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 6232

Студент Н.В. Носов

(*подпись*)

Преподаватель Р.А. Парингер

(*подпись*)

Самара 2025

АРХИТЕКТУРА

Пайплайн представляет собой классический ETL-процесс, оркестрируемый с помощью Prefect. Он спроектирован для работы в контейнерной среде Docker, что обеспечивает изоляцию и воспроизводимость. На рисунке 1 представлена схема пайплайна.

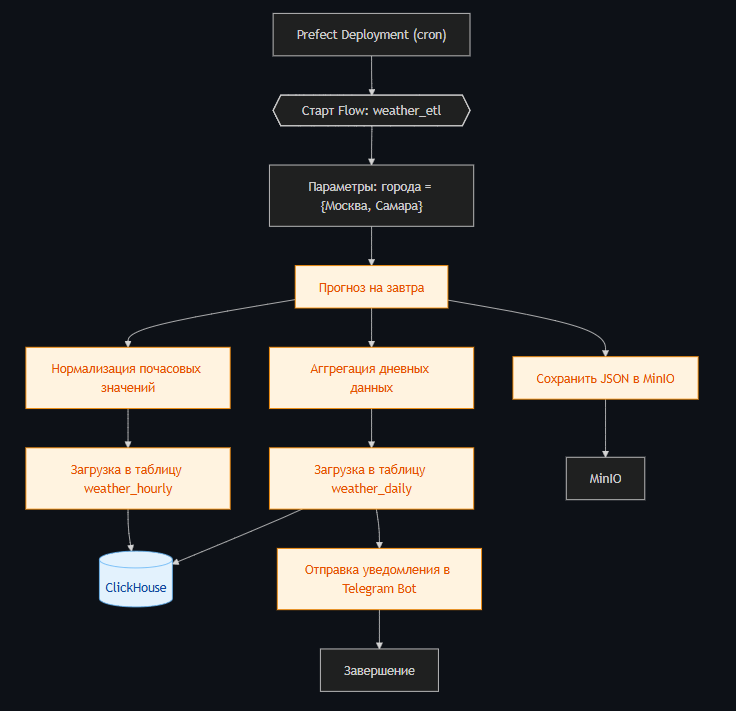


Рисунок 1 – Схема пайплайна

Логическая схема процесса выглядит следующим образом:

1. Планировщик Prefect по расписанию (cron) инициирует запуск потока (flow).

2. Worker Prefect получает задание и последовательно выполняет задачи (tasks) для каждого города из списка:

Extract: Запрос данных о погоде из внешнего API (Open-Meteo).

Stage & Load (Raw): Сохранение необработанного ответа в формате JSON в S3-совместимое хранилище MinIO.

Transform: Преобразование JSON-данных в два структурированных датафрейма (почасовой и дневной) с помощью Pandas.

Load (Transformed): Загрузка обработанных датафреймов в две таблицы в аналитическую базу данных ClickHouse.

Notify: Отправка итоговой сводки в Telegram.

3. БД PostgreSQL используется как бэкенд для самого Prefect, храня состояние потоков и запусков.

4. Redis выступает в роли брокера сообщений для быстрой коммуникации между сервисами Prefect.

Инструменты и их назначение

Prefect: Выбран в качестве оркестратора из-за его гибкости, удобного UI для мониторинга, встроенных механизмов повторных попыток (retries), а также безопасного управления конфигурациями и секретами через Blocks.

Docker/Docker Compose: Обеспечивает создание изолированной, переносимой и легко масштабируемой среды для всех компонентов системы, от баз данных до воркеров.

MinIO: Используется как Data Lake (или Staging Area) для хранения сырых данных. Это позволяет отделить этап извлечения от этапа трансформации и дает возможность повторно обрабатывать данные без обращения к внешнему источнику.

ClickHouse: Выбран в качестве целевой аналитической СУБД (OLAP) благодаря своей высочайшей производительности при выполнении аналитических запросов к большим объемам данных.

Python: Основной язык для написания логики. Библиотеки httpx (для асинхронных HTTP-запросов), pandas (для трансформации данных) и s3fs (для работы с MinIO) являются отраслевыми стандартами для подобных задач.

