#Full Documentation

#Функциональные характеристики

#Общие функциональные характеристики

Модульная платформа безопасной разработки **CodeScoring** обеспечивает решение задач безопасного использования **open source**, идентификации секретов в коде и проверки качества разработки.

Платформа включает в себя следующие модули:

- **CodeScoring.Save** – управление артефактами разработки с поддержкой proxy- и hosted-репозиториев, интеграцией проверок безопасности и контролем доступа;
- **CodeScoring.OSA (Open Source Analysis)** – защита цепочки поставки: проверка сторонних компонентов на соответствие политикам безопасности и блокировка при несоответствии;
- **CodeScoring.SCA (Software Composition Analysis)** – композиционный анализ: автообнаружение **open source** зависимостей, выявление уязвимостей и вредоносных компонентов, проверка совместимости лицензий, формирование перечня программных компонентов и построение графа зависимостей;
- **CodeScoring.Secrets** – поиск секретов в коде: оркестрация проверок исходного кода на наличие конфиденциальной информации (**секретов**) и оценка истинности с помощью собственной модели машинного обучения;
- **CodeScoring.TQI (Teams & Quality Intelligence)** – оценка качества кода через расчет ключевых метрик качества, отслеживание динамики разработки и построение профилей участников разработки с подтвержденной компетенцией в проектах.

Комплексно платформа обеспечивает:

- анализ 20 языков программирования (для отдельных модулей список языков может варьироваться): **Java**, **Kotlin**, **JavaScript**, **TypeScript**, **Python**, **C**, **C++**, **Go**, **PHP**, **Ruby**, **C#**, **Objective-C**, **Swift**, **Rust**, **Scala**, **Bash**, **Perl**, **SQL**, **PLSQL**, **PGSQL**;
- анализ манифестов пакетных экосистем: **Maven**, **Gradle**, **Apache Ivy**, **NPM**, **Yarn**, **pnpm**, **NuGet**, **Paket**, **pip**, **Poetry**, **Pipenv**, **Conda**, **Ruby Gems**, **Cocoapods**, **Swift**, **Composer**, **Conan**, **Go Modules**, **Cargo** и **sbt**;
- интеграцию с менеджерами репозиториев и реестрами контейнерных образов: **Nexus Repository Manager**, **JFrog Artifactory Pro**, **Harbor**, **GitLab**, **GitFlic**, **Сфера.Дистрибутивы и Лицензии**;

- менеджер репозиториев **CodeScoring.Save** с поддержкой proxy- и hosted-режимов для **Maven**, **npm**, **NuGet**, **PyPI**, **Go**, **Docker/OCI** и **raw**;
- интеграцию с **hosted** и **cloud**-системами хостинга кода: **GitFlic**, **GitLab**, **Github**, **Bitbucket** и **Azure DevOps**;
- универсальную интеграцию в системы сборки при помощи агента: **Jenkins**, **GitLab**, **GitFlic**, **TeamCity** и прочие;
- управление политиками безопасности организации;
- автоматизацию запусков сканирования по расписанию;
- интеграцию для почтовых уведомлений;
- интеграцию с системой управления задачами **Jira**;
- систему управления пользователями;
- интеграцию **LDAP**;
- создание вебхуков;
- разграничение прав доступа для разных пользователей;
- журналирование всех выполняемых операций (**аудит-лог**);
- сбор метрик по проведенным операциям;
- открытый и задокументированный программный интерфейс (**API**).

#Функциональные характеристики модуля CodeScoring.Save

- централизованное хранилище артефактов с поддержкой двух типов репозиториев:
  - **proxy-репозитории** — проксирование запросов к upstream-репозиториям с кэшированием артефактов и выполнением проверок безопасности;
  - **hosted-репозитории** — полностью локальные репозитории для хранения собственных артефактов организации;
- поддержка популярных пакетных экосистем:
  - **Maven** (Java, Kotlin, Scala и другие JVM-инструменты поверх Maven layout);
  - **npm** (JavaScript / TypeScript);
  - **NuGet** (.NET);
  - **PyPI** (Python);
  - **Go** (Go Modules);
  - **Docker / OCI** (контейнерные образы, Helm OCI-чарты, oras-артефакты);
  - **RAW** (произвольные файлы);
- микросервисная архитектура с разделением компонентов:
  - **Save API service** — основной сервис работы с артефактами и репозиториями;
  - **Auth / RBAC service** — выделенный микросервис аутентификации и авторизации;
- интеграция проверок безопасности:
  - автоматическое сканирование артефактов через **CodeScoring.OSA**;
  - применение политик безопасности при загрузке компонентов;
  - блокировка загрузки артефактов, не соответствующих политикам;

- управление доступом и безопасность:
  - ролевая модель доступа (**RBAC**) с тремя уровнями (**global / project / repository**);
  - поддержка пользователей (**users**) и сервисных аккаунтов (**robot accounts**);
  - управление членством в проектах с гранулярным контролем прав;
  - выпуск и валидация **JWT-токенов** для API с локальной проверкой подписи на стороне Save через JWKS;
  - поддержка шести типов учётных данных: Basic Auth (пользовательский и сервисный robot), Bearer JWT, opaque Bearer (npm), `X-NuGet-ApiKey` и Docker v2 token;
  - поддержка **OCI Distribution Spec / Docker Registry v2** с эндпоинтом выпуска короткоживущих access-токенов;
  - аудит операций, связанных с доступом и управлением артефактами, в едином журнале платформы;
- организационная структура:
  - иерархия **проект → репозиторий → артефакт**;
  - управление проектами с назначением участников;
  - настройка прав доступа на уровне проектов и репозиториев;
- управление жизненным циклом артефактов:
  - **cleanup policies** — правила для автоматического удаления устаревших или неиспользуемых артефактов;
  - настройка правил очистки по возрасту, количеству версий, по последнему скачиванию, по регулярным выражениям, по размеру;
  - dry-run / preview результата перед запуском;
- веб-интерфейс для управления:
  - создание и настройка проектов и репозиториев;
  - просмотр и скачивание артефактов и их метаданных;
  - управление пользователями, ролями и правами доступа;
  - настройка правил очистки;
  - управление лицензией;
  - просмотр сконфигурированных параметров;
  - мониторинг операций и экспорт журнала событий;
- варианты развертывания:
  - **hosted-инсталляция** на базе **k3s** для небольших команд;
  - **базовая инсталляция** на **Kubernetes** с **PostgreSQL** и **S3-совместимым хранилищем** для продуктивного использования;
  - конфигурация хранения: **SQLite/PostgreSQL** для метаданных, **S3/файловая система** для артефактов;
- производительность и масштабируемость:
  - stateless-архитектура для горизонтального масштабирования;
  - in-pod кэш и локальная валидация Bearer JWT по JWKS;
  - стриминговая обработка больших файлов без полной буферизации в памяти;
  - независимое горизонтальное масштабирование Save и cs-auth;
- мониторинг и наблюдаемость:
  - экспорт метрик в **Prometheus** в формате OpenTelemetry;
  - структурированное логирование операций в JSON;
  - health checks для контроля состояния сервисов;

    - метрики HTTP-запросов, операций хранилища, загрузок/скачиваний артефактов и proxy cache hit-rate;
- открытый программный интерфейс (**API**):
    - REST API для управления проектами, репозиториями и артефактами;
    - совместимость с протоколами пакетных менеджеров (Maven 2 layout, npm Registry API, NuGet v3, PyPI Simple PEP 503/691, Go Module Proxy, OCI Distribution Spec);
        - API для управления пользователями, ролями, robot-аккаунтами и правами доступа;
        - внутренние API для межсервисного взаимодействия между Save и cs-auth, защищённые shared secret.

#Функциональные характеристики модуля CodeScoring.OSA

- блокировка загрузки нежелательных компонентов при попытке их скачивания через командный интерфейс пакетного менеджера или веб-интерфейс;
- формирование перечня программных компонентов (**SBOM**, Software Bill of Materials);
- проверка артефактов в реестрах контейнерных образов;
- проксирование и контроль загрузок пакетов через сервис OSA Proxy для популярных пакетных менеджеров (Maven, NPM, PyPI, NuGet);
- анализ архивов, включая:
    - разбор содержимого популярных архивных форматов (**zip**, **jar**, **tar**, **war**, **tgz** и другие);
    - анализ системных пакетов, включая поддержку популярных менеджеров пакетов:
        - **DEB** (Debian, Ubuntu);
        - **RPM** (RHEL, CentOS, Fedora);
        - **APK** (Alpine Linux);
- настройка критериев безопасности (**политик**) по 40 критериям, включая:
    - метаданные пакетов: название, версия, автор пакета, возраст, дата;
    - критерии уязвимостей: идентификатор, оценка **CVSS**, уровень угрозы, дата публикации, возраст, наличие эксплойта, импакт;
    - категории и типы лицензий, совместимость лицензий (**лицензионная чистота**);
    - признаки применения пакета: директивный/транзитивный и окружение (**scope**);
    - наличие вредоносного кода;
    - черные списки пакетов, включая собственный фид **protestware**;
- определение и дедупликация уязвимостей из 20 баз знаний, включая агрегационные (**БДУ ФСТЭК**, **NVD**, **OSV**, **GHSA** и иные), экосистемные (**Debian**, **RPM**, **Alpine** и иные), коммерческие (**Kaspersky OSS Threats Data Feed**) и собственные, включая данные о **protestware**;

- управление компонентами в интерфейсе, включая:
  - просмотр и фильтрацию списка пакетов;
  - просмотр и фильтрацию списка контейнерных образов;
  - просмотр и фильтрацию запросов компонентов;
  - выгрузку отчетности.

#Функциональные характеристики модуля CodeScoring.SCA

- анализ манифестов пакетных менеджеров;
- формирование перечня программных компонентов (**SBOM, Software Bill of Materials**) с учетом рекомендаций **ФСТЭК России**;
- проверка **open source** на всех этапах цикла разработки с возможностью задания политик безопасности для отдельных этапов цикла разработки:
  - проверка кода в средах разработки (**IDE**): **IntelliJ-based**, **VSCode**, **OpenIDE**;
  - проверка на локальной машине разработчика (**агент CLI**);
  - непрерывный мониторинг кода (сканирование веток репозиториев);
  - проверка в **CI-конвейере** универсальным агентом с возможностью блокирования сборки;
  - пост-релизный мониторинг **SBOM**;
  - пост-релизный мониторинг кода (сканирование тегов репозиториев);
- обнаружение **open source** зависимостей:
  - по экосистеме, названию и версии пакета;
  - по файлам конфигураций (**манифестам**) пакетных менеджеров;
  - через механизм разрешения транзитивных зависимостей;
  - через механизм идентификации **open source**-включений по вендорской базе мета-данных, с применением алгоритмов гибкого поиска;
  - через анализ сборки для языков **C** и **C++**;
- разделение директивных и транзитивных зависимостей и определение отношения к окружениям (**scope**) разработки: **runtime**, **compile**, **test**, **provided** и другим;
- построение графа зависимостей с возможностью трассирования использования транзитивных компонентов;
- актуализация информации об обнаруженных зависимостях:
  - общая информация о зависимости: способ обнаружения; дата релиза пакета; автор пакета; официальная веб-страница; ссылка на размещение в пакетном индексе; окружение применения (**scope**) и ссылка на граф связей; ссылка на манифест;
  - информация об уязвимостях;
  - информация о лицензиях;
  - максимальная версия исправления;
  - используемость в проектах организации;
- предоставление информации об уязвимостях:
  - идентификаторы в базах знаний;

    - описание уязвимости;
    - оценка критичности по **CVSS 2**, **CVSS 3.1** и **CVSS 4.0**;
    - ссылки на дополнительные индексы уязвимостей;
    - рекомендуемая версия пакета для обновления;
    - перечень всех проектов, задетых уязвимостью;
    - дополнительные материалы, содержащие ссылки на патчи, эксплоиты и уязвимые коммиты;
- предоставление информации о лицензиях:
    - лицензионный ландшафт проекта;
    - **SPDX**-идентификатор лицензии;
    - текст лицензии;
    - краткое представление об условиях лицензии;
    - информация о совместимости лицензий (**license compliance**);
- определение и дедупликация уязвимостей из более 20 баз знаний, включая агрегационные (**БДУ ФСТЭК**, **NVD**, **OSV**, **GHSA** и иные), экосистемные (**Debian**, **RPM**, **Alpine** и иные), коммерческие (**Kaspersky OSS Threats Data Feed**) и собственные, включая данные о **protestware**;
- настройка политик безопасности по 40 критериям, включая:
    - метаданные пакетов: название, версия, автор пакета, возраст, дата;
    - критерии уязвимостей: идентификатор, оценка **CVSS**, уровень угрозы, дата публикации, возраст, наличие эксплойта, импакт;
    - категории и типы лицензий, совместимость лицензий (**лицензионная чистота**);
    - признаки применения пакета: директивный/транзитивный и окружение (**scope**);
    - наличие вредоносного кода;
    - черные списки пакетов, включая подготовленный фид **protestware**;
- настройка политик безопасности отдельно для сканирования кода в репозиториях, **CI-конвейере** и локальной машине разработчика с указанием этапа цикла разработки;
- система отображения результатов сканирований для каждого отдельного этапа и возможность управления игнорируемыми событиями;
- возможность настройки защиты от популярных атак на цепочку поставки;
- возможность настройки временного игнорирования политик срабатывания по различным критериям: проекту, технологии, пакету, типу лицензии, идентификатору уязвимости, для отдельных политик;
- выгрузка отчетности в популярных форматах: **CycloneDX**, **SPDX**, **JUnit**, **SARIF**, **CSV**, **GitLab Dependency Scanning Report**, **GitLab Code Quality Report**;
- анализ достижимости уязвимостей.

#Функциональные характеристики модуля CodeScoring.Secrets

- обнаружение чувствительных данных в коде, включая пароли, **API-ключи**, токены, учетные данные;
- использование собственной модели машинного обучения для фильтрации ложных срабатываний;
- интеграция с системами контроля версий (**VCS**) для автоматического мониторинга репозиториев;
- настройка критериев поиска секретов с возможностью конфигурирования правил обнаружения;
- запуск анализа секретов:
  - выбор сканирования отдельной ветки или всего репозитория;
  - ручной запуск анализа в отдельных проектах;
  - общее сканирование всех проектов организации;
  - загрузка отчетов о секретах, найденных сторонними инструментами, в систему;
- предоставление детализированной информации о найденных секретах:
  - контекст обнаружения (проект, файл, автор изменения);
  - оценка вероятности истинности секрета;
  - дата обнаружения и исправления секрета;
- управление найденными секретами:
  - просмотр списка обнаруженных секретов;
  - подтверждение ложных срабатываний и исключение их из дальнейших сканирований;
  - маркировка секретов как устраненных;
- управление процессом обучения **ML-модели** для повышения точности обнаружения:
  - разметка обнаруженных секретов пользователем;
  - дообучение модели на размеченных данных;
  - сравнение результатов дообученной модели с исходной версией;
  - возможность отката к базовой модели;
- интеграция с **DevSecOps**-процессами:
  - работа в **CI/CD-конвейере** с возможностью прерывания сборки при обнаружении секретов;
  - анализ секретов на локальной машине разработчика через **CLI-агент**;
- система отчетности и уведомлений:
  - отображение результатов сканирования на уровне организации, проекта и отдельного репозитория;
  - фильтрация результатов;
  - экспорт отчетов со списком секретов или их свойств.

#Функциональные характеристики модуля CodeScoring.TQI

- построение профиля проектов, детализация технического и авторского состава разработки;

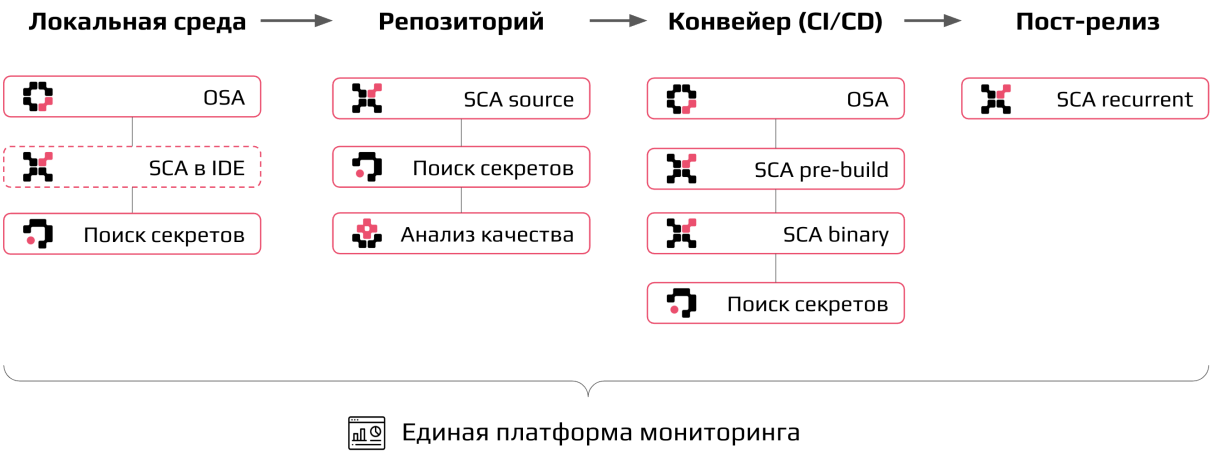
- построение профиля разработчиков, детализация их работы в анализируемых проектах с учетом объема работ и качества произведенных изменений;
- автоматическая оценка сходства разработчиков в анализируемых проектах по опыту участия и технологиях;
- поиск активности разработчиков в **open source** проектах;
- анализ заимствований с применением алгоритмов нечеткого поиска (толерантность к переименованиям объектов в коде):
    - поиск внутри-проектных дубликатов кода;
    - поиск кросс-проектных дубликатов кода;
    - определение объема найденных заимствований;
    - построение интерактивной карты заимствований;
    - определение времени возникновения заимствования;
    - привязка авторства к найденным дубликатам кода;
- оценка сложности сопровождения проекта:
    - расчет цикломатической сложности кода в разрезе проекта;
    - расчет цикломатической сложности кода в разрезе автора;
- построение аналитических ретроспективных карт:
    - развития проектов;
    - работы авторов;
    - эволюции сложности проектов;
- встраивание в цикл разработки программного обеспечения (**SDLC**) посредством интеграции с системами версионирования исходного кода.

#Этапы интеграции

Платформа безопасной разработки **CodeScoring** интегрируется в жизненный цикл разработки программного обеспечения, обеспечивая контроль безопасности, соответствие политикам и анализ качества на каждом ключевом этапе:

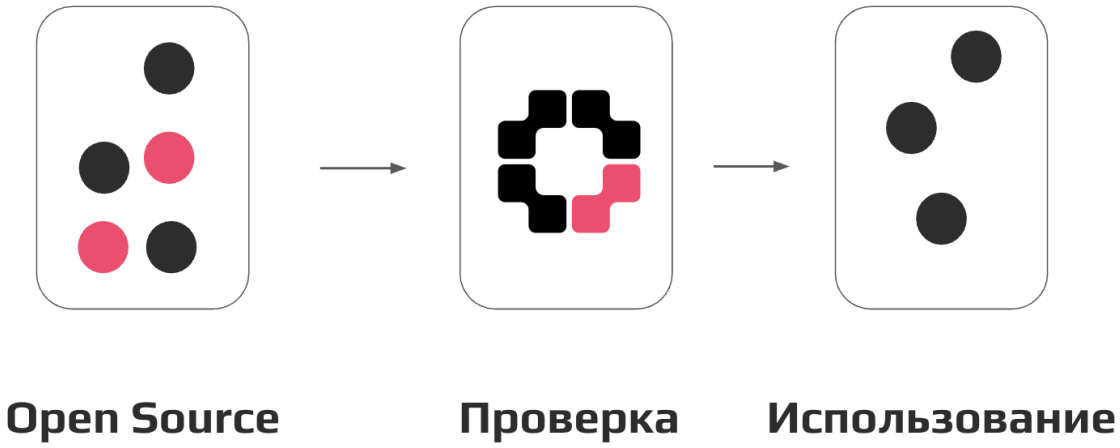
- локальная разработка;
- хранение исходного кода;
- сборка программных продуктов;
- сопровождение после релиза.

Общая схема интеграции представлена ниже:



Важно: перечислена основная функциональность платформы по этапам. Полный перечень возможностей доступен на странице функциональных характеристик.

#Локальная разработка



На этапе локальной разработки CodeScoring помогает предотвратить попадание уязвимых или вредоносных компонентов в кодовую базу, а также провести композиционный анализ и поиск секретов с помощью универсального агента Johnny.

Функциональность:

- анализ и блокировка сторонних компонентов в момент загрузки из прокси-репозиториев;
- инвентаризация и регистрация запросов компонентов;
- композиционный анализ;
- поиск конфиденциальной информации в исходном коде.

Для осуществления блокировки компонентов предоставляются плагины CodeScoring.OSA для **Nexus Repository Manager** и **JFrog Artifactory PRO**.

#Хранение исходного кода

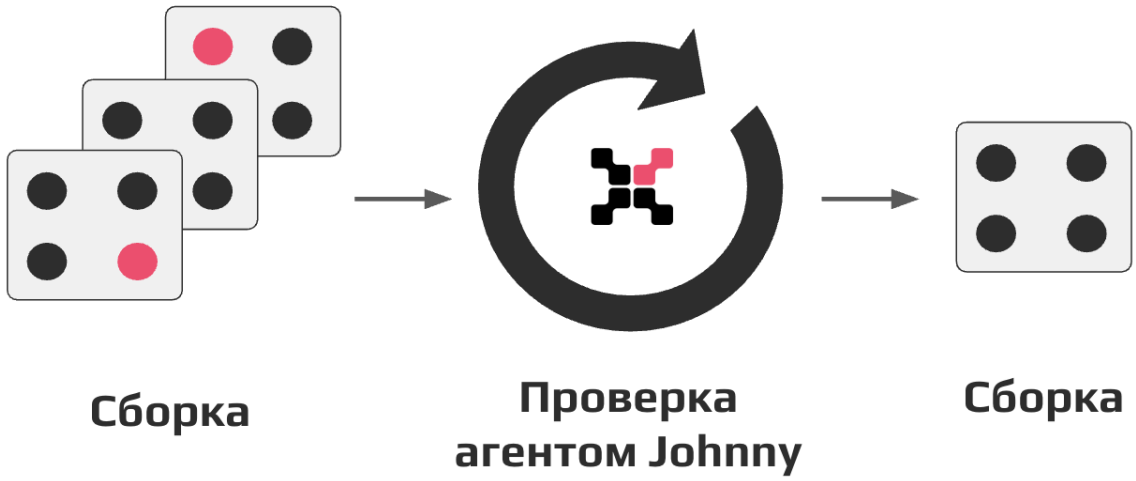
На этапе хранения и управления исходным кодом CodeScoring позволяет обеспечить непрерывный контроль качества и безопасности репозиториев.

Поддерживается интеграция с основными платформами разработки, использующие git: **GitFlic**, **GitHub**, **GitLab**, **Bitbucket**, **Azure DevOps** и др.

Функциональность:

- инвентаризация сторонних компонентов в репозиториях;
- обнаружение уязвимостей и потенциально опасных компонентов;
- поиск секретов;
- анализ качества разработки.

#Сборка программных продуктов



На этапе сборки CodeScoring обеспечивает анализ программного обеспечения в конвейере CI/CD и выполняет проверку используемых артефактов.

Поддерживаются инструменты автоматизации: **GitLab CI/CD**, **Jenkins**, **Bitbucket Pipelines**, **TeamCity**, **Bamboo**, **GitFlic** и др.

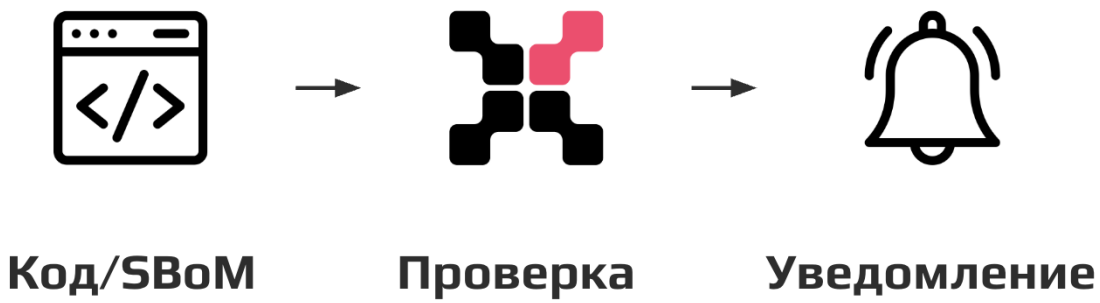
Функциональность:

- автоматическое формирование перечня программных компонентов (SBOM);

- обнаружение уязвимостей и потенциально опасных компонентов;
- анализ лицензионной совместимости;
- контроль соответствия сборки политикам безопасности.

Анализ выполняется с помощью агента Johnny, доступного как бинарный файл или контейнерный образ. При нарушении политик безопасности агент завершает выполнение с соответствующим кодом ошибки, что позволяет остановить сборку до попадания небезопасного артефакта в релиз.

#Сопровождение после релиза



После публикации продукта CodeScoring обеспечивает непрерывный мониторинг безопасности исходного кода и состава компонентов. Это позволяет своевременно реагировать на появление новых уязвимостей и угроз в уже выпущенных версиях.

Функциональность:

- периодическое сканирование исходного кода и компонентов (SBOM);
- автоматическое обновление данных об угрозах;
- отправка уведомлений через Email и менеджеры задач;
- интеграция с **ASPM/SIEM**-системами.

#Поддерживаемые экосистемы и способы анализа

#Манифесты

Для поиска зависимостей CodeScoring в первую очередь опирается на разбор файлов манифестов пакетных менеджеров. Платформа поддерживает разбор следующих технологий:

Экосистема	Пакетный менеджер или инструмент сборки	Формат файла

Java и Kotlin	Gradle	`*.gradle`
		`*.gradle.kts`
		`gradle-dependency-tree.tx`
		`gradle.lockfile`

	Maven	`pom.xml` `maven-dependency-tree.txt`
	Apache Ivy	`ivy.xml`

JavaScript и TypeScript	npm	`package.json`
		`package-lock.json`
		`npm-shrinkwrap.json`

	yarn	`yarn.lock`
		`package.json`
		`package-lock.json`

	pnpm	`pnpm-lock.yaml`
	bun	`bun.lock`

Python	pip	`requirements.txt`
		`requirements.pip`
		`requires.txt`
		`pip-resolved-dependencies`
		`pipdeptree.txt`

	Poetry	`pyproject.toml` `poetry.lock`
	Pipenv	`Pipfile` `Pipfile.lock`

	Conda	`environment.yml`
		`meta.yml`
		`conda-lock.yml`

	uv	`pyproject.toml` `uv.lock`

C и C++	Conan	`conanfile.txt`
		`conan.lock`
		`conanfile.py`

	GCC, Clang, CMake, Make и др.	Сканирование сборки

Экосистема	Пакетный менеджер или инструмент сборки	Формат файла
Go	Go Modules	`go.mod` `go.sum`
PHP	Composer	`composer.json` `composer.lock`

Ruby	RubyGems	`Gemfile`
		`Gemfile.lock`
		`*.gemspec`
		`gems.locked`
		`gems.rb`

.NET	Nuget	`*.nuspec`
		`packages.lock.json`
		`Project.json`
		`Project.lock.json`
		`packages.config`
		`*.csproj`
		`project.assets.json`
		`dependencyReport.json`
		`deps.json`
		`*.sln`

	Paket	`paket.dependencies` `paket.lock`

Objective-C	CocoaPods	`Podfile`
		`Podfile.lock`
		`*.podspec`

Swift	Swift Package Manager	`Package.swift` `Package.resolved`
Rust	Cargo	`Cargo.toml` `Cargo.lock`
Scala	sbt	`scala-dependency-tree.txt` `sbt-dependency-tree.txt`

Лучший результат будет при наличии основного файла манифеста и соответствующего lock-файла, если он предусмотрен механизмом пакетного менеджера.

#Типы PURL и компонентов

Для унифицированного описания зависимостей CodeScoring использует стандарт **Package URL (PURL)**.

Пример PURL

```
pkg:maven/org.apache.logging.log4j/log4j-core@2.17.2
```

При анализе SBOM через команду агента или импорте в платформу CodeScoring распознаёт и поддерживает следующие типы PURL в соответствии со спецификацией:

Тип PURL	**Описание**	**Спецификация**
`cocoapods`	Библиотеки для **Objective-C** / **Swift** через CocoaPods	CocoaPods Definition
`conan`	Пакеты экосистемы **C** / **C++ (Conan)**	Conan Definition
`conda`	Пакеты экосистемы **Python** / **Conda**	Conda Definition
`nuget`	Компоненты **.NET** / **NuGet**	NuGet Definition
`golang`	Пакеты **Go Modules**	Go Definition
`maven`	Артефакты **Java** / **Kotlin** (Maven / Gradle)	Maven Definition
`npm`	Пакеты **JavaScript** / **TypeScript**	NPM Definition
`composer`	Пакеты **PHP (Composer)**	Composer Definition
`pypi`	Пакеты **Python (PyPI)**	PyPI Definition
`gem`	Пакеты **Ruby (RubyGems)**	RubyGems Definition
`cargo`	Пакеты **Rust (Cargo)**	Cargo Definition
`generic`	Общий тип для произвольных бинарных или кастомных артефактов	Generic Definition
`apk`	Системные пакеты **Alpine Linux**	APK Definition
`deb`	Системные пакеты **Debian** / **Ubuntu**	DEB Definition
`rpm`		RPM Definition

Тип PURL	Описание	Спецификация
	Системные пакеты **RHEL / CentOS / Fedora**	
`swift`	Пакеты **Swift Package Manager**	Swift Definition
`oci`	Контейнерные образы **OCI / Docker**	OCI Definition
`docker`	Образы **Docker Hub / Docker**	Docker Definition
`github`	Репозитории **GitHub**	GitHub Definition
`huggingface`	Модели **Hugging Face Hub**	HuggingFace Definition
`mlflow`	Модели **MLflow Model Registry**	MLflow Definition
`pub`	Пакеты **Dart / Flutter (pub.dev)**	Pub Definition
`swid`	**SWID-теги (Software Identification Tags)**	SWID Definition

Каждый компонент с PURL классифицируется по типу, который CodeScoring распознаёт при импорте SBOM-файлов. Тип указывается в поле `type` внутри описания компонента.

Различие типов PURL и компонентов

Тип компонента описывает его функциональную роль внутри продукта — например, библиотека, фреймворк или встроенное ПО. Тип PURL, в свою очередь, определяет экосистему и источник, из которого этот компонент был получен.

Пример компонента в SBOM

```
{
  "components": [
    {
      "name": "log4j-core",
      "version": "2.17.2",
      "purl": "pkg:maven/org.apache.logging.log4j/log4j-
core@2.17.2",
      "type": "library"
    }
  ]
}
```

Поддерживаются следующие типы компонентов:

Тип компонента	Описание
library	Библиотека, пакет или модуль стороннего кода, используемый в проекте.
framework	Инфраструктурный или прикладной фреймворк, включающий набор библиотек.
firmware	Исполняемый бинарный образ или встроенное ПО, анализируемое на наличие сторонних компонентов.

#Системные пакеты

В рамках работы модуля OSA платформа поддерживает разбор системных пакетов следующих форматов:

- Debian;
- Alpine;
- RPM;
- Astra Linux;
- ALT Linux;
- РЕД ОС.

