# Руководство администратора

## Настройка, управление, устранение проблем

SDLPlatform — решение для безопасной разработки, объединяющее несколько open-source инструментов статического анализа кода (SAST и SCA).

## Содержание

## Введение

SDLPlatform (Secure Development Lifecycle Platform) это программная платформа для организации процесса безопасной разработки на стороне заказчика. Она объединяет инструменты статического анализа безопасности исходного кода (SAST), анализа зависимостей (SCA) и проверки выполнения правил кодирования (Linters), предоставляет веб-интерфейс для управления результатами сканирования, приложениями и уязвимостями, а также помогает формализовать выполнение требований ГОСТ Р 56939-2024.

## Инструкция по установке

- ### Требования к системе

**Минимальные и рекомендуемые системные требования:**

- **Операционная система**: Linux (Ubuntu 24.04.1 LTS или выше)
- **Процессор**: 4 ядра
- **Оперативная память**: 8 GB RAM
- **Свободное место**: 50 GB
- **Сетевое подключение**: Требуется для установки и получения обновлений

- ### Установка Docker и Docker Compose

Убедитесь, что Docker и Docker Compose установлены:

```
sudo apt install -y ca-certificates curl gnupg lsb-release

sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.g
pg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" \
  | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin

systemctl enable --now docker
```

Для выполнения дальнейших шагов необходимо запросить файлы *docker-compose.yaml*, *.env.example* и данные для подключения к репозиторию образов контейнеров у технической поддержки

- **Подключение к репозиторию**

Для подключения к репозиторию с образами контейнеров платформы необходимо выполнить *docker login*:

```
docker login registry.sdlplatform.ru -u <login> -p <personal_token>
```

При отсутствии подключения к DockerHub также необходимо выполнить:

```
docker login -u <login_dockerhub> -p <personal_token_dockerhub>
```

- **Подготовка к развертыванию**

Перед запуском необходимо подготовить файл с переменными окружения из *.env.example*

Обязательные переменные окружения в *.env.example*: - *POSTGRES_PASSWORD* — учетные данные для PostgreSQL - *MINIO_ROOT_USER* и *MINIO_ROOT_PASSWORD* — учетные данные для MinIO (S3-совместимое хранилище) - *SECRET_KEY* — секретный ключ для JWT-токенов - *NOTIFICATIONS_ENABLED* - статус службы уведомлений по E-mail - *TAG* - версия приложения для установки (можно получить у технической поддержки)

Необязательные, но возможные к изменению переменные окружения в *.env.example*: - *SMTP_HOST* - адрес SMTP-сервера для отправки уведомлений - *SMTP_PORT* - порт SMTP-сервера для отправки уведомлений - *SMTP_USER* - пользователь SMTP-сервера для отправки уведомлений - *SMTP_PASSWORD* - пароль от пользователя SMTP-сервера для отправки уведомлений - *SMTP_CONNECTION_TYPE* - тип шифрования при подключении к SMTP-серверу (SSL/TLS) - *APPSCREENER_URL* - URL приложения AppScreener для подключения - *APPSCREENER_TOKEN* - токен для подключения к приложению AppScreener - *PTAI_URL* - URL приложения PT AI для подключения - *PTAI_TOKEN* - токен для подключения к приложению PT AI - *PORT* - порт публикации веб-интерфейса приложения - *POSTGRES_USER* - пользователь БД - *POSTGRES_DB* - наименование БД

Необходимо сохранить *docker-compose.yaml* и *.env* в одну рабочую директорию, из которой планируется управление платформой

- **Запуск платформы**

Для первого запуска системы выполните следующую команду:

> Этот шаг может занять до 15 минут в зависимости от пропускной способности сети и производительности системы, но это только для первого запуска.

```
sudo docker compose up -d
```

Платформа включает следующие сервисы: - *postgres* — база данных PostgreSQL - *minio* — S3-совместимое объектное хранилище - *nats* — брокер сообщений для асинхронной коммуникации - *redis* — кеш и хранилище временных данных - *backend* — FastAPI-приложение с REST API - *frontend* — веб-интерфейс - *nginx* — обратный прокси-сервер - *ml_service* — сервис машинного обучения для анализа уязвимостей - *butler_service* — сервис управления задачами сканирования - *worker_** — воркеры для различных сканеров (semgrep, bandit, gosec, codeql, trivy, и др.)

