Дата создания документа: 08.03.2026
Версия документа: v1.0

Руководство по установке и администрированию 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. Назначение документа

Настоящее руководство предназначено для технических специалистов, выполняющих установку, первоначальную настройку и дальнейшее обслуживание программного комплекса «Платформа Линза Метрикс» в режиме автономного (standalone) развёртывания на инфраструктуре правообладателя или заказчика.

Документ охватывает:

- установку всех компонентов системы на центральном сервере;
- настройку окружения, сетевого доступа и обратного прокси;
- установку периферийного узла обработки видео (Edge, NVIDIA Jetson);
- административные процедуры: запуск, остановку, мониторинг, резервное копирование и обновление системы.

Подключение конкретных объектов наблюдения (филиалов), управление камерами и разметка зон — операции, выполняемые после развёртывания и описанные в Руководстве пользователя (часть для технического пользователя/интегратора).

1.1. Заметка по самостоятельной развертке

Важно. Типовая модель поставки «Платформы Линза Метрикс» — облачный сервис (SaaS): правообладатель (ООО «Линза Метрикс») выполняет развёртывание и обслуживание системы самостоятельно и обеспечивает заказчику полную техническую поддержку. Самостоятельная установка и настройка системы силами заказчика не является целевым сценарием — кодовая база не рассчитана на массовое распространение дистрибутивов в режиме самообслуживания.

Развёртывание системы — сложный технический процесс, в ходе которого могут возникать ошибки, обусловленные спецификой инфраструктуры, настройками окружения и человеческим фактором. ***Проводить установку без участия специалистов ООО «Линза Метрикс» не рекомендуется.***

По всем вопросам, связанным с установкой и развёртыванием, обращайтесь:

- Телефон / мессенджеры: +7 913 203-32-20
- Email: a.serdyukov@linzametrics.com

1.2. Заметка по стороннему ПО

Приведённые в настоящем руководстве команды, конфигурации и скрипты установки компонентов окружения для работы системы носят ориентировочный характер. Компоненты включают (не исчерпывающий список):
- ОС Linux
- Docker
- Grafana
- Реверс-прокси: Nginx, Traefik и т.п.
- База данных SQL
- Redis

- Triton Inference Server

Это всё открытые инструменты со свободной лицензией на использование в коммерческих целях.

В зависимости от даты установки, конкретных версий ПО и ОС, доступности дистрибутивов, процесс настройки рабочей среды может меняться. Перечисленные инструменты не являются частью системы Линза Метрикс — это отдельное ПО, и компания не несет ответственности за их распространение и развертку. Предполагается самостоятельная установка, настройка docker-compose и других скриптов инициализации командой заказчика, изъявляющего желания развернуть систему в своем контуре.

Непосредственной продукцией и зоной ответственности компании являются контейнеры celery и markup-frontend, а также шаблоны аналитических панелей, поставляемые в формате JSON для импорта в Grafana или иную совместимую BI-систему.

2. Состав системы

«Платформа Линза Метрикс» ориентирована на работу в рамках набора Docker-сервисов, управляемых через Docker Compose. Основные используемые компоненты:

Сервис	Образ	Назначение
celery	Собирается из services/Dockerfile	Django REST API + Celery Worker: обработка видеопотоков, аналитика, уведомления, API
markup-frontend	Собирается из prod-deployments/polygon_markup/Dockerfile	React-приложение: интерфейс настройки объектов, оборудования и разметки зон
triton-server	Собирается из triton_server/Dockerfile	NVIDIA Triton Inference Server: выполнение нейросетевых моделей (детекция людей, анализ выкладки и др.)
db	postgres:14	PostgreSQL 14: хранение конфигурации, результатов аналитики, журналов
redis	redis:alpine	Redis: брокер задач Celery, кэш

Все сервисы соединены через изолированную Docker-сеть backend. Celery API доступен снаружи Docker-сети на порту 5022 (по умолчанию ограничен адресом 127.0.0.1), frontend — на порту 3000. Внешний HTTP/HTTPS-доступ обеспечивается обратным прокси (Nginx).

