ООО «ИНСТИТУТ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ ГЛАВПРО»

Обучающий центр дополнительного профессионального образования

ТЕХНИЧЕСКАЯ ДОКУМЕНТАЦИЯ

Инструкция по установке экземпляра программного обеспечения

Платформа обучения информационной безопасности: требования к аппаратному и программному обеспечению, порядок установки на чистой машине, первичная инициализация, проверка работоспособности, установка без контейнеризации, устранение типовых неполадок, порядок обновления

Наименование программы	**Платформа обучения информационной безопасности**
Сведения о программе опубликованы	**https://xn--80aeb6arfh.xn--p1ai**
Класс программного обеспечения	**12.17 Программное обеспечение для решения отраслевых задач в области образования**
Тип поставки	**Веб-приложение, устанавливаемое в контуре эксплуатирующей организации**
Язык интерфейса	**Русский**
Дата	**12.10.2026**
Редакция	**1.4**

Оглавление

1 Общие сведения о продукте и назначение документа

2 Требования к аппаратному обеспечению

3 Требования к программному обеспечению

4 Состав дистрибутива и состав служб

5 Порядок установки

- 5.1 Получение дистрибутива
- 5.2 Настройка переменных окружения
- 5.3 Сборка образов без обращения к внутреннему конвейеру
- 5.4 Запуск контейнеров
- 5.5 Ключи приложения
- 5.6 Миграции базы данных
- 5.7 Сборка клиентской части

6 Первичная инициализация

- 6.1 Начальные данные
- 6.2 Учётная запись администратора
- 6.3 Исходящая почта
- 6.4 Хранилище файлов
- 6.5 Push-уведомления (опционально)

7 Проверка работоспособности

8 Установка без использования контейнеризации

9 Типовые неполадки при установке и их устранение

10 Обновление установленного экземпляра

1. Общие сведения о продукте и назначение документа

Установка экземпляра

Документ описывает установку экземпляра программного обеспечения на чистой машине — от получения дистрибутива до проверки работоспособности — независимо от внутреннего конвейера непрерывной интеграции правообладателя и его внутренней инфраструктуры сборки. Форма поставки — исходный код; образы контейнеров собираются на месте установки из этого кода (раздел 5.3). Все команды приведены в виде, пригодном для выполнения на подготовленной машине без обращения к внутренним ресурсам разработчика.

Документ описывает два способа установки: основной, посредством контейнеризации (раздел 5), и альтернативный, на подготовленный сервер с PHP и PostgreSQL без контейнеризации (раздел 8). Сведения о версиях, портах и командах приведены дословно по составу поставки на дату редакции документа.

Платформа обучения информационной безопасности — веб-приложение для организации корпоративного обучения и проверки знаний работников. Обучающий центр создаёт учебные программы, назначает их работникам, работники изучают материалы и проходят тестирование, платформа фиксирует результаты в форме, пригодной для предъявления: сертификатов, протоколов проверки знаний и неизменяемого журнала юридически значимых событий.

Экземпляр программного обеспечения устанавливается в контуре эксплуатирующей организации: отдельная инсталляция на собственных вычислительных мощностях либо арендованном сервере, без разделения по организациям-арендаторам. Настоящий документ описывает установку одного такого экземпляра на подготовленной машине, ранее платформу не эксплуатировавшей («чистая машина»).

Изложение не предполагает доступа читателя к внутреннему конвейеру непрерывной интеграции правообладателя (система сборки, собственный реестр контейнерных образов, внутренние учётные записи разработки). Все действия выполняются самостоятельно на подготовленной машине средствами, входящими в состав дистрибутива, либо общедоступными средствами операционной системы.

Для экспертизы правообладатель предоставляет эксперту удалённый доступ к развёрнутому экземпляру. Порядок доступа и реквизиты приводятся в экземпляре настоящей инструкции, передаваемом эксперту, и в открытом экземпляре не публикуются.

Условные обозначения

Хост — подготовленная машина (физический сервер, виртуальная машина или облачный инстанс), на которую устанавливается экземпляр.

Домен — сетевое имя, по которому экземпляр обращён к пользователям; при установке для внутренней проверки допустимо обращение по IP-адресу или имени **localhost**.

Команды приведены в форме, выполняемой в оболочке POSIX (bash/sh) на хосте. Строки, начинающиеся с **docker compose exec -T app**, выполняются внутри контейнера приложения через оболочку хоста и её ввода не требуют.

2. Требования к аппаратному обеспечению

2.1. Состав нагрузки, из которого выведены требования

Требования рассчитаны по фактическому составу служб поставки (раздел 4) и значениям, заданным в конфигурации по умолчанию, а не по абстрактной оценке. Определяющие факторы:

- обработчик очередей и сервер приложения — php-fpm с **pm.max_children = 25** и **memory_limit = 512 МБ** на процесс (**docker/php/www.conf**, **docker/php/php.ini**); фактически измеренный пик потребления памяти процессом php на самой ресурсоёмкой операции (приём, антивирусная проверка и подсчёт контрольной суммы видеофайла 500 МБ) — 81 МБ, то есть реальное потребление далеко не достигает установленного предела;
- кэш байткода PHP резервирует общую разделяемую память независимо от числа процессов: 512 МБ (**opcache.memory_consumption**) плюс 32 МБ (**opcache.interned_strings_buffer**) плюс 128 МБ (**opcache.jit_buffer_size**) — около 672 МБ разделяемой памяти суммарно;
- обработчик очередей Horizon в производственном окружении масштабируется автоматически до 10 параллельных процессов с пределом 128 МБ каждый (**config/horizon.php**, секция **environments.production**) — до 1,3 ГБ при полной нагрузке;
- антивирусная проверка загружаемых файлов — опциональная служба (раздел 4.2); при включении выполняет до 12 потоков проверки одновременно (**MaxThreads** в конфигурации clamd, **docker-compose.yml**) и требует времени на первичную загрузку базы сигнатур при первом запуске;
- система управления базами данных и кэш/очередь — PostgreSQL и Redis без явно заданных в поставке пределов памяти: их объём планируется по общей практике эксплуатации, конфигурация поставки предельных значений не задаёт.

