

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение  
высшего образования «Самарский национальный исследовательский  
университет  
имени академика С.П. Королева»  
(Самарский университет)**

**Институт информатики и кибернетики  
Кафедра технической кибернетики**

**Отчет по лабораторной работе №1**

**Дисциплина: «Проектирование программных комплексов»**

**Дисциплина: «Инженерия данных»**

**Выполнил: Седых М.Ю.  
Группа: 6232-010402D**

**Самара 2025**

## **Архитектура пайплайна**

Система реализует ETL-пайплайн для обработки данных о погоде с использованием оркестрации, объектного хранилища и колоночной СУБД.

## **Основные компоненты**

Prefect 2.x – оркестрация и планирование

Используется для декларативного описания задач, их переиспользования, логирования, планирования запусков и управления зависимостями.

Работа осуществляется через Prefect Server (REST API), который хранит метаданные выполнения в PostgreSQL.

MinIO – объектное хранилище

Применяется для хранения сырых JSON-ответов, полученных от внешнего API. Совместим с S3 и удобен для версионирования, повторной обработки и аудита данных.

Формат хранения файлов: raw/{город}/{timestamp}.json.

ClickHouse – колоночная СУБД

Используется для аналитических запросов и хранения временных рядов. Оптимизирован для высокопроизводительного чтения и агрегаций.

В системе используются таблицы weather\_hourly для почасовых данных и weather\_daily для дневных агрегатов.

Обе таблицы построены на движке MergeTree и упорядочены по ключам (city, timestamp) или (city, date).

## **Схема потока данных**

Данные проходят последовательность этапов: Open-Meteo API → Extract → MinIO (raw) → Transform → Load (ClickHouse) → Telegram-уведомления.

## **Источник данных**

В качестве источника используется Open-Meteo Forecast API/

Запросы формируются с указанием координат города (latitude, longitude), параметров почасового и дневного прогноза, временной зоны Europe/Moscow и горизонта прогноза в 2 дня.

## **Extract → Transform → Load**

### **Extract**

Получение прогноза выполняется через HTTP GET-запросы с таймаутом 10 секунд. Для каждого города, включая Москву и Самару, формируется отдельный запрос.

В процессе извлечения выполняется проверка статуса ответа через `response.raise_for_status()`, в результат добавляется поле `city`, после чего сырой JSON сохраняется в MinIO с временной меткой в имени файла для обеспечения версионирования.

### **Transform**

Для почасовых данных выполняется фильтрация значений на завтрашний день, извлекаются необходимые поля из массива `hourly` и формируется DataFrame со столбцами `timestamp`, `city`, `temperature_celsius`, `precipitation_mm`, `weather_code`, `wind_speed_kmh` и `wind_direction_degrees`. Дополнительно добавляется техническое поле `inserted_at`.

Для дневных данных извлекаются агрегированные показатели из массива `daily`. Средняя температура за завтрашний день вычисляется на основе почасовых значений. Формируется запись с полями `temp_min_celsius`, `temp_max_celsius`, `temp_avg_celsius`, `precipitation_mm`, `weather_code` и `wind_speed_max_kmh`.

Данные по всем городам объединяются перед загрузкой с помощью `pd.concat()`.

### **Load**

Перед началом загрузки выполняется создание таблиц `weather_hourly` и `weather_daily` при их отсутствии с использованием `CREATE TABLE IF NOT EXISTS`. DataFrame преобразуется в список кортежей, после чего данные вставляются в ClickHouse через `client.execute` с оператором `INSERT INTO`.

Если DataFrame пустой, загрузка пропускается. Количество загруженных записей подсчитывается и возвращается для логирования в `Prefect`.

## Качество данных

В пайплайне реализован ряд проверок качества данных.

На этапе извлечения используется HTTP-валидация через `response.raise_for_status()` для обработки ошибок 4xx и 5xx.

Перед загрузкой выполняется проверка пустоты данных с помощью `df.empty`.

При извлечении дневных данных контролируются границы массивов для предотвращения выхода за пределы индексов.

