ФСБ

Постановление № 935 от 01.11.2023: Разбор технических требований и архитектуры СОРМ

Важное уточнение

По указанной дате (01.11.2023) под номером 935 был принят не правительственный документ, а **Приказ Минцифры России № 935**. Данный нормативно-правовой акт утверждает **требования к вычислительной мощности хостинг-провайдеров для внедрения СОРМ** (систем оперативно-разыскных мероприятий).

Если вы искали именно Постановление Правительства РФ, регулирующее эту же сферу (деятельность хостеров и передачу данных ФСБ), то аналогичные нормы зафиксированы в **Постановлении Правительства РФ № 1952 от 22.11.2023**.

Главное о Приказе Минцифры № 935 от 01.11.2023

Документ детально описывает технические правила взаимодействия между ИТ-системами хостинг-провайдеров и спецслужбами для включения компаний в реестр Роскомнадзора:

- **Технические требования**: Хостеры обязаны предоставить ФСБ удаленный доступ к базам данных и вычислительным мощностям.
- **Протоколы связи**: Для обмена командами и выгрузки информации строго прописано использование архитектуры данных **GraphQL**.
- **Собираемые данные**: Определен жесткий перечень логов, сетевых адресов и регистрационных данных пользователей, которые хостер должен хранить и автоматически передавать по запросу ведомств.

Таблица смежных нормативных актов конца 2023 года

Деятельность хостинг-провайдеров регулируется целым пакетом документов, принятых в этот период:

Документ	Основное содержание
Приказ Минцифры № 935 (от 01.11.2023)	Технические требования к базам данных хостеров под нужды СОРМ
Постановление Правительства № 1952 (от 22.11.2023)	Общие правила взаимодействия провайдеров со спецслужбами
Постановление Правительства № 2008 (от 28.11.2023)	Порядок ведения и вхождения компаний в официальный реестр РКН

Пример предоставления данных по регламентам этого постановления

Согласно **Приказу Минцифры № 935 от 01.11.2023**, обмен данными между ИС хостинг-провайдера и техническими средствами ОРМ (спецслужбами) осуществляется автоматически через **язык запросов GraphQL** по протоколу **WebSocket**. Провайдер не передает файлы вручную, а разворачивает API, к которому удаленно обращается система СОРМ.

Ниже приведен абстрактный технический пример того, как выглядит типичный запрос спецслужб к базе хостера и структурированный ответ на него в соответствии с регламентом.

Сценарий: Запрос данных клиента по IP-адресу

Сотрудник ведомства через пульт управления СОРМ хочет узнать, кому принадлежит виртуальный сервер (VPS), с которого фиксировалась активность под конкретным IP-адресом.

---

1. Исходящий запрос от СОРМ (GraphQL-запрос)

Информационная система ведомства отправляет стандартизированный запрос, где указывает критерий поиска (например, целевой `ipAddress` и временной отрезок):

graphql

```
query GetAbonentByIP {
  # Запрос информации о пользователе ИС хостинга
  findUsersByNetworkActivity(
    ipAddress: "198.51.100.42"
    timestamp: "2026-07-09T12:00:00Z"
  ) {
    id
    registrationDate
    contractNumber
    status
    # Тип пользователя: физическое лицо
    personProfile {
      fullName
      birthDate
      passportSeries
      passportNumber
    }
    # Данные об используемой вычислительной мощности (сервисе)
    activeServices {
      serviceType
      allocatedIpAddresses
      domainNames
    }
  }
}
```

---

2. Автоматический ответ хостинг-провайдера (GraphQL-ответ)

Вычислительные мощности провайдера (специальный модуль COPM) обрабатывают этот запрос, обращаются к внутренней биллинговой системе и логам, а затем возвращают строго структурированный JSON-ответ:

json


```
{
  "data": {
    "findUsersByNetworkActivity": [
      {
        "id": "usr_948201",
        "registrationDate": "2024-03-15",
        "contractNumber": "XCT-2024/77",
        "status": "ACTIVE",
        "personProfile": {
          "fullName": "Иванов Петр Сергеевич",
          "birthDate": "1990-05-24",
          "passportSeries": "4512",
          "passportNumber": "654321"
        },
        "activeServices": {
          "serviceType": "VPS_HOSTING",
          "allocatedIpAddresses": ["198.51.100.42"],
          "domainNames": ["example-target-site.ru"]
        }
      }
    ]
  }
}
```

Какие еще данные предоставляются по схемам Приказа № 935:

- **Логи авторизации**: Информация о дате, времени и IP-адресах, с которых сам клиент заходил в личный кабинет хостинга или по SSH/FTP на свой сервер.
- **Платежные данные**: Номера банковских карт, кошельков или счетов, с которых производилась оплата услуг хостинга.
- **История изменений**: Сведения о смене DNS-записей, привязке новых доменов или изменении конфигурации арендуемых серверов.

