РЕФЕРАТ

Отчёт 50 с., 18 рис., 2 табл.

DOCKER, КОНТЕЙНЕРИЗАЦИЯ, LINUX, МЕДИА-СЕРВЕР, JELLYFIN, АВТОМАТИЗАЦИЯ, VPN, БЕЗОПАСНОСТЬ

В работе рассматривается проектирование и реализация мультиконтейнерной инфраструктуры на базе Docker для управления медиа-контентом и обеспечения безопасного удалённого доступа. Архитектура системы включает набор интегрированных сервисов (Jellyfin, Sonarr, Radarr, Lidarr, Prowlarr, qBittorrent, WireGuard) для полной автоматизации скачивания, обработки и потоковой передачи медиа-контента. Описаны практические аспекты конфигурирования компонентов, включая установку SSL/TLS сертификатов, управление виртуальными сетями Docker и обеспечение безопасности на нескольких уровнях защиты. Представленное решение демонстрирует возможность создания надёжной, расширяемой и полностью контролируемой личной инфраструктуры, альтернативной облачным сервисам, с сохранением конфиденциальности и независимостью от внешних провайдеров.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Традиционный подход к хранению данных полагается на облачные сервисы — удобные, масштабируемые, но контролируемые третьими лицами. Такая модель несёт серьёзные риски:

- **Зависимость от провайдера**. Отключение интернета, блокировка сервиса или закрытие компании означают потерю доступа к данным.
- **Отсутствие контроля над приватностью**. Облачные провайдеры хранят и обрабатывают личную информацию по своим правилам. GDPR и подобные законы лишь ограничивают, но не исключают обработку данных.
- **Постоянные утечки**. Историческое количество инцидентов безопасности показывает, что централизованные хранилища — привлекательная цель для атак.
- **Финансовая зависимость**. Цены растут, условия меняются, сервисы закрываются.

Self-hosted инфраструктура предлагает альтернативу: полный контроль над данными на собственном оборудовании, независимость от внешних сервисов (кроме интернет-провайдера), и способность адаптировать систему под конкретные нужды.

Данная работа описывает проектирование и развертывание домашней лаборатории на базе Docker, которая решает указанные проблемы через комбинацию открытого программного обеспечения, контейнеризации и принципов сетевой безопасности. Результат — полнофункциональная мультиконтейнерная инфраструктура для управления медиа-контентом, мониторинга, и безопасного удалённого доступа.

1 Целевая аудитория и применение

1.1 Для кого предназначена эта работа

Проект ориентирован на три категории пользователей:

- **Студенты и исследователи**. Изучение архитектуры распределённых систем, контейнеризации, сетевой безопасности на практическом примере.
- **Системные администраторы и DevOps инженеры**. Опыт работы с Docker Compose, конфигурированием микросервисов, управлением инфраструктурой.
- **Энтузиасты и продвинутые пользователи**. Создание собственного медиа-сервера с полным контролем, альтернатива облачным сервисам.

1.2 Уровни сложности реализации

Инфраструктура поддерживает прогрессивное усложнение:

- **Базовая конфигурация**. Развертывание медиа-сервера (Jellyfin) и торрент-клиента (qBittorrent) с VPN защитой за 15–20 минут с использованием именованных томов Docker.
- **Промежуточная конфигурация**. Добавление обратного прокси (Nginx Proxy Manager), локальной DNS системы, интеграция с собственным доменом через Cloudflare.
- **Расширенная конфигурация**. Полная автоматизация медиа-контента (Sonarr, Radarr, Lidarr), обработка файлов (FileFlows), мониторинг (Prometheus, Grafana), безопасный удалённый доступ через VPN.

1.3 Основные возможности

Инфраструктура включает набор интегрированных сервисов, каждый из которых решает определённую задачу в экосистеме домашней лаборатории. От управления медиа-контентом до обеспечения безопасного удалённого доступа — система охватывает полный цикл работы с данными. Полный перечень функциональности и соответствующих

компонентов представлен в таблице 1, которая демонстрирует разнообразие возможностей и гибкость архитектуры.

Таблица 1 — Основные возможности домашней лаборатории

Возможность	Назначение	Используемые сервисы
Управление медиа	Автоматизация скачивания контента, управление индексерами и обработка запросов пользователей	Sonarr, Radarr, Lidarr, Prowlarr, Jellyseerr
Потоковая передача	Воспроизведение медиа на различных устройствах, интеграция с Discord	Jellyfin, Discord Music Bot
Защищённое скачивание	Торрент-загрузки через VPN для приватности	qBittorrent, WireGuard
Транскодирование	Конвертация видео и аудио в оптимальные форматы	FileFlows
Обратное прокси	Единая точка входа для всех сервисов с SSL/TLS	Nginx Proxy Manager
Мониторинг инфраструктуры	Сбор метрик и визуализация системы	Prometheus, Grafana, Node Exporter
DNS и управление доменами	Локальное разрешение имён с DOH/TOH и интеграция с внешним доменом	AdGuardHome, Cloudflare
Удалённый доступ	Безопасное подключение с домашней сети извне	WireGuard VPN

2 Архитектура инфраструктуры

2.1 Обзор компонентов

Система состоит из 14 основных сервисов, организованных по функциональным блокам. Каждый компонент работает в изолированном контейнере Docker и взаимодействует с другими через виртуальные сети и API.

Таблица 2 — Компоненты системы и их функции

Сервис	Функция	Порт	Блок
Jellyseerr	Интерфейс запросов	5055	Управление медиа
Sonarr	Автоматизация сериалов	8989	Управление медиа
Radarr	Автоматизация фильмов	7878	Управление медиа
Lidarr	Автоматизация музыки	8686	Управление медиа
Prowlarr	Управление индексерами	9696	Управление медиа
Jellyfin	Потоковая передача медиа	8096	Потоковая передача
Discord Music Bot	Воспроизведение музыки в Discord	3000	Потоковая передача
qBittorrent	Торрент-клиент (через VPN)	8080	Загрузки
FileFlows	Транскодирование видео	5000	Обработка файлов
WireGuard	VPN сервер	51820	Сеть
AdGuardHome	DNS с DOH/TOH	53, 853, 443	Сеть
Nginx Proxy Manager	Обратный прокси + SSL	80, 81, 443	Сеть
Cloudflare	DNS и управление доменом	—	Сеть
Prometheus	Сбор метрик	9090	Мониторинг

Сервис	Функция	Порт	Блок
Grafana	Визуализация графиков	3000	Мониторинг
Node Exporter	Метрики хоста	9100	Мониторинг
Portainer	Управление Docker контейнерами	8000, 9000, 9443	Инфраструктура
Vaultwarden	Хранилище паролей	80	Инфраструктура
Homarr (Dashboard)	Главная страница и управление	7575	Инфраструктура
OpenSpeedTest	Тестирование скорости интернета	3000-3001, 8080	Инфраструктура
Watchtower	Автоматическое обновление контейнеров	—	Инфраструктура

2.2 Сетевая архитектура

2.2.1 Физическая сеть домашней лаборатории

Перед рассмотрением виртуальных сетей Docker необходимо описать физическую инфраструктуру, в которой развёрнута система:

- **Роутер на базе OpenWrt** — маршрутизирует весь трафик домашней сети, управляет правилами firewall, обеспечивает базовую DNS инфраструктуру.
- **Управляемый L3 коммутатор** — обеспечивает физическую связность устройств в сети. Используется VLAN сегментация, однако описана она не будет, так как выходит за рамки простой пользовательской конфигурации. При желании, энтузиаст-читатель этой курсовой работы может изучить этот вопрос самостоятельно.
- **Firewall с маркировкой трафика** — реализован на роутере OpenWrt. Использует списки доменов и CIDR диапазонов для маркировки трафика. Помеченные ресурсы автоматически

направляются в VPN туннель через sign-box. Списки для маршрутизации берутся из проекта podkop.net.

- **Каскадная DNS инфраструктура** — трёхуровневая система разрешения имён:
    1. **dnsmasq** (базовый DNS роутера) — первичная обработка запросов, кеширование, локальные записи
    2. **sign-box DNS** — маркировка доменов для firewall, определение маршрутизации через VPN
    3. **AdGuardHome** — фильтрация рекламного трафика, DOH/DOT к публичным провайдерам

Конфигурация AdGuardHome использует балансировку между несколькими зашифрованными DNS провайдерами:

```
# Основные DNS с шифрованием (DOT/DOH)
upstream:
  - https://dns10.quad9.net/dns-query
  - https://cloudflare-dns.com/dns-query
  - https://dns.google/dns-query
  - tls://dns10.quad9.net
  - tls://1.1.1.1
  - tls://dns.google

# Bootstrap DNS (для разрешения имён основных DNS)
bootstrap-dns:
  - 9.9.9.10          # Quad9 (основной)
  - 149.112.112.10    # Quad9 (резервный)
  - 2620:fe::10       # Quad9 IPv6
  - 2620:fe::fe:10    # Quad9 IPv6 резервный

# Fallback DNS (незашифрованные, крайний случай)
fallback-dns:
  - 1.1.1.1           # Cloudflare
  - 1.0.0.1           # Cloudflare резервный
  - 8.8.8.8           # Google DNS
  - 8.8.4.4           # Google DNS резервный
```

Листинг 1 — Конфигурация DNS балансировщика AdGuardHome

Такая архитектура обеспечивает отказоустойчивость, приватность через шифрование DNS запросов и автоматическую маршрутизацию специфичного трафика через VPN.

2.2.2 Виртуальные сети Docker

Каждый функциональный блок инфраструктуры изолирован в собственной виртуальной сети Docker. Контейнеры внутри одного блока видят друг друга и могут взаимодействовать напрямую по имени хоста, что обеспечивает логическую организацию.

Все остальные виртуальные сети в рамках Docker контейнеров:

- **media-network** — сеть для всех медиа-сервисов (Jellyfin, Sonarr, Radarr, Lidarr, Prowlarr, Jellyseerr, qBittorrent, FileFlows). Обеспечивает взаимодействие между компонентами автоматизации и потоковой передачи.
- **proxiable** — публичная сеть для сервисов, доступных через Nginx Proxy Manager. Позволяет обратному прокси маршрутизировать запросы к нужным контейнерам.
- **monitoring** — выделенная сеть для системы мониторинга (Prometheus, Grafana, Node Exporter). Изолирует сбор метрик от основной инфраструктуры.
- **vpn** — сеть для VPN контейнера WireGuard. Используется qBittorrent для приватных скачиваний через зашифрованное соединение.

Такая организация позволяет контейнерам обращаться друг к другу по имени (например, `http://sonarr:8989`), исключая необходимость в фиксированных IP адресах и упрощая конфигурацию.

2.3 Взаимодействие контейнеров

Инфраструктура содержит множество потоков взаимодействия между сервисами. Ниже описан один из самых сложных флоу — полная автоматизация загрузки и каталогизации медиа-контента, который начинается с пользовательского интерфейса.

- Пользователь подключается к **Nginx Proxy Manager** — единой точке входа для всех интерактивных сервисов. Прокси маршрутизирует запрос к **Jellyseerr** в виртуальной Docker сети.
- Пользователь через **Jellyseerr** отправляет запрос на фильм или сериал
- Запрос автоматически создаётся в **Sonarr** (сериалы) или **Radarr** (фильмы) через REST API
- **Sonarr** / **Radarr** отправляют поисковый запрос в **Prowlarr** (агрегатор индексеров торрентов) для поиска торренто-ссылок
- **Sonarr** / **Radarr** отправляют найденные торренты в **qBittorrent** через REST API для загрузки
- **qBittorrent** работает в виртуальной сети **WireGuard** для обеспечения приватности скачиваний
- Загруженные файлы сохраняются в общем хранилище (**File Storage**)
- **FileFlows** автоматически обрабатывает (транскодирует) видео файлы при необходимости
- **Jellyfin** периодически сканирует хранилище и каталогизирует готовый контент, делая его доступным пользователям через сеть

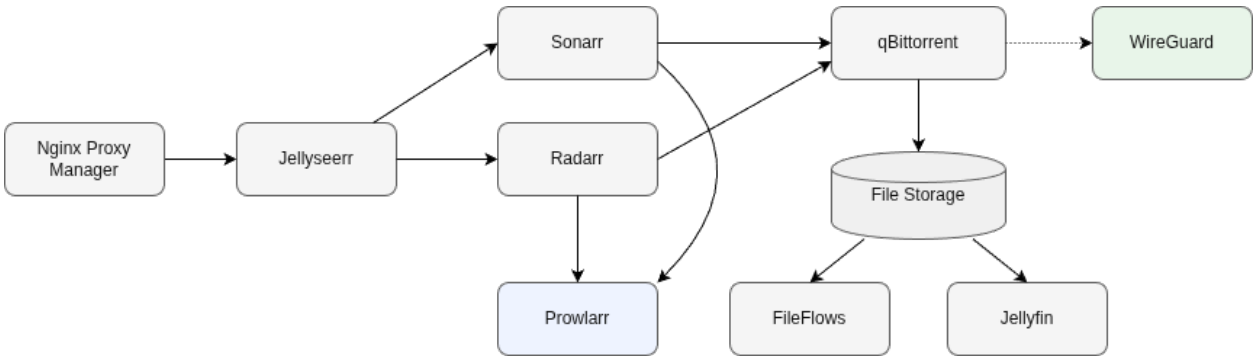

Рисунок 1 — Взаимодействия контейнеров автоматизации медиа-контента

Помимо основного потока, инфраструктура содержит ряд автоматических процессов, недоступных для прямого пользовательского управления, а также простых связок взаимодействий между сервисом и обратным прокси:

- **Watchtower** — мониторит сокет демона Docker, периодически проверяет наличие обновлений для образов контейнеров и автоматически перезапускает сервисы при наличии новых версий.