Перед отправкой уведомлений выполняется проверка наличия переменных окружения токена и id чата, необходимых для корректной работы Telegram-уведомлений.

```
PS R:\Python\ID\lr1> python weather_etl.py
20:37:48.436 | INFO | prefect.engine - Created flow run 'proficient-groundhog' for flow 'weather_etl'
20:37:48.593 | INFO | Flow run 'proficient-groundhog' - Created task run 'fetch_weather_data-0' for task 'fetch_weather_data'
20:37:48.594 | INFO | Flow run 'proficient-groundhog' - Executing 'fetch_weather_data-0' immediately...
20:37:49.768 | INFO | Task run 'fetch_weather_data-0' - Finished in state Completed()
20:37:50.510 | INFO | Flow run 'proficient-groundhog' - Created task run 'save_raw_data_to_minio-0' for task 'save_raw_data_to_minio'
20:37:50.511 | INFO | Flow run 'proficient-groundhog' - Executing 'save_raw_data_to_minio-0' immediately...
20:37:50.667 | INFO | Task run 'save_raw_data_to_minio-0' - Finished in state Completed()
20:37:50.707 | INFO | Flow run 'proficient-groundhog' - Created task run 'fetch_weather_data-1' for task 'fetch_weather_data'
20:37:50.708 | INFO | Flow run 'proficient-groundhog' - Executing 'fetch_weather_data-1' immediately...
20:37:51.436 | INFO | Task run 'fetch_weather_data-1' - Finished in state Completed()
20:37:51.486 | INFO | Flow run 'proficient-groundhog' - Created task run 'save_raw_data_to_minio-1' for task 'save_raw_data_to_minio'
20:37:51.487 | INFO | Flow run 'proficient-groundhog' - Executing 'save_raw_data_to_minio-1' immediately...
20:37:51.645 | INFO | Task run 'save_raw_data_to_minio-1' - Finished in state Completed()
20:37:51.696 | INFO | Flow run 'proficient-groundhog' - Created task run 'normalize_hourly_data-0' for task 'normalize_hourly_data'
20:37:51.697 | INFO | Flow run 'proficient-groundhog' - Executing 'normalize_hourly_data-0' immediately...
20:37:51.798 | INFO | Task run 'normalize_hourly_data-0' - Finished in state Completed()
20:37:51.840 | INFO | Flow run 'proficient-groundhog' - Created task run 'aggregate_daily_data-0' for task 'aggregate_daily_data'
20:37:51.840 | INFO | Flow run 'proficient-groundhog' - Executing 'aggregate_daily_data-0' immediately...
20:37:51.943 | INFO | Task run 'aggregate_daily_data-0' - Finished in state Completed()
20:37:51.985 | INFO | Flow run 'proficient-groundhog' - Created task run 'normalize_hourly_data-1' for task 'normalize_hourly_data'
20:37:51.986 | INFO | Flow run 'proficient-groundhog' - Executing 'normalize_hourly_data-1' immediately...
20:37:52.112 | INFO | Task run 'normalize_hourly_data-1' - Finished in state Completed()
20:37:52.163 | INFO | Flow run 'proficient-groundhog' - Created task run 'aggregate_daily_data-1' for task 'aggregate_daily_data'
20:37:52.164 | INFO | Flow run 'proficient-groundhog' - Executing 'aggregate_daily_data-1' immediately...
20:37:52.274 | INFO | Task run 'aggregate_daily_data-1' - Finished in state Completed()
20:37:52.316 | INFO | Flow run 'proficient-groundhog' - Created task run 'load_hourly_data-0' for task 'load_hourly_data'
20:37:52.317 | INFO | Flow run 'proficient-groundhog' - Executing 'load_hourly_data-0' immediately...
20:37:52.463 | INFO | Task run 'load_hourly_data-0' - Finished in state Completed()
20:37:52.506 | INFO | Flow run 'proficient-groundhog' - Created task run 'load_daily_data-0' for task 'load_daily_data'
20:37:52.507 | INFO | Flow run 'proficient-groundhog' - Executing 'load_daily_data-0' immediately...
20:37:52.643 | INFO | Task run 'load_daily_data-0' - Finished in state Completed()
20:37:52.683 | INFO | Flow run 'proficient-groundhog' - Created task run 'send_telegram_notification-0' for task 'send_telegram_notification'
20:37:53.662 | INFO | Flow run 'proficient-groundhog' - Finished in state Completed()
Pipeline result: {'hourly_records': 48, 'daily_records': 2, 'minio_paths': ['raw/москва/20251214_203750.json', 'raw/камара/20251214_203751.json']}
```

Рисунок 1 — Интерфейс Prefect с результатами выполнения flow

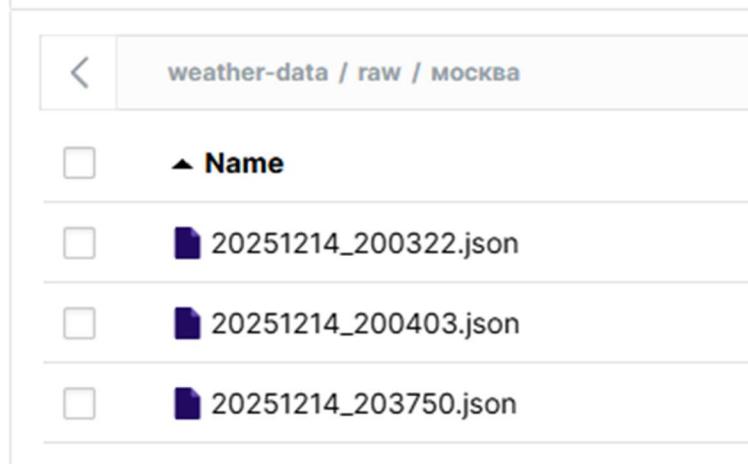


Рисунок 2 — Содержимое бакета weather-data в MinIO

На рисунке 3 показаны почасовые данные о температуре, осадках, скорости ветра и кодах погоды для города Москва на завтрашний день.