2.2. Рекомендуемая конфигурация

Ресурс	Минимум	Рекомендуется	Обоснование
Процессор	4 виртуальных ядра	8 виртуальных ядер	До семи параллельно работающих служб (раздел 4.2, при включённой опциональной антивирусной проверке), которая сама по себе многопоточная и ресурсоёмкая по процессору операция; запас нужен для одновременных загрузок вложений несколькими работниками

Ресурс	Минимум	Рекомендуется	Обоснование
Оперативная память	8 ГБ	16 ГБ	Разделяемая память кэша байткода (около 672 МБ) и до 25 процессов php-fpm плюс до 10 процессов Horizon практически не достигают заданных пределов (см. 2.1), но сумма пределов и память СУБД, кэша и антивирусного движка требуют запаса против пиковой одновременной нагрузки
Дисковое пространство	40 ГБ, твердотельный накопитель	от 100 ГБ, твердотельный накопитель	Контейнерные образы, база данных, журналы и резервные копии (раздел 9 — резервные копии хранятся 30 суток ежедневно и 12 месяцев ежемесячно, **BACKUP_KEEP_DAILY_DAYS**, **BACKUP_KEEP_MONTHLY_MONTHS**). Вложения (документы до 100 МБ, видео до 500 МБ) при использовании внешнего S3-совместимого хранилища на диск хоста не попадают — см. раздел 6.4
Сеть	Обычный канал доступа в интернет на время установки и обновления		При установке и обновлении нужен доступ к источникам образов и пакетов. При эксплуатации необходим доступ к настроенным SMTP, S3 и мониторингу, а при включённой антивирусной проверке (раздел 4.2) — также к источнику обновлений антивирусных баз. Push использует службы доставки браузеров, если включён. Замкнутый контур требует размещения или зеркалирования этих ресурсов внутри него

Минимум 4 ядра и 8 ГБ ОЗУ рассчитан на демонстрационный экземпляр без параллельной нагрузки. Диск 40 ГБ допустим только для такой проверки при внешнем S3; для рабочего контура требуется от 100 ГБ, как в разделе 7.2 описания функциональных характеристик. Конфигурация 8 ядер и 16 ГБ ОЗУ — отправная точка для эксплуатации; достаточность ресурсов подтверждается нагрузочным испытанием.

3. Требования к программному обеспечению

3.1. Способ установки посредством контейнеризации (основной)

Компонент	Требование
Операционная система хоста	Для рабочего контура используется GNU/Linux с Docker Engine. Windows и macOS возможны для разработки через среду Linux. ОС контейнера отличается от ОС хоста: рабочие образы приложения и служб — на Alpine Linux
Docker Engine	С включённым BuildKit (в текущих версиях включён по умолчанию) — сборочные файлы поставки (**docker/php/Dockerfile**, **docker/nginx/Dockerfile**) объявляют синтаксис **# syntax=docker/dockerfile:1.7**. Установка проверена на Docker Engine 25.0
Docker Compose	Compose второго поколения, вызываемый как подкоманда **docker compose** (не отдельная утилита **docker-compose** первого поколения). Установка проверена на Docker Compose v2.24
Свободные сетевые порты хоста	Порт, публикуемый веб-сервером наружу (в поставке по умолчанию не опубликован — см. раздел 5.4), и порт СУБД, если требуется доступ к ней извне контейнерной сети

3.2. Способ установки без контейнеризации (альтернативный, раздел 8)

Компонент	Требование
Операционная система	Дистрибутив Linux. Сборочные и эксплуатационные сценарии поставки (**docker/php/entrypoint.sh**, **deploy/deploy.sh**, **deploy/framework-cache.sh**) — сценарии командной оболочки POSIX, рассчитанные на среду Linux; под Windows или macOS без контейнеризации не проверялись
PHP	Версия не ниже 8.3 (ограничение **composer.json**: **"php": "^8.3"**). Поставка собрана и проверена на PHP 8.4. Обязательные расширения перечислены в разделе 8.2
Веб-сервер	Любой веб-сервер, умеющий проксировать запросы в PHP-FPM по протоколу FastCGI (nginx, Apache с mod_proxy_fcgi). Поставка использует и проверена на nginx 1.27
Composer	Версия 2 — для установки зависимостей PHP

Node.js	Версия 22 — только на время сборки клиентской части (раздел 5.7); в эксплуатации не требуется, собранные файлы статичны

3.3. Версии компонентов данных, зафиксированные в поставке

Компонент	Версия	Примечание
PostgreSQL	17 (образ **postgres:17-alpine**, проверено — 17.10)	Требуются расширения **pg_trgm**, **unaccent**, **pgcrypto** (раздел 4.1, раздел 9)
Redis	8.10 (образ **redis:8.10-alpine**, проверено — 8.10.0)	Используется как хранилище кэша, очередей заданий и, при недоступности PostgreSQL для резервного варианта, сессий (драйвер **failover**)
PHP	8.4 (образ **php:8.4-fpm-alpine**)	Минимальное поддерживаемое приложением значение — 8.3
Node.js	22 (образ **node:22-alpine**)	Только для сборки клиентской части, в поставку конечного экземпляра не входит
Laravel	13 (ограничение **composer.json**: **^13.8**, проверено — 13.20.0)	Фреймворк серверной части

4. Состав дистрибутива и состав служб

4.1. Состав дистрибутива

Дистрибутив — архив исходного кода приложения, получаемый командой **make dist**: она формирует архив командой системы контроля версий (**git archive**) по состоянию текущего изменения. Из архива исключены файлы разработки и внутренняя документация правообладателя (перечень исключений задан атрибутом **export-ignore** файла **.gitattributes**) — они не требуются ни для сборки, ни для установки, ни для работы продукта на стороне эксплуатирующей организации. Сборка контейнерных образов и клиентской части выполняется из исходного кода на хосте (раздел 5.3), а не путём получения готовых образов из внутреннего реестра правообладателя — это и делает установку независимой от конвейера непрерывной интеграции. Значимые для установки составляющие дистрибутива:

