ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

ООО «КОРТЕКС»

Руководство по развертыванию программного обеспечения платформы анализа данных «Кортекс Аналитика»

г. Краснодар, 2026

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

1. Введение

Настоящее руководство описывает порядок установки, первичной настройки, проверки, резервного копирования и обновления программного обеспечения платформы анализа данных «Кортекс Аналитика» (далее — «Кортекс Аналитика», система) на сервере под управлением российской операционной системы ALT Linux, Astra Linux или РЕД ОС.

«Кортекс Аналитика» — клиент-серверное веб-приложение. Пользователи работают с системой через веб-браузер («тонкий клиент»), установка программ на рабочие места не требуется. Серверная часть состоит из следующих сервисов, каждый из которых выполняется в отдельном контейнере Docker:

- **react-front** — веб-сервер nginx: раздаёт клиентское приложение (SPA) и передаёт запросы программного интерфейса (`/api/`) серверному ядру;
- **go-core** — серверное ядро (HTTP API): аутентификация, подключения к внешним БД, чтение структуры БД, преобразование графа пайплайна в SQL-запрос, формирование снимков данных, оркестрация задач, планировщик расписаний, очистка устаревших промежуточных данных, формирование интеллектуальных выводов;
- **python-worker**, **python-worker-heavy**, **python-worker-writeback** — пулы обработчиков: основной (преобразования и методы анализа), ресурсоёмкий (методы машинного обучения) и пул записи результатов во внешние БД (единственный компонент, которому разрешена запись во внешнюю БД);
- **rabbitmq** — брокер сообщений RabbitMQ: очереди задач и событий между ядром и обработчиками;
- **postgres-meta** — встроенная БД метаданных (PostgreSQL 16). Сервис необязательный: вместо него может использоваться внешний сервер PostgreSQL или Postgres Pro (раздел 5.8).

«Кортекс Аналитика» полностью совместима с российскими операционными системами Astra Linux, ALT Linux (Альт) и РЕД ОС, а также с СУБД PostgreSQL и Postgres Pro — как для БД метаданных системы, так и для внешних источников данных и целевых БД записи результатов. Основной способ развертывания — контейнеры Docker: все сервисы системы на одном сервере под управлением Docker Compose (разделы 4–5). Поддерживается также развертывание под управлением российской платформы оркестрации контейнеров Deckhouse (Deckhouse Kubernetes Platform) — раздел 5.9.

Рекомендуемый способ развертывания на одном сервере — Docker Compose v2 по единому файлу `docker-compose.yml`, приведённому в разделе 5. Образы сервисов `analitic-core`, `analitic-worker`, `analitic-front` поставляются правообладателем — ООО «Кортекс» — через приватный реестр образов либо в виде архива (раздел 5.2).

Технические имена сервисов, образов (`analitic-core`, `analitic-worker`, `analitic-front`), имя БД метаданных `analitic_meta` и переменные окружения `ANALITIC_*` являются служебными идентификаторами компонентов системы и не изменяются.

В примерах команд и файлов конфигурации значения в угловых скобках (например, `<адрес_реестра_образов>`, `<версия>`, `<доменное_имя>`) заменяются фактическими значениями, используемыми при развертывании; адрес реестра образов и номер версии сообщает правообладатель.

2. Состав системы

Сервис	Компоненты	Порты	Тома / база данных
react-front	Веб-сервер nginx, клиентское приложение (HTML, JavaScript, CSS); образ `analitic-front`. Конфигурация nginx формируется при запуске контейнера из шаблона; адрес ядра задаётся переменной `CORE_UPSTREAM` (по умолчанию `go-core:8080`)	80 внутри контейнера; публикуется на хосте на порт `PUBLIC_PORT` (по умолчанию 80) — единственный публикуемый порт системы	Без томов
go-core	Серверное ядро (исполняемые файлы `/app/api` и `/app/migrate`, миграции БД встроены в исполняемый файл); образ `analitic-core`	8080 только внутри сети Docker Compose	Тома `snapshots` → `/data/snapshots`, `derived` → `/data/derived`, `trained_models` → `/data/models` (чтение и запись); БД `analitic_meta` (чтение и запись); внешние БД — только чтение
python-worker (N экземпляров, `WORKER_TIER=light`)	Обработчик преобразований и методов анализа (Python: pandas, DuckDB, statsmodels, scikit-learn, statsforecast); образ `analitic-worker`	Нет входящих портов	`snapshots` (только чтение), `derived`, `trained_models`; БД `analitic_meta`
python-worker-heavy (N экземпляров, `WORKER_TIER=heav`	Обработчик ресурсоёмких методов машинного обучения	Нет входящих портов	`snapshots` (только чтение), `derived`, `trained_models`;

Сервис	Компоненты	Порты	Тома / база данных
y)	(классификация, градиентный бустинг, автоматический подбор моделей, иерархическая кластеризация, перестановочная важность признаков); образ `analitic-worker`		БД `analitic_meta`
python-worker-writeback (1 экземпляр, `WORKER_TIER=writeback`)	Обработчик записи результатов во внешние БД PostgreSQL / Postgres Pro; образ `analitic-worker`	Нет входящих портов	`snapshots`, `derived`, `trained_models` — все только чтение; БД `analitic_meta`
rabbitmq	Брокер сообщений RabbitMQ 3 с панелью управления	5672 (AMQP) и 15672 (API управления) только внутри сети Docker Compose	Том `rabbit_data` → `/var/lib/rabbitmq`
postgres-meta (необязательный)	СУБД PostgreSQL 16	5432 только внутри сети Docker Compose	Том `meta_pgdata` → `/var/lib/postgresql/data`; БД метаданных `analitic_meta`

Взаимодействие компонентов:

1. Браузер пользователя обращается к внешнему веб-серверу nginx по протоколу HTTPS (порт 443, раздел 7).
2. Внешний nginx передаёт запросы на опубликованный порт сервиса `react-front` (`PUBLIC_PORT`).
3. `react-front` отдаёт файлы клиентского приложения, а запросы с путём `/api/` передаёт сервису `go-core` (порт 8080) по внутренней сети Docker Compose.
4. `go-core` хранит метаданные (пользователи, проекты, подключения, пайплайны, прогоны, задачи, результаты, модели, дашборды, расписания, алерты, журнал событий) в БД `analitic_meta`, читает данные из внешних БД в транзакциях «только чтение» и сохраняет снимки данных в формате Parquet в том `snapshots`.
5. `go-core` публикует задачи в очереди RabbitMQ; обработчики выполняют преобразования и анализ, записывают производные данные (том `derived`), обученные модели (том `trained_models`) и результаты (БД `analitic_meta`) и сообщают о завершении через очередь событий.

6. При включённых интеллектуальных выводах `go-core` выполняет исходящее подключение к брокеру ИИ «Комрад» (раздел 5.7).

Структура БД `analitic_meta` создаётся и обновляется встроенными миграциями; применённые миграции фиксируются в таблице `goose_db_version`. Миграции запускаются отдельной командой (раздел 5.3).

3. Требования к серверу

3.1. Программное обеспечение

- Операционная система (64-разрядная, x86_64) — одна из российских ОС: ALT Linux (Альт Сервер 10 и новее), Astra Linux Special Edition (1.7 и новее), РЕД ОС (7.3 и новее).
- Docker Engine версии 24 и новее.
- Docker Compose версии 2 (команда `docker compose`).
- Утилита `openssl` — для получения случайных секретов.
- Веб-сервер nginx на хосте — для публикации системы по протоколу HTTPS (раздел 7).
- СУБД: PostgreSQL 16 или Postgres Pro (Standard / Enterprise) 16 и новее. Встроенная БД метаданных поставляется в составе системы (PostgreSQL 16). Для варианта с внешней БД метаданных используется сервер PostgreSQL или Postgres Pro с расширением `pgcrypto` (входит в стандартную поставку). Postgres Pro поддерживается также как внешний источник данных и как целевая БД записи результатов.
- Альтернативная среда исполнения — платформа оркестрации контейнеров Deckhouse (Deckhouse Kubernetes Platform) с модулями ingress-nginx и cert-manager и хранилищем, поддерживающим режим доступа ReadWriteMany (раздел 5.9).

3.2. Аппаратные ресурсы

Ресурс	Минимально	Рекомендуется
Процессор	4 ядра x86_64	8 ядер и более
Оперативная память	8 ГБ	16 ГБ и более
Дисковое пространство	50 ГБ	100 ГБ и более (SSD)

Дисковое пространство расходуется на образы контейнеров (около 5 ГБ), БД метаданных и тома снимков, производных данных и моделей. Объём снимков определяется объёмом анализируемых выборок: одна выборка из внешней БД ограничена 1 млн строк, загружаемый файл — 100 МиБ (`MAX_UPLOAD_BYTES`). Производные данные прогонов старше `DERIVED_RETENTION_RUNS` последних прогонов пайплайна удаляются автоматически.

Потребление памяти и процессора определяется числом обработчиков (`WORKER_REPLICAS`, `HEAVY_WORKER_REPLICAS`): каждый обработчик одновременно

выполняет одну задачу (`WORKER_PREFETCH=1`). Ресурсоёмкие методы машинного обучения выполняются в отдельном пуле и не задерживают выполнение остальных методов.