Если вам нужны **конкретные схемы типов (Type Definitions)** из приложений к приказу или технические условия по организации каналов связи, уточните задачу.

Есть ли в этом регламенте требования к полям ответов?

Да, в Приказе Минцифры № 935 зафиксированы жесткие и исчерпывающие требования к полям ответов. Документ содержит приложения, которые представляют собой готовую **техническую спецификацию (схему GraphQL)**. Провайдер хостинга не имеет права менять названия полей, их типы или структуру по своему усмотрению.

Все поля ответов делятся на строго регламентированные категории:

1. Требования к типам данных (Data Types)

Каждое поле в ответе должно строго соответствовать базовым типам GraphQL или специальным скалярам:

- **Идентификаторы**: Поля типа `id` должны иметь тип `ID!` (уникальная строка, обязательная к заполнению).
- **Временные метки**: Все поля времени (регистрация, авторизация, логи) должны передаваться в формате ISO 8601 (строгий тип `DateTime`), например: `2026-07-09T12:00:00Z`.
- **Сетевые параметры**: Для IP-адресов регламент требует поддержки типов данных под IPv4 и IPv6.

2. Обязательность заполнения полей (Non-Null Fields)

В схеме приказа множество полей отмечены знаком `!`. Это значит, что если уполномоченный орган запрашивает запись, хостер **обязан** вернуть значение в этих полях. Возврат значения `null` (пустоты) по ним считается нарушением регламента. К обязательным полям относятся:

- Внутренний ID пользователя в системе хостинга.
- Дата заключения договора/создания аккаунта.
- Статус учетной записи (активен, заблокирован).

3. Строгая структура вложенности (Объекты)

Регламент запрещает отдавать данные «плоским» текстом. Ответ должен быть строго нормализован:

- Данные о физическом лице должны быть упакованы исключительно в объект профиля (например, `personProfile`) со строго фиксированными дочерними полями: `fullName`, `passportSeries`, `passportNumber`.
- Данные о юридическом лице — в объект организации с полями `inn`, `ogrn`, `companyName`.

4. Использование жестких перечислений (Enum)

Для многих полей ответов регламент устанавливает закрытый список возможных текстовых значений. Хостер не может написать статус «в бане» или «отключен за неуплату». Система должна возвращать строго определенные в API маркеры, например:

- Для статуса аккаунта: `ACTIVE`, `SUSPENDED`, `TERMINATED`.
- Для типа сервиса: `VPS`, `DEDICATED`, `SHARED_HOSTING`.

Если вам необходимо узнать, как в регламенте описываются поля для **конкретного типа данных** (например, платежных транзакций или логов подключения к панели управления), напишите, и мы разберем эти структуры.

Сколько вариантов запросов может поступить по этому регламенту?

В рамках технического регламента Приказа Минцифры № 935 от 01.11.2023 на сервер хостинг-провайдера может поступить **около 20 основных базовых типов GraphQL-запросов (Query)**. Поскольку архитектура GraphQL гибка и позволяет комбинировать аргументы, количество *комбинаций* запросов исчисляется сотнями, но сам жестко зафиксированный перечень целевых поисковых функций (методов) ограничен.

Все регламентированные типы запросов от пульта ФСБ делятся на **5 ключевых категорий**, соответствующих приложениям к приказу:

1. Запросы по идентификации клиентов (Абонентов)

Это базовые поисковые функции для сопоставления виртуальных ресурсов с реальными личностями или компаниями. Спецслужбы могут запрашивать:

- Поиск пользователя по **ФИО** / наименованию организации.
- Поиск по паспортным данным или **ИНН** / ОГРН.
- Поиск по контактным данным (номер телефона, e-mail).
- Поиск по **номеру договора** или уникальному ID в биллинге хостера.

