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

ООО «КОРТЕКС»

Руководство по развертыванию программного обеспечения платформы автоматизации рабочих процессов «Кортекс Потоки»

г. Краснодар, 2026

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

1. Введение

Программное обеспечение платформы автоматизации рабочих процессов «Кортекс Потоки» (далее — ПО, «Кортекс Потоки») — клиент-серверное веб-приложение с «тонким» клиентом (браузер). ПО предназначено для визуального конструирования процессов обработки данных из функциональных узлов, их запуска вручную, по расписанию и по внешним запросам, контроля хода выполнения и хранения файлов в общем хранилище с доступом по протоколу SFTP.

ПО полностью совместимо с российскими операционными системами Astra Linux, ALT Linux (Альт) и РЕД ОС и с системами управления базами данных PostgreSQL и Postgres Pro. Основной способ развертывания — контейнеры Docker (Docker Compose); поддерживается развертывание в среде оркестрации контейнеров Deckhouse Kubernetes Platform.

ПО поставляется в виде контейнерных образов. Основной способ — развертывание на сервере заказчика средствами Docker и Docker Compose (разделы 4–5); для кластеров Deckhouse предусмотрены манифесты Kubernetes (раздел 11). Все компоненты описываются одним файлом `docker-compose.yml` и одним файлом переменных окружения `.env`. Рекомендуемый способ установки — автономная установка на один сервер, при которой СУБД PostgreSQL запускается в составе ПО. Допускается использование внешней СУБД PostgreSQL или Postgres Pro (раздел 5.4).

Настоящее руководство описывает подготовку сервера, установку, проверку работоспособности, первичную настройку, диагностику неисправностей, настройку защищённого доступа по протоколу HTTPS, резервное копирование, профилактику базы данных и обновление ПО. Все пароли, секретные ключи и адреса в примерах являются заглушками и подлежат замене.

2. Состав системы

ПО состоит из трёх сервисов, объединённых в проект Docker Compose с именем `pipeline`. Сервисы взаимодействуют через внутреннюю сеть `app`; наружу публикуются только порт веб-интерфейса и порт SFTP-сервера.

Сервис	Компоненты	Порты	База данных
frontend	nginx, клиентское веб-приложение, проксирование API и потока событий	80 → `FRONTEND_PORT` (80)	—
backend	серверное приложение: API, исполнитель, планировщик, SFTP-сервер, метрики	8080 (внутренний); 2222 → `SFTP_PORT` (2222)	pipeline
db	PostgreSQL 16	5432 (внутренний)	pipeline, том `pipeline_pgdata`

Постоянные данные ПО размещаются в двух местах:

- том Docker `pipeline_pgdata` — файлы базы данных (процессы и их версии, запуски, пользователи, права доступа, системные настройки, журнал событий, ключ SFTP-сервера, настройки ИИ-ассистента);
- каталог сервера `/srv/pipeline-storage` (переменная `STORAGE_HOST_PATH`) — файловое хранилище, монтируется в контейнер backend как `/storage`; то же дерево файлов доступно в разделе «Файлы» веб-интерфейса и по протоколу SFTP.

Порядок запуска сервисов обеспечивается проверками готовности: backend запускается после того, как СУБД ответила на `pg_isready`, frontend — после того, как backend ответил на запрос `/health`.

3. Требования

3.1. Операционная система и программное окружение

Компонент	Требование
Операционная система	российские ОС (полная совместимость): Astra Linux Special Edition 1.7 и новее, ALT Linux (Альт Сервер 10 и новее), РЕД ОС 7.3 и новее; иные 64-разрядные ОС семейства Linux
Контейнеризация	Docker Engine версии 24 и новее
Оркестрация	Docker Compose v2 (команда `docker compose`) версии 2.20 и новее — основной способ; Deckhouse Kubernetes Platform — раздел 11
СУБД	PostgreSQL 16 в составе ПО (по умолчанию) либо внешняя СУБД PostgreSQL версии 13 и новее или Postgres Pro (полная совместимость)
Сетевой доступ	свободные порты 80 (веб-интерфейс) и 2222 (SFTP) либо иные, заданные переменными `FRONTEND_PORT` и `SFTP_PORT`; доступ к реестру образов разработчика либо архив образов (раздел 5.5)
Рабочее место пользователя	браузер (Chrome 110+, Яндекс Браузер 23+, Firefox 110+, Opera 95+, Safari 16+, Microsoft Edge на базе Chromium 110+)

Тестовый экземпляр ПО эксплуатируется на ОС ALT Server 10.4 с Docker 27.1 и Docker Compose 2.29.

3.2. Аппаратные требования

Параметр	Минимальные	Рекомендуемые
Процессор	2 ядра x86-64	4 ядра x86-64
Оперативная память	4 ГБ	8 ГБ
Дисковое пространство	20 ГБ	50 ГБ и более, в зависимости

Параметр	Минимальные	Рекомендуемые
		от объёма файлового хранилища и истории запусков

Объём базы данных определяется числом процессов, глубиной хранения истории запусков и версий (переменные `RUN_RETENTION_DAYS`, `RUN_RETENTION_COUNT`, `VERSION_RETENTION_COUNT`); объём файлового хранилища — файлами, которые загружают пользователи и процессы.

4. Подготовка сервера

Все команды раздела выполняются от имени пользователя `root` или с использованием `sudo`.