Составляющая	Назначение
Исходный код серверной части (app/, routes/, config/, database/)	Бизнес-логика, маршруты, конфигурация, миграции и начальные данные (сиды)
Манифесты зависимостей (composer.json, composer.lock, package.json, package-lock.json)	Точная фиксация версий сторонних библиотек серверной и клиентской части
Исходный код клиентской части (resources/js, resources/css)	Собирается инструментом Vite в статические файлы при сборке образа (раздел 5.7)
Сборочные файлы (docker/, файл docker-compose.yml)	Описание контейнерных образов и состава служб; используются при установке посредством контейнеризации
Шаблон переменных окружения (.env.example)	Полный перечень настраиваемых параметров с пояснениями и значениями по умолчанию для среды разработки; на боевом экземпляре часть значений подлежит замене (раздел 5.2)

4.2. Состав служб

В файле **docker-compose.yml** описаны семь служб. Первые шесть строк таблицы — обязательные службы рабочего контура, седьмая (**clamav**) — опциональная. Исходящая почта и объектное хранилище файлов в состав поставки не входят: это внешние службы, которые предоставляет эксплуатирующая организация, — SMTP-сервер (раздел 6.3) и S3-

совместимое хранилище (раздел 6.4).

Служба	Образ / сборка	Роль	Обязательность
app	Собирается из **docker/php/Dockerfile**, цель **prod**	Сервер приложения (php-fpm): обработка HTTP-запросов, доступных через веб-сервер nginx	Обязательна
nginx	Собирается из **docker/nginx/Dockerfile** поверх образа службы app	Веб-сервер: отдаёт статические файлы, проксирует динамические запросы в службу app	Обязательна
postgres	**postgres:17-alpine**	Основное хранилище данных	Обязательна
redis	**redis:8.10-alpine**	Кэш, очередь фоновых заданий, хранилище сессий	Обязательна
horizon	Тот же образ, что и служба app; отдельный процесс (**supervisord** запускает **php artisan horizon**)	Обработка фоновых заданий из очереди: уведомления, выпуск сертификатов, изменение статусов назначений, обезличивание данных	Обязательна
scheduler	Тот же образ, что и служба app; процесс **php artisan schedule:work**	Планировщик: запускает регламентные задачи по расписанию (раздел 7.2)	Обязательна
clamav	**clamav/clamav:1.4**	Антивирусная проверка загружаемых файлов перед сохранением во вложение	Опциональна: включается профилем **antivirus** и переменной **ANTIVIRUS_ENABLED=true** (раздел 5.4). Выключена — загрузки не проверяются; включена — недоступность сканера отклоняет загрузку, а не пропускает непроверенный файл

5. Порядок установки

Раздел описывает основной способ установки — посредством контейнеризации, без обращения к внутреннему конвейеру непрерывной интеграции и реестру контейнерных образов правообладателя: образы собираются на хосте из исходного кода дистрибутива. Установка без контейнеризации описана отдельно в разделе 8.

5.1. Получение дистрибутива

Разместите исходный код дистрибутива в каталоге на хосте, например **/opt/learning-platform**, и перейдите в него. Архив дистрибутива содержит каталог верхнего уровня **learning-platform-ВЕРСИЯ**; ключ **--strip-components=1** распаковывает его содержимое непосредственно в целевой каталог:

```
mkdir -p /opt/learning-platform
tar -xzf learning-platform-ВЕРСИЯ.tar.gz \
  -C /opt/learning-platform --strip-components=1
cd /opt/learning-platform
```

Дальнейшие команды раздела выполняются из корневого каталога дистрибутива.

5.2. Настройка переменных окружения

Скопируйте шаблон переменных окружения и заполните значения, специфичные для устанавливаемого экземпляра:

```
cp .env.example .env
```

Значения, обязательные к пересмотру перед запуском:

Переменная	Что задать
APP_ENV	**production** для боевого экземпляра
APP_DEBUG	**false** на боевом экземпляре — включённая отладка раскрывает внутреннее устройство приложения при ошибке
APP_URL	Адрес, по которому экземпляр будет доступен пользователям, со схемой (например, **https://learning.example.org**)
APP_KEY	Оставить пустым — значение формируется и вписывается на шаге 5.5
DB_DATABASE, DB_USERNAME, DB_PASSWORD	Имя базы данных, имя пользователя и пароль для контейнера PostgreSQL, поднимаемого этим же файлом **docker-compose.yml**

REDIS_PASSWORD	Пароль для доступа к Redis. Значение по умолчанию **null** — доступ без пароля: служба redis работает только во внутренней сети контейнеров и наружу не опубликована. Если пароль задан, служба redis должна запускаться с тем же паролем — см. файл override в разделе 5.4; иначе приложение и обработчик очередей не подключатся к Redis
MAIL_*	Реквизиты SMTP-сервера, предоставленного эксплуатирующей организацией, либо конфигурация после запуска командой **mail:configure** (раздел 6.3) — SMTP-сервер в состав поставки не входит
AWS_*	Реквизиты S3-совместимого хранилища, предоставленного эксплуатирующей организацией (раздел 6.4), — хранилище в состав поставки не входит
ANTIVIRUS_ENABLED	По умолчанию **false** — антивирусная проверка загружаемых файлов выключена. **true** включает опциональную проверку (раздел 4.2, служба clamav) — решение эксплуатирующей организации, а не обязательное значение
FEATURE_EMAIL_OTP	По умолчанию **true** — при входе после пароля запрашивается код подтверждения из письма (email-OTP), для всех учётных записей платформы. **false** отключает запрос кода полностью — решение эксплуатирующей организации на уровне всей платформы, а не отдельного сотрудника
PASSWORD_UNCOMPROMISED	По умолчанию **false** — пароль не проверяется по перечню скомпрометированных паролей. **true** включает такую проверку: приложение обращается к внешнему сервису и передаёт первые пять символов SHA-1-хеша пароля (k-анонимность, сам пароль наружу не передаётся); при недоступности сервиса проверка не отклоняет пароль, а пропускает его без проверки
INTEGRATIONS_ENCRYPTION_KEY	После сборки образа сформировать отдельный ключ по разделу 5.5 и сохранить в **.env** на хосте до запуска приложения. Значение не должно совпадать с APP_KEY
COMPOSE_FILE	Значение по умолчанию в поставке — только **docker-compose.yml**. Задать **docker-compose.yml:docker-compose.override.yml**: файл override создаётся на шаге 5.4. Все дальнейшие команды выполняются из каталога дистрибутива с этим **.env**
BACKUP_PATH	**/var/www/html/storage/app/backups** — каталог резервных копий базы данных на постоянном томе, который подключается файлом override (раздел 5.4)
MONITORING_TOKEN	Токен защищённого маршрута **/internal/monitoring** (раздел 7.3); пустое значение отключает маршрут

Особенность Windows

В Linux список **COMPOSE_FILE** разделяется двоеточием. Для работы с тем же списком в Windows задайте **COMPOSE_PATH_SEPARATOR=:** в **.env**. Рабочая установка в этой инструкции описана для GNU/Linux. Используется пара **docker-compose.yml** и **docker-compose.override.yml**.