Такой подход позволяет содержать инфраструктуру в актуальном состоянии без ручного вмешательства.

- **Node Exporter** и **cAdvisor** — собирают метрики системы и контейнеров в реальном времени, отправляя их в **Prometheus** для хранения и анализа. **Grafana** визуализирует эти метрики в виде графиков и дашбордов.
- **Публично доступные сервисы** — ряд компонентов (**Jellyfin**, **Vaultwarden**, **Homarr** и другие) доступны конечному пользователю напрямую через их HTTP/HTTPS порты, проксированные **Nginx Proxy Manager**.

3 Выбор оборудования

3.1 Профиль потребления ресурсов

Перед развертыванием инфраструктуры необходимо понимать реальные требования к оборудованию. В отличие от облачных сервисов, где потребление зачастую не волнует нас, домашняя лаборатория требует явного расчёта ресурсов.

3.1.1 Потребление оперативной памяти

В состоянии покоя (без активных пользователей и обработки) все сервисы инфраструктуры в сумме занимают не более **6 гигабайт оперативной памяти**. Это достаточно скромный объём, что позволяет развертывать систему на оборудовании с 8–16 ГБ ОЗУ.

Основные потребители:

- **FileFlows** (транскодирование). По умолчанию содержит 3 параллельных ранера (worker) для одновременной обработки видео. При транскодировании видеоконтента, даже 4K с высоким битрейтом (80 тыс кбит/с), один ранер занимает примерно **1 гигабайт памяти**, однако нагрузка на центральный процессор отсутствует полностью — вся работа выполняется на видеокарте через NVIDIA NVENC/NVDEC ускорители.
- **Jellyfin** (потоковая передача). Аналогично FileFlows использует NVIDIA GPU ускорители для транскодирования и имеет сопоставимое потребление памяти (около 1 ГБ за сеанс при транскодировании). Оба сервиса используют единый CLI плагин FFmpeg, что обеспечивает идентичное поведение и потребление ресурсов. **Важный момент**: Jellyfin разумно избегает ненужного транскодирования — если контент представлен в кодеке, который может аппаратно воспроизвести клиент (например, HEVC на современных мобильных устройствах), транскодирование не происходит вообще. Дополнительно, Jellyfin кеширует результаты транскодирования, поэтому если множество пользователей одновременно смотрят один и тот же фильм, видеокодек

«прогоняется» только один раз, остальные получают кеш из хранилища.

- **Grafana** (мониторинг) и **Homarr** (дашбоард). Эти сервисы могут занимать значительный объём памяти (до 500 МБ–1 ГБ) вследствие сложного пользовательского интерфейса.
- **Остальные сервисы**. Nginx Proxy Manager просто маршрутизирует трафик и периодически обновляет SSL сертификаты — занимает минимум памяти. Watchtower мониторит сокет демона Docker, периодически проверяет обновления образов и перезапускает контейнеры — также не требователен к ресурсам. Медиа-сервисы (Sonarr, Radarr, Lidarr, Prowlarr, Jellyseerr) выполняют лишь API запросы и управление метаданными — их потребление пренебрежимо мало.

3.1.2 Потребление процессора

Инфраструктура всех приложений спроектирована таким образом, чтобы минимизировать нагрузку на центральный процессор. Большинство операций либо асинхронные (проверка обновлений, сбор метрик), либо вовсе I/O-bound (работа с диском, сетью). Единственным исключением является видеотранскодирование, которое полностью разгружено на видеокарту.

Таким образом, процессор может быть относительно скромным — достаточно 4–6 ядер для обеспечения параллельной работы контейнеров и фоновых задач. При использовании GPU ускорения нагрузка на CPU при транскодировании практически отсутствует.

3.1.3 Требования к хранилищу

Хранилище состоит из двух компонентов:

- **Системный диск** (для Docker и конфигурации). Требуется минимум 100–150 ГБ SSD для хранения образов контейнеров, конфигурационных файлов и баз данных сервисов.
- **Медиа-хранилище** (для контента). Объём зависит от размера коллекции фильмов, сериалов и музыки. Для комфортного использования рекомендуется от 1–2 ТБ и выше. Тип хранилища

может быть как SSD (для максимальной производительности), так и HDD (для экономии бюджета, так как медиа в основном читается последовательно).

3.2 NVIDIA GPU

Если планируется транскодирование видеоконтента или использование аппаратного ускорения потоковой передачи, рекомендуется видеокарта NVIDIA с поддержкой CUDA.

Требования:

- Видеокарта: NVIDIA GeForce GTX 1050 и выше (поддержка NVENC/NVDEC)
- Драйверы: NVIDIA Driver 450.x и выше

Система полностью функциональна и без видеокарты, однако транскодирование видео будет выполняться на процессоре, что может быть медленным при обработке множественных потоков или 4K контента.

4 Практическая реализация

4.1 Развертывание с Docker Compose

4.1.1 Docker Compose как инструмент развертывания

Все сервисы инфраструктуры определены в файлах конфигурации **docker-compose.yml**, которые описывают образы контейнеров, переменные окружения, сетевые подключения и тома хранения. Docker Compose позволяет развернуть всю систему одной командой без необходимости запускать каждый контейнер вручную.

Каждый функциональный блок (медиа-сервисы, мониторинг, инфраструктура) имеет собственный каталог в `/src/` с `docker-compose.yml` файлом. Разделение на отдельные файлы обеспечивает модульность и позволяет включать или отключать компоненты по необходимости.

4.1.2 Режимы хранения данных

Docker предоставляет два основных способа монтирования хранилища:

- **Named volumes** — управляемые Docker тома, хранящиеся в стандартной директории хоста (обычно `/var/lib/docker/volumes/`). Идеальны для конфигурационных файлов и баз данных, так как не требуют явного указания пути и автоматически резервируются.
- **Bind mounts** — прямое подключение директорий хоста к контейнерам. Используются для медиа-хранилища и пользовательских конфигураций, позволяя адаптировать пути под разные системы. Проект поддерживает оба режима: базовый вариант использует именованные тома, а расширенный — `docker-compose.bind.yml` для переопределения путей на пользовательское хранилище.

4.1.3 Структура docker-compose.yml

Конфигурация использует стандартный синтаксис YAML с определением сервисов, сетей и томов. Пример базовой конфигурации для прокси-слоя:

```yaml
services:
  nginx-proxy-manager:
    image: jc21/nginx-proxy-manager:latest
    container_name: nginx-proxy-manager
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
      - "81:81"
    environment:
      DISABLE_IPV6: 'true'
    volumes:
      - npm-data:/data
      - npm-letsencrypt:/etc/letsencrypt
    networks:
      - proxiable

volumes:
  npm-data:
  npm-letsencrypt:

networks:
  proxiable:
    external: true
```

Листинг 2 — Пример конфигурации Nginx Proxy Manager

Каждый сервис описывается блоком с параметрами:

- `image` — образ Docker из реестра (Docker Hub или другого)
- `container_name` — имя контейнера в системе
- `restart` — политика перезапуска (unless-stopped, always и т.д.)
- `ports` — проброс портов хоста на контейнер
- `environment` — переменные окружения, параметры конфигурации
- `volumes` — монтирование хранилища
- `networks` — подключение к виртуальным сетям Docker

4.1.4 Переменные окружения и .env файлы

Для обеспечения гибкости и безопасности, чувствительные данные (пароли, API ключи) и пути хранения определяются в файлах `.env`, которые загружаются автоматически. Пример структуры:

```
JELLYFIN_API_KEY=your_secret_key
HOST_APPDATA=/mnt/media/data
TRAEFIK_DASHBOARD_USER=admin
```

Листинг 3 — Пример переменных окружения в файле .env

Переменные в `docker-compose.yml` подставляются через синтаксис `${VARIABLE_NAME}`, что позволяет переносить конфигурацию между системами без изменения основного файла.

4.1.5 Порядок запуска контейнеров

Хотя Docker Compose может запустить все сервисы параллельно, рекомендуемый порядок обеспечивает корректную инициализацию зависимостей:

1. Создание необходимых Docker сетей (если используется `external: true`)
2. Запуск Nginx Proxy Manager (entry point для других сервисов)
3. Запуск DNS сервера (AdGuardHome) для локального разрешения имён
4. Запуск VPN контейнера (WireGuard) для приватных скачиваний
5. Запуск остальных сервисов в произвольном порядке

Команда для запуска всех контейнеров в директории:

```
docker compose up -d
```

Для запуска конкретного стека с переопределением путей:

```
HOST_APPDATA=/mnt/custom/path docker compose -f docker-compose.yml -f
docker-compose.bind.yml up -d
```

4.2 Конфигурация Nginx Proxy Manager

4.2.1 Инициализация и доступ к админ-панели

После запуска контейнера Nginx Proxy Manager необходимо войти в администраторскую панель для конфигурации. По умолчанию панель доступна по адресу `http://localhost:81` с учётными данными по умолчанию (admin@example.com / changeme). После первого входа рекомендуется изменить пароль администратора через раздел Settings.

4.2.2 Создание SSL сертификата

Для безопасного HTTPS соединения требуется SSL сертификат. Nginx Proxy Manager поддерживает интеграцию с Let's Encrypt для автоматического получения и обновления сертификатов.

Перейдите на вкладку **Certificates** в верхней навигации:

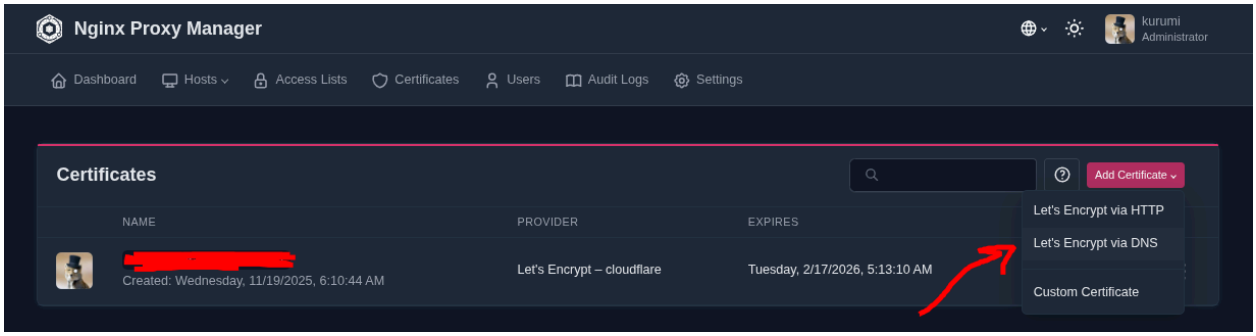

Рисунок 2 — Вкладка Certificates в Nginx Proxy Manager

Let's Encrypt предоставляет два метода подтверждения владения доменом:

- **HTTP метод** — требует открытия портов 80 на интернета и доступности домена с интернета. Подходит для публичных доменов, но требует явного port forwarding на маршрутизаторе.
- **DNS метод** — подтверждение через создание DNS записей у вашего DNS провайдера. Не требует открытия портов и более безопасен для приватных/локальных доменов. Использует API вашего DNS провайдера (Cloudflare, Route53, и т.д.) для автоматического создания временных записей.

Для локальной инфраструктуры рекомендуется **DNS метод**.