http://localhost:8125							v25.11.2.24, uptime 44 min	default	password
SELECT timestamp, city, temperature_celsius, precipitation_mm, wind_speed_kmh, weather_code FROM weather_db.weather_hourly WHERE city = 'Москва' ORDER BY timestamp DESC LIMIT 10							Run (Ctrl/Cmd+Enter)	10 rows in result, 0.01 sec.	100.0%, Read 144 rows, 4.61 KB
#	timestamp	city	temperature_celsius	precipitation_mm	wind_speed_kmh	weather_code			
1	2025-12-15 23:00:00	Москва	-6.8	0	2.6	2			
2	2025-12-15 23:00:00	Москва	-6.8	0	2.6	2			
3	2025-12-15 23:00:00	Москва	-6.8	0	2.6	2			
4	2025-12-15 22:00:00	Москва	-6.7	0	2.3	2			
5	2025-12-15 22:00:00	Москва	-6.7	0	2.3	2			
6	2025-12-15 22:00:00	Москва	-6.7	0	2.3	2			
7	2025-12-15 21:00:00	Москва	-6.6	0	2.8	3			
8	2025-12-15 21:00:00	Москва	-6.6	0	2.8	3			
9	2025-12-15 21:00:00	Москва	-6.6	0	2.8	3			
10	2025-12-15 20:00:00	Москва	-6.6	0	3.1	3			

Рисунок 3 — Результаты запроса к таблице weather\_hourly в ClickHouse.

На рисунке 4 показаны агрегированные дневные данные с минимальной, максимальной и средней температурой, общим количеством осадков и максимальной скоростью ветра для городов Москва и Самара.

N <sup>o</sup>	date	city	temp_min_celsius	temp_max_celsius	temp_avg_celsius	precipitation_mm	wind_speed_max_kmh
1	2025-12-15	Москва	-8.9	-6.3	-7.5583334	0	8.2
2	2025-12-15	Москва	-8.9	-6.3	-7.5583334	0	8.2
3	2025-12-15	Москва	-8.9	-6.3	-7.5583334	0	8.2
4	2025-12-15	Самара	-10.1	-8.7	-9.470834	0.5	21
5	2025-12-15	Самара	-10.1	-8.7	-9.470834	0.5	21
6	2025-12-15	Самара	-10.1	-8.7	-9.470834	0.5	21

Рисунок 4 — Результаты запроса к таблице weather\_daily в ClickHouse.

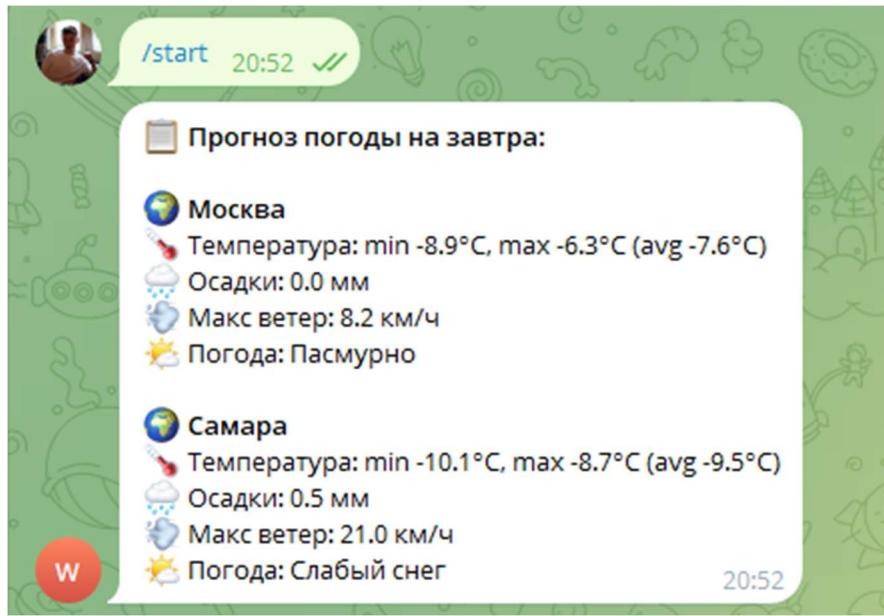


Рисунок 5 — Пример уведомления в Telegram с прогнозом погоды на завтра.

## Выводы

Основной проблемой стала несовместимость Perfect с Python 3.14, пришлось переустанавливать. В будущем стоит добавить retry для запросов к API.