2. Запросы по сетевым идентификаторам и ресурсам

Позволяют установить владельца цифрового следа. Поступают в случаях, когда известен только адрес в сети:

- Поиск по **IP-адресу** (выделенному под VPS, выделенный сервер или shared-хостинг) на определенный момент времени.
- Поиск по **доменному имени**, размещенному на серверах провайдера.
- Поиск по MAC-адресу или внутреннему сетевому интерфейсу.

3. Запросы истории авторизаций и сессий управления

Применяются для отслеживания действий администраторов сайтов:

- Запрос логов входа в **личный кабинет** хостинга.
- Запрос истории подключений к панелям управления (cPanel, ISPmanager и др.).
- Запрос сессий удаленного доступа по протоколам **SSH, FTP, RDP** к арендуемым вычислительным мощностям.

4. Финансовые и транзакционные запросы

Служат для установления источников финансирования интернет-ресурсов:

- Запрос истории **платежей** пользователя за услуги хостинга.
- Поиск клиента по номеру банковской карты или электронного кошелька, с которых поступала оплата.
- Запрос детализации по конкретной банковской транзакции.

5. Служебные (системные) запросы

Используются пультом управления СОРМ для проверки работоспособности самого модуля интеграции:

- Запрос на проверку связи (**Ping / Heartbeat**).
- Запрос актуальной схемы данных (`getSchema`) для подтверждения соответствия версии API.
- Запросы статистики и объемов хранящихся логов.

Как обычно реализовывают обмен по этому постановлению?

Архитектура обмена данными строится именно по принципу предоставления API со стороны наблюдаемого хостинга для оборудования ФСБ. На техническом сленге интеграторов эта схема называется ИС СОРМ (Информационная Система СОРМ).

Однако, вопреки классическому веб-развитию, это **не обычный HTTP API с GET/POST-запросами**, а гибридная схема на базе **WebSockets и HTTP**, где в качестве языка общения строго закреплен **GraphQL**, а данные упакованы в формат **JSON**.

Как это устроено технически: пошаговая схема

1. **Хостер разворачивает ИС (базу данных и API)**: Провайдер своими силами (или покупая готовое ПО у сертифицированных СОРМ-интеграторов, например, «Ящер-935») разворачивает специальный программный модуль. Этот модуль имеет прямой доступ к биллингу, логам серверов, базам авторизации и персональным данным.
2. **Оборудование ФСБ подключается к хостеру**: На стороне хостинга организуется выделенный, защищенный шифрованием (часто по ГОСТ) канал связи. Пульт управления ОРМ (оборудование спецслужб) выступает в роли **клиента**, который подключается к API хостинга.
3. **Запросы поступают от ФСБ к хостеру**: Сотрудник ведомства через свой интерфейс отправляет запрос. Пульт ФСБ генерирует GraphQL-запрос и отправляет его в ИС хостинга. Провайдер обрабатывает его «на лету» и возвращает ответ в JSON.

Почему это НЕ обычный HTTP (GET/POST)?

Хотя GraphQL технически можно гонять через стандартные `POST`-запросы (и в некоторых служебных методах приказа HTTP используется), **основным транспортом для оперативного взаимодействия по Приказу № 935 является протокол WebSocket**.

Критерий	Как в обычном Web	Как реализовано в СОРМ (Приказ № 935)
Протокол (Транспорт)	HTTP / HTTPS (постоянное переоткрытие соединений)	**WebSocket** (устанавливается одно постоянное полнодуплексное соединение)
Методы	GET / POST / PUT / DELETE	**GraphQL операции**: `Query` (запросы) и `Subscription` (подписки на события в реальном времени)
Формат данных	Любой (JSON, XML, Form-Data)	Строго **JSON** (согласно синтаксису GraphQL-ответов)

Зачем спецслужбам понадобился WebSocket и GraphQL?

- **Подписки (Subscriptions)**: Обычный HTTP API требует постоянного опроса (polling) сервера: *«Появились новые логи? А сейчас? А сейчас?»*. WebSocket позволяет пульту ФСБ один раз подписаться на событие. Как только злоумышленник авторизуется в

панели хостинга, сервер хостера **сам моментально проталкивает (push)** JSON-пакет в сторону ведомства.
- **Экономия трафика и гибкость**: GraphQL позволяет пульту ФСБ запрашивать ровно те поля, которые нужны в данный момент. Если оператору нужен только ИНН компании, сервер хостера не будет собирать и упаковывать тяжелые логи её серверов. Это снижает нагрузку на вычислительные мощности провайдера.

