

Описание процессов жизненного цикла программного обеспечения «TurbineCare AI»

Версия ПО: **1.0.0** · Правообладатель: ООО «ИНТЕБРИКС» (ИНН 9707059284) · Редакция документа: 18.09.2026

Регламент жизненного цикла для реестра российского ПО: организация исходного кода, среда разработки, зависимости и контроль лицензий, контроль исходящих обращений, автоматические проверки, миграции, сборка и выпуск, приёмка на российских операционных системах, доставка и установка, сопровождение и сроки, устранение дефектов, вывод версии из поддержки, персонал, контакты службы поддержки, хранение исходного кода — с опорой на фактические сценарии сборки, проверки и обслуживания репозитория.

Правообладатель — Общество с ограниченной ответственностью «ИНТЕБРИКС» (ИНН 9707059284, КПП 770701001, ОГРН 1267700207895; 127051, г. Москва, вн.тер.г. муниципальный округ Тверской, ул. Петровка, д. 20/1, помещ. 17; info@intebrix.ru).

Настоящий документ описывает процессы разработки, сборки, проверки, выпуска, поставки и сопровождения программного обеспечения «TurbineCare AI». Описание составлено по фактическому состоянию исходных текстов и сборочных сценариев продукта; каждый раздел ссылается на файлы, которыми соответствующий процесс закреплён.

1. Общие сведения

Характеристика	Значение
Наименование	Программный комплекс «TurbineCare AI»
Правообладатель	ООО «ИНТЕБРИКС»
Выпущенная версия	1.0.0 от 12.09.2026, метка <code>v1.0.0</code> в хранилище; состав выпуска приведён в <code>CHANGELOG.md</code>
Класс ПО	основной — 08.23 «Программное обеспечение промышленной диагностики оборудования или систем оборудования»; дополнительный — 11.05 «Средства поддержки принятия решений (DSS)»
Правило нумерации	Семантическое версионирование, формат X.Y.Z
Канонический источник версии	Файл <code>VERSION</code> в корне хранилища
Система управления	Git, одно хранилище на весь продукт

Характеристика	Значение
версиями	
Порядок ведения истории изменений	CHANGELOG.md по соглашению Keep a Changelog

Продукт состоит из веб-приложения (Next.js, TypeScript), расчётной службы (Python, FastAPI), отдельного процесса сбора данных и сценариев обслуживания. Все части хранятся и выпускаются вместе, под одним номером версии.

1а. Стадии жизненного цикла и совершенствование ПО

Жизненный цикл ПО организован по стадиям; каждая стадия закреплена разделами настоящего документа и сценариями репозитория.

Стадия	Что выполняется	Где описано
Сбор и анализ требований	источники: техническое задание заказчика, перечень сигналов станции, обращения пользователей степени «Обычный» (предложения по развитию), требования нормативных документов; учёт — перечень задач BACKLOG.md	разделы 11, 12
Проектирование	архитектурные решения фиксируются в документе «Описание технической архитектуры» и планах в docs/; изменения схемы данных — миграциями	разделы 7, 8
Разработка	правки в ветвях с проверками при каждой фиксации; единые правила оформления; закрепление состава зависимостей и контроль лицензий; контроль исходящих обращений	разделы 2–5
Тестирование	модульные проверки, статический анализ, проверка типов, сквозная проверка и сценарии в браузере; предполётная проверка перед выпуском	раздел 6
Приёмка и выпуск	приёмочные испытания на операционных системах из реестра по методике из 16 шагов, протоколы, метка версии, дистрибутив с контрольными суммами	разделы 8, 9
Поставка и установка	архив дистрибутива, мастер установки, работа в закрытом контуре, обновление и откат	раздел 10
Документирование	руководство по эксплуатации, инструкция по установке, описание характеристик и архитектуры ведутся вместе с исходным текстом и проверяются на соответствие продукту автоматически (scripts/check-docs-terms.mjs, scripts/check-retention-consistency.mjs, scripts/check-version.mjs)	раздел 6, 8.2

Стадия	Что выполняется	Где описано
	перед каждым выпуском; журнал изменений — <code>CHANGELOG.md</code>	
Обучение персонала заказчика	инструктаж сменного персонала и главного инженера при опытной эксплуатации (этап 2 внедрения), администратора — при установке; материалы — руководство по эксплуатации	раздел 10, документ «Тарифная политика», разд. 2
Эксплуатация и сопровождение	приём обращений, сроки реакции и устранения, обновления, вывод версии из поддержки	разделы 11–13
Устранение неисправностей	воспроизведение, исправление, закрепление проверкой, внеочередной или плановый выпуск	раздел 12
Совершение (доработка)	плановые выпуски по семантическому версионированию; уведомление о несовместимых изменениях не позднее чем за 30 дней; доработки по требованиям заказчика — отдельным соглашением	разделы 8, 11, 13
Вывод из эксплуатации	предупреждение о снятии версии с поддержки, сохранение работоспособности бессрочной лицензии, выгрузка данных заказчика средствами резервного копирования	раздел 13, 10.4

2. Организация исходного кода и ветвление

2.1. Состав хранилища

Каталог	Содержание
<code>src/</code>	Веб-приложение: страницы, обработчики запросов (<code>src/app/api/</code> , 79 обработчиков), расчётные и служебные модули (<code>src/lib/</code>), процесс сбора данных (<code>src/collector/</code>) — 429 файлов на TypeScript
<code>backend/</code>	Расчётная служба на Python: пороговые зоны, прогноз, термодинамические расчёты, обнаружение аномалий
<code>prisma/</code>	Схема базы данных (<code>schema.prisma</code>), миграции, точка входа мастера установки
<code>scripts/</code>	Сборка выпуска, машинные проверки, установка и обслуживание у заказчика
<code>docker/</code> , <code>nginx/</code> , <code>config/</code>	Файлы сборки образов, настройки обратного прокси, образцы настройки заказчика
<code>deploy/</code> <code>demo/</code>	Демонстрационный стенд: <code>docker-compose.demo.yml</code> , сценарий подъёма <code>up.sh</code> , образец окружения

2.2. Ветвление

Рабочая ветвь выпуска 1.0.0 — `audit-fixes` (в ней велась подготовка к реестру, на ней стоит метка `v1.0.0`); ветвь `main` хранит состояние до начала этих работ и обновляется слиянием рабочей ветви при следующем выпуске. Крупные изменения выполняются в отдельных ветвях и переносятся в рабочую по завершении. Выпуск делается только из состояния, прошедшего предполётную проверку (раздел 6); метка версии вида `vX.Y.Z` ставится после приёмочных испытаний (раздел 8.3, шаг 12).