4.1. Установка Docker в ALT Linux

```
# Обновить индекс пакетов и установить Docker Engine и Docker Compose v2
apt-get update
apt-get install -y docker-engine docker-compose-v2
```

4.2. Установка Docker в Astra Linux

```
# Обновить индекс пакетов и установить Docker и модуль Docker Compose v2
apt update
apt install -y docker.io docker-compose-plugin
```

Если пакет модуля Docker Compose отсутствует в подключённых репозиториях конкретной редакции Astra Linux, установку выполняют по документации Astra Linux для используемой версии ОС (раздел «Docker»), после чего проверяют наличие команды `docker compose`.

4.3. Установка Docker в РЕД ОС

```
# Установить Docker CE и модуль Docker Compose v2
dnf install -y docker-ce docker-compose-plugin
```

В РЕД ОС по умолчанию может быть включён механизм SELinux. Если после запуска серверное приложение не может записать файлы в каталог хранилища, см. раздел 6 (строка «storage root create failed»).

4.4. Запуск Docker и проверка версий

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

# Разрешить администратору ПО работу с Docker без sudo
# (вместо admin укажите имя своей учётной записи; требуется повторный вход в
систему)
usermod -aG docker admin
```

```
# Проверить версии (Docker 24+, Compose v2)
docker version --format '{{.Server.Version}}'
docker compose version
```

Участники группы `docker` фактически обладают правами суперпользователя на сервере; в группу включают только администраторов ПО.

4.5. Создание каталогов

```
# Каталог конфигурации ПО (docker-compose.yml и .env)
mkdir -p /opt/pipeline

# Каталог файлового хранилища (монтируется в контейнер backend как /storage)
mkdir -p /srv/pipeline-storage
```

5. Развертывание

5.1. Шаг 1. Файл docker-compose.yml

Создать файл `/opt/pipeline/docker-compose.yml` следующего содержания. В строках `image` адрес реестра образов берётся из переменной `REGISTRY` файла `.env`. Здесь и далее вместо `<реестр_образов>` указывается адрес приватного реестра образов, а в переменной `IMAGE_TAG` — номер поставляемой версии (в примерах — `v1.0.4`); оба значения сообщает разработчик.

```
# Автономная установка: собственный PostgreSQL + backend + frontend.
# Запуск: cd /opt/pipeline && docker compose up -d
# Переменные окружения — файл .env в том же каталоге.
name: pipeline

services:
  db:
    image: postgres:16-alpine
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${DB_USER:-pipeline}
      POSTGRES_PASSWORD: ${DB_PASSWORD:?DB_PASSWORD is required}
      POSTGRES_DB: ${DB_NAME:-pipeline}
    volumes:
      - pgdata:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U $$POSTGRES_USER -d $$POSTGRES_DB"]
      interval: 5s
      timeout: 5s
      retries: 12
    networks: [app]

  backend:
    image: ${REGISTRY:-<реестр_образов>}/pipeline-backend:${IMAGE_TAG:?IMAGE_TAG
is required}
    restart: unless-stopped
    # корректное завершение: до 30 с на HTTP + до 30 с на завершение запусков
    stop_grace_period: 60s
```

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

```
    depends_on:
      db:
        condition: service_healthy
    environment:
      DB_HOST: db
      DB_PORT: 5432
      DB_USER: ${DB_USER:-pipeline}
      DB_PASSWORD: ${DB_PASSWORD}
      DB_NAME: ${DB_NAME:-pipeline}
      SERVER_PORT: 8080
      SERVER_ENV: production
      JWT_SECRET: ${JWT_SECRET:?JWT_SECRET is required}
      JWT_ACCESS_TTL_MIN: ${JWT_ACCESS_TTL_MIN:-60}
      JWT_REFRESH_TTL_DAYS: ${JWT_REFRESH_TTL_DAYS:-30}
      ADMIN_EMAIL: ${ADMIN_EMAIL:-admin@local}
      ADMIN_PASSWORD: ${ADMIN_PASSWORD:?ADMIN_PASSWORD is required}
      LOG_LEVEL: ${LOG_LEVEL:-info}
      LOG_FORMAT: ${LOG_FORMAT:-json}
      RUN_RETENTION_DAYS: ${RUN_RETENTION_DAYS:-30}
      RUN_RETENTION_COUNT: ${RUN_RETENTION_COUNT:-200}
      VERSION_RETENTION_COUNT: ${VERSION_RETENTION_COUNT:-50}
      RETENTION_CRON: ${RETENTION_CRON:-0 * * * *}
      STORAGE_ROOT_PATH: /storage
      MAX_UPLOAD_BYTES: ${MAX_UPLOAD_BYTES:-104857600}
      SFTP_ENABLED: ${SFTP_ENABLED:-true}
      SFTP_ADDR: ":2222"
    ports:
      - "${SFTP_PORT:-2222}:2222"
    volumes:
      - ${STORAGE_HOST_PATH:-/srv/pipeline-storage}:/storage
    healthcheck:
      test: ["CMD", "wget", "-q", "-O", "/dev/null",
"http://127.0.0.1:8080/health"]
      interval: 10s
      timeout: 5s
      retries: 6
      start_period: 20s
    networks: [app]

  frontend:
    image: ${REGISTRY:-<реестр_образов>}/pipeline-frontend:${IMAGE_TAG:?IMAGE_TAG
is required}
    restart: unless-stopped
    depends_on:
      backend:
        condition: service_healthy
    ports:
      - "${FRONTEND_PORT:-80}:80"
    networks: [app]

volumes:
  pgdata:

networks:
  app:
    driver: bridge
```

Переменные, отмеченные в файле конструкцией `:?… is required` (`DB_PASSWORD`, `JWT_SECRET`, `ADMIN_PASSWORD`, `IMAGE_TAG`), обязательны: без них Docker Compose не запустит сервисы и выведет сообщение об ошибке (раздел 6).

5.2. Шаг 2. Файл переменных окружения .env

Создать файл `/opt/pipeline/.env` и ограничить доступ к нему, так как он содержит пароли и секретный ключ:

```
cd /opt/pipeline
touch .env && chmod 600 .env
```

Содержимое файла `.env`:

```
# ============================================
# Образы: реестр и версия (тег) релиза
# ============================================
# Адрес реестра образов — выдаётся разработчиком вместе с учётными данными
REGISTRY=<реестр_образов>
# Версия ПО; не используйте тег latest
IMAGE_TAG=v1.0.4

# ============================================
# Порты на сервере
# ============================================
# Веб-интерфейс (при внешнем обратном прокси — например, 127.0.0.1:8081, см.
раздел 7)
FRONTEND_PORT=80
# Встроенный SFTP-сервер
SFTP_PORT=2222

# ============================================
# PostgreSQL (сервис db)
# ============================================
DB_USER=pipeline
# Пароль применяется при первой инициализации тома pipeline_pgdata — ОБЯЗАТЕЛЬНО
ЗАМЕНИТЕ!
# Сгенерировать: openssl rand -base64 16
DB_PASSWORD=замените_на_свой_пароль
DB_NAME=pipeline

# ============================================
# Безопасность — ОБЯЗАТЕЛЬНО ЗАМЕНИТЕ
# ============================================
# Секрет подписи токенов доступа. Сгенерировать: openssl rand -hex 64
JWT_SECRET=замените_на_свой_секретный_ключ
# Срок действия токена доступа, минут, и токена обновления, дней
JWT_ACCESS_TTL_MIN=60
JWT_REFRESH_TTL_DAYS=30
# Учётная запись администратора, создаваемая при первом запуске
ADMIN_EMAIL=admin@example.local
# Пароль администратора (не менее 6 символов). Сгенерировать: openssl rand
-base64 16
ADMIN_PASSWORD=замените_на_свой_пароль

# ============================================
# Файловое хранилище и SFTP
```

```
# ==========================================
# Каталог хранилища на сервере
STORAGE_HOST_PATH=/srv/pipeline-storage
# Максимальный размер загружаемых данных за один запрос, байт (100 МБ)
MAX_UPLOAD_BYTES=104857600
# Встроенный SFTP-сервер: true — включён, false — выключен
SFTP_ENABLED=true

# ==========================================
# Журналирование и хранение истории
# ==========================================
# Уровень журналирования: debug | info | warn | error
LOG_LEVEL=info
# Формат журнала: json | text
LOG_FORMAT=json
# Удалять запуски старше N дней (0 — не удалять)
RUN_RETENTION_DAYS=30
# Хранить не более N последних запусков каждого процесса (0 — без ограничения)
RUN_RETENTION_COUNT=200
# Хранить не более N версий каждого процесса (0 — без ограничения)
VERSION_RETENTION_COUNT=50
# Расписание задания очистки истории (формат cron), по умолчанию — ежечасно
RETENTION_CRON=0 * * * *
```

Помимо переменных файла `.env`, серверное приложение поддерживает переменные `RUN_TIMEOUT_SECONDS` (общий таймаут одного запуска, по умолчанию 300 секунд) и `WEBHOOK_RATE_LIMIT` (не более N запросов в минуту на один адрес вебхука, по умолчанию 60; 0 — без ограничения). Для изменения их значений переменные добавляют в раздел `environment` сервиса backend. Таймаут запуска, глубина хранения истории и уровень журналирования также изменяются администратором в интерфейсе (раздел «Админ» → «Настройки») без перезапуска ПО; значения, заданные в интерфейсе, имеют приоритет над переменными окружения.

5.3. Шаг 3. Каталог файлового хранилища

Каталог, указанный в `STORAGE_HOST_PATH`, должен существовать до запуска (раздел 4.5). Серверное приложение в контейнере работает от имени `root` и создаёт в каталоге файлы и папки, загружаемые пользователями. Каталог может располагаться на отдельном разделе диска или сетевом хранилище, подключённом к серверу.

5.4. Вариант с внешней СУБД PostgreSQL или Postgres Pro

При использовании внешней СУБД (PostgreSQL версии 13 и новее или Postgres Pro) сервис db не запускается. На сервере СУБД создают пользователя и базу данных:

```
-- выполнить в psql от имени администратора СУБД
CREATE USER pipeline WITH PASSWORD 'замените_на_свой_пароль';
CREATE DATABASE pipeline OWNER pipeline;
```

В настройках СУБД (`pg_hba.conf`, `listen_addresses`) разрешают подключение пользователя `pipeline` к базе `pipeline` с адреса сервера ПО. Серверное

приложение подключается к СУБД без шифрования канала, поэтому сервер СУБД и сервер ПО должны находиться в одном доверенном сегменте сети.

В файле `docker-compose.yml` выполняют следующие изменения:

- удаляют сервис `db` целиком и раздел `volumes:` с томом `pgdata` в конце файла;
- в сервисе `backend` удаляют блок `depends_on` (зависимость от `db`);
- в сервисе `backend` заменяют строки `DB_HOST` и `DB_PORT` на значения из `.env`.

Фрагмент сервиса backend после изменения:

```
  backend:
    image: ${REGISTRY:-<реестр_образов>}/pipeline-backend:${IMAGE_TAG:?IMAGE_TAG
is required}
    restart: unless-stopped
    stop_grace_period: 60s
    environment:
      DB_HOST: ${DB_HOST:?DB_HOST is required}
      DB_PORT: ${DB_PORT:-5432}
      DB_USER: ${DB_USER:-pipeline}
      DB_PASSWORD: ${DB_PASSWORD:?DB_PASSWORD is required}
      DB_NAME: ${DB_NAME:-pipeline}
      # … остальные переменные — без изменений (раздел 5.1)
```

В файл `.env` добавляют адрес и порт СУБД:

```
# Внешняя СУБД PostgreSQL / Postgres Pro
DB_HOST=<адрес_сервера_СУБД>
DB_PORT=5432
DB_USER=pipeline
DB_PASSWORD=замените_на_свой_пароль
DB_NAME=pipeline
```

Структура базы данных создаётся серверным приложением автоматически при первом запуске; ручное выполнение SQL-скриптов не требуется.

5.5. Шаг 4. Получение образов

Образы ПО `pipeline-backend` и `pipeline-frontend` хранятся в приватном реестре образов разработчика. Учётные данные для чтения выдаются разработчиком по запросу. Авторизация в реестре:

```
# Логин и пароль (токен) выдаются разработчиком ПО
docker login <реестр_образов>
```

Если доступ к реестру образов с сервера невозможен (закрытый контур), используется один из двух альтернативных способов.

Способ 1 — установка из архива образов, предоставленного разработчиком:

```
# На сервере заказчика: загрузить образы из архива
docker load -i pipeline-images-v1.0.4.tar

# Проверить наличие образов
```

```
docker images | grep -E 'pipeline-(backend|frontend)|postgres'
```

Архив содержит образы `pipeline-backend`, `pipeline-frontend` и `postgres:16-alpine` и формируется разработчиком командой `docker save`. При установке из архива значение `REGISTRY` в `.env` должно совпадать с префиксом имени загруженных образов, а команду `docker compose pull` (шаг 5) не выполняют.

Способ 2 — сборка образов из исходного кода (исходный код предоставляется разработчиком):

```
# В каталоге исходного кода ПО
docker build --build-arg VERSION=v1.0.4 -t <реестр_образов>/pipeline-
backend:v1.0.4 ./backend
docker build -t <реестр_образов>/pipeline-frontend:v1.0.4 ./frontend
```

Для сборки требуется доступ сервера к репозиториям базовых образов и пакетов либо их локальные зеркала. Аргумент `VERSION` задаёт номер версии, отображаемый в разделе «Админ» → «Диагностика».

5.6. Шаг 5. Загрузка образов и запуск

```
cd /opt/pipeline

# Загрузить образы из реестра (пропустить при установке из архива)
docker compose pull

# Запустить сервисы в фоновом режиме
docker compose up -d
```

При первом запуске серверное приложение автоматически создаёт структуру базы данных, учётную запись администратора (`ADMIN_EMAIL` / `ADMIN_PASSWORD`) и ключ SFTP-сервера.

5.7. Шаг 6. Проверка работоспособности

Состояние сервисов:

```
cd /opt/pipeline
docker compose ps
```

Сервисы `db` и `backend` должны находиться в состоянии `Up (healthy)`, сервис `frontend` — в состоянии `Up`. Проверка журнала серверного приложения:

```
docker compose logs backend | grep -E 'scheduler started|sftp server listening|
server starting'
```

При успешном запуске журнал (формат JSON) содержит строки:

```
{"level":"INFO","msg":"scheduler started","jobs":1}
{"level":"INFO","msg":"sftp server listening","addr":":2222","root":"/storage"}
{"level":"INFO","msg":"server starting","port":"8080"}
```

Строка `sftp server listening` выводится только при `SFTP_ENABLED=true`. Проверка эндпоинта работоспособности серверного приложения:

```
docker compose exec backend wget -qO- http://127.0.0.1:8080/health
# Ожидаемый ответ:
# {"status":"ok"}
```

Проверка веб-сервера (клиентского приложения):

```
curl -I http://localhost/
# Ожидаемый ответ: HTTP/1.1 200 OK
```

5.8. Шаг 7. Первичная настройка

1. Открыть в браузере адрес `http://<адрес_сервера>/` (при настроенном HTTPS — `https://<доменное_имя>/`) и выполнить вход с e-mail `ADMIN_EMAIL` и паролем `ADMIN_PASSWORD`.
2. Сменить пароль администратора: раздел «Админ» → вкладка «Пользователи» → кнопка «Сбросить пароль» в строке администратора → ввести новый пароль (не менее 6 символов). Изменение переменной `ADMIN_PASSWORD` после первого запуска на существующую учётную запись не влияет: пароль задаётся только при её создании.
3. Создать учётные записи пользователей: вкладка «Пользователи» → кнопка «+ Пользователь» → указать e-mail, имя, пароль и при необходимости признак администратора → «Сохранить». Пользователи также могут зарегистрироваться самостоятельно на странице входа.
4. Проверить системные настройки на вкладке «Настройки» (таймаут запуска, глубина хранения истории, уровень журналирования) и состояние компонентов на вкладке «Диагностика» (версия, СУБД, планировщик, хранилище, SFTP).
5. Проверить доступ к файловому хранилищу по SFTP. Логин — e-mail пользователя, пароль — тот же, что для веб-интерфейса; доступ ограничен корнем хранилища, выполнение команд оболочки не допускается:

```
sftp -P 2222 <email>@<адрес_сервера>
# если e-mail содержит символ @, логин удобнее передать параметром:
sftp -P 2222 -o User=user@example.local <адрес_сервера>
```

6. При необходимости использования ИИ-ассистента администратор настраивает подключение к сервису большой языковой модели (внешнему или размещённому в инфраструктуре заказчика) с совместимым программным интерфейсом: раздел «Пайплайны» → кнопка «Настройки ИИ» → поля «Провайдер», «Модель», «API-ключ», «API URL» → сохранить. До выполнения настройки функция ИИ-ассистента неактивна: при обращении к ней выводится сообщение о необходимости настройки подключения. Серверу ПО должен быть доступен сетевой адрес сервиса большой языковой модели.

6. Диагностика

В случае неисправности в первую очередь проверяют состояние сервисов и журналы:

```
cd /opt/pipeline
docker compose ps
docker compose logs --tail=50 backend
docker compose logs --tail=50 db
docker compose logs --tail=50 frontend
```

Для подробной диагностики уровень журналирования повышают до `debug` в разделе «Админ» → «Настройки» (без перезапуска) или переменной `LOG_LEVEL=debug` с последующим `docker compose up -d`.

Симптом	Причина	Решение
`docker compose up`: `DB_PASSWORD is required` (так же для `JWT_SECRET`, `ADMIN_PASSWORD`, `IMAGE_TAG`)	не задана обязательная переменная или нет файла `.env` рядом с `docker-compose.yml`	заполнить `/opt/pipeline/.env`, запускать команды из `/opt/pipeline`
`docker compose pull`: `unauthorized`, `pull access denied`	нет авторизации в реестре, неверный `REGISTRY` или `IMAGE_TAG`	`docker login` (раздел 5.5) или установка из архива
backend перезапускается, в журнале `db connect failed`	СУБД недоступна или неверны учётные данные	проверить `DB_HOST`, `DB_PORT`, `DB_USER`, `DB_PASSWORD` и журнал db
`password authentication failed` после смены `DB_PASSWORD`	пароль встроенной СУБД задаётся только при первой инициализации тома	вернуть прежний пароль либо выполнить в СУБД `ALTER USER pipeline PASSWORD '…';`
в журнале backend `migrate failed`	нет прав на создание таблиц или СУБД версии ниже 13	сделать пользователя владельцем БД, использовать PostgreSQL 13+ / Postgres Pro
в журнале `storage root create failed`, ошибки записи в «Файлы»	каталог хранилища недоступен; запрет SELinux (РЕД ОС)	проверить каталог; `chcon -Rt container_file_t /srv/pipeline-storage`
frontend не стартует: `container pipeline-backend-1 is unhealthy`	backend не прошёл проверку `/health`	устранить причину по журналу backend
`port is already allocated`	порт 80 или 2222 занят	изменить `FRONTEND_PORT`, `SFTP_PORT`
при входе `invalid credentials`	неверный пароль; `ADMIN_PASSWORD` изменён после первого запуска (не применяется)	войти с прежним паролем; при его утрате задать новые `ADMIN_EMAIL` и `ADMIN_PASSWORD` и выполнить `docker compose`

Симптом	Причина	Решение
		`up -d` — будет создан новый администратор
при входе `account disabled`	учётная запись заблокирована	разблокировать в «Админ» → «Пользователи»
ход выполнения не обновляется через внешний прокси	прокси буферизует поток событий	`proxy_buffering off` (раздел 7)
изменения не сохраняются через внешний прокси (403, 405)	прокси пропускает только GET и POST	разрешить PUT, PATCH, DELETE для `/api/`
вебхук возвращает `webhook rate limit exceeded` (429)	превышена частота запросов	увеличить `WEBHOOK_RATE_LIMIT` (раздел 5.2)
ИИ-ассистент: «Интеграция с ИИ не настроена…»	нет подключения к сервису большой языковой модели	настроить (раздел 5.8, п. 6)

7. Настройка nginx с HTTPS

Для доступа по доменному имени по протоколу HTTPS перед ПО устанавливается внешний обратный прокси-сервер nginx (на том же сервере либо на пограничном сервере). Если прокси-сервер размещается на том же сервере и занимает порты 80 и 443, веб-интерфейс ПО публикуется только на локальном интерфейсе — в `.env` указывается `FRONTEND_PORT=127.0.0.1:8081`, после чего выполняется `docker compose up -d`.

Пример конфигурации `/etc/nginx/conf.d/pipeline.conf`:

```
# ============================================
# 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;