5.3. Сборка образов без обращения к внутреннему конвейеру

Форма поставки — исходный код (раздел 4.1), а не готовые образы контейнеров: правообладатель в собственной эксплуатации использует образы, собранные внутренним конвейером непрерывной интеграции и сохранённые в его реестре контейнерных образов (раздел 3 описания технических средств хранения исходного текста и объектного кода), но для установки экземпляра это не требуется и не используется. Образы собираются на хосте непосредственно из сборочных файлов дистрибутива, без обращения к внутреннему конвейеру и без доступа к этому реестру:

```
docker build --target prod \
    --build-arg WWWUSER=1000 --build-arg WWWGROUP=1000 \
    -f docker/php/Dockerfile \
    -t learning-platform/app:dev .

docker build \
    --build-arg APP_IMAGE=learning-platform/app:dev \
    -f docker/nginx/Dockerfile \
    -t learning-platform/nginx:dev .
```

Первая команда строит сервер приложения: устанавливает системные зависимости и расширения PHP, ставит зависимости Composer без пакетов разработки, попутно собирает клиентскую часть отдельным слоем сборки (раздел 5.7) и включает собранные файлы в готовый образ. Вторая команда строит веб-сервер, копируя из уже собранного образа приложения каталог со статическими файлами и манифестом сборки — так набор файлов у веб-сервера и сервера приложения гарантированно совпадает.

5.4. Запуск контейнеров

После сборки образов по разделу 5.3 создайте **docker-compose.override.yml**. В нём веб-порт публикуется только на loopback — для подключения обратного прокси на том же хосте, — а приложению, обработчику очередей и планировщику подключается постоянный том для резервных копий базы данных:

```
services:
  nginx:
    ports:
      - "127.0.0.1:8080:80"
  app:
    volumes:
      - backups:/var/www/html/storage/app/backups
  horizon:
    volumes:
      - backups:/var/www/html/storage/app/backups
  scheduler:
    volumes:
      - backups:/var/www/html/storage/app/backups
volumes:
  backups:
```

Списки томов объединяются с основным файлом по пути монтирования: том журналов **storage-logs** из **docker-compose.yml** сохраняется. Если в **.env** задан **REDIS_PASSWORD** (раздел 5.2), добавьте в раздел **services** того же файла запуск redis с этим паролем:

```
  redis:
    command: ["redis-server", "--requirepass", "${REDIS_PASSWORD}"]
```

В **.env** задайте **COMPOSE_FILE=docker-compose.yml:docker-compose.override.yml** и **BACKUP_PATH=/var/www/html/storage/app/backups**. Команды ниже учитывают оба файла. При использовании флага **-f** каждый файл требуется указать явно; один **-f docker-compose.yml** исключит override. После этого сохраните ключи в **.env** по разделу 5.5, до запуска контейнеров.

Данные PostgreSQL хранятся в именованном томе **postgres-data**, заданном в **docker-compose.yml**, дополнительной настройки не требуют. Каталог резервных копий внутри удаляемого контейнера не подходит: копии пропали бы при пересоздании контейнера.

Загрузите образы инфраструктурных служб отдельно. Флаг **--pull never** запрещает их загрузку при запуске и требует, чтобы они уже были на хосте:

```
docker compose config --quiet
docker compose pull postgres redis
docker compose up -d --pull never
```

Антивирусная проверка загрузок — опционально

Служба **clamav** (раздел 4.2) в базовый запуск выше не входит: без неё загруженные файлы не проверяются на вирусы. Чтобы включить проверку, добавьте профиль **antivirus** и установите **ANTIVIRUS_ENABLED=true**:

```
docker compose --profile antivirus pull postgres redis clamav
docker compose --profile antivirus up -d --pull never
```

Первый запуск ClamAV в этом случае может занимать до 10 минут из-за загрузки базы сигнатур; проверьте готовность службы, прежде чем считать загрузку файлов работоспособной. Обновление сигнатур штатно идёт с серверов проекта ClamAV из-за рубежа; если эти серверы недоступны из сети установки, эксплуатирующая организация обеспечивает собственное зеркало обновлений.

Настройте HTTPS на внешнем обратном прокси с сертификатом домена и передачей запросов на **127.0.0.1:8080**. Если прокси работает контейнером в общей сети, публикация порта на хосте не нужна. Лимиты загрузки и тайм-ауты приведены в разделе 8.5. Пользователи и эксперты обращаются к HTTPS-адресу, указанному в **APP_URL**.

5.5. Ключи приложения

На чистой установке ключи формируются после сборки образа (раздел 5.3), но до запуска контейнеров (раздел 5.4). Выполните команду дважды: отдельно для **APP_KEY** и **INTEGRATIONS_ENCRYPTION_KEY**:

```
docker compose run --rm --no-deps \
  --entrypoint php app -r \
  'echo "base64:".base64_encode(random_bytes(32)).PHP_EOL;'
```

Образ собран на хосте по разделу 5.3 и из реестра не запрашивается (**pull_policy: missing** в **docker-compose.yml**).

Каждое полученное значение сохраните в соответствующую переменную **.env** на хосте. Ключи должны различаться. Проверьте, что этот файл передаётся в **app**, **horizon** и **scheduler**. Не сохраняйте ключ только внутри контейнера: при его пересоздании значение может исчезнуть. Ключи резервируются отдельно с ограниченным доступом.

На работающем экземпляре существующие ключи не перегенерируются. Их смена требует отдельной процедуры ротации: иначе ранее зашифрованные данные могут стать недоступны.