#Сканирование архивов

Плагины CodeScoring.OSA поддерживают сканирование архивов в следующих форматах:

Экосистема	Формат архива
Maven	`.jar`, `.war`, `.ear`
NPM	`.tgz`
PyPI	`.zip`, `.tar`, `.tgz`, `.tar.gz`, `.tar.bz2`, `.egg`, `.whl`
Nuget	`.nupkg`
Cocoapods	`.tar.gz`, `.zip`
Go	`.mod`, `.zip`
Gems	`.rz`, `.gz`, `.gem`

Экосистема	Формат архива
Debian	`.deb`, `.xz`, `.gz`
Yum	`.rpm`
Alpine	`.apk`
Docker	`.json`

Сканирование архивов через Johnny

Консольный агент Johnny также поддерживает сканирование архивов. Поддерживаемые форматы приведены на странице Сканирование архивов.

#Механизм резолва при отсутствии lock-файла

При отсутствии lock-файла для некоторых пакетных индексов система выполняет разрешение транзитивных OSS зависимостей следующим образом:

- Maven
  - для формата pom.xml и build.gradle генерация maven-dependency-tree через соответствующий плагин maven
  - используются Maven версии 3.8.8 и OpenJDK версии 11
- PyPi
  - генерация poetry.lock с помощью пакетного менеджера Poetry
  - используется Python версии 3.11.7
- NPM
  - генерация yarn.lock с помощью пакетного менеджера Yarn
  - используется Node.js версии 20.9.0
- Nuget
  - для формата csproj и sln генерация project.assets.json с помощью встроенных инструментов nuget
  - используется .NET SDK версии 8.0.404
- Packagist
  - генерация composer.lock с помощью пакетного менеджера Composer
  - используется PHP версии 8.2.26
- Rubygems
  - генерация Gemfile.lock с помощью пакетного менеджера Bundler
  - используется Ruby версии 3.1.2p20

Самостоятельная генерация lock-файлов системой не может давать результат в 100% случаев, так как результат часто зависит от окружения.

#Разрешение зависимостей в окружении

Пакетные менеджеры некоторых экосистем по умолчанию не включают транзитивные зависимости в манифесты. Для качественного проведения композиционного анализа при работе с ними рекомендуется применять механизм разрешения зависимостей в окружении сборки.

При разрешении зависимостей в окружении система проверяет отсутствие lock-файла, самостоятельно запускает пакетный менеджер или инструмент сборки и формирует полный список компонентов с учетом корректной версии сборки. На данный момент функциональность доступна для следующих экосистем:

- .NET
- Go
- Gradle
- Maven
- npm
- Poetry
- sbt
- yarn
- Conda

#Механизм поиска зависимостей по хэшам

Поиск по хэшам подразумевает определение непосредственного включения библиотек в код проектов путём копирования. В рамках этого механизма происходит хэширование всех файлов проекта и сверка этих сигнатур с известными нам open source библиотеками.

В данный момент поиск по хэшам происходит для следующих индексов пакетных менеджеров по следующим типам файлов:

- Maven
  - `.jar`
  - `.war`
  - `.ear`
- npm
  - `.min.js`
- PyPI
  - `.whl`
  - `.egg`
- Nuget
  - `.nupkg`

От платформы в облако **не уходят** хэши файлов, размер которых не превышает 512 байт.

#Сканирование сборки для языков C и C++

В случае, если для сборки C/C++ проекта не используется пакетный менеджер Conan и соответствующие манифесты, для получения списка используемых библиотек можно использовать специальный режим для анализа вывода процесса сборки.

В данном режиме консольный агент Johnny анализирует процесс сборки, используя флаги компилятора и выявляя использованные библиотеки. Далее с помощью системного кэша определяется местоположение библиотек и их источник.

#Соответствие стандартам

Платформа **CodeScoring** помогает организациям соответствовать стандартам безопасной разработки и требованиям регуляторов.

Система покрывает ключевые практики безопасной разработки, включая контроль качества исходного кода, контроль безопасности стороннего ПО, поиск уязвимостей, поиск конфиденциальной информации (секретов) и реагирование на инциденты безопасности в цепочке поставок.

#ГОСТ Р 56939–2024

ГОСТ Р 56939–2024 *«Разработка безопасного программного обеспечения. Общие требования»* регламентирует этапы, требования и методы безопасной разработки программного обеспечения, включая архитектурные практики, анализ кода и обработку зависимостей.

CodeScoring обеспечивает выполнение 8 пунктов стандарта из 25:

Требование	Модуль реализации	Комментарий
5.9 Экспертиза исходного кода	CodeScoring.TQI	Обеспечивается автоматизированная экспертиза качества кода с использованием метрик (цикломатическая сложность, дублирование, динамика изменений). Поддерживается системная оценка качества и сопровождения кода в процессе разработки.
5.12 Использование безопасной системы сборки ПО	CodeScoring.OSA, CodeScoring.SCA, CodeScoring.Save	Анализируются зависимости, применяемые при сборке, выявляются небезопасные компоненты. Контроль состава ПО выполняется до

Требование	Модуль реализации	Комментарий
5.13 Обеспечение безопасности сборочной среды ПО		запуска сборки, что позволяет снизить риск привнесения уязвимостей и ошибок со стороны системы и среды сборки в соответствии с целями 5.12.1.1 и 5.13.1.1.
5.15 Безопасность используемых секретов	CodeScoring.Secrets	Выполняется поиск ключей, паролей, токенов и других секретов в исходном коде, истории коммитов и конфигурационных файлах. Применяются методы машинного обучения для снижения числа ложноположительных срабатываний.
5.16 Композиционный анализ	CodeScoring.SCA	Выполняется анализ всех зависимостей, включая транзитивные, с определением их источников, лицензий и известных уязвимостей. Формируется и актуализируется перечень зависимостей (ППК, SBOM), что обеспечивает контроль состава, выявление уязвимостей и применение корректирующих мер в цепочке поставок.
5.17 Проверка на внедрение вредоносного ПО через цепочку поставок	CodeScoring.OSA, CodeScoring.SCA, CodeScoring.Save	Обеспечивается контроль сторонних библиотек и компонентов, используемых в проекте. Реализуется: • *5.17.2.4* – выявление и контроль предсобранного ПО поставщика; • *5.17.2.5* – анализ зависимостей на предмет внедрения вредоносного кода (за исключением антивирусной проверки, которая относится к дополнительным средствам защиты).
5.23 Реагирование на информацию об уязвимостях	CodeScoring.OSA, CodeScoring.SCA, CodeScoring.Secrets	Обеспечивается регулярное обновление базы уязвимостей и автоматическое оповещение о новых рисках.

Требование	Модуль реализации	Комментарий
		Поддерживается интеграция с корпоративными системами (почта, менеджер задач), что позволяет инициировать своевременное реагирование в рамках политики безопасности организации.
5.24 Поиск уязвимостей при эксплуатации	CodeScoring.SCA	Реализована возможность регулярной перепроверки выпущенных сборок для рекуррентного поиска уязвимостей. Выполняются оповещения о новых уязвимостях в рамках анализа исходного кода, контейнерных образов и SBOM-файлов на этапе эксплуатации.

Более подробно ознакомиться с функциональностью платформы можно в разделе Функциональные характеристики.

#Требования к установке

#Операционная система

Установка on-premise версии возможна на **GNU/Linux** дистрибутивы (включая отечественные: AstraLinux, AltLinux, RedOS и иные).

#Ресурсы сервера

#Базовые требования для рабочей среды

- Минимально допустимые требования для серверов приложения: **32Gb RAM, 16 ядер CPU**.

Рекомендация по ресурсам

Для установки в рабочей среде не рекомендуется опускаться ниже указанных значений, так как это может приводить к деградации производительности и нестабильной работе платформы.

Рекомендация по CPU

Используйте CPU, совместимый с семейством Intel Scalable Gen 2. Для работы CodeScoring требуются инструкции CPU, которые могут отсутствовать в старых CPU.

#Требования к сервисам данных

- **PostgreSQL**: от **32Gb RAM**;
- **Redis**: от **2Gb RAM**;
- Размер `shm` для **PostgreSQL**: не менее **4Gb**;
- При использовании внешней базы данных для **PostgreSQL** рекомендуется от **64Gb RAM**.

#Требования к хранилищу

- Для модуля **CodeScoring.SCA** при использовании VCS-проектов объем тома `analysis-root` рассчитывается из размера анализируемых репозиториев, умноженного на три;
- Для модуля **CodeScoring.SCA** при использовании CLI-проектов и для модуля **CodeScoring.OSA** отдельные требования по формуле ×3 не предъявляются;
- При оффлайн-установке для базы данных CodeScoring Index необходимо выделять от **300Gb** с запасом под последующие обновления;

#Пример высоконагруженной инсталляции

- Сервер базы данных: **96Gb RAM, 24 ядер CPU**;
- Сервер приложений: **192Gb RAM, 48 ядер CPU**.

#Поддерживаемые версии внешних сервисов

При использовании собственных экземпляров баз данных убедитесь, что их версии соответствуют требованиям ниже:

#Redis

- Минимальная версия: **7.0.0**;
- Протестированная версия: **7.4.9**.

#PostgreSQL

- Минимальная версия: **15.x** (любая минорная);
- Протестированная версия: **15.15**.

Использование других мажорных веток не гарантирует корректный результат и может приводить к ошибкам или снижению производительности.

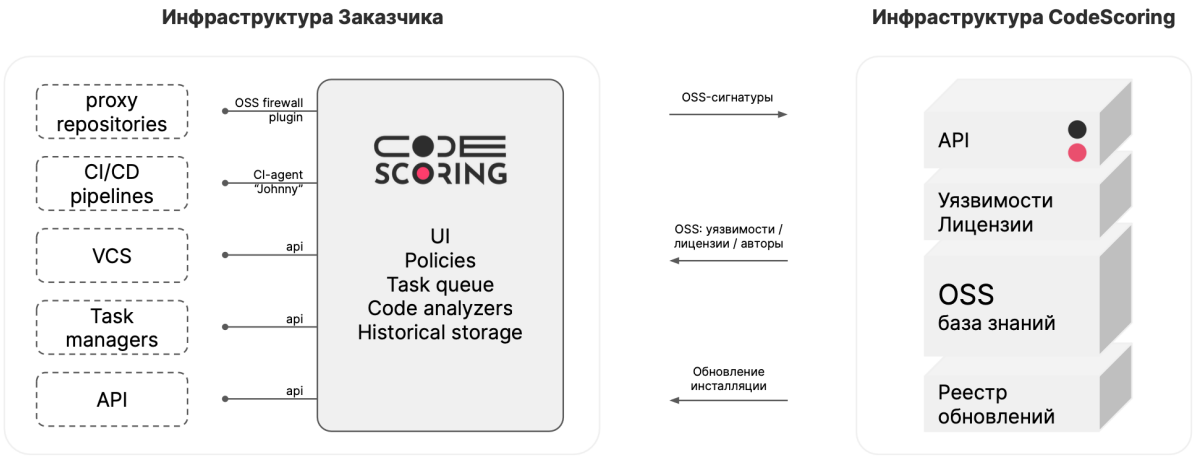
#Внешние запросы

Для установки системы должен быть доступен Docker Registry с образами CodeScoring, адрес которого предоставляется вместе с ключом активации.

Для корректной работы с сервера также должен быть доступен адрес `index.codescoring.ru` с постоянно актуализируемой базой известных пакетов, их атрибутов и уязвимостей.

Из Index API платформа получает обогащенную информацию по найденным зависимостям, их лицензиям и уязвимостями.

Общая архитектура работы представлена на изображении ниже.



Из платформы в облако CodeScoring не отправляется исходный код, но для получения информации по зависимостям и лицензионного контроля отправляются:

1. анонимизированные данные по найденным манифестам пакетных менеджеров и их содержимому;
2. хэши файлов исходного кода для поиска прямых включений Open Source библиотек в код проектов;
3. количество активных авторов за последний год;
4. количество проектов в системе;
5. версия платформы.

Пути манифестов и названия хэшируемых файлов специально анонимизируются. От платформы в облако **не уходят** хэши файлов, размер которых не превышает 512 байт.

Пример содержимого запроса от платформы к Index API с данными по манифестам пакетных менеджеров:

```
[
  {
    "path": "114bc73d-a9ba-433d-9a3e-f2b29d822204",
```

```
    "type": "file",
    "extension": ".txt",
    "result": {
      "platform": "pypi",
      "dependencies": [
        {
          "name": "django",
          "requirement": "==3.0.0",
          "resolved_requirement": "3.0.0",
          "env": "dev"
        }
      ],
      "kind": "manifest",
      "success": true,
      "extra": {}
    }
  },
  {
    "path": "efde2364-dc0c-45a9-905a-a487b3361ac7",
    "type": "file",
    "extension": ".xml",
    "result": {
      "platform": "maven",
      "dependencies": [
        {
          "name": "org.liquibase:liquibase-core",
          "requirement": "3.6.2",
          "resolved_requirement": "3.6.2",
          "env": "compile"
        },
        {
          "name": "xpp3:xpp3",
          "requirement": "1.1.4c",
          "resolved_requirement": "1.1.4c",
          "env": "compile"
        }
      ],
      "kind": "manifest",
      "success": true,
      "extra": {}
    }
  },
  {
    "path": "49dd4c09-b5de-474a-998a-3ce0a94a5221",
    "type": "file",
    "extension": ".txt",
    "result": {
      "platform": "pypi",
      "dependencies": [
        {
          "name": "apt-wrapper",
          "requirement": "==1.18",
```

```
          "resolved_requirement": "1.18",
          "env": "runtime"
        },
        {
          "name": "django",
          "requirement": "==2.0.0",
          "resolved_requirement": "2.0.0",
          "env": "runtime"
        },
        {
          "name": "text-unidecode",
          "requirement": "==1.3",
          "resolved_requirement": "1.3",
          "env": "runtime"
        }
      ],
      "kind": "manifest",
      "success": true,
      "extra": {}
    }
  }
]
```


Пример содержимого запроса от платформы к Index API с данными по хэшам файлов исходного кода:


```
[
  {
    "id": "ca028ad9-0676-4c85-a5b0-9bf81fba6fcc",
    "ext": ".xml",
    "sha256":
"e01c736a351633932e8b3ed041e553f67968e07d35d2c153b02b60e910a8c4
33"
  }
]
```


#Установка системы

1. Установить Docker Engine под нужную операционную систему в соответствии с документацией: https://docs.docker.com/engine/install/.

2. Авторизоваться в приватном реестре Docker-образов системы «CodeScoring» при помощи команды `docker login REGISTRY_URL`, используя адрес, логин и пароль, полученные от вендора.

3. Скачать полученный от вендора архив с установочными файлами, распаковать.

4. Перейти в консоли в созданную директорию.
5. Скопировать шаблонный файл с настройками:

   ```
   cp app.env.template app.env
   ```

   Как правило, для корректной работы никаких изменений в файле не требуется. При необходимости настройки работы **CodeScoring** через прокси, обратите внимание на инструкцию.

6. Скопировать шаблонный файл с секретами:

   ```
   cp .env.template .env
   ```

   Настроить параметры конфигурации в новом файле.

   - Список доменов для правильной работы CSRF защиты. Рекомендуется перечислить localhost на внутреннем и внешнем портах, а также внешний домен (или сочетание ip:порт). Если не менять параметры в файле, то по умолчанию система будет доступна по адресу `http://localhost:8081`. Указание протокола является обязательным, например:
     - `DJANGO_CSRF_TRUSTED_ORIGINS=http://localhost:18000,https://localhost:8081,https://внешний ip:8081`
   - Параметры подключения к базе данных PostgreSQL. База поставляется вместе с установкой. Указание доступов отдельно является мерой предосторожности и контроля. При использовании собственной базы данных необходимо убедиться, что она соответствует требованиям.
     - `POSTGRES_DB` — название базы данных
     - `POSTGRES_USER` — имя пользователя. При использовании собственной базы необходимо убедиться, что пользователь имеет следующие права: **Superuser**, **Create role**, **Create DB**, **Replication**, **Bypass RLS**.
     - `POSTGRES_PASSWORD` — пароль
     - `POSTGRES_HOST` - хост, на котором доступна база данных
     - `POSTGRES_PORT` - порт, на котором доступна база данных
   - Секрет платформы
     - `SECRET_KEY` — случайная строка символов
   - Настройки домена системы
     - `NGINX_HOST` — хост, на котором будет доступна система
     - `NGINX_PORT` — порт, на котором будет доступна система
     - `SITE_SCHEME` - протокол передачи данных, по умолчанию https
   - Пути исключений
     - `ANALYSIS_IGNORED_PATHS` - список путей, которые будут игнорироваться системой при анализе. Подробнее с добавлением путей исключения можно ознакомиться тут

- Версия системы
  - `CODESCORING_VERSION` – обязательная переменная. Актуальную версию можно узнать в разделе Changelog
- Настройки Docker Compose
  - `COMPOSE_PROJECT_NAME` - название проекта в Compose, используется как префикс для ресурсов, создаваемых Docker Compose

Символ #

Не используйте в параметрах символ #, он может некорректно восприниматься системой при установке.

SSL-сертификаты

В случае необходимости работы системы с самоподписанными сертификатами перед запуском прочтите инструкцию по добавлению сертификата.

7. Выполнить команду установки CodeScoring (выполнение команды должно быть с правами суперпользователя системы):

```
docker compose -f ./docker-compose.yml up -d --force-recreate --remove-orphans --renew-anon-volumes
```

8. Для просмотра логов можно использовать команду:

```
docker compose logs -f
```

9. После запуска сервис будет доступен по настроенному домену или адресу `http://localhost:8081`. При первом запуске дополнительно выполняются миграции базы данных, операция может занять больше времени, чем при последующих запусках.

   Примечание: для работы платформы по протоколу HTTPS нужен внешний балансировщик, который реализует терминирование SSL.

10. Для входа в систему необходимо предварительно создать пользователя с правами администратора с помощью следующей команды:

```
docker compose exec -it backend python ./manage.py createsuperuser
```

11. Для изменения пароля администратора можно использовать следующую команду:

```
docker compose exec -it backend python ./manage.py changepassword <user_name>
```

#Работа платформы CodeScoring в Docker Compose со внешней СУБД

1. В случае, если необходимо использовать схему, отличную от `public`, необходимо явно задать `search_path` для пользователя, включив в него целевую схему, чтобы обеспечить корректное разрешение объектов:

```
ALTER USER codescoring_user_name SET search_path =
non_default_schema_name;
```

1. Необходимо сконфигурировать соответствующие параметры в файле `.env`:
    - `POSTGRES_DB`
    - `POSTGRES_USER`
    - `POSTGRES_PASSWORD`
    - `POSTGRES_HOST`
    - `POSTGRES_PORT`
2. Для администрирования платформы необходимо применять файл `external-db.override.yml`, который поставляется вместе с файлом `docker-compose.yml`:
    - Запуск платформы:

      ```
      docker compose -f ./docker-compose.yml -f external-
      db.override.yml up -d --force-recreate --remove-
      orphans --renew-anon-volumes
      ```

    - Просмотр логов:

      ```
      docker compose -f ./docker-compose.yml -f external-
      db.override.yml logs -f
      ```

    - Остановка платформы:

      ```
        docker compose -f ./docker-compose.yml -f external-
      db.override.yml down --remove-orphans
      ```

#Работа системы в Kubernetes

#Установка с помощью Helm-чарта

Порядок установки:

1. Создать namespace.

   ```
   kubectl create namespace codescoring
   ```

2. Создать secret для доступа к приватному реестру Docker-образов системы "CodeScoring", используя адрес (`REGISTRY_URL`), логин (`USERNAME`) и пароль (`PASSWORD`), полученные от вендора.

   ```
   kubectl create secret docker-registry codescoring-regcred
   --docker-server=REGISTRY_URL --docker-username=USERNAME --
   docker-password=PASSWORD -n codescoring
   ```

   Альтернативно, можно создать секрет для доступа к реестру с помощью `values.yaml`, добавив в поле .Values.imagePullSecrets запись вида

   ```
   registry.org: |
     {"auths":{"registry.org":{"auth":"<base64>"}}}
   ```

3. Установить Helm предпочтительным способом.

4. Выполнить следующие команды для добавления актуального Helm-репозитория на локальную машину:

   ```
   helm repo add codescoring-org https://{REGISTRY_URL}/
   repository/helm/ --username USERNAME --password PASSWORD
   helm repo update
   ```

#Настройки параметров Helm-чарта

Важно

Настоятельно рекомендуется вносить необходимые изменения **до установки CodeScoring**, в противном случае может потребоваться полная переустановка системы. Данные инструкции предполагают, что **специалист имеет опыт работы с кластером Kubernetes и утилитой Helm**.

Tip

Все представленные в данном документе ограничения вычислительных ресурсов являются приблизительными. Реальное потребление ресурсов

компонентами зависит от нагруженности и конкретных сценариев использования инсталляции.

Для удобного редактирования параметров CodeScoring можно скачать и распаковать исходный код Helm-чарта командой:

```
helm pull codescoring-org/codescoring-helm --version
CHART_VERSION --untar --untardir codescoring-src && cd
codescoring-src
```

В файле `values.yaml` можно отредактировать нужные переменные, и после этого, находясь в каталоге с исходным кодом Helm-чарта, выполнить команду установки:

```
helm install codescoring . -f values.yaml -n codescoring --
atomic --version CHART_VERSION
```

Структура чарта предполагает, что абсолютно вся конфигурация, вплоть до описания Kubernetes-ресурсов, описывается в `values.yaml`, из-за чего данный файл является довольно объемным и его редактирование может быть не столь удобно. Вполне допустимо и даже рекомендуется, при необходимости в переопределении каких-либо параметров, создать дополнительный файл с произвольным именем, например, `values-override.yaml` (здесь и далее этот файл будет называться именно так), в котором и переопределить значения необходимых полей. Для удобства с чартом поставляется шаблон файла `values-override.yaml`.

В таком случае команда установки будет выглядеть так (порядок указания файлов имеет значение):

```
helm install codescoring . -f values.yaml -f values-
override.yaml -n codescoring --atomic --version CHART_VERSION
```

#Установка с использованием встроенных PostgreSQL и Redis

Чарт предусматривает возможность развертывания PostgreSQL и Redis. Для этого предусмотрены ресурсы StatefulSet. Чтобы активировать установку PostgreSQL и Redis, необходимо определить следующие параметры в `values-override.yaml`:

```
statefulSets:
  codescoring-postgresql:
    enabled: true
    containers:
      postgresql:
        resources:
          limits:
            cpu: 2000m
```

```
            memory: 12Gi
          requests:
            cpu: 1m
            memory: 500Mi

  codescoring-redis:
    enabled: true
    containers:
      redis:
        resources:
          limits:
            cpu: 2000m
            memory: 4Gi

pvcs:
  codescoring-postgresql:
    accessModes:
      - ReadWriteOnce
    size: 20Gi
    storageClassName: "default"
  codescoring-redis:
    accessModes:
      - ReadWriteOnce
    size: 1Gi
    storageClassName: "default"
```

При использовании встроенных баз данных дополнительная конфигурация параметров подключения не требуется.

#Подключение к внешним базам данных

Важно

При использовании собственной базы данных необходимо убедиться, что она соответствует требованиям.

#Подключение к внешнему Redis

Для подключения к внешнему Redis, необходимо указать соответствующие строки подключения в следующих полях, соблюдая соответствие номеров баз данных:

```
configMaps:
  ipcs-backend-env:
    data:
      DJANGO_CACHES_REDIS_URL: "redis://codescoring-
redis:6379/1"
      HUEY_REDIS_URL: "redis://codescoring-redis:6379/0"
```

```
    CELERY_BROKER_URL: "redis://codescoring-redis:6379/2"
    CELERY_RESULT_BACKEND: "redis://codescoring-redis:6379/2"
```

#Подключение к PostgreSQL через пулер Pgbouncer

Важно

Подключение к внешней PostgreSQL необходимо выполнять с использованием пулера соединений.

Данный вариант подходит, если в существующей инфраструктуре уже развернута PostgreSQL, но пулер соединений не используется. Helm-чарт развернет пулер Pgbouncer и подключит его к существующей PostgreSQL. Необходимо выполнить следующие действия:

1. Сконфигурировать пулер соединений, указав соответствующие данные в следующих полях:

```
secrets:
  pgbouncer-secrets:
    data:
      # секреты для подключения pgbouncer к PostgreSQL
      POSTGRES_DB: "codescoring_db"
      POSTGRES_USER: "codescoring_user"
      POSTGRES_PASSWORD: "changeme"
      # секреты для внутренних пулов pgbouncer
      TRANSACTION_POOL_PASSWORD: "changeme"
      SESSION_POOL_PASSWORD: "changeme"
      STATS_USER: "codescoring_user-stats"
      STATS_PASSWORD: "changeme"
      ADMIN_USER: "codescoring_user-admin"
      ADMIN_PASSWORD: "changeme"

configMaps:
  pgbouncer-env:
    data:
      # параметры подключения pgbouncer к PostgreSQL
      POSTGRES_HOST: "codescoring-postgresql"
      POSTGRES_PORT: "5432"
      # параметры для внутренних пулов pgbouncer
      TRANSACTION_POOL_USER: "codescoring_user"
      SESSION_POOL_USER: "codescoring_user-session"
      TRANSACTION_POOL_SIZE: "50"
      TRANSACTION_POOL_MIN_SIZE: "1"
      SESSION_POOL_SIZE: "50"
      SESSION_POOL_MIN_SIZE: "1"
      # виртуальные базы данных pgbouncer, существуют
только внутри него
      TRANSACTION_POOL_DATABASE_NAME:
"codescoring_generic"
      SESSION_POOL_DATABASE_NAME: "codescoring_generic-
```

```
session"
      MAX_CLIENT_CONN: "500"
```

2. Передать созданные в пункте 1 конфигурационные параметры компонентам CodeScoring:

```
secrets:
  ipcs-secrets:
    data:
      # секреты для подключения к пулам из pgbouncer-
secrets
      TRANSACTION_POOL_PASSWORD: "changeme"
      SESSION_POOL_PASSWORD: "changeme"
      DATABASE_PASSWORD: "changeme" # соответствует
TRANSACTION_POOL_PASSWORD

configMaps:
  ipcs-backend-env:
    data:
      # параметры подключения osa-api и judge
      DATABASE_HOST: "pgbouncer"
      DATABASE_PORT: "6432"
      DATABASE_USERNAME: "codescoring_user" #
соответствует TRANSACTION_POOL_USER
      DATABASE_NAME: "codescoring_db" # соответствует
TRANSACTION_POOL_DATABASE_NAME

      TRANSACTION_POOL_HOST: "pgbouncer"
      TRANSACTION_POOL_PORT: "6432"
      TRANSACTION_POOL_USER: "codescoring_user"
      TRANSACTION_POOL_DATABASE_NAME: "codescoring_db"

      SESSION_POOL_HOST: "pgbouncer"
      SESSION_POOL_PORT: "6432"
      SESSION_POOL_USER: "codescoring_user-session"
      SESSION_POOL_DATABASE_NAME: "codescoring_generic-
session"
```

#Настройка томов

#Dynamic Volume Provisioning с использованием требуемого StorageClass

Чарт создает необходимые тома через Dynamic Volume Provisioning с использованием указанного явно `StorageClass`.

Для создания ресурсов PersistentVolumeClaim необходимо заполнить следующую секцию values:

```
pvcs:
  # storageClassName необходимо заменить на используемый в
```

```
кластере
  codescoring-ipcs-django-static:
    accessModes:
      - ReadWriteOnce
    size: 10Gi
    storageClassName: "default"
  codescoring-ipcs-analysis-root:
    accessModes:
      - ReadWriteOnce
    size: 10Gi
    storageClassName: "default"
  codescoring-ipcs-media-root:
    accessModes:
      - ReadWriteOnce
    size: 10Gi
    storageClassName: "default"
```

Если кластер поддерживает тома с типом доступа ReadWriteMany (RWX), то рекомендуется использовать именно его, так как в этом случае допустимо размещение компонентов инсталляции на разных нодах кластера. Если же поддержки RWX в кластере нет, либо есть необходимость в использовании RWO по иным причинам, рекомендуется настроить podAffinity, чтобы поды, использующие одни и те же тома были назначены на одну ноду: Для этого необходимо добавить следующий блок в `values-override.yaml`:

```
deploymentsGeneral:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: codescoring-component
                operator: In
                values:
                  - backend
                  - frontend
                  - huey
                  - celery
          topologyKey: kubernetes.io/hostname
```

#PersistentVolumeClaim для заранее созданных PersistentVolume

Названия предварительно созданных PersistentVolume можно указать в поле `volumeName`, например:

```
pvcs:
  codescoring-ipcs-analysis-root:
    accessModes:
      - ReadWriteOnce
```

```
    size: 10Gi
    storageClassName: "default"
    volumeName: "codescoring-precreated-pv"
```

#Настройка ограничения ресурсов (resource limits)

По умолчанию `requests` и `limits` заданы в демонстрационных объемах, не предназначенных для продуктивного использования. Это сделано для обеспечения возможности запуска системы CodeScoring в кластерах с малым количеством ресурсов (например, minikube) с целью тестирования. При запуске в **production-окружении** может потребоваться настроить ограничение ресурсов. Это можно сделать, отредактировав следующие поля:

#Ограничение ресурсов для init-контейнеров

```
## ограничения, применяемые ко всем init-контейнерам
## редактирование данного поля возможно только в values.yaml,
override невозможен из-за ограничений yaml-структур
init-container-resources: &init-container-resources
  resources:
    limits:
      cpu: 1000m
      memory: 1Gi
    requests:
      cpu: 1m
      memory: 100Mi

## отдельно настраиваются ограничения init-контейнеров
компонента backend
deployments:
  ipcs-backend:
    initContainers:
      ipcs-collectstatic:
        resources:
          limits:
            cpu: 1000m
            memory: 2Gi
          requests:
            cpu: 1m
            memory: 500Mi
      ipcs-migrate:
        resources:
          limits:
            cpu: 1000m
            memory: 2Gi
          requests:
            cpu: 1m
            memory: 500Mi
```

#Настройка ограничений ресурсов для основных контейнеров компонентов

В общем виде ресурсы конфигурируются по следующему шаблону:

```
deployments:
  <deployment-name>:
    containers:
      <container-name>:
        resources:
          limits:
            cpu: 8000m
            memory: 12Gi
          requests:
            cpu: 1m
            memory: 500Mi
```

Возможно указание как `requests` и `limits` вместе, так и по отдельности. Однако, при указании только `limits`, Kubernetes автоматически выставит `requests` в таком же объеме, что может негативно сказаться на назначении подов на ноды.