3.3. Сетевые требования

- Входящие подключения пользователей: TCP 443 (HTTPS) и TCP 80 (перенаправление на HTTPS).
- Исходящий доступ с сервера к приватному реестру образов правообладателя — на время загрузки и обновления образов (при поставке архивом не требуется).
- Исходящий доступ с сервера к подключаемым внешним БД (PostgreSQL / Postgres Pro — как правило TCP 5432, MySQL / MariaDB — TCP 3306, ClickHouse — TCP 9000) и к целевым БД записи.
- Исходящий доступ HTTPS к API брокера ИИ «Комрад» — необязательно, только при использовании интеллектуальных выводов с применением искусственного интеллекта.
- Порты 8080 (ядро), 5672 / 15672 (брокер сообщений) и 5432 (встроенная БД) извне не публикуются.

4. Подготовка сервера

Все команды раздела выполняются от имени пользователя root (или через `sudo`).

4.1. Установка Docker в ALT Linux

```
# ============================================
# 1. Установить Docker Engine и Docker Compose v2
#    из репозитория ALT Linux (ветка p10)
# ============================================
apt-get update
apt-get install -y docker-engine docker-compose-v2

# ============================================
# 2. Включить автозапуск и запустить службу Docker
# ============================================
systemctl enable --now docker
```

4.2. Установка Docker в Astra Linux

```
# ============================================
# 1. Установить Docker Engine и Docker Compose v2
#    (Astra Linux SE 1.8; имя пакета Compose зависит от версии
#     репозитория: docker-compose-v2 или docker-compose-plugin)
# ============================================
apt update
apt install -y docker.io docker-compose-v2

# ============================================
# 2. Включить автозапуск и запустить службу Docker
# ============================================
systemctl enable --now docker
```

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

Если версия пакета `docker.io` в подключённом репозитории ниже 24 (например, в базовом репозитории Astra Linux SE 1.7), используйте Docker Engine из расширенного репозитория Astra Linux либо пакеты `docker-ce` и `docker-compose-plugin`.

4.3. Установка Docker в РЕД ОС

```
# ============================================
# 1. Установить Docker Engine и Docker Compose v2
#    из репозитория РЕД ОС
# ============================================
dnf install -y docker-ce docker-ce-cli docker-compose

# ============================================
# 2. Включить автозапуск и запустить службу Docker
# ============================================
systemctl enable --now docker
```

4.4. Проверка установки и создание каталога

```
# ============================================
# 1. Проверить версии Docker и Docker Compose
#    (Docker Engine 24+, Docker Compose 2.x)
# ============================================
docker version
docker compose version

# ============================================
# 2. (Необязательно) разрешить работу с Docker
#    пользователю развертывания без sudo
# ============================================
usermod -aG docker <пользователь>

# ============================================
# 3. Создать каталог развертывания
# ============================================
mkdir -p /opt/kortex-analitika
chmod 750 /opt/kortex-analitika
cd /opt/kortex-analitika
```

Если команда `docker compose version` не выполняется, установите пакет Compose v2 из репозитория используемой ОС (`docker-compose-v2` или `docker-compose-plugin`).

В каталоге `/opt/kortex-analitika` размещаются два файла: `docker-compose.yml` (описание сервисов) и `.env` (параметры и секреты). Имя проекта Docker Compose задаётся в файле `docker-compose.yml` (`name: kortex-analitika`), поэтому имена томов имеют вид `kortex-analitika_meta_pgdata`, `kortex-analitika_snapshots` и т. д. независимо от имени каталога.

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

5. Развертывание

5.1. Шаг 1. Создать файлы docker-compose.yml и .env

Файл docker-compose.yml

Создайте файл `/opt/kortex-analitika/docker-compose.yml` следующего содержания. Адрес реестра образов и номер версии в файл не вписываются — они задаются переменными `IMAGE_REGISTRY` и `IMAGE_TAG` в файле `.env`.

```
# ============================================
# Платформа анализа данных «Кортекс Аналитика»
# /opt/kortex-analitika/docker-compose.yml
# Единственный публикуемый порт — react-front (PUBLIC_PORT).
# ============================================
name: kortex-analitika

# --------------------------------------------
# Общие параметры: строка подключения к БД метаданных
# и к брокеру сообщений
# --------------------------------------------
x-meta-dsn: &meta-dsn postgres://${META_DB_USER}:${META_DB_PASSWORD}@$
{META_DB_HOST:-postgres-meta}:${META_DB_PORT:-5432}/${META_DB_NAME}?sslmode=$
{META_DB_SSLMODE:-disable}
x-rabbit-url: &rabbit-url amqp://${RABBITMQ_DEFAULT_USER}:$
{RABBITMQ_DEFAULT_PASS}@rabbitmq:5672/

# Параметры, общие для всех обработчиков
x-worker-env: &worker-env
  META_DB_DSN: *meta-dsn
  RABBITMQ_URL: *rabbit-url
  SNAPSHOT_DIR: /data/snapshots
  DERIVED_DIR: /data/derived
  MODELS_DIR: /data/models
  WORKER_PREFETCH: ${WORKER_PREFETCH:-1}

# Ядро и обработчики запускаются после готовности БД и брокера
x-deps: &deps
  postgres-meta:
    condition: service_healthy
  rabbitmq:
    condition: service_healthy

services:
  # --------------------------------------------
  # Встроенная БД метаданных (PostgreSQL 16).
  # Порт на хосте не публикуется.
  # --------------------------------------------
  postgres-meta:
    image: postgres:16
    environment:
      POSTGRES_USER: ${META_DB_USER}
      POSTGRES_PASSWORD: ${META_DB_PASSWORD}
      POSTGRES_DB: ${META_DB_NAME}
    volumes:
      - meta_pgdata:/var/lib/postgresql/data
    healthcheck:
```

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

```
      test: ["CMD-SHELL", "pg_isready -U ${META_DB_USER} -d ${META_DB_NAME}"]
      interval: 5s
      timeout: 5s
      retries: 20
    restart: unless-stopped
    networks: [backend]

  # ------------------------------------------
  # Брокер сообщений RabbitMQ (очереди задач и событий).
  # Порты 5672/15672 доступны только внутри сети Compose.
  # ------------------------------------------
  rabbitmq:
    image: rabbitmq:3-management
    environment:
      RABBITMQ_DEFAULT_USER: ${RABBITMQ_DEFAULT_USER}
      RABBITMQ_DEFAULT_PASS: ${RABBITMQ_DEFAULT_PASS}
    volumes:
      - rabbit_data:/var/lib/rabbitmq
    healthcheck:
      test: ["CMD", "rabbitmq-diagnostics", "-q", "ping"]
      interval: 15s
      timeout: 5s
      retries: 10
    restart: unless-stopped
    networks: [backend]

  # ------------------------------------------
  # Серверное ядро (HTTP API), порт 8080 внутри сети.
  # ------------------------------------------
  go-core:
    image: ${IMAGE_REGISTRY}/analitic-core:${IMAGE_TAG}
    depends_on: *deps
    environment:
      CORE_HTTP_ADDR: ":8080"
      META_DB_DSN: *meta-dsn
      RABBITMQ_URL: *rabbit-url
      ANALITIC_MASTER_KEY: ${ANALITIC_MASTER_KEY}
      JWT_SECRET: ${JWT_SECRET}
      SNAPSHOT_DIR: /data/snapshots
      DERIVED_DIR: /data/derived
      MODELS_DIR: /data/models
      MAX_UPLOAD_BYTES: ${MAX_UPLOAD_BYTES:-104857600}
      DERIVED_RETENTION_RUNS: ${DERIVED_RETENTION_RUNS:-10}
      ADMIN_EMAILS: ${ADMIN_EMAILS:-}
      REGISTRATION_OPEN: ${REGISTRATION_OPEN:-true}
      ANALITIC_LLM_URL: ${ANALITIC_LLM_URL:-}
      ANALITIC_LLM_API_KEY: ${ANALITIC_LLM_API_KEY:-}
      ANALITIC_LLM_MODEL: ${ANALITIC_LLM_MODEL:-}
    volumes:
      - snapshots:/data/snapshots
      - derived:/data/derived
      - trained_models:/data/models
    restart: unless-stopped
    networks: [backend]

  # ------------------------------------------
  # Основной пул обработчиков: преобразования и методы анализа
  # ------------------------------------------
```

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

```
  python-worker:
    image: ${IMAGE_REGISTRY}/analitic-worker:${IMAGE_TAG}
    depends_on: *deps
    environment:
      <<: *worker-env
      WORKER_TIER: light
    volumes:
      - snapshots:/data/snapshots:ro
      - derived:/data/derived
      - trained_models:/data/models
    deploy:
      replicas: ${WORKER_REPLICAS:-2}
    restart: unless-stopped
    networks: [backend]

  # ------------------------------------------
  # Пул ресурсоёмких методов машинного обучения
  # ------------------------------------------
  python-worker-heavy:
    image: ${IMAGE_REGISTRY}/analitic-worker:${IMAGE_TAG}
    depends_on: *deps
    environment:
      <<: *worker-env
      WORKER_TIER: heavy
    volumes:
      - snapshots:/data/snapshots:ro
      - derived:/data/derived
      - trained_models:/data/models
    deploy:
      replicas: ${HEAVY_WORKER_REPLICAS:-1}
    restart: unless-stopped
    networks: [backend]

  # ------------------------------------------
  # Пул записи во внешние БД — единственный компонент
  # с правом записи во внешнюю БД; тома только для чтения.
  # ------------------------------------------
  python-worker-writeback:
    image: ${IMAGE_REGISTRY}/analitic-worker:${IMAGE_TAG}
    depends_on: *deps
    environment:
      <<: *worker-env
      WORKER_TIER: writeback
    volumes:
      - snapshots:/data/snapshots:ro
      - derived:/data/derived:ro
      - trained_models:/data/models:ro
    deploy:
      replicas: ${WRITEBACK_WORKER_REPLICAS:-1}
    restart: unless-stopped
    networks: [backend]

  # ------------------------------------------
  # Веб-сервер nginx: клиентское приложение + прокси /api/.
  # Единственный сервис, публикуемый на хосте.
  # ------------------------------------------
  react-front:
    image: ${IMAGE_REGISTRY}/analitic-front:${IMAGE_TAG}
```

```
    depends_on: [go-core]
    ports:
      - "${PUBLIC_PORT:-80}:80"
    restart: unless-stopped
    networks: [backend]

# ------------------------------------------
# Тома сохраняются при «docker compose down»,
# удаляются только командой «docker compose down -v».
# ------------------------------------------
volumes:
  meta_pgdata:
  rabbit_data:
  snapshots:
  derived:
  trained_models:

networks:
  backend:
```

Файл .env

Создайте файл `/opt/kortex-analitika/.env`. Файл содержит пароли и секретные ключи, поэтому доступ к нему должен быть ограничен.