Резюме для разработчика

Чтобы соответствовать регламенту, вам нужно написать или внедрить приложение, которое:

- Открывает защищенный порт наружу (только для IP-адресов пульта ФСБ).
- Поднимает **WebSocket-сервер**.
- Умеет парсить входящий **GraphQL-синтаксис** (валидировать его по жесткой схеме из Приказа Минцифры).
- Отдает структурированный **JSON** в рамках открытой WebSocket-сессии.

Если вы планируете реализовывать это самостоятельно, уточните: **какой стек технологий или язык программирования** вы используете в биллинге? Я подкажу, какие готовые библиотеки для работы с GraphQL/WebSockets ускорят прохождение проверок.

Используем 1С. В последних версиях платформы присутствуют изменения по WebSockets

Использование встроенных механизмов **платформы «1С:Предприятие» (начиная с версии 8.3.27)** значительно упрощает интеграцию, однако здесь есть важнейший архитектурный нюанс.

Платформа 1С умеет выступать только в роли WebSocket-клиента, а не сервера. В дереве метаданных и встроенном языке появились объекты `Общие -> WebSocket-клиенты`, `WebSocketКлиентСоединения` и `WebSocketКлиенты`. Но по Приказу Минцифры № 935 **хостинг обязан поднять именно WebSocket-сервер**, к которому пульт ФСБ будет подключаться в

качестве клиента.

Прямо «из коробки» запустить прослушивание WebSocket-порта средствами сервера 1С не получится. Чтобы решить эту задачу, используют одну из двух проверенных архитектурных схем.

Схема 1. Использование прокси-сервера (Рекомендуемая)

Поскольку 1С идеально умеет работать со стандартными **HTTP-сервисами**, между пультом ФСБ и 1С ставится промежуточный легковесный веб-сервер.

1. **Транспортный узел (Nginx / Node.js / Go)**: Снаружи открыт порт, который слушает WebSocket-подключения от спецслужб. Он берет на себя авторизацию, удержание постоянного соединения (keep-alive) и шифрование.
2. **Преобразование**: Когда от пульта ФСБ по WebSockets прилетает текстовый GraphQL-запрос, прокси-сервер пересылает его в 1С через обычный внутренний `POST`-запрос на стандартный HTTP-сервис 1С.
3. **Обработка в 1С**: 1С принимает GraphQL (строку), парсит JSON, запрашивает данные из базы, формирует ответный JSON и отдает его HTTP-сервису. Прокси-сервер перенаправляет этот JSON обратно в WebSocket-канал.

Плюсы: Безопасно, не нагружает сеансы 1С удержанием «пустых» соединений, легко масштабируется.

Схема 2. Использование Системы взаимодействия (Enterprise-подход)

Если в вашей инфраструктуре развернут **Сервер системы взаимодействия 1С** (1С:Управление конференциями / 1С:Диалог), задачу можно решить стандартными компонентами.

- Сервер системы взаимодействия внутри себя работает именно на WebSockets и Java.
- Вы можете опубликовать внешние точки подключения (через Webhooks или интеграционные API Системы взаимодействия).

Однако этот способ сложнее адаптировать под жесткие требования к JSON-структурам ответов GraphQL, прописанным в приказе.

Как реализовать GraphQL-логику внутри 1С?

Вне зависимости от того, как WebSocket-запрос попадет в 1С, вам придется обрабатывать спецификацию GraphQL средствами встроенного языка. Для этого используются три механизма платформы:

1. Чтение и Запись JSON

GraphQL-запрос приходит в теле HTTP-запроса в виде структуры `{"query": "..."}`. Для разбора используется стандартный объект платформы:

bsl

```
Чтение = Новый ЧтениеJSON;
Чтение.УстановитьСтроку(СтрокаЗапросаОтФСБ);
ДанныеЗапроса = ПрочитатьJSON(Чтение);
ТекстGraphQL = ДанныеЗапроса.query; // Здесь будет сам текст GraphQL
```

2. Парсинг GraphQL (Поисковых полей)