#Добавление сертификата удостоверяющего центра (CA)

Для доступа CodeScoring к ресурсам с TLS-сертификатами, подписанными корпоративным удостоверяющим центром (CA) необходимо добавить корневой сертификат удостоверяющего центра (RootCA) в `values.yaml` в формате `ключ: значение`, где ключ - имя файла сертификата, включая расширение `.crt`, значение - сертификат в формате PEM, в следующее поле:

```
secrets:
  ca-certificates:
    data:
      my-root-CA.crt: |-
        -----BEGIN CERTIFICATE-----

MIIDTDCCAjSgAwIBAgIBATANBgkqhkiG9w0BAQUFADA3MQswCQYDVQQGEwJERT
EP

MA0GA1UEChMGZWR1UEtJMRcwFQYDVQQDEw5lZHVQS0kgVGVzdCBDQTAeFw0xMD
Az

MzExMjIwMjRaFw0zMDAzMjYxMjIwMjRaMDcxCzAJBgNVBAYTAkRFMQ8wDQYDVQ
QK

EwZlZHVQS0kxFzAVBgNVBAMTDmVkdVBLSSBUZXN0IENBMIIBIjANBgkqhkiG9w0
B
        AQEFAAOCAQ8AMIIBCgKCAQEAt5IxCk/
NQPOLqeA1lGuB3pvqHGQPxRQ1udYGcXQY
```

```
t7EuSMFymUR9m5TsifG1ktktJTtOWyaWFC4ac0vai49wGVeuDYptfZBoHLIUvC
wN

DOofLYHxk04WzfrtSiUTptn1o6QPOw8YR0XH30MEi1zgD8fLMZmVTJ+XwA5Eus6
c
        XtTmI4XhNrHUtvWt4UsNgLmp5/
djUgRMpNqxIdrpFQzl+XycRJRAaoAwUzHFl14t

49qwBhGChxQ8AdDMQGA7kv6VR8o0ktCPv3a4GQbs8+z0cX0w5dC+XhJ1xpqW6T
Og

qAY9XBFIDe5j21hjKmNZ39rsODVGUS2wUtNEhSz+3YqxLwIDAQABo2MwYTAdBg
NV

HQ4EFgQUqHe3saMjZZLan8RlFJs+Xuz4yiAwHwYDVR0jBBgwFoAUqHe3saMjZZ
La
        n8RlFJs+Xuz4yiAwDwYDVR0TAQH/BAUwAwEB/
zAOBgNVHQ8BAf8EBAMCAQYwDQYJ

KoZIhvcNAQEFBQADggEBAEjQGyHZQis47c2kf+zXJJoDDlRgFzr9xfcnrHFaJv
Yx

nuqNE0T+xmujnwGm3VrgddeAQJuW3sD6y0Ox8NgL4z886VFeaDQ0GmFPI6HEVt
g6

mixMhi+YzdkC+PFrEdYUeVNNwVO+bvJb1Rc08BYU4v7VtTkssHjru76E2/ahn/
Ct

kaVTEojEWeRaxsw5/0VLkgyf8SwDaukM2aamqgEzfsw5GTdSAh7ERZKc+zF7Sr5
s

DY8c5lOmyCwuNh9ODuw4cAThICrn7G8bh8ZyxLyj4Znxh0X45SwMZKTmYLfy9a
b8
        b/j7FK8uBNRL+pXl9HGBWAFA01uJw4HkYK+Uo+RcAzo=
        -----END CERTIFICATE-----
```

Допустимо указание неограниченного количества файлов, но **критически важно, чтобы каждый сертификат был в отдельном файле**.

#Переменные окружения

#Управление переменными окружения через values

Переменные окружения, а также конфигурационные файлы, используемые компонентами инсталляции, описаны в блоках `configMaps` и `secrets` в `values.yaml`. В `values-override.yaml` вынесены наиболее часто переопределяемые параметры конфигурации.

#Управление переменными окружения через External Secret

Также присутствует возможность подключать внешние хранилища секретов. Для этого в кластере должен быть установлен **External Secrets Operator (ESO)**. Он добавляет в кластер необходимые CRD (Custom Resource Definition) и обеспечивает связь с хранилищем секретов.

Для создания ресурса `ExternalSecret` и подключения его к существующему `SecretStore` необходимо добавить соответствующую запись в блок `vaults`:

```
vaults:
  ipcs-backend-secret-external: # название секрета, который
будет создан в результате
    apiVersion: external-secrets.io/v1 # по умолчанию external-
secrets.io/v1beta1
    enabled: true
    store:
      name: vault-backend # имя предварительно созданного
SecretStore
      kind: ClusterSecretStore # тип предварительно созданного
SecretStore
    path: secret/data/my-vault-secret # путь до секрета в Vault
```

Для передачи созданного секрета всем ресурсам `Deployment`, в поле `deploymentsGeneral.envSecrets` необходимо добавить в список название ресурса из блока `vaults`, например:

```
deploymentsGeneral:
  envConfigmaps:
    - ipcs-backend-env
  envSecrets:
    - ipcs-secrets
    - ipcs-backend-secret-external # имя ресурса из vaults
```

Для передачи секрета только одному ресурсу `Deployment`, необходимо указать его в блоке `deployments` наряду со всеми секретами, которые уже используются данным ресурсом, например:

```
deployments:
  ipcs-backend:
    envSecrets:
      - ipcs-secrets
      - ipcs-backend-secret-external
```

#Миграция с Helm Chart 2.x.x

Важно

Если использовалась встроенная PostgreSQL, перед любыми манипуляциями необходимо сделать бэкап.

Во время миграции инсталляция будет недоступна, поэтому рекомендуется выделять технологическое окно. Для проведения работ потребуется суммарно 5-10 минут.

Шаги для миграции:

1. Удалить все ресурсы Deployment
   `kubectl delete deployment --all -n codescoring`

2. Удалить все ресурсы Service
   `kubectl delete service --all -n codescoring`

3. Выполнить helm upgrade, предварительно заполнив values.yaml и values-override.yaml согласно пункту Настройки параметров Helm-чарта:

   ```
   helm upgrade codescoring codescoring-org/codescoring-helm
   --version <CHART_VERSION> -f values.yaml -f values-
   override.yaml -n codescoring
   ```

#Работа системы в Kubernetes (legacy)

Legacy-версия

Эта страница описывает legacy-версию Helm-чарта. Для новых установок используйте актуальную инструкцию Работа системы в Kubernetes.

#Установка с помощью Helm-чарта с параметрами по умолчанию

Важно!: Данный вариант установки не предоставляет возможность горизонтального масштабирования CodeScoring. Для установки CodeScoring с поддержкой горизонтального масштабирования обратитесь к соответствующему разделу документации ниже.

Важно!: Необходимо наличие настроенного default `StorageClass` в кластере. По умолчанию создаются тома **объемом 20 GiB**

Порядок установки:

1. Создать namespace.

   ```
   kubectl create namespace codescoring
   ```

2. Создать secret для доступа к приватному реестру Docker-образов системы "CodeScoring", используя адрес (REGISTRY_URL), логин (USERNAME) и пароль (PASSWORD), полученные от вендора.

```
kubectl create secret docker-registry codescoring-regcred
--docker-server=REGISTRY_URL --docker-username=USERNAME --
docker-password=PASSWORD -n codescoring
```

3. Установить Helm предпочтительным способом.

4. Выполнить следующие команды для добавления актуального Helm-репозитория на локальную машину:

```
helm repo add codescoring-org https://{REGISTRY_URL}/
repository/helm/ --username USERNAME --password PASSWORD
helm repo update
```

5. Создать файл values.yaml со следующим содержимым:

   Важно!: Пожалуйста, замените значения в полях с чувствительными данными на собственные. К таким полям относятся secretKey, defaultSuperuserUsername, defaultSuperuserPassword, defaultSuperuserEmail, а также все поля, содержащие username или password. Также важно учитывать, что все подобные переменные являются обязательными.

```
pgbouncer:
  enabled: true
  postgresql:
    host: "codescoring-postgresql"
    port: 5432
    username: "codescoring"
    password: "changeme"
    database: "codescoring"
  config:
    transactionPoolSize: 50
    transactionPoolMinSize: 1
    sessionPoolSize: 50
    sessionPoolMinSize: 1
    transactionPoolDatabaseName: "codescoring"
    sessionPoolDatabaseName: "codescoring-session"

codescoring:
  config:
    ## codescoring-backend configuration parameters
    siteScheme: https # схема сайта http или https
    siteHost: "codescoring.k8s.local" # домен, по которому
будет доступен CodeScoring
    djangoCSRFTrustedOptions: "https://
codescoring.k8s.local" # Домен, по которому будет доступен
```

```
CodeScoring, включая схему
    secretKey: "" # секретный ключ для бэкенда приложения,
случайная строка символов
    defaultSuperuserUsername: "admin" # имя администратора
в системе
    defaultSuperuserPassword: "changeme" # пароль
администратора в системе
    defaultSuperuserEmail: "mail@example.com" # e-mail
администратора в системе
    databaseHost: pgbouncer
    databasePort: 6432
    postgresqlDatabase: "codescoring"
    postgresqlUsername: "codescoring"
    postgresqlPassword: "changeme"

  frontend:
    ingress:
      enabled: true
      className: "nginx"
      hosts:
        - host: codescoring.k8s.local # домен, по которому
будет доступен CodeScoring
          paths:
            - path: /
              pathType: ImplementationSpecific
```

6. Выполнить команду для установки чарта

```
helm install codescoring codescoring-org/codescoring -n
codescoring -f values.yaml --create-namespace --atomic --
version CHART_VERSION
```

#Изменение пароля администратора

Для изменения пароля администратора без ручного редактирования файла `values.yaml` можно использовать следующую команду:

```
kubectl exec -it your-backend-pod -- python manage.py
changepassword <user_name>
```

#Расширенные настройки параметров Helm-чарта

Важно!: Настоятельно рекомендуется вносить необходимые изменения **до установки CodeScoring**, в противном случае может потребоваться полная переустановка системы. Данные инструкции предполагают, что **специалист имеет опыт работы с кластером Kubernetes и утилитой Helm**.

Для удобного редактирования параметров CodeScoring можно скачать и распаковать исходный код Helm-чарта командой:

```
helm pull codescoring-org/codescoring --version CHART_VERSION
--untar --untardir codescoring-src && cd codescoring-src
```

В файле `values.yaml` можно отредактировать нужные переменные, и после этого, находясь в каталоге с исходным кодом Helm-чарта, выполнить команду установки

```
helm install codescoring . -f values.yaml -n codescoring --
atomic --version CHART_VERSION
```

#Подключение к внешним PostgreSQL и Redis

По умолчанию PostgreSQL и Redis запускаются в отдельных `StatefulSet`. Данный вариант может не подходить для использования в **production-окружении** , т.к. не является отказоустойчивым.

Важно: При использовании собственной базы данных необходимо убедиться, что она соответствует требованиям.

#Подключение к внешнему Redis

Для подключения к внешнему Redis, необходимо выполнить следующие действия:

1. Отключить развертывание Redis, указав переменную - `redis.enabled: false`
2. В переменных `codescoring.config.djangoCachesRedisUrls` и `codescoring.config.hueyRedisUrl` указать строки подключения для внешнего Redis.

#Подключение к внешнему Redis с использованием TLS

Для подключения к внешнему Redis с использованием TLS, дополнительно необходимо:

1. Задать значение `true` в переменной `codescoring.trustedCA.enabled`
2. Добавить корневой сертификат сервера Redis в `codescoring.trustedCA.certificates`
3. В переменных `codescoring.config.djangoCachesRedisUrls` и `codescoring.config.hueyRedisUrl` указать строки подключения для внешнего Redis в формате `rediss://redis.example.com:6379/0`, где 0 - номер базы данных в Redis.

#Подключение к PostgreSQL через пулер Pgbouncer

Важно!: Подключение к внешней PostgreSQL необходимо выполнять с использованием пулера соединений.

Данный вариант подходит, если в существующей инфраструктуре уже развернута PostgreSQL, но пулер соединений не используется. Helm-чарт развернет пулер Pgbouncer и подключит его к существующей PostgreSQL. Необходимо выполнить следующие действия:

1. Отключить развертывание PostgreSQL, указав переменную - `postgresql.enabled: false`
2. Подключить пулер Pgbouncer к внешней PostgreSQL, заменив соответствующие параметры на нужные:

```
pgbouncer:
  postgresql:
    host: "postgresql.example.host"
    port: 5432
    username: "codescoring"
    password: "changeme"
    database: "codescoring"
  config:
    transactionPoolSize: 50
    transactionPoolMinSize: 1
    sessionPoolSize: 50
    sessionPoolMinSize: 1
    transactionPoolDatabaseName: "codescoring"
    sessionPoolDatabaseName: "codescoring-session"
codescoring:
  config:
    databaseHost: pgbouncer
    databasePort: 6432
    postgresqlDatabase: "codescoring"
    postgresqlUsername: "codescoring"
    postgresqlPassword: "changeme"
```

#Подключение к внешнему пулеру PostgreSQL

Данный вариант подходит, если в существующей инфраструктуре уже развернута PostgreSQL и пулер соединений (например, PgBouncer). В этом случае развертывание пулера Pgbouncer не требуется. Необходимо выполнить следующие действия:

1. Отключить развертывание PostgreSQL, указав переменную - `postgresql.enabled: false`
2. Отключить развертывание Pgbouncer, указав переменную - `pgbouncer.enabled: false`
3. Подключить codescoring напрямую к внешнему пулеру, в секции `codescoring.config` параметры:

```
posgtresqlHost: "external-pooler.example.host"
posgtresqlPort: 5432
postgresqlDatabase: "codescoring"
postgresqlUsername: "codescoring"
postgresqlPassword: "changeme"
```

1. Включить использование внешнего пулера и настроить параметры подключения в секции `codescoring.config.externalPooler`:

Важно!: Внешний пулер соединений должен поддерживать работу в двух режимах: транзакционном и сессионном. Данные для подключения к пулам указываются в соответствующих секциях: `externalPooler.transactionPool` и `externalPooler.sessionPool`.

```
externalPooler:
  enabled: true
  transctionPool:
    host: "external-pooler.example.host"
    port: 5432
    username: "codescoring"
    password: "changeme"
    database: "codescoring"
  sessionPool:
    host: "external-pooler.example.host"
    port: 5432
    username: "codescoring-session"
    password: "changeme"
    database: "codescoring-session"
```

#Настройка томов (PV)

По умолчанию чарт создает необходимые тома через Dynamic Volume Provisioning с использованием `StorageClass` по умолчанию (default). В случае, если данный вариант развертывания томов не подходит, присутствует возможность гибко настроить создание томов несколькими способами.

Важно!: Описанные ниже опции являются **взаимоисключающими**. Необходимо выбрать **ТОЛЬКО ОДИН** вариант развертывания для каждого тома. Допускается выбор разных вариантов развертывания для разных томов.

Tip

Для изменения размера создаваемых томов (за исключением локальных) необходимо изменить параметр `size` в соответствующих секциях

#Dynamic Volume Provisioning с использованием требуемого StorageClass

Задать требуемый `StorageClass` можно в следующих переменных:

- `codescoring.persistentVolumes.analysisRoot.storageClass`
- `codescoring.persistentVolumes.mediaRoot.storageClass`
- `codescoring.persistentVolumes.djangoStatic.storageClass`
- `codescoring.backup.persistentVolume.storageClass`
- `redis.persistentVolume.storageClass` (если используется встроенный Redis)
- `postgresql.persistentVolume.storageClass` (если используется встроенная PostgreSQL)

В этом случае, будут созданы тома с использованием заданного `StorageClass`

#PersistentVolumeClaim для заранее созданных PersistentVolume

Название предварительно созданных томов можно задать в следующих переменных:

- `codescoring.persistentVolumes.analysisRoot.volumeName`
- `codescoring.persistentVolumes.mediaRoot.volumeName`
- `codescoring.persistentVolumes.djangoStatic.volumeName`
- `codescoring.backup.persistentVolume.volumeName`
- `redis.persistentVolume.volumeName` (если используется встроенный Redis)
- `postgresql.persistentVolume.volumeName` (если используется встроенная PostgreSQL)

В этом случае будут созданы только `PersistentVolumeClaim` для томов, заданных в этих переменных

#Использование предварительно созданных PersistentVolumeClaim

Название предварительно созданных PVC можно задать в следующих переменных:

- `codescoring.persistentVolumes.analysisRoot.existingClaim`
- `codescoring.persistentVolumes.mediaRoot.existingClaim`
- `codescoring.persistentVolumes.djangoStatic.existingClaim`
- `codescoring.backup.persistentVolume.existingClaim`
- `redis.persistentVolume.existingClaim` (если используется встроенный Redis)
- `postgresql.persistentVolume.exsistingClaim` (если используется встроенная PostgreSQL)

В этом случае указанное название PVC будет подставлено в секцию `volumes` для Pod напрямую.

#Использование локальных томов

При отсутствии в кластере Kubernetes внешнего хранилища данных возможен запуск CodeScoring с использованием локальных томов. В этом случае данные будут хранится на одной из нод кластера.

Для создания локальных томов необходимо выполнить следующие действия:

1. Присвоить значение `true` следующим переменным:
    - `codescoring.persistentVolumes.analysisRoot.localVolume.enabled`
    - `codescoring.persistentVolumes.mediaRoot.localVolume.enabled`
    - `codescoring.persistentVolumes.djangoStatic.localVolume.enabled`
    - `codescoring.backup.persistentVolume.localVolume.enabled`
    - `redis.persistentVolume.localVolume.enabled` (если используется встроенный Redis)
    - `postgresql.persistentVolume.localVolume.enabled` (если используется встроенная PostgreSQL)
2. Задать путь до **каталога на ноде кластера**, в котором будут размещены данные в следующих переменных:
    - `codescoring.persistentVolumes.analysisRoot.localVolume.path`
    - `codescoring.persistentVolumes.mediaRoot.localVolume.path`
    - `codescoring.persistentVolumes.djangoStatic.localVolume.path`
    - `codescoring.backup.persistentVolume.localVolume.path`
    - `redis.persistentVolume.localVolume.path` (если используется встроенный Redis)
    - `postgresql.persistentVolume.localVolume.path` (если используется встроенная PostgreSQL)
3. Указать название ноды, на которой будет создан локальный том в следующих переменных:
    - `codescoring.persistentVolumes.analysisRoot.localVolume.nodeHostname`
    - `codescoring.persistentVolumes.mediaRoot.localVolume.nodeHostname`
    - `codescoring.persistentVolumes.djangoStatic.localVolume.nodeHostname`
    - `codescoring.backup.persistentVolume.localVolume.nodeHostname`
    - `redis.persistentVolume.localVolume.nodeHostname` (если используется встроенный Redis)
    - `postgresql.persistentVolume.localVolume.nodeHostname` (если используется встроенная PostgreSQL)

Допускается использование разных нод для разных томов.

#Настройка хранилища для временных файлов сканирований

По умолчанию временные файлы в процессе сканирования хранятся в директории `/tmp` внутри контейнеров, к которой монтируются Ephemeral Volumes типа `emptyDir`:

- `codescoring.huey.ipcsQueue.ephemeralVolumes`
- `codescoring.huey.tasksOsaContainerImageScan.ephemeralVolumes`
- `codescoring.huey.tasksOsaPackageScan.ephemeralVolumes`

Однако в некоторых случаях может потребоваться использовать Persistent Volume вместо Ephemeral Volume. В таком случае следует закомментировать соответствующие секции в `ephemeralVolumes` для одного или нескольких сервисов, в зависимости от того, для каких сервисов требуется монтировать тома:

```
codescoring:
  huey:
    ipcsQueue:
      ephemeralVolumes:
        volumeMounts:
        # - mountPath: /tmp
        #   name: ipcs-queue-tmp
        - mountPath: /etc/ssl/certs
          name: ipcs-queue-ssl-certs
        volumes:
        # - name: ipcs-queue-tmp
        #   emptyDir: {}
        - name: ipcs-queue-ssl-certs
          emptyDir: {}

    tasksOsaContainerImageScan:
      ephemeralVolumes:
        volumeMounts:
        # - mountPath: /tmp
        #   name: container-image-scan-tmp
        - mountPath: /etc/ssl/certs
          name: container-image-scan-ssl-certs
        volumes:
        # - name: container-image-scan-tmp
        #   emptyDir: {}
        - name: container-image-scan-ssl-certs
          emptyDir: {}

    tasksOsaPackageScan:
      ephemeralVolumes:
        volumeMounts:
        # - mountPath: /tmp
        #   name: package-scan-tmp
```

```
        - mountPath: /etc/ssl/certs
          name: package-scan-ssl-certs
        volumes:
        # - name: package-scan-tmp
        #   emptyDir: {}
        - name: package-scan-ssl-certs
          emptyDir: {}
```

После необходимо выставить значение `enabled: true` в одной или нескольких из следующих секций:

- `codescoring.huey.persistentVolumes.hueyTmp`
- `codescoring.huey.persistentVolumes.hueyPackageScanTmp`
- `codescoring.huey.persistentVolumes.hueyContainerImageScanTmp`

В результате будут созданы PersistentVolumeClaim для соответствующих сервисов. Стоит отметить, что возможности конфигурирования данных томов полностью соответствуют описанным в секции Настройка томов (PV).

При горизонтальном масштабировании сервисов, необходимо произвести настройку томов в соответствии с инструкцией в разделе Горизонтальное масштабирование CodeScoring.

#Горизонтальное масштабирование CodeScoring

Важно!: Для горизонтального масштабирования системы CodeScoring необходимо наличие в кластере Kubernetes возможности создания томов с типом доступа **ReadWriteMany (RWX)**

Для горизонтального масштабирования CodeScoring необходимо создать тома `analysis-root`, `media-root` и `django-static` с типом доступа `ReadWriteMany`.

Для этого необходимо заменить значение `ReadWriteOnce` на `ReadWriteMany` в переменных:

- `codescoring.persistentVolumes.analysisRoot.accessModes`
- `codescoring.persistentVolumes.mediaRoot.accessModes`
- `codescoring.persistentVolumes.djangoStatic.accessModes`

Затем, необходимо закоментировать переменные:

- `codescoring.backend.affinity`
- `codescoring.frontend.affinity`

Если этого не сделать, то все поды будут запущены только на одной ноде кластера.

#Настройка ограничения ресурсов (resource limits)

По умолчанию `requests` и `limits` не заданы. Это сделано для обеспечения возможности запуска системы CodeScoring в кластерах с малым количеством ресурсов (например, minikube) с целью тестирования. При запуске в **production-окружении** может потребоваться настроить ограничение ресурсов. Это можно сделать, задав следующие переменные:

- `postgresql.resources` (при использовании встроенной PostgreSQL)
- `redis.resources` (при использовании встроенного Redis)
- `codescoring.backend.resources`
- `codescoring.frontend.resources`
- `codescoring.huey.highPriorityQueue.resources`
- `codescoring.huey.ipcsQueue.resources`
- `codescoring.huey.tasksOsaContainerImageScan.resources`
- `codescoring.huey.tasksOsaPackageScan.resources`
- `codescoring.huey.tasksPolicy.resources`
- `codescoring.huey.tasksMedia.resources`

Возможно указание как `resources` и `limits` вместе, так и по отдельности, например:

```
codescoring:
  backend:
    resources:
      limits:
        cpu: 1000m
        memory: 2000Mi
  huey:
    ipcsQueue:
      resources:
        limits:
          cpu: 2000m
          memory: 3000Mi
        requests:
          cpu: 1000m
          memory: 1000Mi
```

Ниже приведены примерные значения `limits` для платформе CodeScoring с 8-10 проектами:

```
codescoring:
  backend:
    resources:
      limits:
        cpu: 250m
        memory: 2500Mi
  huey:
    ipcsQueue:
      scheduler:
```

```
        resources:
          limits:
            cpu: 500m
            memory: 500Mi
      resources:
        limits:
          cpu: 2250m
          memory: 4000Mi
    highPriorityQueue:
      resources:
        limits:
          cpu: 2250m
          memory: 4000Mi
    tasksOsaContainerImageScan:
      resources:
        limits:
          cpu: 2250m
          memory: 4000Mi
    tasksOsaPackageScan:
      resources:
        limits:
          cpu: 2250m
          memory: 4000Mi
    tasksOsaPackageScan:
      resources:
        limits:
          cpu: 2250m
          memory: 4000Mi
    tasksPolicy:
      resources:
        limits:
          cpu: 2250m
          memory: 4000Mi
    tasksTqi:
      resources:
        limits:
          cpu: 2250m
          memory: 4000Mi
    tasksMedia:
      resources:
        limits:
          cpu: 1000m
          memory: 1500Mi
  frontend:
    resources:
      limits:
        cpu: 250m
        memory: 500Mi
  redis:
    resources:
      limits:
        cpu: 1000m
```

```
        memory: 2000Mi
  postgresql:
    resources:
      limits:
        cpu: 1000m
        memory: 2000Mi
```

Также возможна настройка ресурсов init-контейнеров. Для сервиса backend они настраиваются в разделе

```
codescoring:
  backend:
    initContainers:
      resources:
        limits:
          cpu: 2500m
          memory: 2500Mi
        requests:
          cpu: 500m
          memory: 500Mi
```

Для всех прочих сервисов init-контейнеры являются однотипными и выполняют функции wait-for для обеспечения последовательного запуска сервисов. Ресурсы для них настраиваются в разделе

```
codescoring:
  initContainers:
    limits:
      cpu: 2500m
      memory: 2500Mi
    requests:
      cpu: 200m
      memory: 200Mi
```

#Добавление сертификата удостоверяющего центра (CA)

Для доступа CodeScoring к ресурсам с TLS-сертификатами, подписанными корпоративным удостоверяющим центром (CA) необходимо:

1. Присвоить переменной `codescoring.trustedCA.enabled` значение `true`
2. Добавить корневой сертификат удостоверяющего центра (RootCA) в формате PEM в переменную `codescoring.trustedCA.certificates` в формате `ключ: значение`, где ключ - имя файла сертификата, включая расширение `.crt`, значение - сертификат в формате PEM.

Например:

```
codescoring:
  trustedCA:
    enabled: true
    certificates:
        ## THIS IS AN EXAMPLE ONE! DO NOT USE IN PRODUCTION!
        my-root-ca.crt: |-
          -----BEGIN CERTIFICATE-----

MIIDTDCCAjSgAwIBAgIBATANBgkqhkiG9w0BAQUFADA3MQswCQYDVQQGEwJERT
EP

MA0GA1UEChMGZWR1UEtJMRcwFQYDVQQDEw5lZHVQS0kgVGVzdCBDQTAeFw0xMD
Az

MzExMjIwMjRaFw0zMDAzMjYxMjIwMjRaMDcxCzAJBgNVBAYTAkRFMQ8wDQYDVQ
QK

EwZlZHVQS0kxFzAVBgNVBAMTDmVkdVBLSSBUZXN0IENBMIIBIjANBgkqhkiG9w0
B
          AQEFAAOCAQ8AMIIBCgKCAQEAt5IxCk/
NQPOLqeA1lGuB3pvqHGQPxRQ1udYGcXQY

t7EuSMFymUR9m5TsifG1ktktJTtOWyaWFC4ac0vai49wGVeuDYptfZBoHLIUvC
wN

DOofLYHxk04WzfrtSiUTptn1o6QPOw8YR0XH30MEi1zgD8fLMZmVTJ+XwA5Eus6
c
          XtTmI4XhNrHUtvWt4UsNgLmp5/
djUgRMpNqxIdrpFQzl+XycRJRAaoAwUzHFl14t

49qwBhGChxQ8AdDMQGA7kv6VR8o0ktCPv3a4GQbs8+z0cX0w5dC+XhJ1xpqW6T
Og

qAY9XBFIDe5j21hjKmNZ39rsODVGUS2wUtNEhSz+3YqxLwIDAQABo2MwYTAdBg
NV

HQ4EFgQUqHe3saMjZZLan8RlFJs+Xuz4yiAwHwYDVR0jBBgwFoAUqHe3saMjZZ
La
          n8RlFJs+Xuz4yiAwDwYDVR0TAQH/BAUwAwEB/
zAOBgNVHQ8BAf8EBAMCAQYwDQYJ

KoZIhvcNAQEFBQADggEBAEjQGyHZQis47c2kf+zXJJoDDlRgFzr9xfcnrHFaJv
Yx

nuqNE0T+xmujnwGm3VrgddeAQJuW3sD6y0Ox8NgL4z886VFeaDQ0GmFPI6HEVt
g6

mixMhi+YzdkC+PFrEdYUeVNNwVO+bvJb1Rc08BYU4v7VtTkssHjru76E2/ahn/
Ct

kaVTEojEWeRaxsw5/0VLkgyf8SwDaukM2aamqgEzfsw5GTdSAh7ERZKc+zF7Sr5
```

```
s

DY8c5lOmyCwuNh9ODuw4cAThICrn7G8bh8ZyxLyj4Znxh0X45SwMZKTmYLfy9a
b8
          b/j7FK8uBNRL+pXl9HGBWAFA01uJw4HkYK+Uo+RcAzo=
          -----END CERTIFICATE-----
```

В случае наличия нескольких корневых CA необходимо добавить их в отдельные ключи, например:

```
codescoring:
  trustedCA:
    enabled: true
    certificates:
        my-root-ca.crt: |-
          ...
        my-root-ca-2.crt: |-
          ...
```

#Управление секретами

По умолчанию для шаблонов `ipcs-backend`, `pgbouncer` и `postgresql` предусмотрены объекты типа `Secret`. Значения переменных в этих объектах заполняются из содержимого `values`.

Также присутствует возможность подключать внешние хранилища секретов. Для этого в кластере должен должен быть установлен **External Secrets Operator (ESO)**. Он добавляет в кластер необходимые CRD (Custom Resource Definition) и обеспечивает связь с хранилищем секретов.

Для подключения ESO к внешнему хранилищу секретов необходимо сконфигурировать провайдера для **SecretStore** в разделе `codescoring.secretStore`.

Далее, необходимо настроить объекты **ExternalSecret** для получения секретов из внешнего хранилища в разделах `codescoring.config.externalSecret`, `pgbouncer.externalSecret`, `postgresql.externalSecret`.

Вся конфигурация осуществляется в соответствии с документацией ESO.

Важно!: Некоторые провайдеры могут тарифицировать запросы к хранилищам секретов. Интервал запроса данных регулируется параметром `externalSecret.refreshInterval` для каждого отдельного сервиса.

#Мониторинг

Для сбора метрик с сервисов в чарте предусмотрены ресурсы **ServiceMonitor**. Метрики собираются с сервисов backend и osa-api. Для использования **ServiceMonitor** в кластере должен быть установлен и настроен Prometheus Operator.

ServiceMonitor настраивается в следующих разделах values: `codescoring.backend.prometheus.serviceMonitor`, `codescoring.osa_api.prometheus.serviceMonitor`.

Также для вышеуказанных сервисов предусмотрены ресурсы **PrometheusRule**, необходимые для настройки правил алертинга. Настройка данных ресурсов осуществляется в следующих разделах values: `codescoring.backend.prometheus.alerts`, `codescoring.osa_api.prometheus.alerts`.

Все настройки осуществляются в соответствии с документацией Prometheus Operator.

#Обновление системы

Для обновления системы необходимо актуализировать helm-репозиторий командой

```
helm repo update
```

и далее выполнить команду обновления платформы, где CHART_VERSION - версия чарта, на которую происходит обновление

```
helm upgrade codescoring codescoring-org/codescoring -n
codescoring -f values.yaml --version CHART_VERSION
```

#Установка оффлайн-версии

CodeScoring поддерживает работу в закрытом контуре. Для доступа к базе данных о пакетах и уязвимостях в таком режиме используется сервис Index API Offline.

Предварительная установка продукта

Для использования Index API Offline на сервере должна быть предварительно развернута платформа CodeScoring с использованием Docker или Kubernetes.

#Ресурсы для установки и обновления

Адрес ресурса с установочными файлами, Docker Registry и файлами оффлайн-БД можно узнать у вендора.

В примерах ниже используются плейсхолдеры:

- <REGISTRY_URL> — HTTPS-адрес ресурса с файлами дистрибутива;
- <REGISTRY_HOST> — адрес Docker Registry без протокола.