Предварительно получите случайные значения секретов (каждое значение — отдельной командой):

```
# ==========================================
# Ключ шифрования паролей подключений ANALITIC_MASTER_KEY
# (ровно 32 байта в кодировке base64)
# ==========================================
openssl rand -base64 32

# ==========================================
# Ключ подписи токенов сеанса JWT_SECRET
# ==========================================
openssl rand -hex 32

# ==========================================
# Пароли БД метаданных и брокера сообщений
# (шестнадцатеричные — без символов, требующих
# экранирования в строках подключения)
# ==========================================
openssl rand -hex 32
openssl rand -hex 32
```

Содержимое файла `.env`:

```
# ==========================================
# Образы системы (значения сообщает правообладатель)
# ==========================================
# Адрес приватного реестра образов правообладателя
# (при поставке архивом — префикс имён образов в архиве)
IMAGE_REGISTRY=<адрес_реестра_образов>
# Номер поставляемой версии
```

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

```
IMAGE_TAG=<версия>

# ==========================================
# Публикация веб-интерфейса на хосте
# ==========================================
PUBLIC_PORT=80

# ==========================================
# БД метаданных — ЗАМЕНИТЕ ПАРОЛЬ НА СВОЙ
# ==========================================
META_DB_USER=kortex
META_DB_PASSWORD=замените_на_свой_пароль
META_DB_NAME=analitic_meta
# Только для внешней БД (раздел 5.8):
# META_DB_HOST=<адрес_сервера_БД>
# META_DB_PORT=5432
# META_DB_SSLMODE=require

# ==========================================
# Брокер сообщений RabbitMQ — ЗАМЕНИТЕ ПАРОЛЬ НА СВОЙ
# ==========================================
RABBITMQ_DEFAULT_USER=kortex
RABBITMQ_DEFAULT_PASS=замените_на_свой_пароль

# ==========================================
# Секреты — ОБЯЗАТЕЛЬНО ЗАМЕНИТЕ
# ==========================================
# Ключ шифрования паролей внешних БД (openssl rand -base64 32).
# После создания подключений НЕ ИЗМЕНЯТЬ.
ANALITIC_MASTER_KEY=замените_на_свой_ключ
# Ключ подписи токенов сеанса (openssl rand -hex 32)
JWT_SECRET=замените_на_свой_секретный_ключ

# ==========================================
# Учётные записи
# ==========================================
# Адреса электронной почты (через запятую), которым при входе
# автоматически назначается роль «Администратор»
ADMIN_EMAILS=admin@example.ru
# Самостоятельная регистрация (true/false). При false
# зарегистрироваться может только первый пользователь.
REGISTRATION_OPEN=true

# ==========================================
# Масштабирование обработчиков
# ==========================================
WORKER_REPLICAS=2
HEAVY_WORKER_REPLICAS=1
WRITEBACK_WORKER_REPLICAS=1

# ==========================================
# Хранение данных
# ==========================================
# Хранить производные данные N последних прогонов
# каждого пайплайна (0 — не удалять)
DERIVED_RETENTION_RUNS=10
# Предельный размер загружаемого файла, байт (100 МиБ)
MAX_UPLOAD_BYTES=104857600
```

```
# ==========================================
# Интеллектуальные выводы через брокер ИИ «Комрад»
# (необязательно; пустые значения — выводы по встроенным правилам)
# ==========================================
ANALITIC_LLM_URL=
ANALITIC_LLM_API_KEY=
ANALITIC_LLM_MODEL=
```

Ограничьте доступ к файлу:

```
chmod 600 /opt/kortex-analitika/.env
```

Назначение переменных:

Переменная	Обязательна	Назначение
IMAGE_REGISTRY, IMAGE_TAG	да	Адрес реестра образов (префикс имён образов) и номер версии
PUBLIC_PORT	нет	Порт хоста для сервиса react-front (по умолчанию 80)
META_DB_USER, META_DB_PASSWORD, META_DB_NAME	да	Учётная запись и имя БД метаданных; для встроенной БД создаются при первой инициализации тома
META_DB_HOST, META_DB_PORT, META_DB_SSLMODE	только для внешней БД	Адрес, порт и режим SSL внешнего сервера БД метаданных (по умолчанию `postgres-meta`, `5432`, `disable`)
RABBITMQ_DEFAULT_USER, RABBITMQ_DEFAULT_PASS	да	Учётная запись брокера сообщений; создаётся при первой инициализации тома `rabbit_data`
ANALITIC_MASTER_KEY	да	Ключ шифрования (AES-256-GCM) паролей подключений к внешним БД и целевым БД записи; 32 байта в кодировке base64
JWT_SECRET	да	Секретный ключ подписи токенов сеанса пользователей
ADMIN_EMAILS	нет	Адреса электронной почты, которым при входе назначается роль «Администратор»
REGISTRATION_OPEN	нет	Разрешение самостоятельной регистрации (по умолчанию `true`)
WORKER_REPLICAS, HEAVY_WORKER_REPLICA	нет	Число экземпляров основного, ресурсоёмкого пула и пула

Переменная	Обязательна	Назначение
S, WRITEBACK_WORKER_REPLICAS		записи (по умолчанию 2, 1, 1)
WORKER_PREFETCH	нет	Число задач, одновременно выдаваемых одному обработчику (по умолчанию 1)
DERIVED_RETENTION_RUNS	нет	Число последних прогонов пайплайна, производные данные которых хранятся (по умолчанию 10; 0 — очистка выключена)
MAX_UPLOAD_BYTES	нет	Предельный размер загружаемого файла CSV / XLSX, байт (по умолчанию 104857600)
ANALITIC_LLM_URL, ANALITIC_LLM_API_KEY, ANALITIC_LLM_MODEL	нет	Адрес API брокера ИИ «Комрад», ключ доступа и идентификатор модели (раздел 5.7)

Переменные `CORE_HTTP_ADDR`, `META_DB_DSN`, `RABBITMQ_URL`, `SNAPSHOT_DIR`, `DERIVED_DIR`, `MODELS_DIR` и `WORKER_TIER` формируются в `docker-compose.yml` и в `.env` не указываются. Адрес API управления брокером сообщений (`RABBITMQ_MGMT_URL`) по умолчанию вычисляется ядром из `RABBITMQ_URL` (порт 15672) и задаётся только при нестандартной конфигурации.

5.2. Шаг 2. Получить образы

Используется один из двух вариантов; способ получения сообщает правообладатель.

Вариант А — из реестра образов правообладателя

Учётную запись для доступа к реестру образов (только чтение) выдаёт правообладатель.

```
# ============================================
# 1. Авторизоваться в реестре образов правообладателя
# ============================================
docker login <адрес_реестра_образов>

# ============================================
# 2. Загрузить образы всех сервисов
# ============================================
cd /opt/kortex-analitika
docker compose pull
```

Вариант Б — из архива, поставляемого правообладателем

Архив `kortex-analitika-<версия>.tar.gz` содержит образы `analitic-core`, `analitic-worker`, `analitic-front`, а также образы СУБД PostgreSQL 16 (`postgres:16`) и брокера сообщений (`rabbitmq:3-management`). Доступ к внешним реестрам образов в этом варианте не требуется.

```
# ============================================
# 1. Загрузить образы из архива в локальное хранилище Docker
# ============================================
docker load -i kortex-analitika-<версия>.tar.gz

# ============================================
# 2. Проверить загруженные образы: префикс и тег должны
#    совпадать с IMAGE_REGISTRY и IMAGE_TAG в файле .env
# ============================================
docker images | grep -E "analitic-|postgres|rabbitmq"
```

В варианте Б команда `docker compose pull` не выполняется: при наличии образов в локальном хранилище Docker Compose их не загружает.

5.3. Шаг 3. Запустить систему

Запуск выполняется в три действия в строго указанном порядке:

```
cd /opt/kortex-analitika

# ============================================
# 1. Запустить БД метаданных и брокер сообщений
# ============================================
docker compose up -d postgres-meta rabbitmq

# ============================================
# 2. Применить миграции структуры БД метаданных
#    (дождётся готовности postgres-meta и rabbitmq)
# ============================================
docker compose run --rm --entrypoint /app/migrate go-core up

# ============================================
# 3. Запустить все остальные сервисы
# ============================================
docker compose up -d
```

Миграции обязательно применяются **до первого запуска обработчиков**. При старте каждый обработчик публикует в БД метаданных каталог методов анализа и преобразований (таблицы `analyzer_catalog` и `transform_catalog`), по которому ядро проверяет параметры узлов, а клиентское приложение строит палитру узлов и формы настройки. Если таблицы каталога ещё не созданы, публикация завершается ошибкой, обработчик продолжает работу без каталога, а палитра «Анализ» в редакторе пайплайнов остаётся пустой (устранение — раздел 6).