Платформа 1С не имеет встроенного валидатора схем GraphQL. На практике разработчики пишут простой парсер строки (через `СтрНайти` / `СтрРазделить` или регулярные выражения через `RegExp`), чтобы вытащить:

- Название метода (например, `findUsersByNetworkActivity`)
- Параметры поиска (`ipAddress`, `timestamp`)

3. Формирование строгого ответа

Чтобы вернуть ответ в точном соответствии с регламентом (с учетом обязательных полей и типов данных), структура собирается через `Соответствие` или `Структура` и сериализуется в JSON:

bsl

```
// Формируем профиль физлица строго по схеме приказа
ПрофильФизЛица = Новый Структура;
ПрофильФизЛица.Вставить("fullName",        Выборка.ФИО);
ПрофильФизЛица.Вставить("passportSeries",  Выборка.СерияПаспорта);
ПрофильФизЛица.Вставить("passportNumber",  Выборка.НомерПаспорта);

// Собираем итоговый пакет
ДанныеАбонента = Новый Структура;
ДанныеАбонента.Вставить("id",            Строка(Выборка.Ссылка.УникальныйИдентификатор()));
ДанныеАбонента.Вставить("contractNumber", Выборка.НомерДоговора);
```

```
ДанныеАбонента.Вставить("personProfile",  ПрофильФизЛица);

// Оборачиваем в массив, как требует GraphQL
РезультатПоиска = Новый Массив;
РезультатПоиска.Добавить(ДанныеАбонента);

ОтветGraphQL = Новый Структура;
ОтветGraphQL.Вставить("data", Новый Структура("findUsersByNetworkActivity", РезультатПоиска));

// Записываем в JSON
ЗаписьJSON = Новый ЗаписьJSON;
ЗаписьJSON.УстановитьСтроку();
ЗаписатьJSON(ЗаписьJSON, ОтветGraphQL);
СтрокаОтветаJSON = ЗаписьJSON.Закрыть();
```

Если вы склоняетесь к архитектуре с **Nginx / Node.js проксированием** перед HTTP-сервисами 1С, напишите, и мы детальнее разберем, как **настроить маршрутизацию** или обрабатывать **Subscriptions (подписки на события)**, когда 1С должна сама инициировать отправку логов при их появлении.

Мне нравится схема 1. Покажи пример конфигурации nginx под него

Для реализации первой схемы **Nginx** подходит идеально. Он будет выступать в роли веб-сервера, который принимает постоянное WebSocket-соединение от пульта ФСБ, «приземляет» его и пересылает запросы на стандартный **HTTP-сервис 1С** (публикацию вашей информационной базы).

Поскольку в GraphQL-запросах по WebSockets обмен идет текстовыми сообщениями (фреймами), обычному Nginx для полноценной конвертации WebSocket-сообщений в HTTP-запросы `POST` к 1С требуется дополнительный скрипт-прослойка (например, микросервис на Node.js или Python).

Ниже приведен готовый пример архитектуры, где **Nginx** принимает трафик, защищает его, а легковесный скрипт **Node.js** передает JSON-данные в **1С**.

Архитектурная цепочка данных:

`Пульт ФСБ (WebSocket)` ➡ `Nginx (Порт 443)` ➡ `Node.js прокси (Порт 3000)` ➡ `HTTP-сервис 1С (Порт 80/443)`

---

1. Конфигурация Nginx (`nginx.conf`)

Этот конфиг решает главные задачи: включает шифрование, разрешает только определенные IP-адреса ведомства и проксирует WebSocket-протокол.

nginx

```
# Ограничение по IP: разрешаем доступ только пульту ФСБ
geo $permitted_fsb {
    default        0;
    195.178.0.0/24 1; # Замените на реальные IP-адреса пульта ведомства
    127.0.0.1      1; # Для локальных тестов
}

server {
    listen 443 ssl;
    server_name sorm.your-hosting.ru;

    # Блокируем всех, кто не входит в белый список IP
    if ($permitted_fsb = 0) {
        return 403;
    }

    # Настройки SSL/TLS (Требование безопасности СОРМ)
    ssl_certificate     /etc/ssl/certs/sorm_hosting.crt;
    ssl_certificate_key /etc/ssl/certs/sorm_hosting.key;
    ssl_protocols       TLSv1.2 TLSv1.3;
    ssl_ciphers         HIGH:!aNULL:!MD5;

    # Точка входа для WebSocket (Пульт ФСБ подключается сюда)
    location /graphql-ws {
        # Перенаправляем трафик на внутренний Node.js прокси
        proxy_pass http://127.0.0.1:3000;

        # Магия Nginx для поддержки WebSockets (Upgrade заголовки)
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";

        # Передаем реальный IP-адрес пульта вглубь системы
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # Таймауты для удержания постоянного соединения
        proxy_read_timeout 86400s;
        proxy_send_timeout 86400s;
```

```
    }
}
```

