Дата создания документа: 08.03.2026
Версия документа: v1.0

Описание процессов обеспечения поддержания жизненного цикла 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. Общая модель жизненного цикла

Разработка и сопровождение «Платформы Линза Метрикс» осуществляется по итеративной модели с непрерывной доставкой (Continuous Delivery). Жизненный цикл включает следующие основные стадии:

1. Планирование и управление требованиями
2. Проектирование
3. Разработка и рецензирование кода
4. Тестирование
5. Сборка и контейнеризация
6. Поставка и развёртывание
7. Эксплуатация и мониторинг
8. Устранение неисправностей
9. Совершенствование и обновление моделей машинного обучения
10. Документирование
11. Обучение и квалификация персонала
12. Приобретение компонентов
13. Прекращение поддержки версий Разработка и рецензирование кода

2. Инструменты разработки

Категория	Инструмент	Версия / описание
Система контроля версий	Git	—
Хостинг рабочего репозитория для разработчиков	GitLab CE (self-hosted)	Яндекс Облако, Ubuntu 22.04
CI/CD	GitLab CI/CD	Сборка, доставка образов, развёртывание
Язык программирования	Python	3.10–3.12
Фреймворк	Django	4.2.x
Фронтенд	React, TypeScript	18.x, 4.7.x
Контейнеризация	Docker Engine, Docker Compose v2	—
Реестр образов	Yandex Container Registry	Яндекс Облако
Пайплайны данных и моделей	Dagster	Оркестрация внутренних пайплайнов
Среда разработки (IDE)	JetBrains PyCharm, VS Code	—
Трекер задач	GitLab Issues / Boards	—
Трекинг ошибок	Sentry	Self-hosted / Cloud
Управление контейнерами	Portainer	Веб-интерфейс для Docker
Мониторинг	Prometheus	Сбор метрик

3. Управление требованиями и планирование

Задачи разработки фиксируются в системе управления задачами GitLab Issues. Каждая задача содержит описание требования или дефекта, метку приоритета, исполнителя и срок выполнения.

Планирование итераций (1–2 недели) проводится на еженедельных встречах команды. Приоритизация ведётся исходя из обратной связи от операторов системы, данных о зафиксированных ошибках и стратегических целей продукта.

3.1. Проектирование

Проектирование новых функций и изменений архитектуры выполняется в рамках итераций планирования. При разработке новой функции ответственный разработчик определяет, какие компоненты системы затрагиваются, какие плагины или модули необходимо создать или модифицировать, и какие изменения в модели данных потребуются.

Архитектура системы спроектирована для расширения без модификации ядра: новые функции реализуются как плагины (PeopleProcessingPlugin, ProviderPlugin) с декларативным описанием зависимостей. Это позволяет проектировать новые возможности в изоляции от существующей бизнес-логики. При необходимости изменений в схеме базы данных используется механизм миграций Django, который обеспечивает версионирование и воспроизводимость изменений структуры данных.

Для крупных изменений (новый тип источника данных, новый канал уведомлений, изменение конвейера обработки) проводится обсуждение на встрече команды с фиксацией решений в задаче GitLab Issues.

4. Разработка и контроль версий

4.1. Ветвление

Репозиторий содержит две основные ветки:

- master — стабильная ветка, соответствующая производственной версии;
- dev — ветка интеграции разрабатываемых изменений.

Разработка ведётся в именованных ветках вида автор/описание (например, almdudleer/checklists). Срочные исправления могут выполняться напрямую в master с последующим слиянием в dev.

4.2. Рецензирование кода (Code Review)

Все изменения попадают в основные ветки через Merge Request (запрос на слияние). При рецензировании проверяются корректность реализации, соответствие требованиям и отсутствие нежелательных побочных эффектов.

4.3. Контроль качества кода

В проекте применяются инструменты автоматической проверки стиля и форматирования кода.

4.4. Документирование

Документация поддерживается на протяжении всего жизненного цикла системы и обновляется при каждом значимом изменении функциональности.

Состав документации:

- **Описание функциональных характеристик** — назначение системы, перечень функций, сценарии использования.
- **Описание технической архитектуры** — структура компонентов, диаграммы, интерфейсы взаимодействия, модель данных, реестр зависимостей.
- **Руководство по установке и администрированию** — развёртывание системы, настройка окружения, обслуживание.
- **Руководство пользователя** — работа с системой для технического пользователя (настройка объектов, камер, зон) и конечного пользователя (дашборды, уведомления, чек-листы).
- **Описание жизненного цикла ПО** — настоящий документ.
- **Описание средств хранения исходного текста** — инструменты и процессы хранения кода и сборки.

Документация хранится в системе документооборота правообладателя. Актуализация выполняется ответственным разработчиком или инженером по внедрению по мере внесения изменений в систему.

4.5. Приобретение компонентов

Все программные компоненты, входящие в состав системы и её окружения, являются свободным программным обеспечением с открытым исходным кодом (лицензии MIT, BSD, Apache 2.0, LGPL, GPL и др.) либо собственной разработкой ООО «Линза Метрикс». Приобретение проприетарных программных компонентов не требуется.

