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

ЛАБОРАТОРНАЯ РАБОТА № 1
Введение в ETL пайплайны. Знакомство с Prefect и ClickHouse

по курсу
Инженерия данных

Группа 6231
Студент_____ И.В. Спектор
(*подпись*)

Преподаватель _____ Р.А. Паринге
(*подпись*)

Самара 2025

АРХИТЕКТУРА

Пайплайн, представленный на рисунке 1, был реализован в ходе данной лабораторной работы с использованием стека технологий для обработки данных:

– **Prefect** - главный управляющий. Он следит чтобы все задачи выполнялись вовремя, а если что-то ломается - пытается повторить. Он проще в настройке чем похожие программы.
– **Open-Meteo** - наш поставщик погоды. Бесплатный сервис, который отдает прогноз в понятном формате.
– **MinIO** - хранилище для сырых данных. Сохраняет оригинальные ответы от погодного сервиса, чтобы можно было потом перепроверить.
– **ClickHouse** - быстрая база данных. Отлично подходит для хранения и анализа погодных данных с течением времени.
– **Docker** - упаковывает все программы в контейнеры, чтобы они работали одинаково на любом компьютере.

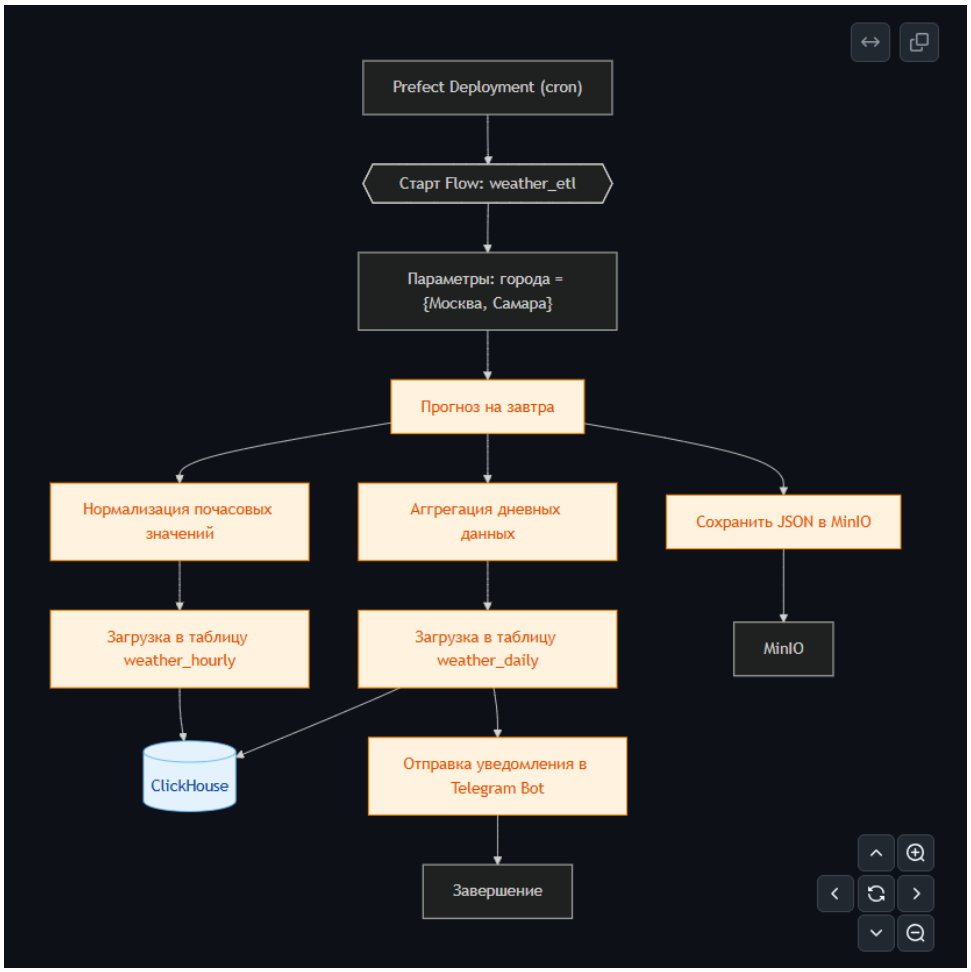


Рисунок 1 – Схема пайплайна первой лабораторной работы.