- **Обновление**

Для обновления необходимо заменить переменную *TAG* в *.env* на новую версию ПО

Скачать обновленные образы контейнеров:

```
docker compose pull
```

Перезапустить платформу:

```
docker compose down
docker compose up -d
```

- ## Установка в Kubernetes

Helm chart и образы размещены в Harbor *registry.sdlplatform.ru*.

**Требования:**

- Kubernetes кластер
- *kubectl*
- Helm 3 с поддержкой OCI (Helm 3.8+)
- Ingress controller: **NGINX**
- Storage: используется **default StorageClass** кластера
- Доступ к *registry.sdlplatform.ru* (учётные данные robot account с правами pull в project *sdlplatform*)

### Подготовка доступа к registry

Для скачивания образов и chart используйте учётные данные robot account, которые вам предоставил поставщик:

- *<name>* — имя robot account (часть после *robot$sdlplatform+*, например *pull-client*)
- *<ROBOT_PASSWORD>* — токен robot account, выданный при создании учётной записи

1) Выполните логин в registry (для Helm OCI):

```
helm registry login registry.sdlplatform.ru -u 'robot$sdlplatform+<name>' -p '
<ROBOT_PASSWORD>'
```

Если токен содержит спецсимволы:

```
printf '%s' '<ROBOT_PASSWORD>' | helm registry login registry.sdlplatform.ru -
u 'robot$sdlplatform+<name>' --password-stdin
```

2) Создайте *imagePullSecret* (чтобы kubelet мог загружать образы):

```
kubectl create namespace sdlplatform || true

kubectl -n sdlplatform create secret docker-registry registry-secret \
  --docker-server=registry.sdlplatform.ru \
  --docker-username='robot$sdlplatform+<name>' \
  --docker-password='<ROBOT_PASSWORD>'
```

3) Если в кластере возникает ошибка *429 Too Many Requests* из-за ограничений DockerHub, добавьте учётные данные DockerHub отдельным secret:

- *<DOCKERHUB_USER>* — логин на hub.docker.com
- *<DOCKERHUB_TOKEN_OR_PASSWORD>* — access token или пароль от Docker Hub

```
kubectl -n sdlplatform create secret docker-registry dockerhub-secret \
  --docker-server=docker.io \
  --docker-username='<DOCKERHUB_USER>' \
  --docker-password='<DOCKERHUB_TOKEN_OR_PASSWORD>'
```

**Установка релиза**

Релизная версия соответствует git-тэгу, например *1.3.0*. Установка и все последующие обновления выполняются через файл *values-prod.yaml*.

Подготовьте *values-prod.yaml*:

```
cat > values-prod.yaml <<'YAML'
global:
  imageRegistry: registry.sdlplatform.ru/sdlplatform/
  imageTag: "1.3.0"
  imagePullSecrets:
    - registry-secret
    - dockerhub-secret

ingress:
  host: sdlplatform.example.com

config:
  data:
```

```
 NOTIFICATIONS_ENABLED: "false"
SMTP_HOST: ""
    SMTP_PORT: ""
    SMTP_USER: ""
    SMTP_CONNECTION_TYPE: ""
    APPSCREENER_URL: ""
    PTAI_URL: ""
    PORT: "8000"

secrets:
  stringData:
    SMTP_PASSWORD: ""
    APPSCREENER_TOKEN: ""
    PTAI_TOKEN: ""
YAML
```

Запустите установку через helm:

```
helm upgrade --install sdlplatform oci://registry.sdlplatform.ru/sdlplatform/s
dlplatform \
  --version 1.3.0 \
  --namespace sdlplatform \
  --create-namespace \
  -f values-prod.yaml
```

Во время установки/обновления backend автоматически применяет миграции БД (Alembic) через initContainer. Первый запуск может занять 1–2 минуты.