ИСТОЧНИК ДАННЫХ

Данные извлекаются из публичного API Open-Meteo, который предоставляет бесплатные погодные прогнозы.

Эндпоинт: `https://api.open-meteo.com/v1/forecast`

Параметры запроса:

latitude, longitude: Географические координаты целевого города (например, 55.7558, 37.6176 для Москвы).

hourly: Список запрашиваемых почасовых метрик. В нашем случае это temperature\_2m (температура), precipitation (осадки), wind\_speed\_10m (скорость ветра) и wind\_direction\_10m (направление ветра).

start\_date, end\_date: Период прогноза. В коде он динамически устанавливается на завтрашний день.

Пример запроса через Postman представлен на рисунке 2:

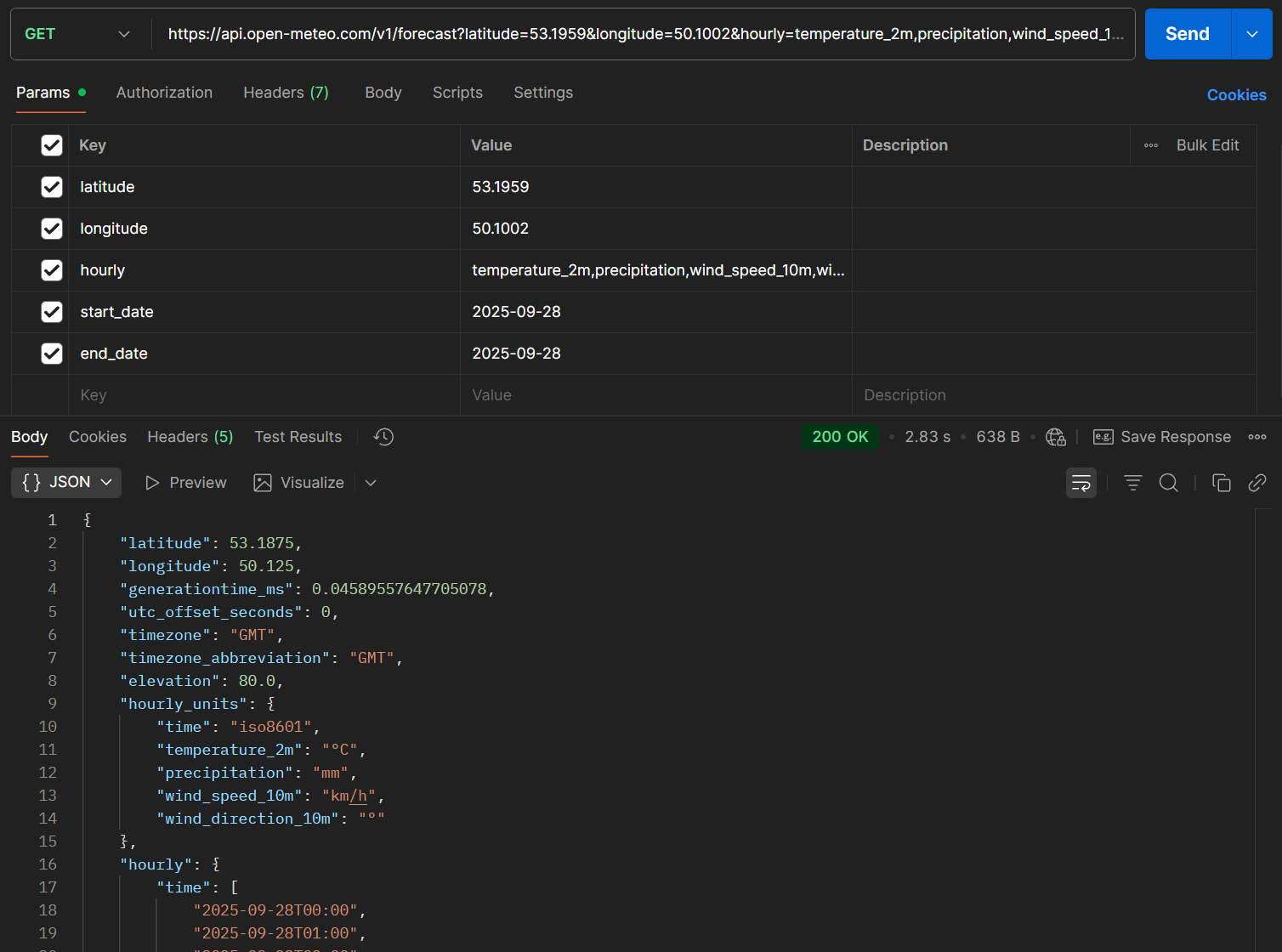


Рисунок 2 – Пример запроса через Postman

ОБРАБОТКА ДАННЫХ

Extract: Этап инициируется задачей fetch\_weather. Она асинхронно отправляет GET-запрос к API Open-Meteo для получения прогноза. В случае сбоя сети или временной недоступности API задача автоматически повторит попытку 3 раза с интервалом в 5 секунд.

Transform: Сырой JSON-ответ обрабатывается в двух задачах. transform\_hourly\_data преобразует его в плоскую почасовую таблицу. Затем transform\_daily\_data агрегирует эти почасовые данные, вычисляя минимальную, максимальную, среднюю температуру и сумму осадков за день.

Load: Процесс загрузки разделен на два потока. Сырой JSON сохраняется в MinIO для архивации и возможных повторных обработок. Трансформированные и очищенные данные в виде датафреймов загружаются в таблицы weather\_hourly и weather\_daily в ClickHouse, где они готовы для анализа.

Реализованные проверки: На данный момент реализована одна базовая проверка: response.raise\_for\_status(). Она гарантирует, что HTTP-запрос к API завершился успешно (код ответа 2xx), что защищает от загрузки пустых или ошибочных данных в случае сбоя API.

Возможные точки сбоя:

1. Если Open-Meteo изменит структуру своего JSON-ответа, задачи трансформации упадут с ошибкой.

2. Несмотря на retries, API может быть недоступен в течение длительного времени.

3. Ошибки в Blocks Prefect для подключения к MinIO, ClickHouse или Telegram приведут к сбою соответствующих задач.

4. Если в ClickHouse изменится схема таблицы (например, тип столбца), задача загрузки load\_to\_clickhouse завершится ошибкой.

РЕЗУЛЬТАТЫ

Лог успешно отработанного флоу в Prefect воркере представлен на рисунке 3.