ИСТОЧНИК ДАННЫХ

У меня не получилось использовать Free Weather API, как предлагалось в описании к лабораторной работе, поэтому я нашёл бесплатный API для погоды, который мне подходил. Им оказался API Open-Meteo с эндпоинтом https://api.open-meteo.com/v1/forecast.

Параметры для запроса (см. рис. 2):

– latitude, longitude - координаты городов (Москва, Самара)
– daily - минимальная/максимальная температура, сумма осадков
– hourly - почасовые данные: температура, осадки, скорость и направление ветра
– forecast_days: 2 - прогноз на 2 дня вперед
– timezone: Europe/Moscow - временная зона

```python
@task(retries=3, retry_delay_seconds=10)
def fetch_weather_data(city: str, latitude: float, longitude: float) -> Dict:
    """Fetch weather forecast for specified city"""

    params = {
        "latitude": latitude,
        "longitude": longitude,
        "daily": "temperature_2m_max,temperature_2m_min,precipitation_sum",
        "hourly": "temperature_2m,precipitation,wind_speed_10m,wind_direction_10m",
        "timezone": "Europe/Moscow",
        "forecast_days": 2
    }

    print(f"Fetching data for {city}")

    try:
        response = requests.get(WEATHER_API_URL, params=params, timeout=30)
        response.raise_for_status()

        data = response.json()
        data.update({
            'city': city,
            'fetched_at': datetime.now().isoformat()
        })

        return data

    except Exception as e:
        print(f"API error for {city}: {e}")
        raise
```

Рисунок 2 – Запрос к Open-Meteo для получения требуемых данных.

EXTRACT. TRANSFROM. LOAD

Получение данных (Extract)

Система берет прогноз погоды с сайта Open-Meteo для Москвы и Самары. Если при запросе возникают проблемы с интернетом, она автоматически пробует еще раз несколько раз с паузами. Все полученные данные сохраняются в хранилище MinIO в виде файлов с пометкой времени, чтобы потом можно было проверить что именно пришло от API.

Обработка данных (Transform)

Данные обрабатываются двумя способами. Для почасового прогноза я оставлял только данные на завтрашний день, чтобы не хранить лишнее. Для суточных данных мы вычисляем среднюю температуру за день и берем общее количество осадков. Если вдруг API не прислал данных на завтра, система напишет об этом в лог и продолжит работать со следующим городом.

Сохранение в базу (Load)

Обработанные данные складываются в две разные таблицы. В одной хранится подробный почасовой прогноз, в другой - сводные данные за весь день. База данных настроена так, чтобы быстро находить информацию по конкретному городу и дате, даже когда данных становится очень много.

КАЧЕСТВО ДАННЫХ

В пайплайне я реализовал базовые механизмы обеспечения надежности обработки данных в виде проверки и обработки ошибок:

На этапе извлечения данных:

- Автоматические повторные запросы при сетевых сбоях (до 3 попыток);
- Проверка HTTP-статуса ответа от API;
- Логирование ошибок подключения.

На этапе обработки:

- Проверка наличия данных на завтрашнюю дату;
- Обработка случаев, когда данные на завтра отсутствуют;
- Преобразование форматов данных (дата/время) для корректного хранения.

На этапе загрузки:

- Проверка наличия данных перед записью в базу;
- Логирование результатов операций загрузки.

Мониторинг выполнения

Система отслеживает ключевые метрики выполнения:

- Количество успешно обработанных городов;
- Объем загруженных данных (количество записей);
- Факты ошибок и их причины.

Обработка сбоев

При возникновении проблем пайплайн:

- Продолжает обработку следующих городов если текущий завершился ошибкой;
- Предоставляет понятные сообщения об ошибках в логах;
- Сохраняет работоспособность при частичных сбоях.

Все ошибки логируются и могут быть просмотрены через Prefect UI.

РЕЗУЛЬТАТЫ И ПРИМЕРЫ РАБОТЫ

В Prefect я запустил default-pool (см. рис. 3, 4).

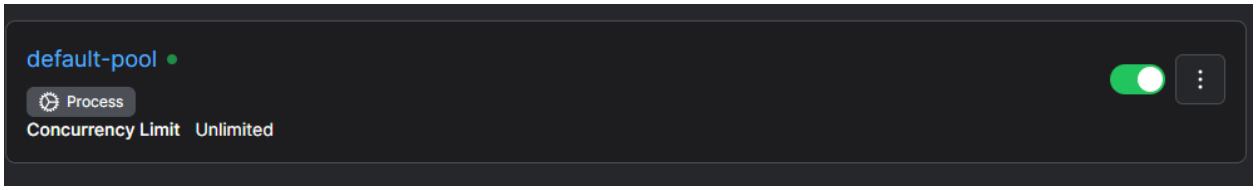


Рисунок 3 – Текущий default-pool в prefect.

