Дата создания документа: 08.03.2026
Версия документа: v1.0

Описание технической архитектуры программного обеспечения

Платформа Линза Метрикс

Платформа автоматизированного мониторинга операционных процессов предприятий

linzametrics.com

Описание ПО: Программное обеспечение «Платформа Линза Метрикс» предназначено для автоматизированного мониторинга и управления операционными процессами предприятий, включая объекты розничной торговли, общественного питания и производственные площадки, с использованием технологий компьютерного зрения и искусственного интеллекта.

Правообладатель: Общество с ограниченной ответственностью «Линза Метрикс»

Регистрация: Свидетельство о государственной регистрации программы для ЭВМ № 2026615367 от 25.02.2026

Оглавление

1. Общие сведения

«Платформа Линза Метрикс» — программный комплекс для автоматизированного мониторинга и управления операционными процессами предприятий розничной торговли, общественного питания и иных объектов с наблюдаемыми производственными и сервисными процессами.

Система в режиме реального времени собирает данные о работе предприятия из различных источников — камер видеонаблюдения, датчиков, учётных систем, отчётов в мессенджерах — и анализирует ключевые операционные показатели: наличие товаров на полках, загрузку зон обслуживания, выполнение стандартов качества и операционных задач. При обнаружении ситуаций, требующих внимания, система автоматически уведомляет сотрудников и руководителей. Накопленные данные отображаются в аналитических панелях для выявления системных проблем и принятия решений на основе данных.

Основной реализованный источник данных — компьютерное зрение на базе IP-камер видеонаблюдения. Архитектура системы спроектирована для подключения произвольных источников через плагинную систему.

Система развёрнута как набор Docker-контейнеров, управляемых через Docker Compose, и может работать на центральном облачном сервере, на периферийных вычислительных устройствах (edge) или в гибридной конфигурации.

2. Архитектурный стиль

Система построена как **монолитный бэкенд с выделенным фронтендом и набором поставляемых артефактов**. Ядро системы — единый Django-процесс, совмещающий REST API, обработку видеопотоков (Celery Worker) и планировщик периодических задач (Celery Beat). Фронтенд (React SPA) — отдельный контейнер, взаимодействующий с бэкендом через REST API.

Разделение на контейнеры выполнено **по функциональным слоям** (бэкенд, фронтенд, СУБД, брокер), а не по доменным сервисам — это стандартная трёхзвенная архитектура, развёрнутая в Docker.

Состав продукта

Непосредственными продуктами ООО «Линза Метрикс» являются:

- **Бэкенд** (celery) — основной монолит: API, обработка данных из источников, бизнес-логика, уведомления.
- **Фронтенд** (markup-frontend) — интерфейс настройки объектов и разметки зон.
- **Шаблоны аналитических панелей** — JSON- и SQL-файлы для импорта в Grafana или иную BI-систему заказчика.
- **Нейросетевые модели** — обученные модели в формате ONNX/TensorRT, загружаемые в сервер вывода.

Среда выполнения (инфраструктура)

Для работы системы необходимы следующие компоненты окружения (стороннее ПО):

- **NVIDIA Triton Inference Server** — сервер нейросетевого вывода (выполняет поставляемые модели).
- **PostgreSQL** — реляционная СУБД.
- **Redis** — брокер сообщений для Celery.
- **Grafana** (или иная BI-система по выбору заказчика) — визуализация данных из БД. Не является частью ядра системы.
- **Nginx** (или иной обратный прокси) — HTTPS-терминация и маршрутизация.

Инструменты мониторинга системы

- **Prometheus** — сбор метрик с бэкенда.
- **Portainer** — управление Docker-контейнерами.
- **Sentry** — трекинг ошибок.

Инструменты разработки, CI/CD и внутренних пайплайнов

- **GitLab** (self-hosted) — репозиторий исходного кода, CI/CD пайплайны сборки и доставки образов.
- **Yandex Container Registry** — реестр Docker-образов.
- **Dagster** — оркестрация внутренних пайплайнов (сбор данных, подготовка моделей и др.).

Обработка данных

Многопоточный конвейер обработки видео. Для каждой активной камеры создаётся отдельный поток захвата, который читает видеопоток и складывает кадры в общую очередь. Из этой очереди кадры поступают в единый конвейер обработки: предобработка, отправка на Triton для нейросетевого вывода, затем последовательный вызов всех подписанных плагинов. Плагины получают кадры синхронно — каждый обязан обработать кадр быстро и не блокировать конвейер.