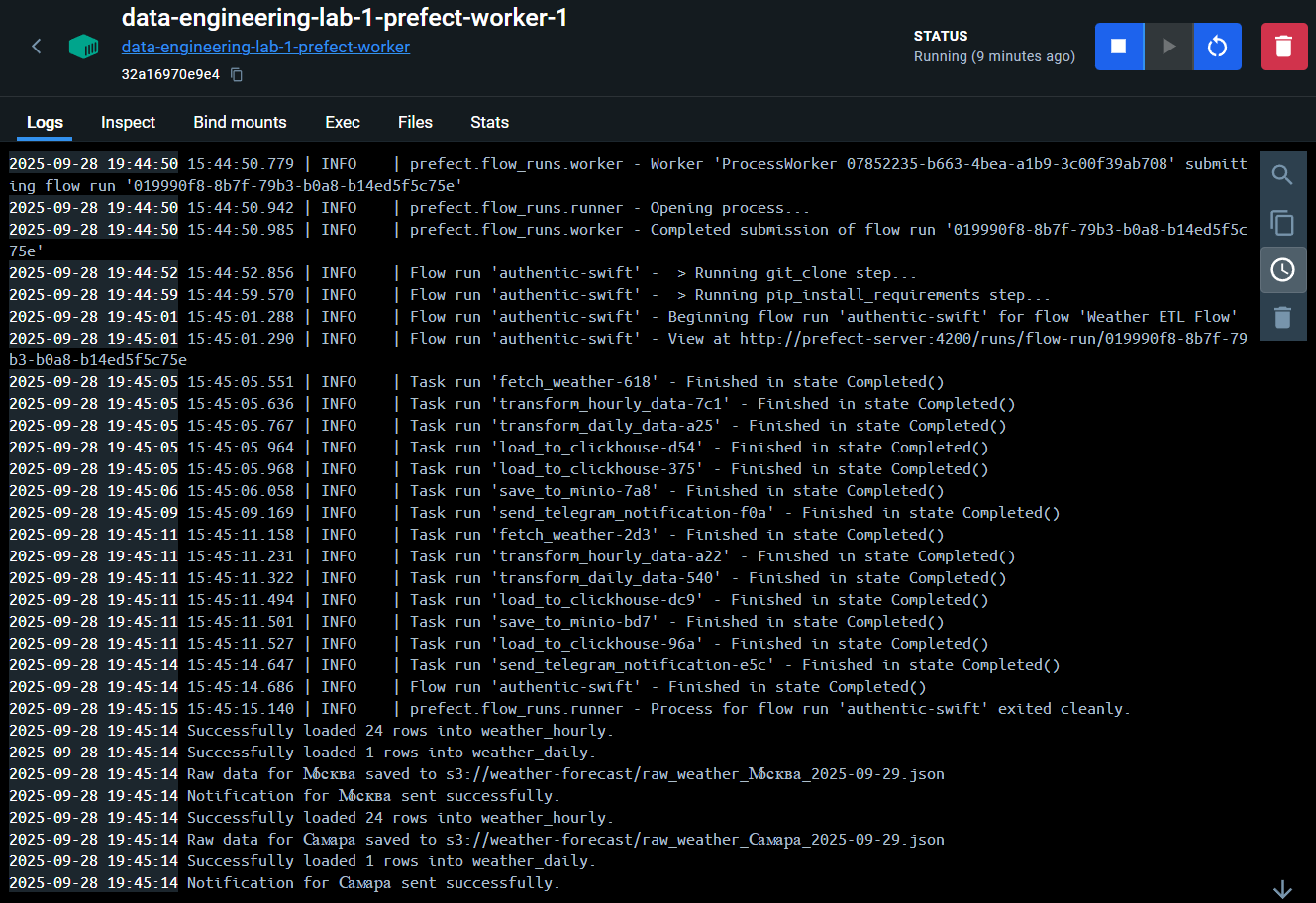


Рисунок 3 – Логи Prefect воркера

На рисунке 4 представлен UI Prefect’a для успешного флоу.