Команда миграции выводит перечень применённых миграций и завершается строкой вида `goose: successfully migrated database to version: <номер>`.

Повторный запуск безопасен: уже применённые миграции пропускаются. Текущее состояние выводит команда `docker compose run --rm --entrypoint /app/migrate go-core status`.

При первом запуске контейнер `postgres-meta` несколько секунд инициализирует БД, а `rabbitmq` — около 30 секунд. Ядро и обработчики запускаются только после того, как проверки готовности обоих сервисов (`healthcheck`) завершатся успешно.

5.4. Шаг 4. Проверить работоспособность

```
cd /opt/kortex-analitika

# ============================================
# 1. Состояние контейнеров: все — Up;
#    postgres-meta и rabbitmq — (healthy)
# ============================================
docker compose ps

# ============================================
# 2. Проверка работоспособности API ядра
# ============================================
curl -s http://localhost/api/v1/healthz

# ============================================
# 3. Журнал ядра
# ============================================
docker compose logs go-core --tail 20

# ============================================
# 4. Журнал обработчиков
# ============================================
docker compose logs python-worker python-worker-heavy python-worker-writeback --
tail 20
```

Ответ проверки работоспособности (HTTP 200):

```
{"service":"analitic-core","status":"ok"}
```

При использовании нестандартного порта `PUBLIC_PORT` укажите его в адресе: `http://localhost:<PUBLIC_PORT>/api/v1/healthz`.

Ядро пишет журнал в стандартный вывод в формате JSON (одна запись — одна строка). После успешного запуска журнал `go-core` содержит строки:

```
{"time":"…","level":"INFO","msg":"http server listening","addr":":8080"}
{"time":"…","level":"INFO","msg":"core consuming job events","queue":"q.events"}
```

Журнал каждого обработчика после успешного запуска содержит строки публикации каталога и подключения к очередям (число методов и преобразований соответствует поставляемой версии):

```
{"ts":"…","level":"INFO","msg":"published 40 analyzers to catalog"}
```

```
{"ts":"…","level":"INFO","msg":"published 15 transforms to catalog"}
{"ts":"…","level":"INFO","msg":"worker tier=light consuming q.analysis,
q.transform (prefetch=1)"}
```

Для пула `python-worker-heavy` последняя строка имеет вид `worker tier=heavy consuming q.analysis.heavy (prefetch=1)`, для пула `python-worker-writeback` — `worker tier=writeback consuming q.writeback (prefetch=1)`.

Откройте в браузере адрес системы (до настройки HTTPS — `http://<адрес_сервера>/`). Состояние компонентов (ядро, БД метаданных, брокер сообщений, пулы обработчиков, дисковое пространство тома снимков) администратор системы может проверить в веб-интерфейсе: «Администрирование» → «Диагностика».

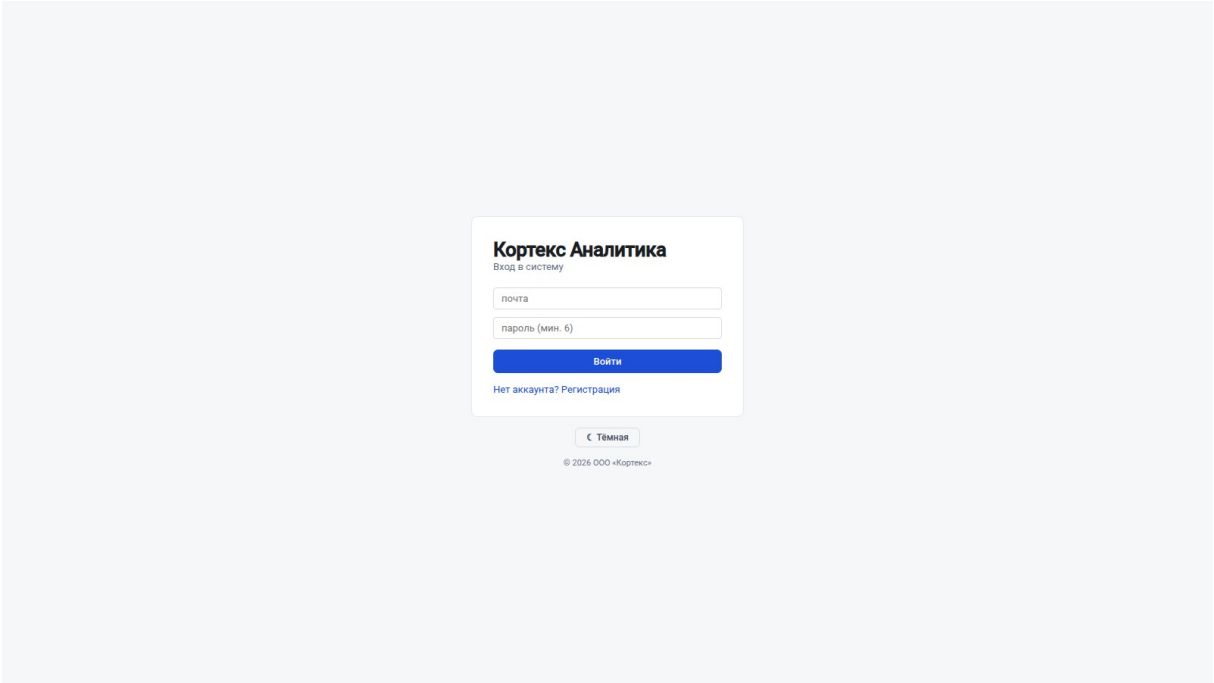


Рисунок 1 — Страница входа в систему

5.5. Шаг 5. Первичная настройка

Учётная запись администратора

Учётные записи создаются в веб-интерфейсе; логином служит адрес электронной почты, пароль — не короче 6 символов.

1. Откройте адрес системы и зарегистрируйте первого пользователя. **Первый зарегистрированный пользователь автоматически получает роль «Администратор».**
2. Пользователям, адреса которых перечислены в переменной `ADMIN_EMAILS`, роль «Администратор» назначается при регистрации и при каждом входе.
3. После создания учётных записей администраторов рекомендуется закрыть самостоятельную регистрацию: указать в `.env` значение

`REGISTRATION_OPEN=false` и пересоздать контейнер ядра командой `docker compose up -d go-core`. При закрытой регистрации попытка регистрации завершается сообщением «регистрация закрыта — обратитесь к администратору».

4. Остальные учётные записи создаёт администратор: «Администрирование» → «Пользователи» → «+ Добавить пользователя». В этом же разделе выполняются смена роли, блокировка, сброс пароля и удаление учётных записей.

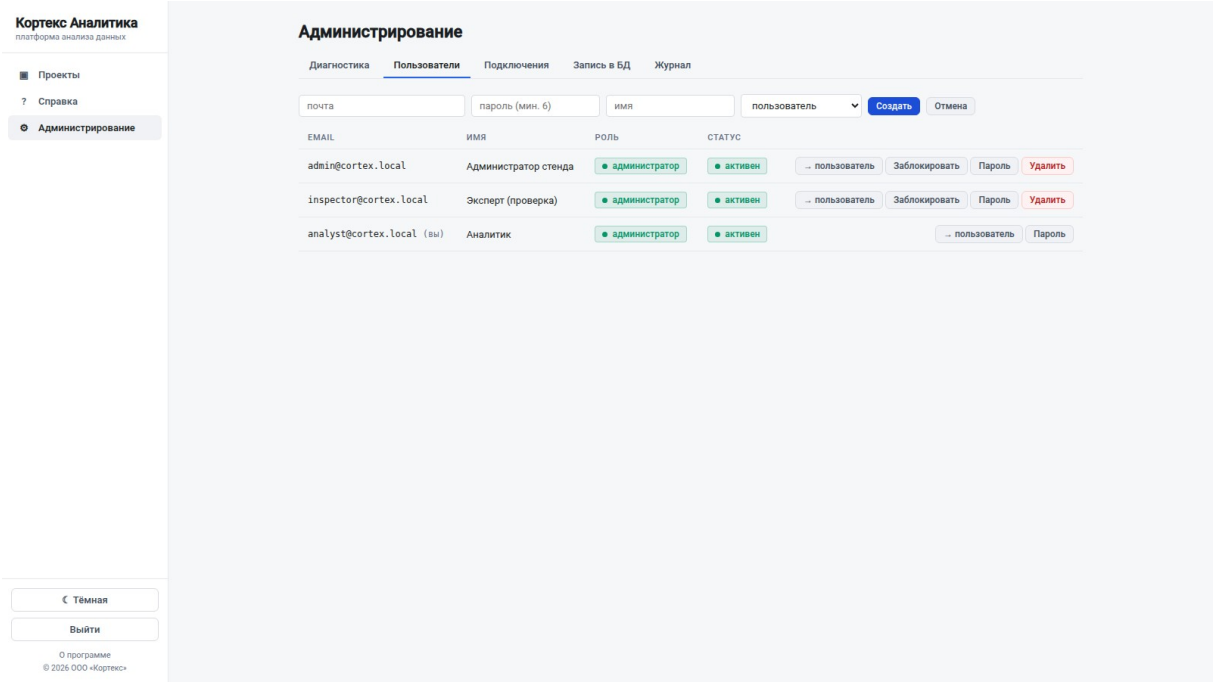


Рисунок 2 — Администрирование — Пользователи

Подключения к внешним БД

Для каждой внешней БД рекомендуется создать в СУБД-источнике отдельную роль с правом только на чтение. Пример для PostgreSQL / Postgres Pro (выполняет администратор источника):

```
-- ============================================
-- Роль только для чтения для «Кортекс Аналитика»
-- ============================================
CREATE ROLE kortex_reader LOGIN PASSWORD 'замените_на_свой_пароль';
ALTER ROLE kortex_reader SET default_transaction_read_only = on;

GRANT CONNECT ON DATABASE <база_источника> TO kortex_reader;
\c <база_источника>
GRANT USAGE ON SCHEMA public TO kortex_reader;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO kortex_reader;
ALTER DEFAULT PRIVILEGES IN SCHEMA public
  GRANT SELECT ON TABLES TO kortex_reader;
```

Регистрация подключения в системе:

1. Войдите под учётной записью администратора.
2. Откройте «Администрирование» → «Подключения» и добавьте подключение.

