

Инструкция по установке программного обеспечения «INTEBRIX Optimization Platform»

Версия ПО: **1.0.0** (сборка 0ec5bd7d от 21.08.2026) · Правообладатель: ООО «ИНТЕБРИКС» (ИНН 9707059284) · Редакция документа: 19.09.2026

Полная инструкция по развёртыванию платформы из комплекта закрытого контура и в контуре с доступом в интернет — требования, состав комплекта, команды, переменные окружения, инициализация БД, проверка, запуск и остановка, обновление и откат, типовые проблемы, развёртывание проверочного экземпляра для экспертизы.

1. Назначение документа

Документ описывает порядок установки, первичной настройки, проверки работоспособности, запуска и остановки, обновления и отката программного обеспечения «INTEBRIX Optimization Platform» (далее — платформа, ПО).

Версия ПО, к которой относится документ: **1.0.0**, сборка из фиксации исходного кода 0ec5bd7d от 21.08.2026. Номер версии единый для всей поставки и задаётся файлом `VERSION` в корне дистрибутива; работающая установка сообщает его в ответе `GET /health`.

Документ рассчитан на системного администратора, владеющего Linux, Docker и PostgreSQL. Все приведённые команды выполняются в оболочке `bash` на целевом сервере, если не указано иное.

Штатный способ развёртывания — контейнерный, на базе Docker Compose. Основной путь поставки — комплект закрытого контура `intebrix-airgap-1.0.0.tar.gz` с готовыми образами контейнеров (раздел 5): он не требует доступа к внешним реестрам образов и репозиториям пакетов и одинаково применим в подключённом и изолированном контуре. Установка со сборкой образов из исходного дистрибутива (раздел 4) — вспомогательный путь для организаций, получивших исходный дистрибутив по договору. Ручная установка компонентов на хост (без контейнеров) в настоящем документе не рассматривается.

2. Требования к аппаратному и программному обеспечению

2.1. Аппаратные требования (одиночный сервер, все компоненты)

Параметр	Минимально	Рекомендуется
Процессор	2 ядра x86-64	8 и более ядер x86-64
Оперативная память	4 ГБ	16 ГБ и более
Дисковое пространство	30 ГБ SSD, из них не	200 ГБ и более SSD

Параметр	Минимально	Рекомендуется
	менее 15 ГБ свободно в каталоге данных Docker	
Сеть	100 Мбит/с	1 Гбит/с

Стенды установочных испытаний на российских операционных системах (4 vCPU, около 4 ГБ оперативной памяти, см. документ 05 «Протокол испытаний на российских операционных системах») удовлетворяют минимальной конфигурации. Установщик `scripts/install.sh` проверяет свободное место в каталоге данных Docker (порог 15 ГБ, переопределяется переменной `INTEBRIX_MIN_FREE_GB`; на Linux при нехватке места установка прерывается) и объём оперативной памяти (порог 4 ГБ, переменная `INTEBRIX_MIN_RAM_GB`; при нехватке выводится предупреждение).

Пояснения к требованиям:

- Расчётные задачи (решатели HiGHS и CBC) выполняются в контейнере `worker`; параллелизм по умолчанию — 2 задачи (`- - concurrency=2`), процесс перезапускается каждые 50 задач (`- - max-tasks-per-child=50`). Увеличение числа одновременных расчётов требует пропорционального увеличения ядер и памяти.
- API-сервис запускается под `Gunicorn` с 4 рабочими процессами (`-w 4`).
- Кэш и брокер очередей ограничены 256 МБ (`- - maxmemory 256mb - - maxmemory-policy allkeys-lru`).
- Дисковое пространство расходуется под образы контейнеров, том БД PostgreSQL, том пользовательских файлов и резервные копии. Только бинарные сборки решателя CBC (пакет `cbsbox`) занимают около 181 МБ внутри образов `api` и `worker`.

Архитектура процессора: целевая платформа — `linux/amd64` (x86-64). Комплект закрытого контура собирается сценарием `scripts/build-airgap.sh` именно под эту платформу (значение по умолчанию переменной `INTEBRIX_TARGET_PLATFORM`, сборка через `docker buildx`), а установщик при загрузке образов сверяет их архитектуру с архитектурой сервера и при несовпадении прерывает установку. Базовые образы (`python:3.12-slim`, `node:20-alpine`, `nginx:1.27-alpine`, `postgres:16-alpine`, `valkey/valkey:8-alpine`) многоархитектурные; сборка под `arm64` возможна для стендов разработки и в поставку не входит.

2.2. Требования к системному программному обеспечению

Компонент	Версия	Назначение
ОС	Astra Linux Special Edition 1.8.2 или РЕД ОС 8.0.2 (x86-64)	Хостовая система
Docker Engine	24.0 и выше	Среда исполнения контейнеров

Компонент	Версия	Назначение
Docker Compose	2.20 и выше (плагин docker compose)	Оркестрация состава сервисов
tar, gzip, sha256sum, awk, sed, grep	базовые пакеты дистрибутива	Распаковка комплекта, проверка контрольных сумм, работа установщика
curl	опционально	Проверки с хоста (при отсутствии установщик выполняет проверку изнутри контейнера)
openssl	опционально	Генерация секретов (при отсутствии установщик использует /dev/urandom)
gpg	опционально	Шифрование резервных копий

Поддерживаемые операционные системы — те, на которых проведены установочные испытания: **Astra Linux Special Edition 1.8.2 «Орёл»** (ядро 6.1.124, x86_64) и **РЕД ОС 8.0.2** (ядро 6.12.56, x86_64). Испытания проведены 18.08.2026; их ход и результаты изложены в документе 05 «Протокол испытаний на российских операционных системах». Средства контейнеризации входят в штатные репозитории обеих операционных систем, подключение сторонних источников не требуется. На Astra Linux устанавливаются пакеты `docker.io`, `docker-compose-v2` и `docker-buildx` (пакет с именем `docker-compose` содержит непригодную версию 1.29.2). Штатный набор автоматических проверок на обеих операционных системах дал совпадающий результат.

Иные дистрибутивы Linux x86-64 с Docker Engine 24 и выше и Docker Compose 2.20 и выше — по запросу, без гарантии: установочные испытания на них не проводились.

2.3. Компоненты, поставляемые в составе контейнеров

Отдельная установка этих компонентов на хост не требуется:

Компонент	Версия/образ
Python	3.12 (python:3.12-slim)
PostgreSQL	16 (postgres:16-alpine)
Valkey (кэш, брокер Celery)	8 (valkey/valkey:8-alpine)
nginx	1.27 (nginx:1.27-alpine)
Node.js	20 (node:20-alpine, только на этапе сборки фронтенда)
Решатель HiGHS	highspy 1.15.1 (версии

Компонент	Версия/образ
Решатель CBC	зафиксированы файлом backend/requirements.lock) через python-mip 1.17.6 (бинарные сборки — пакет cbcbox 2.929)

Решатели HiGHS и CBC — свободные, входят в состав образов `api` и `worker`.
Проприетарные решатели в поставку не входят и для работы не требуются;
предусмотренная в исходном коде необязательная интеграция с CPLEX по умолчанию
выключена и без установленного заказчиком пакета не задействуется (см. 6.3 и 8.2).

Замена образа `redis:*` на `valkey/valkey:8-alpine` выполнена намеренно
(лицензионные ограничения Redis 7/8). Возврат к образам Redis не допускается.

2.4. Требования к рабочему месту пользователя (браузер)

Веб-интерфейс — одностраничное приложение (React 19), собираемое Vite. Целевые
браузеры заданы списком `browserslist` в `package.json`:

```
> 0.5%
last 2 versions
Firefox ESR
not dead
not ie 11
```

Практически это означает поддержку двух последних версий Google Chrome, Microsoft
Edge, Mozilla Firefox (включая ветку ESR), Safari и браузеров на их основе (в том числе
«Яндекс Браузер»). Internet Explorer 11 не поддерживается.

Требуются включённые JavaScript и файлы cookie; для отображения хода расчётов
используется WebSocket-соединение.

Интерфейс работает в актуальных версиях браузеров на движке Chromium (Google
Chrome, Microsoft Edge, Яндекс Браузер) и в Mozilla Firefox. Автоматизированные
проверки интерфейса выполняются на движке Chromium.

2.5. Требования к сетевому доступу

Порты, публикуемые на хосте составом Docker Compose:

Порт хоста	Сервис	Доступность
80/tcp	frontend (nginx, веб- интерфейс и прокси к API)	внешний
127.0.0.1:5001/tcp	api (FastAPI)	только локально (петлевой интерфейс)
127.0.0.1:5433/tcp	postgres	только локально (петлевой интерфейс)

Порт хоста	Сервис	Доступность
127.0.0.1:6380/tcp	redis (образ Valkey)	только локально (петлевой интерфейс)

Единственный порт, открытый наружу, — 80/tcp контейнера frontend: все обращения к API из браузера и внешних систем проходят через nginx (пути /api/, /ws/), который применяет ограничение частоты запросов (заголовки безопасности для ответов API формирует само приложение). Привязка API, СУБД и кэша к 127.0.0.1 задана в docker-compose.yml намеренно: внутри сети Docker сервисы обращаются к API по имени api:5001, а публикация на петлевом интерфейсе хоста нужна только для служебных команд с самого сервера (curl http://localhost:5001/health/live, см. 8.2). Публиковать эти порты на внешний интерфейс не следует; при необходимости доступа к БД клиентом используется SSH-туннель:

```
ssh -L 5433:127.0.0.1:5433 <пользователь>@<адрес_сервера>
```

Рекомендуемая настройка межсетевого экрана на хосте — разрешить входящие соединения только на порты 22, 80 и 443. Настройка выполняется штатным средством операционной системы; пример для firewalld (РЕД ОС):

```
firewall-cmd --permanent --add-service=ssh
firewall-cmd --permanent --add-service=http
firewall-cmd --permanent --add-service=https
firewall-cmd --reload
```

На Astra Linux применяется межсетевой экран, входящий в состав операционной системы, с тем же набором разрешённых портов.