    # Размер загружаемых файлов — не меньше MAX_UPLOAD_BYTES
    client_max_body_size 100m;

    # Поток событий хода выполнения процессов (SSE): без буферизации
    location ~* ^/api/v1/runs/.+/events {
        proxy_pass http://127.0.0.1:8081;
        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;
        proxy_set_header Connection '';
```

```
        proxy_buffering off;
        proxy_cache off;
        proxy_read_timeout 600s;
    }

    # Клиентское приложение и API
    location / {
        proxy_pass http://127.0.0.1:8081;
        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;
        proxy_read_timeout 600s;
    }
}

# ============================================
# HTTP-сервер (порт 80) — перенаправление на HTTPS
# ============================================
server {
    listen 80;
    server_name ваш_домен;
    return 301 https://$host$request_uri;
}
```

Если прокси-сервер размещён на отдельном сервере, вместо `127.0.0.1:8081` указывается адрес и порт `FRONTEND_PORT` сервера ПО. Проверка и применение конфигурации:

```
nginx -t && systemctl reload nginx
curl -I https://ваш_домен/
```

Время ожидания `proxy_read_timeout` выбирается не меньше общего таймаута запуска процесса (по умолчанию 300 секунд), чтобы синхронные запуски и поток событий не прерывались прокси-сервером. Прокси-сервер должен пропускать методы HTTP GET, POST, PUT, PATCH и DELETE. Протокол SFTP (порт 2222) через HTTP-прокси не проходит и публикуется отдельно.

8. Резервное копирование

Резервному копированию подлежат база данных и каталог файлового хранилища. Файлы `docker-compose.yml` и `.env` копируются отдельно при каждом их изменении. Резервные копии содержат учётные данные пользователей (хэши паролей), ключ SFTP-сервера и настройки подключения к сервису большой языковой модели, поэтому хранятся с ограниченным доступом.

```
# Каталог резервных копий
mkdir -p /backup/pipeline && chmod 700 /backup/pipeline
cd /opt/pipeline

# Дамп базы данных (встроенная СУБД)
docker compose exec -T db pg_dump -U pipeline pipeline \
```

```
  | gzip > /backup/pipeline/db_$(date +%Y%m%d).sql.gz

# Архив файлового хранилища
tar -czf /backup/pipeline/storage_$(date +%Y%m%d).tar.gz -C /srv/pipeline-storage
.
```

Для внешней СУБД дамп выполняется штатной утилитой `pg_dump` с сервера СУБД или с сервера, имеющего к ней доступ: `pg_dump -h <адрес_сервера_СУБД> -U pipeline pipeline | gzip > /backup/pipeline/db_$(date +%Y%m%d).sql.gz`.

Для регулярного выполнения команды оформляются сценарием и добавляются в планировщик `cron` (например, ежесуточно в 02:00), с удалением копий старше установленного срока хранения.

Восстановление из резервной копии:

```
cd /opt/pipeline

# Остановить серверное приложение и веб-сервер
docker compose stop frontend backend

# Пересоздать базу данных и загрузить дамп
docker compose exec -T db psql -U pipeline -d postgres -c "DROP DATABASE
pipeline;"
docker compose exec -T db psql -U pipeline -d postgres -c "CREATE DATABASE
pipeline OWNER pipeline;"
gunzip -c /backup/pipeline/db_ГГГГММДД.sql.gz | docker compose exec -T db psql -U
pipeline pipeline

# Восстановить файловое хранилище
tar -xzf /backup/pipeline/storage_ГГГГММДД.tar.gz -C /srv/pipeline-storage

# Запустить ПО
docker compose up -d
```

9. Профилактика базы данных

В СУБД включена автоматическая очистка (autovacuum), а серверное приложение по расписанию `RETENTION_CRON` удаляет устаревшие запуски и версии процессов. После массового удаления данных или при снижении производительности выполняют сборку мусора и обновление статистики:

```
# Оптимизация производительности БД (VACUUM ANALYZE)
cd /opt/pipeline
docker compose exec db psql -U pipeline -d pipeline -c "VACUUM ANALYZE;"
```

Наибольший объём, как правило, занимает таблица истории запусков `pipeline_runs` (выходные данные узлов). При её росте уменьшают глубину хранения истории в разделе «Админ» → «Настройки».

10. Обновление

Обновление выполняется заменой версии образов; структура базы данных приводится к новой версии автоматически при запуске серверного приложения, ручное выполнение миграций не требуется.

```
cd /opt/pipeline

# 1. Создать резервную копию (раздел 8)
docker compose exec -T db pg_dump -U pipeline pipeline \
  | gzip > /backup/pipeline/db_before_update_$(date +%Y%m%d).sql.gz