3. Выберите движок (PostgreSQL, MySQL / MariaDB, ClickHouse — порт по умолчанию подставляется автоматически), укажите название, хост, порт, имя БД, пользователя, пароль и режим SSL.
4. При необходимости ограничьте перечень доступных схем — пользователи смогут строить выборки только по таблицам выбранных схем.
5. Сохраните подключение и проверьте результат проверки: сообщение «подключено; роль только на чтение» подтверждает корректную настройку роли.

Независимо от прав роли система читает внешние БД только в транзакциях «только чтение» с ограничением времени выполнения запроса и числа строк. Пароль подключения хранится в БД метаданных в зашифрованном виде (ключ `ANALITIC_MASTER_KEY`) и пользователям не отображается.

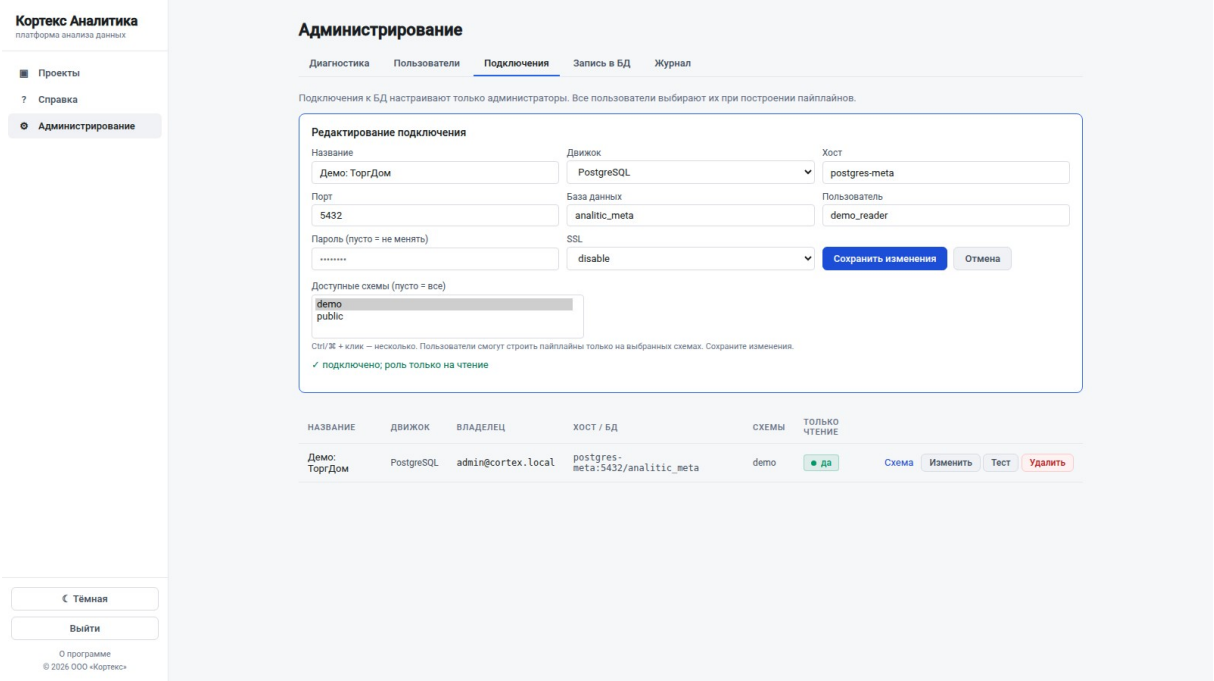


Рисунок 3 — Администрирование — Подключения

Целевые БД записи

Запись результатов во внешние БД (узел «Запись в БД») выполняется только в целевые БД, зарегистрированные администратором: «Администрирование» → «Запись в БД». Для целевой БД указываются параметры подключения PostgreSQL / Postgres Pro, перечень разрешённых схем (обязателен) и признак разрешения создания таблиц. Для целевой БД создаётся отдельная роль с правами на запись только в разрешённые схемы, например:

```
-- ============================================
-- Роль записи результатов для «Кортекс Аналитика»
-- ============================================
CREATE ROLE kortex_writer LOGIN PASSWORD 'замените_на_свой_пароль';
CREATE SCHEMA IF NOT EXISTS analytics_out;
GRANT USAGE, CREATE ON SCHEMA analytics_out TO kortex_writer;
```

5.6. Масштабирование обработчиков

Число экземпляров обработчиков задаётся переменными `WORKER_REPLICAS` (основной пул) и `HEAVY_WORKER_REPLICAS` (ресурсоёмкий пул). Обработчики получают задачи из общих очередей брокера сообщений по принципу конкурирующих потребителей, поэтому увеличение числа экземпляров повышает число одновременно выполняемых задач без дополнительной настройки. Рекомендуемое значение `WORKER_REPLICAS` — не больше числа ядер процессора; каждому экземпляру ресурсоёмкого пула следует предусмотреть не менее 2 ГБ оперативной памяти. Пул записи (`WRITEBACK_WORKER_REPLICAS`) рекомендуется оставлять в одном экземпляре.

```
# После изменения значений в .env
cd /opt/kortex-analitika
docker compose up -d
```

Число подключённых обработчиков основного, ресурсоёмкого пулов и пула записи во внешние БД отображается в разделе «Администрирование» → «Диагностика».

5.7. Настройка интеллектуальных выводов

Отчёт по прогону пайплайна содержит интеллектуальные выводы по результатам анализа. Выводы формируются с помощью искусственного интеллекта через российский брокер ИИ «Комрад» либо, если брокер не настроен или недоступен, по встроенным правилам. Источник выводов указывается в отчёте.

Для использования брокера ИИ заполните в `.env` три переменные (значения предоставляются по договору с оператором брокера ИИ «Комрад»):

```
# Адрес API брокера ИИ «Комрад» (базовый адрес до версии API включительно)
ANALITIC_LLM_URL=<адрес_API_брокера_ИИ>
# Ключ доступа к API
ANALITIC_LLM_API_KEY=<ключ_доступа>
# Идентификатор модели
ANALITIC_LLM_MODEL=<идентификатор_модели>
```

и пересоздайте контейнер ядра:

```
cd /opt/kortex-analitika
docker compose up -d go-core
```

Брокер ИИ используется только при заполнении всех трёх переменных; при пустом значении хотя бы одной из них выводы формируются по встроенным правилам. Для обращения к брокеру ядру требуется исходящий доступ HTTPS к адресу `ANALITIC_LLM_URL`; время ожидания ответа — 50 секунд.

Проверка: выполните прогон любого пайплайна с узлами анализа, откройте отчёт по прогону и нажмите «Сформировать выводы». Под текстом выводов отображается источник: «Выводы сформированы с помощью искусственного интеллекта (брокер ИИ «Комрад»). Проверяйте их по результатам анализа.» либо «Выводы сформированы по встроенным правилам.».

