Система выполнения команд по расписанию на сетевых устройствах

🎯 Цель

Разработать микросервисную систему управления выполнением команд на сетевых устройствах по расписанию, с использованием **Temporal**, **FastAPI**, **асинхронного программирования** и **любой очереди сообщений** (NATS, RabbitMQ, Redis Streams, Kafka и т.д.).

---

1. Архитектура системы

Система состоит из следующих сервисов:

1. **API-сервис** (FastAPI)
    - CRUD для устройств, команд и расписаний (**ресурсно-ориентированный дизайн**).
    - Хранение данных в PostgreSQL (**только этот сервис работает с БД напрямую**).
    - При создании нового активного расписания запускает Workflow в Temporal.
    - Принимает результаты выполнения команд через REST API и обновляет их в БД.
2. **Temporal Worker**
    - Реализует Workflow:
        - Ждёт времени следующего запуска по cron-выражению.
        - Публикует задачу в **брокер сообщений** (`tasks` topic/queue).
        - Ждёт результата (`results` topic/queue).

    - Вызывает REST API API-сервиса для сохранения результата.

3. **Executor-сервис**
    - Получает задачи из **брокера сообщений** (`tasks`), выполняет их (эмуляция выполнения: задержка + фиктивный вывод).
    - Отправляет результат в **брокер сообщений** (`results`).
4. **Message Broker** (на выбор кандидата)
    - Очередь/топик для задач (`tasks`).
    - Очередь/топик для результатов (`results`).
    - **Допустимые варианты**: Redis Streams, RabbitMQ, NATS, Kafka.
5. **PostgreSQL**
    - Хранение данных об устройствах, командах, расписаниях и результатах.

---

2. Структура БД

Devices — сетевое устройство

Хранит данные необходимые для описания устройства: ip, тип , учетные данные и т.д.

Commands — команда для выполнения на устройстве

Хранит список возможных команд

Schedules — расписание выполнения команды на конкретном устройстве

Расписание составляется для конкретной команды с параметрами и конкретного сетевого устройства

CommandResults — результат выполнения команды

Хранит результат выполнения команд для конкретного расписания

---

3. REST API (ресурсно-ориентированное)

Устройства

- `POST /devices` — создать устройство
- `GET /devices` — список устройств
- `GET /devices/{device_id}` — получить устройство
- `DELETE /devices/{device_id}` — удалить устройство

Команды устройства

- `POST /devices/{device_id}/commands` — создать команду для устройства
- `GET /devices/{device_id}/commands` — список команд устройства
- `GET /devices/{device_id}/commands/{command_id}` — получить команду

Расписания команды

- `POST /devices/{device_id}/commands/{command_id}/schedules` — создать расписание
- `GET /devices/{device_id}/commands/{command_id}/schedules` — список расписаний команды
- `GET /devices/{device_id}/commands/{command_id}/schedules/{schedule_id}` — получить расписание

Результаты выполнения

- `POST /devices/{device_id}/schedules/{schedule_id}/result` — обновить результат выполнения команды (вызывается из Temporal Workflow)
- `GET /devices/{device_id}/results` — история выполнения всех команд устройства

---

4. Логика взаимодействия

1. Пользователь создаёт устройство → команду → расписание через API.
2. При создании активного расписания API запускает Workflow в Temporal.
3. Workflow ждёт времени по cron и отправляет задачу в **брокер сообщений** (`tasks`).

4. Executor получает задачу, выполняет команду (эмуляция).
5. Executor отправляет результат в **брокер сообщений** (`results`).
6. Workflow получает результат и отправляет его через REST API в API-сервис.
7. API-сервис сохраняет результат в `CommandResult`.

---

5. Требования к реализации

- Python 3.10+
- FastAPI (async)
- Temporal Python SDK
- PostgreSQL (async ORM: SQLAlchemy async / Tortoise / ...)
- Любой брокер сообщений (NATS, RabbitMQ, Redis Streams, Kafka и т.д.)
- Docker + docker-compose
- Логирование
- Конфигурация через `.env`

---

6. Docker Compose

Обязательно поднять в `docker-compose.yml`:

- API-сервис
- Temporal Server + Temporal Web
- Worker
- Executor
- Message Broker (на выбор кандидата)
- PostgreSQL

---

7. Тестирование

- **Юнит-тесты** для API и функций.
- **Интеграционные тесты** — проверка:
  - Создания расписания.
  - Запуска Workflow.
  - Отправки задачи в брокер.
  - Получения результата.
  - Обновления результата в БД.

---

8. Результат выполнения

- Репозиторий (GitHub/GitLab)
- `README.md` с инструкцией по запуску и описанием API