5.6. Миграции базы данных

После запуска приложения проверьте состояние миграций. Если точка входа уже применила их, повторное выполнение не требуется:

```
docker compose exec -T app php artisan migrate:status
docker compose exec -T app php artisan migrate --force
```

Для PostgreSQL необходимы расширения **pg_trgm**, **unaccent** и **pgcrypto**. Скрипты каталога **docker/postgres/init/** выполняются только при первичном создании кластера. Для уже существующей базы данных (повторное использование существующего тома) расширения создаются отдельно — порядок указан в разделе 9.

5.7. Сборка клиентской части

При контейнеризации клиентские стили, скрипты, шрифты и набор значков собираются на стадии **frontend** многоступенчатой сборки образа приложения (раздел 5.3). Результат включается в готовый образ и копируется в образ nginx. Дополнительный запуск npm на рабочем контейнере не нужен.

Установка без контейнеризации описана в разделе 8.3.

6. Первичная инициализация

6.1. Начальные данные

Начальные данные (роли и права доступа) заполняются командой заполнения базы данных. На боевом экземпляре (**APP_ENV=production**) выполняется только базовый набор — тестовые и демонстрационные учётные записи не создаются:

```
docker compose exec -T app php artisan db:seed --force
```

Команда идемпотентна (использует **firstOrCreate**): повторный запуск не создаёт дублей и безопасен. Создаются роли **admin**, **manager**, **user** и соответствующая им матрица прав доступа.

6.2. Учётная запись администратора

Публичная регистрация на боевом экземпляре выключена настройкой (**FEATURE_REGISTRATION=false** по умолчанию), поэтому первая учётная запись заводится штатной консольной командой:

```
docker compose exec -T app php artisan auth:create-admin \
  admin@example.org "Администратор"
```

Команда заводит учётную запись с ролью «Администратор» и генерирует стойкий пароль, удовлетворяющий действующей политике (не короче 12 символов, заглавные и строчные буквы, цифры и специальные символы). Сгенерированный пароль выводится в консоль один раз, повторно получить его нельзя — передайте его администратору по защищённому каналу. Поскольку пароль остаётся в истории команд терминала, платформа потребует сменить его при первом входе.

Пароль можно задать явно вместо автогенерации ключом **--password=**; в этом случае платформа не требует его смены при первом входе, а сам пароль проверяется той же политикой, включая проверку по перечню скомпрометированных паролей, если она включена (раздел 8 описания функциональных характеристик). Ключ **--promote** назначает роль администратора уже существующей учётной записи вместо создания новой:

```
docker compose exec -T app php artisan auth:create-admin \
  sotrudnik@example.org "Имя Сотрудника" --promote
```

6.3. Исходящая почта

По умолчанию (**FEATURE_EMAIL_OTP=true**, раздел 5.2) вход в платформу защищён кодом подтверждения из письма (email-OTP), поэтому работающая почта нужна с первого входа.

Реквизиты SMTP-сервера задаются двумя равнозначными способами.

Через переменные окружения — значения **MAIL_HOST**, **MAIL_PORT**, **MAIL_USERNAME**, **MAIL_PASSWORD**, **MAIL_FROM_ADDRESS** в **.env** до запуска контейнеров (раздел 5.2).

Консольной командой после запуска — реквизиты сохраняются в базе данных в зашифрованном виде и применяются немедленно, без пересоздания контейнеров:

```
docker compose exec -T app php artisan mail:configure \
  --host=smtp.example.org --port=587 \
  --username=USER --password=PASSWORD \
  --from-address=noreply@example.org \
  --from-name="Платформа обучения информационной безопасности" --test
```

Флаг **--test** проверяет соединение и вход сразу после сохранения реквизитов, не отправляя письмо. Текущее состояние — командой с флагом **--show** (пароль в выводе маскируется).

6.4. Хранилище файлов

Файловые вложения (документы, изображения, видео, сертификаты, аватары) хранятся в S3-совместимом объектном хранилище, адресуемом переменными **AWS_ENDPOINT**, **AWS_ACCESS_KEY_ID**, **AWS_SECRET_ACCESS_KEY**, **AWS_BUCKET** в **.env**. Подходит любое S3-совместимое хранилище — приложение не зависит от конкретной реализации.

Контейнер (бакет) в хранилище создаётся автоматически при старте контейнера приложения; ручной запуск нужен только после смены настроек хранилища на уже работающем экземпляре или после удаления контейнера:

```
docker compose exec -T app php artisan storage:ensure-bucket
```

Объектное хранилище в состав поставки не входит: эксплуатирующая организация указывает адрес и реквизиты уже существующего S3-совместимого хранилища либо разворачивает собственное — приложение не зависит от конкретной реализации.

6.5. Push-уведомления (опционально)

Push-уведомления в браузер — дополнительный канал доставки, не влияющий на основную функциональность: при выключенном канале уведомления доставляются электронной почтой и отображаются внутри интерфейса. Канал выключен, пока не заданы ключи VAPID:

```
docker compose exec -T app php artisan push:vapid-keys
```

Команда выводит публичный и приватный ключи. Запишите их в переменные **VAPID_PUBLIC_KEY** и **VAPID_PRIVATE_KEY** файла **.env**; **VAPID_SUBJECT** задайте в форме **mailto:АДРЕС**, где адрес — контактный адрес электронной почты оператора экземпляра. Применение — **docker compose up -d --pull never --force-recreate --no-deps app horizon**

scheduler (правки **.env** подхватываются только пересозданием контейнеров, раздел 9). Смена уже заданной пары ключей обесценивает все существующие подписки браузеров.