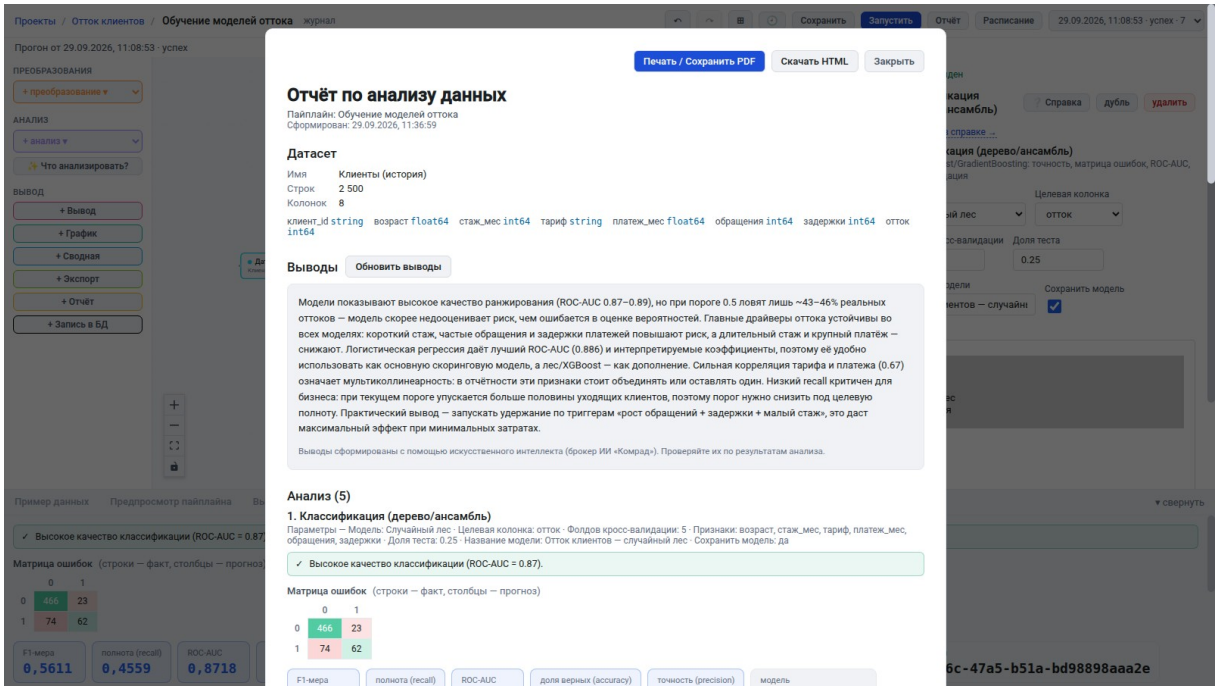


Рисунок 4 — Интеллектуальные выводы в отчёте

5.8. Вариант с внешней БД метаданных

Вместо встроенного сервиса `postgres-meta` может использоваться внешний сервер PostgreSQL 16 или Postgres Pro 16 и новее. Администратор внешнего сервера создаёт роль и БД (от имени суперпользователя):

```
-- ============================================
-- Роль и БД метаданных «Кортекс Аналитика»
-- ============================================
CREATE ROLE kortex LOGIN PASSWORD 'замените_на_свой_пароль';
CREATE DATABASE analitic_meta OWNER kortex;

-- ============================================
-- Расширение pgcrypto (создаётся и первой миграцией;
-- заранее — если у роли нет права создавать расширения)
-- ============================================
\c analitic_meta
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

Сервер СУБД должен принимать подключения с сервера развертывания (параметр `listen_addresses` и запись в `pg_hba.conf`).

Изменения в конфигурации развертывания:

1. В `.env` раскомментируйте и заполните `META_DB_HOST`, `META_DB_PORT`, `META_DB_SSLMODE` (для подключения с шифрованием — `require`); `META_DB_USER`, `META_DB_PASSWORD`, `META_DB_NAME` укажите по созданной роли и БД.
2. В `docker-compose.yml` удалите блок сервиса `postgres-meta`, том `meta_pgdata` и зависимость от `postgres-meta` в блоке `x-deps`:

```
x-deps: &deps
  rabbitmq:
    condition: service_healthy
```

3. Запуск выполняется по шагу 3 с той разницей, что первой командой запускается только брокер сообщений: `docker compose up -d rabbitmq`.

Резервное копирование и профилактика внешней БД выполняются средствами её администратора (утилиты `pg_dump`, `vacuumdb`).

5.9. Развертывание под управлением платформы Deckhouse

Система может эксплуатироваться в кластере под управлением российской платформы оркестрации контейнеров Deckhouse (Deckhouse Kubernetes Platform). Используются те же образы `analitic-core`, `analitic-worker`, `analitic-front`, что и при развертывании Docker Compose; образы размещаются в реестре образов, доступном кластеру. Ниже приведено соответствие сервисов объектам Kubernetes и примеры основных манифестов; конкретные значения (пространство имён, классы хранилища, ресурсы, доменное имя) согласуются при внедрении.

Соответствие сервисов объектам Kubernetes

Сервис	Объект Kubernetes	Примечание
react-front	Deployment + Service (порт 80)	Переменная `CORE_UPSTREAM` — полное доменное имя сервиса ядра: `go-core.<namespace>.svc.cluster.local:8080` (nginx не использует домены поиска из `/etc/resolv.conf`, поэтому краткое имя `go-core` не применяется)
go-core	Deployment + Service (порт 8080)	Тома `snapshots`, `derived`, `trained_models`; переменные окружения — из ConfigMap и Secret
python-worker, python-worker-heavy, python-worker-writeback	Deployment (по одному на пул), число реплик — `replicas`	`WORKER_TIER` = `light` / `heavy` / `writeback`; входящих портов нет, Service не требуется
rabbitmq	StatefulSet + Service (5672, 15672)	Постоянный том для `/var/lib/rabbitmq`
БД метаданных	Рекомендуется внешний сервер Postgres Pro или PostgreSQL (строка подключения `META_DB_DSN`); допускается StatefulSet PostgreSQL 16 в кластере	Подготовка внешней БД — раздел 5.8
Миграции БД	Job с командой	Выполняется до запуска

Сервис	Объект Kubernetes	Примечание
	`/app/migrate up` (образ `analitic-core`) либо initContainer пода ядра	обработчиков (раздел 5.3)
Публикация	Ingress (модуль ingress-nginx Deckhouse), TLS — модуль cert-manager Deckhouse	Требования к прокси — как в разделе 7

Тома и секреты

Тома `snapshots`, `derived`, `trained_models` оформляются как PersistentVolumeClaim с режимом доступа **ReadWriteMany**, так как ядро и обработчики используют их совместно и могут выполняться на разных узлах кластера. Для них используется хранилище, поддерживающее ReadWriteMany (подключается соответствующим модулем хранилища Deckhouse). Режимы монтирования повторяют конфигурацию Docker Compose: обработчики `light` и `heavy` монтируют `snapshots` только на чтение (`readOnly: true`), пул `writeback` — все три тома только на чтение.

Секретные значения (`ANALITIC_MASTER_KEY`, `JWT_SECRET`, `META_DB_DSN`, `RABBITMQ_URL`, пароли брокера сообщений, `ANALITIC_LLM_API_KEY` и при необходимости `ANALITIC_LLM_URL`, `ANALITIC_LLM_MODEL`) хранятся в объекте Secret; несекретные параметры (`ADMIN_EMAILS`, `REGISTRATION_OPEN`, `DERIVED_RETENTION_RUNS`, `MAX_UPLOAD_BYTES`, каталоги томов) — в ConfigMap или непосредственно в манифесте.

Ядро и обработчики выполняются от непривилегированного пользователя с UID 10001; для записи в тома в `securityContext` пода задаются `runAsUser: 10001`, `runAsNonRoot: true` и `fsGroup: 10001`.

Пример: Secret, Deployment и Service ядра

```
# ============================================
# Секреты системы (значения — заглушки)
# ============================================
apiVersion: v1
kind: Secret
metadata:
  name: kortex-analitika-secrets
  namespace: <пространство_имён>
type: Opaque
stringData:
  ANALITIC_MASTER_KEY: замените_на_свой_ключ
  JWT_SECRET: замените_на_свой_секретный_ключ
  META_DB_DSN:
postgres://kortex:замените_на_свой_пароль@<адрес_сервера_БД>:5432/analitic_meta?
sslmode=require
  RABBITMQ_URL: amqp://kortex:замените_на_свой_пароль@rabbitmq:5672/
  ANALITIC_LLM_URL: ""
  ANALITIC_LLM_API_KEY: ""
  ANALITIC_LLM_MODEL: ""
---
```

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

```
# ============================================
# Серверное ядро
# ============================================
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-core
  namespace: <пространство_имён>
spec:
  replicas: 1
  selector:
    matchLabels: {app: go-core}
  template:
    metadata:
      labels: {app: go-core}
    spec:
      securityContext:
        runAsUser: 10001
        runAsNonRoot: true
        fsGroup: 10001
      containers:
        - name: go-core
          image: <адрес_реестра_образов>/analitic-core:<версия>
          ports:
            - containerPort: 8080
          envFrom:
            - secretRef: {name: kortex-analitika-secrets}
          env:
            - {name: CORE_HTTP_ADDR, value: ":8080"}
            - {name: SNAPSHOT_DIR, value: /data/snapshots}
            - {name: DERIVED_DIR, value: /data/derived}
            - {name: MODELS_DIR, value: /data/models}
            - {name: ADMIN_EMAILS, value: "admin@example.ru"}
            - {name: REGISTRATION_OPEN, value: "false"}
          readinessProbe:
            httpGet: {path: /api/v1/healthz, port: 8080}
            periodSeconds: 10
          livenessProbe:
            httpGet: {path: /api/v1/healthz, port: 8080}
            initialDelaySeconds: 15
            periodSeconds: 20
          volumeMounts:
            - {name: snapshots, mountPath: /data/snapshots}
            - {name: derived, mountPath: /data/derived}
            - {name: models, mountPath: /data/models}
      volumes:
        - name: snapshots
          persistentVolumeClaim: {claimName: snapshots}
        - name: derived
          persistentVolumeClaim: {claimName: derived}
        - name: models
          persistentVolumeClaim: {claimName: trained-models}
---
apiVersion: v1
kind: Service
metadata:
  name: go-core
  namespace: <пространство_имён>
```

```
spec:
  selector: {app: go-core}
  ports:
    - {port: 8080, targetPort: 8080}
```

Deployment обработчиков строятся аналогично: образ `analitic-worker`, переменные `META_DB_DSN` и `RABBITMQ_URL` из того же Secret, `WORKER_TIER`, `SNAPSHOT_DIR`, `DERIVED_DIR`, `MODELS_DIR`, число реплик — по разделу 5.6. Deployment фронта — образ `analitic-front`, переменная `CORE_UPSTREAM: go-core.<namespace>.svc.cluster.local:8080`, пробы `GET /` на порт 80.

Пример: Job миграций