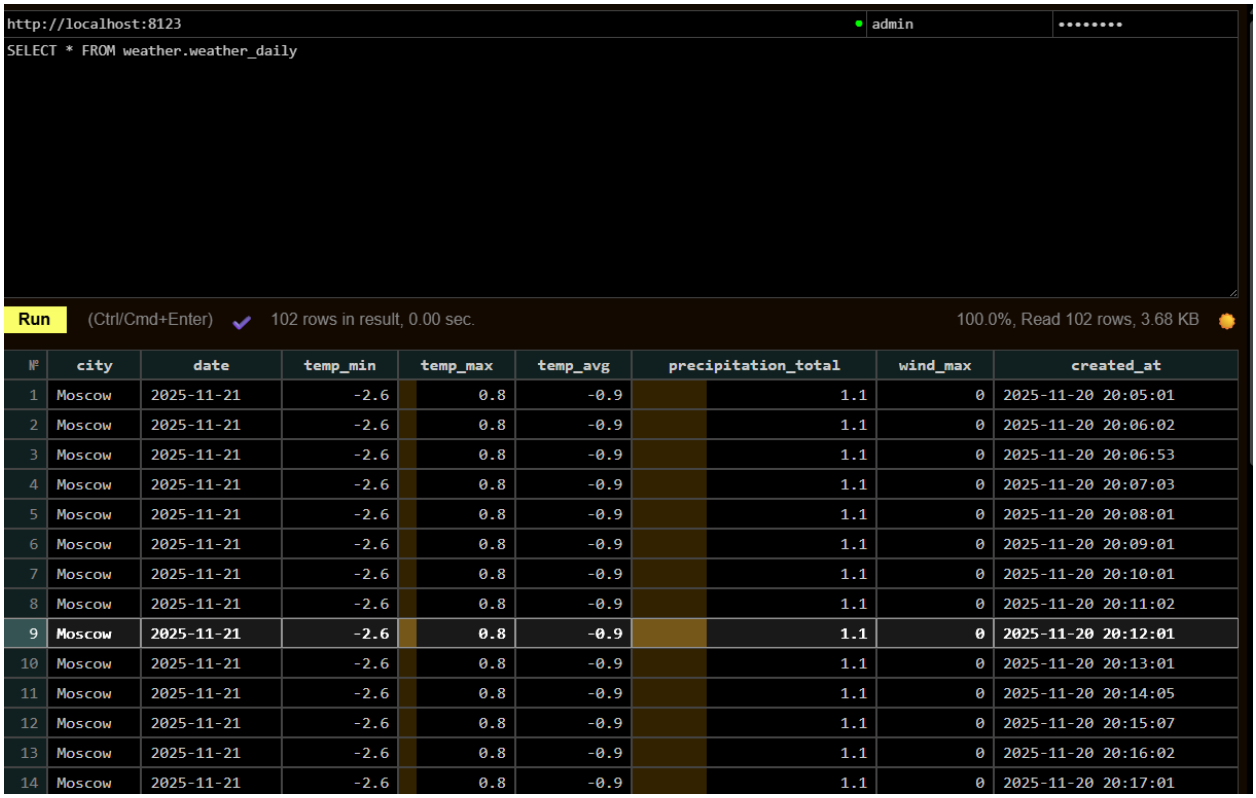
```
PS D:\Lab1-data-engeneering-Spektor> docker exec -it lab1-data-engeneering-spektor-prefect-worker-1 pr
efect work-pool create "default-pool" --type process
```

Рисунок 4 – Запрос для создания default-pool в prefect.

Перед этим я создал таблицы для хранения данных в clickhouse с помощью скрипта create_tables.py (см. рис. 5-7).

