHOW_LOG = True


class PredictionInput(BaseModel):
    """
    Pydantic model for input data validation.
    Attributes:
    X (List[Dict[str, float]]): List of feature dictionaries.
    y (List[Dict[str, float]]): List of target dictionaries.
    """
    X: List[Dict[str, float]]
    y: List[Dict[str, float]]

class WebApp:
    """
    Web application class using FastAPI for serving a machine learning model.
    """

    def __init__(self):
    """
    Initializes the web application, loads the model, and creates the FastAPI app.

    Args:
        args (argparse.Namespace): Parsed command-line arguments.
    """
    logger = Logger(SHOW_LOG)
    self.log = logger.get_logger(__name__)

    self.app = self._create_app()
    self.log.info('FastAPI app initialized')

    self.database_client = RedisClient()

    self.kafka_producer = KafkaProducer(
        broker="broker:9092",
        topic="predictions",
    )

    def _create_app(self):
    """
    Creates and configures the FastAPI application.

    Returns:
        FastAPI: Configured FastAPI app instance.
    """
    app = FastAPI()

    @app.get("/")
    async def root():
        """Root endpoint for health check."""
        return {"message": "Hello World"}

    @app.post("/predict")
    async def predict(input_data: PredictionInput):
        """
        Endpoint for making predictions using the trained model.

        Args:
            input_data (PredictionInput): Input data containing X (features) and y (target values).

        Returns:
            dict: A dictionary containing predictions and model score.
        """
        try:
            prediction_id = str(uuid.uuid4())

            prediction_data = {
                "prediction_id": prediction_id,
                "input_data": input_data.model_dump_json()
            }

            self.kafka_producer.send_message(prediction_data)

            return {"prediction_id": prediction_id}
        except Exception as e:
            self.log.error(traceback.format_exc())
            raise                HTTPException(status_code=500,
detail=str(e))

    @app.get("/predictions/{prediction_id}")
    async def get_prediction(prediction_id: str):
        """
        Endpoint for retrieving a prediction from Redis by its
ID.

        Args:
            prediction_id (str): The ID of the prediction to
retrieve.

        Returns:
            dict: The prediction data stored in Redis.
        """
        prediction_data                                        =
self.database_client.get(prediction_id)
        if prediction_data is None:
            raise                HTTPException(status_code=404,
detail="Prediction not found")
        return json.loads(prediction_data)

    return app

    def run(self, host: str = "0.0.0.0", port: int = 8000):
    """
    Runs the FastAPI application using Uvicorn.

    Args:
        host (str): Host address to run the server.
        port (int): Port number to run the server.
    """
    uvicorn.run(self.app, host=host, port=port)
```

```
if __name__ == "__main__":
      web_app = WebApp()
      web_app.run()
```