**Переменные окружения**

Backend получает переменные из ConfigMap *sdlplatform-config* и Secret *sdlplatform-secrets*.

В ConfigMap задайте:

- *NOTIFICATIONS_ENABLED* (*true/false*)
- *SMTP_HOST*, *SMTP_PORT*, *SMTP_USER*, *SMTP_CONNECTION_TYPE* (*tls/ssl*)
- *APPSCREENER_URL*, *PTAI_URL* (опционально)
- *PORT* (опционально, по умолчанию — 80)

В Secrets задайте:

- *SMTP_PASSWORD*
- *APPSCREENER_TOKEN*, *PTAI_TOKEN* (опционально)

ConfigMap и Secret создаются chart'ом. Для изменения настроек отредактируйте *values-prod.yaml* и выполните *helm upgrade* с этим файлом.

### Обновление релиза

Укажите версию:

```
RELEASE_VERSION=1.3.1

sed -i -E 's/^(  imageTag: ).*$/\1"'"$RELEASE_VERSION"'"/' values-prod.yaml
```

Выполните обновление:

```
helm upgrade --install sdlplatform oci://registry.sdlplatform.ru/sdlplatform/sdlplatform \
  --version "$RELEASE_VERSION" \
  --namespace sdlplatform \
  -f values-prod.yaml
```

### Удаление

```
helm uninstall sdlplatform -n sdlplatform
```

## Управление пользователями

### Добавление пользователя через API

Для создания пользователя через API используйте эндпоинт */api/v1/users/* (требуется токен суперпользователя).

### Добавление пользователя напрямую в БД

Для добавления пользователя напрямую в базу данных PostgreSQL выполните следующие шаги:

1. **Сгенерируйте bcrypt-хеш пароля**:

```
docker compose exec backend python -c "from passlib.context import CryptContex
t; pwd_context = CryptContext(schemes=['bcrypt'], deprecated='auto'); print(pw
d_context.hash('ВАШ_ПАРОЛЬ'))"
```

Хеш должен быть длиной ~60 символов и начинаться с *$2b$12$*. Пример:
*$2b$12$yPhCZhH.wtAFPri2o2xJPOGulZ8U0WXPmKWzVWmebc1xzBbBE7RQm*

2. **Добавьте пользователя в БД**:

```
# Экранируйте символы $ в хеше: замените $ на \$
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
INSERT INTO \"user\" (email, hashed_password, is_active, is_superuser, full_na
me)
VALUES ('user@example.com', '\$2b\$12\$ОСТАЛЬНАЯ_ЧАСТЬ_ХЕША', true, false, 'По
лное Имя');
"
```

⚠️ **Важно:** - Символы *$* в хеше нужно экранировать обратными слешами: *\$* - Bcrypt-хеш *$2b$12$abc...* должен превратиться в *\$2b\$12\$abc...* - Без экранирования bash интерпретирует *$* как переменные, и хеш будет неполным

Для создания суперпользователя установите *is_superuser* в *true*:

```
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
INSERT INTO \"user\" (email, hashed_password, is_active, is_superuser, full_na
me)
VALUES ('admin@example.com', '\$2b\$12\$ОСТАЛЬНАЯ_ЧАСТЬ_ХЕША', true, true, 'Ад
министратор');
"
```

**Полный пример:**

```
# 1. Генерируем хеш
docker compose exec backend python -c "from passlib.context import CryptContex
t; pwd_context = CryptContext(schemes=['bcrypt'], deprecated='auto'); print(pw
```

```
d_context.hash('mypassword123'))"
# Получаем: $2b$12$sze/sfRE5RiG8KEFh42.QOwT8hgyipXVsBqt8qb3paWuDxwJpA4gu

# 2. Вставляем с экранированием $ → \$
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
INSERT INTO \"user\" (email, hashed_password, is_active, is_superuser, full_name)
VALUES ('newuser@example.com', '\$2b\$12\$sze/sfRE5RiG8KEFh42.QOwT8hgyipXVsBqt8qb3paWuDxwJpA4gu', true, false, 'New User');
"
```