Письменный регламент именования рабочих ветвей и порядка их слияния в исходных текстах не закреплён: при одной рабочей ветви на выпуск порядок задаётся правилом «выпуск только из состояния, прошедшего предполётную проверку», и отдельного регламента не требует.

2.3. Единые правила оформления файлов

Правила заданы машинно и не зависят от настроек рабочего места разработчика:

- `.editorconfig` — кодировка UTF-8, перевод строки LF, отступ в два пробела, завершающая пустая строка;
- `.gitattributes` — нормализация переводов строк при записи в хранилище;
- `.gitignore` — перечень того, что в хранилище не попадает: файлы окружения (`.env`, `.env.local`), сертификаты TLS, каталог зависимостей, результаты сборки, файлы баз данных, вспомогательные выгрузки.

Запрет на хранение учётных данных в хранилище закреплён и в `CHANGELOG.md` как исправленный дефект.

3. Среда разработки и требования к ней

Составляюща

я	Требование	Где закреплено
Node.js	версия 20 и выше	DEV_GUIDE.md; сборочный образ <code>node:20-alpine</code> (Dockerfile)
Диспетчер пакетов	<code>npm</code>	<code>package.json</code> , <code>package-lock.json</code>
PostgreSQL	версия 16	DEV_GUIDE.md, <code>docker-compose.dev.yml</code>
Python	версия 3.11	<code>backend/Dockerfile</code> (<code>python:3.11-slim</code>), <code>backend/ruff.toml</code> (<code>target-version = "py311"</code>)
Контейнеризация	Docker и Docker Compose	Dockerfile, <code>docker-compose*.yml</code> ; требования к версиям — в инструкции по установке комплекта

Составляющая	Требование	Где закреплено
Основные средства разработки	TypeScript 5 (не ниже 5.4), Next.js 14.2, Prisma 7, Vitest 4	package.json
Средства проверки расчётной службы	ruff, pip-tools, fonttools	backend/requirements-dev.txt

Средства разработки в поставку не входят: образ расчётной службы устанавливает только зафиксированный перечень рабочих зависимостей, а перечень лицензий собирается по каталогу установки, а не по окружению разработчика.

Порядок первоначальной настройки рабочего места описан в DEV_GUIDE.md: установка зависимостей, запуск базы данных, применение миграций, загрузка демонстрационной телеметрии, запуск приложения. Для запуска в контейнерах предусмотрены отдельные файлы: docker-compose.dev.yml (разработка), docker-compose.yml (локальный запуск и демонстрация), docker-compose.prod.yml (промышленная эксплуатация), docker-compose.db.yml (необязательная встроенная база данных).

4. Управление зависимостями и контроль лицензий

4.1. Закрепление состава

Часть продукта	Файл закреплён	Как устанавливается
Веб-приложение	package-lock.json	npm ci в сборочном слое Dockerfile
Расчётная служба	backend/requirements.lock	pip install --require-hashes в backend/Dockerfile
Образ обслуживания	docker/ops/package-lock.json	npm ci в docker/ops.Dockerfile; замок порождается из дерева приложения (scripts/ops/obnovit-zamok-obsluzhivaniya.mjs), совпадение версий проверяет гейт

Установка при сборке образов идёт из архива пакетов (vendor/), который собирается один раз сценарием scripts/ops/sobrat-vendor.sh и содержит все пакеты npm обоих замков, колёса PyPI и движки Prisma; системные пакеты базового образа прокси (apk) и сами базовые образы контейнеров получаются из общедоступных реестров при подготовке сборочной машины и в архив пакетов не входят. Ключи --offline и --no-index запрещают обращение к сети, а не «предпочитают» архив: недостающий пакет

обрывает сборку, а не доезжает из Интернета молча. Полноту архива относительно замков проверяет условие гейта «Сборка не зависит от чужих реестров».

В `backend/requirements.txt` заданы только нижние границы версий; фактические версии закрепляются в `requirements.lock`, собираемом командой `pip-compile --generate-hashes`. Замок содержит 34 пакета и 828 хешей (`docs/evidence/6.3-6.4_гейт_выпуска_и_замок_зависимостей.md`). Установка требует хеш для каждого пакета, включая приходящие транзитивно: подменённый пакет не установится. Замок пересобирается в среде сборочного образа (`python:3.11-slim`), а не на рабочем месте разработчика, — иначе разрешение версий зависело бы от версии Python у разработчика.

4.2. Контроль лицензий

Контроль выполняется двумя сценариями с общей частью `scripts/licenses.mjs`:

`scripts/check-licenses.mjs` — проверка допустимости лицензий. Сборка останавливается в трёх случаях: лицензия не входит в разрешённый перечень (19 наименований: MIT, ISC, Apache-2.0, BSD-2-Clause, BSD-3-Clause, 0BSD, BlueOak-1.0.0, Unlicense, CC0-1.0, MPL-2.0, OFL-1.1 и другие разрешительные); лицензия разделительная, а выбор ветви письменно не зафиксирован; лицензию определить не удалось. Выбор ветви для компонентов с двойным лицензированием зафиксирован в перечне `ELECTED` (6 компонентов: `jszip`, `expand-template`, `dompurify`, `rc`, `rgbcolor`, `python-dateutil`), источник условий для компонентов без объявленной лицензии — в перечне `RESOLVED`.