Нажмите кнопку «Add Certificate» и выберите «Let’s Encrypt via DNS»:

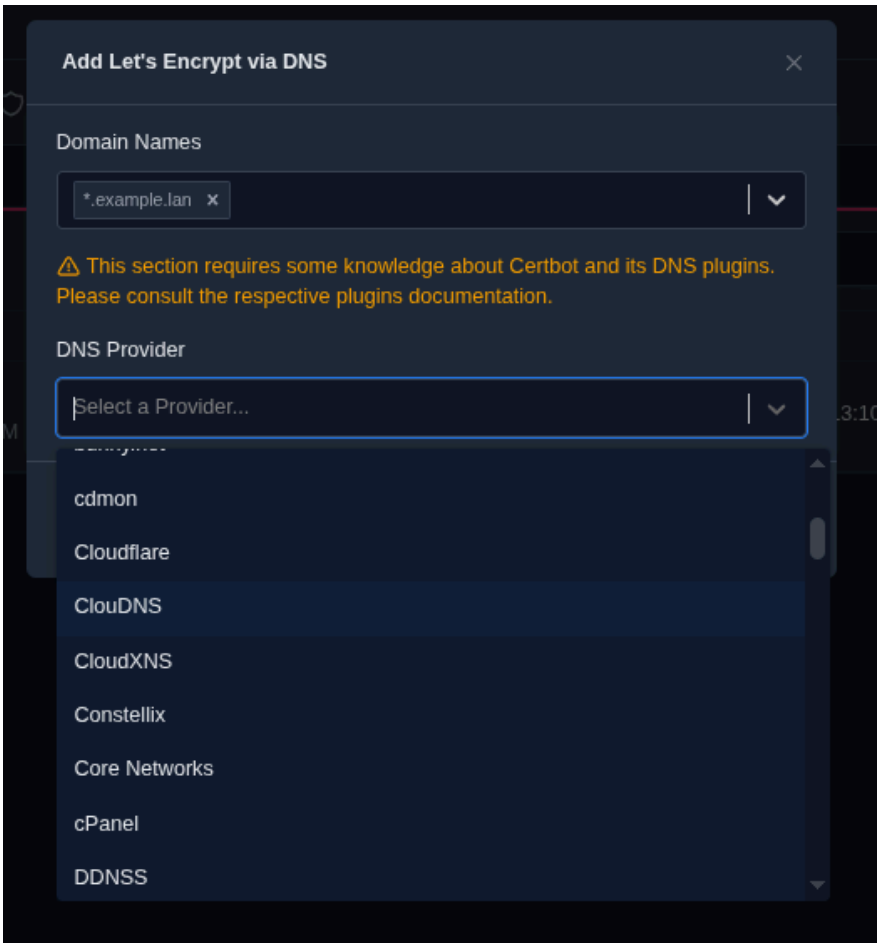

Рисунок 3 — Диалог добавления Let’s Encrypt сертификата через DNS

В поле **Domain Names** введите wildcard домен для вашей локальной сети, например `*.example.lan`. Это позволит использовать один сертификат для всех поддоменов (sonarr.example.lan, radarr.example.lan и т.д.).

В выпадающем списке **DNS Provider** выберите вашего провайдера DNS (Cloudflare, Quad9, и т.д.) и введите необходимые учётные данные — API токен, логин/пароль или другую информацию, требуемую провайдером. Nginx Proxy Manager автоматически создаст DNS записи для подтверждения владения доменом и получит сертификат.

После успешного создания сертификат появится в списке и будет автоматически обновляться за 30 дней до истечения.

4.2.3 Добавление хостов для прокси-маршрутизации

После создания SSL сертификата необходимо добавить хосты — правила маршрутизации трафика к контейнерам. Перейдите на вкладку **Hosts** и нажмём кнопку «Add Proxy Host»:

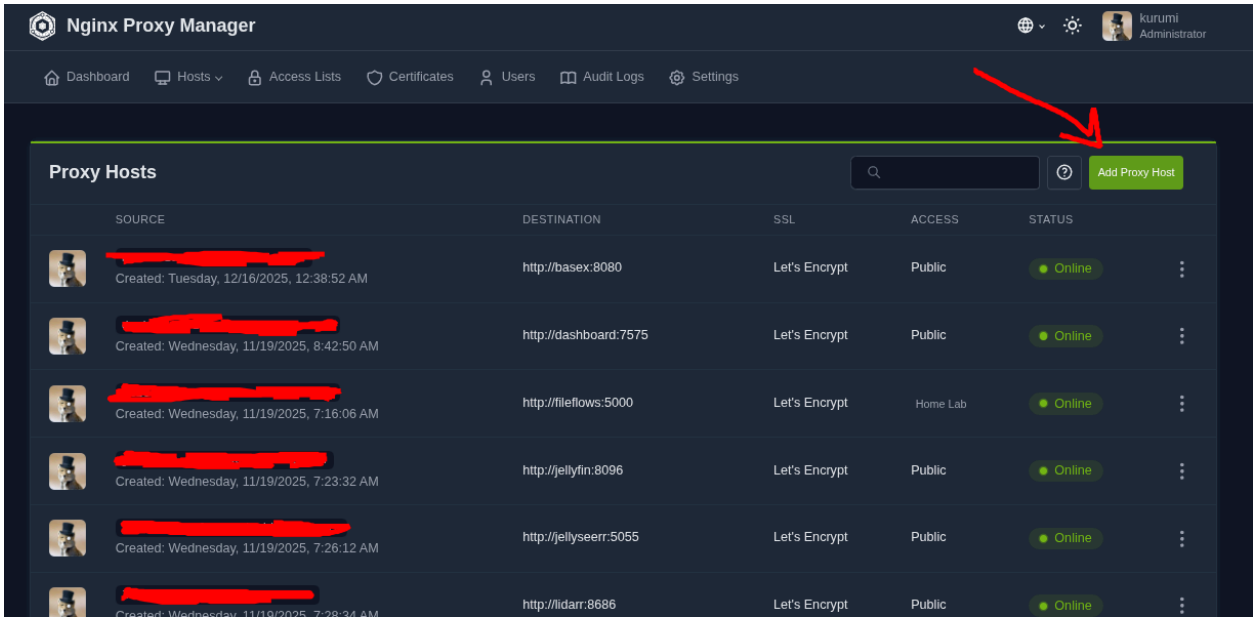

Рисунок 4 — Вкладка Hosts для управления прокси-хостами

Откроется диалог добавления нового Proxy Host с несколькими вкладками. Начнём с вкладки **Details**:

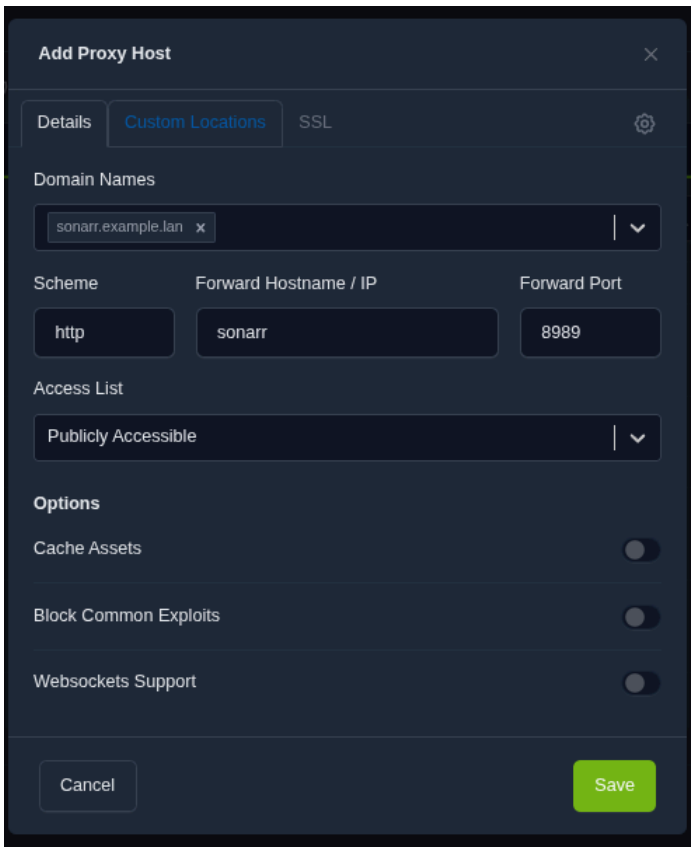

Рисунок 5 — Диалог добавления Proxy Host с параметрами маршрутизации

- **Domain Names** — доменное имя, соответствующее вашему wildcard сертификату, например `sonarr.example.lan` для сервиса Sonarr.
- **Forward Hostname / IP** — имя сервиса в Docker сети. **Важно**: это НЕ `container_name`, а лейбл сервиса из `docker-compose.yml` (строка `sonarr:` в блоке `services:`). Docker DNS автоматически разрешит это имя в IP контейнера внутри сети.
- **Forward Port** — порт, на котором слушает сервис согласно конфигурации (например, 8989 для Sonarr, 8096 для Jellyfin).
- **Scheme** — протокол (HTTP или HTTPS) для взаимодействия с контейнером. Обычно используется HTTP, так как контейнеры общаются внутри локальной Docker сети без необходимости в HTTPS.
- **Cache Assets** — отключите для динамического контента (Sonarr, Radarr, API сервисы). Включение этой опции приведёт к кешированию ответов и неправильному отображению обновляющихся данных.
- **Block Common Exploits** — оставьте включённым для защиты от распространённых векторов атак.

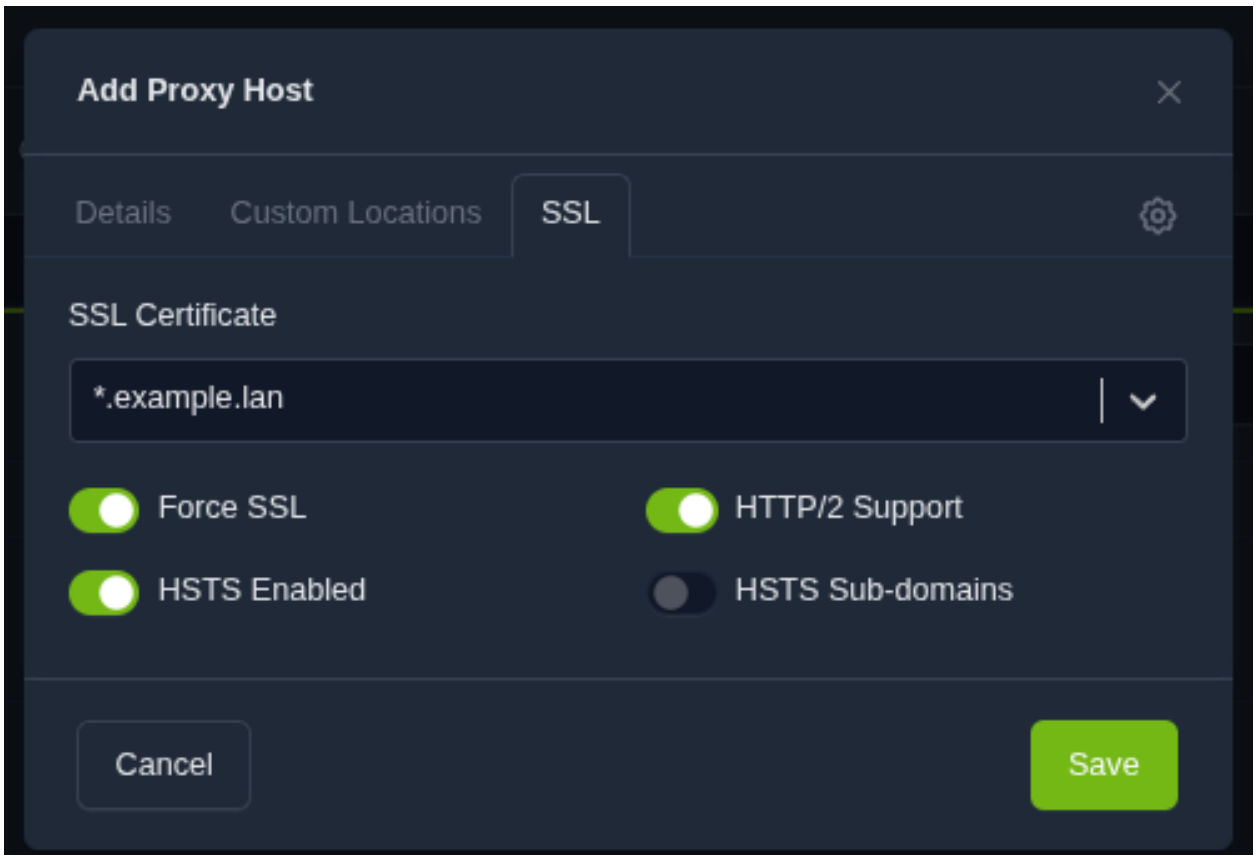

Рисунок 6 — Вкладка SSL для настройки сертификата и безопасности

- **SSL Certificate** — выпадающий список с доступными сертификатами. Выбираем созданный wildcard сертификат.
- **Force SSL** — опиция включения принудительного перенаправления HTTP трафика на HTTPS.
- **HSTS Enabled** — опция добавления включения HTTP Strict Transport Security.
- **HTTP/2 Support** — поддержка современного протокола HTTP/2.