**Структура таблицы user:** - *id* — автоинкрементный первичный ключ - *email* — уникальный email пользователя - *hashed_password* — bcrypt-хеш пароля - *is_active* — флаг активности (true/false) - *is_superuser* — флаг суперпользователя (true/false) - *full_name* — полное имя пользователя (опционально)

### Просмотр существующих пользователей

```
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "SELECT id, email, is_active, is_superuser, full_name FROM \"user\";"
```

## Удаление пользователя

Перед удалением пользователя необходимо проверить, есть ли у него связанные данные (приложения, комментарии и т.д.).

**Проверка связанных приложений:**

```
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
SELECT id, name, owner_id FROM application WHERE owner_id = (SELECT id FROM \"user\" WHERE email='user@example.com');
"
```

**Вариант 1: Удаление со всеми связанными данными**

Удаляет пользователя вместе со всеми его приложениями и сканами (используйте с осторожностью!):

```
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
BEGIN;
-- Удаляем приложения пользователя (каскадно удалятся все связанные сканы)
DELETE FROM application WHERE owner_id = (SELECT id FROM \"user\" WHERE email='user@example.com');
-- Удаляем комментарии пользователя
DELETE FROM comments WHERE user_id = (SELECT id FROM \"user\" WHERE email='user@example.com');
-- Удаляем пользователя
DELETE FROM \"user\" WHERE email='user@example.com';
COMMIT;
"
```

**Вариант 2: Переназначение приложений другому пользователю**

Передаёт все приложения пользователя другому владельцу перед удалением:

```
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
BEGIN;

-- Переназначаем приложения на другого пользователя
UPDATE application
SET owner_id = (SELECT id FROM \"user\" WHERE email='new_owner@example.com')
WHERE owner_id = (SELECT id FROM \"user\" WHERE email='user@example.com');
-- Удаляем комментарии пользователя
DELETE FROM comments WHERE user_id = (SELECT id FROM \"user\" WHERE email='use
r@example.com');
-- Удаляем пользователя
DELETE FROM \"user\" WHERE email='user@example.com';
COMMIT;
"
```

**Вариант 3: Деактивация пользователя (рекомендуется)**

Вместо удаления можно деактивировать пользователя, сохранив историю:

```
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
UPDATE \"user\" SET is_active = false WHERE email='user@example.com';
"
```

## Исправление пароля пользователя

Если при входе возникает ошибка *hash could not be identified*, значит пароль был сохранён неправильно (не как bcrypt-хеш).

**Исправление пароля:**

```
# 1. Сгенерируйте новый правильный хеш
docker compose exec backend python -c "from passlib.context import CryptContex
t; pwd_context = CryptContext(schemes=['bcrypt'], deprecated='auto'); print(pw
d_context.hash('НОВЫЙ_ПАРОЛЬ'))"

# 2. Обновите пароль в БД (не забудьте экранировать $ → \$)
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
UPDATE \"user\" SET hashed_password = '\$2b\$12\$ОСТАЛЬНАЯ_ЧАСТЬ_ХЕША' WHERE e
mail='user@example.com';
"
```

***Проверка правильности хешей всех пользователей:***

```
docker compose exec postgres psql -U sdlplatform -d sdlplatform -c "
SELECT id, email, length(hashed_password) as hash_length, substring(hashed_pas
sword, 1, 7) as hash_start
FROM \"user\";
"
```

Правильный bcrypt-хеш должен иметь длину 60 символов и начинаться с *$2b$12$*.

## Интеграция с CI/CD

SDLPlatform поддерживает интеграцию с CI/CD для автоматического запуска анализа безопасности при сборке кода. Это позволяет внедрить анализ кода в каждую сборку и обнаруживать уязвимости до развертывания приложения.