`scripts/gen-licenses.mjs` — сборка файла `THIRD-PARTY-LICENSES.md` из фактического дерева зависимостей. Файл содержит наименование и версию каждого компонента, его лицензию, полный текст лицензии и уведомления об авторстве. По состоянию выпуска 1.0.0 в нём 563 компонента: 527 из npm, 34 из PyPI и 2 вложенных файлами. Режим `--check` сравнивает файл с деревом и не допускает выпуск с устаревшим перечнем. Вручную файл не редактируется.

`THIRD-PARTY-LICENSES.md` копируется в образ приложения (`Dockerfile`) и входит в состав поставки: лицензии MIT, BSD и ISC требуют сохранять уведомление об авторстве в распространяемом продукте.

5. Контроль исходящих обращений

Сценарий `scripts/check-outbound-hosts.mjs` ведёт учёт обращений продукта к внешним адресам. Каждое заявленное обращение обязано называть назначение и признак развёртывания, который его отключает:

Адрес	Назначение	Чем закрывается
<code>api.telegram.org</code>	уведомления о тревогах; только на экземпляре центра удалённого мониторинга	<code>isOperatorInstance()</code> в <code>src/lib/telegram.ts</code> — на экземпляре объекта не вызывается ни при каком признаке развёртывания; в центре дополнительно признак <code>telegram/api/telegram</code> и <code>/api/send-alert</code> на объекте отвечают 404 из

Адрес	Назначение	Чем закрывается
		<code>src/middleware.ts</code>

Перечень исчерпывающий: наружу продукт обращается по одному адресу.

Обращения к внешним языковым моделям (`llm.api.cloud.yandex.net`, `gigachat.devices.sberbank.ru`, `ngw.devices.sberbank.ru`) из продукта удалены вместе с самим помощником: языковой модели в комплексе нет ни своей, ни внешней.

Раздел «Выводы» построен на детерминированных экспертных правилах (`src/lib/dashboard/insights.ts`) — одинаковый вход даёт одинаковый текст, объяснимый по правилу, и наружу раздел не ходит. Отсутствие обращений закреплёно проверкой: появление в коде незаявленного адреса останавливает выпуск.

Отдельно перечислены каналы, ведущие не наружу, а **внутри периметра заказчика**: почта (`SMTP_HOST`, `SMTP_PORT`) — уведомления дежурному персоналу; адрес задаётся при установке, в коде его нет, и сканер его не найдёт, поэтому этот канал назван в самом сценарии отдельной таблицей. Прочие настраиваемые назначения — сервер OPC UA, внешняя база данных, корпоративный каталог — задаются администратором в интерфейсе и находятся внутри контура заказчика.

Что именно проверяется. Проверка выполняется по исходному тексту в каталогах `src` и `backend`, а не по собранному образу. Причина указана в самом сценарии и подтверждена попыткой: в собранном пакете находятся сотни адресов из комментариев, полей `homepage` и `bugs` в описаниях пакетов и текстов лицензий сторонних компонентов; в сжатом и переименованном коде отличить строку-упоминание от фактического вызова нечем, и проверка теряет смысл. Проверять следует там, где виден вызов, — в исходных текстах, за которые отвечает правообладатель.

Границы проверки. Сценарий находит адреса, записанные в коде явной строкой вида `http://...` или `https://...`. Обращения, собираемые из частей во время работы, и обращения внутри сторонних компонентов им не покрываются. Со стороны браузера дополнительным ограничением служит политика содержимого (`next.config.js`): источники ограничены собственным адресом приложения, внешних адресов в политике нет.

6. Автоматические проверки и испытания

Все проверки собраны в один сценарий — `scripts/release-check.sh`. Он выполняет их **все** и не останавливается на первом отказе, показывая полную картину. Код возврата 0 означает, что выпускать можно; иной — что нельзя. Сценарий ничего не изменяет и запускается в любой момент; ключ `- - fast` пропускает сборку и сквозную проверку.

Условий тридцать два, они разбиты на пять групп. Каждое условие даёт один из трёх исходов: пройдено, отказ либо пропущено — когда проверить нечем (нет сети, нет браузера). Пропуск виден отдельной строкой и не засчитывается как прохождение.

Единство и состав поставки

Проверка	Средство	Что подтверждает
Версия единая во всех местах	<code>scripts/check-version.mjs</code>	Версия совпадает во всех местах, где объявлена
Лицензии компонентов разрешены	<code>scripts/check-licenses.mjs</code>	Все лицензии в разрешённом перечне
Перечень лицензий не устарел	<code>scripts/gen-licenses.mjs --check</code>	<code>THIRD-PARTY-LICENSES.md</code> описывает фактическую поставку
Образы поставки все именованы	<code>scripts/check-bundle-images.mjs</code>	Каждый образ файла развёртывания назван и попадёт в дистрибутив
Известных уязвимостей нет	<code>scripts/check-vulnerabilities.mjs</code>	В поставляемых компонентах нет критических уязвимостей; принятые риски — со сроком и перечнем известных уязвимостей
Образы соберутся у заказчика	<code>scripts/check-dockerfile-portability.mjs</code>	Файлы сборки образов не используют возможностей, которых нет у классического сборщика Docker на целевых ОС
Хранилище кода в РФ не отстало	<code>scripts/check-zerkalo.mjs</code>	Вершины ветвей рабочей копии и хранилища GitFlic совпадают
Сборка не зависит от чужих реестров	<code>scripts/check-vendor.mjs</code>	Архив пакетов покрывает оба замка зависимостей целиком, движки Prisma на месте
Замок образа обслуживания на месте	<code>scripts/check-zamok-obsluzhivaniya.mjs</code>	Версии Prisma в образе обслуживания совпадают с деревом приложения
Сценарии запускаются	<code>scripts/check-scripts-executable.mjs</code>	У сценариев поставки есть право на исполнение
Протоколы испытаний не врут	<code>scripts/check-protokoly.mjs</code>	Протоколы относятся к дистрибутиву в <code>dist/</code> (по контрольной сумме) и приложения к ним на месте