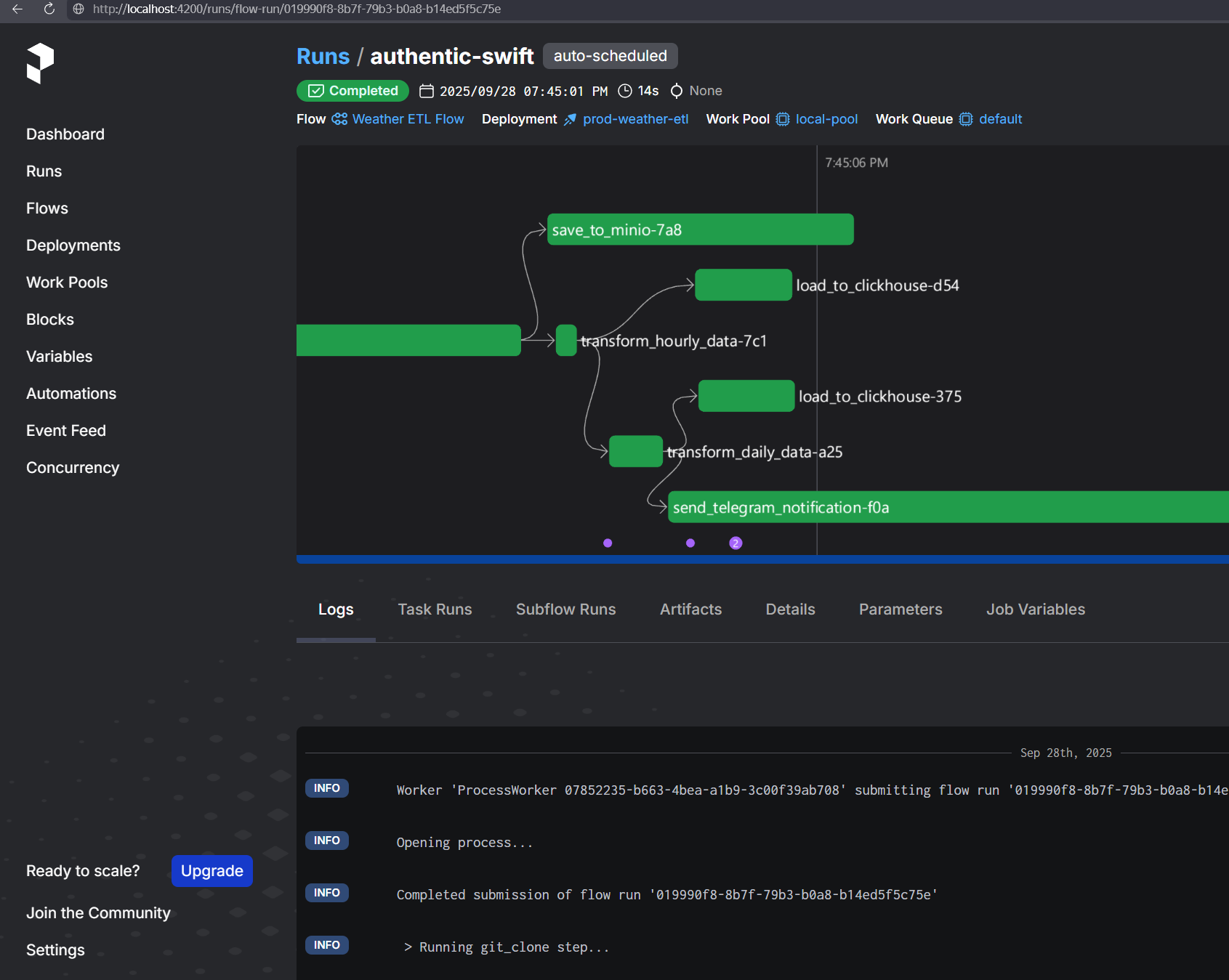


Рисунок 4 – Интерфейс Prefect’а корректно отработанного флоу.

На рисунке 5 представлено содержимое бакета weather-forecast в minio. Видно, что было сохранено два JSON файла с данными по погоде в Самаре и Москве.