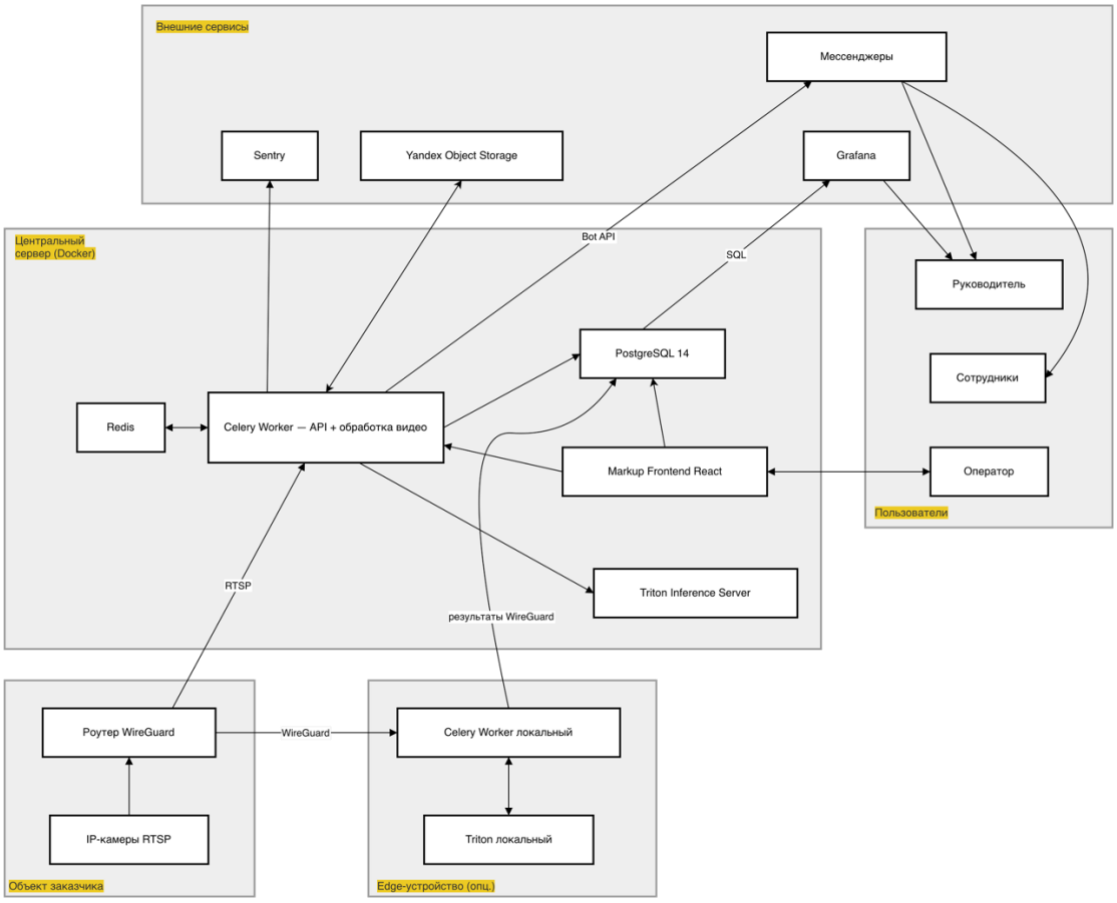
Асинхронная обработка побочных эффектов. Всё, что требует длительного ввода-вывода — отправка уведомлений, формирование отчётов, синхронизация чек-листов, сохранение медиа в S3 — выполняется через Celery-задачи (брокер — Redis). Периодические задачи управляются Celery Beat. Таким образом, основной конвейер обрабатывает кадры максимально быстро, а длительные операции не блокируют его.

Расширяемость

Бизнес-логика бэкенда организована через систему **плагинов и модулей** Django, загружаемых динамически из переменных окружения (PLUGINS, MODULES). Каждый плагин — отдельное Django-приложение с декларативным описанием зависимостей. Порядок выполнения определяется топологической сортировкой графа зависимостей. Это позволяет конфигурировать набор функций под конкретного заказчика без изменения ядра — например, для розницы подключаются плагины контроля выкладки и очередей, для охранного сценария — плагин детекции проникновений.

3. Структурные элементы системы

3.1. Общая архитектура (компонентная диаграмма)



3.2. Описание компонентов

Celery Worker (celery) — центральный компонент системы (монолит), реализованный на Python (Django 4.x + Django REST Framework + Celery). Совмещает в одном контейнере REST API (Django development server, порт 5022), многопоточный конвейер обработки данных из источников (видеопотоки, датчики, внешние системы), нейросетевой анализ и планировщик периодических задач Celery Beat. REST API обеспечивает CRUD-операции над объектами, камерами, зонами разметки и конфигурацией системы; поддерживается аутентификация через токены DRF, JWT и сессии. Документация API генерируется автоматически в формате Swagger/OpenAPI.

Triton Inference Server (triton-server) — сервер нейросетевого вывода от NVIDIA. Выполняет модели в форматах ONNX и TensorRT. При запуске контейнера модели загружаются из S3-хранилища (Yandex Object Storage) — локального хранилища моделей на сервере не требуется. В текущей конфигурации обслуживает несколько моделей одновременно: yolov5 (детекция людей и объектов), yolov5-freezer (детекция пустых зон на полках), effnetb1 и effnetb1_tables (классификационные модели). Взаимодействие с бэкендом — по HTTP (порт 8325) внутри Docker-сети через клиентскую библиотеку tritonclient.

Markup Frontend (markup-frontend) — одностраничное веб-приложение для настройки объектов наблюдения, управления камерами и интерактивной полигональной разметки зон на видеокадрах. Стек: React 18, TypeScript, Redux Toolkit (управление состоянием),

Material UI (интерфейс), Konva (2D-рисование полигонов на canvas). Приложение собирается в статику и раздаётся через встроенный Nginx (порт 3000). Всё взаимодействие с данными — через REST API бэкенда.

PostgreSQL 14 (db) — реляционная СУБД, хранящая конфигурацию системы (объекты, камеры, зоны разметки, настройки плагинов), результаты аналитики (метрики заполненности, события очередей, результаты детекции), данные чек-листов и журнал уведомлений. Данные изолированы через механизм PostgreSQL Schema, что позволяет размещать несколько независимых инсталляций в одной СУБД.

Redis — используется как брокер сообщений и бэкенд результатов для Celery, а также как кэш. Обеспечивает асинхронную передачу задач между API, worker и планировщиком.

Grafana — стороннее ПО с открытым исходным кодом, не входящее в состав поставки. Подключается к базе данных системы как внешний клиент с правами только на чтение и визуализирует накопленные данные. Линза Метрикс поставляет шаблоны дашбордов в формате JSON для импорта.

4. Система плагинов и модулей

Расширяемость системы реализована через два механизма: плагины и модули. Оба представляют собой Django-приложения, загружаемые динамически при старте на основании переменных окружения PLUGINS и MODULES (списки через запятую). При запуске система добавляет их в INSTALLED_APPS и регистрирует API-маршруты автоматически.