После сохранения первого хоста (например, для Sonarr) по аналогии создаём все остальные сервисы. Для каждого сервиса необходимо создать отдельный хост с соответствующим доменным именем, лейблом контейнера и портом согласно таблице 2.

4.3 Конфигурация Prowlarr

4.3.1 Инициализация и методы доступа

Prowlarr — агрегатор индексеров торрентов, который управляет подключением к торрент-сайтам и предоставляет единый интерфейс для

поиска. После запуска контейнера Prowlarr доступен по адресу `https://prowlarr.example.lan` (через Nginx Proxy Manager).

При первом входе система предложит выбрать метод защиты:

- **Аутентификация по паролю** — рекомендуется для приватных сетей. Установите надёжный пароль и используйте его при каждом входе.
- **Веб-форма** — более простой вариант, но менее безопасный. Используйте только в изолированных локальных сетях.

Выберите один из методов и продолжите инициализацию.

4.3.2 Добавление индексаторов

После инициализации перейдите на главную страницу Prowlarr:

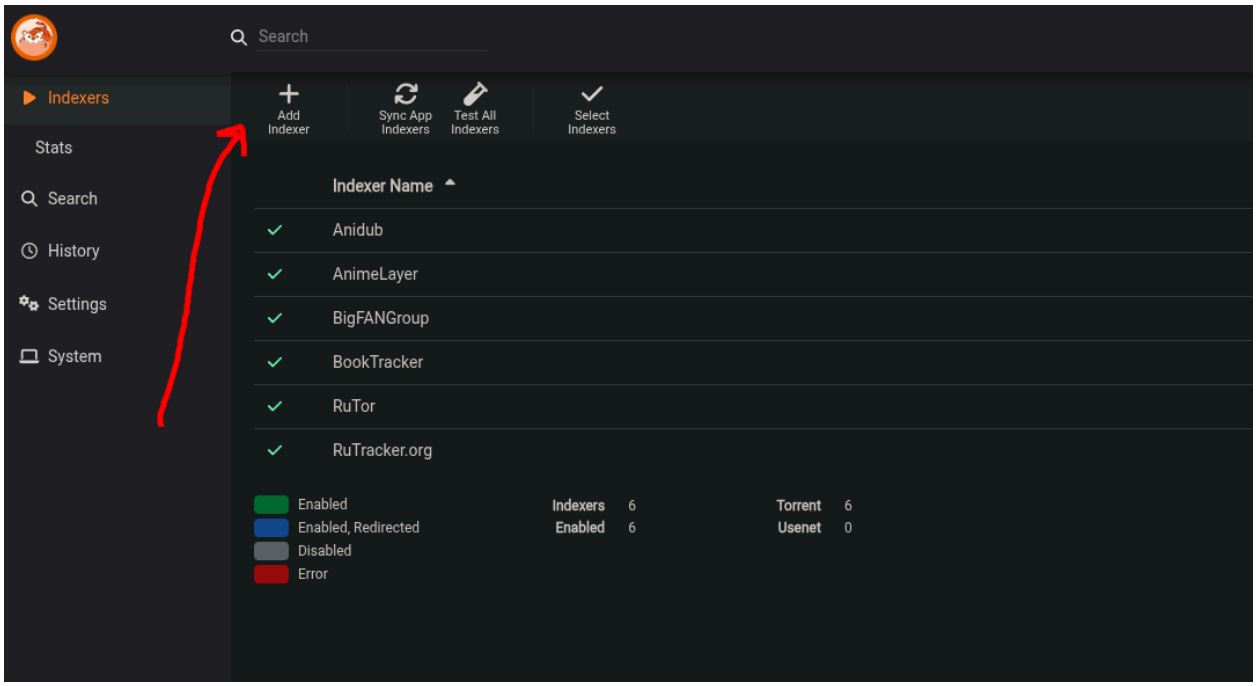

Рисунок 7 — Главная страница Prowlarr с кнопкой добавления индексера

Нажмите кнопку «Add New Indexer» (или «+» в левой панели). Откроется список всех доступных индексаторов:

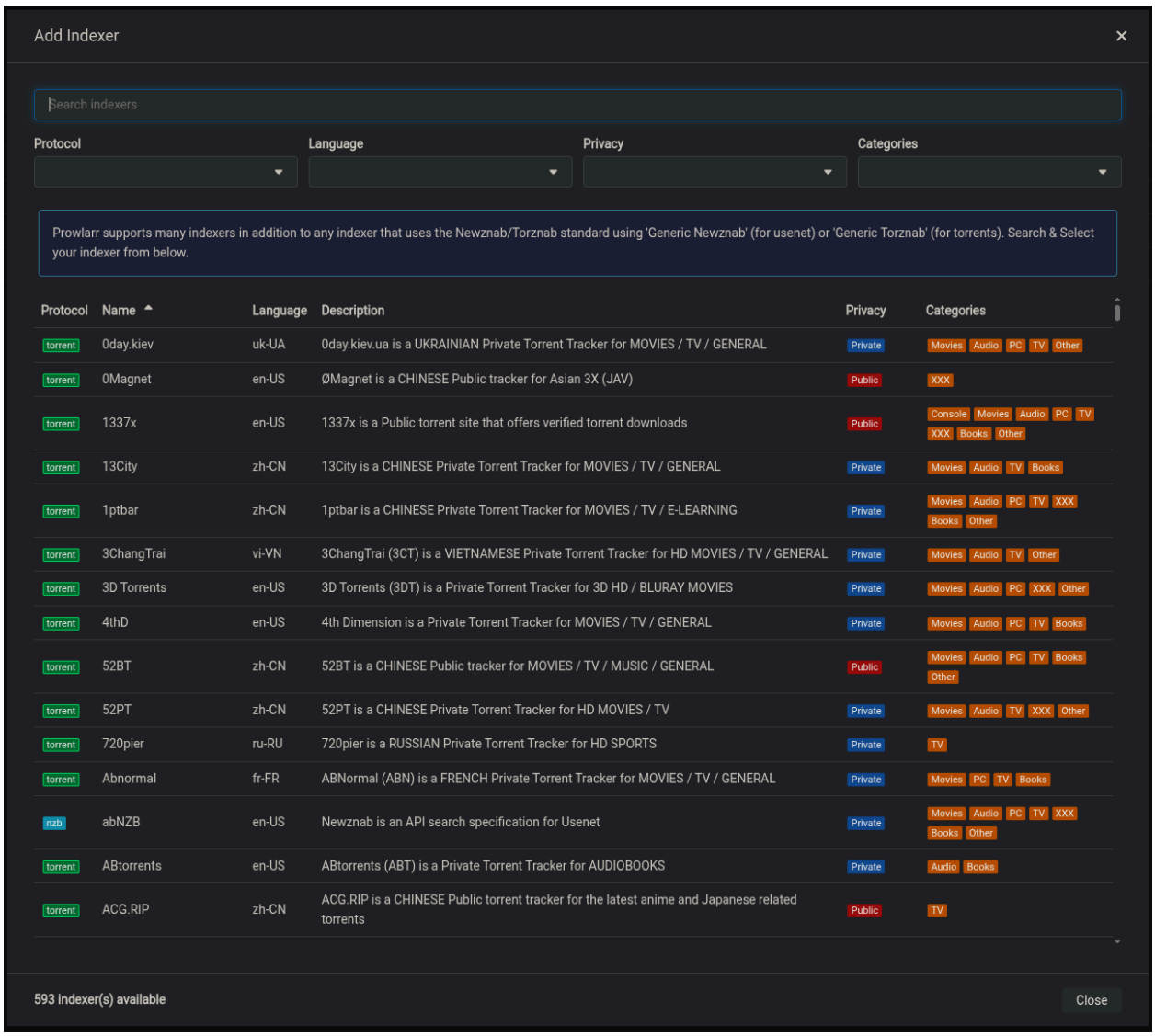

Рисунок 8 — Список доступных индексаторов для добавления

Список содержит более сотни индексаторов, поддерживаемых и обновляемых сообществом Prowlarr. Каждый индексатор может быть:

- **Публичным** — доступен без регистрации. Просто выберите и создайте индексатор.
- **Semi-private** — доступны с бесплатной регистрацией или с минимальными ограничениями. Требуют заполнения логина и пароля, но могут быть созданы любым пользователем без приглашения.
- **Приватным** — требует учётные данные (логин, пароль, API ключ или токен). После выбора приватного индексатора вам будет предложено заполнить поле для авторизации (обычно это приватный URL с токеном или API ключом). Доступ ограничен зарегистрированными пользователями или членами сообщества.

Общий процесс:

1. Найдите нужный индексатор в списке (используйте поиск)
2. Нажмите на него для открытия конфигурации
3. Если это приватный индексатор, введите учётные данные авторизации
4. Нажмите «Add Indexer» для добавления

После добавления индексатора он становится доступным для поиска из Sonarr, Radarr и Lidarr через встроенный API Prowlarr. Все поиски и запросы будут автоматически распределяться между добавленными индексаторами.

4.3.3 Добавление клиента загрузок

Prowlarr должен знать, на какой клиент отправлять найденные торренты для загрузки. Для этого необходимо добавить qBittorrent в качестве download client.

Перейдите в раздел **Settings → Download Clients**:

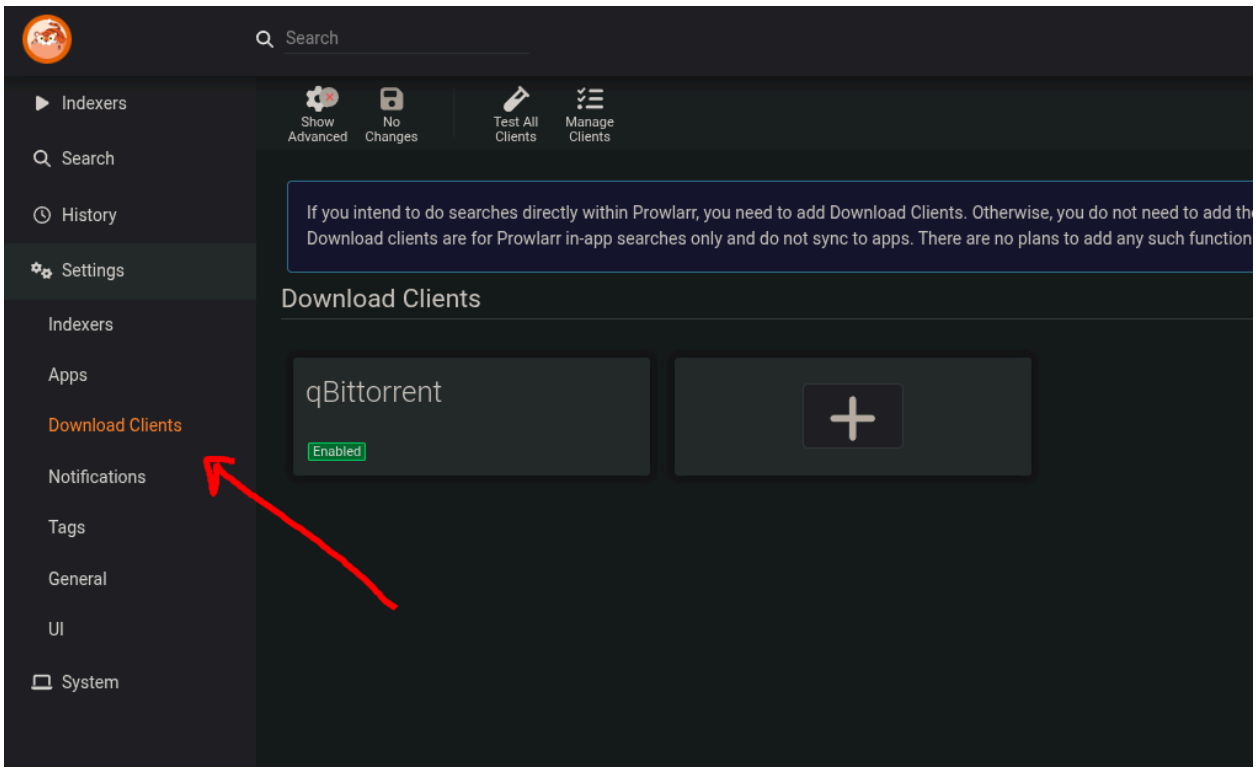

Рисунок 9 — Раздел Download Clients в настройках Prowlarr

Нажмите кнопку «+» для добавления нового клиента и выберите **qBittorrent** из списка:

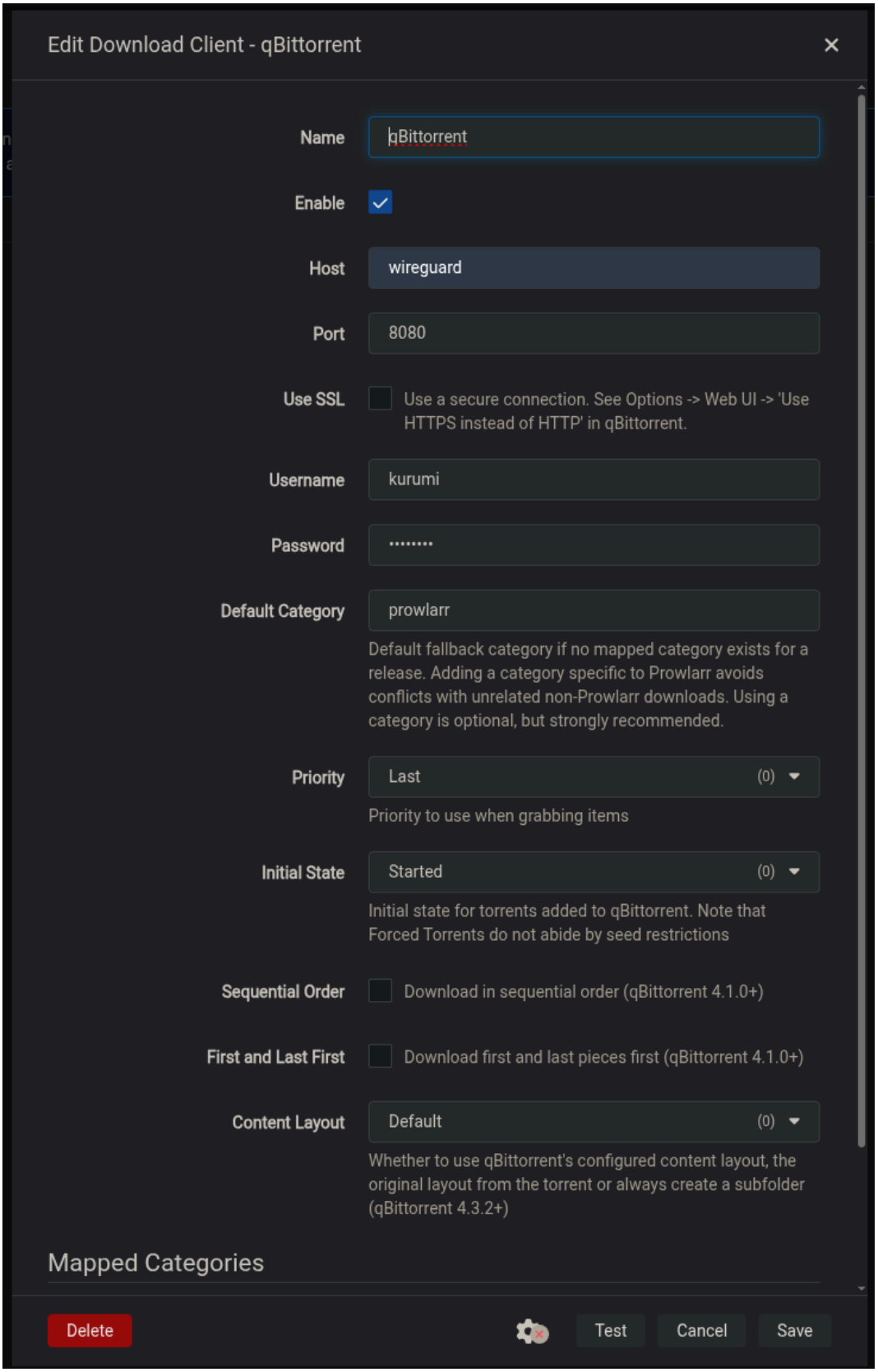

Рисунок 10 — Конфигурация qBittorrent как download client

Ключевой момент: так как qBittorrent запущен в виртуальной сети WireGuard (VPN) для приватности скачиваний, в поле **Host** необходимо указать лейбл контейнера VPN сервера — это `wireguard`. Docker DNS автоматически разрешит это имя в IP адрес контейнера WireGuard, через который будет маршрутизироваться весь трафик qBittorrent.

Остальные параметры (Port, Username, Password) остаются со значениями по умолчанию или заполняются в зависимости от конкретной конфигурации qBittorrent в docker-compose. После сохранения Prowlarr сможет отправлять найденные торренты непосредственно в qBittorrent для загрузки.

4.4 Конфигурация Sonarr, Radarr и Lidarr

4.4.1 Инициализация и download client

Sonarr (для сериалов), Radarr (для фильмов) и Lidarr (для музыки) — это так называемый **servarr стек** приложений для автоматизации управления медиа-контентом. Каждое приложение — независимая единица, которая может работать без других компонентов.

При первом входе в каждое приложение выберите метод аутентификации (пароль или веб-форма), как и для Prowlarr.

Затем необходимо добавить download client. Хотя Prowlarr тоже имеет download client, **каждое приложение servarr стека должно иметь свой собственный download client**. Это обусловлено архитектурой: Sonarr, Radarr и Lidarr — независимые приложения, которые могут существовать без Prowlarr (например, пользователь может вручную добавлять магнет-ссылки или использовать другие источники).

Процесс добавления download client в каждое приложение идентичен процессу для Prowlarr: перейдите в **Settings → Download Clients**, нажмите «+», выберите qBittorrent и заполните хост как `wireguard` (лейбл контейнера VPN).

4.4.2 Добавление Prowlarr как индексатора

Ключевое преимущество архитектуры — **Prowlarr выступает в качестве адаптера для множества индексаторов**. Вместо того, чтобы конфигурировать индексаторы в каждом приложении Sonarr/Radarr/Lidarr отдельно, вы добавляете Prowlarr один раз, и все приложения получают доступ к унифицированному списку индексаторов.

Для этого необходимо установить интеграцию между Prowlarr и каждым приложением.

В каждом приложении (Sonarr/Radarr/Lidarr) перейдите в раздел **Settings → General** и скопируйте API ключ:

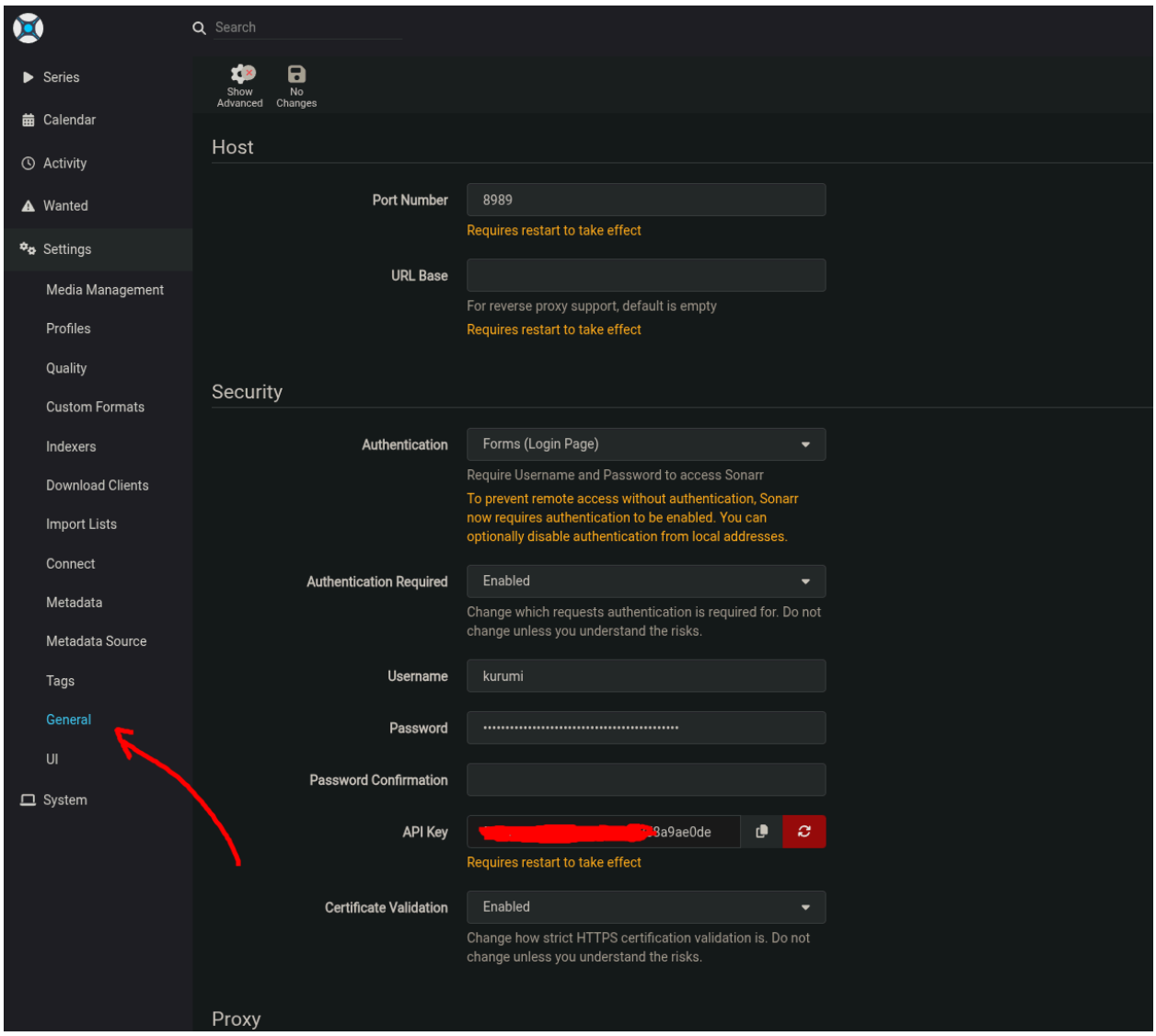

Рисунок 11 — Копирование API ключа приложения Sonarr

Перейдите в Prowlarr → **Settings** → **Apps** и добавьте новое приложение:

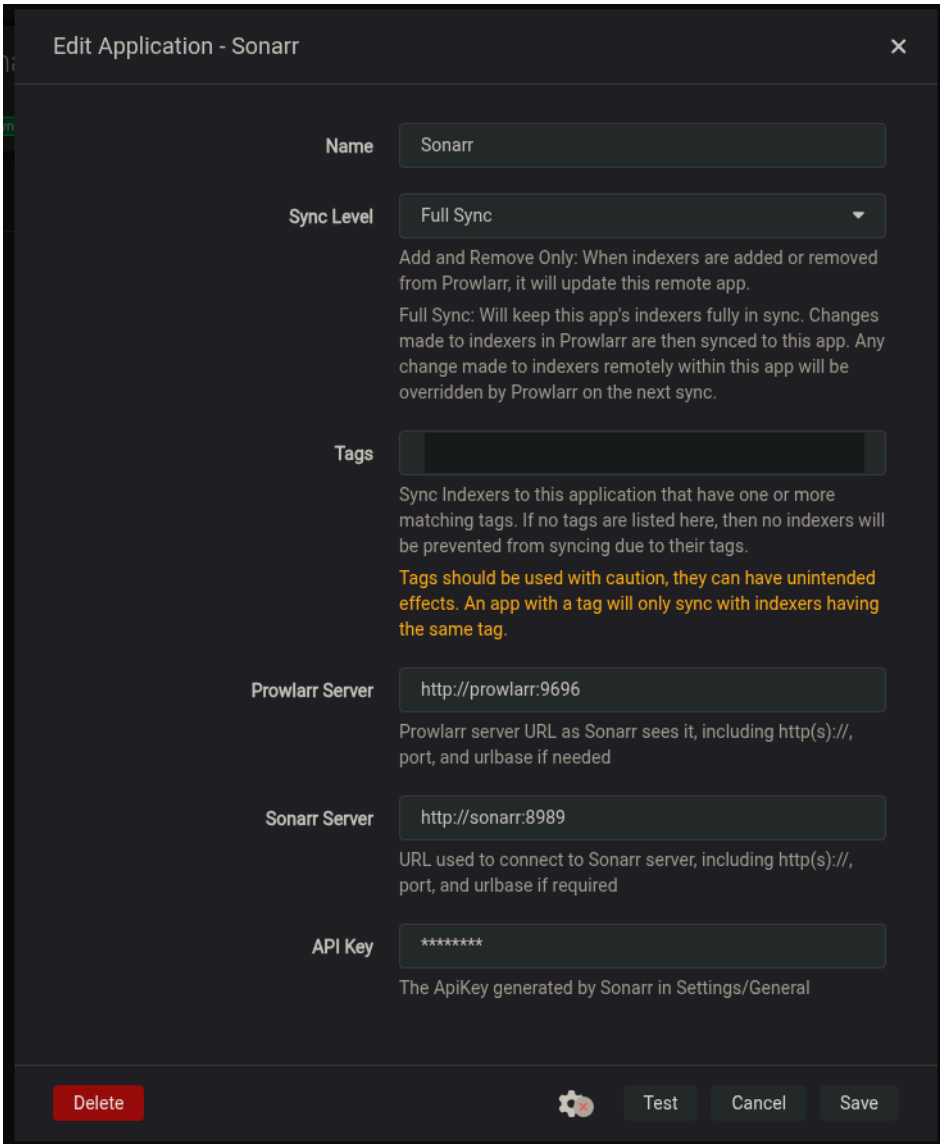

Рисунок 12 — Добавление Sonarr в качестве приложения в Prowlarr

Заполните следующие параметры:

- **Name** — название приложения (например, «Sonarr», «Radarr», «Lidarr»)
- **Prowlarr Host** — адрес Prowlarr, доступный для приложения. Используйте внутренний адрес Docker сети: `http://prowlarr:9696`
- **Application Host** — адрес самого приложения. Используйте его лейбл из docker-compose: `http://sonarr:8989` для Sonarr, `http://radarr:7878` для Radarr, `http://lidarr:8686` для Lidarr
- **API Key** — вставьте API ключ, скопированный на предыдущем шаге

После сохранения Prowlarr установит синхронизацию с приложением и добавит в него список доступных индексаторов. Все поиски, выполняемые из Sonarr/Radarr/Lidarr, будут автоматически распределяться между индексаторами, добавленными в Prowlarr.

4.5 Конфигурация папок и путей медиа-контента

4.5.1 Определение структуры хранилища в servarr приложениях

Для корректной организации медиа-контента каждое приложение servarr стека (Sonarr, Radarr, Lidarr) должно знать, где хранятся загруженные файлы. Это достигается через конфигурацию «Root Folders» — корневых директорий, в которых будут размещаться загруженные сериалы, фильмы или музыка.

Перейдите в **Settings → Media Management** в любом из приложений (например, в Sonarr):

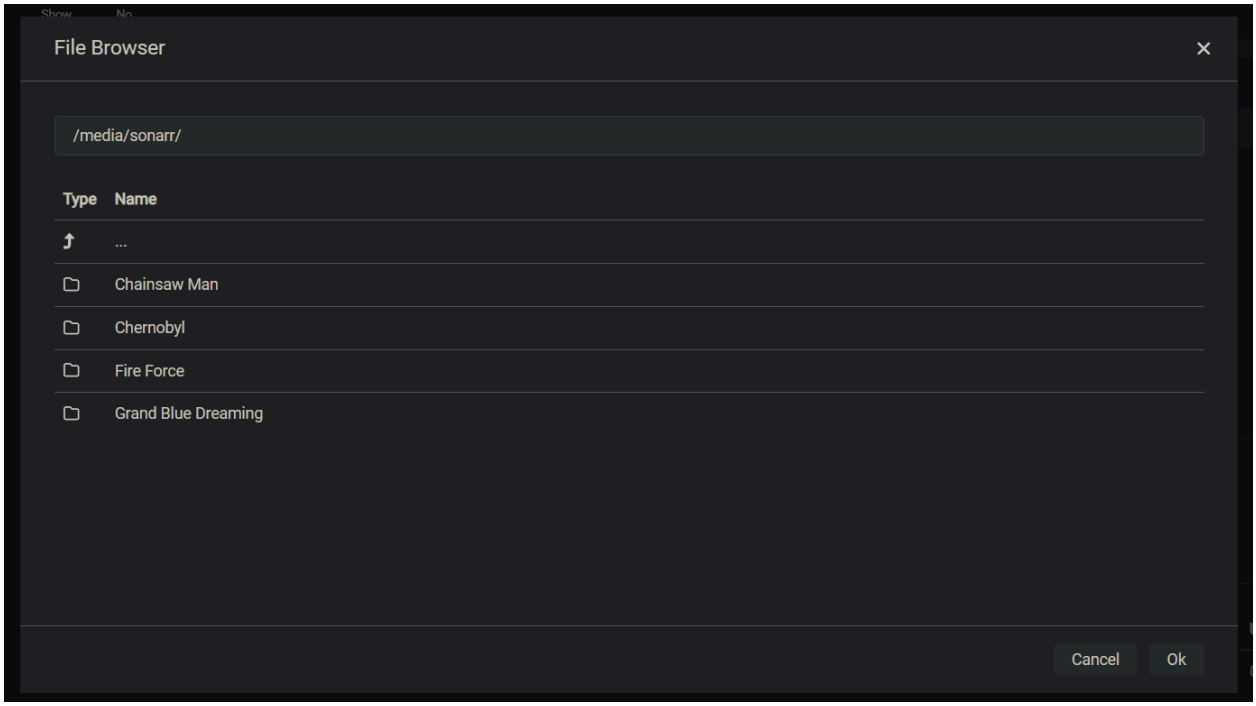

Рисунок 13 — Раздел Media Management для конфигурации папок в Sonarr

В списке **Root Folders** нажмите кнопку «Add Root Folder» и укажите путь, где будут храниться загруженные сериалы:

– **Путь** — директория на хосте, монтированная в контейнер. Например, если в `docker-compose.yml` определена строка `- /mnt/`

`media/series:/media/sonarr`, то в Root Folder следует указать `/media/sonarr` (путь **внутри контейнера**). Docker автоматически проксирует этот путь на хостовую директорию `/mnt/media/series`.

Таким же образом конфигурируются:

- **Sonarr** → Root Folder для сериалов (обычно `/media/sonarr` или `/media/series`)
- **Radarr** → Root Folder для фильмов (обычно `/media/radarr` или `/media/movies`)
- **Lidarr** → Root Folder для музыки (обычно `/media/lidarr` или `/media/lidarr`)

Структура хранилища может выглядеть следующим образом:

```
/mnt/media/
├── sonarr/              # монтируется в Sonarr как /media/sonarr
├── radarr/              # монтируется в Radarr как /media/radarr
├── lidarr/              # монтируется в Lidarr как /media/lidarr
├── prowlarr/            # конфигурация и кеш Prowlarr
└── downloads/           # папка загрузок торрентов
    ├── sonarr/          # загрузки для сериалов
    ├── radarr/          # загрузки для фильмов
    ├── lidarr/          # загрузки для музыки
    └── prowlarr/        # загрузки для индексеров (если нужны)
```

Критически важная деталь: qBittorrent настраивается отправлять загруженные торренты в категории, соответствующие подпапкам `downloads/sonarr/`, `downloads/radarr/` и `downloads/lidarr/`. Впоследствии Sonarr, Radarr и Lidarr перемещают файлы из папок загрузок в соответствующие финальные директории (`/media/sonarr/`, `/media/radarr/`, `/media/lidarr/`).

Оптимизация с помощью move strategy: поскольку все директории (загрузки и финальные папки) монтируются из одной физической файловой системы (`/mnt/media/`), при перемещении файлов используется стратегия `move` вместо `copy + delete`. Это означает, что файловая система просто изменяет ссылку (inode) на файл вместо его физического копирования и последующего удаления, что экономит время и дисковое пространство. Если бы загрузки монтировались из другого хранилища (например, `/mnt/`

`downloads/` на отдельном диске), то использовалась бы более дорогостоящая операция копирования, что замедлило бы обработку больших файлов.

4.5.2 Добавление путей в Jellyfin

После конфигурирования Root Folders в приложениях servarr стека загруженный контент автоматически появляется в соответствующих директориях. Теперь необходимо сделать этот контент доступным для пользователей через Jellyfin — медиа-сервер потоковой передачи.

Перейдите в веб-интерфейс Jellyfin и войдите с администраторскими правами. Откройте раздел **Dashboard → Libraries**:

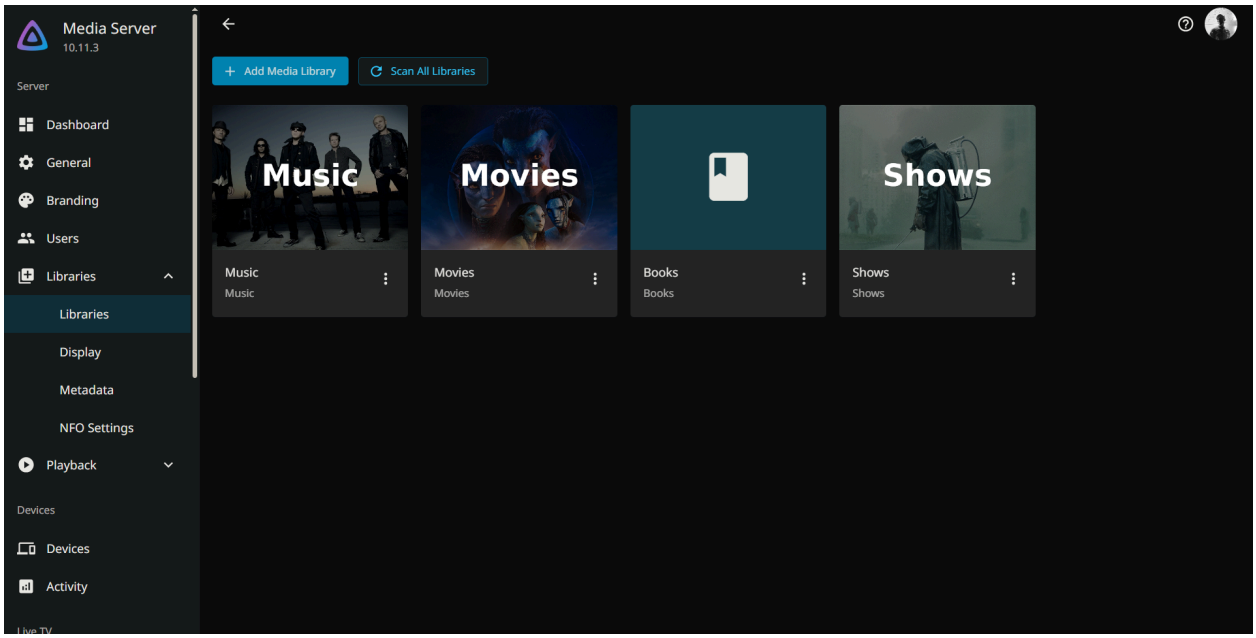

Рисунок 14 — Раздел Libraries для добавления медиа-библиотек в Jellyfin

Нажмите кнопку «Add Media Library» и выберите тип контента:

- **Movies** — для библиотеки фильмов
- **Shows** — для библиотеки сериалов
- **Music** — для библиотеки музыки

После выбора типа нажмите «Add Folder» и укажите путь к соответствующей директории:

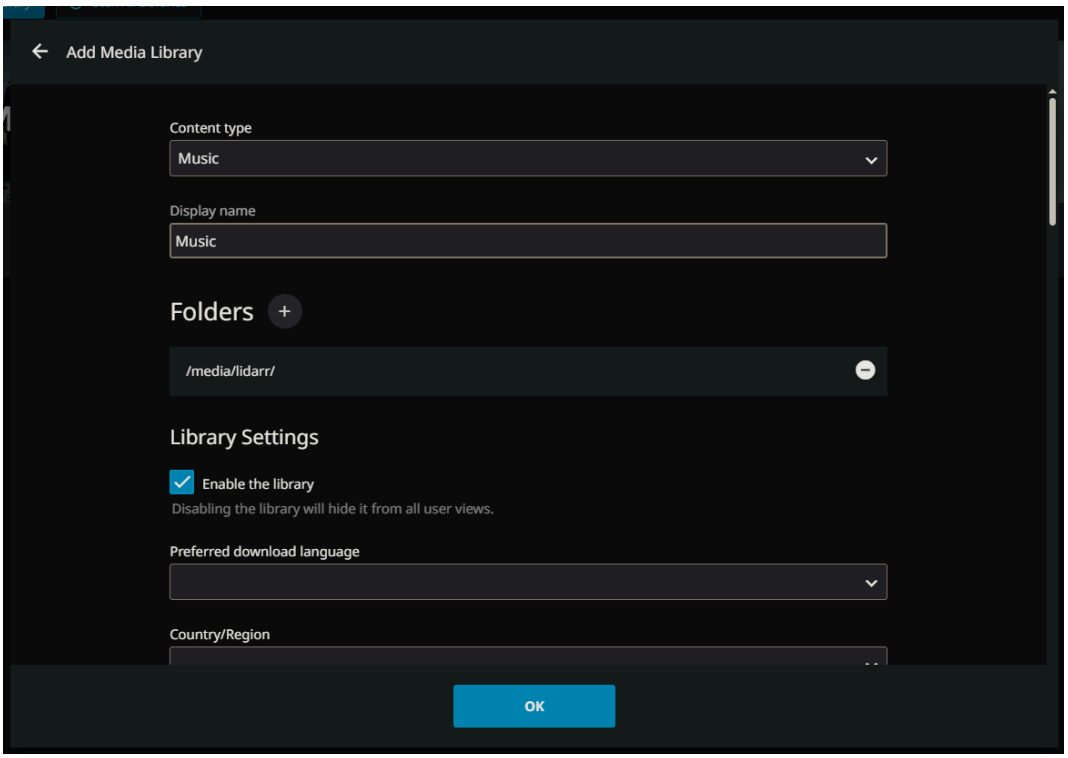

Рисунок 15 — Конфигурация папок для медиа-библиотеки в Jellyfin

Пути должны соответствовать монтированиям в `docker-compose.yml`:

- **Movies library** → добавить папку `/media/radarr` (соответствует монтированию Radarr)
- **Shows library** → добавить папку `/media/sonarr` (соответствует монтированию Sonarr)
- **Music library** → добавить папку `/media/lidarr` (соответствует монтированию Lidarr)

После сохранения конфигурации Jellyfin автоматически отсканирует указанные директории, распознает медиа-файлы, загрузит метаданные (плакаты, описания, рейтинги) и сделает контент доступным в пользовательском интерфейсе.

Помимо базовых путей, каждая библиотека имеет множество дополнительных параметров, влияющих на автоматизацию и точность метаданных:

- **Тип контента** — автоматическое определение типа медиа (фильм, сериал, музыкальный альбом). Jellyfin использует эту информацию для поиска правильного провайдера метаданных и применения соответствующих правил организации.

- **Язык контента** — предпочтительный язык для отображения названий и описаний. Выбор английского или русского языка влияет на качество поиска метаданных и локализацию информации.
- **Язык загрузки метаданных** — язык, на котором Jellyfin будет загружать плакаты, описания и другую информацию из онлайн-сервисов (IMDb, TMDb, MusicBrainz). Правильный выбор этого параметра обеспечивает точность и полноту информации о контенте.
- **Поставщик метаданных** — источники информации для заполнения метаданных (IMDb для фильмов, The Movie Database для сериалов, MusicBrainz для музыки и т.д.). Jellyfin использует несколько источников и объединяет информацию для максимальной полноты.
- **Параметры сканирования** — частота автоматического сканирования папки на предмет новых файлов, минимальный размер файла для признания его медиа-контентом, игнорирование определённых подпапок.

Все эти параметры работают совместно, обеспечивая не только корректное отображение контента, но и предпосылки для интеграции с автоматизацией. Например, правильная конфигурация типа контента и языка метаданных позволяет Sonarr/Radarr автоматически давать правильные названия загруженным файлам и папкам, что позволяет Jellyfin корректно их распознавать и каталогизировать.

4.5.3 Синхронизация между сервисами

Ключевой момент архитектуры — все сервисы используют **единое хранилище**. Когда Sonarr загружает новый сериал через qBittorrent:

1. qBittorrent скачивает файлы в директорию, указанную в категории торрента (например, `/media/downloads/sonarr`)
2. Sonarr перемещает (или копирует) файлы в Root Folder (`/media/sonarr`)
3. Jellyfin периодически сканирует `/media/sonarr` и обнаруживает новые файлы

4. Jellyfin загружает метаданные и добавляет сериал в библиотеку

Аналогичный процесс происходит для фильмов (через Radarr) и музыки (через Lidarr). Эта синхронизация позволяет пользователям видеть новый контент в Jellyfin буквально через несколько минут после его загрузки и обработки.

4.6 Конфигурация обработки контента в FileFlows

FileFlows — это сервис автоматической обработки медиа-файлов, который выполняет задачи, требующие вычислительных ресурсов: транскодирование видео, нормализацию аудио, обработку субтитров, удаление ненужных языковых дорожек и многое другое. В отличие от Sonarr/Radarr/Lidarr, которые управляют загрузкой и организацией контента, FileFlows занимается его трансформацией и оптимизацией.

Основные функции FileFlows:

- **Транскодирование видео** — конвертация видео из одного кодека в другой (например, из AV1 в H.264 для совместимости), изменение разрешения или битрейта.
- **Оптимизация для потоковой передачи** — адаптация видео для быстрой потоковой передачи на различные устройства.
- **Обработка аудио** — нормализация громкости, удаление ненужных аудиодорожек или субтитров на основе языковых предпочтений.
- **Удаление незначительных файлов** — автоматическое удаление образцов видео, sample файлов, папок с нежелательным контентом.
- **Организация файлов** — переименование и перемещение файлов согласно определённым правилам.

4.6.1 Создание флоу с помощью полу-автоматического инсталлятора

Для создания типового флоу обработки контента FileFlows предоставляет удобный мастер конфигурации. После входа в веб-интерфейс FileFlows перейдите в раздел **Flows** и нажмите кнопку создания нового флоу.

Система предложит вам ряд пошаговых вопросов:

1. **Тип контента** — выберите, какой тип медиа будет обрабатываться (фильм, сериал, музыка и т.д.). Это определяет набор доступных операций обработки.
2. **Исходный кодек видео** — укажите, из какого кодека нужна транскодировка (например, AV1, HEVC, H.264). Если транскодирование не требуется, выберите опцию пропуска.
3. **Целевой кодек видео** — выберите, в какой кодек транскодировать (обычно H.264 для максимальной совместимости с клиентами).
4. **Обработка аудиодорожек** — настройте, какие языки аудио нужно сохранить, какие удалить, нормализировать ли громкость.
5. **Обработка субтитров** — определите языки субтитров, которые должны оставаться в файле, и удалите ненужные.
6. **Загрузка метаданных** — выберите, нужно ли автоматически загружать и добавлять метаданные в файл (информацию о фильме, сезон, эпизод и т.д.).

На каждом шаге инсталлятор объясняет назначение параметра и предлагает рекомендуемые значения. После завершения всех шагов система автоматически создаст готовый флоу, который можно будет применять к медиа-файлам.

4.6.2 Добавление папки для сканирования и назначение флоу

После создания флоу необходимо определить, к каким файлам его применять. Это делается через добавление «папок библиотеки» (library folders) для сканирования.

Перейдите в раздел **Library** в левой навигации:

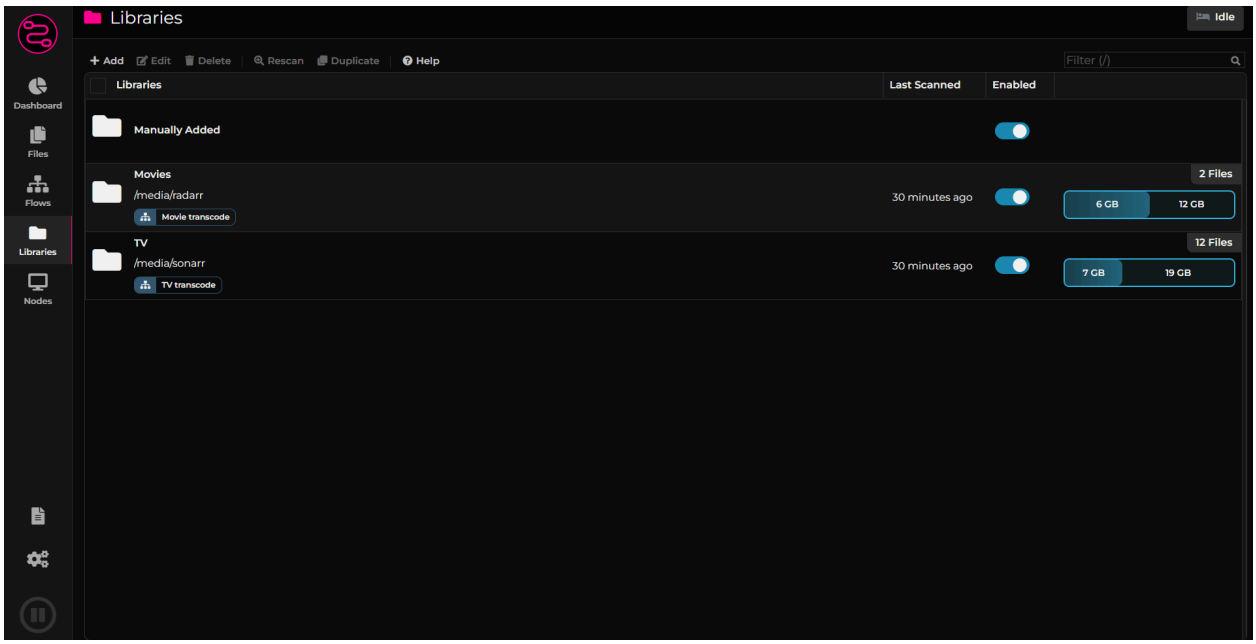

Рисунок 16 — Раздел Library в FileFlows для управления папками сканирования

Нажмите кнопку «Add Library» для создания новой папки сканирования. Откроется пошаговый диалог конфигурации:

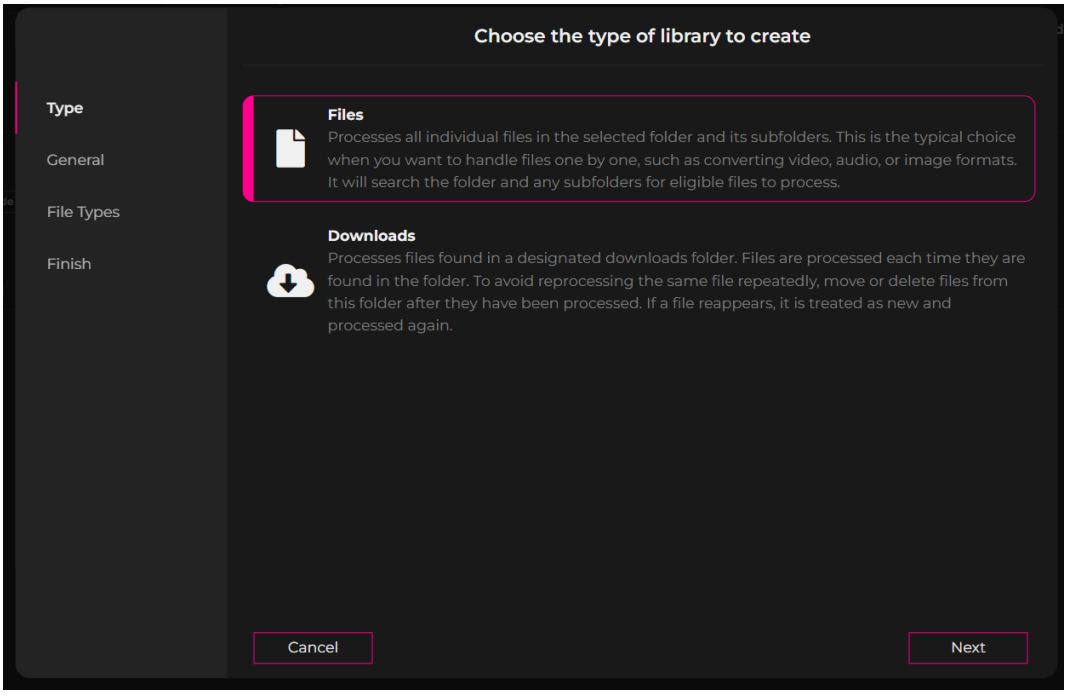

Рисунок 17 — Пошаговый диалог добавления новой папки библиотеки в FileFlows

Шаг 1: Тип библиотеки

На первом шаге выберите тип папки:

- **Files** — обычная папка с медиа-файлами, которые будут обработаны в заданном порядке
- **Downloads** — папка загрузок, из которой файлы будут перемещаться после обработки

Для большинства случаев используется тип **Files** для папок медиа (например, `/media/radarr`, `/media/sonarr`).