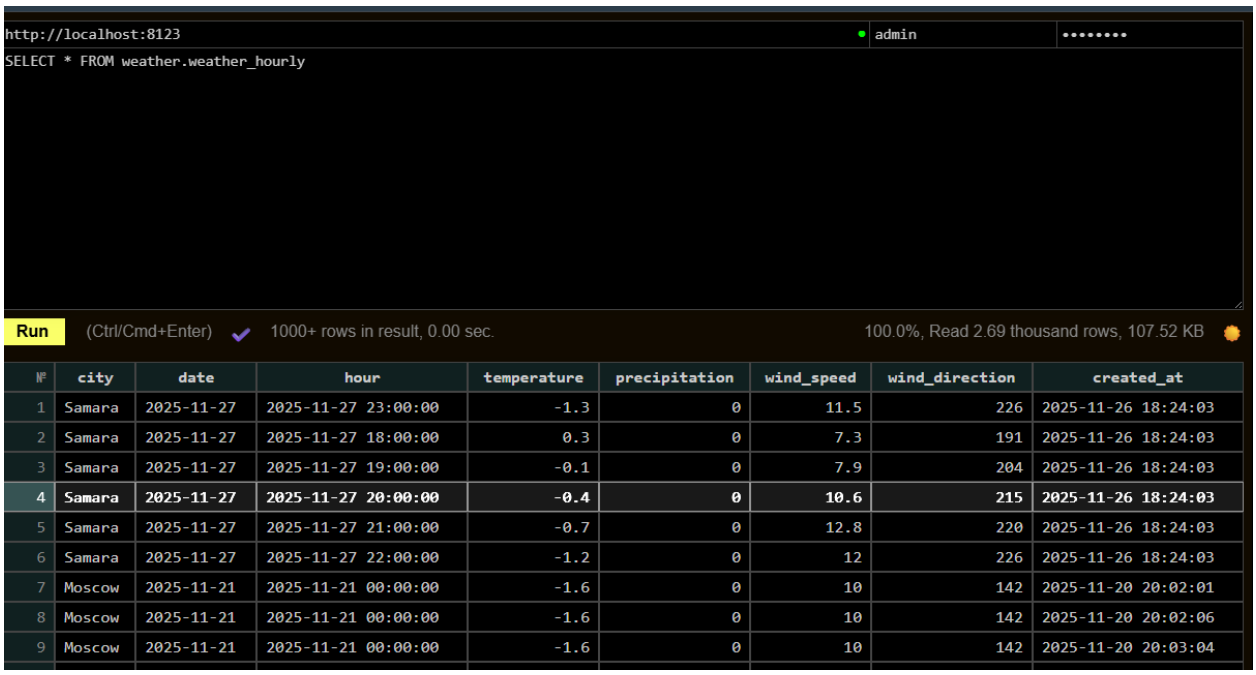
```
ng-spektor-clickhouse-1
PS D:\Lab1-data-engeneering-Spektor> docker exec -it lab1-data-engeneering-spektor-prefect-worker-1 py
thon /app/flows/create_tables.py
Database 'weather' created/verified
Table 'weather_hourly' created/verified
Table 'weather_daily' created/verified
All tables created successfully
```

Рисунок 5 – Запрос для создания таблиц данных в clickhouse.



№	city	date	temp_min	temp_max	temp_avg	precipitation_total	wind_max	created_at
1	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:05:01
2	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:06:02
3	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:06:53
4	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:07:03
5	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:08:01
6	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:09:01
7	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:10:01
8	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:11:02
9	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:12:01
10	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:13:01
11	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:14:05
12	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:15:07
13	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:16:02
14	Moscow	2025-11-21	-2.6	0.8	-0.9	1.1	0	2025-11-20 20:17:01

Рисунок 6 – Запрос для просмотра таблицы погоды по дням.



№	city	date	hour	temperature	precipitation	wind_speed	wind_direction	created_at
1	Samara	2025-11-27	2025-11-27 23:00:00	-1.3	0	11.5	226	2025-11-26 18:24:03
2	Samara	2025-11-27	2025-11-27 18:00:00	0.3	0	7.3	191	2025-11-26 18:24:03
3	Samara	2025-11-27	2025-11-27 19:00:00	-0.1	0	7.9	204	2025-11-26 18:24:03
4	Samara	2025-11-27	2025-11-27 20:00:00	-0.4	0	10.6	215	2025-11-26 18:24:03
5	Samara	2025-11-27	2025-11-27 21:00:00	-0.7	0	12.8	220	2025-11-26 18:24:03
6	Samara	2025-11-27	2025-11-27 22:00:00	-1.2	0	12	226	2025-11-26 18:24:03
7	Moscow	2025-11-21	2025-11-21 00:00:00	-1.6	0	10	142	2025-11-20 20:02:01
8	Moscow	2025-11-21	2025-11-21 00:00:00	-1.6	0	10	142	2025-11-20 20:02:06
9	Moscow	2025-11-21	2025-11-21 00:00:00	-1.6	0	10	142	2025-11-20 20:03:04

Рисунок 7 – Запрос для просмотра таблицы погоды по часам.