- ### Интеграция с GitLab CI

Для интеграции с GitLab CI необходимо вызвать API SDLPlatform и передать исходный код для сканирования. Пример файла *.gitlab-ci.yml*:

```
stages:
- build
- security

build:
stage: build
script:
  - echo "Building the project..."
  - zip -r /tmp/sources.zip .

security_scan:
stage: security
script:
  - echo "Running security scan with SDLPlatform..."
  - curl -i -X POST -H "Content-Type: multipart/form-data" -F
"file=@/tmp/sources.zip"
https://<SDLPlatform_HOST>/api/v1/applications/<APPLICATION_ID>/scan_upload
  - rm /tmp/sources.zip
allow_failure: true
```

- **stages** — определяет этапы в пайплайне (сборка и безопасность).
- **build** — этап сборки на котором код архивируется для дальнейшеей отправки на анализ.
- **security_scan** — отправка ZIP-архива с исходным кодом на сервер SDLPlatform для сканирования.

- ### Интеграция с Jenkins

Для интеграции с Jenkins можно использовать команду curl, чтобы отправить код на анализ в SDLPlatform:

- Создайте задачу (job) в Jenkins для запуска сканирования.
- В разделе Build Steps добавьте выполнение следующего shell-скрипта:

```
#!/bin/bash
# Архивирование исходного кода
zip -r /tmp/sources.zip .
```

```
# Отправка в SDLPlatform для сканирования
curl -i -X POST -H "Content-Type: multipart/form-data" -F "file=@/tmp/sources.
zip"
https://<SDLPlatform_HOST>/api/v1/applications/<APPLICATION_ID>/scan_upload

# Очистка временных файлов
rm /tmp/sources.zip
```

Этот скрипт архивирует исходный код, отправляет его в SDLPlatform через API, а затем удаляет временные файлы.

- **Интеграция с другими CI/CD системами**

Для других систем, таких как CircleCI, Travis CI, или TeamCity, вы можете использовать аналогичные команды для архивирования исходного кода и вызова API SDLPlatform. Пример для CircleCI:

```
version: 2.1
jobs:
build:
docker: - image: circleci/node:latest
steps: - checkout - run:
name: Archive source code
command: zip -r /tmp/sources.zip . - run:
name: Send to SDLPlatform
command: |
curl -i -X POST -H "Content-Type: multipart/form-data" \
-F "file=@/tmp/sources.zip" \
https://<SDLPlatform_HOST>/api/v1/applications/<APPLICATION_ID>/scan_upload -
run:
name: Clean up
command: rm /tmp/sources.zip
```

- **Получение результатов сканирования**

После выполнения сканирования результаты доступны в интерфейсе приложения, а также через API:

```
# Прогресс сканирования приложения
curl -H "Authorization: Bearer <TOKEN>" \

https://<SDLPlatform_HOST>/api/v1/applications/<APPLICATION_ID>/scan_progress
# Перечень уязвимостей (в контексте приложения и общий список)
curl -H "Authorization: Bearer <TOKEN>" \

"https://<SDLPlatform_HOST>/api/v1/applications/<APPLICATION_ID>/vulnerabiliti
```

```
es?type=sast&skip=0&limit=50"

curl -H "Authorization: Bearer <TOKEN>" \

"https://<SDLPlatform_HOST>/api/v1/applications/<APPLICATION_ID>/vulnerabiliti
es?type=sca&skip=0&limit=50"
curl -H "Authorization: Bearer <TOKEN>" \
  "https://<SDLPlatform_HOST>/api/v1/vulnerability/?type=sast&skip=0&limit=50"
curl -H "Authorization: Bearer <TOKEN>" \
  "https://<SDLPlatform_HOST>/api/v1/vulnerability/?type=sca&skip=0&limit=50"
```

## Резервное копирование и восстановление данных

- ### Резервное копирование базы данных

  **Ручное резервное копирование** базы данных PostgreSQL из контейнера Docker с помощью *pg_dump*:

Чтобы выполнить резервное копирование базы данных, работающей в контейнере Docker, выполните следующую команду:

```
docker compose exec -T postgres \
  pg_dump -U sdlplatform sdlplatform > SDLPlatform_backup.sql
```

