Проектное домашнее задание: Построение системы микросервисов для поддержки дипломной работы

Кафедра Интеллектуальных Информационных Технологий
Группа 420

09 сентября 2025

1. Введение

Цель задания — разработать распределенную систему микросервисов (с использованием Python) с контейнеризацией (Docker, docker-compose) для поддержки научно-исследовательской и/или дипломной работы.

Для выполнения задания необходимо: **(1) сформулировать проектную или научную задачу, (2) определить список из нескольких (не менее 4) сценариев использования (должны быть понятны и полезны целевому пользователю, хотя бы один должен использовать математическую/ML модель), (3) спроектировать микросервисную архитектуру и (4) реализовать программную систему**.

Система должна демонстрировать разделение обязанностей между независимыми сервисами: сбор данных, хранилище, ML-платформа, веб-сервер (мастер), сервис визуализации и дополнительные сервисы (при необходимости). Система должна состоять (как минимум) из следующих независимых сервисов (каждый сервис — отдельный контейнер):

1. **Collector (сборщик данных)** — собирает данные по запросу, ведет лог загрузок. Для упрощения допускается использовать собранный набор данных, из которого Collector возвращает батчи по запросу.

2. **Storage (хранилище данных)** — интерфейс к файловой системе и/или базе данных (Postgres, ClickHouse, MinIO и других). Обеспечивает API для записи/чтения сырых и обработанных данных.

3. **ML Service (платформа обучения и инференса)** — хранит ML-артефакты, предоставляет API для подготовки данных, обучения, валидации, инференса моделей (или иное).

4. **Web Master (главный веб-сервер)** — единая точка входа (API gateway), отвечает за маршрутизацию запросов к сервисам, демонстрацию состояния системы, и API для реализации сценариев использования.

5. **Visualization Service** — интерактивный GUI (Plotly Dash/Bokeh/Flask с JS) для выполнения пользователем сценариев использования, можно реализовать оболочкой для Web Master.

6. **Иные сервисы (опционально)**.

Каждый сервис должен содержать:

- Dockerfile и образ, описанный в docker-compose.yml;
- минимальный REST API (или gRPC) для ключевых операций;
- конфигурационный файл (YAML/JSON/TOML) с параметрами среды;
- механизм обработки ошибок.

Например, рассмотрим задачу поиска релевантных фильмов по текстовому запросу пользователя. Можно определить следующие сценарии использования: «сбор данных из внешнего источника», «построение/обновление модели рекомендации», «рекомендация N фильмов по запросу пользователя», «построение отчета значимых признаков модели». В качестве источника данных взять Imdb/Кинопоиск (или собранные данные в csv). В качестве GUI реализовать веб-страницу на Flask с 4 кнопками (по сценариям использования), текстовым полем запроса пользователя, таблицей рекомендуемых фильмов (указать название, ссылку). Источник данных фиксировать в конфигурационных файлах или определить в GUI, отчет о значимых признаках модели оформить в виде скачиваемого html/pdf файла.

2. Формат сдачи

Срок сдачи: 10 декабря 2025 года. Сдача производится в очном формате, при котором проверяющий определяет финальную оценку работы. Доработки и исправления предусмотрены только при досрочной сдаче (при условии, что проверяющий успел ознакомиться с работой).
Промежуточная оценка работы: 10 ноября 2025 года. При промежуточной оценке работы проверяющий определяет текущую оценку работы и определяет возможные слабые стороны системы для дальнейшей доработки.

Реализованный проект оформляется в **GitHub-репозитории** (можно разрабатывать приватно и открыть доступ проверяющему по почте). Репозиторий должен содержать файлы README, requirements и набор директорий для каждого из этапов конвейера.

Задание выполняется индивидуально. По реализованной системе готовится отчет (документация), отражающий поставленную задачу, выбранные сценарии использования, архитектуру системы, API, конфигурации, данные, примеры использования. В репозитории также должны быть представлены средства для запуска и управления системой. Например, демонстрационный скрипт и инструкции для проведения основных сценариев.

3. Критерии оценки

Общая шкала — 100 баллов. Оценка складывается из следующих частей.

1. **Архитектура и модульность (до 20 баллов)**:
   (a) Обязательная часть (10 баллов): наличие независимых сервисов, Dockerfile, docker-compose, четкая структура;
   (b) Дополнительная часть:
      i. Масштабируемость: возможность добавления нескольких источников или разных моделей;
      ii. Открытость: перенос системы на GitHub actions или иное;
2. **Collector (до 10 баллов)**:
   (a) Обязательная часть (5 баллов): API получения батча данных (для эмуляции достаточно подавать последовательные части заранее заданного набора), обработка ошибок;
   (b) Дополнительная часть:
      i. Сбор данных не из заданного файла, а из внешнего источника;
      ii. Расширенный лог действий и запросов (запись в файл);
3. **Storage (до 10 баллов)**:
   (a) Обязательная часть (5 баллов): API CRUD операций базы данных;
   (b) Дополнительная часть:
      i. Использование специализированных баз данных (связанных с предметной областью);
      ii. Расширенный лог действий и запросов (запись в файл);
4. **ML Service (до 20 баллов)**:
   (a) Обязательная часть (10 баллов): API запросов создания, обучения, применения и описания состояния (summary) моделей;
   (b) Дополнительная часть:
      i. Сложность модели;
      ii. Версионность моделей;
      iii. Построение хранилища моделей на основе БД;
5. **Web Master (до 10 баллов)**:
   (a) Обязательная часть (5 баллов): API сценариев использования;
   (b) Дополнительная часть:
      i. Аутентификация/авторизация;
      ii. Разделение сценариев использования по группам пользователей;
6. **Visualization (до 20 баллов)**:

(a) Обязательная часть (10 баллов): интерактивный GUI (прослойка между пользователем и Web Master);

(b) Дополнительная часть:

i. Сложность GUI: дашборд, отчеты (PDF/HTML), веб-сайт;

ii. Поддержка и визуализация истории выполнения сценариев.

7. **Документирование проекта (до 10 баллов)**:

(a) Добавление раздела в README с инструкциями по развертыванию;

(b) Текстовое описание поставленной задачи, сценариев использования и аргументация выбранного подхода;

(c) Разработка средств запуска и управления системой.

4. Рекомендации и ограничения

- Код API сервисов должен быть на Python (например, FastAPI/Flask/Falcon или иное). Использование других языков для реализации функционала отдельных сервисов допустимо;
- Контейнеризация: Docker + docker-compose для локального развертывания.
- Конфигурация должна быть вынесена в файлы (не "хардкодить"пути/порты);
- Не использовать тяжелые проприетарные MLOps-решения (MLFlow/Airflow и другие) как основу архитектуры. Цель задания — научиться строить интеграцию собственными средствами. Интеграция таких инструментов дается как опция для бонусных баллов;
- В качестве хранилища моделей допустимо использовать файловую систему (с версионированием по именам), либо простую таблицу в Postgres/SQLite с метаданными;
- Объем набора данных можно ограничить (например, 10k строк), но в отчете указать характеристики исходных данных.