4.1. Плагины

Плагины реализуют специализированную бизнес-логику и наследуют один из базовых классов:

BasePlugin — базовый класс. Поддерживает декларацию зависимостей от других плагинов и модулей (depends_on_modules, depends_on_plugins). Зависимости разрешаются автоматически при старте через топологическую сортировку (алгоритм Кана) с обнаружением циклов.

PeopleProcessingPlugin — плагины, которые обрабатывают результаты детекции людей на кадре. Реализуют метод process_persons(), вызываемый конвейером обработки для каждого пакета кадров. Примеры: checkout_control (контроль очередей и касс), occupancy_engine (заполненность зон), empty_shelf (обнаружение пустых полок), security (детекция проникновений), working_zone_analyzer (присутствие персонала).

ProviderPlugin — плагины-провайдеры источников данных (камеры, датчики, внешние системы). Реализуют метод provide_camera_threads(), создающий потоки захвата данных. Реализованные примеры: trassir (интеграция с Trassir VMS), hik_snapshot_provider (снимки через Hikvision ISAPI).

4.2. Модули

Модули — лёгковесные Django-приложения, не требующие реализации специального интерфейса времени исполнения. Предоставляют модели данных, утилиты и интеграции, используемые плагинами. Основные модули: people_core (базовые модели детекции),

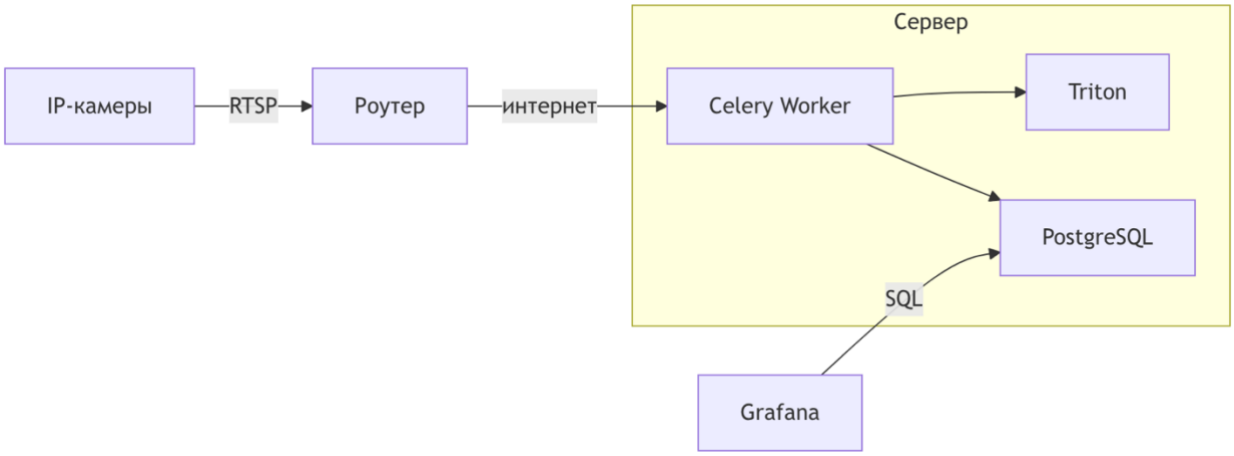
occupancy_core (расчёт заполненности), telegram_messages (интеграция с Telegram Bot API), google_apis (синхронизация чек-листов из Google Sheets).

5. Сценарии развёртывания

Система поддерживает три сценария развёртывания в зависимости от условий объекта.

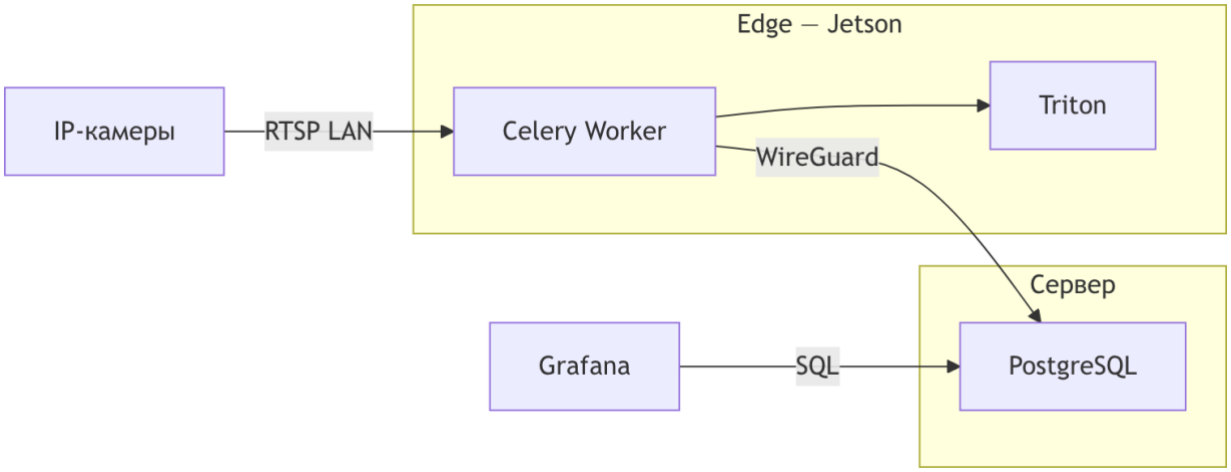
Сценарий А: Облачная обработка (основной)

Камеры объекта передают видеопоток по интернету на центральный сервер. Вся обработка — захват, нейросетевой анализ, хранение результатов — выполняется в облаке. Применяется при наличии стабильного широкополосного канала (от 2 Мбит/с на камеру).