Документы описывают этот продукт

Проверка	Средство	Что подтверждает
Запрещённых строк в документах нет	<code>scripts/check-docs-terms.mjs</code>	В пакете подачи и сопроводительных документах нет строк, описывающих удалённое: языковой помощник и внешние языковые модели, тарифные ступени, учётная запись по умолчанию, прежние команды запуска, переключатель языка и светлая тема, следы незаконченной работы
Сроки хранения совпадают везде	<code>scripts/check-retention-consistency.mjs</code>	Глубина архива, обещанная в документах и на сайте, не короче той, по которой удаляет код

Безопасность

Проверка	Средство	Что подтверждает
Все роуты под проверкой доступа	<code>scripts/check-route-guards.mjs</code>	Проверка вызывающего, принадлежность машины организации, требуемая роль на изменяющих запросах
Наружу смотрит только прокси	<code>scripts/check-exposed-ports.mjs</code>	Публикуется единственный порт обратного прокси
Исходящие обращения заявлены	<code>scripts/check-outbound-hosts.mjs</code>	Незаявленных обращений наружу нет
В артефакте сборки нет секретов	<code>scripts/check-artifact-secrets.mjs</code>	В сборке нет файлов окружения и значений секретов
Изменяющие действия записываются	<code>scripts/check-audit-coverage.mjs</code>	Каждый изменяющий обработчик пишет в журнал действий
Режим только чтения не растерян	<code>scripts/check-license-readonly.mjs</code>	Признак <code>readOnly</code> проверяется всюду, где изменяются данные

Корректность

Проверка	Средство	Что подтверждает
Чтение телеметрии ограничено окном	<code>scripts/check-unbounded-reads.mjs</code>	Чтение телеметрии ограничено окном, периодом или числом строк

Проверка	Средство	Что подтверждает
Аномалии считает обслуживаение	<code>scripts/check-anomaly-wired.mjs</code>	Цепочка от процесса сбора до записи оценки аномальности цела: обнаружение аномалий выполняется обслуживанием по расписанию, а не запускается вручную
Статический анализ Python	<code>ruff</code> , настройка <code>backend/ruff.toml</code>	Отсутствие дефектов, выявляемых статически
Типы	<code>tsc --noEmit</code>	Типы согласованы
Тесты приложения	<code>Vitest</code> , <code>npx vitest run</code>	Полный набор модульных проверок приложения проходит
Тесты расчётной службы	<code>backend/tests/test_pgdb.py</code>	Работа с базой
Тесты обнаружения аномалий	<code>backend/tests/test_anomaly.py</code>	Обнаружение аномалий

Сборка

Проверка	Средство	Что подтверждает
Приложение собирается	<code>npm run build</code>	Продукт собирается
Middlewar е попал в сборку	<code>scripts/check-middlewar e-built.mjs</code>	Заслоны ролей и доступа есть в собранном образе. Проверка идёт строго после сборки: раньше она разбирала манифест от прошлого раза
Состав описывает отгружаемое	<code>scripts/check-sostav-sborki.mjs</code>	Каждый пакет собранного приложения назван в перечне лицензий, и в той версии, что уезжает
Коллектор собирается	<code>npm run build:collector</code>	Процесс сбора собирается
Собранное приложение	<code>scripts/smoke.mjs</code>	Сквозная проверка на чистой базе

Проверка	Средство	Что подтверждает
ие работает		
Сквозные сценарии в браузере	Playwright, npm run e2e	Двадцать девять сценариев в десяти файлах работы пользователя на собранном приложении: вход со сменой временного пароля, выбор машины, разделы и группы, период, снятие и возврат тревоги, карта уставок, журнал ТО, отчёт, выход, заслоны ролей. Шаг не выполняется при ключе - - fast; если пакет Playwright не установлен, шаг помечается пропущенным и выпуск не останавливает — в закрытом контуре его поставить неоткуда

Число проверок Vitest в отчёте не фиксируется числом: оно меняется с каждым исправлением, закреплённым проверкой (раздел 12), и берётся из вывода `npx vitest run` на момент выпуска.

Сквозная проверка поднимает чистую базу данных, применяет миграции, заводит администратора, запускает **собранное** приложение и обращается к нему запросами, после чего проверяет состояние базы и убирает за собой. Её ограничения записаны в самом сценарии и не скрываются: разделы с данными машины собираются в браузере, поэтому по ним проверка не проходит; работы по снятию этого ограничения учтены в перечне задач.

Проверки заведены не «на всякий случай», а по конкретным случаям: причина появления каждой изложена в заголовке её сценария. Ни одна из них не требует доступа в сеть и не обращается к внешним службам.

Отдельного сервера непрерывной сборки в хранилище не настроено: перечисленные проверки выполняются сценарием `release-check.sh` — вручную перед выпуском и автоматически внутри сценария выпуска.

7. Миграции базы данных

Схема базы данных описана в `prisma/schema.prisma`; изменения оформляются миграциями в каталоге `prisma/migrations`. На момент составления документа миграций 21.

Соглашение об именовании. Имя каталога миграции состоит из метки даты и времени из четырнадцати цифр (ГГГГММДДЧЧММСС) и краткого имени латиницей через подчёркивание, например `20260824140000_signal_registry`, `20260904090000_dvustoronnie_porogi`. Порядок применения определяется возрастом имени. Тип базы данных закреплён файлом `migration_lock.toml` (`provider = "postgresql"`) и не может быть изменён незаметно.