3. Системные требования

3.1. Центральный сервер (облачный или on-premise)

Параметр	Минимальные требования	Рекомендуемые
ОС	Ubuntu 20.04 LTS (x86-64)	Ubuntu 22.04 LTS (x86-64)
ЦПУ	4 ядра	8+ ядер
ОЗУ	16 ГБ	32+ ГБ
GPU	—	NVIDIA (VRAM 6+ ГБ)
Диск	100 ГБ SSD	500+ ГБ SSD
Сеть	100 Мбит/с	1 Гбит/с
ПО	Docker Engine 24+, Docker Compose v2	—

3.2. Периферийный узел обработки (Edge, опционально)

Параметр	Значение
Устройство	NVIDIA Jetson Nano / Orin Nano / Orin NX
ОС	Ubuntu 20.04 LTS (JetPack 5.x/6.x)
ОЗУ	8–32 ГБ (объединённая RAM/VRAM)
Хранилище	64+ ГБ microSD или SSD через M.2
Сеть	Ethernet / Wi-Fi с доступом к интернету (WireGuard)

3.3. Требования к сетевой инфраструктуре подключаемого к системе объекта

- IP-камеры с поддержкой RTSP (H.264/H.265);
- роутер с поддержкой WireGuard **или** наличие публичного IP-адреса с возможностью проброса портов;
- пропускная способность канала: не менее 2 Мбит/с на одну камеру при центральной обработке; при edge-режиме — 1 Мбит/с для передачи результатов.

3.4. Открытые порты центрального сервера

Порт	Протокол	Назначение
80	TCP	HTTP (перенаправление на HTTPS и certbot challenge)
443	TCP	HTTPS (API + Markup UI через Nginx)
5432	TCP	PostgreSQL (при необходимости прямого доступа к БД)

4. Установка (центральный сервер)

4.1. Установка Docker

```
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
```

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo $VERSION_CODENAME) stable" \
  | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin
sudo usermod -aG docker $USER

# Перелогиньтесь или выполните newgrp docker
```

4.1. Установка Nginx и Certbot

```
sudo apt-get install -y nginx certbot python3-certbot-nginx
```

4.3. Получение исходного кода

Доступ к репозиторию предоставляется правообладателем. Клонирование:

```
git clone https://gitlab.linzametrics.com/linza/cctv-backend.git
cd cctv-backend
```

Для клонирования по SSH необходимо передать правообладателю публичный ключ SSH-пары сервера и получить права доступа к репозиторию.

4.4. Настройка переменных окружения

4.4.1. База данных (services/environments/db.env)

```
mkdir -p services/environments
cat > services/environments/db.env << 'EOF'
POSTGRES_USER=<ПОЛЬЗОВАТЕЛЬ_БД>
POSTGRES_PASSWORD=<ПАРОЛЬ_БД>
POSTGRES_DB=cctv
POSTGRES_HOST=db
POSTGRES_PORT=5432
POSTGRES_SCHEMA=<ИМЯ_СХЕМЫ>
EOF
```

Переменная	Описание
`POSTGRES_DB`	Имя базы данных
`POSTGRES_USER`	Пользователь PostgreSQL
`POSTGRES_PASSWORD`	Пароль PostgreSQL
`POSTGRES_HOST`	Хост БД (по умолчанию: `db`)
`POSTGRES_PORT`	Порт БД (по умолчанию: `5432`)
`POSTGRES_SCHEMA`	Схема PostgreSQL

Файл infra/db/init.sql должен содержать команду создания схемы, соответствующей POSTGRES_SCHEMA. Шаблон уже присутствует в репозитории; при необходимости отредактируйте его:

```
CREATE SCHEMA IF NOT EXISTS <ИМЯ_СХЕМЫ>;
```

4.4.2. Приложение (services/environments/app.env)

```
cat > services/environments/app.env << 'EOF'
VM_ADDRESS=<IP_СЕРВЕРА>
DEPLOYMENT_HOST=<IP_СЕРВЕРА>
DEPLOYMENT_KEY=<КЛЮЧ_ДЕПЛОЙМЕНТА>
DOMAIN=<ДОМЕН_API>
CORS_EXTRA_ORIGINS=https://<ДОМЕН_ФРОНТЕНДА>,http://<ДОМЕН_ФРОНТЕНДА>

DEFAULT_TRITON_URL=triton-server:8325
GUNICORN_WORKERS=2
DEBUG=False

PLUGINS=occupancy_engine,ops_tasks,periodic_reports,checkout_control,empty_shelf,table
_control,security
MODULES=people_core,telegram_messages,occupancy_core,google_apis