3. Состав дистрибутива

3.1. Исходный дистрибутив (сценарий со сборкой образов)

Исходный дистрибутив передаётся правообладателем по договору (архивом или доступом к репозиторию). Состав:

Путь	Содержимое
VERSION	Номер версии поставки (1.0.0)
docker-compose.yml	Состав из пяти сервисов: frontend, api, worker, postgres, redis (образ Valkey)
docker-compose.dev.yml	Состав для стенда разработки (не для промышленной эксплуатации)
docker-compose.glitchtip.yml	Опциональное наложение: локальный приёмник ошибок GlitchTip
docker/Dockerfile.api	Сборка образа API (Python 3.12,

Путь	Содержимое
	Gunicorn + Uvicorn)
<code>docker/Dockerfile.worker</code>	Сборка образа расчётного исполнителя (Celery)
<code>docker/Dockerfile.frontend</code>	Сборка образа веб-интерфейса (Node 20 → nginx 1.27)
<code>docker/nginx/default.conf</code>	Конфигурация nginx: проксирование <code>/api/</code> , <code>/ws/</code> , отдача статики, лимиты
<code>docker/nginx/security-headers.conf</code>	Заголовки безопасности и политика CSP
<code>backend/</code>	Серверная часть: <code>app/</code> (FastAPI), <code>alembic/</code> (миграции), <code>requirements.txt</code>
<code>apps/, packages/</code>	Исходный код веб-интерфейса (рабочие пространства <code>prn</code>)
<code>scripts/install.sh</code>	Установщик: предпроверки, генерация <code>.env</code> , запуск, миграции, начальное наполнение, дымовая проверка
<code>scripts/intebrix-install.sh</code>	Установка из комплекта одной командой: проверка машины, распаковка со сверкой полноты, проверка контрольных сумм, вызов <code>install.sh</code> в неинтерактивном режиме
<code>scripts/build-airgap.sh</code>	Сборка комплекта для закрытого контура
<code>scripts/update.sh</code>	Обновление версии с резервным копированием и автоматическим откатом
<code>scripts/backup.sh</code> , <code>scripts/restore.sh</code>	Резервное копирование и восстановление
<code>.env.example</code>	Образец файла параметров
<code>THIRD-PARTY-NOTICES.txt</code>	Перечень сторонних компонентов и тексты лицензий
<code>deployment/</code>	Примеры конфигурации мониторинга и правил оповещения (в комплект закрытого контура не входит)
<code>docs/</code>	Эксплуатационная документация

3.2. Комплект для закрытого контура

Формируется сценарием `scripts/build-airgap.sh` в виде архива `intebrix-airgap-<версия>.tar.gz`; для версии 1.0.0 — `intebrix-airgap-1.0.0.tar.gz`. Сценарий собирает образы под целевую платформу `linux/amd64`, сохраняет их в `images.tar`, добавляет файлы состава и сопровождения, формирует `README.md` с датой и хостом сборки, вычисляет контрольные суммы всех файлов (`SHA256SUMS`), упаковывает каталог без расширенных атрибутов файловой системы и сверяет полноту архива. Рядом с архивом (не внутри него) сценарий кладёт распаковщик `intebrix-install.sh`.

Состав архива (единственный каталог верхнего уровня, внутри него):

Файл	Назначение
<code>images.tar</code>	Образы <code>intebrix/frontend:1.0.0</code> , <code>intebrix/api:1.0.0</code> , <code>intebrix/worker:1.0.0</code> , <code>postgres:16-alpine</code> , <code>valkey/valkey:8-alpine</code> , сохранённые командой <code>docker save</code>
<code>docker-compose.yml</code>	Состав сервисов
<code>docker/</code>	Файлы сборки (<code>Dockerfile.api</code> , <code>Dockerfile.worker</code> , <code>Dockerfile.frontend</code>) и каталог <code>nginx/</code> с конфигурацией
<code>scripts/install.sh</code> , <code>backup.sh</code> , <code>restore.sh</code> , <code>update.sh</code> , <code>build-airgap.sh</code> <code>.env.example</code>	Сценарии установки и сопровождения Образец параметров с предустановленным <code>DEPLOY_MODE=air_gapped</code> и записанной сборщиком версией набора образов <code>INTEBRIX_VERSION=1.0.0</code>
<code>tiles/</code>	Каталог локального набора тайлов карты; в стандартной поставке содержит только <code>README.md</code> с порядком подготовки набора
<code>THIRD-PARTY-NOTICES.txt</code>	Уведомления о сторонних компонентах
<code>VERSION</code>	Номер версии комплекта (читается приложением и сценарием обновления)
<code>README.md</code>	Краткая памятка по установке, дата

Файл	Назначение
SHA256SUMS	и хост сборки Контрольные суммы всех файлов комплекта

Файлы, передаваемые вместе с архивом:

Файл	Назначение
intebrix-install.sh	Распаковщик и установщик одной командой (см. 5.6)
контрольная сумма SHA-256 архива	Сообщается отдельным каналом связи для проверки целостности самого архива

Исходный код в комплект не входит; сборка образов в закрытом контуре не требуется и не выполняется.

Порядок передачи: комплект передаётся согласованным с заказчиком каналом — защищённой ссылкой поставщика либо на носителе (для закрытого контура). Целостность подтверждается контрольными суммами SHA-256: перечень сумм по файлам входит в комплект (файл SHA256SUMS, проверка — `sha256sum -c SHA256SUMS`), контрольная сумма самого архива сообщается отдельным каналом связи и сверяется командой `sha256sum intebrix-airgap-1.0.0.tar.gz`. Для экспертизы при включении сведений о ПО в единый реестр российского программного обеспечения комплект версии 1.0.0 и файл с контрольной суммой архива размещены для скачивания на сайте правообладателя.

Комплект версии 1.0.0 (дата сборки 19.09.2026) собран из фиксации исходного кода `0ec5bd7d` без изменений в составе образов. Сценарий `scripts/build-airgap.sh` в составе комплекта отличается от той же фиксации одной строкой: команда `docker save` дополнена параметром `-platform linux/amd64`, необходимым для выгрузки образов целевой платформы при сборке на станции иной архитектуры. На состав образов и порядок установки это отличие не влияет.

4. Установка со сборкой образов из исходного дистрибутива (контур с доступом в интернет)

Путь применяется, когда организация получила исходный дистрибутив (3.1) и собирает образы на своём сервере; для сборки нужен доступ к реестру образов Docker, PyPI и npm. Установочные испытания этим путём проведены на Astra Linux SE 1.8.2 (документ 05). Для установки без сборки — из комплекта с готовыми образами — применяется раздел 5.

4.1. Подготовка операционной системы

Astra Linux SE 1.8.2 (пакеты штатного репозитория):

```
apt-get install docker.io docker-compose-v2 docker-buildx
systemctl enable --now docker
```

РЕД ОС 8.0.2 (пакеты штатного репозитория):

```
dnf install docker-ce docker-compose-plugin
systemctl enable --now docker
```

Отдельный непривилегированный пользователь для эксплуатации состава:

```
useradd -m intebrix
usermod -aG docker intebrix
```

Проверка среды:

```
docker --version          # ожидается 24.0 и выше
docker compose version    # ожидается 2.20 и выше
```

4.2. Получение дистрибутива

Исходный дистрибутив, переданный архивом, распаковывается в домашний каталог пользователя `intebrix`; далее команды выполняются из каталога дистрибутива (в нём лежат `docker-compose.yml`, `VERSION` и каталог `scripts/`). При предоставлении доступа к репозиторию правообладателя дистрибутив получается командой `git clone` по адресу, указанному в договоре.

4.3. Установка сценарием `install.sh`

```
./scripts/install.sh
```

Сценарий последовательно выполняет:

1. Предпроверки: наличие `docker` и `docker compose`, утилит `tar`, `gzip`, `sha256sum`, `awk`, `sed`, `grep`; занятость портов; свободное место в каталоге данных Docker (не менее 15 ГБ); объём оперативной памяти (рекомендуется от 4 ГБ).
2. Генерацию файла `.env` с правами 600. Секреты `JWT_SECRET_KEY`, `SESSION_SECRET`, `LICENSE_SECRET` формируются как 64 шестнадцатеричных символа, пароли `POSTGRES_PASSWORD` и `REDIS_PASSWORD` — как 48 шестнадцатеричных символов (`openssl rand -hex`, при отсутствии `openssl` — из `/dev/urandom`). Остальные значения запрашиваются интерактивно (режим развёртывания, наименование и краткое имя организации, электронная почта и имя администратора, пароль администратора, уровень лицензии, срок её действия, адрес приёмника ошибок, адрес поставщика OIDC).
3. Запуск состава: `docker compose up -d` (при первом запуске образы собираются из исходного кода).
4. Ожидание состояния `healthy` для сервисов `postgres`, `redis`, `api` (не более 120 секунд на сервис).
5. Применение миграций: `alembic upgrade head`.
6. Начальное наполнение БД, если таблица `users` пуста: `python -m app.scripts.seed`.

7. Дымовую проверку GET /health/live изнутри контейнера api (при наличии curl на хосте — дополнительно с хоста).

Пароль администратора выводится на экран один раз — его необходимо сохранить.

Неинтерактивный режим (все значения берутся по умолчанию, пароль администратора генерируется случайно):

```
./scripts/install.sh --non-interactive
```

Повторный запуск безопасен: существующий .env и данные БД сохраняются. Полная переустановка с удалением томов — ./scripts/install.sh --reset (все данные будут потеряны).