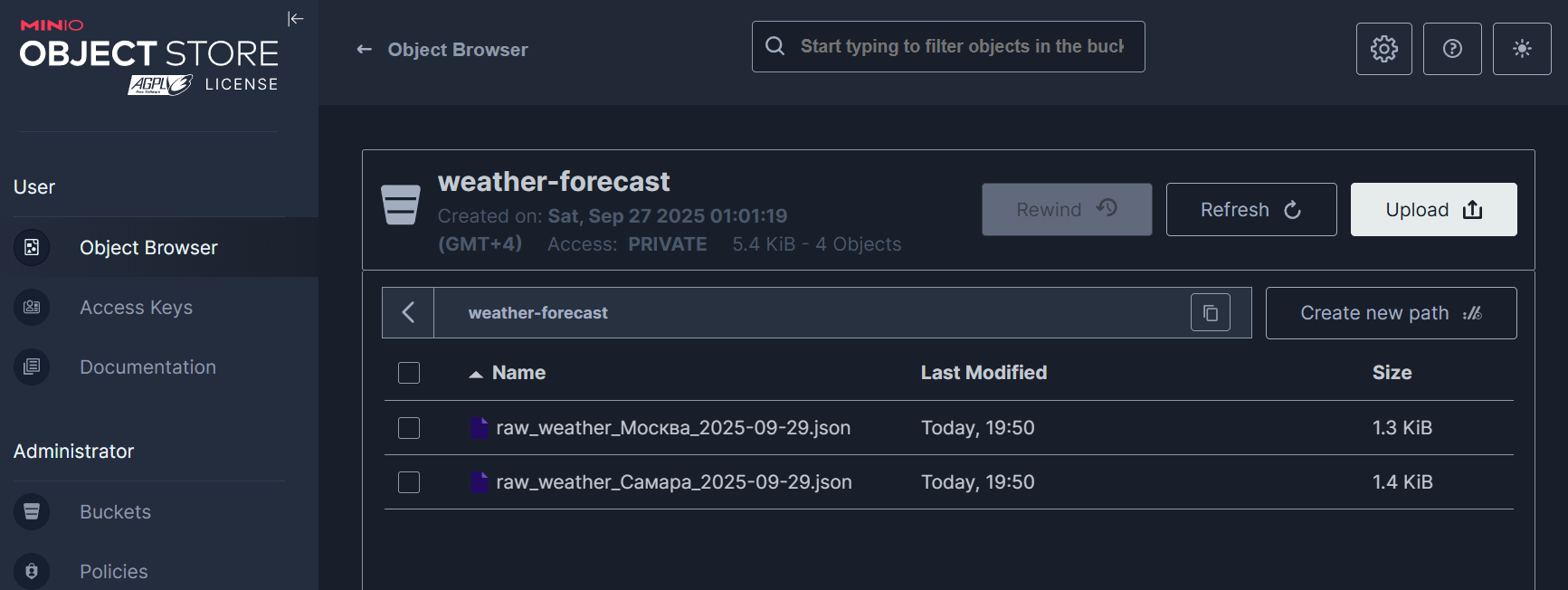


Рисунок 5 – Содержимое бакета weather-forecast в minio

На рисунках 6 и 7 представлены фрагменты данных из таблиц weather\_hourly и weather\_daily из БД ClickHouse.