- *postgres* – имя сервиса PostgreSQL в docker-compose.yml
- *sdlplatform* – имя пользователя и базы данных (по умолчанию совпадают)
- *SDLPlatform_backup.sql* – путь к файлу резервной копии на хосте
- **Автоматическое резервное копирование**: Настройте Cron для регулярного выполнения резервного копирования базы данных. Пример для выполнения резервного копирования каждый день в 2:00:

```
0 2 * * * cd /path/to/monorepo && docker compose exec -T postgres pg_dump -U s
dlplatform sdlplatform > /backups/SDLPlatform_db_$(date +\%F).sql
```

- ### Резервное копирование файлов конфигурации

Кроме базы данных, важно регулярно сохранять конфигурационные файлы системы. Эти файлы включают:

- Файлы конфигурации Docker (*docker-compose.yml*, *docker-compose.override.yml*)
- Переменные окружения (*.env*)
- Файлы конфигурации Nginx (*nginx.conf*, *nginx-dev.conf*)
- SSL-сертификаты (*keys/ssl/*)
- Файлы аутентификации (*keys/.htpasswd*)

Пример команды для резервного копирования конфигурационных файлов:

```
tar -czvf /backups/config_backup_$(date +\%F).tar.gz /path/to/configs/
```

- **Восстановление базы данных**

Чтобы восстановить базу данных из резервной копии, выполните команду *psql* внутри контейнера Docker:

```
docker compose exec -T postgres psql -U sdlplatform -d sdlplatform < SDLPlatform_backup.sql
```

Эта команда восстановит данные из резервной копии базы данных *sdlplatform*.

- **Восстановление файлов конфигурации**

Для восстановления файлов конфигурации выполните следующую команду:

```
tar -xzvf /path/to/backup/config_backup.tar.gz -C /path/to/configs/
```

- **Дополнительные рекомендации**
  - **Периодичность резервного копирования**: Определите частоту резервного копирования в зависимости от объема данных и критичности системы. Рекомендуется выполнять ежедневные резервные копии базы данных и еженедельные полные резервные копии системы.
  - **Внешние хранилища**: Храните резервные копии на внешних носителях или в облаке для защиты от сбоев на сервере (например, AWS S3, Google Cloud Storage).
  - **Шифрование резервных копий**: Для безопасности рекомендуется шифровать резервные копии с помощью таких инструментов, как gpg:

```
gpg --encrypt --recipient your_email@example.com /path/to/backup.sql
```

## Мониторинг и диагностика

- **Журналирование**

Настройте систему журналирования для отслеживания событий в системе и выявления проблем.

Например, используйте docker logs для журналирования событий из контейнеров:

```
docker compose logs -f
```

Для просмотра логов конкретного сервиса:

```
docker compose logs -f backend
docker compose logs -f ml_service
docker compose logs -f worker_semgrep
```

- **Системы мониторинга**

Интеграция с системами мониторинга, такими как Prometheus или Grafana, поможет отслеживать состояние системы в реальном времени, нагрузку на CPU, память и другие параметры.

Пример конфигурации для Prometheus:

```
scrape_configs:
- job_name: 'docker'
  static_configs:
    - targets: ['localhost:8080']
```

- **Диагностика ошибок**

Проверяйте журналы ошибок Nginx и других компонентов системы.

## Обновления и патчи

- **Обновление приложения SDLPlatform**
  - **Обновление кода приложения**:

Для обновления SDLPlatform необходимо загрузить последние изменения из репозитория:

```
git pull origin main
```

*Это обновит код приложения до последней версии.*

  - **Пересборка Docker-контейнеров** с учетом новых изменений:

```
sudo docker compose up -d --build
```

Эта команда пересоберет контейнеры с учетом обновленного кода и перезапустит приложение.