```
# ============================================
# Применение миграций БД метаданных.
# Выполняется до запуска (обновления) обработчиков и ядра.
# ============================================
apiVersion: batch/v1
kind: Job
metadata:
  name: kortex-analitika-migrate-<версия>
  namespace: <пространство_имён>
spec:
  backoffLimit: 3
  template:
    spec:
      restartPolicy: Never
      securityContext:
        runAsUser: 10001
        runAsNonRoot: true
      containers:
        - name: migrate
          image: <адрес_реестра_образов>/analitic-core:<версия>
          command: ["/app/migrate", "up"]
          envFrom:
            - secretRef: {name: kortex-analitika-secrets}
```

Порядок первичного развертывания: Secret, PersistentVolumeClaim, брокер сообщений (и БД метаданных, если она размещается в кластере) → Job миграций (ожидание завершения: `kubectl -n <namespace> wait --for=condition=complete job/<имя> --timeout=300s`) → ядро, обработчики, фронт, Ingress. Требование о применении миграций до запуска обработчиков то же, что в разделе 5.3.

Пример: Ingress

```
# ============================================
# Публикация по HTTPS через модуль ingress-nginx;
# сертификат выпускается модулем cert-manager
# ============================================
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: kortex-analitika
  namespace: <пространство_имён>
  annotations:
    nginx.ingress.kubernetes.io/proxy-body-size: "120m"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "300"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "300"
    nginx.ingress.kubernetes.io/proxy-buffering: "off"
    cert-manager.io/cluster-issuer: <имя_ClusterIssuer>
spec:
  ingressClassName: nginx
  tls:
    - hosts: [<доменное_имя>]
      secretName: kortex-analitika-tls
  rules:
    - host: <доменное_имя>
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: react-front
                port: {number: 80}
```

Все запросы (клиентское приложение и `/api/`) направляются сервису фронта, который сам проксирует `/api/` на ядро. Ограничения методов HTTP на Ingress не задаются: система использует методы GET, POST, PUT, PATCH и DELETE.

Проверка: `kubectl -n <namespace> get pods` (все поды — Running, Job миграций — Completed), `kubectl -n <namespace> logs deploy/go-core` (строка `http server listening`), `curl -s https://<доменное_имя>/api/v1/healthz` (ответ `{"service":"analitic-core","status":"ok"}`). Резервное копирование БД метаданных выполняется по разделу 8 средствами используемого сервера PostgreSQL / Postgres Pro, томов — средствами хранилища кластера.

6. Диагностика

В случае проблем выполните:

```
cd /opt/kortex-analitika

# ============================================
# Состояние контейнеров
# ============================================
docker compose ps

# ============================================
# Журналы сервисов (последние 100 строк)
# ============================================
docker compose logs go-core --tail 100
docker compose logs python-worker --tail 100
docker compose logs python-worker-heavy --tail 100
docker compose logs python-worker-writeback --tail 100
docker compose logs react-front --tail 100
docker compose logs rabbitmq --tail 100
```

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

```
docker compose logs postgres-meta --tail 100

# ============================================
# Журнал ядра в реальном времени
# ============================================
docker compose logs -f go-core

# ============================================
# Потребление ресурсов контейнерами
# ============================================
docker stats --no-stream

# ============================================
# Проверка API ядра
# ============================================
curl -s http://localhost/api/v1/healthz
```

Наиболее частые проблемы:

Симптом	Причина	Решение
Палитра «Анализ» в редакторе пуста; при запуске пайплайна — сообщение «неизвестный анализатор …»; в журнале обработчика `"msg":"catalog publish failed (continuing)"` и `relation "analyzer_catalog" does not exist`	Обработчики запущены до применения миграций, каталог методов не опубликован	Применить миграции (`docker compose run --rm --entrypoint /app/migrate go-core up`) и перезапустить обработчики: `docker compose restart python-worker python-worker-heavy python-worker-writeback`
При входе сообщение «неверные логин или пароль»	Неверный адрес электронной почты или пароль	Проверить раскладку и адрес; администратор может задать новый пароль: «Администрирование» → «Пользователи» → «Сбросить пароль»
При входе сообщение «учётная запись заблокирована»	Учётная запись заблокирована администратором	Разблокировать в «Администрирование» → «Пользователи»
При регистрации сообщение «регистрация закрыта — обратитесь к администратору»	Задано `REGISTRATION_OPEN=false`, в системе уже есть пользователи	Учётную запись создаёт администратор в «Администрирование» → «Пользователи»
Контейнер go-core постоянно перезапускается, в журнале `"msg":"config load failed","err":"META_DB_DSN is required"`	Не сформирована строка подключения к БД метаданных	Проверить переменные `META_DB_*` в `.env` и выполнить `docker compose up -d --force-recreate go-core`
go-core перезапускается, в журнале `"msg":"crypto init failed"` с текстом	Не задан или неверной длины ключ `ANALITIC_MASTER_KEY`	Получить ключ командой `openssl rand -base64 32`, указать в `.env`,

Симптом	Причина	Решение
`master key is empty` или `master key must be 32 bytes, got N`		пересоздать контейнер go-core
go-core перезапускается, в журнале `"msg":"store init failed"` (`connection refused`, `password authentication failed`)	БД метаданных недоступна либо пароль не совпадает с паролем роли	Проверить состояние postgres-meta; сверить `META_DB_PASSWORD` с паролем роли БД. Пароль встроенной БД задаётся только при первой инициализации тома — для смены выполнить `ALTER ROLE` в БД
go-core перезапускается, в журнале `"msg":"rabbitmq publisher init failed"`; обработчики завершаются с ошибкой подключения к брокеру	Брокер сообщений недоступен или неверны `RABBITMQ_DEFAULT_USER` / `RABBITMQ_DEFAULT_PASS`	Проверить `docker compose ps rabbitmq` и журнал rabbitmq. Учётная запись брокера создаётся только при первой инициализации тома `rabbit_data` — после смены значений в `.env` изменить пароль командой `docker compose exec rabbitmq rabbitmqctl change_password <пользователь> <пароль>`
В разделе «Диагностика» брокер сообщений или пулы обработчиков в состоянии «недоступен» / «неизвестно»	Нет доступа к API управления брокером (порт 15672) либо обработчики не запущены	Проверить журналы rabbitmq и обработчиков, `docker compose ps`; при нестандартной конфигурации брокера задать `RABBITMQ_MGMT_URL`
Прогон пайплайна не выполняется, узлы остаются в состоянии ожидания	Нет запущенных обработчиков нужного пула	Проверить `docker compose ps`, значения `WORKER_REPLICAS` и `HEAVY_WORKER_REPLICAS`, журналы обработчиков
При загрузке файла CSV / XLSX сообщение «файл слишком большой» (HTTP 413)	Размер файла превышает `client_max_body_size` внешнего nginx (по умолчанию 1 МБ) либо предел `MAX_UPLOAD_BYTES` (100 МиБ)	Задать во внешнем nginx `client_max_body_size 120m` (раздел 7); при необходимости увеличить `MAX_UPLOAD_BYTES`
Проверка подключения к внешней БД: «не удалось подключиться: …»	Недоступен хост или порт, неверные учётные данные, режим SSL или имя БД	Проверить сетевую доступность с сервера, `pg_hba.conf` источника, параметры подключения
Проверка подключения:	Роль подключения имеет	Создать роль только для

Симптом	Причина	Решение
«подключено, но роль может изменять данные — используйте роль только на чтение»	права на запись	чтения (раздел 5.5). Чтение в любом случае выполняется в транзакциях «только чтение»
При обзоре схемы или проверке подключения — «ошибка расшифровки»	Изменён ключ `ANALITIC_MASTER_KEY` после создания подключений	Вернуть прежний ключ из резервной копии `.env` либо заново ввести пароли подключений и целевых БД записи
После нажатия «Сформировать выводы» отображается «Выводы сформированы по встроенным правилам.», хотя брокер ИИ настроен; в журнале go-core `"msg":"narrative: ai broker failed, using rules"`	Брокер ИИ недоступен, неверны адрес, ключ или идентификатор модели, нет исходящего доступа	Проверить значения `ANALITIC_LLM_*` и исходящий доступ HTTPS с сервера; система продолжает работу, выводы формируются по встроенным правилам
Статусы узлов не обновляются в реальном времени, запросы обрываются	Внешний nginx буферизует потоковые ответы или ограничивает время ответа	Задать `proxy_buffering off` и `proxy_read_timeout 300s` (раздел 7)
Браузер получает ошибку 502 на запросы `/api/`	Ядро не запущено	Проверить `docker compose logs go-core --tail 100`
`docker compose pull` завершается ошибкой авторизации	Не выполнен вход в реестр образов	Выполнить `docker login <адрес_реестра_образов>` с учётными данными правообладателя
Изменения в `.env` не применяются после `docker compose restart`	Команда `restart` не перечитывает переменные окружения	Пересоздать контейнеры: `docker compose up -d`

7. Настройка внешнего nginx с HTTPS

Для публикации системы по доменному имени с шифрованием трафика на хосте устанавливается веб-сервер nginx, который принимает подключения на порт 443 и передаёт их на порт `PUBLIC_PORT` сервиса `react-front`. В этом случае рекомендуется опубликовать `react-front` на отдельном порту только на локальном интерфейсе, например `PUBLIC_PORT=8080` и в `docker-compose.yml`: `"127.0.0.1:${PUBLIC_PORT:-80}:80"`.

Пример конфигурации (`/etc/nginx/conf.d/kortex-analitika.conf`):

```
# ============================================
# HTTP сервер (порт 80) — перенаправление на HTTPS
# ============================================
```