# 2. Указать новую версию в .env
sed -i 's/^IMAGE_TAG=.*/IMAGE_TAG=<новая_версия>/' .env

# 3. Загрузить образы и перезапустить сервисы
docker compose pull
docker compose up -d

# 4. Проверить работоспособность (раздел 5.7)
docker compose ps
docker compose exec backend wget -qO- http://127.0.0.1:8080/health
```

При обновлении из архива вместо `docker compose pull` выполняется `docker load -i` с архивом новой версии. Серверное приложение завершает работу корректно: в течение до 60 секунд (`stop_grace_period`) оно прекращает приём запросов и дожидается завершения выполняемых запусков. Номер установленной версии отображается в разделе «Админ» → «Диагностика». Для возврата к предыдущей версии указывают прежнее значение `IMAGE_TAG` и восстанавливают базу данных из резервной копии, созданной перед обновлением.

11. Развертывание в среде оркестрации Deckhouse

ПО поддерживает развертывание в среде оркестрации контейнеров Deckhouse Kubernetes Platform (далее — Deckhouse) и в других кластерах Kubernetes. Для этого в состав ПО входит каталог `deploy/kubernetes` с манифестами, собираемыми средством kustomize (встроено в `kubectl`). Используются те же контейнерные образы, что и при развертывании средствами Docker Compose.

11.1. Состав манифестов

Файл	Назначение
`kustomization.yaml`	перечень манифестов, пространство имён `pipeline`, адрес реестра и версия образов
`namespace.yaml`	пространство имён `pipeline`
`secret.example.yaml`	образец секрета с паролями (копируется в `secret.yaml`)
`configmap.yaml`	параметры ПО (подключение к СУБД, журналирование, хранилище, SFTP)
`postgres.yaml`	встроенная СУБД PostgreSQL 16 (StatefulSet,

Файл	Назначение
	том 10 ГБ); не используется при внешней СУБД
`backend.yaml`	серверное приложение (Deployment, одна реплика), том файлового хранилища 20 ГБ, сервис `backend`
`frontend.yaml`	клиентское приложение с веб-сервером nginx (Deployment, сервис `frontend`)
`ingress.yaml`	веб-доступ через ingress-nginx и сервис `backend-sftp` для доступа по SFTP извне кластера

11.2. Требования

- кластер Deckhouse (или иной кластер Kubernetes версии 1.27 и новее) с включёнными модулями `ingress-nginx` (класс Ingress `nginx`) и, для HTTPS, `cert-manager`;
- класс хранения по умолчанию (StorageClass) с томами режима ReadWriteOnce;
- доступ узлов кластера к реестру образов разработчика (или загрузка образов во внутренний реестр заказчика);
- утилита `kubectl` с доступом к кластеру;
- для внешней СУБД — PostgreSQL версии 13 и новее или Postgres Pro.

11.3. Порядок развертывания

1. Скопируйте образец секрета и задайте пароли:

```
cd deploy/kubernetes
cp secret.example.yaml secret.yaml
# в secret.yaml замените значения DB_PASSWORD, JWT_SECRET, ADMIN_PASSWORD
# сгенерировать: openssl rand -hex 32
```

2. Укажите адрес реестра и версию образов в `kustomization.yaml`:

```
# Развертывание «Кортекс Потоки» в Kubernetes / Deckhouse Kubernetes Platform.
#   1. Скопируйте secret.example.yaml в secret.yaml и задайте пароли.
#   2. Укажите реестр и версию образов ниже, домен — в ingress.yaml.
#   3. kubectl apply -k deploy/kubernetes
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: pipeline
resources:
  - namespace.yaml
  - secret.yaml
  - configmap.yaml
  - postgres.yaml
  - backend.yaml
  - frontend.yaml
  - ingress.yaml
images:
  - name: pipeline-backend
```

```
    newName: <реестр_образов>/pipeline-backend
    newTag: v1.0.4
  - name: pipeline-frontend
    newName: <реестр_образов>/pipeline-frontend
    newTag: v1.0.4
```

3. Укажите доменное имя в `ingress.yaml` (поле `host`) и при необходимости включите HTTPS: раскомментируйте аннотацию `cert-manager.io/cluster-issuer` и блок `tls`.
4. Если реестр образов требует авторизации, создайте секрет доступа и укажите его в манифестах `backend.yaml` и `frontend.yaml` (поле `imagePullSecrets`):

```
kubectl create namespace pipeline
kubectl -n pipeline create secret docker-registry registry-auth \
  --docker-server=<реестр_образов> --docker-username=<логин> --docker-
password=<пароль>
```

5. Примените манифесты и дождитесь готовности:

```
kubectl apply -k deploy/kubernetes
kubectl -n pipeline rollout status statefulset/postgres
kubectl -n pipeline rollout status deploy/backend
kubectl -n pipeline rollout status deploy/frontend
kubectl -n pipeline get pods
```

Все три пода должны перейти в состояние `Running` и готовность `1/1`. Серверное приложение запускается после готовности СУБД (init-контейнер `wait-db`), изменения структуры базы данных применяются автоматически.

11.4. Манифест серверного приложения

```
# Серверное приложение. Строго одна реплика: планировщик и поток событий
# выполнения работают внутри процесса.
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pipeline-storage
spec:
  accessModes: ["ReadWriteOnce"]
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: backend
```