Полный реестр используемых библиотек и их лицензий приведён в Приложении А документа «Описание технической архитектуры».

Аппаратная инфраструктура (серверы, edge-устройства) приобретается или арендуется в соответствии с потребностями конкретного развёртывания. Типовая инфраструктура: виртуальные серверы Яндекс Облака, периферийные устройства NVIDIA Jetson для edge-сценариев.

5. Тестирование

5.1. Виды тестов

Вид	Описание
Unit-тесты	Тестирование отдельных функций и методов (модули ops_tasks, workplaces, isapp, ядро)
Integration-тесты	Тестирование API-эндпоинтов и взаимодействия компонентов
Нагрузочное тестирование	Проверка на стейджинг-среде при крупных изменениях

5.2. Запуск тестов

Тесты запускаются в изолированной среде с помощью специального Docker Compose конфигурации (docker-compose-ci-tests.yml), которая поднимает тестовую базу данных и брокер сообщений. Запуск может выполняться локально разработчиком или на сервере.

5.3. Тестовые данные

Для тестирования конвейера обработки видео используются наборы тестовых изображений. Инференс нейросетевой модели при тестировании выполняется с использованием подменных объектов (mock).

6. Сборка и контейнеризация

6.1. Сборочные образы

Программное обеспечение распространяется в виде Docker-контейнеров. Предусмотрены сборочные конфигурации:

Dockerfile	Назначение
services/Dockerfile	Основной бэкенд (x86-64) — API, обработка данных, уведомления
services/Dockerfile.jetson	Бэкенд для NVIDIA Jetson (ARM64, CUDA)
triton_server/Dockerfile	Сервер нейросетевого вывода (x86-64)
triton_server/Dockerfile.jetson	Сервер нейросетевого вывода для Jetson (ARM64)
prod-deployments/polygon_markup/Dockerfile	Фронтенд — интерфейс настройки (React SPA)

6.2. Процесс сборки

Сборка образов бэкенда выполняется через GitLab CI. Пайплайн запускается вручную на ветках master и dev. Собранный образ публикуется в Yandex Container Registry с тегом версии (v1.<номер_пайплайна>) и тегом latest.

Образы для Jetson-устройств и фронтенда собираются локально или в рамках процедуры развёртывания.

6.3. Доставка на периферийные устройства (Jetson)

Для edge-устройств (NVIDIA Jetson) применяется отдельная процедура: GitLab CI подключается к устройству по SSH (при необходимости через VPN), копирует исходный код и выполняет сборку и запуск контейнеров непосредственно на устройстве. Это обусловлено различиями в архитектуре процессоров (ARM64 vs. x86-64).

6.4. Воспроизводимость сборки

Все зависимости зафиксированы с точными версиями в requirements.txt (Python) и package.json (frontend). Для ускорения повторных сборок применяется кеширование промежуточных слоёв Docker.

7. Развёртывание

7.1. Серверная инфраструктура

Производственные серверы размещены в Яндекс Облаке (Яндекс Cloud) на виртуальных машинах под управлением Ubuntu 22.04 LTS. Конфигурации развёртывания для разных объектов хранятся в выделенном репозитории.

7.2. Процедура обновления

1. После прохождения тестов и слияния изменений в основную ветку GitLab CI автоматически публикует новый Docker-образ.
2. Инженер подключается к производственному серверу и выполняет загрузку новых образов.
3. Сервисы перезапускаются без прерывания работы системы.
4. При наличии изменений схемы базы данных применяются миграции.
5. Работоспособность проверяется через системы мониторинга.

7.3. Откат версии

При выявлении критических проблем после обновления система переводится на предыдущий тег образа. Резервные копии базы данных создаются перед каждым обновлением, содержащим изменения схемы БД.

8. Мониторинг и эксплуатация

8.1. Метрики работоспособности

Сервис предоставляет метрики для системы мониторинга:

- число активных потоков камер;
- глубина очереди обработки кадров;
- время ответа на запросы;
- число ошибок при обработке видео.

8.2. Алертинг

Настроены автоматические уведомления команды при:

- недоступности камеры более 5 минут;
- накоплении необработанных задач в очереди;
- превышении времени ответа системы.

8.3. Логирование

Структурированные записи журнала событий собираются от всех компонентов системы и доступны для диагностики. Выполняется ротация журналов по времени.

9. Устранение неисправностей

9.1. Классификация инцидентов

Приоритет	Описание	Целевое время реакции
P1 — Критический	Система недоступна, данные не собираются	1 час
P2 — Высокий	Сбой отдельных функций (алерты не отправляются, часть камер недоступна)	4 часа
P3 — Средний	Деградация качества детекций, задержки данных	24 часа
P4 — Низкий	Косметические проблемы, пожелания по улучшению	По плану итерации

9.2. Типичные проблемы и решения

Симптом	Вероятная причина	Решение
Камера помечена как офлайн	Обрыв сетевого соединения	Проверить сеть/WireGuard, перезапустить службу наблюдения
Нет данных в дашборде	Служба обработки недоступна	Перезапустить службу обработки и брокер сообщений
Ошибка распознавания	Сервер нейросетевого вывода недоступен	Перезапустить сервер вывода, проверить загрузку моделей
Telegram-уведомления не приходят	Неверная привязка чата	Проверить настройки чата в административном интерфейсе

Подробнее см. в Руководстве по установке и администрированию системы.