---

2. Скрипт-прослойка Node.js (`proxy.js`)

Так как Nginx сам по себе не умеет парсить внутренности WebSocket-сообщений и вызывать HTTP-сервисы, этот крошечный скрипт (всего около 30 строк кода) принимает текстовый GraphQL-пакет из WebSocket и делает обычный `POST`-запрос в вашу 1С.

Для его работы нужно установить две библиотеки: `npm install ws axios`.

javascript

```
const { WebSocketServer } = require('ws');
const axios = require('axios');

// Запускаем WebSocket сервер на внутреннем порту 3000
const wss = new WebSocketServer({ port: 3000 });

wss.on('connection', function connection(ws) {
    console.log('Пульт ФСБ подключился к WebSocket');

    // Слушаем входящие сообщения (GraphQL запросы) от пульта
    ws.on('message', async function message(data) {
        try {
            // Преобразуем входящий текстовый фрейм в читаемый вид
            const incomingQuery = data.toString();

            // Пересылаем запрос в HTTP-сервис 1С
            const responseFrom1C = await axios.post(
                'http://1c-server/base/hs/sorm_api/v1/query', // URL публикации вашего HTTP-серв
                { query: incomingQuery },
                {
                    headers: { 'Content-Type': 'application/json' },
                    auth: { username: 'SormUser', password: 'HardPassword1C' } // Авторизация в
                }
            );

            // Возвращаем JSON-ответ из 1С обратно в WebSocket-канал пульта ФСБ
            ws.send(JSON.stringify(responseFrom1C.data));

        } catch (error) {
            console.error('Ошибка обработки запроса:', error.message);
            ws.send(JSON.stringify({ errors: [{ message: "Внутренняя ошибка ИС хостинга" }] }));
        }
    });
});
```

Что нужно сделать в самой 1С?

В вашей конфигурации 1С создается стандартный **HTTP-сервис** (например, с именем `sorm_api`):

1. В нем объявляется шаблон URL: `v1/query`.
2. Метод: `POST`.
3. В обработчике метода вы пишете:

bsl

```
Функция v1queryPOST(Запрос)
    // Читаем входящий GraphQL-запрос, переданный из Node.js
    ТелоСтрокой = Запрос.ПолучитьТелоКакСтроку();

    // ... Ваша логика парсинга, поиска в БД и формирования структуры ответа ...

    // Возвращаем результат
    Ответ = Новый HTTPСервисОтвет(200);
    Ответ.УстановитьТелоИзСтроки(СтрокаJSONОтвета, КодировкаТекста.UTF8, ИспользованиеByteOrderM
    Возврат Ответ;
КонецФункции
```

Если вы хотите протестировать эту связку прямо сейчас на тестовом контуре, подсказать, как настроить **эмуляцию пульта ФСБ (клиента)** для проверки отправки запросов, или разобрать логику авторизации по ГОСТ-сертификатам?

Какой наиболее легковесный вариант вместо Node.js можно взять? Можно ли решить задачу с помощью Lua?

Да, вариант с Lua — это самое легковесное, быстрое и элегантное решение. Вам даже не придется запускать отдельный фоновый процесс (как Node.js), так как Lua встраивается прямо внутрь Nginx с помощью модуля **OpenResty** или `ngx_http_lua_module`.

В такой архитектуре Nginx сам превращается в полноценный WebSocket-сервер, парсит пакеты с помощью Lua и мгновенно перенаправляет их в HTTP-сервис 1С. Потребление оперативной памяти снизится со ~150 МБ (для Node.js) до **нескольких мегабайт**.