На ресурсе доступны:

- Docker Registry с образами CodeScoring и Index API Offline
- Полная версия оффлайн-БД Index API: <REGISTRY_URL>/db/index-api.db
- Инкрементальные обновления оффлайн-БД: <REGISTRY_URL>/#browse/browse:codescoring-offline-files:db%2Fv1%2Fupdates

Оффлайн база данных представляет собой шифрованный файл SQLite. Инкрементальные обновления распространяются в виде WAL-файлов.

#Скачивание базы данных

1. Выберите сервер с доступным дисковым пространством не менее 300 ГБ.

2. Создайте файл curl.config со следующими параметрами доступа:

```
user = <REGISTRY_LOGIN>:<REGISTRY_PASSWORD>
```

3. Запустите скачивание базы данных:

```
curl --config curl.config \
     -C - \
     --output index-api.db \
     <REGISTRY_URL>/db/index-api.db
```

Рекомендуется выполнять загрузку в screen или tmux, так как процесс может занимать значительное время.

#Установка Index API Offline

1. Выполните авторизацию в Docker Registry:

```
docker login <REGISTRY_HOST>
```

2. Скачайте архив с установочными файлами и распакуйте его:

```
curl -u <REGISTRY_LOGIN>:<REGISTRY_PASSWORD> \
     -C - \
     <REGISTRY_URL>/repository/codescoring-offline-files/
index-api/docker-compose/<INDEX_API_OFFLINE_VERSION>.tar.
```

```
gz \
     -o index-api-offline.tar.gz
```

3. Переместите скачанный файл базы данных в директорию сервиса, например:

```
mv index-api.db index-api-offline/db
```

4. Перейдите в директорию с конфигурацией:

```
cd index-api-offline/
```

5. Скопируйте шаблон конфигурации `.env.template` в `.env` и заполните его:

   Основные параметры: - `INDEX_API_OFFLINE_VERSION` — версия Index API Offline; - `NGINX_SSL_ENABLED` — включение SSL (true | false); - `NGINX_HOST` — имя хоста для nginx; - `OFFLINE_DB_UPDATE_ENABLED` — включение автоматического обновления базы данных; - `WAL_CHECK_INTERVAL` — интервал проверки обновлений; - `NEXUS_HOST` — адрес репозитория обновлений; - `NEXUS_USERNAME` — логин для доступа к Nexus; - `NEXUS_PASSWORD` — пароль для доступа к Nexus; - `CODESCORING_ACTIVATION_KEY` — лицензионный ключ.

   Параметры аутентификации для обновления базы данных и получения метрик через Prometheus (опционально): - `MAINTENANCE_USERNAME` - Basic Auth логин для метода `/system/update_database`; - `MAINTENANCE_PASSWORD` - Basic Auth пароль для метода `/system/update_database`; - `METRICS_USERNAME` - Basic Auth логин для метода `/metrics`; - `METRICS_PASSWORD` - Basic Auth пароль для метода `/metrics`.

6. При использовании SSL (`NGINX_SSL_ENABLED=true`) разместите сертификат и ключ в каталоге `ssl`.

   Важно

   Формат сертификата – PEM. Файл сертификата должен иметь расширение `.crt`, а файл ключа сертификата должен иметь расширение `.key`

7. Запустите сервис:

```
docker compose up -d --remove-orphans
```

После установки и запуска в интерфейсе CodeScoring можно посмотреть версию последнего загруженного в Index API Offline WAL-файла.

#Обновление базы данных

#Автоматическое обновление оффлайн базы данных

Процесс обновления базы данных можно выполнить автоматически.

Механизм обновления работает следующим образом:

1. Запущенный сервис Index API Offline с заданной периодичностью опрашивает репозиторий обновлений.
2. Сервис хранит номер текущей версии базы данных (например, 72).
3. При обнаружении следующего доступного обновления (например, 73) файл загружается автоматически.
4. Загруженный WAL-файл применяется к локальной базе данных.

Интервал проверки задаётся параметром `WAL_CHECK_INTERVAL`.

При отключённом параметре `OFFLINE_DB_UPDATE_ENABLED` автоматическое обновление не выполняется.

#Ручное обновление базы данных

Ручное обновление используется в случаях, когда автоматическая загрузка WAL-файлов отключена (`OFFLINE_DB_UPDATE_ENABLED=false`) или невозможна по сетевым ограничениям.

#Обновление с использованием отдельных файлов обновлений

Если развёрнута **первая версия оффлайн базы данных**, необходимо последовательно скачать и загрузить **все доступные WAL-файлы обновлений**, начиная с версии, следующей за текущей версией базы данных.

Файлы обновлений необходимо размещать в директории, указанной в параметре `WAL_DIR` в конфигурации Index API Offline. Сервис отслеживает появление файлов в этой директории и применяет их к базе данных.

В комплект обновления входят:

- `*.wal` — файл с изменениями (основные данные обновления);
- `*.shm` — вспомогательный служебный файл, необходимый SQLite для корректного применения WAL.

#Обновление с использованием полной версии базы данных

В случае загрузки актуального файла **полной оффлайн-БД** необходимо выполнить следующие действия:

1. Остановить сервис Index API Offline.
2. Переместить текущий файл базы данных в резервную директорию (рекомендуется сохранить копию до завершения обновления).

3. Разместить новый файл базы данных в директории, указанной в параметре DB_FILE.
4. Запустить сервис Index API Offline.
5. Убедиться, что сервис успешно запущен.

После этого, при необходимости, можно продолжить обновление с помощью WAL-файлов, размещая их в директории WAL_DIR в стандартном порядке.

#Запуск процесса обновления

При использовании ручного режима обновления запуск процесса обновления также выполняется вручную.

1. В файле .env необходимо указать параметры аутентификации:

```
MAINTENANCE_USERNAME=
MAINTENANCE_PASSWORD=
```

2. Файлы обновлений (*.wal, *.shm) необходимо разместить в директории, указанной в параметре WAL_DIR.

Для запуска процесса обновления выполните запрос:

```
curl -X GET --location "{{INDEX_API_URL}}/system/
update_database" \
     -H "Accept: application/json" \
     -H "Authorization: Basic {{metrics_auth}}"
```

Где:

- {{INDEX_API_URL}} — значение из переменной INDEX_API_URL в .env;
- {{metrics_auth}} — Base64-кодированная строка вида user:password.

Статус обновления отслеживается в логах сервиса Index API Offline.

Возможные сообщения:

- WAL files download scheduler started with interval: — сервис обновлений запущен;
- missing mandatory Nexus configuration parameters — допустимое сообщение для ручного режима;
- The WAL file [%s] was successfully loaded into the main database file
  — обновление успешно применено;
- Failed to checkpoint WAL file: — ошибка применения WAL-файла;
- No WAL files to checkpoint — отсутствуют файлы для применения;

- `Error getting latest database update:` — ошибка получения информации об обновлениях.

#Обновление системы

#Стандартная инструкция по обновлению

Резервное копирование

Перед обновлением обязательно выполните резервное копирование платформы.

Обновите compose-файлы перед запуском новой версии

Недостаточно изменить только значение `CODESCORING_VERSION` в `.env`. Перед обновлением обязательно замените `docker-compose.yml`, `external-db.override.yml`, `app.env` и `.env` на актуальные версии, полученные у вендора. Если оставить старые compose-файлы, платформа может запуститься с неполной или некорректной конфигурацией.

В переменной `CODESCORING_VERSION` внутри файла `.env` указывается требуемая версия системы. Актуальную версию можно узнать в разделе Changelog.

Затем нужно выполнить следующие шаги:

1. Перейти в директорию с файлами запуска:

   ```
   cd /path/to/docker/compose
   ```

2. Выполнить команду обновления образов:

   ```
   docker compose pull
   ```

3. Перезапустить платформу:

   ```
   docker compose down --remove-orphans
   docker compose up -d --renew-anon-volumes
   ```

#Восстановление предыдущей версии

Если после обновления возникли ошибки или система работает нестабильно, можно восстановить предыдущую версию платформы из резервной копии:

1. Остановите текущую инсталляцию:

```
docker compose down
```

2. Очистите базу данных любым удобным способом:
    - через Docker:

      ```
      docker volume rm <db_volume_name>
      ```

    - или удалив БД напрямую (например, `DROP DATABASE`);
    - или, при использовании Kubernetes:

      ```
      kubectl delete pvc <db_pvc_name>
      ```

3. Восстановите базу данных из ранее созданного бэкапа.
4. В файле `.env` установите прежнее значение переменной `CODESCORING_VERSION`.
5. Перезапустите платформу:

   ```
   docker compose up -d
   ```

Подробная инструкция по созданию резервных копий доступна в разделе Резервное копирование.

#Инструкции по обновлению на версии с измененной конфигурацией

#[2025.21.0] – 2025-05-21

Начиная с данной версии, значение переменной окружения $SECRET_KEY будет использоваться для шифрования чувствительных данных в базе данных и изменение значения этой переменной будет требовать дополнительных операций.

Перед обновлением необходимо убедиться, что в файле `.env` указано корректное (**уникальное, непредсказуемое**) значение $SECRET_KEY, а не значение по умолчанию.

#[2025.13.0] - 2025-03-28

- Необходимо убедиться, что версия `Docker Engine` больше или равна 25. Для этого нужно выполнить команду `docker version` на машине с платформой. В случае, если версия Docker Engine ниже, чем 25, необходимо обновить Docker.
    - **ВАЖНО!** Перед обновлением Docker необходимо штатно остановить платформу.

- Необходимо внести название проекта docker compose в конфигурацию:
  - Перед выключением системы для обновления, необходимо отметить название docker compose проекта, в котором сейчас запущена платформа.
  - Это либо значение, передаваемое с параметром `-p` для `docker compose`, либо название директории, в которой находился `docker-compose.yml` файл, по умолчанию `--on-premise` или `on-premise-split-db`
  - Это значение используется как префикс в названии ресурсов, создаваемых compose: томов, контейнеров, сетей
  - Необходимо вписать это значение в `.env` файл с ключом `COMPOSE_PROJECT_NAME=`
  - **ВАЖНО!** Если этого не сделать, то платформа не запустится. Если вписать некорректное значение, то создадутся томы с новым префиксом, и платформа на новой версии запустится "с нуля"
  - После того, как значение добавлено в `.env` файл, вызовы к `docker compose` можно делать без опции `-p PROJECT_NAME`
- Необходимо скачать из реестра CodeScoring обновлённые файлы `docker-compose.yml` и `external-db.override.yml` и поместить их в директорию с compose файлом.

#Обновление PostgreSQL в Docker

Обновление мажорной версии PostgreSQL требует инициализации БД с использованием новой версии PostgreSQL, создания дампа и его восстановления. Ниже описана процедура для инсталляций CodeScoring, развёрнутых с помощью Docker Compose.

Корректный порядок обновления

Не рекомендуется выполнять обновление инсталляции и базы данных одновременно. Обновление следует проводить последовательно: сначала обновить инсталляцию и убедиться в корректной работе сервиса, после чего выполнять обновление базы данных.

#Необходимые условия

#Дисковое пространство

Требуется свободное дисковое пространство в объёме не менее **размера тома `db-data`**.

Пример утилизации дискового пространства при обновлении:

- версии PostgreSQL:
  - исходная: 13.21
  - новая: 15.15

- данные инсталляции CodeScoring:
  - ~500 000 пакетов
  - ~1 200 проектов
  - ~22 000 образов
- размер тома `db-data`: ~50 GiB
- потребление дискового пространства в процессе обновления:
  - сжатая резервная копия: ~9 GiB
  - сжатый дамп: ~3 GiB
  - размер `db-data` после восстановления: ~32 GiB

#Окно планового технического обслуживания

Процедура требует полной остановки CodeScoring. Фактическое время выполнения зависит от объёма данных.

Для примера выше:

- создание сжатой резервной копии: ~11 минут 45 секунд
- восстановление резервной копии: ~2 минуты 30 секунд
- `pg_dump` + `pg_restore`: ~3 минуты 30 секунд
- перенос данных между томами: ~1 минута
- сбор статистики планировщика: ~1 секунда

#Переопределение конфигурации Docker Compose

Для обновления требуется файл переопределения конфигурации Docker Compose. Файл `postgres-upgrade.override.yml` поставляется, начиная с версии CodeScoring 2026.3.1, и располагается в той же директории, что и файл `docker-compose.yml`.

#Описание компонентов файла переопределения конфигурации

#Тома

- `upgrade-new-db-data` - временный pgdata PostgreSQL новой версии
- `upgrade-dump` - сжатый дамп
- `upgrade-backup` - сжатый бэкап

#Сервисы

- `psql-new` - PostgreSQL новой версии с конфигурацией, идентичной сервису `psql`
- `upgrade-dump` - создание сжатого дампа
- `upgrade-dump-restore` - восстановление дампа в PostgreSQL новой версии
- `upgrade-cleanup-transfer` - очистка и перенос данных в основной том `db-data`
- `upgrade-analyze-in-stages` - итеративный сбор статистики планировщика
- `upgrade-backup` - создание резервной копии
- `upgrade-backup-restore` - восстановление резервной копии

#Резервное копирование

#Создание резервной копии

Остановите CodeScoring:

```
docker compose \
  -f docker-compose.yml \
  -f postgres-upgrade.override.yml \
  down
```

Создайте резервную копию:

```
docker compose \
    -f docker-compose.yml \
    -f postgres-upgrade.override.yml \
    run --rm upgrade-backup
```

#Восстановление резервной копии

Остановите CodeScoring:

```
docker compose \
  -f docker-compose.yml \
  -f postgres-upgrade.override.yml \
  down
```

Восстановите резервную копию:

```
docker compose \
    -f docker-compose.yml \
    -f postgres-upgrade.override.yml \
    run --rm upgrade-backup-restore
```

Убедитесь, что значение `POSTGRES_IMAGE` в файле `.env` соответствует версии PostgreSQL **до** обновления:

```
POSTGRES_IMAGE=postgres:13.23-rev1 # версия, с которой
обновляемся
```

#Процедура обновления

Установите в .env:

```
POSTGRES_IMAGE=postgres:13.23-rev1 # версия, с которой
обновляемся
```

```
POSTGRES_UPGRADE_IMAGE=postgres:15.15-rev1
# версия, на которую обновляемся
```

Загрузите образы, необходимые для обновления:

```
docker compose \
  -f docker-compose.yml \
  -f postgres-upgrade.override.yml \
  pull
```

Остановите все сервисы:

```
docker compose \
  -f docker-compose.yml \
  -f postgres-upgrade.override.yml \
  down
```

Инициализируйте PostgreSQL новой версии, выполните дамп и восстановление:

```
docker compose \
  -f docker-compose.yml \
  -f postgres-upgrade.override.yml \
  run --rm upgrade-dump-restore
```

Снова остановите сервисы:

```
docker compose \
  -f docker-compose.yml \
  -f postgres-upgrade.override.yml \
  down
```

Перенесите данные в основной том `db-data`:

```
docker compose \
  -f docker-compose.yml \
  -f postgres-upgrade.override.yml \
  run --rm upgrade-cleanup-transfer
```

Установите версию PostgreSQL для образа, используемого сервисом `psql`, в файле `.env`:

```
POSTGRES_IMAGE=postgres:15.15-rev1 # версия, на которую
обновились
```

Запустите сбор статистики (можно не дожидаться завершения):

```
docker compose \
  -f docker-compose.yml \
  -f postgres-upgrade.override.yml \
  run --rm upgrade-analyze-in-stages
```

Запустите CodeScoring:

```
docker compose -f docker-compose.yml up \
  --detach \
  --force-recreate \
  --remove-orphans \
  --renew-anon-volumes
```

Проверьте логи и убедитесь в работоспособности инсталляции CodeScoring:

```
docker compose logs -f
```

Удалите следующие тома:

- `${COMPOSE_PROJECT_NAME}_upgrade-new-db-data`
- `${COMPOSE_PROJECT_NAME}_upgrade-dump`
- `${COMPOSE_PROJECT_NAME}_upgrade-backup`

#Резервное копирование

#Создание резервной копии установки

1. Перейти в директорию с файлами запуска:

   ```
   cd /path/to/docker/compose
   ```

2. Для создания резервной копии выполнить команду:

   ```
   docker compose run backup create
   ```

   Файл резервной копии сохранится в директорию backup.

#Восстановление из резервной копии

1. Для восстановления из резервной копии выполнить команду:

   ```
   docker compose run backup restore BACKUP_FILENAME
   ```

BACKUP_FILENAME — имя файла резервной копии. Список доступных резервных копий можно получить выполнив команду:

```
ls -la ./backup
```

2. Перезапустить платформу:

```
docker compose up -d --force-recreate --renew-anon-volumes
```

#Работа через прокси

При необходимости работы системы через прокси необходимо раскомментировать и задать значения соответствующих переменных в файле app.env:

- HTTP_PROXY и HTTPS_PROXY — адрес прокси-сервера

  Важно: для обеих переменных в значении должна использоваться схема http.

- NO_PROXY — список URL внешних систем, запросы к которым не должны идти через прокси. Среди значений возможно указание IP адресов, например:

```
NO_PROXY=127.0.0.1,192.168.0.1/24,example.com,domain.example.com,.subdomain.example.com
```

Если добавить список URL VCS в переменную NO_PROXY невозможно, то необходимо добавить сертификат прокси в каталог SSL.

Важно: при заданных настройках HTTP Proxy мимо прокси-сервера будут идти все запросы к URL внешних систем, указанных в переменной NO_PROXY. Это относится как к системам контроля версий, так и, например, к подключенным таск-менеджерам.

#Работа с самоподписанными сертификатами

При работе с внешними системами CodeScoring проверяет валидность SSL-сертификатов удалённых хостов и по умолчанию не будет подключать проекты из систем, сертификаты которых не прошли валидацию или подписаны неизвестным системе удостоверяющим центром.

Чтобы подключить внешнюю систему, SSL-сертификат домена которой является самоподписанным, требуется добавить корневой сертификат в

доверенные (trusted) на уровне установки. Для этого перед запуском системы его необходимо положить в директорию `ssl` в установочных файлах системы. Желательно дать файлу говорящее название, например, `codescoring-root-CA.crt`.

Важно: расширение файла обязательно должно быть `crt`.

Чтобы посмотреть всю цепочку используемых для домена сертификатов и выделить корневой, можно использовать команду:

```
openssl s_client -showcerts -partial_chain -connect
DOMAIN.NAME:443
```

#Подключение к PostgreSQL/PgBouncer с использованием TLS

Подключение к PostgreSQL/PgBouncer с использованием TLS обеспечивает безопасную и зашифрованную передачу данных между клиентом и сервером. Данная функциональность доступна начиная с версии CodeScoring 2025.21.0.

#Инструкция для подключение

1. Перед запуском системы разместить файлы сертификатов и ключей в каталоге `ssl` в установочных файлах системы;
2. Дать файлам говорящие названия, например `pgbouncer_server.crt`, `pgbouncer_server.key`;

   Важно: расширение файлов сертификатов обязательно должно быть `crt`;
3. Сменить владельца файлов ключей следующим образом:

   ```
   chown 999:0 ./ssl/postgresql_server.key
   chown 1050:0 ./ssl/pgbouncer_server.key
   ```
4. Раскомментировать и отредактировать файлы конфигурации SSL/TLS.

#Пример включения PostgreSQL/PgBouncer в режим TLS с самоподписанными сертификатами

1. Сгенерировать сертификаты в каталоге `ssl` в установочных файлах системы, используя утилиту `mkcert`

   ```
   docker run -v ./ssl:/ssl -it --rm alpine/mkcert -cert-
   file /ssl/pgbouncer_server.crt -key-file /ssl/
   ```

```
pgbouncer_server.key pgbouncer
docker run -v ./ssl:/ssl -it --rm alpine/mkcert -cert-
file /ssl/postgresql_server.crt -key-file /ssl/
postgresql_server.key psql
```

2. Сменить владельцев файлов ключей

```
chown 999:0 ./ssl/postgresql_server.key
chown 1050:0 ./ssl/pgbouncer_server.key
```

3. Скопировать шаблоны конфигурации SSL/TLS

```
cp postgres/pgbouncer_tls_include.ini.template postgres/
pgbouncer_tls_include.ini
cp postgres/postgresql_ssl_include.conf.template postgres/
postgresql_ssl_include.conf
```

4. Раскомментировать и отредактировать следующие строчки в файле конфигурации `postgres/pgbouncer_tls_include.ini`

```
client_tls_sslmode = require
client_tls_ca_file = /usr/local/share/ca-certificates/
pgbouncer_server.crt
client_tls_key_file = /usr/local/share/ca-certificates/
pgbouncer_server.key
client_tls_cert_file = /usr/local/share/ca-certificates/
pgbouncer_server.crt
server_tls_sslmode = require
server_tls_ca_file = /usr/local/share/ca-certificates/
postgresql_server.crt
```

5. Раскомментировать и отредактировать следующие строчки в файле конфигурации `postgres/postgresql_ssl_include.conf`

```
ssl = on
ssl_cert_file = '/usr/local/share/ca-certificates/
postgresql_server.crt'
ssl_key_file = '/usr/local/share/ca-certificates/
postgresql_server.key'
```

#Пути анализа и исключения

#Значения по умолчанию

В анализе сконфигурированы исключения для путей, по которым **не производится** поиск манифестов, файлов и не происходит анализ

качества. По умолчанию в исключения добавлены следующие значения (формат выражений — glob):

- `**/.git*`
- `**/.git/**`
- `**/fixtures/**`
- `**/tests/**`
- `**/doc/**`
- `**/docs/**`
- `**/samples/**`

#Добавление исключений

Чтобы добавить в список свои значения, в файле `app.env` в переменную `ANALYSIS_IGNORED_PATHS` необходимо **добавить** значения в формате:

- `**/ignoring_prj_1/**` - для исключения из анализа директории `ignoring_prj_1`;
- `**/ignoring_projects_*` - для исключения из анализа директорий у которых в названии присутствует `ignoring_projects_`;
- `**/ignoring_file.pom` - для исключения из анализа файла `ignoring_file.pom`.

Важно: не рекомендуется удалять пути исключений, указанные в переменной по умолчанию.

Пути добавляются через `,`. Пример переменной с добавленным исключением `**/migrations/**`:

```
ANALYSIS_IGNORED_PATHS=**/.git*,**/.git/**,**/fixtures/**,**/
tests/**,**/doc/**,**/docs/**,**/samples/**,**/migrations/**
```

#Скрипты для управления платформой

#Порядок запуска скриптов

Скрипты запускаются в backend-сервисе платформы:

- для Docker Compose:

```
docker exec -it <backend service> ./manage.py runscript
<команда>
```

- для Helm:

```
kubectl exec -it <backend service> ./manage.py runscript <кома
нда>
```

Переменные, передаваемые в скрипт, являются строго позиционными и обозначаются ключом `--script-arg`, опциональные переменные указаны в квадратных скобках.

#Доступные команды

#update_cwes

Обновляет данные о CWE (Common Weakness Enumeration) и уязвимостях в базе данных инсталляции. Опциональным аргументом можно управлять датой, с которой будет осуществлено обновление, по умолчанию дата и офсет обновления будут взяты из кэша последнего успешного обновления.

Синтаксис

```
update_cwes [--script-arg="2025-12-31"]
```

#load_licenses

Обновляет данные о лицензиях в базе данных инсталляции. Опциональным аргументом можно управлять датой, с которой будет осуществлено обновление, по умолчанию дата и офсет обновления будут взяты из кэша последнего успешного обновления.

Синтаксис

```
load_licenses [--script-arg="2025-12-31"]
```

#set_new_secret_key

Данная команда устанавливает новое значение переменной окружения SECRET_KEY. Чтобы избежать проблем с кодировкой, алиасами или других неожиданных действий консоли, значение NEW_SECRET_KEY рекомендуется скопировать из вывода в терминале после окончания работы скрипта.

Синтаксис

```
set_new_secret_key --script-arg="NEW_SECRET_KEY" [--script-
arg="OLD_SECRET_KEY"]
```

Варианты использования

1. Изменение с явно указанного OLD_SECRET_KEY на явно указанный NEW_SECRET_KEY. Данное изменение требуется в ситуации, когда во время запуска платформы значение переменной окружения SECRET_KEY отличалось от OLD_SECRET_KEY.

```
./manage.py runscript set_new_secret_key --script-arg="NE
W_SECRET_KEY" --script-arg="OLD_SECRET_KEY"
```

Данная команда перешифрует все чувствительные поля с переданного OLD_SECRET_KEY на NEW_SECRET_KEY.

Важно

Если значение переменной SECRET_KEY на момент запуска скрипта не совпадало с NEW_SECRET_KEY, после успешного окончания скрипта необходимо изменить значение переменной SECRET_KEY на NEW_SECRET_KEY и перезапустить платформу.

2. Изменение на явно указанный NEW_SECRET_KEY без указания OLD_SECRET_KEY. Требуется в ситуации, когда OLD_SECRET_KEY совпадает с указанным в settings.SECRET_KEY.

```
./manage.py runscript set_new_secret_key --script-arg="NE
W_SECRET_KEY"
```

Данная команда перешифрует все чувствительные поля с ключа, установленного в settings.SECRET_KEY на NEW_SECRET_KEY.

После успешного окончания скрипта необходимо изменить значение переменной SECRET_KEY на NEW_SECRET_KEY и перезапустить платформу.

#Описание переменных

В этом разделе представлено описание переменных, необходимых для установки и настройки платформы, включая те, которые описаны в разделах Установка системы в Docker и Работа системы в Kubernetes.

Внесение изменений

Перед внесением любых изменений в стандартное значение переменных, необязательных к заполнению во время установки, проконсультируйтесь с технической поддержкой. Изменение стандартных значений может оказать влияние на производительность и эффективность работы системы.

Не используйте в параметрах символ

#, он может некорректно восприниматься системой при установке.

#env.template

Файл env.template содержит переменные, необходимые для запуска инсталляции CodeScoring с использованием Docker Compose. Он

определяет параметры, необходимые для корректного запуска инсталляции в контейнеризированной среде.

- **COMPOSE_PROJECT_NAME** - название проекта в Docker Compose, используемое как префикс для ресурсов, создаваемых Docker Compose;
- **CODESCORING_VERSION** - определяет версию CodeScoring для установки. Актуальную версию можно узнать в разделе Changelog;
- **SECRET_KEY** - случайная строка символов, используемая CodeScoring как секрет инсталляции;
- **DJANGO_CSRF_TRUSTED_ORIGINS** - список доменов для правильной работы CSRF защиты. Указание протокола является обязательным, например:

```
   DJANGO_CSRF_TRUSTED_ORIGINS=http://
localhost:18000,https://localhost:8081,https://
внешний_ip:8081
```

#Nginx

Для настройки домена системы CodeScoring использует Nginx.

- **NGINX_HOST** - хост, на котором будет доступна система;
- **NGINX_PORT** - порт, на котором будет доступна система;
- **SITE_SCHEME** - протокол передачи данных, по умолчанию `https`.

#PostgreSQL

Для управления, обработки и хранения данных CodeScoring использует `PostgreSQL`.

- **POSTGRES_DB** - название базы данных;
- **POSTGRES_USER** - имя пользователя. При использовании собственной базы необходимо убедиться, что пользователь имеет следующие права:
  `Superuser, Create role, Create DB, Replication, Bypass RLS`;
- **POSTGRES_PASSWORD** - пароль;
- **POSTGRES_HOST** - хост, на котором доступна база данных;
- **POSTGRES_PORT** - порт, на котором доступна база данных.

#Sentry

По умолчанию интеграция с Sentry отключена, **SENTRY_ENABLE** = `False`. При необходимости использования Sentry для отправки данных

об ошибках и других событий системы укажите **SENTRY_ENABLED** = True и заполните следующие переменные:

- **SENTRY_DSN** - идентификатор проекта внутри Sentry, в который будут отправляться данные;
- **SENTRY_ENVIRONMENT** - окружение к которому будут относится отправляемые данные;
- **SENTRY_RELEASE** - релиз с которым будут связаны отправляемые данные.

#app.env.template

Файл `app.env.template` содержит переменные окружения, необходимые для конфигурирования инсталляции CodeScoring. Он определяет настройки базы данных, веб-сервера и очередей, а также другие важные параметры системы.

- **PATH** - определяет путь к исполняемым файлам, файлам виртуального окружения (venv) и стандартным системным путям:

```
   PATH=/venv/bin:${PATH}:/jscpd/node_modules/jscpd/bin:/
sbin:/bin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/
sbin:/root/bin
```

- **ANALYSIS_IGNORED_PATHS** - содержит список путей, по которым не производится поиск манифестов, файлов и не происходит анализ качества. По умолчанию указаны следующие пути:

```
   **/.git*,**/.git/**,**/fixtures/**,**/tests/**,**/doc/
**,**/docs/**,**/samples/**
```

- Переменные, указывающие на расположение основных файлов инсталляции:
  - **HOME** - значение по умолчанию: `ipcs-backend`;
  - **BASE_DIR** - значение по умолчанию: `ipcs-backend`;
  - **AZURE_DEVOPS_CACHE_DIR** - значение по умолчанию: `ipcs-backend`.
- **INDEX_API_URL** - URL-адрес API для индекса, необходимого для обновления базы знаний, значение по умолчанию: `https://index.codescoring.ru`. Данный параметр не требует изменений;
- Устарело **INDEX_API_TIMEOUT** - устаревшая переменная для единого таймаута запросов к Index API. Используйте отдельные таймауты ниже;
- **INDEX_API_POOL_TIMEOUT** - время ожидания получения соединения из пула (секунды). Значение по умолчанию: `60`;

- **INDEX_API_CONNECT_TIMEOUT** - время установки соединения (секунды). Значение по умолчанию: `60`;
- **INDEX_API_WRITE_TIMEOUT** - время записи тела запроса (секунды). Значение по умолчанию: `60`;
- **INDEX_API_READ_TIMEOUT** - время чтения тела ответа (секунды). Значение по умолчанию: `60`;
- **INDEX_PROXY_URL** - указывает внутри сети докер на URL-адрес контейнера Index Proxy. Не нуждается в изменении. Значение по умолчанию: `http://index-proxy:8000`;
- **ALLOWED_HOSTS** - разрешенные хосты для Django. Определяет допустимые адреса, с которых Django может принимать запросы. При отсутствии хоста в списке, Django отклоняет входящие запросы. По умолчанию разрешены все хосты *;
- **DJANGO_CACHES_REDIS_URL** - адрес сервера Redis, используемого в качестве системы кэширования Django. Значение по умолчанию: `redis://redis:6379/1`;
- **JOHNNY_BIN** - путь к исполняемому файлу агента сканирования JOHNNY. Значение по умолчанию: `/agents/johnny-linux-amd64`;
- **HASHER_BIN** - путь к исполняемому файлу агента HASHER, отвечающего за хеширование файлов. Значение по умолчанию: `/agents/hasher-linux-amd64`;
- **ANALYSIS_ROOT** - корневая директория для клонирования репозиториев для анализа. Файлы копируются как временные и после анализа удаляются. Значение по умолчанию: `/analysis-root`;
- **MEDIA_ROOT** - рабочая директория для промежуточного хранения скачиваемых и загружаемых файлов (таких как отчёт, SBOM и т.д.). Значение по умолчанию: `/analysis-root`;
- **NODE_PATH** - директория расположения `node_modules`. Значение по умолчанию: `/jscpd/node_modules`;
- **REQUESTS_CA_BUNDLE** и **SSL_CERT_FILE** - указывают на расположение SSL-сертификатов, используемых инсталляцией для работы:

```
REQUESTS_CA_BUNDLE=/etc/ssl/certs/ca-certificates.crt
```

```
SSL_CERT_FILE=/etc/ssl/certs/ca-certificates.crt
```

- **USE_SMART_FILTERS** - отвечает за использование "умных фильтров" в Celery. Эти фильтры позволяют более гибко управлять порядком выполнения задач. Доступны значения `True` или `False`, значение по умолчанию: `True`;

- В случае необходимости использования прокси для предоставления доступа инсталляции в интернет или к системе контроля версий, раскомментируйте и заполните необходимые переменные, указав URL-адрес прокси с протоколом, а также имя пользователя и пароль (при необходимости), например:
    - **HTTP_PROXY**

      ```
      HTTP_PROXY=http://
      proxy_user:password@proxy_host:proxy_port
      ```

    - **HTTPS_PROXY**

      ```
      HTTPS_PROXY=http://
      proxy_user:password@proxy_host:proxy_port
      ```

    - **NO_PROXY**

      ```
      NO_PROXY=localhost,gitlab.domain.local
      ```

#HUEY

HUEY - сервис, отвечающий за очереди и задачи. Количество воркеров рекомендуется изменять, только после консультации со специалистами технической поддержки.