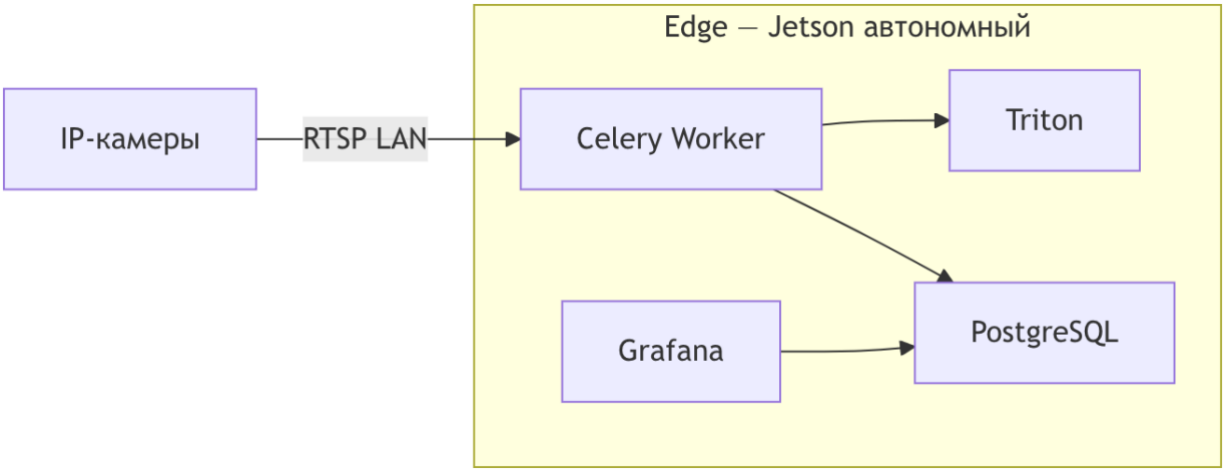
Сценарий Б: Edge-обработка с центральным хранением

На объекте устанавливается edge-устройство (NVIDIA Jetson), которое захватывает и обрабатывает видео локально. Результаты анализа (метрики, события) передаются в центральную базу данных через защищённый WireGuard-туннель. Видеопоток за пределы объекта не выходит.



Сценарий B: Автономный Edge (полностью локальный)

Edge-устройство работает полностью автономно: обрабатывает видео, хранит данные и предоставляет дашборды локально. Не требует постоянного подключения к интернету.



6. Конвейер обработки данных

Система спроектирована для обработки данных из различных источников через плагинную систему. Ниже описан основной реализованный конвейер — обработка видеопотоков с IP-камер.

Обработка каждого видеопотока проходит через последовательность стадий. Для каждой активной камеры создаётся выделенный поток захвата (наследник AbstractCameraWatcherThread), который читает RTSP- или RTMP-поток через GStreamer-пайплайн OpenCV. При потере соединения поток автоматически переподключается с рандомизированной задержкой (от 0 до 60 секунд).

Захваченные кадры проходят фильтр движения: статичные кадры (без изменений относительно предыдущего) отбрасываются для снижения нагрузки на GPU. Прошедшие фильтр кадры группируются в пакеты (размер задаётся переменной BATCH_SIZE) и передаются в пул потоков для параллельной нейросетевой обработки.

Каждый пакет проходит предобработку (масштабирование, паддинг до 640×640 пикселей) и отправляется на Triton Inference Server для детекции объектов моделью YOLO. Результаты детекции — ограничивающие прямоугольники с классами и уровнями уверенности — передаются в цепочку плагинов в порядке топологической сортировки. Каждый плагин анализирует результаты в контексте размеченных полигональных зон и формирует метрики (заполненность полок, число людей в очереди, присутствие персонала) или события (пустая полка, превышение порога очереди). Метрики записываются в PostgreSQL, события — одновременно в базу данных и в систему уведомлений.