Архитектурная цепочка данных:

`Пульт ФСБ (WebSocket)` ➡ `Nginx + Lua (Порт 443)` ➡ `HTTP-сервис 1С (Внутренний POST-запрос)`

Инструкция по реализации на Nginx + Lua

Для работы этой схемы вам понадобится сборка Nginx с поддержкой Lua. В современных дистрибутивах Linux (Ubuntu/Debian) достаточно установить пакет **OpenResty** или стандартные пакеты `nginx` и `libnginx-mod-http-lua`. Также потребуется стандартная Lua-библиотека для работы с WebSockets (`lua-resty-websocket`).

1. Конфигурация Nginx со встроенным Lua-скриптом

Вся логика проксирования и подмены протоколов описывается прямо внутри блока `location` вашего `nginx.conf`:

nginx

```
server {
    listen 443 ssl;
    server_name sorm.your-hosting.ru;

    ssl_certificate     /etc/ssl/certs/sorm_hosting.crt;
    ssl_certificate_key /etc/ssl/certs/sorm_hosting.key;

    # Точка входа для пульта ФСБ
    location /graphql-ws {
        # Указываем Nginx, где искать DNS для внутренних HTTP-запросов к 1С
        resolver 8.8.8.8;

        # Вызов встроенного Lua-скрипта
        content_by_lua_block {
            -- Подключаем библиотеки OpenResty для работы с WebSocket и HTTP
            local server = require "resty.websocket.server"
            local http = require "resty.http"
            local cjson = require "cjson"

            -- Инициализируем WebSocket-соединение с пультом
            local wb, err = server:new{
                timeout = 86400000, -- Таймаут удержания соединения (в мс)
                max_payload_len = 65535
            }
            if not wb then
                ngx.log(ngx.ERR, "Ошибка инициализации WS: ", err)
```

```
            return ngx.exit(444)
        end

        -- Запускаем бесконечный цикл ожидания запросов от пульта ФСБ
        while true do
            local data, typ, err = wb:recv_frame()

            -- Если пульт закрыл соединение или произошла ошибка — выходим
            if not data or typ == "close" then
                break
            end

            -- Обрабатываем только текстовые фреймы (GraphQL текстовые запросы)
            if typ == "text" then
                -- Создаем HTTP-клиент для отправки запроса в 1С
                local httpc = http.new()

                -- Формируем JSON-пакет для 1С, упаковывая туда входящий GraphQL
                local payload = cjson.encode({ query = data })

                -- Делаем синхронный POST-запрос к HTTP-сервису 1С
                local res, http_err = httpc:request_uri("http://1c-server/base/hs/sorm_api/v
                    method = "POST",
                    body = payload,
                    headers = {
                        ["Content-Type"] = "application/json",
                        ["Authorization"] = "Basic U29ybVVzZXI6SGFyZFBhc3N3b3JkMUM=" -- Base
                    }
                })

                -- Отправляем ответ из 1С обратно в WebSocket-канал пульта ФСБ
                if res and res.status == 200 then
                    wb:send_text(res.body)
                else
                    ngx.log(ngx.ERR, "1С вернула ошибку: ", http_err)
                    wb:send_text(cjson.encode({ errors = { { message = "Ошибка ИС хостинга"
                end
            end
        end

        wb:send_close()
    }
}
}
```

Почему решение на Lua идеальное для СОРМ?

1. **Минимальный след в системе**: Скрипт выполняется в контексте рабочих процессов (workers) самого Nginx. Ему не нужны Node.js-рантаймы или сторонние менеджеры процессов вроде PM2.
2. **Максимальное быстродействие**: Архитектура LuaJIT (Just-In-Time компилятор в OpenResty) по скорости обработки ввода-вывода практически не уступает чистому Си. Проксирование происходит с субмиллисекундными задержками.
3. **Безопасность**: Чем меньше сторонних пакетов и зависимостей (`node_modules`) крутится на сервере, тем меньше потенциальных уязвимостей в контуре СОРМ, что облегчает сдачу узла связи контролирующим органам.

Если вы решите внедрять OpenResty с Lua, подсказать, как правильно **сгенерировать строку авторизации (Base64)** для заголовков 1С или как организовать **ротацию логов ошибок** самого Nginx, чтобы они не переполнили диск при частых запросах?

Revision #2
Created 2026-07-10 02:53:32 UTC by Admin
Updated 2026-07-10 03:21:31 UTC by Admin