ENABLE_S3=True
FRAME_OCCUPANCY_ENABLED=False
POLYGON_OCCUPANCY_ENABLED=False
BATCH_SIZE=1
QUEUE_THREADS_NUM=2

DJANGO_SALT=<СЛУЧАЙНАЯ_СТРОКА_≥50_СИМВОЛОВ>

S3_ACCESS_KEY=<S3_KEY>
S3_SECRET_ACCESS_KEY=<S3_SECRET>
S3_BUCKET=<ИМЯ_S3_БАКЕТА>
S3_URL=https://storage.yandexcloud.net
S3_MEDIA_PREFIX=<ПРЕФИКС_В_БАКЕТЕ>

TELEGRAM_ALERT_TOKEN=<ТОКЕН_БОТА>
TELEGRAM_BOT_NAME=<ИМЯ_БОТА>
TELEGRAM_DATA_SAVE_PATH=telegram/<ПРЕФИКС>

YOLO_MODEL_NAME=yolov5
YOLO_MODEL_VERSION=1
YOLO_DETECTABLE_CLASS_IDS=0
MODEL_EMPTY_SHELF_NAME=yolov5-freezer

REDIS_URL=redis:6379
AUTH=True
```

Переменная	Описание
`DJANGO_SALT`	Секретный ключ Django (случайная строка ≥50 символов)
`DEPLOYMENT_KEY`	Ключ деплоймента — сервис обрабатывает только объекты (Zone), у которых поле deployment совпадает с этим значением
`DEPLOYMENT_HOST`	IP-адрес или домен сервера (добавляется в ALLOWED_HOSTS Django)
`DOMAIN`	Домен API (добавляется в ALLOWED_HOSTS и CSRF-настройки)
`CORS_EXTRA_ORIGINS`	Разрешённые источники CORS (адреса, с которых работает frontend)
`REDIS_URL`	Адрес Redis (по умолчанию: `redis:6379`)
`DEFAULT_TRITON_URL`	Адрес Triton-сервера (по умолчанию: `triton-server:8325`)
`YOLO_MODEL_NAME`	Имя модели детектирования объектов в Triton
`YOLO_DETECTABLE_CLASS_IDS`	Список идентификаторов классов (через запятую)
`PLUGINS`	Список активных плагинов (через запятую)

`MODULES`	Список активных модулей (через запятую)

4.4.3. Ключи доступа Triton к S3 (.env в корне проекта)

Triton загружает нейросетевые модели напрямую из S3-бакетов. Ключи передаются через переменные окружения Docker Compose:

```
cat > .env << 'EOF'
TRITON_AWS_ACCESS_KEY_ID=<КЛЮЧ_ДОСТУПА_К_S3>
TRITON_AWS_SECRET_ACCESS_KEY=<СЕКРЕТНЫЙ_КЛЮЧ_S3>
EOF
```

Эти переменные подставляются в docker-compose.yml через механизм интерполяции Docker Compose.

4.4.4. Учётная запись Google API (services/environments/service_account.json)

Для работы модуля `google_apis` (синхронизация чек-листов из Google Sheets) необходим файл сервисного аккаунта Google. Получите JSON-файл сервисного аккаунта в консоли Google Cloud и разместите его:

```
cp /path/to/your-service-account.json services/environments/service_account.json
```

4.4.5. Конфигурация frontend (prod-deployments/polygon_markup/.env)

```
cat > prod-deployments/polygon_markup/.env << 'EOF'
REACT_APP_API_BASE=https://<ДОМЕН_API>
REACT_APP_SERVER_URLS=https://<ДОМЕН_API>
EOF
```

4.5. Сборка и запуск сервисов

```
cd ~/cctv-backend
mkdir -p logs/celery

# Сборка Docker-образов
docker compose build

# Запуск всех сервисов
docker compose up -d
```

Миграции базы данных и инициализация схемы выполняются автоматически при первом запуске контейнера celery. Дождитесь, пока все контейнеры перейдут в состояние healthy / running:

```
docker compose ps
```

Ожидаемый результат:

Сервис	Статус
cctv-backend-db-1	healthy
cctv-backend-redis-1	running
cctv-backend-celery-1	running
cctv-backend-triton-server-1	running
cctv-backend-markup-frontend-1	running

Запуск Triton может занять несколько минут — сервис загружает модели из S3 при старте. Следите за его логами: docker compose logs -f triton-server.

4.6. Настройка обратного прокси (Nginx)

```
sudo tee /etc/nginx/sites-available/linza-metrics << 'NGINX'
server {
    listen 80;
    server_name <ДОМЕН_API>;

    location / {
        proxy_pass http://127.0.0.1:5022;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        client_max_body_size 100m;
    }
}

server {
    listen 80;
    server_name <ДОМЕН_ФРОНТЕНДА>;