4.4. Установка вручную (эквивалент сценария)

```
cp .env.example .env
chmod 600 .env
${EDITOR:-vi} .env          # заполнить JWT_SECRET_KEY,
SEED_ADMIN_PASSWORD и др.
```

```
docker compose up -d --build # первая сборка образов занимает 5-10
минут
docker compose ps
```

```
docker compose exec -T api sh -c 'cd /app && alembic upgrade head'
docker compose exec -T api sh -c 'cd /app && python -m
app.scripts.seed'
```

Внимание: используется именно модуль app.scripts.seed. Файл /app/seed.py в корне образа устарел (не содержит принудительной фиксации записи администратора перед созданием ключей активации) и запускаться не должен.

4.5. Публикация по HTTPS (при необходимости)

Контейнер frontend слушает порт 80 по протоколу HTTP. Терминирование TLS выполняется внешним обратным прокси на хосте. Пример конфигурации nginx:

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

server {
    listen 443 ssl http2;
    server_name <домен>;

    ssl_certificate      /etc/letsencrypt/live/<домен>/fullchain.pem;
    ssl_certificate_key  /etc/letsencrypt/live/<домен>/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
```

```

location / {
    proxy_pass http://127.0.0.1:80;
    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 https;
    proxy_read_timeout 1200s;      # длительные расчёты
    proxy_send_timeout 60s;
}
}

```

Заголовок X-Real-IP обязателен: без него внутренний nginx видит все запросы как исходящие от шлюза сети Docker и применяет ограничение частоты запросов ко всем пользователям совместно.

После публикации по HTTPS следует привести в соответствие список разрешённых источников:

INTEBRIX_ALLOWED_ORIGINS=https://<домен>

(параметр добавляется в секцию `environment` сервиса `api` — см. раздел 6.3).

5. Установка из комплекта закрытого контура (основной путь)

Платформа не требует обращения к внешним сервисам для работы расчётных ядер, аутентификации, хранения и выгрузки данных. Все обязательные компоненты поставляются в составе образов контейнеров, поэтому установка из комплекта одинаково выполняется в изолированном контуре и в контуре с доступом в интернет. Установочные испытания этим путём проведены на РЕД ОС 8.0.2 (документ 05); обращений в сеть Интернет в ходе установки не выполнялось.

5.1. Шаг 1. Сборка комплекта правообладателем

Комплект собирается на рабочей станции правообладателя из исходного дистрибутива:

```

cd "<каталог_дистрибутива>"
./scripts/build-airgap.sh

```

Результат: `intebrix-airgap-1.0.0.tar.gz` в корне дистрибутива и распаковщик `intebrix-install.sh` рядом с ним. Сценарий собирает образы командой `docker compose build` под целевую платформу `linux/amd64` (через `docker buildx`, если архитектура сборочной станции иная), сохраняет их в `images.tar` (`docker save`), добавляет `docker-compose.yml`, каталог `docker/`, сценарии сопровождения, образец `.env` (с заменой `DEPLOY_MODE` на `air_gapped` и дописанной версией набора образов `INTEBRIX_VERSION`), файлы `VERSION` и `THIRD-PARTY-NOTICES.txt`, каталог `tiles/`, памятку `README.md` и файл контрольных сумм `SHA256SUMS`, после чего проверяет полноту архива. Базовые образы (`python`, `node`, `nginx`, `postgres`, `valkey`) нужны

только сборочной станции: заказчику и эксперту они передаются готовыми в составе `images.tar`, и внешние реестры образов при установке не требуются.

5.2. Шаг 2. Перенос комплекта

Архив передаётся на целевой сервер разрешённым в организации способом (съёмный носитель, однонаправленный шлюз, выделенный канал).

5.3. Шаг 3. Проверка целостности и распаковка

Сначала сверяется контрольная сумма архива с суммой, сообщённой правообладателем отдельным каналом, затем архив распаковывается и проверяются суммы файлов комплекта:

```
sha256sum intebrix-airgap-1.0.0.tar.gz
mkdir -p /opt/intebrix && tar -xzf intebrix-airgap-1.0.0.tar.gz -C
/opt/intebrix --strip-components=1
cd /opt/intebrix
sha256sum -c SHA256SUMS
```

Все строки должны завершаться словом OK. Каталог установки (`/opt/intebrix` в примере) выбирается администратором; в нём должны находиться `docker-compose.yml`, `.env` и каталог `scripts/`, так как сценарии сопровождения работают относительно него.

Установка одной командой (распаковка, проверка сумм, загрузка образов, запуск, миграции и наполнение выполняются автоматически, `.env` со случайными секретами создаётся установщиком):

```
./intebrix-install.sh intebrix-airgap-1.0.0.tar.gz /opt
```

Второй аргумент — каталог, внутри которого создаётся каталог установки с именем каталога верхнего уровня комплекта (по умолчанию — текущий каталог); итоговый путь, адрес и учётную запись администратора сценарий печатает по завершении. При таком пути шаги 5.4–5.6 выполняются сценарием; их описание ниже нужно для понимания результата и для установки по шагам. Отличия от установки по шагам: сценарий создаёт `.env` самостоятельно и не копирует `.env.example`, поэтому значения `COMPOSE_PROJECT_NAME` и `INTEBRIX_VERSION` в `.env` отсутствуют (имена образов берутся по умолчанию — `intebrix/<сервис>:1.0.0`, имя проекта Compose — по имени каталога), а режим развёртывания записывается как `on_prem`. Для закрытого контура после установки в `.env` задаётся `DEPLOY_MODE=air_gapped` и выполняется `docker compose up -d`. Если имена контейнеров и томов должны не зависеть от каталога, `.env` создаётся из `.env.example` по 5.5 до запуска установщика — существующий `.env` установщик сохраняет.

5.4. Шаг 4. Имена образов и имя проекта Docker Compose

Образы в комплекте сохранены под именами, заданными в `docker-compose.yml` явно ключом `image:` (`intebrix/api:${INTEBRIX_VERSION}`, `intebrix/frontend:${INTEBRIX_VERSION}`, `intebrix/worker:${INTEBRIX_VERSION}`), и от имени каталога не зависят. Для версии 1.0.0 это:

```
intebrix/api:1.0.0
intebrix/frontend:1.0.0
intebrix/worker:1.0.0
```

Номер версии подставляется из переменной `INTEBRIX_VERSION`, которую сборщик комплекта записывает в `.env.example`; менять её вручную не требуется. После `docker load -i images.tar` образы появляются под этими именами (проверка — `docker images | grep '^intebrix/'`), и Compose берёт их без перехода к сборке.

Имя проекта Compose зафиксировано в `.env.example` значением `COMPOSE_PROJECT_NAME=intebrix`, чтобы имена контейнеров и томов не зависели от имени каталога, в который распакован комплект; на имена образов оно не влияет. Задавать или менять его не требуется — значение переносится в `.env` вместе с остальным содержимым образа.

5.5. Шаг 5. Параметры установки

```
cp .env.example .env
chmod 600 .env
${EDITOR:-vi} .env
```

Обязательные значения для закрытого контура:

```
JWT_SECRET_KEY=<результат openssl rand -hex 32>
SESSION_SECRET=<результат openssl rand -hex 32>
POSTGRES_PASSWORD=<результат openssl rand -hex 24>
REDIS_PASSWORD=<результат openssl rand -hex 24>
DEPLOY_MODE=air_gapped
SEED_ORG_NAME=<наименование организации>
SEED_ORG_SLUG=<краткое имя латиницей>
SEED_ADMIN_EMAIL=<адрес администратора>
SEED_ADMIN_PASSWORD=<пароль не короче 12 символов>
SEED_ADMIN_NAME=<ФИО администратора>
SEED_TIER=enterprise
SEED_LICENSE_DAYS=365
SENTRY_DSN=
YANDEX_API_KEY=
YANDEX_FOLDER_ID=
```

Значение `air_gapped` пишется через подчёркивание. Вариант `air-gapped` приложением не принимается.

Пароли `POSTGRES_PASSWORD` и `REDIS_PASSWORD` должны состоять только из шестнадцатеричных символов: значения подставляются внутрь строк подключения, и символы `@`, `:`, `/`, `?`, `#` нарушили бы их разбор. Значения `COMPOSE_PROJECT_NAME=intebrix` и `INTEBRIX_VERSION=1.0.0` уже записаны в `.env.example` и переносятся в `.env` без изменений (см. 5.4). Необязательное значение `LICENSE_SECRET` требуется только при наполнении базы из подписанного файла начальных данных (см. 7.3); в проверке лицензионного документа оно не участвует.

5.6. Шаг 6. Загрузка образов и запуск

```
./scripts/install.sh --air-gap images.tar
```

Сценарий выполняет предпроверки (4.3, п. 1), загружает образы (`docker load -i images.tar`), сверяет архитектуру загруженного образа `intebrix/api:1.0.0` с архитектурой сервера (при несовпадении установка прерывается с указанием платформы, под которую нужен комплект), сохраняет существующий `.env` (создан на шаге 5.5), поднимает состав, дожидается готовности сервисов, применяет миграции, выполняет начальное наполнение БД и дымовую проверку `GET /health/live`.

Эквивалентная последовательность вручную:

```
docker load -i images.tar
docker images | grep '^intebrix/'
docker compose up -d
docker compose exec -T api sh -c 'cd /app && alembic upgrade head'
docker compose exec -T api sh -c 'cd /app && python -m
app.scripts.seed'
```

5.7. Вариант с внутренним реестром образов

Если в организации развёрнут внутренний реестр Docker, образы после загрузки помечаются и публикуются в нём. После `docker load -i images.tar` образы платформы появляются под теми именами, что заданы в `docker-compose.yml` явно (ключ `image:`), — `intebrix/api:1.0.0`, `intebrix/frontend:1.0.0`, `intebrix/worker:1.0.0`; номер версии в имени подставляется из переменной `INTEBRIX_VERSION`, которую сборщик комплекта записывает в `.env.example`.