Порядок применения. На рабочей базе применяются только зафиксированные миграции, командой `prisma migrate deploy`. Применение выполняется отдельным одноразовым образом обслуживания (`docker/ops.Dockerfile`, служба `ops` в `docker-compose.prod.yml`), а не образом приложения. Приведение схемы командой `db push`

- `-accept-data-loss` на рабочей базе запрещено: она удаляет столбцы вместе с данными.

Демонстрационные данные на рабочую установку не попадают. Наполнение демонстрационным набором выполняется только явным профилем `seed` и только на демонстрационном стенде (`deploy/demo/`), где заведены отдельные учётные записи и действует ночной сброс. Запасной демонстрационный набор в рабочем режиме отключён кодом: он подставляется лишь при `ALLOW_DEMO_DATASET=1`, чего в эксплуатации быть не должно.

8. Сборка и выпуск

8.1. Нумерация версий

Применяется семантическое версионирование в формате `X.Y.Z`. Канонический источник — файл `VERSION`; сценарий выпуска отклоняет значение иного формата. Версия названа ещё в нескольких местах, и их единство проверяется машинно (`scripts/check-version.mjs`):

Место	Назначение
<code>package.json</code>	Версия пакета, из неё берётся версия в интерфейсе
<code>src/lib/openapi.ts</code>	Описание программного интерфейса, читаемое заказчиком как документ
<code>backend/api/main.py</code>	Заголовок расчётной службы, виден в её ответах
<code>docker-compose.prod.yml</code>	Запасные значения тегов образов при развёртывании

Дополнительно `next.config.js` сверяет `VERSION` с `package.json` и прекращает сборку при расхождении.

8.2. История изменений

`CHANGELOG.md` ведётся по соглашению Keep a Changelog с разделами «Добавлено», «Изменено», «Удалено» и «Исправлено». Выпуск 1.0.0 датирован 12.09.2026 — датой установки тега версии после приёмочных испытаний 11.09.2026 (протоколы № 1 и № 2); датировать выпуск раньше приёмки нельзя. Дата проставляется в журнале в хранилище при установке тега. Дистрибутив собирается до приёмки и тега, поэтому в архиве 1.0.0 раздел выпуска ещё не датирован — состав выпуска в нём тот же. Файл копируется в дистрибутив и в образ приложения.

8.3. Порядок выпуска

Сборка выполняется сценарием `scripts/release.sh [версия] [--tag]` в следующем порядке:

1. Проверка формата номера версии; отказ при несоответствии `X.Y.Z`.
2. Запись номера в `VERSION` и `package.json`.

3. Предполётная проверка (`release-check.sh`, раздел 6); при отказе выпуск прекращается. При выпуске наличие архива пакетов (`vendor/`) обязательно: поставка не имеет права уехать, если её нельзя пересобрать без доступа к сети Интернет.
4. Сборка четырёх образов — `turbinecare-web`, `turbinecare-python-calc`, `turbinecare-ops`, `turbinecare-nginx` — под платформу поставки (по умолчанию `linux/amd64`, независимо от машины сборки) из архива пакетов, без обращения к реестрам пакетов; базовые образы контейнеров и образ СУБД для стенда берутся из локального хранилища сборочной машины (при отсутствии — из общедоступного реестра образов).
5. Сверка архитектуры каждого образа с объявленной платформой и выгрузка образов в файлы `images/*.tar` — только под платформу поставки.
6. Копирование файлов развёртывания `docker-compose.prod.yml` и `docker-compose.db.yml`, настроек обратного прокси, образца настройки заказчика, образца файла окружения, сценариев установки, обновления, отката, резервного копирования и обслуживания, руководства по установке `DEPLOY.md`, файла `PLATFORM`.
7. Копирование архива пакетов (`vendor/npm-<версия>.tar.gz`, `vendor/pypi-<версия>.tar.gz`), с которым поставку можно пересобрать из исходных текстов без сети.
8. Копирование `CHANGELOG.md` и запись файла `VERSION` в дистрибутив.
9. Повторная проверка **собранного дистрибутива** на наличие секретов.
10. Расчёт контрольных сумм SHA-256 по всем файлам (`checksums.sha256`).
11. Упаковка в архив `turbinecare-<версия>.tar.gz`.
12. Метка версии `vX.Y.Z` устанавливается отдельно, после приёмочных испытаний собранного дистрибутива (раздел 9): либо ключом `- - tag` этого сценария, либо командой `git tag` на той же фиксации; по умолчанию сборка метку не создаёт.

При ошибке на любом шаге частично собранный дистрибутив удаляется; остаток незавершённой сборки отличается от готового дистрибутива по отсутствию файла контрольных сумм и убирается следующим запуском `сам`, готовый дистрибутив молча не затирается.

9. Приёмка выпуска

Приёмка состоит из двух частей: машинной, встроенной в сборку, и приёмочных испытаний собранного дистрибутива на целевых операционных системах.

9.1. Машинная часть

Признаком готовности выпуска к испытаниям служит совокупность:

- код возврата 0 у `scripts/release-check.sh` — пройдены все условия раздела 6, пропущенные названы поимённо;
- отсутствие секретов в собранном дистрибутиве (повторная проверка на шаге 9 раздела 8.3);

- наличие в дистрибутиве файла контрольных сумм, истории изменений, файла версии и архива пакетов (перечень лицензий сторонних компонентов входит в образ приложения).