    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
NGINX

sudo ln -sf /etc/nginx/sites-available/linza-metrics \
  /etc/nginx/sites-enabled/linza-metrics
sudo rm -f /etc/nginx/sites-enabled/default
sudo nginx -t && sudo systemctl restart nginx
```

Замените <ДОМЕН_API> и <ДОМЕН_ФРОНТЕНДА> на фактические доменные имена, указанные в DNS A-записях сервера.

4.7. Получение TLS-сертификатов (Let's Encrypt)

```
sudo certbot --nginx \
  -d <ДОМЕН_API> \
  -d <ДОМЕН_ФРОНТЕНДА> \
  --non-interactive --agree-tos --email <EMAIL_АДМИНИСТРАТОРА>
```

Certbot автоматически обновит конфигурацию Nginx для HTTPS и настроит автообновление сертификатов (через systemd timer или cron).

4.8. Инициализация данных

4.8.1. Создание суперпользователя

```
docker exec cctv-backend-celery-1 python manage.py shell -c "
from django.contrib.auth import get_user_model
User = get_user_model()
User.objects.create_superuser('admin', 'admin@example.com', '<ПАРОЛЬ>')
"
```

4.8.2. Инициализация настроек обработки

После создания первого объекта наблюдения (Zone) необходимо создать запись PeopleCoreSettings — без неё пайплайн обработки видео не запустится:

```
docker exec cctv-backend-celery-1 python manage.py shell -c "
from modules.people_core.models import PeopleCoreSettings
from cctv_core.models import Zone
for zone in Zone.objects.all():
```

```
    PeopleCoreSettings.objects.get_or_create(
        zone=zone,
        defaults={'occupancy_threshold': 0.5}
    )
print('PeopleCoreSettings: OK')
"
```

Перезапустите celery для применения:

```
docker compose restart celery
```

4.9. Проверка работоспособности

Компонент	Адрес	Ожидаемый результат
API (список зон)	https://<ДОМЕН_API>/core/api/zones/?format=json	JSON со списком зон
API (список камер)	https://<ДОМЕН_API>/core/api/cameras/?format=json	JSON со списком камер
Markup UI	https://<ДОМЕН_ФРОНТЕНДА>/	Страница входа в систему

Дополнительные проверки:

```
# Статус всех контейнеров
docker compose ps

# Логи основного сервиса (последние 50 строк)
docker compose logs --tail 50 celery

# Логи сервера моделей
docker compose logs --tail 50 triton-server

# Проверка очереди Celery
docker compose exec celery celery -A services.celery_config inspect active
```

5. Установка периферийного узла (Edge, NVIDIA Jetson)

5.1. Первичная настройка устройства

Выполняется скриптом `setup_jetson.sh` (поставляется отдельно), который:

- устанавливает Docker Engine с поддержкой NVIDIA Container Runtime;
- переносит хранилище Docker на внешний накопитель (SD / SSD);
- отключает графический интерфейс для экономии ресурсов;
- создаёт swap-файл 4 ГБ;
- настраивает автозапуск Docker при старте системы.

5.2. Настройка WireGuard-подключения

```
sudo apt install wireguard

# Сгенерировать пару ключей
wg genkey | tee privatekey | wg pubkey > publickey

# Создать конфигурацию /etc/wireguard/wg0.conf
# (конфигурация предоставляется правообладателем)

sudo systemctl enable wg-quick@wg0
```

```
sudo systemctl start wg-quick@wg0
```

После подключения Jetson становится доступен через WireGuard с центрального сервера.

5.3. Запуск сервисов на Jetson

Используется специализированный профиль Docker Compose с Jetson-образами:

```
docker compose -f docker-compose.yml up -d celery-jetson triton-server-jetson db redis
```

Jetson-образы собраны на базе NVIDIA L4T и используют GPU-ускорение через NVIDIA Container Runtime.

6. Администрирование и обслуживание

6.1. Управление сервисами

```
# Запуск всех сервисов
docker compose up -d

# Остановка всех сервисов (данные сохраняются)
docker compose stop

# Перезапуск конкретного сервиса
docker compose restart celery
docker compose restart triton-server

# Остановка и удаление контейнеров (данные в томах сохраняются)
docker compose down
```

6.2. Просмотр журналов

```
# Логи всех сервисов (с момента запуска, последние 100 строк)
docker compose logs --tail 100