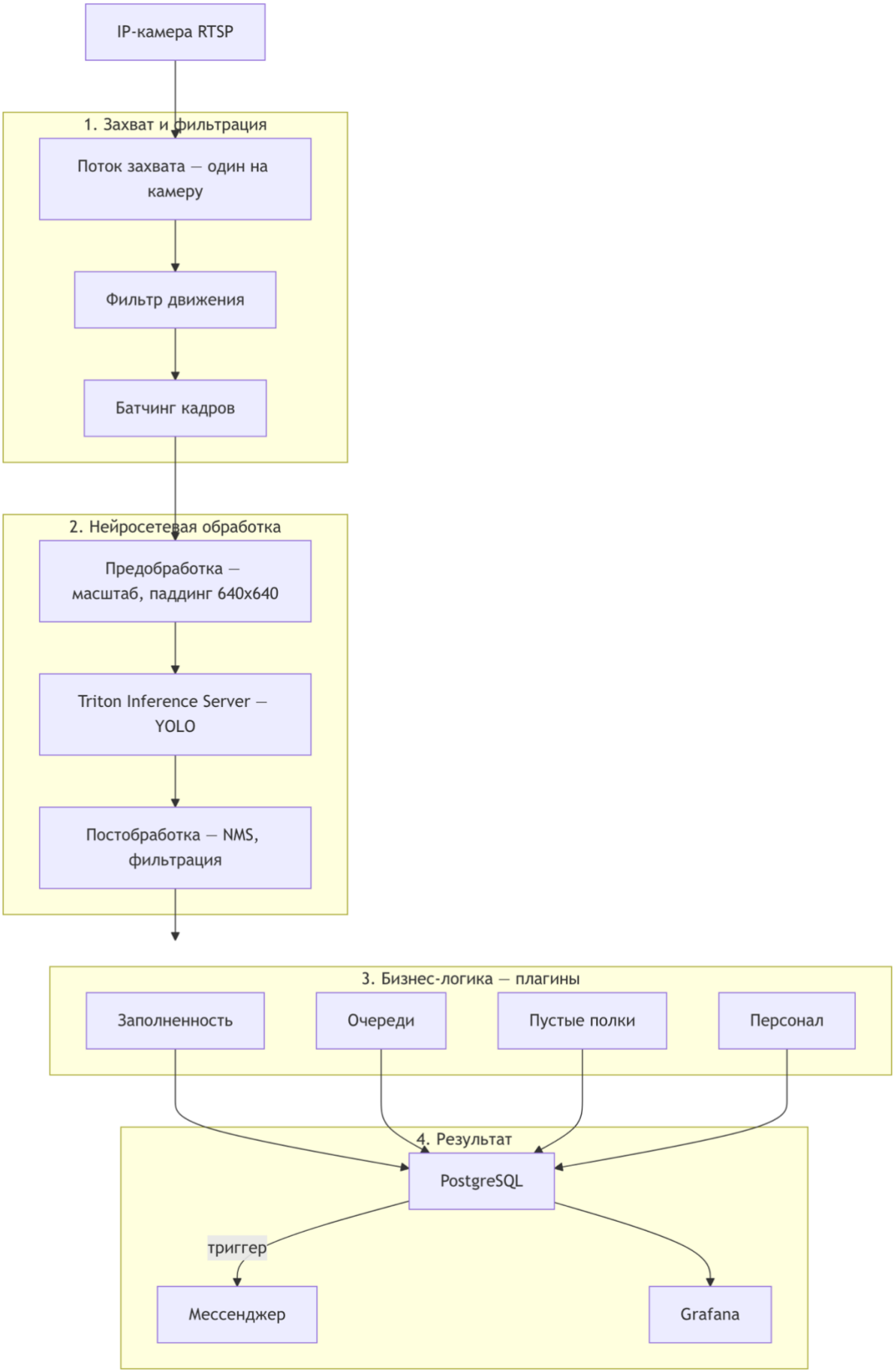
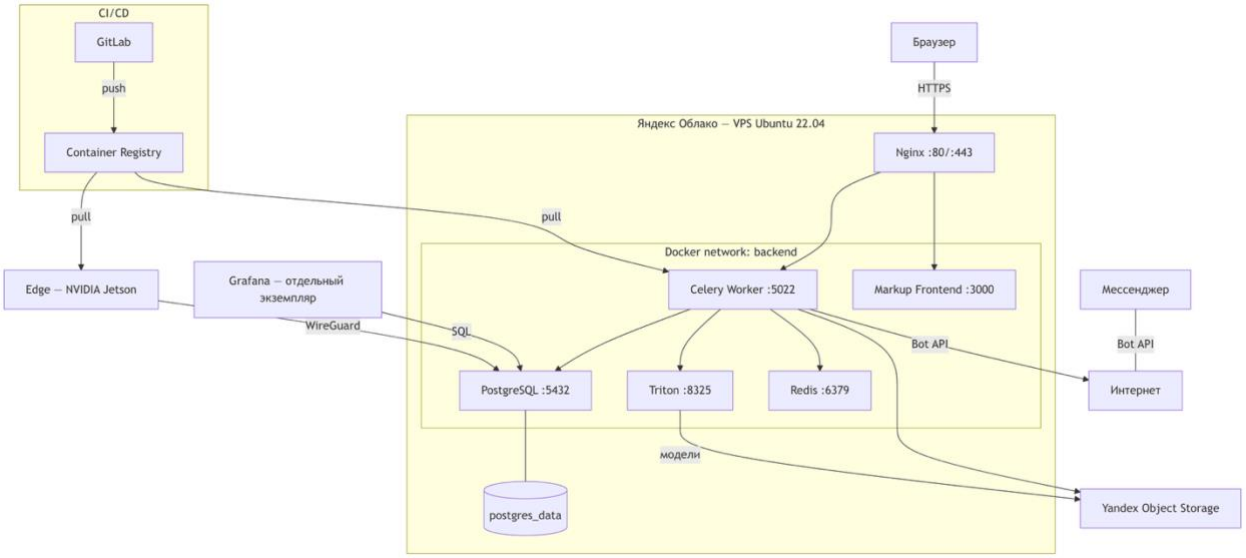


Рисунок к параграфу 6. Конвейер обработки данных.