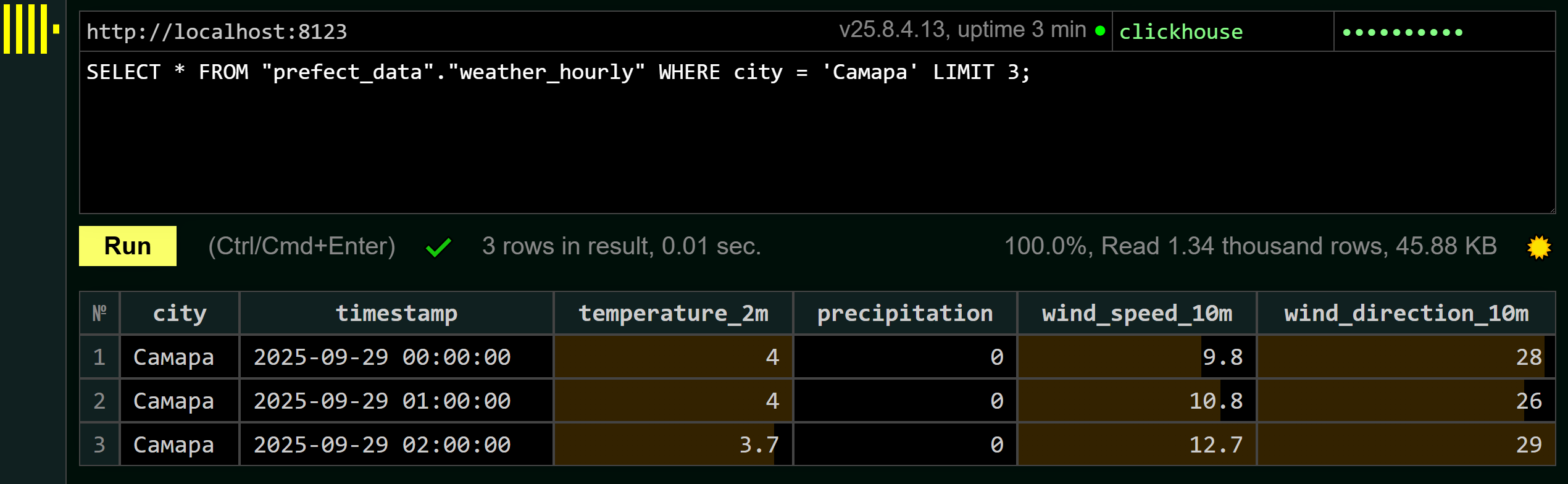


Рисунок 6 – Результат запроса к таблице weather\_hourly в ClickHouse

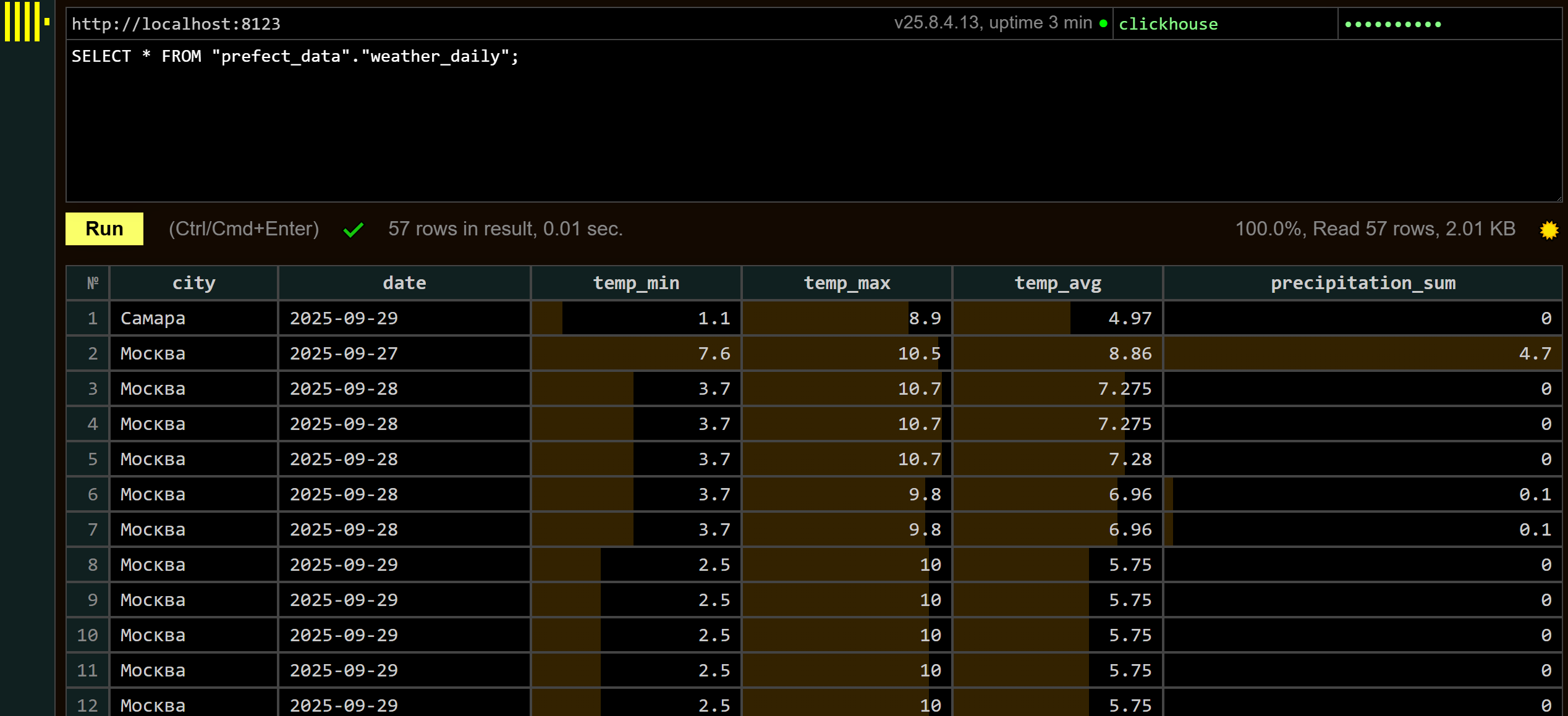


Рисунок 7 – Результат запроса к таблице weather\_daily в ClickHouse

На рисунке 8 представлены уведомления в Telegram, которые приходят на последнем этапе работы флоу.