7. Проверка работоспособности

7.1. Проверка веб-приложения

Проверка	Ожидаемый результат
curl -fsS -o /dev/null -w '%{http_code}\n' http://XOCT/up	Код ответа **200** — встроенная проверка работоспособности фреймворка
Открыть http://XOCT/login в браузере	Отображается форма входа на русском языке
docker compose ps	Все шесть обязательных служб (раздел 4.2), а при включённой антивирусной проверке — и **clamav**, в состоянии **Up**; службы **postgres** и **redis**, для которых заданы проверки состояния, — дополнительно **healthy**. Состояние **Up** контейнера horizon не доказывает работу обработчика очередей — она проверяется командой **horizon:status** (раздел 7.2)
Вход учётной записью администратора (раздел 6.2)	После ввода пароля запрашивается код из письма (email-OTP); письмо приходит на указанный при создании учётной записи адрес (раздел 6.3) — подтверждает, что почта настроена и путь очереди фоновых заданий работает end-to-end (письмо ставится в очередь и обрабатывается службой horizon)

7.2. Проверка фоновых задач и очередей

Проверка	Ожидаемый результат
docker compose exec -T app php artisan horizon:status	Ответ **Horizon is running** — служба обработки очередей активна
docker compose exec -T app php artisan queue:failed	Ответ **No failed jobs found** — упавших фоновых заданий нет

docker compose exec -T app php artisan schedule:list	Выводится список задач и время их следующего запуска (заполнение назначений по правилам, открытие циклов переаттестации, ежедневный резервный экземпляр базы данных, обезличивание данных уволенных работников и другие — заданы в **routes/console.php**). Это проверка регистрации расписания в приложении, а не работы контейнера **scheduler**: пустой список требует проверки конфигурации и регистрации задач, а запуск планировщика проверяется отдельно командами **ps** и **logs**
docker compose logs --tail=50 scheduler	Просмотрите ошибки в логе. Пустой лог сам по себе не доказывает остановку процесса — проверьте дополнительно **docker compose ps**

7.3. Проверка состояния системы

Защищённый маршрут **/internal/monitoring** возвращает сводное состояние ключевых зависимостей. Токен задаётся переменной **MONITORING_TOKEN** в **.env** (раздел 5.2; пустое значение отключает маршрут, правка вступает в силу после пересоздания контейнеров). Подставьте действительные адрес и токен вместо обозначений примера:

```
curl -fsS -H "X-Monitoring-Token: TOKEN" \
  "https://ХОСТ/internal/monitoring"
```

На самом хосте, в обход обратного прокси, тот же запрос выполняется по адресу, опубликованному в разделе 5.4:

```
curl -fsS -H "X-Monitoring-Token: TOKEN" \
  "http://127.0.0.1:8080/internal/monitoring"
```

На исправно установленном экземпляре поля **database.ok**, **redis.ok**, **horizon.running** и **object_storage.ok** имеют значение **true**. Значение **object_storage.configured=false** указывает на незаполненные параметры **AWS_*** (раздел 6.4); **bucket_exists=false** — на отсутствие бакета.

Создание бакета — командой **docker compose exec -T app php artisan storage:ensure-bucket**, порядок настройки хранилища приведён в разделе 6.4. Проверяйте также фактическую загрузку и чтение файла: один ответ диагностики не заменяет сквозную проверку.

Внешний доступ выполняется по HTTPS-адресу экземпляра. HTTP-адреса в таблицах раздела 7 используются только для внутренней проверки за TLS-прокси. Токен не передаётся по незащищённому внешнему соединению.

8. Установка без использования контейнеризации

8.1. Применимость

Приложение — стандартное веб-приложение на PHP и не привязано к контейнеризации архитектурно: контейнеризация в поставке используется для повторяемости и изоляции служб, а не потому, что приложение требует именно её. Установка на подготовленный сервер с PHP и PostgreSQL без Docker технически возможна и описана ниже. Организационные сценарии (сборка образов, обновление единым файлом) при этом теряются — администратор берёт на себя то, что при контейнеризации делают сборочные файлы поставки и точка входа контейнера (**docker/php/entrypoint.sh**).

8.2. Программное окружение

Установить на сервер:

- PHP 8.4 (минимум 8.3) с расширениями: **pdo_pgsql**, **pgsql**, **bcmath**, **intl**, **gd** (со сборкой freetype, jpeg, webp), **exif**, **pcntl**, **zip**, **opcache**, **sodium**, **redis** (через PECL) — точный перечень зафиксирован в **docker/php/Dockerfile**, стадия **base**;
- Composer 2;
- Node.js 22 — только на время сборки клиентской части (раздел 8.3), в дальнейшей эксплуатации не требуется;
- PostgreSQL 17 с правами на создание расширений **pg_trgm**, **unaccent**, **pgcrypto** в целевой базе данных;
- Redis 8.10;
- nginx либо иной веб-сервер, проксирующий запросы в PHP-FPM;
- демон антивирусной проверки ClamAV (пакет **clamav-daemon** либо аналог дистрибутива) — опционально, только если проверка загружаемых файлов включается (раздел 4.2).

8.3. Порядок установки

1. Разместить исходный код дистрибутива в целевом каталоге на сервере.
2. Настроить **.env** — те же переменные, что и в разделе 5.2, со значениями хостов и портов уже установленных служб (**DB_HOST**, **REDIS_HOST** и другие — вместо имён служб Docker Compose указываются адреса или **127.0.0.1**, если служба на том же сервере).
3. Установить зависимости PHP без пакетов разработки:

```
composer install --no-dev --prefer-dist --no-interaction
composer dump-autoload --no-dev --optimize
```

4. Собрать клиентскую часть:

```
npm ci --no-audit --no-fund
npm run build
```

Команда собирает файлы в **public/build** (стили, скрипты, шрифты, набор значков) и записывает манифест сборки, по которому приложение находит собранные файлы. После сборки Node.js для работы приложения больше не требуется.

5. Сформировать **APP_KEY** и **INTEGRATIONS_ENCRYPTION_KEY** — принцип тот же, что в разделе 5.5, но без **docker compose run**, напрямую (выполнить дважды, значения не должны совпадать):

```
php -r 'echo "base64:".base64_encode(random_bytes(32)).PHP_EOL;'
```

Применить миграции и заполнить начальные данные — команды те же, что в разделах 5.6, 6.1, выполняются без **docker compose exec**, напрямую: **php artisan migrate --force**, **php artisan db:seed --force**.

6. Создать символическую ссылку публичного диска и контейнер объектного хранилища:

```
php artisan storage:link
php artisan storage:ensure-bucket
```

7. Собрать кэши каркаса приложения (конфигурация, маршруты, события, шаблоны) — при контейнеризации это делает точка входа контейнера при каждом старте (**deploy/framework-cache.sh**); без контейнеризации выполняется вручную после каждого изменения конфигурации или кода:

```
php artisan config:cache
php artisan route:cache
php artisan event:cache
php artisan view:cache
```

8. Создать первую учётную запись администратора (раздел 6.2), настроить почту и хранилище файлов (разделы 6.3, 6.4).