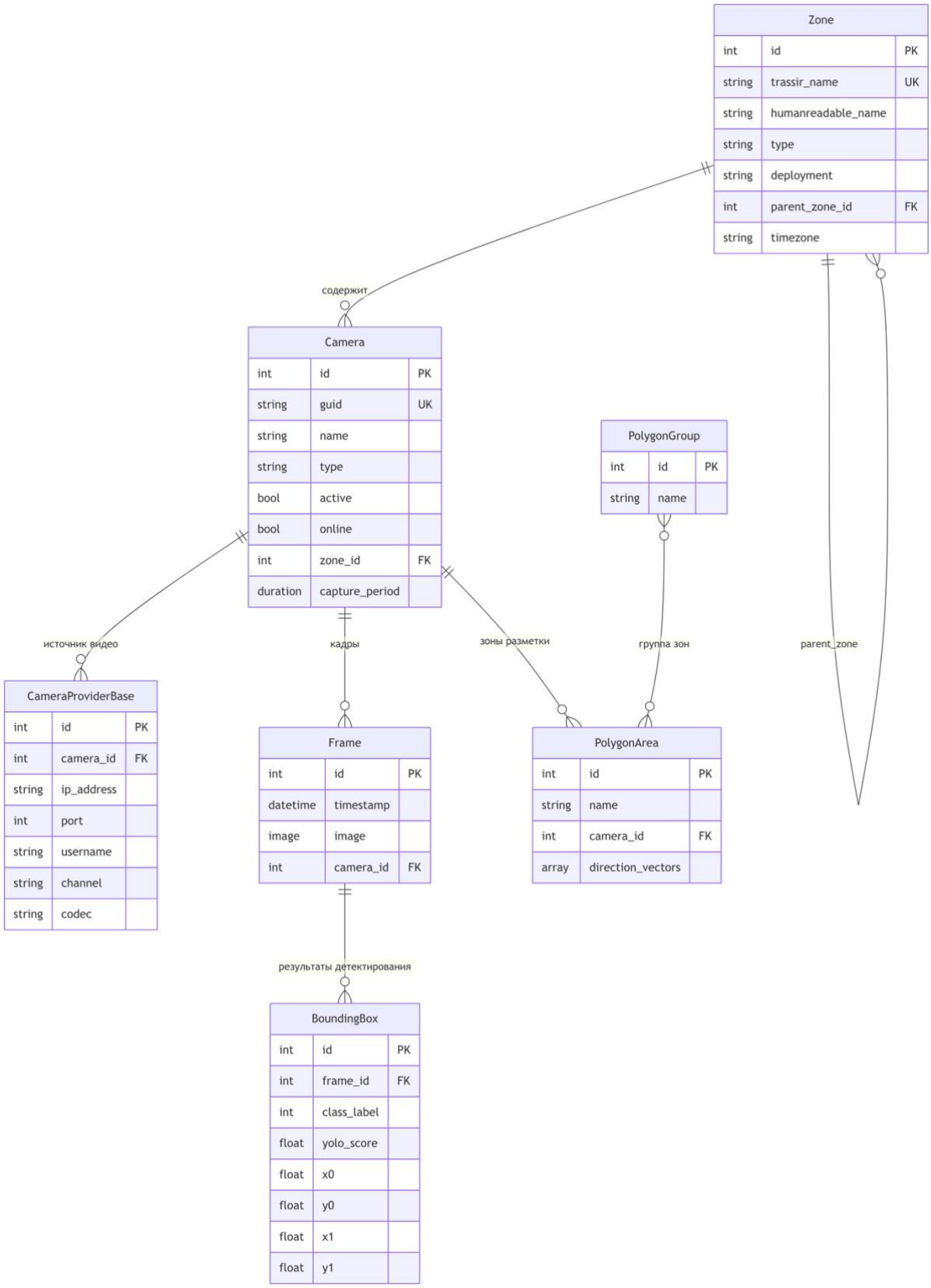
7. Схема производственного развёртывания

Диаграмма отражает типовое развёртывание для сценария А (облачная обработка) с центральным сервером на базе Яндекс Облака.



8. Модель данных (основные сущности)

Ниже представлены основные сущности предметной области. Полная схема базы данных содержит дополнительные таблицы для хранения метрик, событий, настроек плагинов и чек-листов.



9. Интерфейсы и протоколы взаимодействия

Взаимодействие между компонентами системы осуществляется исключительно через сетевые протоколы, без прямых зависимостей между исходным кодом контейнеров.

RTSP/RTMP — приём видеопотоков от IP-камер и видеорегистраторов. Захват реализован через GStreamer-пайплайн библиотеки OpenCV.

HTTP REST API — управление системой. Django REST Framework, порт 5022. Формат обмена: JSON. Документация: Swagger/OpenAPI (drf-yasg). Аутентификация: токены DRF, JWT (django-simple-jwt), сессии.

HTTP (Triton) — запросы нейросетевого вывода от Celery Worker к Triton Inference Server (порт 8325 — HTTP API, порт 8327 — gRPC, порт 8326 — метрики Prometheus). Клиентская библиотека: tritonclient.http.

Redis Protocol — взаимодействие между API, worker и планировщиком через брокер сообщений (порт 6379).

PostgreSQL Wire Protocol — доступ всех компонентов к базе данных (порт 5432). Grafana подключается в режиме только для чтения.

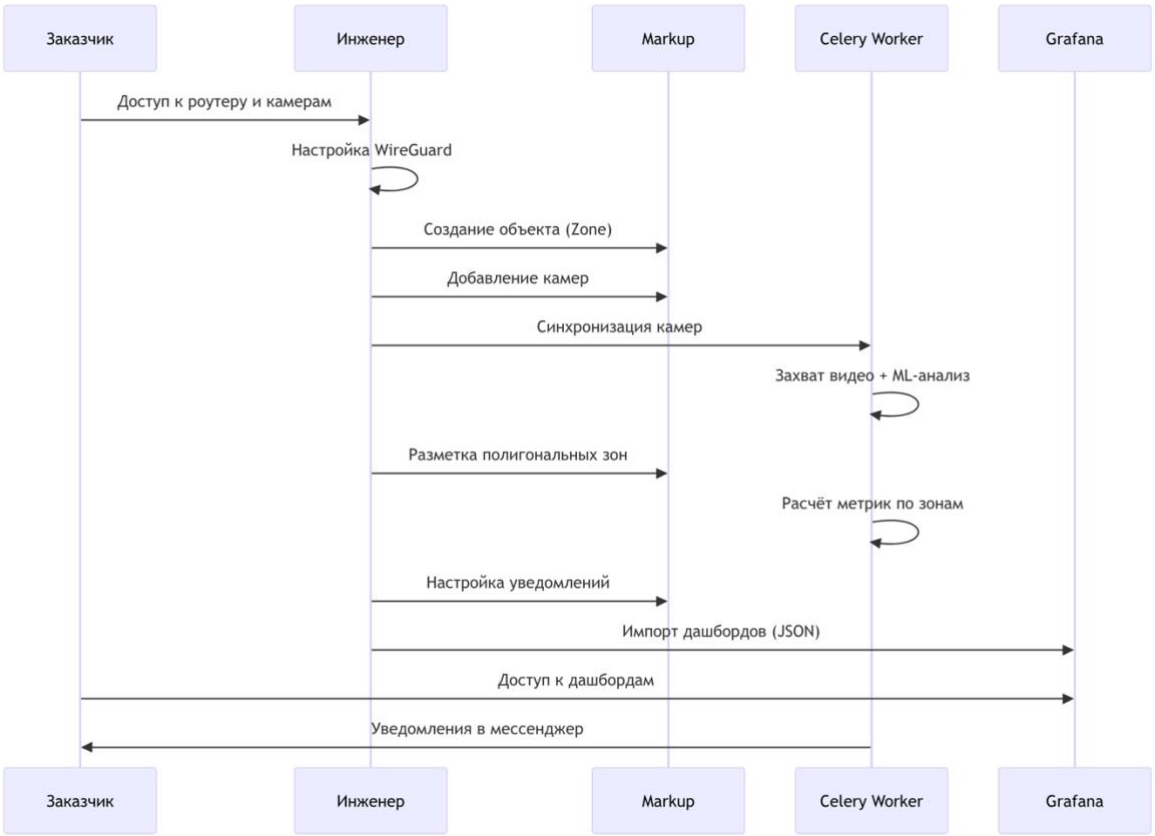
S3 API (HTTPS) — хранение медиафайлов (кадров с камер) и загрузка нейросетевых моделей. Endpoint: Yandex Object Storage.