9.2. Приёмочные испытания на операционных системах

Собранный дистрибутив испытывается на операционных системах из Единого реестра российского ПО по методике `docs/testing/Методика_испытаний_ОС.md`. Испытания выполняет сценарий `scripts/os-acceptance.sh` на чистой машине с целевой ОС, подготовленной сценарием `scripts/ops/prepare-os-host.sh` (Docker, Compose, СУБД из репозитория ОС). Шестнадцать шагов:

1. Сведения об операционной системе и среде.
2. Распаковка дистрибутива.
3. Сверка контрольных сумм.
4. Загрузка образов из дистрибутива (одноимённые образы, оставшиеся на машине, предварительно снимаются — испытывается именно архив).
5. Мастер первичной настройки с внешней СУБД без вопросов.
6. Готовность приложения по пробе живости.
7. Наружу смотрит только обратный прокси.
8. Дым: живость и заслон входа.
9. Заведение учётной записи с временным паролем.
10. Вход временным паролем, отказ разделов до смены, смена, вход постоянным паролем, отзыв временного.
11. Приём точек по HTTP, подтверждение в базе данных, ряд для графика по расчётному показателю.
12. Резервная копия штатным сценарием и восстановление в отдельную базу с совпадением содержимого.
13. Версия СУБД: `server_version_num` не ниже 140000.
14. Создание, наполнение, отсоединение и снос секции.
15. Прогон в замкнутом контуре: состав десять минут работает в сети без маршрута наружу; отсутствие выхода из сети подтверждается из той же сети.
16. Уборка: состав остановлен, база заказчика не тронута.

Сценарий записывает протокол по форме `docs/testing/Протокол_шаблон.md` с выдержкой из каждого шага и складывает полные выводы шагов рядом с протоколом; пароли из выводов вычищаются до записи. Дистрибутив считается совместимым с операционной системой, если все шестнадцать шагов пройдены.

Протоколы хранятся в `docs/testing/протоколы/` и записывают контрольную сумму испытанного дистрибутива. Условие гейта «Протоколы испытаний не врут» (`scripts/check-protokoly.mjs`) сверяет эту сумму с архивом в `dist/`: изменение продукта после испытаний делает протокол недействительным и останавливает следующий выпуск до перепрогона. Для выпуска 1.0.0 оформлены протоколы № 1 (Astra Linux Special Edition 1.8.2) и № 2 (ПЕД ОС 8.0.2) от 11.09.2026.

9.3. Итог приёмки

Тег версии `vX.Y.Z` устанавливается только после подписания протоколов испытаний. Число пройденных проверок Vitest на момент выпуска берётся из вывода `npm vitest run`.

10. Доставка и установка у заказчика

10.1. Состав дистрибутива

Дистрибутив — архив `turbinecare-<версия>.tar.gz`: выгруженные образы контейнеров (`images/`, пять файлов), файлы развёртывания `docker-compose.prod.yml` и `docker-compose.db.yml`, настройки обратного прокси (`nginx/`), образец настройки заказчика (`config/client-config.example.json`), образец файла окружения (`.env.example`), сценарии установки и обслуживания (`scripts/`), архив пакетов для повторной сборки без сети (`vendor/`), руководство по установке `DEPLOY.md`, файлы `VERSION` и `PLATFORM`, `CHANGELOG.md`, `checksums.sha256`.

10.2. Установка

Порядок установки: перенос архива на сервер, распаковка, загрузка образов командой `docker load`, заполнение файла окружения по образцу, запуск `docker compose up -d`. Миграции применяются одноразовым образом обслуживания; им же применяется описание станции и заводится первый администратор (`scripts/ops/create-admin.cjs`) — с разовым случайным паролем, который печатается один раз и подлежит обязательной смене при первом входе. До смены пароля обращения к программному интерфейсу отклоняются. При установке правообладатель проводит инструктаж администратора площадки по настоящему разделу и руководству по эксплуатации; сменный персонал и главный инженер обучаются на этапе опытной эксплуатации (документ «Тарифная политика», разд. 2).

Учётных записей с заранее известными паролями в поставке нет. Демонстрационные данные на рабочую базу не заводятся.

Отдельного шаблона развёртывания «для клиента» в поставке нет: она описывается файлом `docker-compose.prod.yml`, к которому при отсутствии у заказчика своей СУБД добавляется `docker-compose.db.yml`.

В дистрибутив входит мастер установки `scripts/setup-wizard.sh` (Linux) и сценарии обслуживания `update.sh`, `rollback.sh`, `backup.sh`; сценарии для рабочих станций разработчика в поставку не входят. Отдельного сценария установки Docker для Linux в поставке нет намеренно: он ходил за пакетами в внешнее хранилище, что несовместимо с закрытым контуром, — Docker на объекте устанавливает администратор из репозитория своей операционной системы.

10.3. Работа в закрытом контуре

Дистрибутив всегда содержит все образы, на которые ссылаются файлы развёртывания, — включая базу данных для стенда, обратный прокси и образ обслуживания, — поэтому установка не требует доступа к внешним хранилищам образов.

Продукт работоспособен в закрытом контуре целиком, без единственного исходящего обращения: языковых моделей в нём нет, а мессенджер — канал экземпляра центра и на объекте закрыт (раздел 5). Раздел «Выводы», индекс состояния, деградация, прогноз и обнаружение аномалий считаются внутри периметра.

Внешняя СУБД заказчика — штатный режим поставки: система управления базами данных вынесена из поставки и подключается одной строкой соединения, что позволяет использовать СУБД, выбранную и лицензированную заказчиком, и его же средства резервного копирования. Для стенда и малой установки, где своей СУБД нет, база поднимается файлом `docker-compose.db.yml`.

10.4. Обновление, откат и резервное копирование

Операция	Средство	Содержание
Обновление	<code>scripts/update.sh</code>	Резервная копия перед обновлением, применение миграций, проверка работоспособности после запуска, автоматический откат при неуспехе
Откат	<code>scripts/rollback.sh</code>	Возврат к предыдущей версии с восстановлением состояния
Резервное копирование	<code>scripts/backup.sh</code>	Копия базы данных; полная копия с файлом окружения, настройками и сертификатами; восстановление из копии; выгрузка на внешний носитель; перечень копий; срок хранения по умолчанию 30 дней, задаётся в <code>client-config.json</code>

11. Сопровождение и сроки

Регламент сопровождения закреплён в исходных текстах сайта продукта (`src/lib/site.ts`, константа `SUPPORT`) и публикуется на нём.