- **HUEY_REDIS_URL** - адрес для подключения к серверу Redis, используемому Huey. Значение по умолчанию: `redis://redis:6379/0`;
- **HUEY_WORKERS** - количество воркеров HUEY, разбирающих очереди `tasks-main`. Значение по умолчанию: 8;
- **HUEY_HIGH_PRIORITY_WORKERS** - количество воркеров HUEY для высокоприоритетных задач. Значение по умолчанию: 8;
- **HUEY_OSA_PACKAGE_SCAN_WORKERS** - количество воркеров HUEY для модуля сканирования пакетов OSA. Значение по умолчанию: 4;
- **HUEY_OSA_CONTAINER_IMAGE_SCAN_WORKERS** - количество воркеров HUEY для модуля сканирования контейнеров OSA. Значение по умолчанию: 4;
- **HUEY_POLICY_WORKERS** - количество воркеров HUEY для работы политик безопасности. Значение по умолчанию: 4;
- **HUEY_TQI_WORKERS** - количество воркеров HUEY для модуля TQI. Значение по умолчанию: 4;
- **HUEY_SECRETS_WORKERS** - количество воркеров HUEY для модуля поиска секретов. Значение по умолчанию: 4;
- **HUEY_MEDIA_WORKERS** - количество воркеров Huey, отвечающих за загрузку медиа. Значение по умолчанию: 4;
- **HUEY_SCA_EXTERNAL_SCAN_WORKERS** - количество воркеров отвечающих за очередь анализа, запущенного через агент Джонни (`tasks-sca-external-scan`). Значение по умолчанию: 4;

- **HUEY_WEBHOOKS_WORKERS** - стандартное количество воркеров HUEY, отвечающих за работу вебхуков. Значение по умолчанию: 2;
- **HUEY_WORKER_MAX_TASKS** - максимальное количество задач, которое может обработать один воркер HUEY. Значение по умолчанию: `500`.

#Redis

Redis - хранилище данных, предназначенное для кэширования и управления данными в реальном времени. Переменные, определяющие параметры конфигурации для взаимодействия с сервером Redis:

- **REDIS_BACKOFF_CAP** - максимальное количество повторных попыток соединения при возникновении ошибок. Если после этого количества попыток соединение не установлено, будет возвращена ошибка. Значение по умолчанию: 5;
- **REDIS_BACKOFF_BASE** - базовое время задержки (в секундах) между повторными попытками при возникновении ошибок. Значение по умолчанию: `0.08`;
- **REDIS_RETRIES** - общее количество попыток подключения к серверу `Redis`. Значение по умолчанию: 5;
- **REDIS_SOCKET_CONNECT_TIMEOUT** - время ожидания (в секундах) при выполнении операций с сервером Redis. Если операция не завершится в течение этого времени, будет выдана ошибка. Значение по умолчанию: `5.0`;
- **REDIS_SOCKET_TIMEOUT** - время ожидания (в секундах) при выполнении операций с сервером Redis. Если соединение не установлено в течение этого времени, будет осуществлена повторная попытка или выдана ошибка. Значение по умолчанию: `5.0`;
- **REDIS_SOCKET_KEEPALIVE** - включает механизм поддержки активного соединения с сервером Redis, чтобы избежать его разрыва из-за бездействия. Доступны значения `True` или `False`, значение по умолчанию: `True`;
- **TASK_RESULT_EXPIRATION_PERIOD** - указывает, как долго (в секундах) результаты выполнения задач будут храниться в `Redis`. После истечения этого времени они будут удалены. Значение по умолчанию: `14400`.

#Celery

Celery - сервис, отвечающий за управления асинхронными задачами, который позволяет разгрузить основные процессы приложения и повысить его отзывчивость. Переменные, отвечающие за настройку воркеров Celery:

- **CELERY_WORKER_CONCURRENCY** - количество задач, которые может обрабатывать один воркер Celery одновременно, значение по умолчанию: 6;
- **CELERY_WORKER_MAX_CONCURRENCY** - максимальное количество задач, которые может обрабатывать один воркер Celery. Этот

параметр используется для предотвращения перегрузки системы при большом количестве задач, значение по умолчанию: 12;
- **CELERY_MEDIA_WORKER_CONCURRENCY** - количество задач, которые может обрабатывать воркер Celery, отвечающий за обработку медиа-задач (например, генерация отчётов), значение по умолчанию: 2;
- **CELERY_MEDIA_WORKER_MAX_CONCURRENCY** - максимальное количество задач, которые может обрабатывать воркер Celery, отвечающий за обработку медиа-задач. Этот параметр используется для предотвращения перегрузки системы при большом количестве задач, значение по умолчанию: 4;

#Архивирование данных CodeScoring OSA

Следующие переменные отвечают за настройку архивирования данных сервиса системы CodeScoring - OSA. По умолчанию переменные `OSA_ARCHIVE_THRESHOLD_DAYS, OSA_ARCHIVE_AUTO_CLEANUP_ENABLED, OSA_ARCHIVE_RETENTION_PERIOD_DAYS, OSA_ARCHIVE_CHUNK_SIZE` закомментированы, при необходимости их использования, необходимо их расскомментировать.

- **OSA_ARCHIVE_THRESHOLD_DAYS** - период (в днях), после которого пакет или контейнерный образ, не получивший запросов, архивируется. Значение по умолчанию: 14;
- **OSA_ARCHIVE_RETENTION_PERIOD_DAYS** - период (в днях) указывающий, сколько дней архивные пакеты или контейнерные образы будут храниться перед тем, как быть удаленными, при включенной автоматической очистке `OSA_ARCHIVE_AUTO_CLEANUP_ENABLED`. Значение по умолчанию: `30`;
- **OSA_ARCHIVE_AUTO_CLEANUP_ENABLED** - включение или отключение автоматической очистки архивных компонентов OSA. Если выставлено значение `True`, то система будет автоматически удалять архивные пакеты после периода, указанного в `OSA_ARCHIVE_RETENTION_PERIOD_DAYS`, если `False` - очистка выполняется вручную, значение по умолчанию: `False`;
- **OSA_ARCHIVE_CHUNK_SIZE** - определяет размер "порций" (чанков) для обработки архивных компонентов. Этот параметр может влиять на производительность и эффективность процесса архивирования/очистки. Значение по умолчанию: `1000`.

#Judge

Следующие переменные отвечают за настройку компонента Judge, отвечающего за работу политик в рамках работы сервиса OSA Proxy. Их необходимо настраивать только в случае использования OSA Proxy:

- **WEB_CONCURRENCY** - определяет количество воркеров для параллельной проверки компонентов или контейнеров на соответствие политикам безопасности. Значение по умолчанию: 5;

- **MIN_DATABASE_CONNECTION_POOL_SIZE** - определяет минимальный пул активных соединений к базе данных. По умолчанию: переменная закомментирована и имеет значение `0`;
- **MAX_DATABASE_CONNECTION_POOL_SIZE** - определяет максимальный пул активных соединений к базе данных. По умолчанию: переменная закомментирована и имеет значение `10`.

#Описание служб

В данном разделе представлен обзор основных служб, используемых для работы системы.

#Основные компоненты

- **Frontend** – пользовательский интерфейс, обеспечивающий взаимодействие с системой. Отображает результаты анализа, настройки и параметры политик.
- **Backend** – обрабатывает запросы от Frontend и координирует работу других компонентов системы;
- **Celery-worker** – выполняет задачи, полученные от различных сервисов (например, Celery-beat и Backend);
- **Celery-beat** – планировщик задач, запускающий задачи в Celery-worker по заданному расписанию;
- **Collectstatic** – отвечает за сборку статических файлов, необходимых для работы системы;
- **Fluentd** – агент сбора логов, собирающий логи из всех компонентов системы;
- **Index-proxy** – обогащает запросы к индексу (https://index.codescoring.ru) дополнительными данными и заголовками;
- **Migrate** – выполняет конфигурацию системы при запуске и обновление базы данных;
- **Osa-api** – обеспечивает взаимодействие между основной системой и плагином OSA (установленным в Sonartype Nexus Repository или Jfrog Artifactory) через API;
- **Osa-registration** – отвечает за сканирование и проверку пакетов и контейнерных образов на соответствие политикам;
- **Pgbouncer** – обеспечивает управление и оптимизацию запросов к базе данных;
- **Psql** – клиент PostgreSQL, используемый для управления базой данных;
- **Redis** – кэш и брокер сообщений, обеспечивающий хранение данных и координацию работы компонентов;
- **Tasks-high-priority** – очередь для выполнения задач с высоким приоритетом;
- **Tasks-main** – очередь для выполнения основных задач, таких как сканирование, загрузка проектов и отправка политик;
- **Tasks-main-scheduler** – планировщик периодических задач, например сканирование по расписанию;
- **Tasks-media** – очередь для загрузки и выгрузки медиафайлов;

- **Tasks-osa-container-image-scan** – отвечает за сканирование образов в стандартах Docker, OCI и Singularity на предмет уязвимостей;
- **Tasks-osa-package-scan** – отвечает за сканирование пакетов на наличие уязвимостей;
- **Tasks-policy** – выполняет задачи по проверке соответствия политикам безопасности;
- **Tasks-sca-external-scan** – очередь для сканирований, запущенных через консольного агента Johnny;
- **Tasks-tqi** – агент модуля CodeScoring TQI (Teams and Quality Intelligence), анализирующий качество исходного кода;
- **Tasks-secrets** – агент модуля CodeScoring Secrets, отвечающий за поиск чувствительной информации в коде;
- **Judge** – отвечает за работу политик в рамках работы сервиса OSA Proxy.

#Диагностика неполадок

#Работа с логами

1. Перейти в директорию с файлами запуска:

   ```
   cd /path/to/docker/compose
   ```

2. Выполнить команду копирования файла логов из контейнера в файл `codescoring_onprem.log`

   ```
   docker cp -L PROJECT_NAME-fluentd-1:/fluentd/log/
   docker.log codescoring_onprem.log
   ```

3. Отправить вендору файл `codescoring_onprem.log`.

#Переустановка системы

При необходимости начать процесс установки с нуля нужно выполнить очистку томов.

Если на сервере с докером нет других контейнеров, кроме проекта CodeScoring, выполнить команду:

```
docker system prune --all --volumes
```

Если на сервере есть ещё другие проекты на docker:

1. остановить docker compose:

```
docker compose down --remove-orphans
```

2. выполнить команду:

```
docker volume rm PROJECT_NAME__db-data
```

3. Если возникнет ошибка, что данный том используется контейнером, следует выполнить команду и повторить предыдущие шаги (CT_HASH будет в сообщении об ошибке):

```
docker rm CT_HASH
```

Если проблема не решается, обратиться к контактному лицу вендора, оказывающему сопровождение, для получения дальнейших инструкций.

#Активация системы

#Ввод ключа активации

Для работы системы необходимо её активировать с помощью ключа, который передается клиенту в отдельном `txt` файле.

Для ввода ключа необходимо перейти в раздел `Настройки -> Ключ активации`, скопировать текст из файла в поле без каких-либо изменений и нажать кнопку **Сохранить**. В случае успеха на странице появится информация по ключу и поле Статус перейдет в значение **Активен**.

#Параметры ключа

При успешной активации системы в разделе `Настройки -> Ключ активации` отображаются параметры используемого ключа:

- **Статус** – статус активации системы;
- **Владелец** – наименование организации, на чье имя выдан ключ;
- **Дата выпуска** – день выпуска ключа;
- **Дата истечения** день истечения действия ключа;
- **Ограничение по количеству авторов** – максимальное количество авторов (разработчиков), на которое лицензирована система;
- **Частные базы уязвимостей** – список подключенных частных баз уявимостей, например **Kaspersky OSS Threats Data Feed**;
- **Доступные модули** – список подключенных модулей с отображением даты истечения ключа для каждого.

#Истечение ключа активации

При истечении срока действия ключа активации платформа продолжает работать с ограничениями:

- Ранее выполненные сканирования и их результаты остаются доступными для просмотра;
- Попытка выполнения нового сканирования приводит к ошибке, связанной с истечением ключа активации.

#Аудит количества авторов

Функция аудита количества авторов доступна в платформе, начиная с версии **2025.37.0**. Она позволяет получить статистику об уникальных авторах коммитов в выбранных репозиториях.

Совершить расчет можно в разделе `Настройки → Ключ активации` по кнопке **Рассчитать**.

#Поддерживаемые платформы

На данный момент аудит доступен для платформы GitLab.

#Настройка параметров

После нажатия кнопки открывается форма с параметрами анализа:

- **Провайдер** — источник данных о репозиториях. На текущий момент поддерживается **GitLab**;
- **VCS URL** — адрес хоста, на котором размещены репозитории (без указания конкретного репозитория). Пример:

  ```
  https://gitlab.svc.cdscrng.ru/
  ```

- **Ключ доступа** — персональный токен с правами `read_api`. Токен не сохраняется и используется только для текущего расчёта. В GitLab токен можно получить по адресу:

  ```
  <VCS-URL>/-/user_settings/personal_access_tokens
  ```

- **Фильтр по email авторов** — список доменов или регулярных выражений для отбора авторов. Можно указать несколько значений через `|`.

Важно

VCS URL должен начинаться с протокола передачи данных. Закрывающий слэш необязателен.

#Выполнение анализа

После заполнения формы нажмите **Рассчитать**. Пока идёт сканирование, процесс можно отменить нажатием кнопки **Отменить**.

#Результаты расчёта

После завершения аудита в разделе ключей активации отображаются следующие данные:

- **Статус расчёта** – показывает результат выполнения. При ошибке причину можно посмотреть в разделе **Аудит-лог**;
- **Время расчёта** – дата и время выполнения аудита;
- **Уникальных авторов за год** – количество авторов коммитов за последние 365 дней. Через API параметр можно изменить;
- **Успешно обработанных репозиториев** – количество репозиториев, обработанных без ошибок;
- **Репозиториев с ошибкой** – количество репозиториев, где возникли ошибки чтения. Подробности можно увидеть в логах бэкенда;
- **URL отчёта (JSON)** – ссылка на полный отчёт в формате JSON с деталями анализа и списком обработанных репозиториев.

#Диагностика ошибок

Если анализ завершился с ошибкой:

1. Проверьте **VCS URL** и корректность **токена**. При ошибке **401** обычно указаны неверный адрес или недействительный токен.
2. Дополнительную информацию можно найти в разделе **Аудит-лог**.

#Управление учетными записями

#Создание учетных записей

Платформа CodeScoring поддерживает работу множества пользователей с отдельными учетными записями. Создание и управление учетными записями пользователей происходит в разделе `Настройки -> Пользователи`.

Для создания нового пользователя необходимо перейти на форму по кнопке **Create New** и заполнить следующие поля:

- **Имя пользователя** — имя пользователя в системе;
- **Имя** — имя;
- **Фамилия** — фамилия;
- **Email** — электронная почта;
- **Подразделение** — принадлежность к подразделению организации в рамках системы;
- **Уровень доступа** — уровень доступа в рамках системы;

- **Пароль** – пароль для входа в систему;
- **Может создавать CLI проекты через API** – возможность создавать проекты типа CLI при использовании API.

Список созданных пользователей можно отфильтровать по следующим параметрам:

- **Подразделение**;
- **Уровень доступа**;
- **Активен** — признак действующей учетной записи;
- **Сервер LDAP** — сервер LDAP, подключенный к системе.

#Настройка учетных записей

Созданные учетные записи можно отредактировать или удалить в разделе Настройки -> Пользователи. Добавить пользователя в проект с указанной ролью можно по кнопке **Добавить пользователи** на вкладке "Проекты" страницы редактирования пользователя.

Время сессии для неактивного пользователя ограничено. По умолчанию сессия пользователя заканчивается через 2 недели с момента последней активности, после чего нужно произвести повторный вход в систему.

Для конфигурации времени жизни сессии доступна переменная окружения (в секундах): `SESSION_COOKIE_AGE`.

Важно

Изменение имени УЗ и пароля для пользователей из внешних провайдеров идентичности невозможно.

#Разделение уровней доступа

При создании учетной записи ей должен быть присвоен один из следующих уровней доступа - **User** (пользователь), **Administrator** (администратор), **Auditor** (аудитор ИБ) или **Security Manager** (Менеджер безопасности).

#Уровень доступа Administrator

Для уровня доступа Administrator предоставляется доступ во все проекты. Данный уровень доступа также дает возможность просматривать и изменять все настройки в системе без ограничений.

#Уровень доступа Security Manager

Для уровня доступа Security Manager предоставляется доступ во все проекты. Данный уровень доступа также дает возможность просматривать все разделы, связанные с безопасностью, и запускать все виды сканирования, но изменять - только политики.

#Уровень доступа Auditor

Для уровня доступа Auditor предоставляется доступ во все проекты. Данный уровень доступа также дает право просмотра всех настроек и проектов в системе без возможности вносить и сохранять изменения.

#Уровень доступа User

Для уровня доступа User доступ организуется индивидуально. Для каждого проекта может быть предоставлен доступ со следующими ролями:

- **Viewer** — доступ только на просмотр результатов анализов в рамках проекта;
- **Developer** — все права роли Viewer, а также доступ к запуску анализа в веб-интерфейсе, через агента и через плагин прокси-репозитория;
- **Owner** — все права роли Developer, доступ к изменению настроек проекта и управлению доступами других пользователей проекта.

Для каждой роли в рамках уровня доступа **User** доступно создание CLI проектов через API при активации параметра **Может создавать CLI проекты через API** в профиле пользователя.

В проекте может быть несколько пользователей с одинаковыми ролями, в том числе несколько **Owner**. При отсутствии пользователей в роли **Owner** проектом может управлять только пользователь с уровнем доступа **Administrator**.

#Доступные действия

Более подробное перечисление доступных действий для каждого уровня доступа представлено в таблице ниже:

Действие	User (Viewer)	User (Developer)	User (Owner)	Auditor
Analysis: просмотр результатов анализа	✓	✓	✓	✓
Analysis: изменение статуса анализа	✗	✓	✓	✗
Analysis: запуск SCA анализа	✗	✓	✓	✗
Analysis: запуск анализа Secrets	✗	✓	✓	✗

Действие	User (Viewer)	User (Developer)	User (Owner)	Auditor
Analysis: изменение статуса Secrets	✗	✗	✓	✗
Analysis: запуск Authors анализа	✗	✗	✗	✗
Analysis: запуск Quality анализа	✗	✗	✗	✗
Activation key: просмотр информации об активационном ключе	✗	✗	✗	✓
Activation key: сохранение активационного ключа	✗	✗	✗	✗
Audit log: просмотр аудит лога	✗	✗	✗	✓
Audit log: экспорт аудит лога	✗	✗	✗	✓
Authors merge: просмотр правил	✗	✗	✗	✓
Authors merge: создание правил	✗	✗	✗	✗
Dashboard: просмотр страницы	✓	✓	✓	✓
Dependencies: просмотр списка зависимостей	✓	✓	✓	✓
Dependencies: экспорт списка зависимостей	✓	✓	✓	✓
	✗	✗	✗	✓

Действие	User (Viewer)	User (Developer)	User (Owner)	Audito
Email: просмотр настроек почты				
Email: редактирование настроек почты	✘	✘	✘	✘
Groups: просмотр групп пользователей	✘	✘	✘	✔
Groups: создание групп пользователей	✘	✘	✘	✘
Groups: редактирование групп пользователей	✘	✘	✘	✘
Groups: удаление групп пользователей	✘	✘	✘	✘
LDAP: просмотр настроек LDAP	✘	✘	✘	✔
LDAP: редактирование настроек LDAP	✘	✘	✘	✘
OSS Index: просмотр настроек OSS Index	✘	✘	✘	✔
OSS Index: редактирование настроек OSS Index	✘	✘	✘	✘
Policies: просмотр политик	✔	✔	✔	✔
Policies: создание политик	✘	✘	✘	✘

Действие	User (Viewer)	User (Developer)	User (Owner)	Auditor
Policies: редактирование настроек политик	✗	✗	✗	✗
Policies: удаление политик	✗	✗	✗	✗
Policy alerts: просмотр списка алертов	✓	✓	✓	✓
Policy alerts: экспорт списка алертов	✓	✓	✓	✓
Policy alerts: отправка уведомлений	✓	✓	✓	✓
Policy ignores: просмотр правил	✓	✓	✓	✓
Policy ignores: создание правил	✗	✗	✗	✗
Policy ignores: редактирование правил	✗	✗	✗	✗
Policy ignores: удаление правил	✗	✗	✗	✗
Projects: просмотр проектов	✓	✓	✓	✓
Projects: просмотр Contribution map	✓	✓	✓	✓
Projects: просмотр Complexity map	✗	✗	✗	✓
	□	□	□	✗

Действие	User (Viewer)	User (Developer)	User (Owner)	Auditor
Projects: создание проектов				
Projects: редактирование настроек проектов	✗	✗	✗	✗
Projects: удаление проектов	✗	✗	✗	✗
Projects: управление правами доступа групп для проектов	✗	✗	✓	✗
Projects: управление правами доступа пользователей для проектов	✗	✗	✓	✗
Projects: загрузка SBOM	✗	✓	✓	✗
Projects: редактирование зависимостей для экспорта SBOM	✗	✓	✓	✗
Project categories: просмотр категорий	✗	✗	✓	✓
Project categories: создание категорий	✗	✗	✗	✗
Project categories: редактирование категорий	✗	✗	✗	✗

Действие	User (Viewer)	User (Developer)	User (Owner)	Auditor
Project categories: удаление категорий	✖	✖	✖	✖
Proprietors: просмотр владельцев кода	✖	✖	✔	✔
Proprietors: создание владельцев кода	✖	✖	✖	✖
Proprietors: редактирование владельцев кода	✖	✖	✖	✖
Proprietors: удаление владельцев кода	✖	✖	✖	✖
Task managers: просмотр интеграций	✖	✖	✖	✔
Task managers: добавление интеграций	✖	✖	✖	✖
Task managers: редактирование настроек интеграций	✖	✖	✖	✖
Task managers: удаление интеграций	✖	✖	✖	✖
Task managers: выполнение проверки настроек	✖	✖	✖	✔
Users: просмотр пользователей	✖	✖	✖	✔
Users: создание пользователей	✖	✖	✖	✖

Действие	User (Viewer)	User (Developer)	User (Owner)	Auditor
Users: редактирование настроек пользователей	✗	✗	✗	✗
Users: удаление пользователей	✗	✗	✗	✗
VCS: просмотр репозиториев	✗	✗	✗	✓
VCS: добавление репозиториев	✗	✗	✗	✗
VCS: редактирование настроек репозиториев	✗	✗	✗	✗
VCS: удаление репозиториев	✗	✗	✗	✗
VCS: выполнение проверки настроек	✗	✗	✗	✓
Vulnerabilities: просмотр списка уязвимостей	✓	✓	✓	✓
Vulnerabilities: экспорт списка уязвимостей	✓	✓	✓	✓

Важно

Для возможности запуска сканирования также убедитесь, что в лицензии включен соответствующий модуль анализа.

#Группы пользователей

Пользователи внутри системы могут быть распределены в группы. Создание и управление группами происходит в разделе `Настройки -> Группы`.

Для создания новой группы пользователей необходимо перейти на форму по кнопке **Создать** и заполнить следующие поля:

- **Название** — название группы;
- **Описание** — описание группы.

Группы могут быть добавлены к созданным проектам для более удобного отслеживания пользователей, связанных с проектом.

#Создание групп

Группы используются для объединения пользователей в списки и привязки к определенным проектам. Это позволяет удобно управлять массовым доступом к проектам в системе.

Создание групп происходит в разделе Настройки -> Группы. Перейти на форму создания группы можно по кнопке **Создать**. В форме необходимо обязательно заполнить поле названия **Название** и опционально задать описание группы, цвет иконки.

#Настройка групп

Каждая группа имеет свой список пользователей и набор привязанных проектов. Изменить данные параметры можно по кнопке **Редактировать**.

Привязать проект к группе можно по кнопке **Добавить проект**. После привязки группы каждый ее пользователь будет иметь доступ к проекту согласно назначенной роли.

При добавлении нового пользователя в группу по кнопке **Добавить пользователя** необходимо выбрать существующую учетную запись в системе CodeScoring и назначить одну из ролей в рамках группы:

- **Наблюдатель** — доступ только на просмотр результатов анализов проекта;
- **Разработчик** — доступ к запуску анализа в веб-интерфейсе, через агента и через плагин прокси-репозитория;
- **Владелец** — доступ к просмотру политик проекта, изменению настроек проекта и управлению доступами других пользователей проекта.

В рамках системы также имеется возможность настроить маппинг групп LDAP на группы и роли CodeScoring.

#Работа с LDAP

#Возможности интеграции с LDAP

CodeScoring поддерживает аутентификацию и авторизацию пользователей по протоколу **LDAP** и маппинг атрибутов записей о пользователях в **LDAP** на атрибуты пользователей в системе.

#Страница аутентификации CodeScoring

На странице аутентификации доступно меню с выбором провайдера аутентификации. Помимо провайдера по умолчанию (локальные учётные записи, `internal directory`), доступны для выбора активные интеграции с **LDAP** серверами.

Login provider

Internal directory

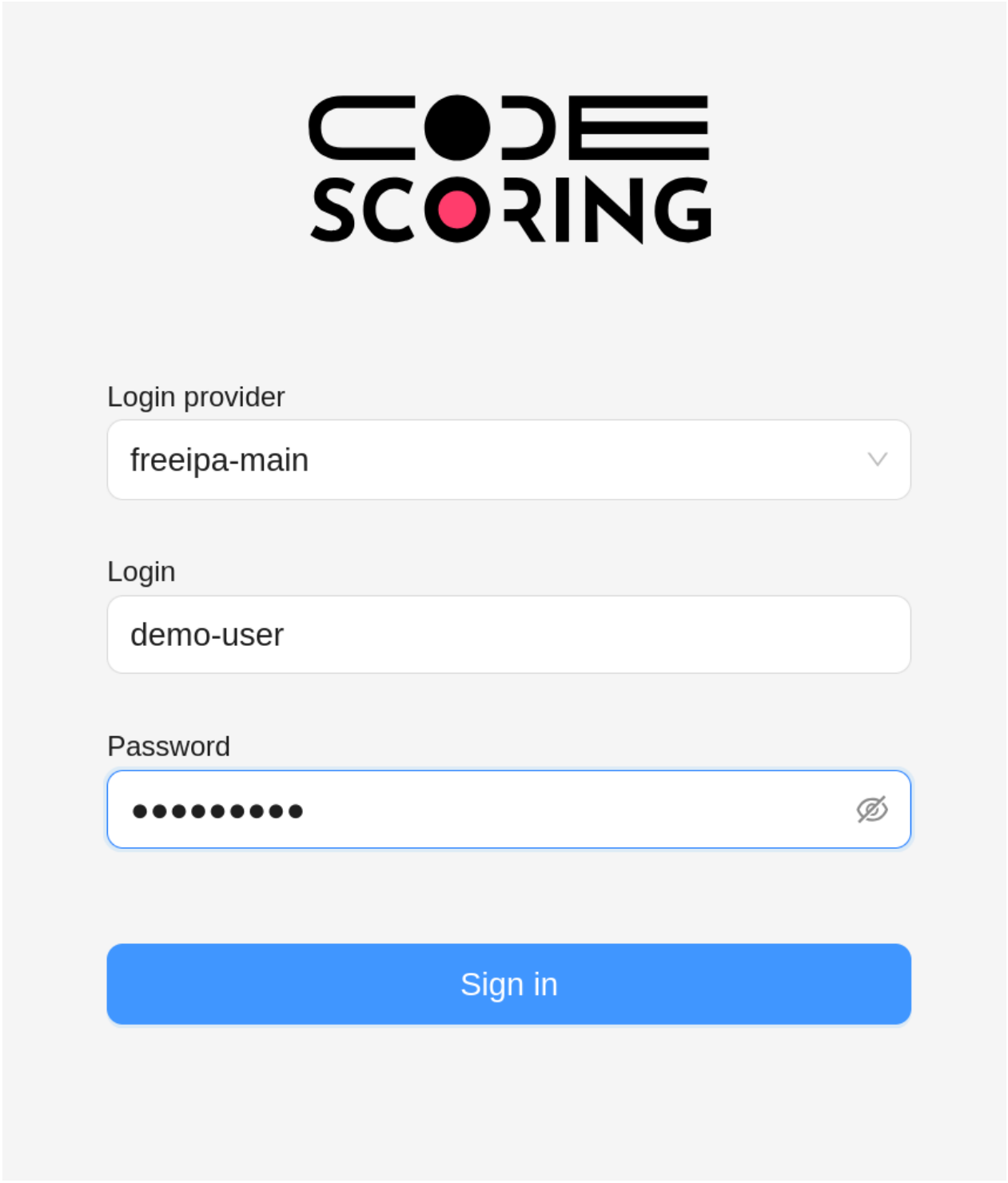
Internal directory internal

freeipa-main ldap

keycloak-main oidc

Password

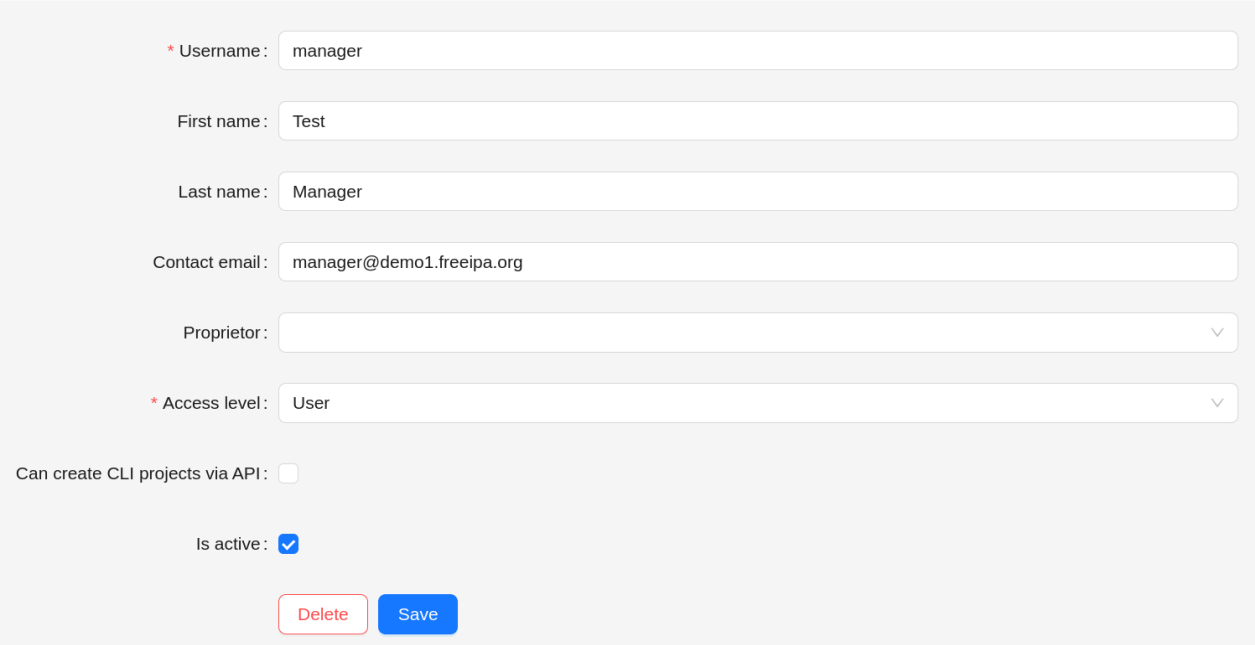
Sign in



#Маппинг атрибутов записей о пользователях LDAP на атрибуты пользователей CodeScoring

При аутентификации через **LDAP** происходит маппинг следующих данных из записи в директории на учётную запись в CodeScoring:

- название УЗ (`username`);
- имя;
- фамилия;
- электронная почта.