Telegram Bot API / Max API (HTTPS) — отправка уведомлений и обработка ответов пользователей.

WireGuard (UDP) — защищённый VPN-туннель для подключения камер объектов и edge-устройств к центральному серверу.

10. Цикл подключения нового объекта

Диаграмма отражает типовой процесс подключения нового объекта наблюдения (филиала) к системе.



11. Используемые технологии и библиотеки

Серверная часть (Python)

Основной бэкенд реализован на Python. Веб-фреймворк Django обеспечивает ORM, миграции схемы базы данных и административный интерфейс. Django REST Framework — построение REST API, сериализацию данных и управление аутентификацией. Celery — распределённую обработку задач. HTTP-запросы обслуживаются встроенным сервером Django.

Захват видеопотоков выполняется через OpenCV с GStreamer-пайплайнами. Взаимодействие с сервером нейросетевого вывода — через клиентскую библиотеку tritonclient (HTTP-протокол). Предобработка изображений для нейросетей — PyTorch и torchvision. Работа с S3-хранилищем — Boto3. Интеграция с Telegram — python-telegram-bot. Генерация API-документации — drf-yasg (Swagger/OpenAPI). JWT-аутентификация — django-simple-jwt. Трекинг ошибок — Sentry SDK.

Фронтенд (TypeScript)

Интерфейс настройки реализован как одностраничное приложение на React 18 с TypeScript. Управление состоянием — Redux Toolkit. Компонентная библиотека — Material UI (MUI) 5. Интерактивное рисование полигонов — Konva и React-Konva (2D canvas). HTTP-клиент — Axios. Кэширование и ревалидация данных — SWR. Маршрутизация — React Router 6.

Инфраструктура (стороннее ПО)

Контейнеризация и оркестрация — Docker и Docker Compose. Реляционная СУБД — PostgreSQL 14. Брокер сообщений — Redis. Сервер нейросетевого вывода — NVIDIA Triton Inference Server. Обратный прокси — Nginx. VPN — WireGuard. TLS-сертификаты — Certbot / Let's Encrypt.

Все перечисленные компоненты инфраструктуры распространяются под открытыми лицензиями (Apache 2.0, MIT, BSD, PostgreSQL License).

Приложение А. Реестр используемых библиотек и их лицензий

А.1. Серверная часть (Python, services/requirements.txt)

Библиотека	Версия	Лицензия
django	~=4.2.13	BSD 3-Clause
djangorestframework	~=3.15.2	BSD 3-Clause
celery[redis]	==5.4.0	BSD 3-Clause
gunicorn	~=20.1.0	MIT
tritonclient[http]	==2.30.0	BSD 3-Clause
torch	~=2.4.1	BSD 3-Clause
torchvision	~=0.19.1	BSD 3-Clause
numpy	~=1.26.4	BSD 3-Clause
pandas	~=1.3.3	BSD 3-Clause
scipy	~=1.14.1	BSD 3-Clause
scikit-learn	~=1.5.2	BSD 3-Clause
pillow	~=10.4.0	HPND
matplotlib	~=3.9.2	PSF (matplotlib)
seaborn	==0.13.2	BSD 3-Clause
psycopg2-binary	~=2.9.1	LGPL 3.0
boto3	~=1.34.136	Apache 2.0
django-cors-headers	~=4.4.0	MIT
django-filter	~=24.2	BSD 3-Clause
django-storages	~=1.14.3	BSD 3-Clause
django-prometheus	==2.3.1	Apache 2.0
django-nose	==1.4.7	BSD 3-Clause
djangorestframework-simplejwt	~=5.3.1	MIT
djoser	~=2.2.3	MIT
social-auth-app-django	~=5.4.1	BSD 3-Clause
drf-yasg	~=1.21.7	BSD 3-Clause
PyJWT	==2.9.0	MIT

Библиотека	Версия	Лицензия
PyYAML	~=6.0.2	MIT
requests	==2.26.0	Apache 2.0
requests-oauthlib	==1.3.0	ISC
urllib3	==1.26.11	MIT
python-dotenv	~=1.0.1	BSD 3-Clause
sentry-sdk	~=1.28.1	MIT
raven	~=6.10.0	BSD 3-Clause
pyTelegramBotAPI	~=4.22.1	GPL 2.0
astral	~=3.2.0	Apache 2.0
croniter	~=6.0.0	MIT
websocket-client	~=1.9.0	Apache 2.0
google-api-python-client	—	Apache 2.0
pygsheets	—	MIT
gspread	—	MIT
oauth2client	—	Apache 2.0
openpyxl	—	MIT
pydantic	—	MIT
openai	—	Apache 2.0
requests[socks]	—	Apache 2.0
PySocks	—	BSD 3-Clause
parameterized	==0.9.0	BSD 2-Clause
django-translations-version-4	—	BSD 3-Clause
Jetson.GPIO	—	MIT

Также в проекте используются два внутренних пакета, устанавливаемых из монорепозитория: cctv_core (ядро системы) и point_processing (библиотека обработки полигонов). Эти пакеты являются собственной разработкой ООО «Линза Метрикс».