9.3. Обращение в техподдержку

Сообщения об инцидентах направляются в систему трекинга задач или напрямую команде. К обращению рекомендуется прикладывать:

- описание проблемы и шаги воспроизведения;
- фрагмент записей журнала;
- скриншот из системы мониторинга;
- идентификатор камеры и объекта.

10. Совершенствование ПО

10.1. Обновление нейросетевых моделей

Цикл улучшения моделей компьютерного зрения:

1. **Сбор данных** — накопление изображений с производственных камер, помеченных в процессе эксплуатации.
2. **Разметка** — аннотирование данных в инструменте разметки изображений.
3. **Обучение** — запуск тренировочного процесса на GPU-серверах. Оркестрация пайплайнов обучения выполняется через Dagster.
4. **Валидация** — проверка метрик качества (точность, полнота) на тестовой выборке.
5. **Экспорт** — конвертация обученных весов в производственный формат (ONNX/TensorRT).
6. **Доставка** — загрузка новых моделей в S3-хранилище (для облачного сценария) или на локальное устройство (для edge-сценария) и перезапуск сервера вывода.

10.2. Цикл обратной связи

Операторы фиксируют ложные срабатывания и пропуски через интерфейс аннотирования. Накопленные данные поступают в обучающую выборку следующего цикла. Качество отслеживается через метрики мониторинга и периодические ревью.

10.3. Политика версионирования

Используется семантическое версионирование: **MAJOR.MINOR.PATCH**.

- **MAJOR** — кардинальные изменения архитектуры, несовместимые с предыдущими версиями.
- **MINOR** — новые функции без потери обратной совместимости.
- **PATCH** — исправление ошибок.

11. Персонал и инфраструктура разработки

11.1. Организация-разработчик

Разработку, совершенствование, устранение неисправностей и техническую поддержку «Платформы Линза Метрикс» осуществляет правообладатель — ООО «Линза Метрикс».

11.2. Состав команды

Роль	Ответственность
Backend-разработчик (Python/Django)	Разработка, поддержка, исправление ошибок
Frontend-разработчик (React/TypeScript)	Frontend-разработчик (React/TypeScript)
ML-инженер	Обучение и валидация нейросетевых моделей, оптимизация
DevOps-инженер	CI/CD, инфраструктура, мониторинг, резервное копирование
Инженер по внедрению	Подключение объектов, настройка WireGuard, разметка зон
Оператор поддержки	Диагностика инцидентов, взаимодействие с клиентами

Минимальный состав для обеспечения поддержки системы: **2 специалиста** (разработчик + инженер по внедрению).

11.3. Размещение персонала

Фактический адрес размещения разработчиков: по адресу нахождения ООО «Линза Метрикс», хотя в компании широко используется удаленный формат работы (с территории Российской Федерации, в основном, г. Санкт-Петербург).

Фактический адрес размещения службы поддержки: совпадает с разработчиками, для отдельных случаев могут привлекаться подрядчики для выездов на объекты установки системы для настройки соединения с сервером компании, а также специалисты по разметки данных в удаленном формате (с территории Российской Федерации, в основном, г. Симферополь).

11.4. Размещение инфраструктуры разработки

Хранение оригинала ПО: локальное хранилище по адресу нахождения ООО «Линза Метрикс».

Серверная инфраструктура (production, staging): Яндекс Облако (Yandex Cloud), дата-центры на территории Российской Федерации.

GitLab CE (репозиторий, CI/CD): self-hosted, Яндекс Облако.

Yandex Container Registry (реестр Docker-образов): Яндекс Облако.

Sentry (трекинг ошибок): self-hosted, Яндекс Облако.

11.5. Контакты службы поддержки

Телефон / мессенджеры: +7 913 203-32-20

Email: a.serdyukov@linzametrics.com

11.6. Обучение и квалификация персонала

Обучение новых сотрудников проводится в формате наставничества: новый участник команды работает совместно с опытным разработчиком или инженером на протяжении первых задач. Основные источники знаний:

- Документация системы (перечислена в разделе «Документирование»).
- Исходный код и комментарии в репозитории.
- История задач и обсуждений в GitLab Issues.
- Внутренние встречи команды.

Для инженеров по внедрению предусмотрено практическое обучение на тестовом стенде: полный цикл подключения объекта от настройки VPN до конфигурации дашбордов.

Квалификация разработчиков поддерживается через рецензирование кода (Merge Request), обсуждение архитектурных решений на встречах команды и самостоятельное изучение технологий стека (Django, Celery, Triton, React).