#Маппинг групп LDAP на группы и роли CodeScoring

При аутентификации через LDAP CodeScoring может запрашивать данные LDAP-групп пользователя и производить их маппинг на свои уровни доступа, группы и роли. Для активации этой возможности в настройках LDAP необходимо заполнить следующие поля:

- `База для поиска групп`.
- `Фильтр поиска для всех групп`
- `Поле с названием группы`
- `Поле с группами пользователя` (если значение поля `Метод поиска по группе` равно "Пользовательская запись")
- `Фильтр поиска групп пользователя` (если значение поля `Метод поиска по группе` равно "Поиск по группе")

CodeScoring производит маппинг согласно правилам, управление которыми доступно в разделе
`Настройки -> Провайдеры идентификации -> Группы LDAP`.

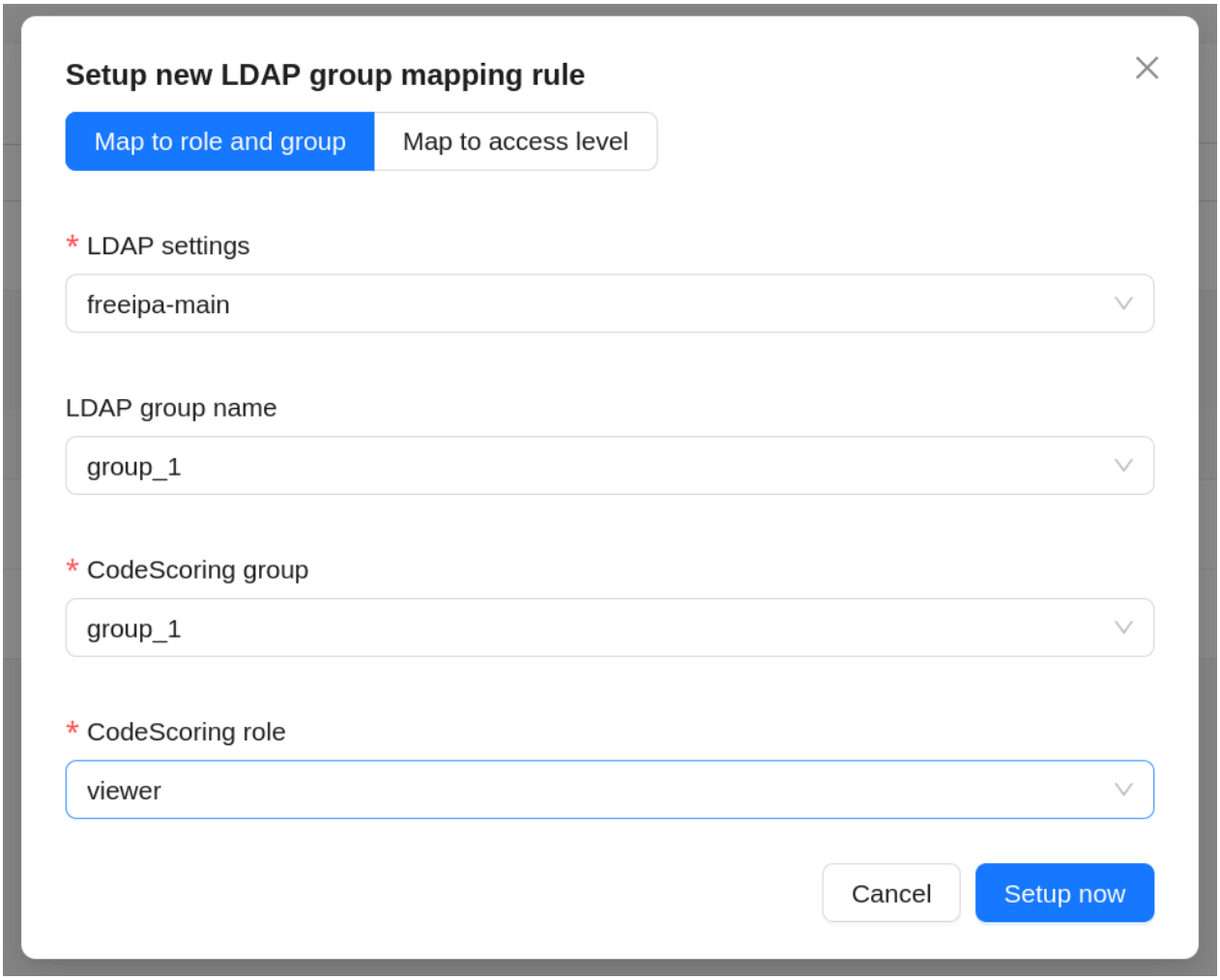
Маппинг запускается только в двух случаях:

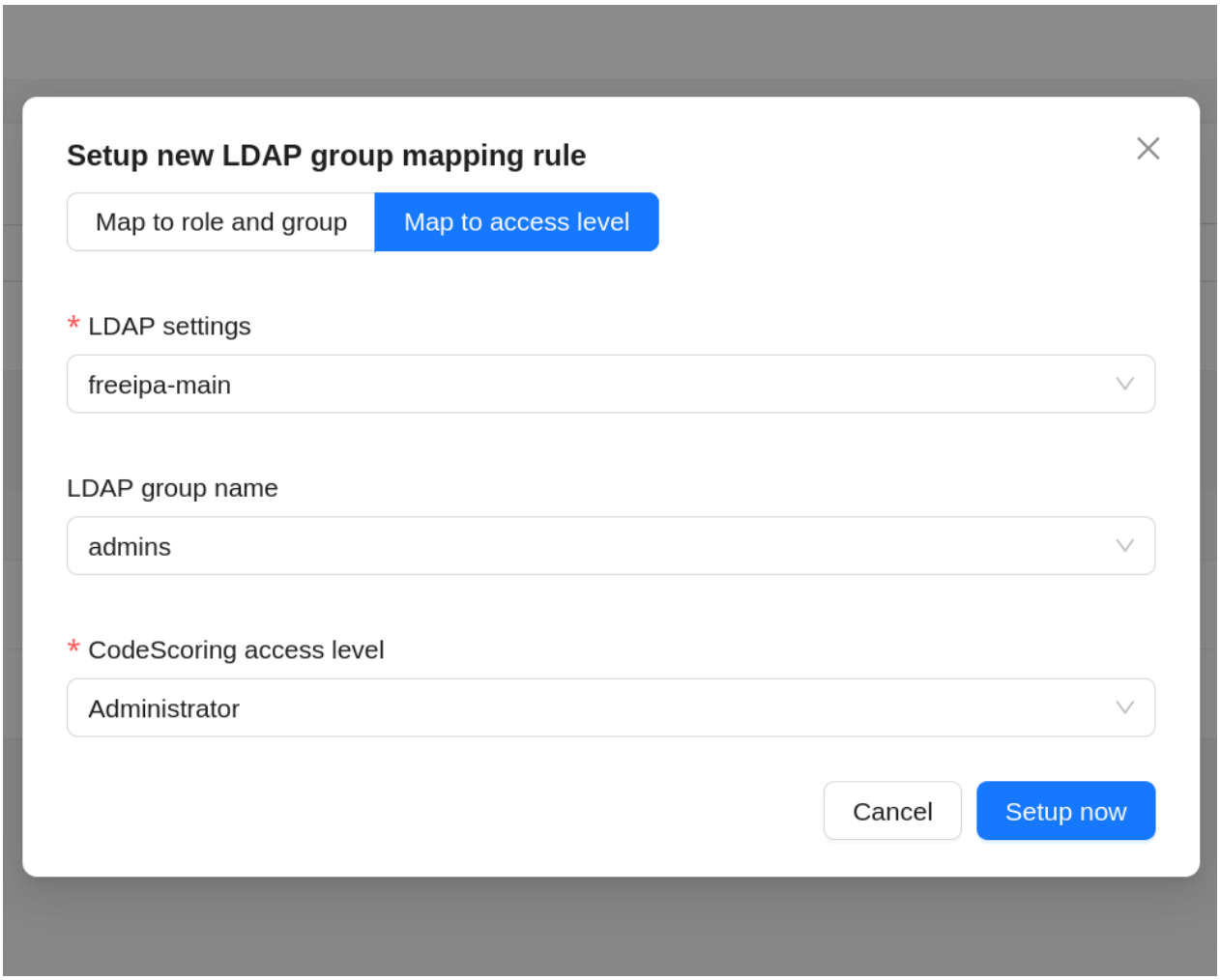
- в момент входа пользователя в систему;
- при нажатии кнопки `Применить все` правила в разделе `Настройки -> Провайдеры идентификации -> Группы LDAP`.

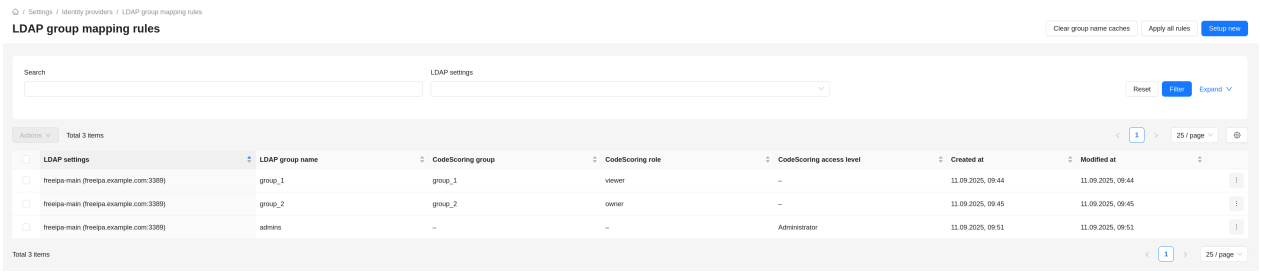
Правила маппинга применяются следующим образом:

- в процессе маппинга обновляется состав групп пользователя;
- добавление, изменение и удаление правил маппинга не приведёт к обновлению групп пользователя. Для применения изменённых правил, необходимо инициировать процесс маппинга одним из двух способов, указанных выше;

- группы пользователя, добавленные или отредактированные администратором в ручном режиме, не изменяются в процессе маппинга, то есть приоритет отдаётся ручным изменениям;
- если при поиске LDAP-групп после успешной аутентификации в LDAP произошла ошибка, группы пользователей, применённые согласно правилам маппинга, будут удалены.

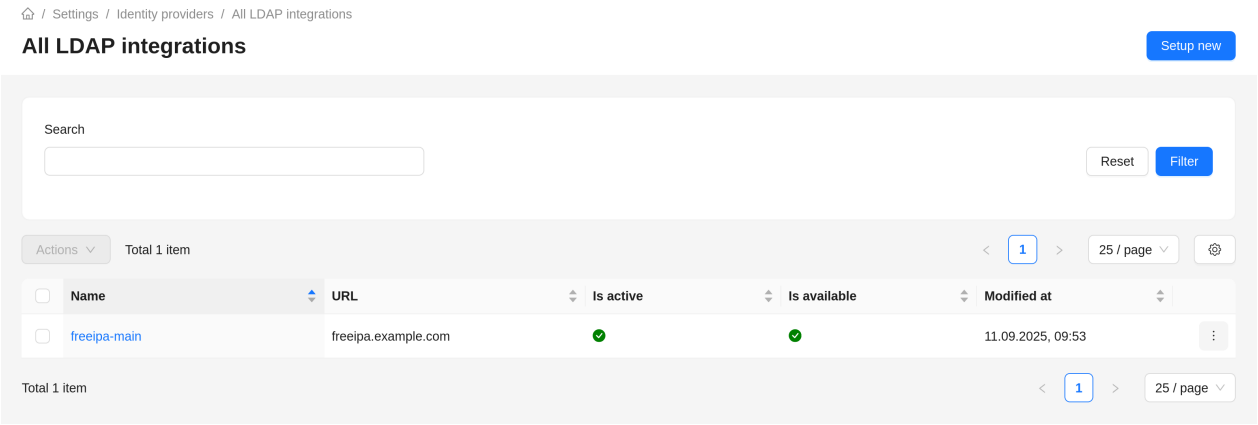






#Просмотр существующих интеграций с LDAP

Просмотр существующих интеграций доступен в разделе `Настройки -> Провайдеры идентификации -> LDAP`. В разделе отображаются таблица со списком настроенных интеграций LDAP, кнопка для создания новой интеграции (`Добавить`) и окно поиска.



#Просмотр деталей о существующей интеграции с LDAP

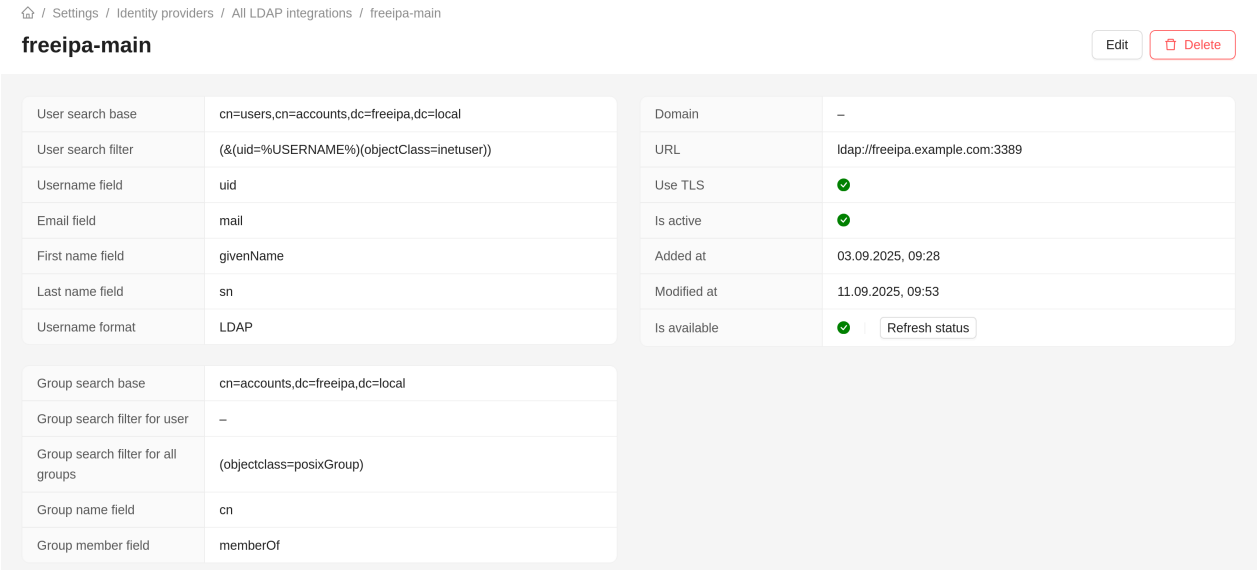
Просмотр деталей открывается при нажатии на гиперссылку с названием интеграции либо при нажатии на кнопку **View** в разделе Actions.

При просмотре доступны следующие действия:

- удаление интеграции;
- редактирование интеграции;
- проверка доступности (0бновить статус).

Помимо основных полей настроек (описаны ниже), при просмотре деталей об интеграции с **LDAP** доступны данные о:

- дате создания;
- дате последнего обновления;
- статусе доступности;
- (опционально) причине недоступности.



#Создание или редактирование интеграции с LDAP

Для того чтобы создать новую интеграцию с **LDAP**, необходимо на просмотре списка интеграций нажать на кнопку Добавить. Для того чтобы отредактировать существующую интеграцию, необходимо нажать на кнопку Редактировать на странице просмотра или списка интеграций. Формы создания и редактирования идентичны.

#Описание полей формы

- Название — название интеграции, отображается на странице аутентификации;
- LDAP протокол — выбор между протоколами **LDAP** и **LDAPS** (**LDAP** поверх **SSL**);
- Имя хоста LDAP — доменное имя или IP адрес сервера **LDAP**;
- Порт LDAP — порт **LDAP** сервера;
- База для поиска пользователя — указание места поиска записей о пользователе;
- Фильтр поиска пользователя — фильтр для поиска записи о пользователе. Должен содержать шаблон %USERNAME%, при поиске он будет заменён на имя УЗ, для которой производится аутентификация через LDAP;
- Поле с именем учетной записи — атрибут хранящий название УЗ пользователя в записи о нём;
- Поле с email — атрибут хранящий email пользователя в записи о нём;
- Поле с именем — атрибут хранящий имя пользователя в записи о нём;
- Поле с фамилией — атрибут хранящий фамилию пользователя в записи о нём;
- Формат имени учетной записи — выбор формата строки авторизации;
- Домен — указание домена **Active Directory** (появляется при выборе соответствующего **Username format**);
- Сервисный пользователь — логин пользователя с правами чтения каталогов на сервере **LDAP**;
- Пароль — пароль пользователя с правами чтения каталогов на сервере **LDAP**;
- Поле с названием группы — атрибут хранящий название группы в записи о ней;
- База для поиска групп — указание места поиска записей о группах;
- Фильтр поиска групп пользователя — фильтр для поиска записей о группах, к которым принадлежит пользователь. Должен содержать шаблон %USERNAME%, при поиске он будет заменён на имя УЗ, для которой производится аутентификация через **LDAP**;
- Фильтр поиска для всех групп — фильтр для поиска записей о всех группах в **LDAP**;
- Поле с группами пользователя — атрибут хранящий идентификаторы членов группы в записи о ней;
- Использовать TLS;

- Активно — используется ли интеграция с **LDAP** для аутентификации пользователей.

* Name

freeipa-main

The value will be shown on the login page

* LDAP protocol

ldap

* LDAP hostname

freeipa.example.com

* LDAP port

3389

* User search base

cn=users,cn=accounts,dc=freeipa,dc=local

* User search filter

(&(uid=%USERNAME%)(objectClass=inetuser))

The field must contain the %USERNAME% placeholder

* Username field

uid

* Email field

mail

* First name field

givenName

* Last name field

sn

* Username format

LDAP

Service user

admin

Service user password

Edit

If specified, these credentials will be used to retrieve the attributes of the user that is logging in

Group name field

cn

Group search base

cn=accounts,dc=freeipa,dc=local

Group search filter for all groups

(objectclass=posixGroup)

User group search method

User record | Group search

Group member field

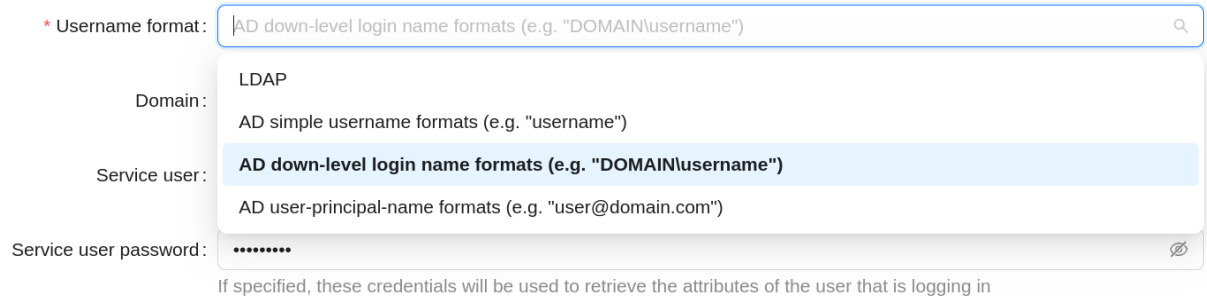
memberOf

Is active

Use TLS

Save Cancel

#Доступные опции для username format



#Тестирование конфигурации интеграции с LDAP

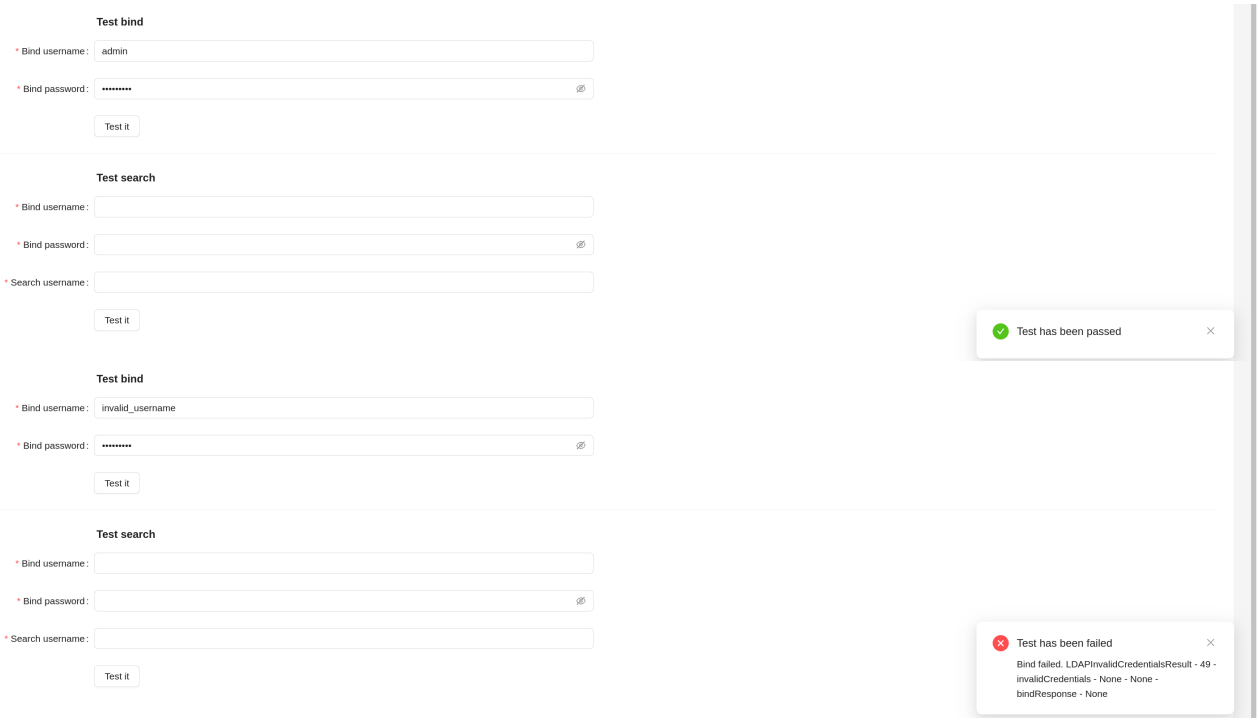
Для удобства конфигурации пользователям доступны 2 формы для тестирования подключения:

- тестирование подключения и аутентификации (`Тестирование подключения`);
- тестирование поиска (`Тест поиска пользователя`).

Для обоих тестов комбинируются данные из основной формы с данными формы тестирования. Данные из полей `Сервисный пользователь` и `Пароль сервисного пользователя` игнорируются.

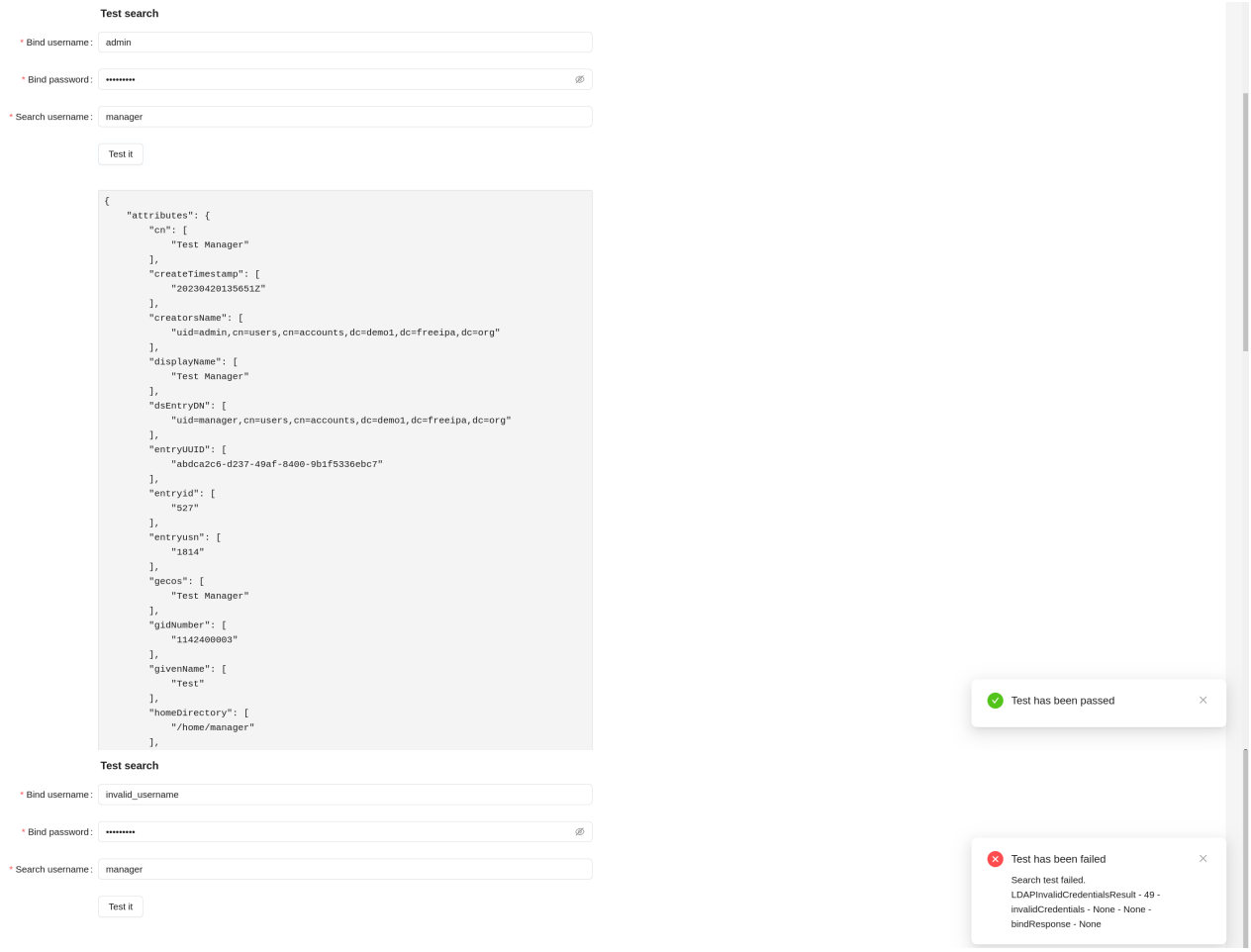
#Тестирование подключения и аутентификации

При нажатии на кнопку теста (`Проверить подключение`) в секции **Тестирование подключения** происходит подключение к LDAP серверу (операция `bind`). В случае успешного теста выводится уведомление об успехе операции, в случае провала теста — сообщение об ошибке.



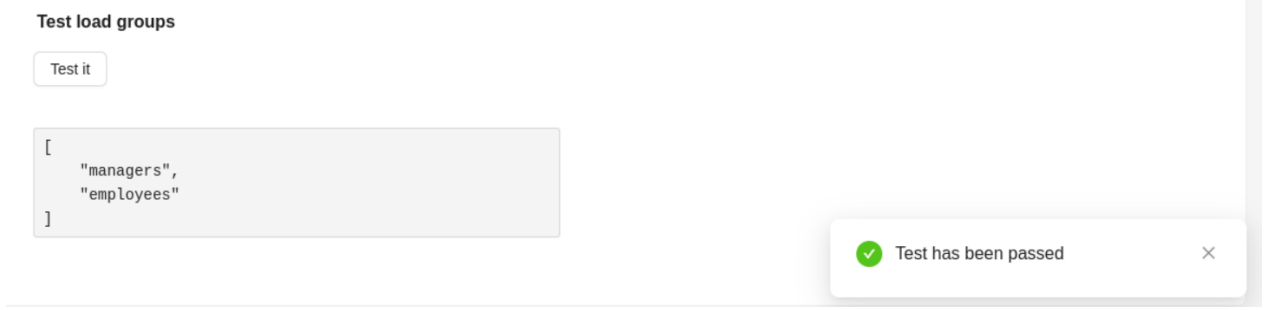
#Тестирование загрузки данных о пользователе

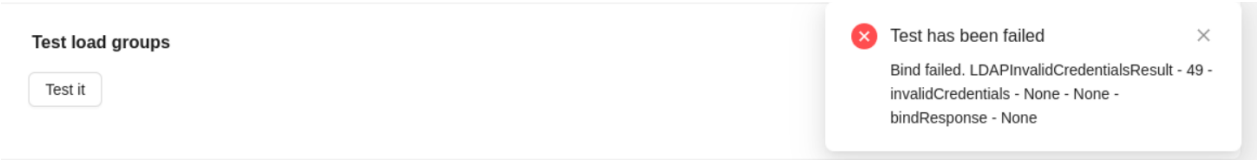
При нажатии на кнопку теста (Проверить подключение) в секции **Тест поиска пользователя** происходит подключение к LDAP серверу (операция `bind`) и поиск данных о пользователе (операция `search`) согласно данным в форме. В случае успешного теста выводится уведомление об успехе операции и результат поиска, в случае провала теста — сообщение об ошибке.