8.4. Организация фоновых процессов

При контейнеризации обработку очереди и планировщик держат в рабочем состоянии отдельные контейнеры с автоматическим перезапуском (**restart: unless-stopped**). Без контейнеризации ту же роль берёт на себя система инициализации сервера — например, **systemd**: модуль службы для обработчика очередей с командой запуска **php artisan horizon** и **Restart=always**.

Планировщик в контейнере запускается процессом **php artisan schedule:work**, удерживающим цикл проверки задач в собственном процессе. Без контейнеризации предпочтительна классическая для Laravel схема — запись в **cron**, выполняющая проверку раз в минуту. Вызов идёт по полному пути к файлу **artisan**, без предварительного перехода в каталог, — так запись остаётся одной строкой crontab независимо от длины пути установки:

```
* * * * * php /opt/learning-platform/artisan schedule:run >> /dev/null 2>&1
```

8.5. Веб-сервер

Конфигурация nginx поставки (**docker/nginx/default.conf**) — рабочий образец для переноса на самостоятельно устанавливаемый веб-сервер: корневой каталог **public/**, единственная точка входа **index.php**, два примечательных отклонения от значений по умолчанию:

- общий предел тела запроса — 16 МБ, но для маршрута загрузки вложений (форма адреса **^/livewire-[0-9a-f]{8}/upload-file$**, префикс зависит от **APP_KEY** и свой на каждом экземпляре) предел поднят до 520 МБ — документы до 100 МБ и видео до 500 МБ плюс служебные накладные расходы;
- время ожидания ответа PHP на этом же маршруте увеличено до 600 секунд — антивирусная проверка крупного файла может занимать до минуты.

Если перед приложением дополнительно стоит внешний обратный прокси-сервер с завершением TLS-соединения (типовая схема боевой эксплуатации — раздел 5.4), те же два значения обязаны быть подняты и на нём; значение по умолчанию большинства веб-серверов (около 1 МБ) отклонит загрузку видео раньше, чем запрос дойдёт до приложения:

```
client_max_body_size 520M;
proxy_read_timeout 600s;
proxy_request_buffering off;
```

9. Типовые неполадки при установке и их устранение

Признак	Причина и устранение
Ошибка вида CreateFile ...docker-compose.yml: docker-compose.override.yml: The system cannot find the file specified	Хост — Windows, переменная **COMPOSE_PATH_SEPARATOR** не задана в **.env**: Docker Compose по умолчанию ждёт разделителем **;** и читает список файлов как одно имя. Задать **COMPOSE_PATH_SEPARATOR=:** (раздел 5.2)
Служба app либо nginx не запускается: Docker пытается получить образ из внешнего реестра и завершается ошибкой доступа	Образ не собран локально под именем и тегом, которые ожидает Compose (**REGISTRY_IMAGE_APP**, **REGISTRY_IMAGE_NGINX** и **IMAGE_TAG** в **.env** обязаны совпадать с именами и тегами, использованными при сборке на шаге 5.3, — по умолчанию **learning-platform/app:dev** и **learning-platform/nginx:dev**). Политика **pull_policy: missing** запрашивает отсутствующий локально образ из внешнего реестра, а там такого образа нет. Устранение: повторить сборку раздела 5.3 с именами и тегами из **.env** и запускать контейнеры с флагом **--pull never** (раздел 5.4) как дополнительной гарантией того, что образ не будет запрошен извне
При обращении к странице — ошибка о недостающей таблице базы данных либо сообщение об отсутствующем расширении pg_trgm, unaccent или pgcrypto	Миграции не применены (раздел 5.6), либо каталог данных PostgreSQL не был пуст при первом старте контейнера — файлы каталога **docker/postgres/init/** выполняются образом только при инициализации пустого кластера. Устранение: подключиться к базе и создать расширения вручную — **docker compose exec -T postgres psql -U ПОЛЬЗОВАТЕЛЬ -d БАЗА -c 'CREATE EXTENSION IF NOT EXISTS pg_trgm; CREATE EXTENSION IF NOT EXISTS unaccent; CREATE EXTENSION IF NOT EXISTS pgcrypto;'**, затем повторить миграции
При входе не приходит код подтверждения (email-OTP)	Почта не настроена либо недоступна (раздел 6.3). Проверить: **docker compose exec -T app php artisan mail:configure --show** и **docker compose exec -T app php artisan queue:failed** — письмо доставляется через очередь фоновых заданий, упавшая доставка попадает в список упавших заданий
Загрузка файла отклоняется с ошибкой размера уже на небольшом файле	Предел тела запроса не согласован на всех уровнях цепочки (внешний обратный прокси-сервер, если есть, — nginx контейнера — php-fpm — приложение). Значения и обоснование — раздел 8.5