A.2. Фронтенд (TypeScript, prod-deployments/polygon_markup/package.json)

Библиотека	Версия	Лицензия
react	^18.0.0	MIT
react-dom	^18.0.0	MIT
react-router-dom	^6.0.0	MIT
@reduxjs/toolkit	^1.8.0	MIT
react-redux	^8.0.0	MIT
@mui/material	^5.0.0	MIT
@mui/icons-material	^5.18.0	MIT
@emotion/react	^11.14.0	MIT
@emotion/styled	^11.14.0	MIT
konva	^8.3.5	MIT
react-konva	^18.0.0	MIT
use-image	^1.0.0	MIT
axios	^1.12.0	MIT
swr	^2.0.0	MIT
@visx/axis	^3.12.0	MIT
@visx/grid	^3.12.0	MIT
@visx/scale	^3.12.0	MIT
react-scripts	^5.0.1	MIT

А.3. Базовые Docker-образы

Образ	Назначение	Лицензия
ordkill/opencv-gstreamer-python3.10	Базовый образ бэкенда (Python + OpenCV + GStreamer)	MIT (OpenCV: Apache 2.0, GStreamer: LGPL 2.1)
nvcr.io/nvidia/tritonserver:24.12-py3	Сервер нейросетевого вывода (облако)	BSD 3-Clause
federicolanzani/opencv-cuda-jetson:l4t-base-r32.7.1	Базовый образ бэкенда для Jetson (ARM64 + CUDA)	MIT (OpenCV: Apache 2.0)
nvcr.io/nvidia/l4t-base:r32.7.1	Базовый образ Triton для Jetson (ARM64)	Apache 2.0
node:18-alpine	Сборка фронтенда	MIT
nginx:alpine	Раздача фронтенда	BSD 2-Clause

А.4. Инфраструктурное ПО

Компонент	Лицензия
Docker / Docker Compose	Apache 2.0
PostgreSQL 14	PostgreSQL License
Redis	BSD 3-Clause
NVIDIA Triton Inference Server	BSD 3-Clause
Nginx	BSD 2-Clause
WireGuard	GPL 2.0
Certbot / Let's Encrypt	Apache 2.0
Grafana	AGPL 3.0

Приложение Б. Описание лицензий

Все библиотеки и инструменты, используемые в системе, распространяются под открытыми лицензиями, одобренными Open Source Initiative (OSI). Ни один компонент не является проприетарным. Ниже приведено описание каждого типа лицензии, встречающегося в реестре.

MIT License — разрешительная лицензия, допускающая свободное использование, копирование, модификацию, распространение и коммерческое применение без ограничений при сохранении текста лицензии. Источник: opensource.org/licenses/MIT.

BSD 2-Clause License (Simplified) — разрешительная лицензия, аналогичная MIT. Требует сохранения уведомления об авторских правах при распространении в исходной и бинарной форме. Источник: opensource.org/licenses/BSD-2-Clause.

BSD 3-Clause License (New/Modified) — расширение BSD 2-Clause с дополнительным условием: имя правообладателя не может использоваться для продвижения производных продуктов без письменного разрешения. Источник: opensource.org/licenses/BSD-3-Clause.

Apache License 2.0 — разрешительная лицензия, допускающая свободное коммерческое использование, модификацию и распространение. Требует сохранения уведомления об авторских правах и текста лицензии, а также явного указания внесённых изменений. Включает явное предоставление патентных прав. Источник: opensource.org/licenses/Apache-2.0.

ISC License — разрешительная лицензия, функционально эквивалентная MIT. Источник: opensource.org/licenses/ISC.

PostgreSQL License — разрешительная лицензия, аналогичная MIT/BSD. Допускает свободное использование, копирование, модификацию и распространение в любых целях. Источник: opensource.org/licenses/PostgreSQL.

HPND (Historical Permission Notice and Disclaimer) — разрешительная лицензия, используемая проектом Pillow. Функционально аналогична MIT: допускает свободное использование и распространение при сохранении уведомления. Источник: opensource.org/licenses/HPND.

PSF License (Python Software Foundation) — разрешительная лицензия, допускающая свободное коммерческое использование. Используется библиотекой matplotlib. Источник: docs.python.org/3/license.html.

LGPL 2.1 / 3.0 (GNU Lesser General Public License) — лицензия с ограниченным копилефтом. Допускает использование библиотеки в коммерческом ПО без необходимости раскрытия исходного кода вызывающего приложения при условии, что библиотека подключается динамически (что выполняется при импорте Python-пакетов и использовании GStreamer). Источник: opensource.org/licenses/LGPL-3.0.

GPL 2.0 (GNU General Public License v2) — лицензия с копилефтом. Применяется к pyTelegramBotAPI и WireGuard. Библиотека pyTelegramBotAPI используется как отдельный компонент для взаимодействия с Telegram Bot API и не модифицируется. WireGuard используется как инфраструктурный инструмент на уровне ОС и не входит в состав поставляемого ПО. Источник: opensource.org/licenses/GPL-2.0.

AGPL 3.0 (GNU Affero General Public License v3) — лицензия с копилефтом, расширяющая GPL на сетевое взаимодействие. Применяется к Grafana. Grafana не входит в состав поставляемого ПО — это отдельный продукт, развёртываемый и используемый как внешняя BI-система. Источник: opensource.org/licenses/AGPL-3.0.