#Тестирование загрузки данных о группах

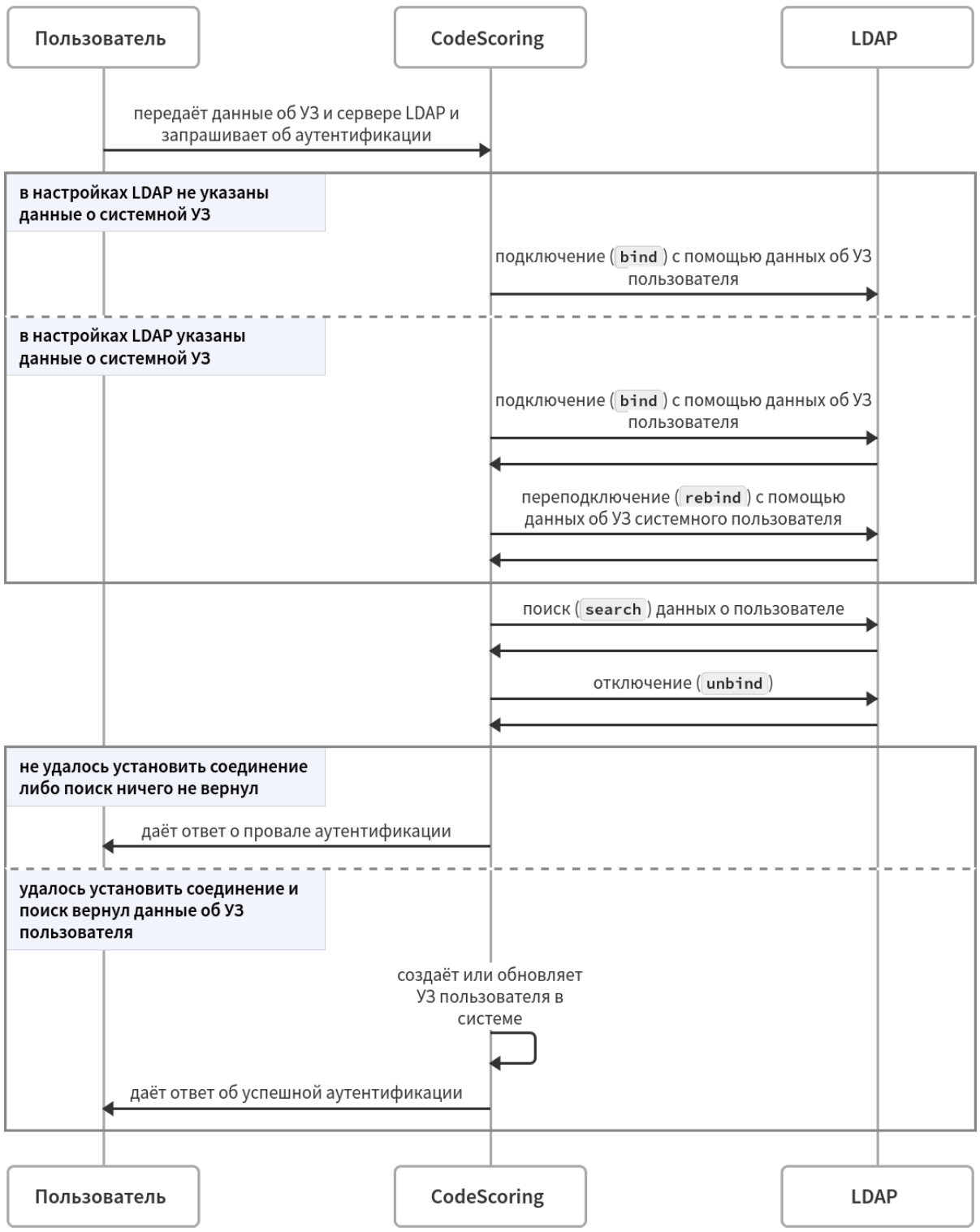
При нажатии на кнопку теста (Проверить подключение) в секции **Тест загрузки групп** происходит подключение к LDAP серверу (операция `bind`) и поиск данных о группах (операция `search`) согласно данным в форме. В случае успешного теста выводится уведомление об успехе операции и результат поиска, в случае провала теста — сообщение об ошибке.





#Механизм аутентификации с помощью LDAP

Механизм аутентификации в CodeScoring с помощью LDAP



#Замечания

- Использование авторизации через **LDAP** не подразумевает полную синхронизацию директории с информацией о пользователях из **Службы каталогов**.
- Редактирование название УЗ (`username`) и назначение пароля пользователю из **LDAP** невозможно.
- Возможно наличие пользователей из различных провайдеров аутентификации с одинаковым названием УЗ (`username`).
- Возможно назначение пользователю из **LDAP** любого уровня доступа (`User`, `Auditor`, `Administrator`, `Security Manager`).

#Настройка интеграции с OpenID Connect

#Страница аутентификации CodeScoring

CodeScoring поддерживает аутентификацию и авторизацию пользователей по протоколу **OpenID Connect**. На странице аутентификации доступно меню с выбором провайдера аутентификации. Помимо провайдера по умолчанию (локальные учётные записи, `internal directory`) и интеграций с LDAP, доступны для выбора активные интеграции с **OpenID Connect**.

Login provider

Internal directory

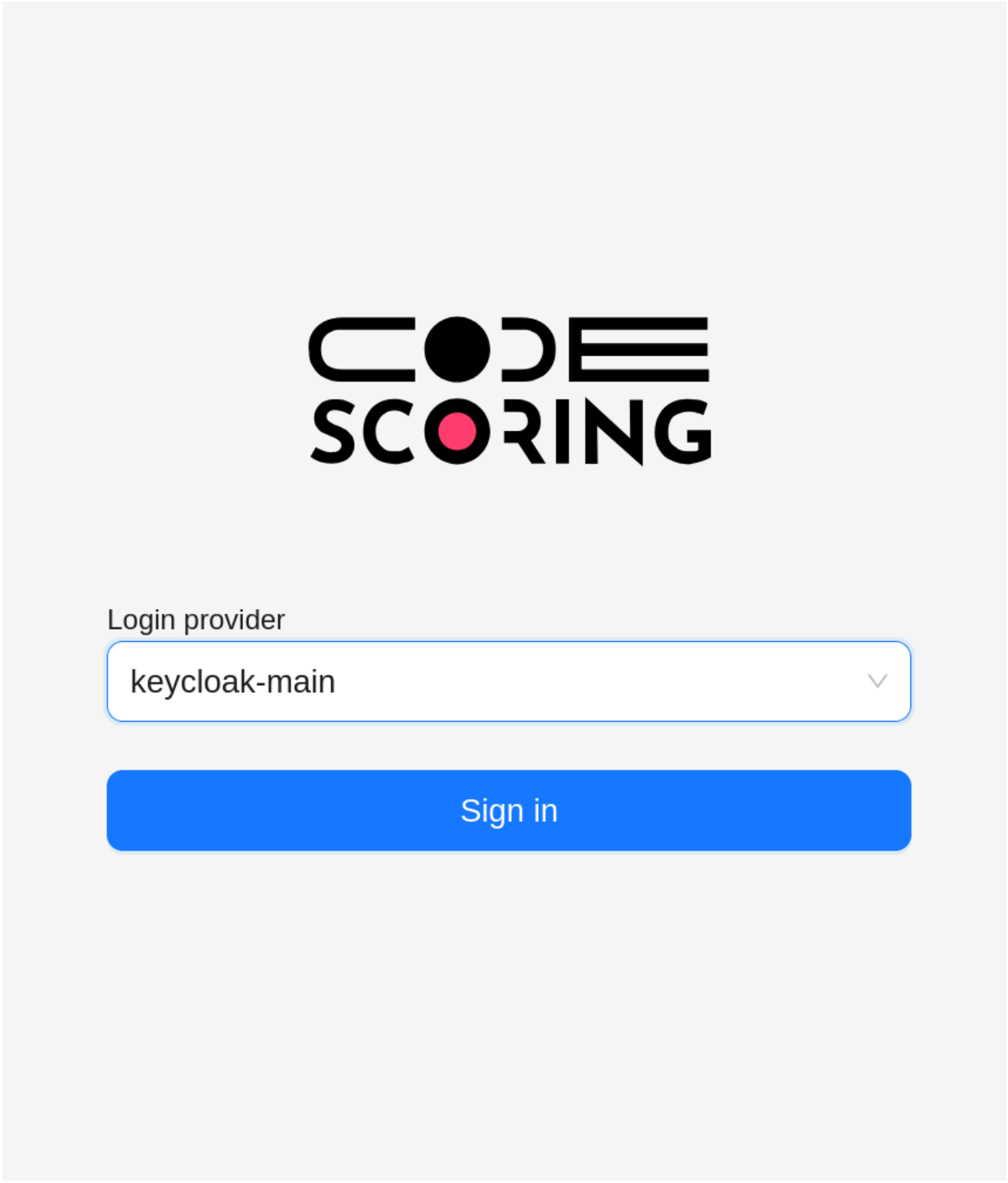
Internal directory internal

freeipa-main ldap

keycloak-main oidc

Password

Sign in



#Настройка клиента на стороне провайдера OpenID Connect

Ниже представлен пример валидной конфигурации клиента OpenID Connect на стороне провайдера при условии, что URL платформы CodeScoring - `https://codescoring.example.com/`:

- **Root url** - `https://codescoring.example.com/`;
- **Home url** - `https://codescoring.example.com/cabinet/dashboard/`;
- **Valid redirect urls** - `https://codescoring.example.com/auth/oidc/callback*`;

- **Web origins** - `https://codescoring.example.com/`;
- **Admin url** - `https://codescoring.example.com/`;
- **Authentication flow** - standard flow.

#Настройка интеграции с OpenID Connect на стороне CodeScoring

Сконфигурировать OpenID Connect на стороне CodeScoring можно в разделе `Настройки -> Провайдеры идентичности -> OpenID Connect`.

Переход на форму создания нового подключения осуществляется по кнопке **Создать**. В форме необходимо заполнить следующие поля:

#Основные настройки OIDC

- **Наименование** - название интеграции, будет отображаться на экране аутентификации;
- **Активно** - флаг активности интеграции, в зависимости от значения, эта интеграция будет доступна как провайдер аутентификации;
- **Идентификатор клиента** - значение должно соответствовать полю "идентификатор клиента" (client id) в конфигурации клиента на стороне провайдера;
- **Секрет клиента** - значение должно соответствовать полю "секрет клиента" (client secret) в конфигурации клиента на стороне провайдера.

#Настройки подключения

CodeScoring может настроить поля ниже на основе эндпоинта `/.well-known/openid-configuration` издателя OIDC при нажатии на кнопку `Получить настройки`.

- **Issuer URL провайдера** - адрес издателя провайдера OpenID Connect;
- **Authorization URL провайдера** - URL для авторизации в провайдере OpenID Connect, значение можно получить по `$ISSUER_URL/.well-known/openid-configuration`;
- **Token URL провайдера** - URL для получения у провайдера токена авторизации, значение можно получить по `$ISSUER_URL/.well-known/openid-configuration`;
- **UserInfo URL провайдера** - UserInfo эндпоинт провайдера, значение можно получить по `$ISSUER_URL/.well-known/openid-configuration`;
- **JWKS URL провайдера** - JWKS эндпоинт провайдера, значение можно получить по `$ISSUER_URL/.well-known/openid-configuration`;
- **Области доступа клиента** - значения передаются в параметре "scope" запроса авторизации.

#Настройки соотнесения полей

- **Поле с идентификатором субъекта** - название поля с идентификатором субъекта (sub) в ответе, возвращаемом эндпоинтом UserInfo;
- **Поле с именем учётной записи** - название поля с предпочитаемым именем УЗ пользователя в ответе, возвращаемом эндпоинтом UserInfo;
- **Поле с именем** - название поля с именем пользователя в ответе, возвращаемом эндпоинтом UserInfo;
- **Поле с фамилией** - название поля с фамилией пользователя в ответе, возвращаемом эндпоинтом UserInfo;
- **Поле с email** - название поля с email пользователя в ответе, возвращаемом эндпоинтом UserInfo.

#УЗ CodeScoring, созданные при аутентификации через OpenID Connect

При аутентификации через OpenID Connect, CodeScoring будет создавать новые или обновлять существующие учётные записи.

Соотносятся следующие данные:

- Идентификатор субъекта в провайдере OpenID Connect;
- Предпочитаемое название УЗ;
- Имя;
- Фамилия;
- Email.

УЗ, созданные таким образом, можно добавлять в группы, для них можно менять уровень доступа так же, как и для обычных УЗ.

Для УЗ, созданных при аутентификации через KeyCloak, нельзя менять имя УЗ или устанавливать пароль.

#Архитектура

CodeScoring.Save построен на микросервисной архитектуре. Backend API и контур аутентификации / авторизации вынесены в отдельные сервисы: **Save API service** и **cs-auth (Auth/RBAC service)**.

#Технологический стек

- **Язык разработки:** Go 1.25+.
- **Хранилище метаданных:** PostgreSQL 14+ (рекомендуется для production) или SQLite 3.x.
- **Объектное хранилище:** S3-совместимое (MinIO, AWS S3, Yandex Object Storage) или локальная файловая система.
- **Аутентификация:** RSA-2048 JWT с JWKS, выпускаемый cs-auth и локально валидируемый Save через публичный ключ.

#Архитектурная диаграмма

О схеме базы данных

В развёртывании с **PostgreSQL** оба сервиса могут делить один экземпляр БД и одну схему; данные разделены префиксами имён таблиц. В развёртывании с **SQLite** у каждого сервиса свой отдельный файл базы.

#Компоненты системы

#Save API service

Основной backend-сервис, обрабатывающий входящие запросы к репозиториям и административному API.

Функции:

- Обработка HTTP/HTTPS запросов от пакетных менеджеров и веб-интерфейса.
- Валидация токенов и интеграция с выделенным Auth/RBAC service.
- Маршрутизация запросов к storage backend.
- Интеграция с CodeScoring.OSA через OSA Proxy для сканирования артефактов при маршрутизации загрузок через прокси.
- Применение политик безопасности и политик очистки.
- Структурированное логирование и экспорт метрик в Prometheus.

Технические характеристики:

- Stateless-архитектура для горизонтального масштабирования.
- Graceful shutdown при обновлении.
- Health checks для Kubernetes.

#Auth/RBAC service

Выделенный сервис аутентификации и авторизации, разворачиваемый отдельно от Save API service.

Функции:

- Аутентификация пользователей и сервисных аккаунтов.
- Выпуск и обновление JWT-токенов (RSA-2048, alg=RS256).
- Управление ролями и разрешениями (RBAC) на уровнях global / project / repository.
- Управление API keys, robot accounts и project membership.
- Аудит операций, связанных с доступом и общесистемными событиями.
- Internal API для взаимодействия с Save API service.
- OCI Distribution Spec token endpoint для Docker / Helm / oras-клиентов.

Технические характеристики:

- Отдельная кодовая база и отдельная точка входа.
- Собственная схема в PostgreSQL при общем экземпляре БД, либо отдельный SQLite-файл.
- Горизонтальное масштабирование независимо от Save API service.
- **Обязательное условие масштабирования:** все реплики должны делить один и тот же RSA-ключ для подписи JWT.

#Storage Backend

Подсистема хранения артефактов:

- **Metadata Store** — PostgreSQL или SQLite для хранения метаданных.
- **Blob Store** — объектное хранилище для артефактов: S3-совместимое или filesystem.

Операции: сохранение и извлечение артефактов, управление версиями, генерация контрольных сумм.

#Background Workers

Фоновые процессы для выполнения отложенных задач:

- Политики очистки — удаление устаревших артефактов по расписанию.
- Фоновые операции с метаданными.
- Асинхронная отправка аудит-событий в cs-auth.

#Web UI

Веб-интерфейс для управления проектами, репозиториями, артефактами и настройками доступа. Общее описание возможностей модуля приведено на странице Функциональные характеристики.

#Взаимодействие сервисов

- Внешние клиенты (Web UI и пакетные менеджеры) обращаются только к Save API service через ingress; cs-auth напрямую снаружи кластера не доступен
- Save API service делегирует аутентификацию, проверку ролей и выпуск токенов в Auth/RBAC service
- Оба сервиса могут использовать отдельные или общий экземпляр PostgreSQL, но работают они в отдельных схемах/базах данных
- Save API service напрямую работает с metadata store и object storage; cs-auth работает только с metadata store и не имеет доступа к object storage
- Save обращается к cs-auth в нескольких случаях:
  - **Проксирование части административного API** — Save форвардит запросы, не обрабатывая их сам

    - **Проверка учётных данных**, отсутствующих в локальном кэше пода — (Basic Auth, API key, NuGet API key, NPM token, Docker token)
    - **Оповещение о событиях для аудита** - единый журнал событий для всех сервисов
- Все internal-эндпоинты дополнительно защищены shared secret.
- На горячем пути Save валидирует Bearer JWT **локально**, используя кэшированный публичный RSA-ключ JWKS без сетевого вызова в cs-auth.

#Модель данных

#Основные сущности

#Project (Проект)

```
- ID
- Name
- Description
- Owner
- Created/Updated timestamps
```

#Repository (Репозиторий)

```
- ID
- Project ID
- Name
- Type (proxy/hosted)
- Format (maven, npm, docker, nuget, pypi, go, raw)
- Configuration
- Storage Backend
- Cleanup Policies
- Created/Updated timestamps
```

#Artifact (Артефакт)

```
- ID
- Repository ID
- Group/Namespace
- Name
- Version
- Format-specific metadata
- Size
- Checksums (MD5, SHA1, SHA256)
- Upload date
- Last accessed
```

#**User (Пользователь)**

```
- ID
- Username
- DisplayName
- Email
- PasswordHash (bcrypt)
- IsActive
- AuthSource   (local / ldap / oidc)
- LastLoginAt
- Created/Updated timestamps
```

#**Role / Robot account (Роль или сервисный аккаунт)**

```
- ID
- Name
- Description
- Permissions (global / project / repository scopes)
- IsAdmin
- IsSystem
- API keys (only for robot accounts)
- Created/Updated timestamps
```

#**APIKey (API-ключ)**

```
- ID
- UserID            (FK на user с IsService=true)
- KeyPrefix         (первые 12 hex-символов plaintext-ключа — 
индекс для O(1)-поиска)
- KeyHash           (bcrypt(plaintext))
- Description
- ExpiresAt         (NULL = бессрочный)
- LastUsedAt
- CreatedAt
```

Полный plaintext-ключ возвращается клиенту единожды при создании.

#**Cleanup Policy (Политика очистки)**

```
- ID
- Name
- Type (cleanup/security)
- Rules
- Repositories
- Enabled
- Created/Updated timestamps
```

#Варианты установки

CodeScoring.Save поддерживает несколько профилей установки. Требования, различия между профилями и порядок развертывания описаны в разделе Варианты установки.

#Безопасность

#Аутентификация и авторизация

- JWT tokens для API, выпускаемые Auth/RBAC service
- Basic Auth / API keys для пакетных менеджеров и service accounts
- Ролевая модель (RBAC) с тремя scope'ами (global / project / repository), управляемая выделенным Auth/RBAC service
- Межсервисная валидация токенов через JWKS-эндпоинт

Способы аутентификации со стороны клиента подробно описаны в разделе Аутентификация.

#Шифрование данных

Сами сервисы Save и cs-auth не выполняют шифрование на уровне приложения; используются возможности нижележащих компонентов:

- **Encryption at rest** обеспечивается выбранным object storage (S3 server-side encryption, MinIO encryption-at-rest) и PostgreSQL (через TDE-расширения или volume-level шифрование).
- **Encryption in transit** обеспечивается TLS на ingress / load balancer и в межсервисных соединениях.
- **Пароли пользователей и API-ключи robot-аккаунтов** хранятся как bcrypt-хэши в cs-auth.

#Аудит

- Полное логирование всех операций — единый журнал, в который пишут оба сервиса.
- Экспорт в машинно-читаемый формат (JSON) и HTML. Максимум 10 000 записей за один экспорт.
- Поддерживаемые фильтры списка/экспорта: `from`, `to`, `username`, `action`, `resource_type`, `q`, `user_id`. В web-UI доступен только поиск по имени пользователя, типу ресурса и действию.

#Производительность

#Оптимизации

- **Кэширование:**
  - In-process TTL-кэш `credential → PermissionSet` в каждом поде Save API service — устраняет round-trip в cs-auth при повторных запросах с теми же учётными данными
  - Локальная валидация Bearer JWT по публичному ключу JWKS — без сетевого вызова в cs-auth на горячем пути. JWKS обновляется в фоне периодически.
- **Connection pooling:**
  - Пул соединений к БД
  - Переиспользование HTTP клиентов к upstream-репозиториям
- **Параллельная обработка:**
  - Concurrent uploads / downloads через горутины.
  - Batch (bulk) operations API
- **Streaming больших файлов:**
  - Артефакты передаются через `MultiWriter` без буферизации целиком в памяти.
  - Атомарная запись на диск через temp-file rename, чтобы избежать повреждения при конкурентной записи.

#Целевые показатели

Для базовой инсталляции:

- Latency (p95): < 100ms для cached requests
- Throughput: 300+ requests/sec
- Concurrent uploads: 100+
- Max artifact size: 5 GB

#Масштабирование

#Горизонтальное масштабирование

- Save API service — stateless, масштабируется добавлением реплик.
- Auth/RBAC service — stateless, масштабируется добавлением реплик. Обязательное условие: все реплики должны использовать один и тот же RSA-ключ для подписи JWT.
- PostgreSQL — масштабируется вертикально, при необходимости используются read-реплики.
- Объектное хранилище — масштабируется средствами провайдера: S3 или MinIO в кластере.

#Вертикальное масштабирование

- Увеличение ресурсов подов.
- Увеличение ресурсов PostgreSQL.
- Расширение объектного хранилища.

#Мониторинг и наблюдаемость

#Метрики (Prometheus)

Save экспортирует метрики в формате OpenTelemetry с Prometheus exporter'ом.

HTTP-метрики:

- `http_requests_total{method,path,status}` — счётчик HTTP-запросов.
- `http_request_duration_seconds{method,path,status}` — гистограмма latency.
- `http_requests_active` — gauge активных запросов.

Метрики операций:

- `storage_operations_total{operation,backend,success}` — операции с хранилищем.
- `artifact_uploads_total{artifact_type,repository}` — загрузки артефактов.
- `artifact_downloads_total{artifact_type,repository}` — скачивания артефактов.
- `proxy_requests_total{artifact_type,repository,cache_hit,success}` — proxy-запросы. Cache hit rate вычисляется как `rate(proxy_requests_total{cache_hit="true"}[5m]) / rate(proxy_requests_total[5m])` .
- `cleanup_operations_total{policy_type,repository,artifacts_deleted,success}` — операции очистки.

#Логирование

- Структурированные логи в формате JSON (или text, по конфигурации).
- Уровни: debug, `info`, `warn`, `error`.
- Вывод: stdout, stderr или файл.
- Совместимы с любым log-aggregator'ом (ELK, Loki, Vector), которые умеют JSON.

#Доступность и устойчивость

CodeScoring.Save не реализует автоматическое переключение при отказе, восстановление на конкретный момент времени и встроенное

резервное копирование самостоятельно. Эти возможности обеспечиваются стандартными средствами инфраструктуры, в которой развёрнут сервис:

- **Несколько реплик Save и cs-auth** в Kubernetes за балансировщиком нагрузки обеспечивают доступность при отказе одного из подов.
- **Репликация PostgreSQL** и **резервное копирование базы данных** настраиваются средствами PostgreSQL-оператора или managed-сервиса.
- **Снапшоты и репликация объектного хранилища** выполняются средствами S3-совместимого хранилища.
- **Многоузловой Kubernetes-кластер** обеспечивает устойчивость к отказу отдельного узла.

Save не теряет данные при перезапуске, потому что всё состояние хранится во внешних компонентах: метаданные — в базе данных, артефакты — в объектном хранилище. Stateless-архитектура Save позволяет масштабировать сервис и пересоздавать поды без потери данных.

#Требования к установке CodeScoring.Save

CodeScoring.Save поддерживает два варианта установки: SQLite-вариант в k3s и PostgreSQL-вариант в Kubernetes. Оба варианта устанавливаются через Helm-чарт save.

#Общие требования

- Linux-based OS (Ubuntu 20.04+, CentOS 8+, RHEL 8+)
- Kubernetes-совместимый кластер: k3s для SQLite-варианта, полноценный Kubernetes-кластер для PostgreSQL-варианта
- Helm 3.x
- Доступ к реестру контейнерных образов CodeScoring.Save

#Hosted-инсталляция

Минимальные требования:

- 2 ядра CPU
- 4 GB RAM
- 50 GB дискового пространства (рекомендуется SSD)

Рекомендуемые требования:

- 4 ядра CPU
- 8 GB RAM
- 100 GB дискового пространства (SSD)

Программное обеспечение:

- k3s v1.25+
- SQLite 3.x (включен в дистрибутив)

#PostgreSQL-инсталляция

Kubernetes-кластер:

- 3+ рабочих узла
- 4 ядра CPU на узел
- 8 GB RAM на узел
- 50 GB дискового пространства на узел

PostgreSQL:

- 4 ядра CPU
- 8 GB RAM
- 100 GB storage (SSD рекомендуется)
- PostgreSQL 14+

Объектное хранилище:

- встроенный MinIO из зависимости Helm-чарта `minio` либо внешнее S3-совместимое хранилище: MinIO, AWS S3, Yandex Object Storage
- 500 GB+ хранилища, зависит от объема артефактов

#Варианты установки CodeScoring.Save

CodeScoring.Save поддерживает два варианта установки: SQLite-вариант в k3s и PostgreSQL-вариант в Kubernetes. Оба варианта устанавливаются через Helm-чарт save, но отличаются инфраструктурой и сценарием эксплуатации.

Адрес Helm-репозитория и реестра контейнерных образов CodeScoring.Save предоставляет вендор. В примерах установки используются плейсхолдеры <HELM_REPOSITORY_URL> и <REGISTRY_HOST>.

#Доступные варианты

Вариант	Когда использовать
Hosted-инсталляция	Для тестирования, демонстраций, небольших команд и быстрого старта без отдельной инфраструктуры PostgreSQL и S3
PostgreSQL-инсталляция	Для продуктивного использования, масштабирования и эксплуатации с внешней или встроенной PostgreSQL и S3-совместимым хранилищем

Перед выбором варианта проверьте требования к установке и подготовьте общий файл `values.yaml`. Основные параметры задаются в секциях:

- `image`
- `envs`
- `configMaps`
- `deploymentsGeneral`
- `postgres`
- `minio`

В `image` задаются общий реестр, тег и pull secret для образов backend, frontend и auth. Переменные окружения сервисов Save задаются в `envs`.

Чтобы значения из `envs` попали в Pod, включите ConfigMap `envs` и подключите его к deployments через `deploymentsGeneral.envConfigmaps`.

#Hosted-инсталляция CodeScoring.Save

Hosted-вариант разворачивается в k3s и использует SQLite-профиль. Такой вариант подходит для тестирования, демонстрации возможностей, небольших команд и быстрого старта без отдельной инфраструктуры PostgreSQL и S3.

#Установка k3s

```
## Установка k3s
curl -sfL https://get.k3s.io | sh -

## Проверка статуса
sudo systemctl status k3s

## Настройка kubectl
mkdir -p ~/.kube
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
sudo chown $USER:$USER ~/.kube/config
export KUBECONFIG=~/.kube/config

## Проверка кластера
kubectl get nodes
```

#Подготовка Helm и пространства имен

Адрес Helm-репозитория и реестра контейнерных образов CodeScoring.Save предоставляет вендор. В примерах ниже используются плейсхолдеры <HELM_REPOSITORY_URL> и <REGISTRY_HOST>.

```
## Добавление Helm-репозитория CodeScoring
helm repo add codescoring <HELM_REPOSITORY_URL>
helm repo update

## Создание пространства имен
kubectl create namespace codescoring-save --dry-run=client -o
yaml | kubectl apply -f -
```

Создайте imagePullSecret для доступа к приватному реестру. Имя секрета должно совпадать со значением image.pullSecrets в values.yaml.

```
kubectl create secret docker-registry codescoring-pvt-regcred \
  --namespace codescoring-save \
  --docker-server=<REGISTRY_HOST> \
  --docker-username=<USERNAME> \
  --docker-password=<PASSWORD> \
  --dry-run=client -o yaml | kubectl apply -f -
```

#Конфигурация SQLite

Для SQLite-варианта отключите встроенный PostgreSQL и S3:

```
postgres:
  enabled: false

minio:
  enabled: false
```

В image укажите реестр, тег и pull secret для образов Save:

```
image:
  registry: <REGISTRY_HOST>
  tag: <SAVE_VERSION>
  pullSecrets:
    - name: codescoring-pvt-regcred
```

Переменные окружения задаются в envs. Чтобы они попали в Pod, включите ConfigMap envs и подключите его к deployments:

```
envs:
  DATABASE_DRIVER: "sqlite"
  STORAGE_TYPE: "filesystem"
  DATABASE_SQLITE_PATH: "./data/repository_manager.db"

configMaps:
  envs:
    enabled: true
```

```
deploymentsGeneral:
  envConfigmaps:
    - envs

deployments:
  auth:
    containers:
    - name: auth
      image:
        repository: repository-manager-auth
      env:
        - name: DATABASE_DRIVER
          value: "sqlite"
        - name: DATABASE_SQLITE_PATH
          value: "/app/data/auth.db"
```

#Установка

```
helm install codescoring-save codescoring/save \
  --namespace codescoring-save \
  --create-namespace \
  --values values.yaml
```

#Проверка установки

```
## Проверка статуса подов
kubectl get pods -n codescoring-save

## Проверка логов backend
kubectl logs -n codescoring-save -l app=backend --tail=100

## Проверка логов auth
kubectl logs -n codescoring-save -l app=auth --tail=100

## Проверка логов frontend
kubectl logs -n codescoring-save -l app=frontend --tail=100

## Проверка сервисов
kubectl get svc -n codescoring-save

## Проверка ingress
kubectl get ingress -n codescoring-save
```

#Первоначальная настройка

Пароль администратора задаётся в переменной `AUTH_ADMIN_PASSWORD`:

```
envs:
  AUTH_ADMIN_PASSWORD: <STRONG_PASSWORD>
```

После установки откройте веб-интерфейс по адресу, указанному в настройках ingress.

#Следующие шаги

После успешной установки:

1. Создайте первый репозиторий
2. Настройте политики очистки
3. Создайте пользователей и назначьте роли

#PostgreSQL-инсталляция CodeScoring.Save

PostgreSQL-инсталляция предназначена для продуктивного использования, больших команд, высоких требований к доступности, масштабируемости и производительности.

#Подготовка инфраструктуры

#PostgreSQL

По умолчанию PostgreSQL разворачивается как зависимость Helm-чарта. Для этого в `values.yaml` включите `postgres.enabled=true` и задайте параметры подключения:

```
postgres:
  enabled: true
  fullnameOverride: postgres
  auth:
    database: save_db
    username: save_user
    password: <STRONG_PASSWORD>
```

При использовании встроенного PostgreSQL backend и auth должны подключаться к хосту `postgres`.

Для внешней PostgreSQL отключите встроенную зависимость:

```
postgres:
  enabled: false
```

После этого укажите внешний хост PostgreSQL в `envs.DATABASE_HOST`.

Подготовьте базу данных и пользователя во внешней PostgreSQL:

```
CREATE DATABASE save_db;
CREATE USER save_user WITH PASSWORD '<STRONG_PASSWORD>';
GRANT ALL PRIVILEGES ON DATABASE save_db TO save_user;
```

#Объектное хранилище

По умолчанию MinIO разворачивается как зависимость Helm-чарта. Для этого в `values.yaml` включите `minio.enabled=true`. Сервис будет доступен как `save-minio`.

```
minio:
  enabled: true
  fullnameOverride: save-minio
  auth:
    rootUser: <ACCESS_KEY>
    rootPassword: <SECRET_KEY>
  provisioning:
    enabled: true
    buckets:
      - save
```

Для установки с сохранением данных включите persistence у PostgreSQL и MinIO:

```
postgres:
  persistence:
    enabled: true
    storageClassName: default
    mountPath: /var/lib/postgresql/data
    size: 5Gi

minio:
  persistence:
    enabled: true
    storageClassName: default
    mountPath: /data
    size: 20Gi
```

Общий блок `pvcs` не должен использоваться для PostgreSQL и MinIO в поставляемом чарте. Если `postgres.persistence.enabled` или `minio.persistence.enabled` выключены, данные соответствующего сервиса не сохраняются после пересоздания Pod.