После развертывания таблиц и запуска deafault-pool я задеплоил weather-etl-production (см. рис. 8) с flow weather_etl_flow (см. рис. 9)

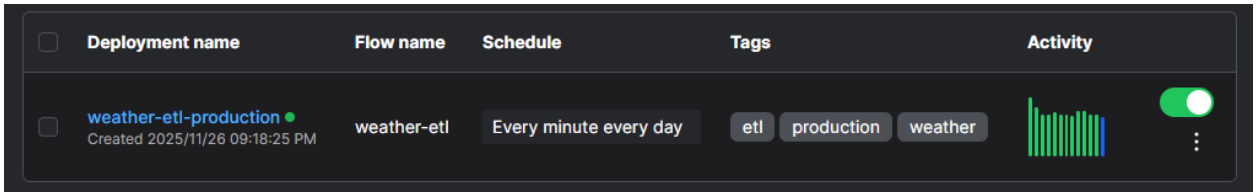


Рисунок 8 – Развертывание weather-etl-production

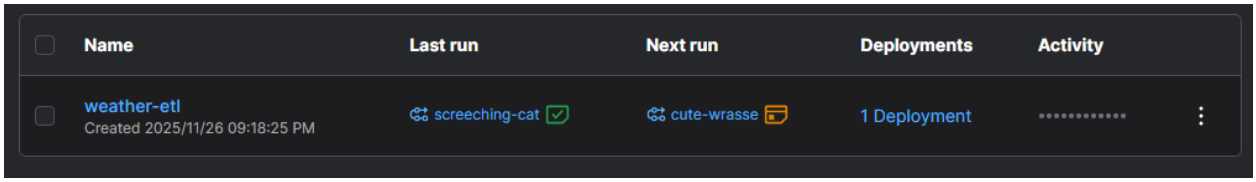


Рисунок 9 – Развертывание weather-etl flow

В flow (см. рис. 10) мы видим полный пайплайн, который мы используем. Тут видно и сохранение данных в clickhouse, о котором мы говорили и отправка сообщений в телеграм через бота. Также видно, что мы сохраняем сырые данные в бакет в minio (см. рис. 11). На рисунке 12 продемонстрированы логи из Prefect.

Рисунок 10 – Flow для двух городов

Рисунок 11 – Сохраненные в бакет данные в minio

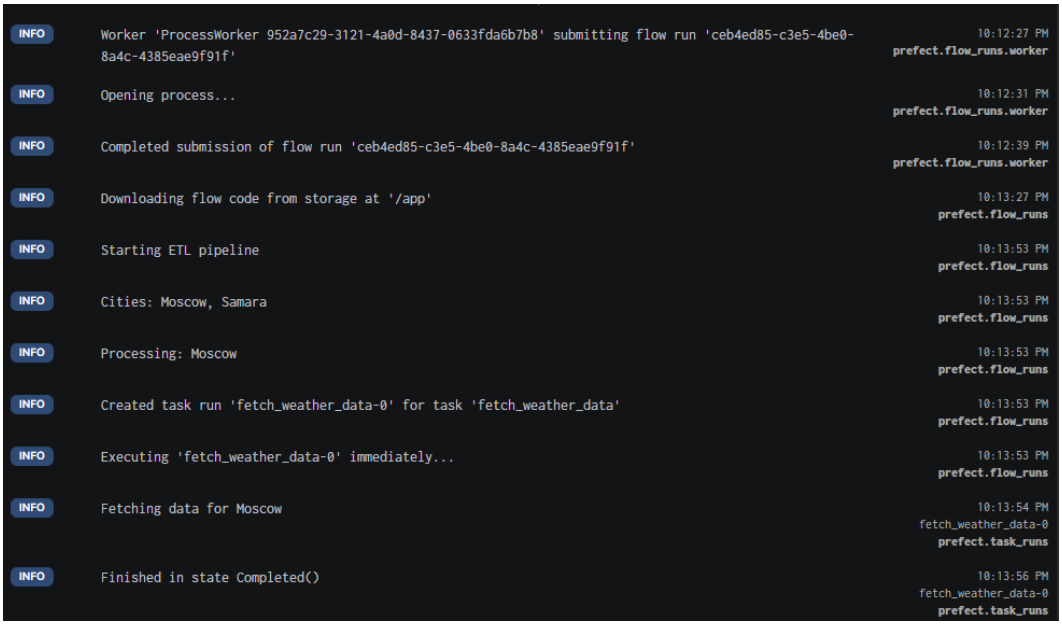


Рисунок 12 – Логи в Prefect

В процессе тестирование я отправлял данные через бота каждую минуту. На рисунке 13 представлены сообщения от бота с данными двух городов каждую минуту.