```
server {
    listen 80;
    server_name ваш_домен;
    return 301 https://$host$request_uri;
}

# ============================================
# HTTPS сервер (порт 443)
# ============================================
server {
    listen 443 ssl;
    server_name ваш_домен;

    # SSL-сертификаты — укажите свои пути
    ssl_certificate     /etc/nginx/ssl/cert.crt;
    ssl_certificate_key /etc/nginx/ssl/private.key;
    ssl_protocols       TLSv1.2 TLSv1.3;

    # Загрузка файлов CSV / XLSX до 100 МиБ, импорт пайплайнов
    client_max_body_size 120m;

    # Все запросы (клиентское приложение и /api/) передаются
    # сервису react-front; он сам проксирует /api/ на ядро.
    # Пропускаются все методы HTTP (GET, POST, PUT, PATCH, DELETE):
    # ограничения методов (limit_except) не задаются.
    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_http_version 1.1;
        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;

        # Потоки статуса задач (server-sent events) —
        # без буферизации
        proxy_buffering off;
        proxy_cache off;

        # Длительные операции: формирование снимков,
        # интеллектуальные выводы
        proxy_read_timeout 300s;
        proxy_send_timeout 300s;
    }
}
```

Значение в `proxy_pass` должно совпадать с `PUBLIC_PORT`. Проверка и применение конфигурации:

```
nginx -t
systemctl reload nginx

# Проверка пропуска методов изменения данных через прокси:
# ответ не должен быть 403 или 405 от внешнего nginx
curl -s -o /dev/null -w "%{http_code}\n" -X PUT https://ваш_домен/api/v1/healthz
```

Система использует методы HTTP GET, POST, PUT, PATCH и DELETE; запрет любого из них на внешнем прокси сделает невозможным сохранение изменений.

8. Резервное копирование

Подлежат резервному копированию:

- БД метаданных `analitic_meta` — пользователи, проекты, подключения (с зашифрованными паролями), пайплайны и их ревизии, прогоны, результаты анализа, реестр моделей, дашборды, расписания, алерты, журнал событий;
- тома `snapshots`, `derived`, `trained_models` — снимки данных, производные данные и файлы обученных моделей (записи о них хранятся в БД метаданных, поэтому БД и тома копируются вместе);
- файлы `docker-compose.yml` и `.env` — в том числе ключ `ANALITIC_MASTER_KEY`, без которого сохранённые пароли подключений не могут быть расшифрованы.

Том `rabbit_data` содержит только текущие очереди и копированию не подлежит.

```
# ============================================
# Каталог для резервных копий
# ============================================
mkdir -p /backup
chmod 700 /backup
cd /opt/kortex-analitika

# ============================================
# 1. Дамп БД метаданных (встроенная БД)
# ============================================
docker compose exec -T postgres-meta sh -c \
  'pg_dump -U "$POSTGRES_USER" -d "$POSTGRES_DB" -Fc' \
  > /backup/analitic_meta_$(date +%Y%m%d).dump

# ============================================
# 2. Архив томов снимков, производных данных и моделей.
#    Архив формируется утилитой tar внутри контейнера ядра,
#    к которому подключены все три тома, и выводится на хост.
# ============================================
docker compose exec -T go-core \
  tar -czf - -C /data snapshots derived models \
  > /backup/kortex-analitika-data_$(date +%Y%m%d).tar.gz

# ============================================
# 3. Файлы конфигурации
# ============================================
tar -czf /backup/kortex-analitika-config_$(date +%Y%m%d).tar.gz \
  -C /opt/kortex-analitika docker-compose.yml .env
chmod 600 /backup/*
```

Для согласованности БД и томов копирование рекомендуется выполнять в период, когда не выполняются прогоны пайплайнов и расписания.

Восстановление:

```
cd /opt/kortex-analitika

# ==========================================
# 1. Остановить ядро и обработчики
# ==========================================
docker compose stop go-core python-worker python-worker-heavy python-worker-
writeback

# ==========================================
# 2. Восстановить БД метаданных из дампа
# ==========================================
docker compose exec -T postgres-meta sh -c \
  'pg_restore -U "$POSTGRES_USER" -d "$POSTGRES_DB" --clean --if-exists' \
  < /backup/analitic_meta_ГГГГММДД.dump

# ==========================================
# 3. Восстановить тома через временный контейнер ядра
# ==========================================
docker compose run --rm --no-deps -T --entrypoint tar go-core \
  -xzf - -C /data < /backup/kortex-analitika-data_ГГГГММДД.tar.gz

# ==========================================
# 4. Запустить сервисы
# ==========================================
docker compose up -d
```

При восстановлении на новый сервер предварительно восстанавливаются файлы `docker-compose.yml` и `.env` (с прежним ключом `ANALITIC_MASTER_KEY`), выполняются шаги 2 и 3 раздела 5 (миграции применяются к восстановленной БД), после чего — приведённые выше команды. Для варианта с внешней БД метаданных дамп и восстановление выполняются утилитами `pg_dump` / `pg_restore` на внешнем сервере БД.

9. Профилактика БД

При снижении производительности выполните:

```
# ==========================================
# Оптимизация БД метаданных (VACUUM ANALYZE)
# ==========================================
cd /opt/kortex-analitika
docker compose exec postgres-meta sh -c \
  'psql -U "$POSTGRES_USER" -d "$POSTGRES_DB" -c "VACUUM ANALYZE;"'

# ==========================================
# Размер БД метаданных и занятость дискового пространства
# ==========================================
docker compose exec postgres-meta sh -c \
  'psql -U "$POSTGRES_USER" -d "$POSTGRES_DB" -c "SELECT
pg_size_pretty(pg_database_size(current_database()));"'
docker system df -v | grep kortex-analitika
```

Производные данные прогонов, не входящих в `DERIVED_RETENTION_RUNS` последних прогонов пайплайна, удаляются ядром автоматически (первый проход — через минуту после запуска, далее — ежечасно). Записи о таких данных сохраняются в БД со статусом «очищен», чтобы результаты анализа оставались доступны. Производные данные, на основе которых обучены сохранённые модели или построены пайплайны, не удаляются. Размер БД метаданных и свободное место на томе снимков отображаются в разделе «Администрирование» → «Диагностика».

10. Обновление версии

Для обновления системы укажите в `.env` номер новой версии (`IMAGE_TAG`; значение сообщает правообладатель) и выполните:

```
cd /opt/kortex-analitika

# Рекомендуется предварительно выполнить резервное копирование (раздел 8)

# ============================================
# 1. Получить образы новой версии
# ============================================
# Вариант А — из реестра образов
docker compose pull
# Вариант Б — из архива
docker load -i kortex-analitika-<новая_версия>.tar.gz

# ============================================
# 2. Применить миграции новой версии
# ============================================
docker compose up -d postgres-meta rabbitmq
docker compose run --rm --entrypoint /app/migrate go-core up

# ============================================
# 3. Пересоздать сервисы с новыми образами
# ============================================
docker compose up -d
```

Миграции применяются до пересоздания ядра и обработчиков: новая версия ядра рассчитывает на новую структуру БД, а обработчики при запуске публикуют обновлённый каталог методов. Данные в томах сохраняются. Результат обновления проверяется командами раздела 5.4; номер версии отображается в веб-интерфейсе в разделе «О программе». При развертывании под управлением Deckhouse обновление выполняется так: в манифестах указывается новый тег образов, запускается Job миграций с образом новой версии и после его завершения выполняется обновление Deployment (`kubectl -n <namespace> rollout status deploy/go-core` и т. д. для остальных Deployment). Для возврата к предыдущей версии укажите прежнее значение `IMAGE_TAG` и восстановите БД метаданных из резервной копии, сделанной перед обновлением.

ООО «КОРТЕКС»
350063, г. Краснодар, ул. Северная, д. 324, офис 12/2
www.cortex.ooo

11. Примечания

- Все пароли, адреса и секретные ключи в примерах — заглушки. Обязательно замените их на собственные значения.
- Для получения ключа `ANALITIC_MASTER_KEY` используйте команду `openssl rand -base64 32`; для `JWT_SECRET` и паролей — `openssl rand -hex 32`.
- Ключ `ANALITIC_MASTER_KEY` после создания подключений к внешним БД и целевых БД записи не изменяется: при его замене сохранённые пароли не будут расшифрованы, и их потребуется ввести заново.
- Файл `.env` содержит секреты: права доступа `600`, копия хранится только в защищённом резервном хранилище.
- Все сервисы запускаются с политикой перезапуска `restart: unless-stopped`: контейнеры автоматически запускаются после перезагрузки сервера и перезапускаются при сбое, если не были остановлены администратором вручную.
- На хосте публикуется единственный порт — `PUBLIC_PORT` сервиса `react-front`. Порты ядра, брокера сообщений и БД метаданных доступны только внутри сети Docker Compose.
- Ядро и обработчики выполняются в контейнерах от имени непривилегированного пользователя (UID 10001).
- При развертывании под управлением Deckhouse секреты хранятся в объекте Secret, тома `snapshots`, `derived`, `trained_models` — в PersistentVolumeClaim с режимом ReadWriteMany; наружу публикуется только Ingress, сервисы ядра, брокера и БД доступны только внутри кластера.
- Команда `docker compose down` сохраняет тома; команда `docker compose down -v` удаляет тома со всеми данными системы и без резервной копии не применяется.
- Пароли целевых БД записи передаются пулу записи через внутреннюю очередь брокера сообщений; брокер не публикуется за пределы сети Docker Compose.