```
  template:
    metadata:
      labels:
        app: backend
    spec:
      terminationGracePeriodSeconds: 60
      # Ожидание готовности СУБД: серверное приложение подключается к ней один
раз при старте.
      initContainers:
        - name: wait-db
          image: postgres:16-alpine
          envFrom:
            - configMapRef: { name: pipeline-config }
          command: ["sh", "-c", "until pg_isready -h \"$DB_HOST\" -p
\"$DB_PORT\"; do echo waiting for db; sleep 2; done"]
      containers:
        - name: backend
          image: pipeline-backend
          envFrom:
            - configMapRef: { name: pipeline-config }
            - secretRef: { name: pipeline-secrets }
          ports:
            - { name: http, containerPort: 8080 }
            - { name: sftp, containerPort: 2222 }
          readinessProbe:
            httpGet: { path: /health, port: http }
            periodSeconds: 10
          livenessProbe:
            httpGet: { path: /health, port: http }
            initialDelaySeconds: 30
            periodSeconds: 20
          resources:
            requests: { cpu: 100m, memory: 128Mi }
            limits: { memory: 1Gi }
          volumeMounts:
            - name: storage
              mountPath: /storage
      volumes:
        - name: storage
          persistentVolumeClaim:
            claimName: pipeline-storage
---
# Имя сервиса «backend» используется веб-сервером клиентского приложения.
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: backend
  ports:
    - { name: http, port: 8080, targetPort: http }
    - { name: sftp, port: 2222, targetPort: sftp }
```

11.5. Манифест веб-доступа и SFTP

```
# Веб-доступ через ingress-nginx (в Deckhouse — модуль ingress-nginx, класс
```

```
«nginx»).
# TLS: раскомментируйте аннотацию cert-manager и блок tls (модуль cert-manager
Deckhouse).
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: pipeline
  annotations:
    # Поток событий выполнения (SSE) и загрузка файлов
    nginx.ingress.kubernetes.io/proxy-buffering: "off"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "300"
    nginx.ingress.kubernetes.io/proxy-body-size: 110m
    # cert-manager.io/cluster-issuer: letsencrypt
spec:
  ingressClassName: nginx
  # tls:
  #   - hosts: [pipeline.example.local]
  #     secretName: pipeline-tls
  rules:
    - host: pipeline.example.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend
                port: { name: http }
---
# Доступ к файловому хранилищу по SFTP извне кластера (порт 2222).
# В Deckhouse с балансировщиком — type: LoadBalancer; без него — NodePort.
apiVersion: v1
kind: Service
metadata:
  name: backend-sftp
spec:
  type: NodePort
  selector:
    app: backend
  ports:
    - { name: sftp, port: 2222, targetPort: sftp, nodePort: 30222 }
```

Аннотации Ingress обязательны: `proxy-buffering: "off"` обеспечивает передачу событий хода выполнения процессов в реальном времени, `proxy-body-size` — загрузку файлов до 100 МБ.

11.6. Проверка и эксплуатация

```
# Проверка работоспособности серверного приложения — ответ {"status":"ok"}
kubectl -n pipeline exec deploy/backend -- wget -qO- http://127.0.0.1:8080/health
# Журналы
kubectl -n pipeline logs deploy/backend --tail=50
# Резервная копия встроенной СУБД
kubectl -n pipeline exec postgres-0 -- pg_dump -U pipeline pipeline | gzip >
backup-$(date +%F).sql.gz
# Обновление: указать новую версию в kustomization.yaml (newTag) и применить
kubectl apply -k deploy/kubernetes
```

Веб-интерфейс открывается по доменному имени, указанному в `ingress.yaml`; вход выполняется под учётной записью `ADMIN_EMAIL` / `ADMIN_PASSWORD`. Доступ по SFTP — через сервис `backend-sftp` (NodePort 30222 или, при наличии балансировщика, тип LoadBalancer).

11.7. Особенности

- серверное приложение работает строго в одной реплике (стратегия обновления `Recreate`): планировщик запусков и поток событий выполнения находятся внутри процесса; горизонтальное масштабирование не применяется;
- тома файлового хранилища и СУБД используют режим ReadWriteOnce;
- для внешней СУБД PostgreSQL или Postgres Pro укажите её адрес в `configmap.yaml` (`DB_HOST`, `DB_PORT`, `DB_USER`, `DB_NAME`), пароль — в `secret.yaml`, и удалите `postgres.yaml` из списка `resources` в `kustomization.yaml`;
- манифесты проверены в кластере Kubernetes с контроллером ingress-nginx: развертывание, вход в систему, запуск процессов, передача событий хода выполнения через Ingress, загрузка файлов, доступ по SFTP, сохранность файлов при перезапуске серверного приложения.

12. Примечания

- Все пароли, секретные ключи, адреса и доменные имена в примерах — заглушки; перед запуском их обязательно заменяют на собственные.
- Генерация секрета подписи токенов: `openssl rand -hex 64`; генерация пароля: `openssl rand -base64 16`.
- Файл `.env` содержит пароли и секрет подписи токенов, права доступа к нему — `600`.
- Смена `JWT_SECRET` делает недействительными все выданные токены: пользователям потребуется повторный вход.
- Для промышленной эксплуатации рекомендуется HTTPS (раздел 7), регулярное резервное копирование (раздел 8) и контроль состояния по эндпоинту `/health` и метрикам `/metrics`.
- Команда `docker compose down` останавливает и удаляет контейнеры, но сохраняет том `pipeline_pgdata` и каталог хранилища; команда `docker compose down -v` удаляет том базы данных со всеми данными.
- Для всех сервисов задана политика перезапуска `unless-stopped`: после перезагрузки сервера ПО запускается автоматически вместе со службой Docker.
- ПО не использует лицензионные ключи и не требует доступа к сети Интернет для работы, за исключением узлов процессов, обращающихся к внешним сервисам, и подключения к внешнему сервису большой языковой модели, если оно настроено.
- Порядок работы с ПО описан в документе «Руководство пользователя программного обеспечения платформы автоматизации рабочих процессов «Кортекс Потоки»».