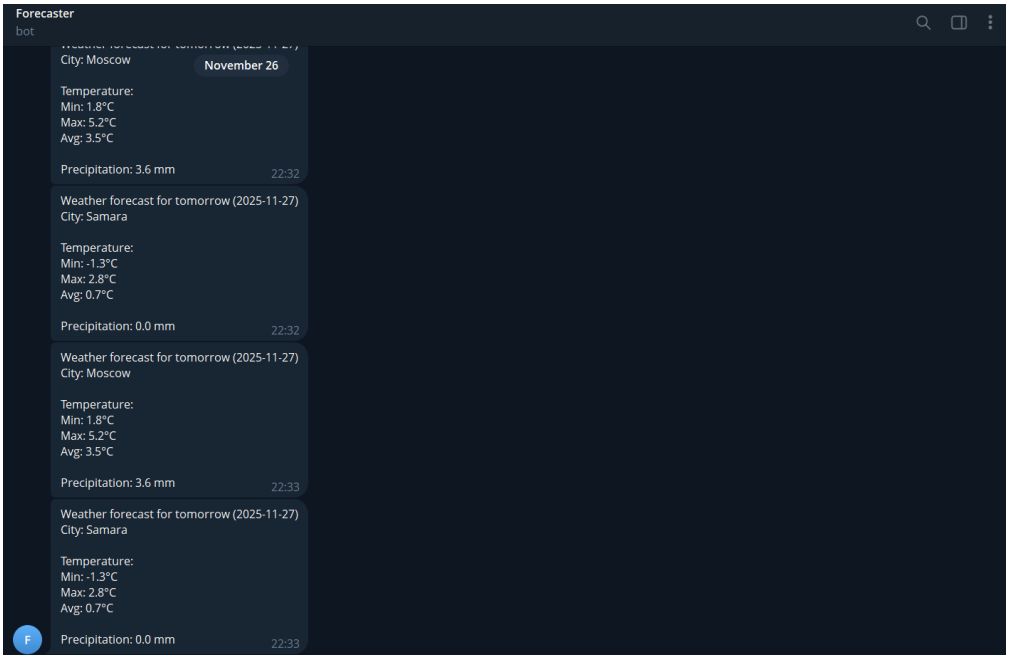


Рисунок 13 – Сообщения от бота в телеграме

ВЫВОДЫ И ИТОГИ

Данная лабораторная работа демонстрирует полный цикл работы с данными (от их получения из внешнего API до визуализации и оповещений). Это достаточно простой пример, на котором можно строить data пайплайны для очень сложных production проектов. Я ни разу не сталкивался с этими инструментами и мне было интересно осваивать новый стек технологий. Теперь, если мне когда-нибудь понадобится сделать бота в телеграм я это смогу сделать.

Я пишу этот отчёт уже после выполнения 2 и 3 лабораторных. Эта лабораторная далась мне достаточно легко и никаких проблем в процессе у меня не возникало. В начале пришлось повозиться, чтобы понять, как настроить рабочий docker-compose и какие версии использовать. В остальном это очень простые инструменты, с которыми было приятно работать.

Больше всего мне понравился Prefect с его удобной настройкой flow и логированием каждого шага. Если какой-то процесс у меня отрабатывал неправильно, то я быстро находил ошибку. Clickhouse и minio удобные хранилища данных, но с ними меньше пришлось взаимодействовать.

Говоря о том, что можно было бы добавить или расширить. Во время выполнения проекта я хотел поделиться с друзьями своим телеграм ботом и для этого разобрался, как отправлять запросы не только на свой чат айди. Наверное, можно было бы улучшить сам текст сообщения в телеграме и сделать его визуально лучше. В базах данных настроить очистку данных годовой давности, чтобы использовать меньше места.

В заключение, мне кажется, что это был очень интересный и полезный опыт. Я считаю, что эти инструменты мне ещё пригодятся для работы или хобби.