# Следить за логами основного сервиса в реальном времени
docker compose logs -f celery

# Логи сервера моделей
docker compose logs -f triton-server

# Логи фронтенда
docker compose logs -f markup-frontend
```

Логи сервиса celery также пишутся в файлы в директории logs/celery/ (смонтирована как том).

6.3. Проверка состояния очереди задач

```
# Активные задачи Celery
docker compose exec celery celery -A services.celery_config inspect active

# Запланированные задачи (Celery Beat)
docker compose exec celery celery -A services.celery_config inspect scheduled
```

6.4. Резервное копирование базы данных

```
# Создать дамп базы данных
docker compose exec db pg_dump \
  -U $POSTGRES_USER $POSTGRES_DB \
  > backup_$(date +%Y%m%d_%H%M%S).sql
```

Рекомендуется настроить регулярное резервное копирование через cron или аналогичный механизм. Дамп следует сохранять на отдельном хранилище (S3, NFS и т.д.).

6.5. Обновление системы

```
cd ~/cctv-backend

# Получить обновления из репозитория
git pull

# Пересобрать изменившиеся образы
docker compose build

# Перезапустить сервисы
docker compose up -d
```

Миграции базы данных применяются автоматически при рестарте контейнера celery.

Перед обновлением рекомендуется создать резервную копию базы данных (см. п. 6.4).

6.6. Мониторинг

Prometheus-метрики публикуются контейнером celery на порту 9091 (внутри Docker-сети). Они включают:

- время обработки кадров;
- глубину очереди Celery;
- число активных видеопотоков;
- использование памяти и CPU.

При наличии отдельного экземпляра Grafana подключите Prometheus как источник данных и импортируйте дашборд «System Health».

6.7. Трекинг ошибок (Sentry)

Если в app.env задана переменная SENTRY_DSN, все необработанные исключения в сервисе celery автоматически отправляются в Sentry. Для просмотра ошибок используйте web-интерфейс вашего Sentry-инстанса.

Приложение. Перечень переменных окружения

services/environments/app.env (основные)

Переменная	Описание
DEPLOYMENT_KEY	Ключ деплоймента: сервис обрабатывает только Zone с совпадающим значением
DEPLOYMENT_HOST	IP или домен сервера — добавляется в ALLOWED_HOSTS
DOMAIN	Домен API — добавляется в ALLOWED_HOSTS и CSRF
CORS_EXTRA_ORIGINS	Разрешённые источники CORS (адреса frontend)
DJANGO_SALT	Секретный ключ Django (≥50 символов, случайная строка)
DEBUG	False в production
AUTH	True — обязательная авторизация API
PLUGINS	Список активных плагинов через запятую
MODULES	Список активных модулей через запятую
DEFAULT_TRITON_URL	Адрес Triton: triton-server:8325
ENABLE_S3	True — хранить медиафайлы в S3
S3_ACCESS_KEY / S3_SECRET_ACCESS_KEY	Ключи доступа к S3 для хранения медиа
S3_BUCKET	Имя S3-бакета для медиафайлов
S3_URL	Endpoint S3 (напр. https://storage.yandexcloud.net)
S3_MEDIA_PREFIX	Префикс пути в бакете (изоляция данных разных инсталляций)
TELEGRAM_ALERT_TOKEN	Токен Telegram-бота уведомлений
TELEGRAM_BOT_NAME	Имя Telegram-бота
YOLO_MODEL_NAME	Имя модели детекции объектов в Triton
YOLO_DETECTABLE_CLASS_IDS	Идентификаторы классов для детекции (через запятую)
REDIS_URL	Адрес Redis: redis:6379

services/environments/db.env

Переменная	Описание

POSTGRES_USER	Пользователь PostgreSQL
POSTGRES_PASSWORD	Пароль PostgreSQL
POSTGRES_DB	Имя базы данных
POSTGRES_HOST	Хост БД — db (имя Docker-сервиса)
POSTGRES_PORT	Порт — 5432
POSTGRES_SCHEMA	Схема PostgreSQL

.env (корень проекта, переменные для Docker Compose)

Переменная	**Описание**
TRITON_AWS_ACCESS_KEY_ID	Ключ доступа S3 для загрузки моделей Triton
TRITON_AWS_SECRET_ACCESS_KEY	Секретный ключ S3 для загрузки моделей Triton
CELERY_PORT_BINDING	Привязка порта celery (по умолчанию 127.0.0.1:5022)
FRONTEND_PORT_BINDING	Привязка порта frontend (по умолчанию 127.0.0.1:3000)