Показатель	Значение
Время работы службы поддержки	Рабочие дни, с 10:00 до 19:00 по московскому времени
Доступность	Не менее 99,5 % в календарный месяц — для демонстрационного экземпляра правообладателя; доступность экземпляра, установленного в контуре заказчика, определяется его инфраструктурой и регламентом эксплуатации
Поддержка предыдущей основной версии	Не менее 12 месяцев

Показатель	Значение
Уведомление об основных и несовместимых изменениях	Не позднее чем за 30 календарных дней

Сроки реакции и устранения по степеням значимости обращения:

Степень	Время реакции	Срок устранения
Критический	Не более 4 рабочих часов	Восстановление — не более 1 рабочего дня
Высокий	Не более 8 рабочих часов	Не более 5 рабочих дней либо в ближайшем плановом выпуске
Обычный	Не более 2 рабочих дней	В плановом порядке

Каналы обращения — электронная почта info@intebrix.ru, форма обратной связи на странице <https://turbinecare.ru/contacts> и телефон +7 916 925-54-71 в рабочие дни с 10:00 до 19:00 по московскому времени. Поддержка оказывается на русском языке. Порядок обработки обращений и полные контакты — разделы 12 и 16.

Модель приобретения различает две стороны:

- **станция** покупает комплекс единовременно и ставит его **перманентно**. Лицензии на экземпляре объекта нет: ни ключа, ни срока, ни подписки, и ни одна проверка лицензии там не выполняется (`src/lib/licenseGuard.ts`). Право пользоваться продуктом не истекает никогда; заканчивается только поддержка — то есть прекращается поступление обновлений, а работа установленного продукта не ограничивается ничем. На объекте критической информационной инфраструктуры иное недопустимо: мониторинг не должен зависеть от состояния расчётов с поставщиком;
- **центр удалённого мониторинга** (`INSTANCE_ROLE=operator`) продлевает подписку. По истечении срока его экземпляр переходит в режим только чтения: данные, история и выгрузки остаются доступны, изменяющие обращения отклоняются (`src/lib/licenseValidity.ts`, признак `readOnly`). Полная блокировка не применяется и здесь.

О приближении окончания срока подписки пользователь предупреждается за 30 дней.

12. Порядок исправления дефектов

1. **Приём обращения** — по каналу, указанному в разделе 11, с присвоением степени значимости по той же таблице.
2. **Воспроизведение** — дефект воспроизводится на среде разработки или на развёртывании в контейнерах, повторяющем состав поставки.
3. **Исправление** в исходном тексте.

4. **Закрепление проверкой.** Исправленный дефект закрепляется так, чтобы он не мог вернуться незамеченным: модульной проверкой в наборе Vitest либо, если дефект относится к классу, а не к отдельному месту, отдельной машинной проверкой в составе предполётного набора. Все проверки раздела 6 заведены именно так — по конкретным случаям; причина появления каждой изложена в заголовке соответствующего сценария.
5. **Предполётная проверка** (`release-check.sh`) целиком.
6. **Выпуск** по разделу 8 и запись в `CHANGELOG.md` в раздел «Исправлено».

Учёт выявленных, но ещё не устранённых недостатков ведётся в перечне задач (`BACKLOG.md`) с идентификатором, содержанием, оценкой и состоянием каждой позиции; результаты проведённых проверок и разборов хранятся в `docs/evidence/`. Внешняя система регистрации обращений заказчиков по исходным текстам не определяется: обращения принимаются по каналу раздела 11 и учитываются правообладателем.

13. Вывод версии из поддержки

Правило	Значение	Основание
Срок поддержки предыдущей основной версии	Не менее 12 месяцев с выпуска следующей основной версии	<code>SUPPORT.majorSupport</code> в <code>src/lib/site.ts</code>
Предварительное уведомление об основных и несовместимых изменениях	Не позднее чем за 30 календарных дней	<code>SUPPORT.notice</code> там же
Работа экземпляра объекта после окончания сопровождения	Не ограничивается: проверок лицензии на экземпляре объекта нет	<code>src/lib/licenseGuard.ts</code>
Предупреждение об окончании срока подписки в интерфейсе центра	За 30 дней	<code>EXPIRY_WARNING_DAYS</code> в <code>src/lib/licenseValidity.ts</code>

Перечень поддерживаемых версий и уведомление о снятии конкретной версии с поддержки публикуются правообладателем на сайте продукта по тому же регламенту (`src/lib/site.ts`, константа `SUPPORT`): предварительное уведомление — не позднее чем за 30 календарных дней.

14. Хранение исходного кода

Исходный код хранится в системе управления версиями Git. Хранилище для целей реестра — сервис GitFlic, размещённый на территории Российской Федерации: <https://gitflic.ru/project/info-intebrix-ru/turbinecare-ai> (приложение) и `turbinecare-site` (сайт правообладателя с документацией) в той же учётной записи. Доступ закрытый, по SSH. Мобильное приложение правообладателя ведётся отдельным проектом и в состав ПО не входит; в состав ПО входит только программный интерфейс для мобильного клиента (`/api/mobile/*`, описан в спецификации OpenAPI).

Рабочая копия репозитория — на рабочей станции правообладателя. Условие гейта «Хранилище кода в РФ не отстало» (`scripts/check-zerkalo.mjs`) сверяет вершины ветвей рабочей копии и хранилища GitFlic перед каждым выпуском и останавливает его при расхождении.

Подробно технические средства хранения исходного текста и объектного кода описаны в документе «Описание технических средств хранения исходного текста и объектного кода».