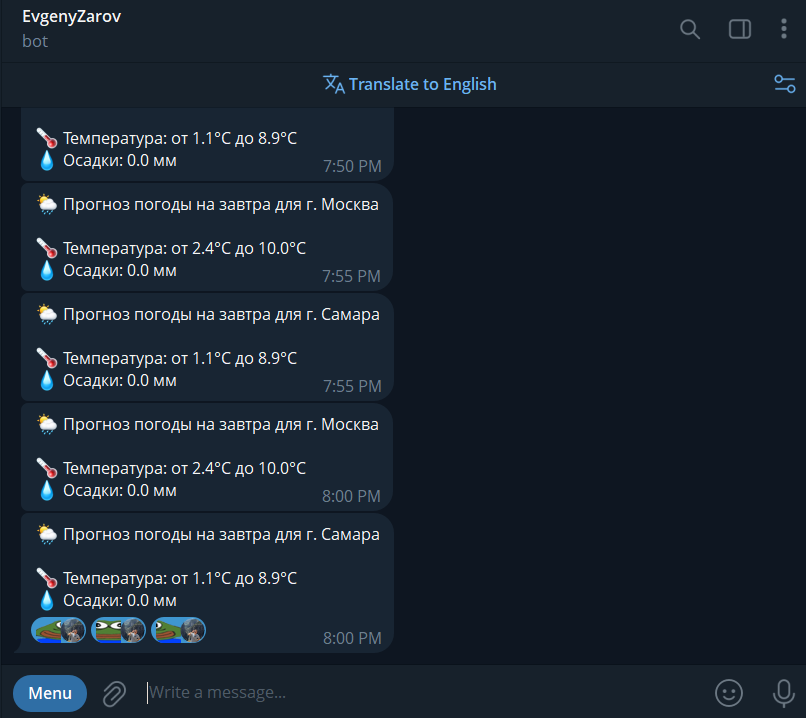


Рисунок 8 – Пример уведомлений в Telegram

ВЫВОДЫ

Самым сложным было скорее поднятие всей инфраструктуры для третьей версии Prefect. В интернете сейчас очень много информации и примеров для старой версии Prefect2, но я решил помучаться и сделать всё на новой третьей версии. Для корректной работы Prefect3 требует также поднятия PostgreSQL и Redis, это можно заметить по примеру docker-compose файла из документации по ссылке: <https://docs.prefect.io/v3/how-to-guides/self-hosted/docker-compose>. Далее была проблема с тем, что при попытке зарегистрировать флоу в Prefect поднимался временный сервис prefect на другом порту и задача регистрировалась не там где я хотел, данная проблема была решена путём создания файла prefect.toml с явным указанием url, где развернут Prefect. Также я решил воспользоваться деплоем конфига через описание в prefect.yaml файле, что породило небольшие проблемы на воркере с отсутствием плагина для клонирования рапозитория, пришлось делать DockerfileWorker, в котором также есть этап установки плагинов prefect-github и prefect-shell, кстати они вроде в итоге не пригодились, но для отладки они были полезны.

Из улучшений можно интегрировать библиотеку типа Great Expectations или Pandera для валидации данных после этапа трансформации. Например, проверять, что температура находится в разумном диапазоне (от -70 до +70) или что в данных нет пропусков. Также в текущей реализации сбой при обработке одного города может потенциально остановить весь flow. Можно обернуть логику внутри цикла в try...except, чтобы изолировать ошибки и позволить пайплайну успешно завершиться для других городов Можно также внедрить инструмент для миграций схемы данных ClickHouse (например, clickhouse-migrations), чтобы изменения в таблицах можно было применять версионно и автоматически.

Подводя итоги, хочу подчеркнуть, что моей целью была настройка процесса, готового к реальному промышленному развертыванию. Я сфокусировался на трех основных аспектах, отличающих просто работающий прототип от надежного решения:

1. Поддерживаемость и CI/CD. Я сознательно отказался от монтирования кода в воркер через volumes. Вместо этого реализовал клонирование кода из Git-репозитория. Такой подход является стандартом в индустрии, так как он обеспечивает контроль версий, упрощает автоматизацию развертывания и делает систему прозрачной для команды.
2. Безопасность. В репозитории полностью отсутствуют какие-либо секретные данные: нет ни файлов .env, ни учетных данных в коде. Вся конфиденциальная информация (ключи API, пароли от баз данных) надежно хранится и управляется через UI Prefect с помощью механизма Blocks.
3. Актуальность технологий. Проект реализован на последних версиях используемых библиотек и инструментов, что гарантирует лучшую производительность, безопасность и долгосрочную поддержку.

Эта лабораторная работа стала ценным опытом, в ходе которого я открыл для себя Prefect как мощное и удобное средство для оркестрации сложных потоков данных.