Для пересборки конкретного сервиса:

```
sudo docker compose up -d --build backend
sudo docker compose up -d --build ml_service
```

- **Установка патчей безопасности**

Патчи безопасности необходимы для предотвращения уязвимостей в системе. Обычно они могут включать обновления операционной системы, баз данных и серверного ПО.

  - **Обновление операционной системы**:

На сервере, где запущен Docker, регулярно устанавливайте обновления безопасности операционной системы (например, на Ubuntu):

```
sudo apt update && sudo apt upgrade -y
```

Это обеспечит актуальность пакетов и установку критических патчей безопасности.

  - **Автоматическое обновление безопасности**:

Включите автоматическое обновление безопасности для критических пакетов:

```
sudo apt install unattended-upgrades
sudo dpkg-reconfigure --priority=low unattended-upgrades
```

- ## Проверка состояния после обновлений

После применения любых обновлений выполните следующие шаги для проверки работоспособности системы:

- **Проверьте статус контейнеров**:

```
docker compose ps
```

Убедитесь, что все контейнеры запущены и работают корректно.

- **Проверьте логи системы**: Используйте следующую команду для просмотра логов контейнеров и поиска возможных ошибок:

```
docker compose logs -f
```

- **Тестирование функциональности**: Пройдите по основным функциям системы, чтобы убедиться, что обновления не вызвали проблем в работе.

- ## Откат обновлений

Если после обновления возникли проблемы, можно откатить изменения:

- **Откатить обновленные образы Docker**:

Если обновление контейнера привело к проблемам, можно вернуться к предыдущей версии:

```
docker compose down
docker pull <previous-image-tag>
docker compose up -d
```

- **Откат к предыдущей версии кода**:

Верните предыдущую версию кода приложения:

```
git checkout <previous-commit-hash>
```

- **Восстановление из резервной копии**:

Если откат образов или кода не помогает, выполните восстановление базы данных и файлов конфигурации из резервной копии (см. раздел **Восстановление базы данных** и **Восстановление файлов конфигурации**).

## Безопасность

- ### Настройка брандмауэра

**Рекомендации по настройке брандмауэра**: Откройте только необходимые порты для работы платформы, например, порты для Docker, Nginx, и SSH. Все другие порты должны быть закрыты.

Пример команды для настройки правил брандмауэра на основе UFW (Uncomplicated Firewall):

```
sudo ufw allow OpenSSH
sudo ufw allow 443/tcp
sudo ufw enable
```

- ### SSL-сертификаты

**Обновление SSL-сертификатов**:

Регулярно обновляйте SSL-сертификаты для поддержания HTTPS-соединения.

- ### Политики паролей

**Создание и внедрение сложных политик паролей**:

Установите минимальные требования к длине пароля (например, 12 символов), обязательные символы разных типов (заглавные и строчные буквы, цифры и спецсимволы).

- ### Аудит безопасности

**Регулярный аудит безопасности**:

Используйте инструменты для анализа безопасности системы, такие как Lynis или OpenSCAP, для выявления уязвимостей и несоответствий конфигурации.

Пример выполнения аудита с помощью Lynis:

```
sudo lynis audit system
```

## Часто задаваемые вопросы

Проблемы с запуском

Убедитесь, что установлены все зависимости и Docker настроен корректно.
Если система не запускается, проверьте логи контейнеров (`docker compose logs`).

Как улучшить скорость сканирования?

Получите API-ключ для OWASP Dependency-Check и укажите его в переменной окружения — это значительно ускорит загрузку базы уязвимостей.

## Техническая поддержка

В случае возникновения проблем свяжитесь с технической поддержкой: support@sdlplatform.ru

КИБЕРЩИТ (CYBER SHIELD)

ОГРН 1197746511930
ИНН 7734428610
РФ, Московская область



Отдел продаж
sales@sdlplatform.ru
+7 (934) 055-88-00
+7 (495) 790-66-88 (доб. 1)

Поддержка
support@sdlplatform.ru
Пн – Пт / 10:00 – 18:00
+7 (495) 790-66-88 (доб. 3)