```
docker load -i images.tar
```

```
for svc in api frontend worker; do
  docker tag intebrix/$svc:1.0.0 \
    registry.internal.local/intebrix/$svc:1.0.0
  docker push registry.internal.local/intebrix/$svc:1.0.0
done
```

```
# Базовые образы инфраструктуры
docker tag postgres:16-alpine
registry.internal.local/intebrix/postgres:16-alpine
docker tag valkey/valkey:8-alpine
registry.internal.local/intebrix/valkey:8-alpine
docker push registry.internal.local/intebrix/postgres:16-alpine
docker push registry.internal.local/intebrix/valkey:8-alpine
```

В `docker-compose.yml` для сервисов `api`, `worker`, `frontend` значение ключа `image:` (`intebrix/<сервис>:${INTEBRIX_VERSION:-1.0.0}`) заменяется на адрес образа во внутреннем реестре (`registry.internal.local/intebrix/<сервис>:1.0.0`), а образы `postgres` и `redis` переадресуются на внутренний реестр. Секция `build` при этом не задействуется: Compose берёт готовый образ по имени и к сборке не переходит. После этого штатно работает `docker compose pull`.

5.8. Отключение обращений к внешним сервисам

Все внешние интеграции опциональны и по умолчанию либо выключены, либо отключаются одним параметром.

Интеграция	Как отключается	Поведение после отключения
Приёмник ошибок (Sentry/GlitchTip)	SENTRY_DSN= (пусто). Дополнительно инициализация выполняется только при DEPLOY_MODE=cloud	Ошибки пишутся в локальный файл logs/client_errors.jsonl
YandexGPT (текстовые пояснения)	LLM_PROVIDER=none, пустые YANDEX_API_KEY и YANDEX_FOLDER_ID	Функции текстовых пояснений недоступны, расчёт не затрагивается
OSRM (матрица расстояний и времён)	По умолчанию выключен (INTEBRIX_OSRM_ENABLED=false); при DEPLOY_MODE=air_garage обращения выключаются принудительно. Собственный экземпляр в контуре подключается значениями INTEBRIX_OSRM_ENABLED=true и INTEBRIX_OSRM_BASE_URL=http://<внутренний_osrm>:5000	При отключении расстояния рассчитываются как ортодромия $\times 1.3$, время — по скорости INTEBRIX_OSRM_FALLBACK_SPEED_KMH (по умолчанию 50 км/ч)
Telegram-уведомления	INTEBRIX_TELEGRAM_BOT_TOKEN= (пусто)	Уведомления не отправляются
Единый вход OIDC	OIDC_ISSUER_URL= (пусто)	Используется встроенная аутентификация. При наличии внутреннего Keycloak — указать его адрес

Отдельно о картографической подложке. Поставщик растровых тайлов задаётся переменной времени запуска контейнера frontend — INTEBRIX_MAP_PROVIDER (в .env — MAP_PROVIDER, подставляется через docker-compose.yml). Допустимые значения:

yandex (по умолчанию), osm-ru, osm, 2gis, local. При старте контейнера сценарий `docker/nginx/runtime-config.sh` записывает значение в файл `runtime-config.js`, который веб-интерфейс читает до загрузки приложения; значение времени запуска имеет приоритет над значением времени сборки `VITE_MAP_PROVIDER`. Пересборка образа не требуется. Значение `local` означает раздачу заранее подготовленного набора тайлов собственным `nginx` контейнера из каталога `/tiles/` (на хосте — каталог `./tiles` рядом с `docker-compose.yml`, монтируется томом только для чтения); внешних обращений при этом нет, а действующая политика CSP (`img-src 'self'`) такой раздаче не препятствует. Если набор тайлов не подготовлен, в закрытом контуре карта остаётся без фона: узлы, связи, маршруты и все расчёты работают в полном объёме, на отсутствующий тайл `nginx` отвечает пустым ответом без ошибок. Для отображения подложки в закрытом контуре:

1. Подготовить набор тайлов в раскладке `{z}/{x}/{y}.png` и поместить его в каталог `./tiles` (при сборке комплекта сценарий `build-airgap.sh` включает содержимое `tiles/` в архив, если оно подготовлено; порядок подготовки набора описан в `tiles/README.md`).
2. В `.env` указать `MAP_PROVIDER=local` и `MAP_LOCAL_MAX_ZOOM=<предельное приближение набора>` (значение должно совпадать с тем, до какого уровня отрисован набор; по умолчанию 14).
3. Перезапустить контейнер веб-интерфейса: `docker compose up -d frontend`.

Построение дорожных маршрутов на стороне сервера (матрица расстояний и времён) отключается или переводится на внутренний OSRM переменными `INTEBRIX_OSRM_ENABLED` и `INTEBRIX_OSRM_BASE_URL` (см. таблицу выше и 6.3). Обращение к OSRM из веб-интерфейса по умолчанию выключено: адрес задаётся только параметром сборки `VITE_OSRM_ENDPOINT` (см. 6.4), и в поставляемом образе он пуст.

Состав комплекта закрытого контура определён: картографические растровые подложки (кроме подготовленного заказчиком или поставщиком набора в `tiles/`) и сервис построения дорожных маршрутов в комплект не входят. Карта, узлы, связи и все расчёты работают в полном объёме без подложки; расстояния и длительности берутся из данных модели заказчика. Подключение собственного экземпляра сервиса маршрутизации в контуре заказчика возможно через переменные окружения; по умолчанию обращения к внешним сервисам отключены.

5.9. Лицензия в закрытом контуре

Проверка лицензии выполняется целиком внутри установки, без обращений к сети, и одинакова для всех режимов развёртывания: серверов активации, периодического сетевого подтверждения, исходящих контрольных обращений и удалённого управления лицензией в продукте нет. Предусмотрены два состояния лицензии.

Стартовая лицензия (переходный режим). При начальном наполнении БД (`install.sh` → `python -m app.scripts.seed`) в базе создаётся запись лицензии с параметрами `SEED_TIER`, `DEPLOY_MODE`, `SEED_LICENSE_DAYS`, `SEED_MAX_USERS` и полным набором расчётных ядер. Подписанного лицензионного документа у неё нет: права берутся из полей таблицы, и каждая такая проверка сопровождается предупреждением в журнале

серверной части (сервисы `api` и `worker`) о необходимости внести документ правообладателя. Режим позволяет ввести установку в эксплуатацию до получения лицензионного документа.

Лицензионный документ (штатный режим). Правообладатель выпускает подписанный лицензионный документ формата INTEBRIX2 — одну строку вида INTEBRIX2.<kid>.<данные>.<подпись>. Поле <данные> — JSON в кодировке `base64url`: продукт и редакция сборки, которым адресован документ, идентификатор организации, идентификатор лицензии, уровень, режим развёртывания, предел пользователей, перечень расчётных ядер и возможностей, даты выпуска и окончания действия, срок поддержки, длительность льготного периода, при необходимости — аппаратный отпечаток. Поле <подпись> — подпись Ed25519 закрытым ключом правообладателя; <kid> — идентификатор ключа (`prod-2026-08`). Открытые ключи проверки вшиты в код серверной части (`backend/app/services/license_keys.py`) и из настроек или базы данных не читаются; закрытого ключа на установке нет, поэтому установка умеет только проверять документ, но не выпускать его. Документ не является секретом и передаётся любым согласованным каналом — файлом на носителе или по электронной почте; строка не содержит переносов, пробелов и символа `=`.

Внесение документа выполняет администратор установки методом `POST /api/licensing/import` (тело запроса — `{"license_data": "<строка документа>"}`, требуется роль `admin`). До любой записи в базу проверяется подпись; затем идентификатор организации из документа сверяется с организацией вносящего — документ, адресованный другой организации, не принимается. Внесение заменяет действующую лицензию, а не добавляет новую; изменения вступают в силу сразу, без перезапуска сервисов. При каждой проверке прав, в том числе при каждом обращении к расчётным ядрам, документ перечитывается: сверяются продукт и редакция сборки и, если правообладатель включил в документ аппаратный отпечаток (только для режимов `on_prem` и `air_gapped`), — отпечаток машины. Документ, адресованный другой сборке или другой машине, считается недействительным. Сведения о машине, по которым вычисляется отпечаток (имя узла, платформа, архитектура, процессор), администратор получает методом `GET /api/licensing/hardware` и сообщает правообладателю до выпуска привязанной лицензии.

Срок действия. По истечении даты окончания начинается льготный период (`grace_period_days`, по умолчанию 7 дней): все функции продолжают работать, в журнал пишется предупреждение о необходимости продления. По окончании льготного периода обращения к расчётным ядрам отклоняются с кодом 403 до внесения нового документа. Допускается бессрочная лицензия (без даты окончания); истечение срока поддержки на работоспособность не влияет. Продление, добавление ядра и изменение лимитов выполняются выпуском нового документа и его внесением тем же методом.

Выгрузка. `GET /api/licensing/export` (роль `admin`) возвращает ровно тот документ, который был внесён, в виде файла `intebrix-<первые восемь символов идентификатора лицензии>.license`; ничего заново не подписывается. Метод предназначен для переноса действующей лицензии на другой узел и передачи в поддержку. Для стартовой лицензии без документа метод возвращает код 409 с указанием запросить документ у правообладателя.

6. Параметры конфигурации

Секреты в примерах приведены в виде плейсхолдеров. Реальные значения формируются на месте установки и в документацию не выносятся.

6.1. Файл `.env` (подстановка значений в `docker-compose.yml`)

Имя	Назначение	По умолчанию	Обязательность
<code>JWT_SECRET_KEY</code>	Ключ подписи маркеров доступа (не менее 32 символов)	<code>default-dev-secret-change-me</code>	Обязателен. При <code>INTEBRIX_ENVIRONMENT=production</code> значение по умолчанию отвергается, API не стартует
<code>SESSION_SECRET</code>	Ключ подписи серверных сессий (используется механизмом OIDC)	пусто (используется <code>JWT_SECRET_KEY</code>)	Обязателен при включённом OIDC
<code>LICENSE_SECRET</code>	Ключ HMAC-SHA256 для проверки подписи файла начальных данных <code>seed_data.json</code> (см. 7.3); передаётся в контейнер <code>api</code> как <code>INTEBRIX_LICENSE_SECRET</code> . В проверке лицензионного документа не участвует: документ подписывается асимметрично ключом правообладателя (см. 5.9)	пусто (используется <code>JWT_SECRET_KEY</code>)	Не обязателен; нужен только при наполнении из подписанного файла

Имя	Назначение	По умолчанию	Обязательность
POSTGRES_PASSWORD	Пароль пользователя СУБД <code>intebrix</code> ; подставляется в строки подключения сервисов <code>api</code> и <code>worker</code> (только шестнадцатеричные символы)	нет	Обязателен , без него состав не запускается; для уже созданной базы — значение, с которым база создавалась (см. 6.2)
REDIS_PASSWORD	Пароль кэша <code>Valkey</code> ; подставляется в строки подключения <code>api</code> и <code>worker</code> (только шестнадцатеричные символы)	нет	Обязателен , без него состав не запускается
DEPLOY_MODE	Режим развёртывания: <code>cloud</code> , <code>on_prem</code> , <code>air_gapped</code>	<code>cloud</code>	Обязателен для локальных установок
SEED_ORG_NAME	Наименование первой организации	<code>Intebrix Demo</code>	Рекомендуется задать
SEED_ORG_SLUG	Краткое имя организации (строчные латинские буквы, без пробелов)	<code>intebrix-demo</code>	Рекомендуется задать
SEED_ADMIN_EMAIL	Адрес электронной почты администратора (он же логин)	<code>admin@intebrix.ru</code>	Рекомендуется задать
SEED_ADMIN_PASSWORD	Пароль администратора	пусто	Обязателен при <code>INTEBRIX_ENVIRONMENT=production</code> ; значение <code>demo123</code>

Имя	Назначение	По умолчанию	Обязательность
			отвергается
SEED_ADMIN_NAME	Отображаемое имя администратора	Admin	Не обязателен
SEED_TIER	Уровень лицензии: free, professional, enterprise	enterprise	Не обязателен
SEED_MAX_USERS	Предельное число пользователей организации	20	Не обязателен
SEED_LICENSE_DAYS	Срок действия лицензии, дней	365	Не обязателен
SEED_ACTIVATION_KEYS	Количество предгенерируемых ключей активации	5	Не обязателен
RATELIMIT_TRUSTED_IPS	Префиксы адресов, освобождённых от ограничения частоты запросов (через запятую)	127.0.0.1	Не обязателен
LLM_PROVIDER	Поставщик текстовых пояснений: yandex или none	yandex	В закрытом контуре — none
YANDEX_API_KEY, YANDEX_FOLDER_ID	Реквизиты доступа к YandexGPT	пусто	Только при LLM_PROVIDER=yandex
YANDEX_STATIC_API_KEY	Ключ статических карт (используется серверной частью, в браузер не передаётся)	пусто	Не обязателен
SENTRY_DSN	Адрес приёмника	пусто (отчётность)	Не обязателен

Имя	Назначение	По умолчанию	Обязательность
	ошибок, совместимого с протоколом Sentry	отключена)	
SENTRY_TRACES _SAMPLE_RATE	Доля трассируемых запросов	0.1	Не обязателен
OIDC_ISSUER_U RL	Адрес поставщика единого входа	пусто (единый вход отключён)	Не обязателен
OIDC_CLIENT_I D, OIDC_CLIENT_S ECRET	Реквизиты клиента OIDC	пусто	Только при включённом OIDC
OIDC_REDIRECT _URI	Адрес возврата после аутентификации	пусто	Только при включённом OIDC
OIDC_POST_LOG IN_URL	Адрес перехода в интерфейсе после успешного входа	пусто	Не обязателен
OIDC_PROVISIO N_USERS	Автоматическое создание пользователя при первом входе через OIDC	true	Не обязателен
IDLE_TIMEOUT_ MINUTES	Автоматический выход при бездействии, минут (0 — отключено)	30	Не обязателен
MAP_PROVIDER	Поставщик картографическо й подложки: yandex, osm- ru, osm, 2gis, local; передаётся в контейнер frontend как INTEBRIX_MAP_ PROVIDER (см.	пусто (действует yandex)	В закрытом контуре — local при наличии набора тайлов

Имя	Назначение	По умолчанию	Обязательность
MAP_LOCAL_MAX_ZOOM	5.8) Предельное приближение локального набора тайлов; передаётся в контейнер frontend как INTEBRIX_MAP_LOCAL_MAX_ZOOM	пусто (действует 14)	Только при MAP_PROVIDER=local
INTEBRIX_VERSION	Версия набора образов; подставляется в имена intebrix/<сервис>:<версия> (см. 5.4); записывается сборщиком комплекта	1.0.0	Задан в комплекте, не менять
COMPOSE_PROJECT_NAME	Имя проекта Docker Compose (определяет имена контейнеров и томов; имена образов заданы в docker-compose.yml явно)	intebrix (из .env.example)	Задан в образце, не менять

6.2. Примечание о пароле СУБД

Строки подключения сервисов api и worker формируются в docker-compose.yml из значения POSTGRES_PASSWORD:

```
INTEBRIX_DATABASE_URL: postgresql://intebrix:${POSTGRES_PASSWORD}
@postgres:5432/intebrix
```

Поэтому одно значение в .env задаёт пароль и для СУБД, и для подключающихся к ней сервисов; отдельной правки строк подключения или исходного кода не требуется. Образ PostgreSQL применяет POSTGRES_PASSWORD только при первичной инициализации тома pgdata: для уже созданной базы значение в .env должно совпадать с тем, с которым база создавалась. Смена пароля на действующей базе выполняется командой ALTER USER

intebrix PASSWORD '<новый пароль>' с последующим изменением POSTGRES_PASSWORD в .env и пересозданием сервисов api и worker.

6.3. Параметры приложения (префикс INTEBRIX_)

Значения читаются процессом внутри контейнера. Задаются в секции `environment` соответствующего сервиса в `docker-compose.yml`. Файл `.env` в корне дистрибутива используется Docker Compose только для подстановки значений и **не передаётся** в контейнер автоматически: чтобы задать параметр из таблицы ниже, его нужно добавить в секцию `environment` (либо подключить `env_file: .env`).

Имя	Назначение	По умолчанию	Обязательность
INTEBRIX_ENVIRONMENT	Режим работы: <code>development</code> , <code>staging</code> , <code>production</code>	<code>development</code> (в составе Compose — <code>production</code>)	Задан в составе
INTEBRIX_DEBUG	Отладочный режим (раскрывает трассировки)	<code>false</code>	Оставить <code>false</code>
INTEBRIX_ENABLE_API_DOCS	Публикация <code>/docs</code> , <code>/redoc</code> , <code>/openapi.json</code>	<code>false</code>	Не обязателен
INTEBRIX_DATABASE_URL	Строка подключения к PostgreSQL	<code>postgresql://intebrix:intebrix@localhost:5432/intebrix</code>	Задан в составе
INTEBRIX_REDIS_URL	Адрес кэша	<code>redis://localhost:6379/0</code>	Задан в составе
INTEBRIX_CELERY_BROKER_URL	Брокер очереди задач	<code>redis://localhost:6379/0</code>	Задан в составе
INTEBRIX_CELERY_RESULT_BACKEND	Хранилище результатов задач	<code>redis://localhost:6379/1</code>	Задан в составе
INTEBRIX_HOST, INTEBRIX_PORT	Адрес и порт прослушивания	<code>0.0.0.0, 5001</code>	Не обязательны
INTEBRIX_WORKERS	Число рабочих процессов	<code>4</code>	Не обязателен
INTEBRIX_SOLVER_NAME	Унаследованный параметр; выбор решателя для расчётных ядер	<code>CBC</code>	Не задавать

Имя	Назначение	По умолчанию	Обязательность
	выполняется автоматически (HiGHS, при его отказе — CBC). Значение CPLEX относится к не входящему в поставку внешнему решателю и без установленного заказчиком пакета не действует		
INTEBRIX_SOLVER_TIMEOUT	Резервное предельное время расчёта ядра объёмного планирования, секунд (применяется только при отсутствии бюджета времени в запросе)	1200	Не обязателен
INTEBRIX_QUEUE_TIME_CEILING_SEC	Потолок бюджета времени расчёта, секунд: значение сверх потолка приводится к нему	600	Не обязателен
INTEBRIX_MAX_PAYLOAD_MB	Предельный размер тела запроса, МБ	100	Не обязателен
INTEBRIX_SOLVER_RUNS_RETENTION_MAX	Предельное число хранимых записей о расчётах (0 — без ограничения)	0	Рекомендуется задать для долгоживущих установок
INTEBRIX_OSRM_ENABLED	Использование сервиса дорожной	false	По умолчанию выключен; включать только

Имя	Назначение	По умолчанию	Обязательность
	маршрутизации OSRM (в .env — OSRM_ENABLED); при DEPLOY_MODE=air_gapped обращения выключаются принудительно		при наличии собственного экземпляра OSRM в контуре
INTEBRIX_OSRM_BASE_URL	Адрес собственного экземпляра OSRM (в .env — OSRM_BASE_URL); применяется только при INTEBRIX_OSRM_ENABLED=true	не задан (маршрутизация выключена)	Только при включённой маршрутизации
INTEBRIX_OSRM_FALLBACK_SPEED_KMH	Расчётная скорость при недоступности OSRM, км/ч	50.0	Не обязателен
INTEBRIX_ACCESS_TOKEN_EXPIRE_MINUTES	Срок жизни маркера доступа, минут	60	Не обязателен
INTEBRIX_REFRESH_TOKEN_EXPIRE_DAYS	Срок жизни маркера обновления, суток	30	Не обязателен
INTEBRIX_ALLOWED_ORIGINS	Разрешённые источники запросов (через запятую)	http://localhost:5170,http://localhost:5173,http://localhost:80	Задать при публикации по доменному имени
INTEBRIX_RATE_LIMIT_DEFAULT	Общее ограничение с одного адреса, запросов в минуту (совокупно на все рабочие	400	Не обязателен

Имя	Назначение	По умолчанию	Обязательность
	процессы)		
INTEBRIX_RATE_LIMIT_SOLVER	Ограничение на запуск расчётов и обращения к языковой модели с одного адреса, запросов в минуту	40	Не обязателен
INTEBRIX_RATE_LIMIT_PER_USER_PER_HOUR	Ограничение на пользователя, запросов в час	4000	Не обязателен
INTEBRIX_RATE_LIMIT_PER_USER_SOLVER_PER_HOUR	Ограничение на запуск расчётов на пользователя, запросов в час	240	Не обязателен
INTEBRIX_LOG_LEVEL	Уровень протоколирования	INFO	Не обязателен
INTEBRIX_LOG_JSON	Структурированный формат журнала	true	Не обязателен
INTEBRIX_DATA_DIR	Каталог пользовательских файлов	user_data	Не обязателен
INTEBRIX_CSP_EXTRA, INTEBRIX_CSP_OVERRIDE	Дополнение и полная замена политики CSP	пусто	Не обязательны

Сверх ограничений приложения nginx контейнера frontend применяет собственные ограничения частоты по адресу клиента: 600 запросов в минуту на пути /api/ и 10 запросов в минуту на пути /api/auth/ (с допуском кратковременных всплесков); при превышении возвращается код 429.

Полный перечень параметров с типами и значениями по умолчанию — backend/app/config.py. Объявленные там параметры license_server_url, license_file_path и heartbeat_* в проверке лицензии не участвуют и на поведение продукта не влияют: проверка выполняется целиком внутри установки по подписанному документу (см. 5.9).

6.4. Параметры сборки веб-интерфейса

Задаются на этапе сборки образа frontend (внедряются в собранный пакет и после сборки не меняются). Исключение — параметры карты, для которых предусмотрены

переменные времени запуска контейнера: они переопределяют значения сборки без пересборки образа (см. 5.8).

Имя	Назначение	По умолчанию
VITE_API_URL	Базовый адрес API	пусто (обращения идут на тот же узел через nginx)
VITE_MAP_PROVIDER	Поставщик тайлов по умолчанию для сборки: yandex, osm-ru, osm, 2gis, local. На работающей установке переопределяется переменной времени запуска INTEBRIX_MAP_PROVIDER (MAP_PROVIDER в .env)	yandex
VITE_MAP_LOCAL_MAX_ZOOM	Предельное приближение локального набора тайлов для сборки; переопределяется переменной времени запуска INTEBRIX_MAP_LOCAL_MAX_ZOOM (MAP_LOCAL_MAX_ZOOM в .env)	14
VITE_OSRM_ENDPOINT	Адрес OSRM для построения дорожных маршрутов из веб-интерфейса. Задаётся только при сборке; переменной времени запуска нет. При пустом значении обращения к OSRM из интерфейса не выполняются, маршруты оцениваются по ортодромии	пусто (выключено)

Имя	Назначение	По умолчанию
VITE_SENTRY_DSN	Адрес приёмника клиентских ошибок	пусто
VITE_ENABLE_SENTRY_SDK	Включение библиотеки отчётности в сборку	не задано (библиотека исключается)
VITE_APP_VERSION	Номер версии, отображаемый в интерфейсе	dev
VITE_SOURCEMAP	Генерация карт исходного кода	false
VITE_DEMO_WATERMARK	Демонстрационная подложка в интерфейсе	включена; отключается значением false

7. Инициализация базы данных, миграции и создание первой организации

7.1. Схема базы данных

Схема создаётся системой миграций Alembic. Цепочка ревизий:

```
001_baseline → 002_audit_logs_anonymize → 003_dispatch_days → 004_license_grant
```

Применение:

```
docker compose exec -T api sh -c 'cd /app && alembic upgrade head'
```

Проверка текущей ревизии:

```
docker compose exec -T api sh -c 'cd /app && alembic current'
docker compose exec -T api sh -c 'cd /app && alembic history'
```

Адрес БД для миграций берётся из INTEBRIX_DATABASE_URL (файл backend/alembic/env.py переопределяет sqlalchemy.url во время выполнения), поэтому один и тот же alembic.ini пригоден и внутри контейнера, и на хосте.

Если база была создана вручную до внедрения Alembic, вместо повторного применения миграций выполняется фиксация текущего состояния:

```
docker compose exec -T api sh -c 'cd /app && alembic stamp head'
```

7.2. Создание первой организации и администратора

```
docker compose exec -T api sh -c 'cd /app && python -m app.scripts.seed'
```

Операция создаёт:

- организацию (наименование SEED_ORG_NAME, краткое имя SEED_ORG_SLUG);
- лицензию (уровень SEED_TIER, режим DEPLOY_MODE, срок SEED_LICENSE_DAYS, предел пользователей SEED_MAX_USERS) с включёнными расчётными ядрами dispatcher, process-scheduler, strategic-production, network-intelligence, pipeline;
- учётную запись администратора (SEED_ADMIN_EMAIL, пароль SEED_ADMIN_PASSWORD, хеширование bcrypt);
- ключи активации в количестве SEED_ACTIVATION_KEYS для регистрации остальных пользователей.

Операция идемпотентна: при наличии организации с тем же кратким именем повторный запуск не выполняет изменений.

Ограничения в промышленном режиме (INTEBRIX_ENVIRONMENT=production):

- пустое значение SEED_ADMIN_PASSWORD — запуск прерывается с сообщением о необходимости задать пароль;
- значение demo123 — запуск прерывается.

7.3. Альтернативный способ — подписанный файл начальных данных

```
docker compose exec -T api sh -c 'cd /app && python -m
app.scripts.seed --file /app/seed_data.json'
```

Файл `seed_data.json` содержит те же сведения, что и переменные `SEED_*` (организация, администратор, параметры стартовой лицензии, число ключей регистрации), и поле `signature` — подпись HMAC-SHA256 по содержимому файла ключом `INTEBRIX_LICENSE_SECRET` (при пустом значении — `INTEBRIX_JWT_SECRET_KEY`) той установки, на которой файл будет применён. При несовпадении подписи загрузка отклоняется; файл без подписи принимается только в режиме `INTEBRIX_ENVIRONMENT=development`, в промышленном режиме — отклоняется. Файл формируется из текущих переменных окружения командой:

```
docker compose exec -T api sh -c 'cd /app && python -m
app.scripts.seed --generate-seed-file /app/seed_data.json'
```

Способ применяется, когда набор параметров первой организации готовится заранее и переносится между узлами с одинаковыми секретами (например, при повторной установке или переносе на резервный узел). Созданная таким образом лицензия — стартовая, без подписанного лицензионного документа (см. 5.9); лицензионный документ правообладателя вносится отдельно методом `POST /api/licensing/import`.

8. Проверка работоспособности после установки

8.1. Состояние контейнеров

```
docker compose ps
```

Ожидаемый результат — пять сервисов в состоянии `running (healthy)`. Контроль состояния настроен так (проверки `frontend`, `api`, `postgres`, `redis` заданы в `docker-compose.yml`, проверка `worker` — в файле сборки образа):

Сервис	Проверка	Периодичность
<code>frontend</code>	<code>wget -q --spider http://127.0.0.1/nginx-health</code>	30 с
<code>api</code>	HTTP-запрос <code>http://localhost:5001/health</code>	15 с
<code>worker</code>	<code>celery -A app.workers.celery_app inspect ping</code>	30 с
<code>postgres</code>	<code>pg_isready -U intebrix</code>	5 с
<code>redis</code>	<code>valkey-cli -a <пароль> ping</code>	5 с

8.2. Проверка адресов

Через порт 80 (веб-интерфейс и прокси `nginx`) снаружи доступны страницы интерфейса, пути `/api/`, `/ws/`, `/health`, `/health/live`, `/health/ready` и `/nginx-health`. Порт 5001 опубликован только на петлевом интерфейсе, поэтому пути `/ready`, `/solvers` и `/metrics` (без префикса `/api/`) проверяются с самого сервера.

Адрес	Ожидаемый ответ
<code>http://<адрес_сервера>/</code>	Веб-интерфейс, HTTP 200, страница входа
<code>http://<адрес_сервера>/nginx-health</code>	ok, HTTP 200 (проверка <code>nginx</code> , не зависит от API)
<code>http://<адрес_сервера>/health</code>	<code>{"status": "ok", "service": "Intebrix API Gateway", "version": "1.0.0"}</code>
<code>http://<адрес_сервера>/health/live</code>	То же; подтверждает отзывчивость процесса API
<code>http://<адрес_сервера>/health/ready</code>	<code>{"status": "ok", "checks": {...}}</code> , HTTP 200. При деградации — HTTP 503
<code>http://<адрес_сервера>/api/metrics</code>	Метрики приложения в формате JSON (требуется авторизация)
<code>http://127.0.0.1:5001/ready</code> (с сервера)	То же, что <code>/health/ready</code> , напрямую к API
<code>http://127.0.0.1:5001/solvers</code> (с сервера)	Перечень доступных решателей

Адрес	Ожидаемый ответ
<code>http://127.0.0.1:5001/metrics</code> (с сервера)	Метрики в формате Prometheus, без авторизации

Команды (выполняются на сервере):