Шаг 2: Имя библиотеки

Введите название папки библиотеки (например, «Movies» для фильмов, «Series» для сериалов). Это внутреннее имя для идентификации в интерфейсе FileFlows.

Шаг 3: Путь к папке

Укажите полный путь к директории, которую FileFlows будет сканировать. Примеры:

- `/media/radarr` для фильмов
- `/media/sonarr` для сериалов
- `/media/lidarr` для музыки

Путь должен соответствовать монтированию в `docker-compose.yml` контейнера FileFlows.

Шаг 4: Назначение флоу

Выберите флоу, созданный на предыдущем этапе, которое будет применяться ко всем файлам в этой папке. Выпадающий список содержит все доступные флоу, поэтому убедитесь, что нужный флоу уже создан перед добавлением папки.

Шаг 5: Типы файлов

Укажите расширения файлов, которые нужно обрабатывать (например, `*.mkv;*.mp4;*.avi;*.mov`). FileFlows будет игнорировать все остальные файлы в папке, что позволяет избежать обработки неправильных или дополнительных файлов (например, `.nfo` файлов с метаданными или `.srt` файлов субтитров, если они не требуют обработки).

После завершения всех пяти шагов нажмите «Save» или «Finish» для сохранения конфигурации.

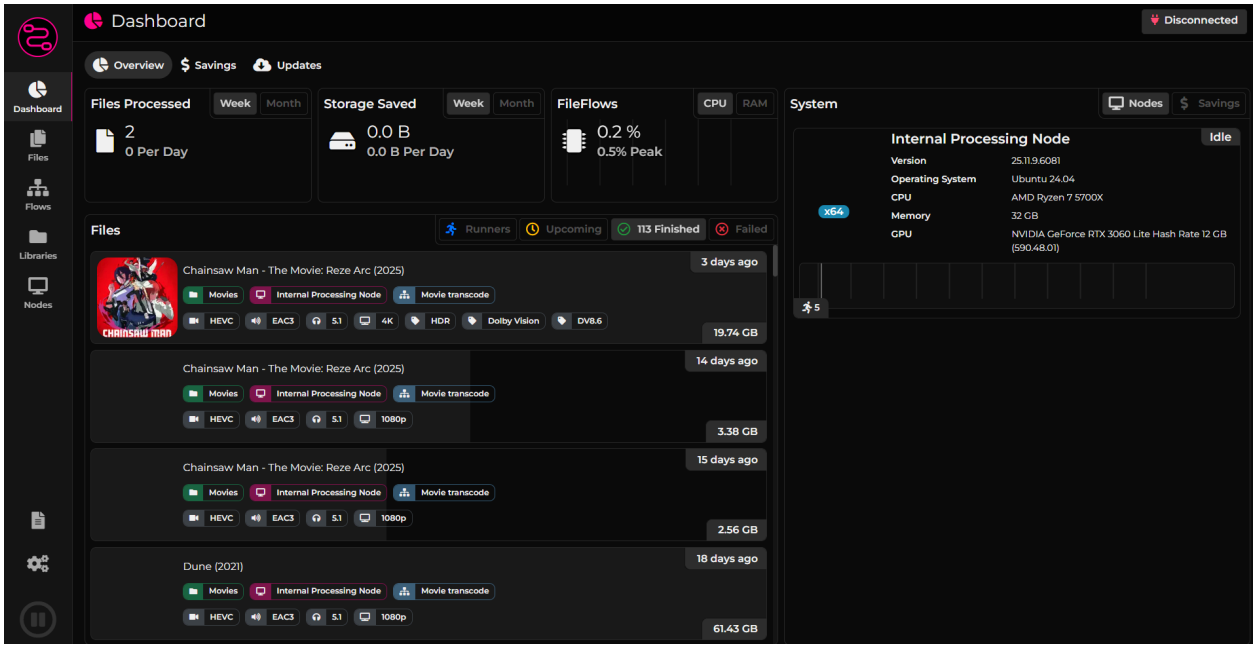

Рисунок 18 — Дашбоард FileFlows с добавленными библиотеками и активными флоу

После сохранения конфигурации FileFlows автоматически активирует флоу для выбранной папки. Система немедленно начнёт сканирование указанной директории и применять назначенный флоу ко всем существующим файлам, соответствующим критериям (расширение, размер и т.д.).

Процесс работает следующим образом:

1. **Периодическое сканирование** — FileFlows постоянно мониторит указанную папку на предмет новых или изменённых файлов согласно расписанию, установленному в конфигурации.
2. **Обнаружение новых файлов** — когда новый медиа-файл попадает в папку (например, когда Sonarr/Radarr загружает новый контент), FileFlows сразу же его обнаруживает.
3. **Применение флоу** — файл автоматически обрабатывается согласно созданному флоу: транскодируется видео, нормализуется аудио, удаляются ненужные дорожки и субтитры.

4. **Сохранение результата** — обработанный файл остаётся в той же папке с применёнными преобразованиями, заменяя исходный файл или сохраняясь рядом в зависимости от конфигурации.

Все обработанные файлы останутся в той же директории, но с оптимизированными параметрами видео, аудио и метаданными. Jellyfin затем сканирует уже обработанный контент и делает его доступным пользователям с улучшенной совместимостью и производительностью потоковой передачи.

Этот процесс полностью автоматизирован и интегрирован с остальной инфраструктурой: Sonarr/Radarr загружает контент → FileFlows обрабатывает → Jellyfin каталогизирует → пользователи получают оптимизированный контент.

5 Безопасность

Self-hosted инфраструктура предоставляет полный контроль над данными и системой, но возлагает на пользователя ответственность за её защиту. В отличие от облачных провайдеров, которые поддерживают централизованные команды безопасности, пользователь самостоятельно отвечает за все аспекты безопасности: от конфигурации сетей до обновления компонентов и мониторинга подозрительной активности.

Архитектура домашней лаборатории построена на нескольких уровнях защиты, которые работают совместно:

Первый уровень: Обратный прокси и единая точка входа. Все запросы к сервисам проходят через Nginx Proxy Manager, который служит единой точкой входа и фильтра для всего трафика. Это позволяет централизованно управлять доступом, применять правила безопасности и логировать все входящие запросы. Прокси также выполняет функцию WAF (Web Application Firewall), блокируя распространённые векторы атак.

Второй уровень: SSL/TLS шифрование. Все соединения с веб-интерфейсами защищены HTTPS с использованием сертификатов Let's Encrypt, автоматически обновляемых Nginx Proxy Manager. Это предотвращает перехват трафика и обеспечивает аутентификацию сервера. Внутри локальной сети между контейнерами используется незашифрованный HTTP, так как трафик не покидает сеть хоста.

Третий уровень: Сегментация Docker сетей. Инфраструктура разделена на несколько виртуальных сетей (media-network, monitoring, vpn, proxiable), что обеспечивает логическую изоляцию сервисов. Контейнеры в одной сети могут общаться напрямую, но контейнеры в разных сетях изолированы друг от друга. Это предотвращает распространение скомпрометированного контейнера на остальные сервисы и ограничивает повреждения в случае инцидента безопасности.

Четвёртый уровень: VPN для защищённого удалённого доступа. Если пользователь хочет получить доступ к домашней лаборатории извне локальной сети (например, с работы или во время путешествия), необходимо использовать VPN. Прямое экспонирование сервисов в интернет крайне опасно и не рекомендуется. WireGuard, используемый в инфраструктуре,

обеспечивает встроенное шифрование трафика на уровне VPN-протокола, что означает, что весь трафик между клиентом и домашней лабораторией автоматически зашифрован, даже если используется незащищённый HTTP на уровне приложения.

Пятый уровень: Автоматическое обновление компонентов. Watchtower постоянно мониторит образы контейнеров на предмет обновлений и автоматически перезапускает сервисы с новыми версиями. Это обеспечивает своевременное применение патчей безопасности без необходимости ручного вмешательства. Критические уязвимости часто исправляются в течение часов или дней, поэтому автоматизация этого процесса критична для безопасности.

Дополнительный уровень: Мониторинг и логирование. Prometheus и Grafana собирают метрики системы и контейнеров, что позволяет выявить аномальную активность (например, неожиданный скачок трафика, ошибки аутентификации или отказ в обслуживании). Логирование контейнеров позволяет отследить источник проблемы и провести анализ после инцидента.

Комбинация этих уровней обеспечивает глубокую защиту (defence in depth): даже если один уровень будет скомпрометирован, остальные уровни продолжат функционировать и предотвращать дальнейший ущерб. Ключевой момент: **никакая система не является полностью защищённой**. Пользователь должен постоянно мониторить инфраструктуру, держать компоненты в актуальном состоянии и быть готовым к реагированию на инциденты.

ЗАКЛЮЧЕНИЕ

Данная работа описала полный цикл проектирования и реализации персональной серверной инфраструктуры на базе Docker и Linux. От теоретических основ архитектуры до практических пошаговых инструкций конфигурирования — документация охватывает все ключевые аспекты создания self-hosted решения для управления медиа-контентом, обеспечения безопасного удалённого доступа и поддержания надёжной работы системы.

Основные компоненты, описанные в работе:

- **Архитектура инфраструктуры** с виртуальными Docker сетями, обеспечивающими сегментацию и изоляцию сервисов
- **Стек управления медиа** (Sonarr, Radarr, Lidarr, Prowlarr, Jellyseerr) для полной автоматизации скачивания и организации контента
- **Потоковая передача** (Jellyfin, Discord Music Bot) для доступа к контенту с любых устройств
- **Обработка контента** (FileFlows) с поддержкой GPU ускорения для эффективного транскодирования
- **Обратный прокси** (Nginx Proxy Manager) для единой точки входа и управления SSL/TLS
- **VPN** (WireGuard) для безопасного удалённого доступа с встроенным шифрованием
- **Безопасность** на нескольких уровнях: сегментация сетей, автоматическое обновление, шифрование трафика

Практическая ценность работы заключается в том, что читатель получает не просто теоретические знания, но и пошаговые инструкции для реальной реализации. Каждый компонент сопровождён примерами конфигурации, скриншотами интерфейсов и объяснением логики взаимодействия компонентов.

Объём и ограничения. Данная работа сосредоточена на ядре инфраструктуры и пропустила некоторые аспекты, осознанно отложенные для будущих расширений:

– **Логирование и аналитика** — централизованное сбор логов (ELK Stack или аналогичное решение) для анализа событий и поиска проблем
– **Масштабирование и распределение** — архитектуры для горизонтального масштабирования (Kubernetes, Swarm) при увеличении нагрузки
– **Детальный мониторинг метрик** — расширенная конфигурация Prometheus и Grafana с кастомными дашбордами и алертингом

Эти компоненты могут быть добавлены в инфраструктуру по мере необходимости и по мере углубления знаний пользователя.

Применение результатов. Описанная архитектура может быть использована как базис для:

– Личного медиа-сервера в домашних условиях
– Обучающего проекта для изучения Docker, сетевой безопасности и DevOps практик
– Основы для собственного расширения и кастомизации под конкретные требования
– Демонстрационного проекта для понимания complexity распределённых систем

Заключительные рекомендации. Для успешного внедрения инфраструктуры пользователю рекомендуется:

1. Начать с базовой конфигурации (Jellyfin + qBittorrent) и постепенно добавлять компоненты
2. Постоянно обновлять образы контейнеров (Watchtower помогает в этом автоматически)
3. Регулярно проверять логи и метрики для раннего выявления проблем
4. Поддерживать актуальную документацию своей конфигурации для быстрого восстановления после сбоев
5. Рассмотреть резервное копирование критичных данных (конфигурация, метаданные баз данных)

Инфраструктура, описанная в данной работе, демонстрирует, как современные инструменты (Docker, открытое ПО, Linux) позволяют любому пользователю создать надёжную, безопасную и расширяемую систему, которая обеспечивает полный контроль над личными данными и медиа-контентом без зависимости от облачных провайдеров.