Резервное копирование рабочих данных заказчика выполняется на стороне заказчика средствами, входящими в поставку (`scripts/backup.sh`, раздел 10.4), и территории размещения хранилища исходного кода не касается.

15. Персонал

15.1. Состав работ, требующих квалифицированного персонала

Роль	Область ответственности
Разработчик веб-приложения и программного интерфейса	TypeScript, Next.js, React, Node.js, Prisma, PostgreSQL; разделы интерфейса, программный интерфейс приёма и выгрузки, разграничение доступа, журнал действий
Разработчик процесса сбора данных	клиент OPC UA (<code>node-opcua</code>), политики защиты и сертификаты, буферизация и сопоставление тегов, обслуживание архива
Инженер-теплотехник (разработчик расчётной части)	термодинамика газотурбинного цикла, приведение параметров к стандартным условиям (ISO 2314), психрометрия, методики оценки деградации и обледенения; Python, NumPy, pandas, scikit-learn, CoolProp
Инженер по развёртыванию и эксплуатации	Docker, Docker Compose, nginx, PostgreSQL; сборка дистрибутива без обращения к сети, приёмочные испытания на операционных системах из реестра, резервное копирование и восстановление
Инженер по тестированию	Vitest, Playwright; сценарии приёмочных испытаний; проверка воспроизводимости расчётов
Специалист технической поддержки	приём и разбор обращений, работа с журналами служб, консультирование по эксплуатации и пусконаладке

Одна роль может исполняться одним работником, и один работник может совмещать несколько ролей.

15.2. Требования к квалификации

- высшее образование в области информационных технологий, теплоэнергетики, прикладной математики или смежных областях;
- для разработчиков — практический опыт промышленной разработки на соответствующем стеке от трёх лет;

- для инженера-теплотехника — подготовка в области тепловых двигателей и энергетических установок, знание методик испытаний газотурбинных установок;
- для инженера по развёртыванию — практический опыт эксплуатации контейнеризованных приложений, в том числе в закрытых контурах без доступа в сеть Интернет;
- знание русского языка на уровне, достаточном для ведения технической документации и взаимодействия с заказчиками.

15.3. Фактическая численность

Штатных сотрудников правообладатель не имеет; разработку, сопровождение, техническую поддержку, гарантийное обслуживание и модернизацию программы выполняют специалисты — граждане Российской Федерации, привлекаемые правообладателем по гражданско-правовым договорам (внештатные), в том числе автор программы (свидетельство о государственной регистрации программы для ЭВМ № 2026684187 от 09.08.2026). Численность на дату настоящей редакции:

Направление работ	Численность	Форма привлечения	Гражданство
Разработка и совершенствование ПО (роли 15.1: разработчики, инженер-теплотехник)	2	гражданско-правовые договоры	Российская Федерация
Сопровождение, техническая поддержка, гарантийное обслуживание, модернизация (все роли 15.1)	3	гражданско-правовые договоры	Российская Федерация

Один специалист может участвовать в обоих направлениях; общее число привлечённых специалистов — 3. Персональные данные специалистов в документе не приводятся и предоставляются по запросу.

15.4. Территория и адреса выполнения работ

Работы по разработке, тестированию, выпуску версий, технической поддержке и сопровождению выполняются на территории Российской Федерации. Фактические адреса: инфраструктура разработки — рабочая станция правообладателя по адресу 127051, г. Москва, ул. Петровка, д. 20/1, помещ. 17, и хранилище исходного текста на сервисе GitFlic (Российская Федерация); разработчики — г. Москва (удалённая работа на территории Российской Федерации); служба технической поддержки — 127051, г. Москва, ул. Петровка, д. 20/1, помещ. 17.

16. Контакты службы поддержки

Канал	Реквизит
Электронная почта (приём обращений, консультации, общие	info@intebrix.ru

Канал	Реквизит
вопросы)	
Форма обратной связи	https://turbinecare.ru/contacts
Телефон	+7 916 925-54-71 (приём обращений в рабочие дни с 10:00 до 19:00 по московскому времени)
Почтовый адрес	127051, г. Москва, вн.тер.г. муниципальный округ Тверской, ул. Петровка, д. 20/1, помещ. 17
Наименование организации	ООО «ИНТЕБРИКС», ИНН 9707059284, ОГРН 1267700207895

Язык обращения и ответа — русский. Режим работы службы поддержки: рабочие дни, с 10:00 до 19:00 по московскому времени. Обращение по критическому инциденту, поступившее вне режима работы, регистрируется; отсчёт срока реакции начинается с ближайшего рабочего часа. Расширенный режим приёма обращений предоставляется по отдельному соглашению о сопровождении.

17. Сопутствующие документы

Документ	Назначение
Описание функциональных характеристик программного обеспечения	состав функций, технические характеристики, сторонние компоненты, ограничения
Инструкция по установке программного обеспечения	требования к окружению, порядок установки, обновления, отката, доступ к проверочному экземпляру
Руководство по эксплуатации программного обеспечения	руководство пользователя и администратора, учётная запись эксперта, техническая поддержка
Настоящее описание процессов жизненного цикла	организация разработки и сопровождения
Протоколы приёмочных испытаний № 1 и № 2 от 11.09.2026	совместимость с Astra Linux Special Edition 1.8.2 и РЕД ОС 8.0.2

Внутренние источники, на которых основано настоящее описание, хранятся в репозитории продукта: `CHANGELOG.md` (журнал изменений; входит в дистрибутив), `DEPLOY.md` (руководство по развёртыванию; входит в дистрибутив), `THIRD-PARTY-LICENSES.md` (состав сторонних компонентов и их лицензии; входит в образ приложения), `docs/testing/Методика_испытаний_ОС.md` (методика приёмочных испытаний), сценарии `scripts/release-check.sh`, `scripts/release.sh`, `scripts/os-acceptance.sh`.