```
curl -fsS http://localhost/nginx-health
curl -fsS http://localhost/health
curl -fsS http://localhost/health/ready
curl -fsS http://127.0.0.1:5001/solvers
```

Ответ `/health/ready` содержит разбивку по компонентам: `api`, `postgres`, `redis`, `celery`, а также по решателям (`solver_CBC`, `solver_HiGHS`, `solver_CPLEX`). Штатное состояние поставки — `solver_HiGHS` и `solver_CBC` доступны, `solver_CPLEX = false`: этот решатель в поставку не входит, и его отсутствие на итоговое состояние готовности не влияет.

Адреса `/docs`, `/redoc`, `/openapi.json` по умолчанию возвращают HTTP 404 — интерактивная документация API выключена и на уровне приложения (`INTEBRIX_ENABLE_API_DOCS=false`), и на уровне `nginx`. Для проведения аудита её включают параметром `INTEBRIX_ENABLE_API_DOCS=true` и снятием блокировки в `docker/nginx/default.conf`.

8.3. Проверка входа

```
curl -sS -X POST http://localhost/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"<адрес_администратора>","password":"<пароль>"}' | jq .
```

Ожидается ответ, содержащий поле `access_token`.

8.4. Проверка расчётного контура

1. Войти в веб-интерфейс под учётной записью администратора.
2. Убедиться, что в боковом меню присутствуют разделы всех расчётных ядер, включённых лицензией.
3. Загрузить демонстрационный набор данных и выполнить расчёт; убедиться, что задача переходит в состояние выполнения и возвращает результат.
4. Проверить журнал исполнителя расчётов:

```
docker compose logs -f worker --tail 100
```

8.5. Проверка журналов на предмет ошибок

```
docker compose logs api --since 10m | grep -i -E "error|warning|
  traceback"
docker compose logs worker --since 10m | grep -i -E "error|warning"
```

На установке со стартовой лицензией без подписанного лицензионного документа (см. 5.9) в журналах `api` и `worker` присутствуют предупреждения о необходимости внести документ правообладателя; после внесения документа они прекращаются.

8.6. Запуск, остановка и перезапуск состава

Все команды выполняются из каталога установки (там, где лежат `docker-compose.yml` и `.env`).

Действие	Команда	Результат
Запуск	<code>docker compose up -d</code>	Создаёт отсутствующие контейнеры и запускает состав в фоновом режиме; уже работающие контейнеры не пересоздаются
Состояние	<code>docker compose ps</code>	Перечень контейнеров, их состояние и опубликованные порты
Журналы	<code>docker compose logs -f api</code> (аналогично <code>worker</code> , <code>frontend</code> , <code>postgres</code> , <code>redis</code>)	Вывод журнала сервиса; <code>--since 10m</code> — за последние 10 минут, <code>--tail 100</code> — последние 100 строк
Остановка	<code>docker compose stop</code>	Останавливает контейнеры, сохраняя их и тома; данные не затрагиваются
Возобновление	<code>docker compose start</code>	Запускает ранее остановленные контейнеры
Перезапуск	<code>docker compose restart api</code> (или без имени сервиса — весь состав)	Перезапуск сервиса без пересоздания контейнера
Пересоздание после смены параметров	<code>docker compose up -d --force-recreate api worker</code>	Применяет изменённые значения <code>.env</code> к указанным сервисам
Удаление контейнеров	<code>docker compose down</code>	Останавливает и удаляет контейнеры и сеть; тома <code>pgdata</code> , <code>redisdata</code> , <code>user_data</code> , <code>api_logs</code> , <code>worker_logs</code> сохраняются, данные

Действие	Команда	Результат
		восстанавливаются при следующем <code>docker compose up -d</code>
Полное удаление	<code>docker compose down -v</code>	Удаляет также тома — база данных, пользовательские файлы и журналы теряются; применяется только для переустановки (см. 10.16)

Порядок корректной остановки при выполняющихся расчётах: убедиться в разделе «История решений» веб-интерфейса или командой `docker compose logs --tail 20 worker`, что активных заданий нет (при необходимости отменить их через интерфейс), затем выполнить `docker compose stop`. Расчёт, прерванный остановкой контейнера `worker`, не завершается и после запуска состава автоматически не возобновляется — его следует запустить заново.

Автоматический запуск после перезагрузки сервера обеспечивается политикой `restart: unless-stopped` в `docker-compose.yml`: контейнеры поднимаются вместе со службой Docker, если не были остановлены вручную командой `docker compose stop`.

9. Обновление версии и откат

9.1. Резервное копирование (выполняется до любого обновления)

`./scripts/backup.sh`

Формируется архив `backups/intebrix-<метка_времени>.tar.gz`, содержащий:

- логический дамп PostgreSQL в специальном формате (`pg_dump -Fc -Z 9`);
- журналы формата JSONL за последние 30 суток;
- файл `manifest.json` с версией, меткой времени и именем узла.

Шифрование резервной копии (для хранения вне контура):

```
INTEBRIX_BACKUP_GPG_RECIPIENT=<идентификатор_ключа>
./scripts/backup.sh
# результат: backups/intebrix-<метка>.tar.gz.gpg
```

Автоматическое ежедневное копирование:

```
# /etc/cron.d/intebrix-backup
0 3 * * * intebrix cd /opt/intebrix && ./scripts/backup.sh
```

9.2. Обновление в закрытом контуре

```
# 1. На технологическом узле собрать новый комплект
./scripts/build-airgap.sh --version 1.0.1

# 2. Перенести и проверить целостность на целевом сервере
sha256sum -c SHA256SUMS

# 3. Обновить
./scripts/update.sh --air-gap images.tar
```

Порядок действий сценария `update.sh`:

1. Резервное копирование (запоминается как точка отката).
2. Загрузка новых образов (`docker load -i`).
3. Остановка `api` и `worker`; `postgres` продолжает работать.
4. Запуск `api` на новом образе, ожидание ответа `/health/live` (до 60 секунд), применение `alembic upgrade head`.
5. Пересоздание `api` и `worker` (`docker compose up -d --force-recreate`).
6. Контрольная проверка `/health/live` (до 90 секунд).

Ход обновления протоколируется в `logs/update.log`.

9.3. Обновление установки, собранной из исходного дистрибутива

В `docker-compose.yml` для сервисов `api`, `worker` и `frontend` заданы и имя образа (`image: intebrix/<сервис>:${INTEBRIX_VERSION}`), и сборка из исходных текстов (`build`). Публичного реестра, откуда эти образы можно было бы получить, нет, поэтому команда `docker compose pull` для них завершается ошибкой и не применяется. Обновление из исходных текстов выполняется пересборкой после получения новой версии дистрибутива от правообладателя (архивом либо `git pull` при доступе к репозиторию); собранные образы получают те же имена:

```
./scripts/backup.sh
docker compose build
docker compose up -d --force-recreate api worker frontend
docker compose exec -T api sh -c 'cd /app && alembic upgrade head'
curl -fsS http://localhost:5001/health/live
```

Образы распространяются исключительно в составе комплекта поставки, что одинаково работает в подключённом и закрытом контурах. Штатный путь обновления — `./scripts/update.sh` с комплектом новой версии.

9.4. Откат

Автоматический откат встроен в `update.sh`: при отказе загрузки образов, миграции, запуска или контрольной проверки сценарий восстанавливает предобновленную резервную копию (`restore.sh <файл> --force`) и записывает исход в журнал. Отключается ключом `--no-rollback` (тогда стек остаётся в исходном состоянии для разбора).

Ручной откат:

```
./scripts/restore.sh backups/intebrix-<метка_времени>.tar.gz
```

Сценарий останавливает `api` и `worker`, пересоздаёт базу данных, восстанавливает дампы (`pg_restore --no-owner --jobs=4`), возвращает журналы, запускает сервисы, применяет миграции и проверяет `/health/live`. Зашифрованные копии (`.tar.gz.gpg`) распознаются по расширению и расшифровываются автоматически при наличии закрытого ключа на узле восстановления.

Полная переустановка (все данные теряются):

```
./scripts/backup.sh                                # если в базе есть значимые
данные
./scripts/install.sh --reset
./scripts/restore.sh backups/<файл>                # при необходимости вернуть
данные
```

10. Типовые проблемы и их устранение

10.1. Сервис `api` не запускается: ошибка ключа подписи

Признак. В журнале `api`: `JWT_SECRET_KEY must be set to a secure value in production`.

Причина. При `INTEBRIX_ENVIRONMENT=production` значение `JWT_SECRET_KEY` осталось равным значению по умолчанию.

Устранение. Задать в `.env` стойкое значение и пересоздать сервисы:

```
echo "JWT_SECRET_KEY=$(openssl rand -hex 32)" >> .env
docker compose up -d --force-recreate api worker
```

10.2. Начальное наполнение прерывается из-за пароля администратора

Признак. `INTEBRIX_SEED_ADMIN_PASSWORD is required in production` либо `Refusing to seed with ... 'demo123'`.

Причина. Защитная проверка: в промышленном режиме пустой или демонстрационный пароль не принимается.

Устранение. Задать `SEED_ADMIN_PASSWORD` (не менее 12 символов), пересоздать `api`, повторить наполнение.

10.3. Приложение отвергает режим развёртывания

Признак. `ValueError: 'on-prem' is not a valid DeployMode`.

Причина. Используется дефис вместо подчёркивания.

Устранение. Допустимые значения — `cloud`, `on_prem`, `air_gapped`. Исправить `.env`, выполнить `docker compose restart api`.

10.4. Ошибка при применении миграций на чистой базе

Признак. `ProgrammingError: type "subscriptiontier" already exists (либо userrole, licensestatus)`.

Причина. База была частично инициализирована ранней цепочкой миграций.

Устранение (только на пустой базе, данные теряются):

```
docker compose exec postgres psql -U intebrix -d intebrix \  
-c "DROP SCHEMA public CASCADE; CREATE SCHEMA public; \  
GRANT ALL ON SCHEMA public TO intebrix; \  
CREATE EXTENSION IF NOT EXISTS pgcrypto;"  
docker compose exec api sh -c 'cd /app && alembic upgrade head'  
docker compose exec api sh -c 'cd /app && python -m app.scripts.seed'
```

10.5. Ошибка внешнего ключа при наполнении

Признак. `Key (created_by_id)=(...) is not present in table "users"`.

Причина. Запущен устаревший сценарий `/app/seed.py` из корня образа.

Устранение. Использовать только `python -m app.scripts.seed`.

10.6. В закрытом контуре Compose пытается собрать образы

Признак. `failed to solve: failed to read dockerfile` или обращения к `PyPI/pypi` при `docker compose up -d`.

Причина. В `.env` отсутствует или искажена переменная `INTEBRIX_VERSION`, поэтому Compose ищет образ с другим тегом, либо образы из `images.tar` не загружены.

Устранение. Сверить значение `INTEBRIX_VERSION=1.0.0` в `.env` с тегом загруженных образов (`docker images | grep '^intebrix/'`, см. 5.4); при отсутствии образов выполнить `docker load -i images.tar` и повторить `docker compose up -d`.

10.7. Сервисы api и worker не стартуют после смены образа кэша

Признак. Контейнер `redis` циклически перезапускается; `api` и `worker` не поднимаются, ожидая состояния `service_healthy`.

Причина. Том `redisdata` содержит файлы журнала дозаписи (`appendonlydir`) и снимок (`dump.rdb`) в формате Redis 7; образ `valkey:8-alpine` их не читает. Проявляется при обновлении установок, развёрнутых до перехода на Valkey (зафиксировано 03.08.2026).

Устранение. Остановить состав, перенести внутри тома `appendonlydir` и `dump.rdb` в резервный каталог, запустить состав заново. Кэш и очередь восстанавливаются пустыми; данные в PostgreSQL не затрагиваются. Перед каждым `docker compose up -d` следует

просматривать `git diff` по `docker-compose.yml`, чтобы не применить незапланированную смену образа.

10.8. Конфликт портов при запуске

Признак. `bind: address already in use` для порта 80, 5001, 5433 или 6380.

Причина. Порт занят другой службой (нередко — системный `nginx` на 80).

Устранение. Остановить конфликтующую службу либо изменить публикуемые порты в `docker-compose.yml`. Учтите, что `install.sh` в предпроверках анализирует порты 80, 5001, 5432 и 6379, тогда как фактически публикуются 80, 5001, 5433 и 6380 — предупреждение сценария носит справочный характер.

10.9. Ошибка 413 при отправке крупного набора данных

Признак. `Solve failed: 413` либо `413 Request Entity Too Large`.

Причина. Превышен предел размера тела запроса. В `nginx` установлено `client_max_body_size 25m`; синхронные методы построения задачи дополнительно ограничены 250 заказами и 250 узлами.

Устранение. Использовать асинхронный запуск расчёта (постановка задачи в очередь), а для крупных импортов — загрузку файла через раздел данных. При необходимости увеличить `client_max_body_size` и `INTEBRIX_MAX_PAYLOAD_MB` согласованно.

10.10. Ошибка 504 при длительном расчёте

Признак. Обрыв соединения через несколько минут после запуска расчёта.

Причина. Внешний обратный прокси разрывает соединение раньше, чем завершается расчёт.

Устранение. Во всех прокси-слоях выставить `proxy_read_timeout 1200s` (значение, уже установленное во внутреннем `nginx`) и согласовать его с `INTEBRIX_SOLVER_TIMEOUT`.

10.11. Контейнер frontend в состоянии unhealthy

Признак. Веб-интерфейс открывается, но состояние контейнера — `unhealthy`.

Причина. Проверка обращается к `http://127.0.0.1/nginx-health`. Отказ означает либо ошибку конфигурации `nginx`, либо изменение адреса проверки.

Устранение. Проверить вручную:

```
docker compose exec frontend wget -q0- http://127.0.0.1/nginx-health
docker compose exec frontend nginx -t
```

10.12. Адрес /docs возвращает 404

Признак. По завершении `install.sh` выводится ссылка `http://localhost:5001/docs`, которая не открывается.

Причина. Интерактивная документация API отключена по умолчанию (и в приложении, и в nginx). Сообщение установщика не учитывает эту настройку.

Устранение. Для аудита включить `INTEBRIX_ENABLE_API_DOCS=true` в секции `environment` сервиса `api`, снять блокировку `location = /docs` в `docker/nginx/default.conf` и пересоздать сервисы. В промышленной эксплуатации документацию рекомендуется оставлять выключенной.

10.13. В закрытом контуре не отображается картографическая подложка

Признак. Карта без фона; при поставщике по умолчанию запросы к внешним узлам тайлов не выполняются или отклонены, при значении `local` ошибок в консоли браузера нет.

Причина. По умолчанию тайлы запрашиваются у внешнего поставщика, недоступного в закрытом контуре, либо не подготовлен локальный набор тайлов (см. 5.8).

Устранение. В `.env` указать `MAP_PROVIDER=local` и `MAP_LOCAL_MAX_ZOOM=<предельное приближение набора>`, поместить набор тайлов в каталог `./tiles` в раскладке `{z}/{x}/{y}.png` и перезапустить контейнер: `docker compose up -d frontend` (см. 5.8). Пересборка образа и изменение политики CSP не требуются. Если набор тайлов не подготовлен, карта остаётся без фона; расчётные функции сохраняют работоспособность.

10.14. Расчётные ядра отсутствуют в меню

Признак. После изменения уровня лицензии новый раздел не появляется.

Причина. Перечень отображаемых ядер хранится в настройках организации (`enabled_cores`) отдельно от уровня лицензии: уровень задаёт допустимое множество, а включение выполняет администратор.

Устранение.

```
docker compose exec postgres psql -U intebrix -d intebrix -c \  
    "SELECT slug, enabled_cores FROM organizations;"
```

```
curl -X PUT http://localhost:5001/api/admin/branding \  
    -H "Authorization: Bearer <маркер_доступа>" -H "Content-Type: \  
application/json" \  
    -d '{"enabled_cores":["dispatcher","process-scheduler","strategic- \  
production","network-intelligence","pipeline"]}'
```

10.15. Журналы приложения теряются при пересоздании контейнера

Признак. После `docker compose up -d --force-recreate api` каталог `/app/logs` пуст; резервная копия не содержит журналов за прошлые периоды.

Причина. В составе поставки до версии 1.0.0 для сервиса `api` монтировался только том `user_data`, а каталог `/app/logs` располагался внутри слоя контейнера и терялся при его пересоздании.

Устранение. Начиная с версии 1.0.0 тома для журналов входят в поставляемый `docker-compose.yml`: `api_logs` смонтирован в `/app/logs` сервиса `api`, `worker_logs` — в `/app/logs` сервиса `worker`. Журналы переживают пересоздание контейнеров и попадают в резервную копию. Для установок, развёрнутых из более ранних версий, тома добавляются при обновлении по `scripts/update.sh`.

10.16. Крайняя мера — полная переустановка

Если установка не восстанавливается перечисленными способами:

```
./scripts/backup.sh          # если данные значимы
./scripts/install.sh --reset # удаление томов, повторные миграции и
                             # наполнение
```

11. Контроль после установки: краткий перечень

- ❑ `JWT_SECRET_KEY`, `SESSION_SECRET`, `POSTGRES_PASSWORD`, `REDIS_PASSWORD` заданы стойкими значениями; файл `.env` имеет права `600`.
 - ❑ `INTEBRIX_ENVIRONMENT=production`, `INTEBRIX_DEBUG=false`.
 - ❑ Для уже созданной базы значение `POSTGRES_PASSWORD` совпадает с паролем, с которым база создавалась (см. 6.2).
 - ❑ Порты СУБД и кэша не опубликованы на внешнем интерфейсе.
 - ❑ `INTEBRIX_ALLOWED_ORIGINS` соответствует фактическому доменному имени.
 - ❑ Включён TLS на внешнем обратном прокси; передаётся заголовок `X-Real-IP`.
 - ❑ Межсетевой экран разрешает входящие соединения только на 22, 80, 443.
 - ❑ `/health/ready` возвращает HTTP 200; компоненты `api`, `postgres`, `redis`, `celery`, `solver_highs`, `solver_cbc` в состоянии `true` (`solver_cplex = false` — штатно).
 - ❑ `docker compose ps` показывает пять сервисов в состоянии `running (healthy)`.
 - ❑ Настроено регулярное резервное копирование, восстановление проверено на тестовом узле.
 - ❑ Для закрытого контура: `DEPLOY_MODE=air_gapped`, `SENTRY_DSN` пуст, `LLM_PROVIDER=none`, `INTEBRIX_OSRM_ENABLED=false` либо указан внутренний сервис.
 - ❑ Внесён лицензионный документ правообладателя (5.9); в журналах `api` и `worker` нет предупреждений о стартовой лицензии.
-