Загрузка файла отклоняется с сообщением о недоступности проверки содержимого	**ANTIVIRUS_ENABLED=true**, а служба **clamav** не поднята (профиль **antivirus** не указан при запуске, раздел 5.4) либо ещё загружает базу сигнатур после первого старта — это ожидаемо занимает несколько минут. Поведение намеренное: при включённой проверке недоступность демона не пропускает файл, а отклоняет загрузку
Изменения в .env не влияют на поведение приложения	Контейнеры не пересозданы либо сохранён кэш конфигурации. После правки **.env** выполните **docker compose up -d --pull never --force-recreate** с теми же файлами Compose и теми же постоянными томами — для локально собранных образов флаг **--pull never** обязателен. Без контейнеризации — раздел 8.3, шаг 7 (пересобрать кэши каркаса вручную)
После пересоздания контейнеров исчезли ежедневные резервные копии базы данных	Резервные копии по умолчанию пишутся внутри контейнера (**storage/app/backups**), а файл **docker-compose.yml** монтирует отдельным томом только каталог логов (**storage/logs**) — каталог с резервными копиями монтированному тому не принадлежит и вместе с контейнером теряется. Устранение: смонтировать **storage/app/backups** отдельным томом (файл **docker-compose.override.yml** на хосте, раздел 5.4) либо задать **BACKUP_PATH** вне каталога приложения и выгружать копии за пределы хоста регулярным заданием
Обработчик очередей не запускается, в журнале службы horizon — ERR AUTH <password> called without any password configured	В **.env** задан **REDIS_PASSWORD**, а служба redis запущена без пароля. Добавьте запуск redis с паролем в файл override (раздел 5.4) и пересоздайте контейнеры, затем проверьте миграции (раздел 5.6): при первом запуске без доступа к Redis точка входа контейнера их не применяет
Страница открывается, интерактивные элементы не реагируют, в консоли браузера — ошибки вида Refused to evaluate a string as JavaScript	Не относится к первичной установке — признак того, что уже открытая вкладка браузера использует прежнюю версию клиентской сборки после последующего обновления экземпляра (раздел 10): её отпечаток не совпадает с версией, которую отдаёт сервер, и политика безопасности содержимого блокирует выполнение несовпадающего кода. В норме открытая вкладка обнаруживает это сама и предлагает обновиться полосой «Доступна новая версия платформы», без вмешательства администратора. Если полоса не появилась, жёсткая перезагрузка страницы (**Ctrl+Shift+R**) устраняет случай. Если не помогает — сверьте отпечаток сборки на сервере (**curl -fsS https://XOCT/app-release**) с версией на странице входа и в файле **/sw.js**: расхождение означает, что часть контейнеров ещё работает со старым кэшем конфигурации, устраняется командами **php artisan config:clear**, **php artisan route:clear** и пересозданием контейнеров (**docker compose up -d --pull never --force-recreate**)

10. Обновление установленного экземпляра

10.1. Обновление через контейнеризацию

1. Назначьте окно обслуживания. Сохраните используемые образы под отдельными тегами для отката, а также конфигурацию, ключи и резервные копии базы данных и файлов — копии должны находиться вне пересоздаваемых контейнеров. Приостановите пользовательские изменения, штатно завершите активные задания, остановите **horizon** и **scheduler**.
2. Получите новую версию исходного кода и соберите образы приложения и nginx по разделу 5.3. Не изменяйте **APP_KEY** и **INTEGRATIONS_ENCRYPTION_KEY**. При необходимости отдельно загрузите новые образы инфраструктурных служб.
3. Примените миграции кодом нового образа — запуск **exec** в ещё работающем старом контейнере не применяет миграции новой версии. При остановленных обработчиках выполните:

   ```
   docker compose run --rm --no-deps \
     --entrypoint php app artisan migrate --force
   ```

4. Пересоздайте контейнеры из новых образов с той же конфигурацией и постоянными томами. Без антивирусной проверки (**ANTIVIRUS_ENABLED=false**, значение по умолчанию):

   ```
   docker compose up -d --pull never --force-recreate
   ```

   С включённой антивирусной проверкой (**ANTIVIRUS_ENABLED=true**) добавьте профиль **antivirus**, иначе служба **clamav** не пересоздастся вместе с остальными:

   ```
   docker compose --profile antivirus up -d \
     --pull never --force-recreate
   ```

5. Проверьте миграции, фоновые задачи, вход с email-OTP (раздел 6.3, если включён) и доступность прежних данных по разделу 7; при включённой антивирусной проверке — также загрузку файла с антивирусной проверкой. Возобновите пользовательский доступ после успешных проверок. При автоматическом применении миграций точкой входа контейнера повторный запуск не должен менять уже применённую схему.

10.2. Обновление без контейнеризации

В том же окне обслуживания сохраните резервную копию и предыдущую версию. Остановите обработчики очередей и планировщик; обновите код, сохранив **.env** и **storage/**. Установите зависимости Composer и npm, соберите клиентскую часть, примените миграции новым кодом

и пересоберите кэши каркаса (раздел 8.3). Перезапустите PHP-FPM, обработчики и планировщик. Выполните проверки раздела 7 и возобновите доступ.

10.3. Откат

Откат выполняется в окне обслуживания с остановленными обработчиками и планировщиком. Сначала оцените совместимость предыдущего кода с текущей схемой базы данных и последствия для данных, записанных после обновления.

Если схема обратно совместима, используйте сохранённые образы предыдущей версии и прежнюю конфигурацию. Если несовместима — восстановите согласованную резервную копию базы данных и, при необходимости, файлов, либо выполните проверенный обратимый сценарий миграции. Автоматически применять **migrate:rollback --force** без проверки состава миграций и последствий для данных нельзя.

После восстановления проверьте права рабочей роли, защитные триггеры и хеш-цепочку неизменяемого журнала, затем выполните проверки раздела 7. Запись о восстановлении должна содержать использованную версию, момент снятия копии и результат проверки.

Резервная копия перед обновлением

```
docker compose exec -T app php artisan db:backup
```

Команда запускается до замены рабочего экземпляра. Убедитесь, что копия сохранена на постоянном хранилище и дополнительно вне рабочего сервера. Файлы объектного хранилища, конфигурация и ключи не входят автоматически в дамп базы данных и сохраняются отдельно.

Копия PostgreSQL формата **custom** восстанавливается **pg_restore** с ключами **--clean --if-exists** (удаляет прежние объекты базы перед восстановлением):

```
docker compose exec -T app sh -c 'PGPASSWORD="$DB_PASSWORD" pg_restore \
  --host="$DB_HOST" --username="$DB_USERNAME" --dbname="$DB_DATABASE" \
  --clean --if-exists storage/app/backups/ФАЙЛ.dump'
```

Копия в формате **sql.gz** восстанавливается утилитой **psql** после распаковки:

```
gunzip -c ФАЙЛ.sql.gz | docker compose exec -T postgres \
  psql -U "$DB_USERNAME" -d "$DB_DATABASE"
```

Проверки после восстановления — раздел 9 руководства пользователя.

Поддержка при установке

> info@glav.pro; +7 909 539-22-49. Режим работы: понедельник — пятница, с 09:00 до 18:00 по московскому времени (MSK, UTC+3). Электронные обращения принимаются круглосуточно, обрабатываются в указанное рабочее время.

Перед передачей экземпляра на экспертизу разработчик выполняет установку на чистой машине по этой инструкции и фиксирует версию дистрибутива, использованные образы, результат входа, отправки письма, загрузки файла, обновления и восстановления. Проверка текста инструкции не заменяет этот прогон.