Для встроенного MinIO в `envs` используйте endpoint `http://save-minio:9000`. Секция `minio.provisioning` создает бакет при установке чарта, поэтому `S3_BUCKET` должен совпадать с одним из значений `minio.provisioning.buckets`.

Для внешнего S3-совместимого хранилища отключите встроенный MinIO:

```
minio:
  enabled: false
```

После этого задайте `S3_ENDPOINT`, `S3_BUCKET`, `S3_ACCESS_KEY` и `S3_SECRET_KEY` в `envs`.

#Конфигурация CodeScoring.Save

Используйте базовый `values.yaml` из Helm-чарта и измените нужные параметры.

Для встроенных PostgreSQL и MinIO используйте такие значения:

```
image:
  registry: <REGISTRY_HOST>
  tag: <SAVE_VERSION>
  pullSecrets:
    - codescoring-pvt-regcred

envs:
  DATABASE_DRIVER: postgres
  DATABASE_HOST: postgres
  DATABASE_PORT: "5432"
  DATABASE_NAME: save_db
  DATABASE_USER: save_user
  DATABASE_PASSWORD: <STRONG_PASSWORD>

  STORAGE_TYPE: s3
  S3_ENDPOINT: http://save-minio:9000
  S3_BUCKET: save
  S3_ACCESS_KEY: <ACCESS_KEY>
  S3_SECRET_KEY: <SECRET_KEY>

  AUTH_SERVICE_URL: http://cs-auth.example.com:9100
  AUTH_JWKS_URL: http://cs-auth.example.com:9100/internal/v1/
jwks
  AUTH_ADMIN_PASSWORD: <STRONG_PASSWORD>
  AUTH_INTERNAL_SECRET: <STRONG_SECRET>

configMaps:
  envs:
    enabled: true

deploymentsGeneral:
  envConfigmaps:
    - envs
```

```
postgres:
  enabled: true
  fullnameOverride: postgres
  auth:
    database: save_db
    username: save_user
    password: <STRONG_PASSWORD>
  persistence:
    enabled: true
    storageClassName: default
    mountPath: /var/lib/postgresql/data
    size: 5Gi

minio:
  enabled: true
  fullnameOverride: save-minio
  auth:
    rootUser: <ACCESS_KEY>
    rootPassword: <SECRET_KEY>
  provisioning:
    enabled: true
    buckets:
      - save
  persistence:
    enabled: true
    storageClassName: default
    mountPath: /data
    size: 20Gi
```

Значение AUTH_INTERNAL_SECRET используется для внутреннего взаимодействия сервисов Save. Секция `minio.provisioning` создает бакет для встроенного MinIO, а S3_BUCKET должен совпадать с одним из значений `minio.provisioning.buckets`.

#Установка

```
helm install codescoring-save codescoring/save \
  --namespace codescoring-save \
  --create-namespace \
  --values values.yaml
```

#Проверка установки

```
## Проверка статуса подов
kubectl get pods -n codescoring-save -w

## Проверка всех ресурсов
kubectl get all -n codescoring-save
```

```
## Проверка логов backend
kubectl logs -n codescoring-save -l app=backend --tail=100

## Проверка логов auth
kubectl logs -n codescoring-save -l app=auth --tail=100

## Проверка логов frontend
kubectl logs -n codescoring-save -l app=frontend --tail=100

## Проверка готовности
kubectl get pods -n codescoring-save -o wide
```

#Следующие шаги

После успешной установки:

1. Создайте первый репозиторий
2. Настройте политики очистки
3. Создайте пользователей и назначьте роли

#Обновление системы

Обновление выполняется через Helm. Перед обновлением сохраните используемый `values.yaml`, проверьте целевую версию Helm-чарта и подготовьте резервную копию данных для PostgreSQL-инсталляции.

#Hosted-инсталляция

```
## Обновление Helm-чарта
helm repo update

## Обновление до новой версии
helm upgrade codescoring-save codescoring/save \
  --namespace codescoring-save \
  --values values.yaml \
  --version <NEW_VERSION>
```

#PostgreSQL-инсталляция

```
## Резервное копирование перед обновлением
kubectl exec -n codescoring-save postgres-0 -- \
  pg_dump -U save_user -d save_db > backup-$(date +%Y%m%d).sql

## Обновление
helm upgrade codescoring-save codescoring/save \
  --namespace codescoring-save \
  --values values.yaml \
  --version <NEW_VERSION>
```

```
## Проверка развертываний
kubectl get deploy -n codescoring-save

## Проверка статуса обновления backend
kubectl rollout status deployment/codescoring-save -n
codescoring-save

## Проверка статуса обновления auth
kubectl rollout status deployment/codescoring-save-auth -n
codescoring-save

## Проверка статуса обновления frontend
kubectl rollout status deployment/codescoring-save-save-front
-n codescoring-save
```

#Удаление инсталляции

Удаление пространства имен

Удаление Helm-релиза останавливает компоненты CodeScoring.Save. Удаление пространства имен дополнительно удаляет все ресурсы внутри него, поэтому перед этой операцией нужно сохранить необходимые данные и конфигурацию.

```
## Удаление Helm-релиза
helm uninstall codescoring-save --namespace codescoring-save

## Удаление пространства имен
kubectl delete namespace codescoring-save
```

#Настройка SSL/TLS для CodeScoring.Save

SSL/TLS настраивается на уровне ingress. Эта страница описывает базовую структуру `values.yaml` и два типовых способа подключить сертификат: через cert-manager или через заранее созданный Kubernetes secret.

Параметры ingress и TLS задаются в `values.yaml` в секции `app.ingresses`. Формат секции должен соответствовать схеме `codescoring-generic`.

Note

Точный набор аннотаций зависит от используемого ingress-контроллера и способа выпуска сертификатов.

Пример структуры:

```
app:
  ingresses:
    frontend:
      className: nginx
      annotations:
        cert-manager.io/cluster-issuer: letsencrypt-prod
      hosts:
        - host: save.example.com
          paths:
            - path: /
              pathType: Prefix
              service:
                name: frontend
                port:
                  number: 8081
      tls:
        - secretName: save-tls
          hosts:
            - save.example.com
```

#Использование cert-manager

```
## Установка cert-manager
kubectl apply -f https://github.com/cert-manager/cert-manager/
releases/download/v1.13.0/cert-manager.yaml

## Создание ClusterIssuer
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: admin@example.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
EOF
```

#Использование самоподписанных сертификатов

```
## Генерация сертификата
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout tls.key -out tls.crt \
```

```
  -subj "/CN=save.example.com"

## Создание secret
kubectl create secret tls save-tls \
  --namespace codescoring-save \
  --cert=tls.crt \
  --key=tls.key
```

#Диагностика неполадок CodeScoring.Save

Этот раздел помогает быстро собрать первичную информацию о состоянии инсталляции: состояние pod'ов, логи сервисов, доступность базы данных и хранилища. Команды ниже не исправляют проблему автоматически, а дают данные для дальнейшего анализа.

#Проблемы с pods

```
## Описание pod
kubectl describe pod <POD_NAME> -n codescoring-save

## Логи pod
kubectl logs <POD_NAME> -n codescoring-save --previous

## Вход в pod для диагностики
kubectl exec -it <POD_NAME> -n codescoring-save -- /bin/sh
```

#Проблемы с базой данных

```
## Проверка подключения к PostgreSQL из postgres pod
kubectl exec -it -n codescoring-save postgres-0 -- \
  psql -U save_user -d save_db

## Проверка таблиц внутри psql
\dt
\q
```

#Проблемы с хранилищем

```
## Проверка доступа к MinIO
kubectl port-forward -n codescoring-save svc/save-minio
9001:9001

## Откройте в браузере:
## http://localhost:9001
```

#Общее

Раздел содержит базовые сценарии работы с платформой:

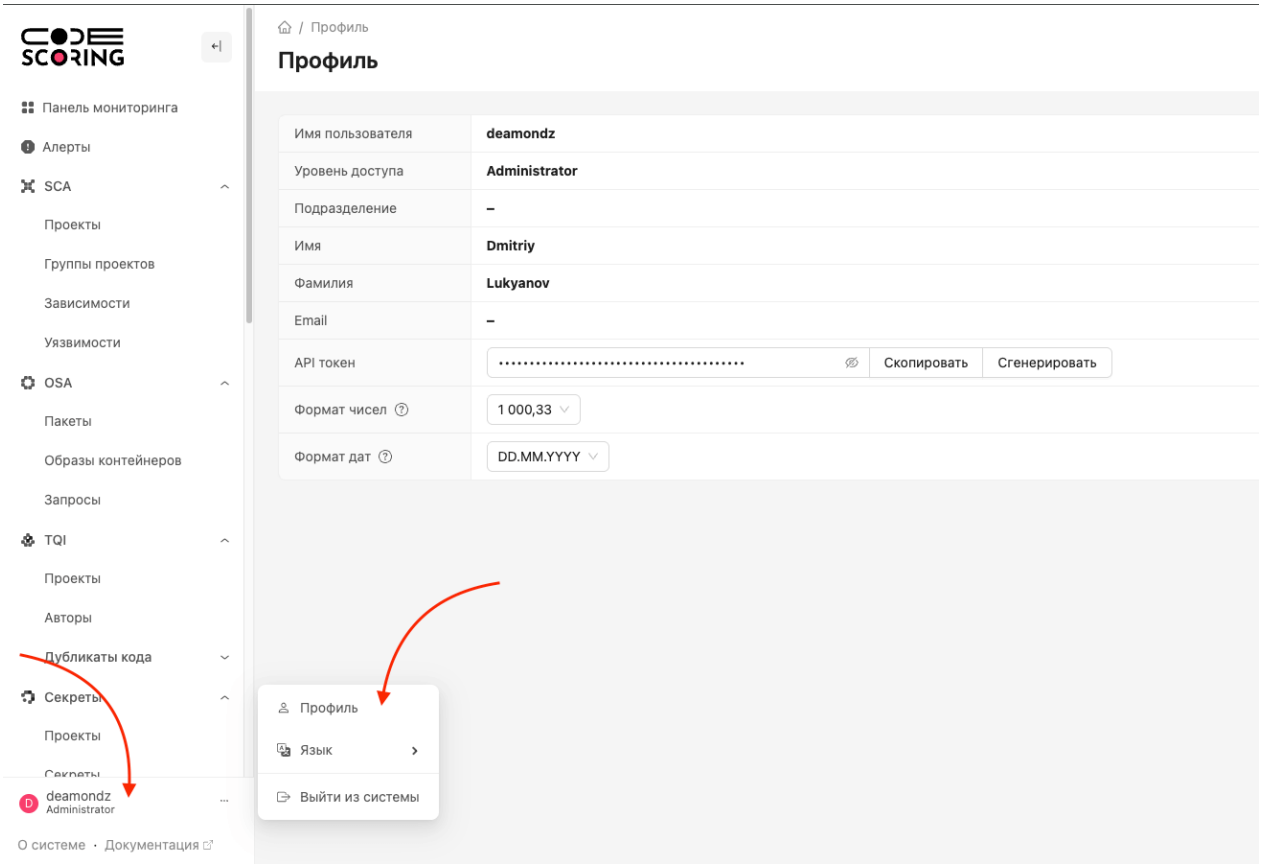
- настройка профиля пользователя;
- создание и настройка политик безопасности;
- подключение VCS и управление проектами;
- расширенные настройки (подразделения, уведомления, аудит-лог, метрики, webhooks, API).

Используйте навигацию слева для перехода к нужному подразделу.

#Настройка профиля пользователя

Раздел **Профиль** позволяет пользователю просматривать и изменять персональные данные, а также настроить отображение интерфейса.

Просмотр профиля доступен по нажатию на имя пользователя в левом нижнем углу интерфейса.



На странице отображаются следующие параметры:

- **Имя пользователя** — уникальный логин, используемый для входа в систему. Не подлежит редактированию;
- **Уровень доступа** — определяет полномочия в системе. Возможные значения: `User`, `Auditor` или `Administrator`;

- **Подразделение** — организационная единица, к которой привязан пользователь (если используется);
- **Имя** – отображается в интерфейсе и отчетах;
- **Фамилия** — отображается в интерфейсе и отчетах;
- **Email** — контактный адрес, отображаемый в списках пользователей;
- **API токен** — используется для интеграции с внешними инструментами. Можно скопировать или сгенерировать новый (предыдущий станет недействительным);
- **Формат чисел** — отображение чисел в системе;
- **Формат дат** — отображение дат в системе.

Важно: формат чисел и дат влияет только на отображение. На ввод данных и экспорт эти настройки не распространяются.

#Редактирование профиля

По нажатию кнопки **Редактировать** открывается форма редактирования профиля. Доступно изменение следующих полей:

- Имя;
- Фамилия;
- Email.

Изменения сохраняются немедленно после нажатия кнопки **Сохранить**.

#Настройка политик

#Принципы работы

Политики на платформе CodeScoring представляют собой механизм отслеживания и блокирования open source компонентов в процессе разработки программного обеспечения. Они могут быть связаны с проверкой безопасности, совместимости лицензий или других критериев включения сторонних компонентов в разработку.

Политики можно создавать для:

- всей организации;
- подразделения;
- группы проектов;
- проекта;
- окружения разработки;
- репозитория;
- типа компонента.

Механизм политик учитывает указанный этап разработки программного обеспечения: от поступления сторонних компонентов в периметр организации до отслеживания сборок и написания нового кода.

Политики настраиваются по условиям, объединенным логическими выражениями **И/ИЛИ**. Помимо стандартных настроек политик безопасности по уровню критичности уязвимостей, условия могут быть настроены согласно метаданным компонента: дата релиза, лицензия, автор и другим. Всего поддерживается **40+ типизированных условий**. Среди проверок также присутствует встроенная вендорская политика проверки на лицензионную чистоту.

В результате срабатывания политики в CodeScoring создаются соответствующие **алерты**. Алерты можно временно или навсегда проигнорировать, а также выгрузить их в виде отчета.

Политики могут быть **блокирующими**: при срабатывании такой политики используемые компоненты блокируются в хранилище артефактов (прокси-репозитории), либо останавливается сборка программного продукта до устранения выявленного дефекта.

Дополнительно, по срабатыванию политики может быть направлено уведомление ответственным специалистам в систему управления задачами или электронное письмо с описанием проблемы.

#Этапы работы политик

Этапы работы политик настраиваются пользователем при редактировании параметров проекта в разделе **Настройки → Проекты** в поле **Этап политики** или указываются через параметр `--stage` при запуске консольного агента Johnny. Наименования стадий имеют следующие значения:

- `dev` – этап разработки;
- `stage` – промежуточный (предпромышленный) этап;
- `test` – этап тестирования;
- `prod` – промышленный контур.

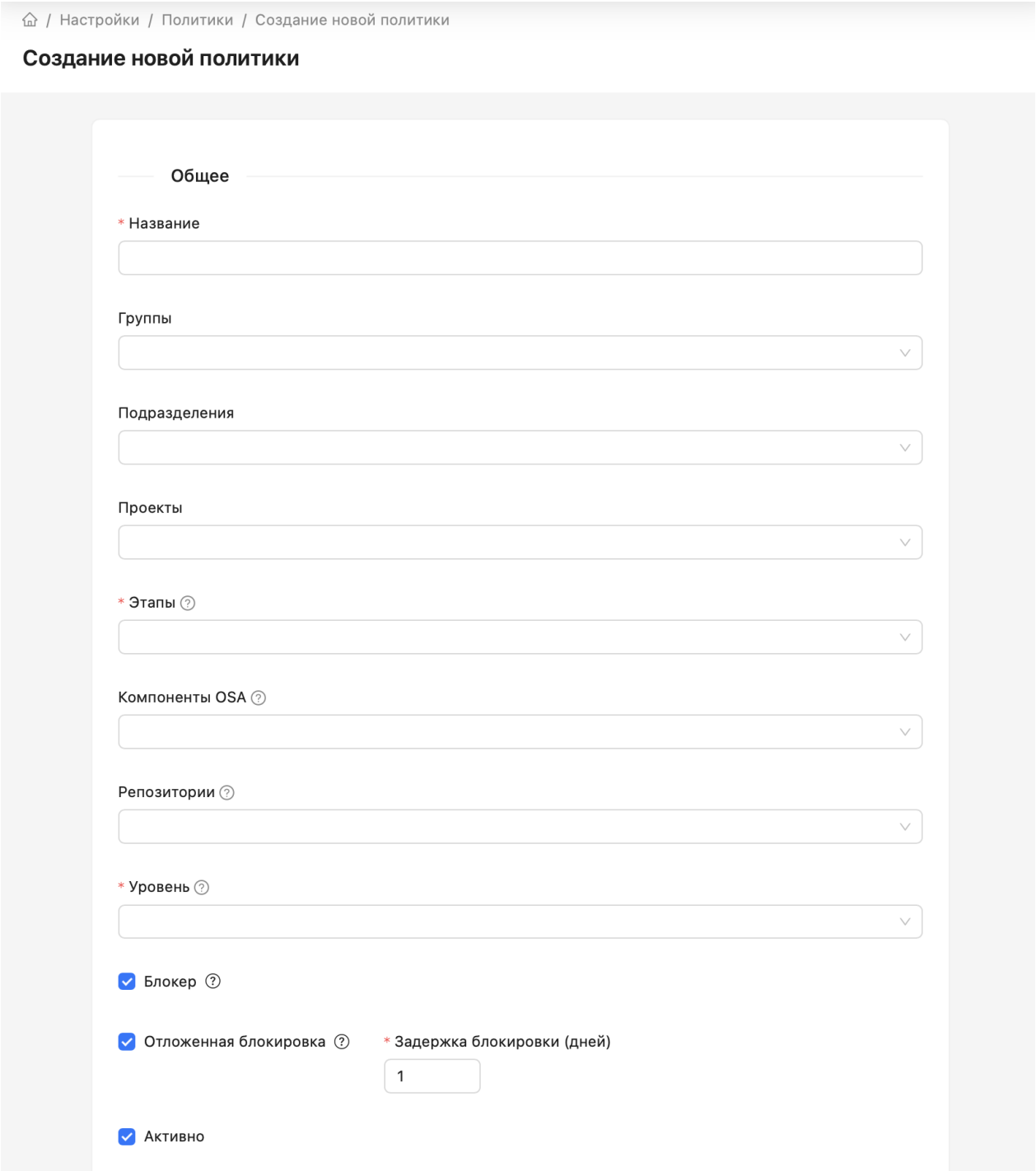
При создании или редактировании политики в разделе **Настройки → Политики** необходимо указать стадии, к которым она будет применяться.

Кроме того, существуют специальные значения стадий, используемые по умолчанию для определённых задач:

- `proxy` – для плагина в модуле OSA;
- `source` – для анализа VCS-проектов;
- `build` – для консольного агента Johnny.

#Создание политики

Политики создаются в разделе Настройки -> Политики. Перейти на форму создания политики можно по кнопке **Создать**.



В форме создания политики задается контекст работы политики по следующим параметрам:

- **Название**;
- **Группы** — группы проектов, на которые распространяется политика. Если параметр пустой – политика применяется для всей организации;
- **Подразделения** — подразделения организации, на которого распространяется политика. Если параметр пустой – политика применяется для всей организации;
- **Проекты** — проекты, на которые применяется политика;
- **Этапы** — стадии цикла разработки, на которые применяется политика;

- **Компоненты OSA** – тип компонентов в менеджере репозиториев с плагином OSA, на который применяется политика (**Пакеты** или **Контейнерные образы**);
- **Репозитории** – список репозиториев с плагином OSA, на которые применяется политика;
- **Уровень** — уровень критичности политики (не влияет на действия в системе);
- **Блокер** — блокировка сборки или загрузки компонента из прокси-репозитория;
- **Отложенная блокировка** — задержка (в днях) от первого срабатывания политики до блокировки;
- **Активно** — состояние политики;
- **Описание** — описание политики;
- **Условия** — перечень условий в политике.

Далее настраиваются условия срабатывания политики, поддерживаются следующие параметры:

- **PURL** — package URL, идентификатор компонента;
- **Название зависимости** — имя используемого компонента;
- **Версия зависимости** — конкретная версия компонента, обнаруженная в проекте;
- **Автор зависимости** — имя или организация, указанные как автор компонента;
- **Дата публикации зависимости** — дата первого появления версии зависимости в открытых источниках;
- **Возраст зависимости (в днях)** — количество дней с момента публикации зависимости;
- **Количество уязвимостей в зависимости** — общее число уязвимостей, связанных с компонентом;
- **Зависимость опасна** – зависимость считается опасной, если у нее есть уязвимости с:
  - префиксом `MAL-` (фид вредоносных пакетов OSV);
  - одним из следующих CWE:
    - CWE-506: Внедренный вредоносный код;
    - CWE-507: Троянское ПО;
    - CWE-509: Нераспространяющийся вредоносный код;
    - CWE-509: Распространяющийся вредоносный код (вирус или червь);
    - CWE-510: Скрытые средства несанкционированного доступа;
    - CWE-511: Логическая или временная бомба;
    - CWE-512: Шпионское ПО;
    - CWE-912: Скрытые функции;
  - или одним из следующих значений поля Импакт в базе Kaspersky OSS Threats Data Feed:
    - Вредоносное ПО;
    - Другое влияние.
- **Зависимость является протестным ПО** – зависимость считается протестным ПО, если у нее есть уязвимости из проприетарного фида protestware;

- **Зависимость является ПО-вымогателем** – зависимость считается ПО-вымогателем, если у нее есть уязвимости, используемые в ПО-вымогателе по данным CISA-KEV;
- **Зависимость отозвана** - зависимость отозвана из источника;
- **Зависимость является потомком** – поиск на нижестоящих уровнях транзитивных зависимостей, относящихся к указанному родительскому компоненту. Поиск зависимости осуществляется для более глубоких уровней вложенности графа зависимостей. Например, для случая цепочки зависимостей a<-b<-c<-d потомками b будут c и d;
- **Глубина транзитивности зависимости** - контролирует глубину поиска зависимостей, где 1 - прямая зависимость, 2 и более - транзитивные. Допускается указывать только натуральные числа;
- **Технология** — язык программирования или экосистема;
- **Лицензия** – SPDX-идентификатор лицензии;
- **Категория лицензии** — классификация лицензии по типу (например, copyleft, permissive);
- **ID уязвимости** — идентификатор уязвимости;
- **Оценка CVSS2** — численная оценка угрозы по стандарту CVSS 2;
- **Уровень угрозы CVSS2** — уровень угрозы по стандарту CVSS 2;
- **CVSS2 Access Vector (AV)** – путь эксплуатации уязвимости (физический или сетевой);
- **CVSS2 Access Complexity (AC)** – сложность эксплуатация уязвимости;
- **CVSS2 Authentication (Au)** – требования аутентификации для эксплуатации уязвимости;
- **CVSS2 Availability Impact (A)** – степень потери доступности данных;
- **CVSS2 Confidentiality Impact (C)** – степень потери конфиденциальности данных;
- **CVSS2 Integrity Impact (I)** – степень потери целостности данных;
- **Оценка CVSS3** — численная оценка угрозы по стандарту CVSS 3;
- **Уровень угрозы CVSS3** — уровень угрозы по стандарту CVSS 3;
- **CVSS3 Attack Vector (AV)** — вектор атаки;
- **CVSS3 Attack Complexity (AC)** — сложность атаки;
- **CVSS3 Priviliges Required (PR)** — требуемый уровень доступа для эксплуатации уязвимости;
- **CVSS3 User Interaction (UI)** — наличие взаимодействия с пользователем;
- **CVSS3 Scope (S)** — область безопасности компонента;
- **CVSS3 Confidentiality (C)** — степень потери конфиденциальности данных;
- **CVSS3 Integrity (I)** — степень потери целостности данных;
- **CVSS3 Availability (A)** — степень потери доступности данных;
- **Оценка CVSS4** — численная оценка угрозы по стандарту CVSS 4;
- **Уровень угрозы CVSS4** — уровень угрозы по стандарту CVSS 4;
- **CVSS4 Attack Vector (AV)** — вектор атаки;
- **CVSS4 Attack Complexity (AC)** — сложность атаки;
- **CVSS4 Attack Requirements (AT)** — наличие требований к атаке;
- **CVSS4 Privileges Required (PR)** — требуемый уровень доступа для эксплуатации уязвимости;

- **CVSS4 User Interaction (UI)** — наличие взаимодействия с пользователем;
- **CVSS4 Confidentiality Impact to the Vulnerable System (VC)** — влияние на конфиденциальность уязвимой системы;
- **CVSS4 Confidentiality Impact to the Subsequent System (SC)** — влияние на конфиденциальность последующей системы;
- **CVSS4 Integrity Impact to the Vulnerable System (VI)** — влияние на целостность уязвимой системы;
- **CVSS4 Integrity Impact to the Subsequent System (SI)** — влияние на целостность последующей системы;
- **CVSS4 Availability Impact to the Vulnerable System (VA)** — влияние на доступность уязвимой системы;
- **CVSS4 Availability Impact to the Subsequent System (SA)** — влияние на доступность последующей системы;
- **Дата публикации уязвимости** — дата, когда уязвимость была впервые опубликована;
- **Дата обновления уязвимости** — дата последнего обновления информации об уязвимости;
- **Результат эксплуатации уязвимости (Kaspersky)** – поле Импакт в базе Kaspersky OSS Threats Data Feed;
- **Уязвимость имеет эксплойт** – признак наличия публичного эксплойта в базе знаний уязвимостей (NVD, GHSA, БДУ ФСТЭК или другой);
- **Уязвимость имеет исправление** – признак наличия версии, на которую можно обновиться для устранения уязвимости;
- **Уязвимости достижима** – уязвимый метод компонента используется в исходном коде. Более подробно об анализе достижимость можно прочесть в документации агента Johnny.
- **Возраст уязвимости (в днях)** — количество дней с момента публикации уязвимости;
- **Окружение** — окружение разработки;
- **Способ обнаружения** — способ обнаружения зависимости (по манифесту, содержанию проекта или в результате разрешения зависимостей);
- **Связь** — связь зависимости в проекте (прямая или транзитивная);
- **CWE** — идентификатор типа уязвимости по стандарту Common Weakness Enumeration
- **Категория протестного ПО** — категория протестной узявимости;
- **SSVC2 Эксплуатация** — эксплуатируемость уязвимости;
- **SSVC2 Автоматизируемо** — автоматизируемость уязвимости;
- **SSVC2 Техническое влияние** — техническое влияние на уязвимую систему;
- **EPSS процент** — вероятность использования уязвимости в реальных условиях в ближайшие 30 дней;
- **EPSS процентиль** — доля уязвимостей с таким же или более низким баллом.

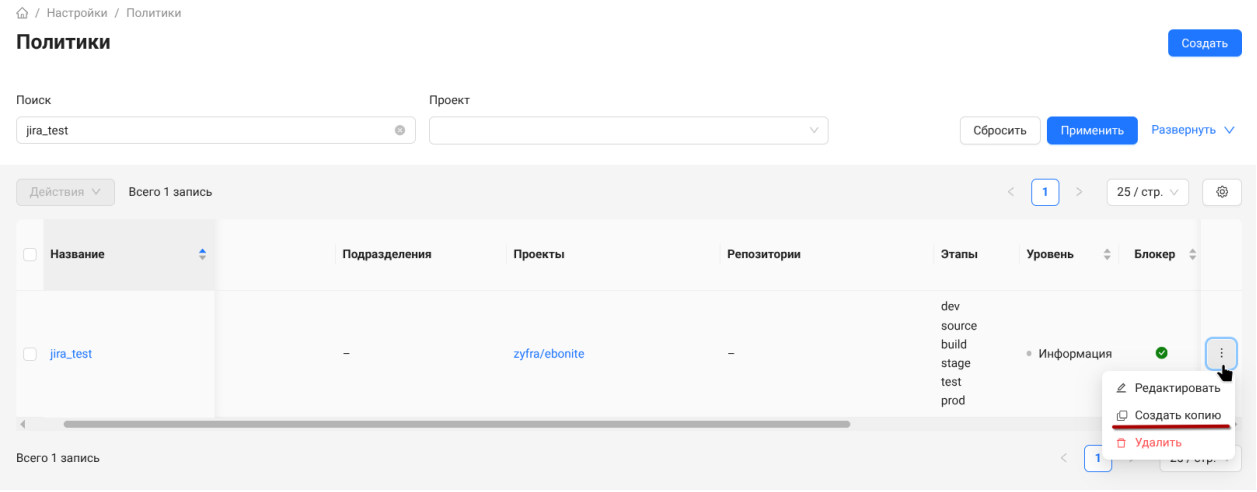
Есть возможность задать списковые значения для следующих параметров:

- **Список PURL**
- **Список названий зависимостей**

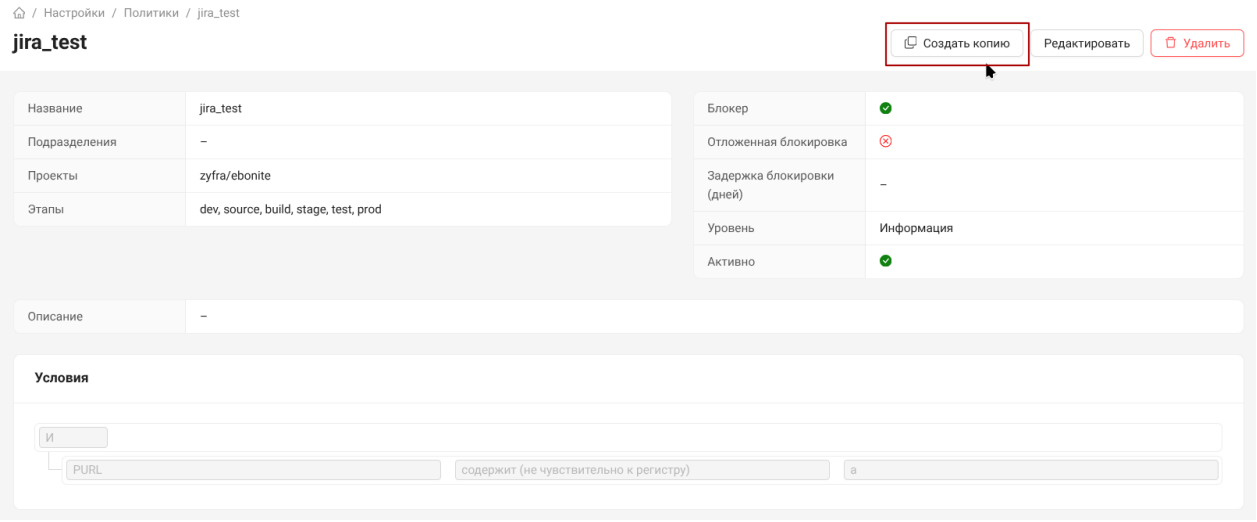
- **Список email авторов зависимостей**
- **Список ID уязвимостей (CVE)**
- **Список окружений**
- **Список технологий**
- **Список лицензий**
- **Список категорий лицензий**
- **Список CWE**
- **Список уровней угрозы CVSS2**
- **Список уровней угрозы CVSS3**
- **Список уровней угрозы CVSS4**
- **Список категорий протестного ПО**

#Создание копии политики

При необходимости продублировать уже имеющуюся политику можно воспользоваться пунктом контекстного меню **Создать копию**.



Или на форме политики нажать кнопку **Создать копию**.



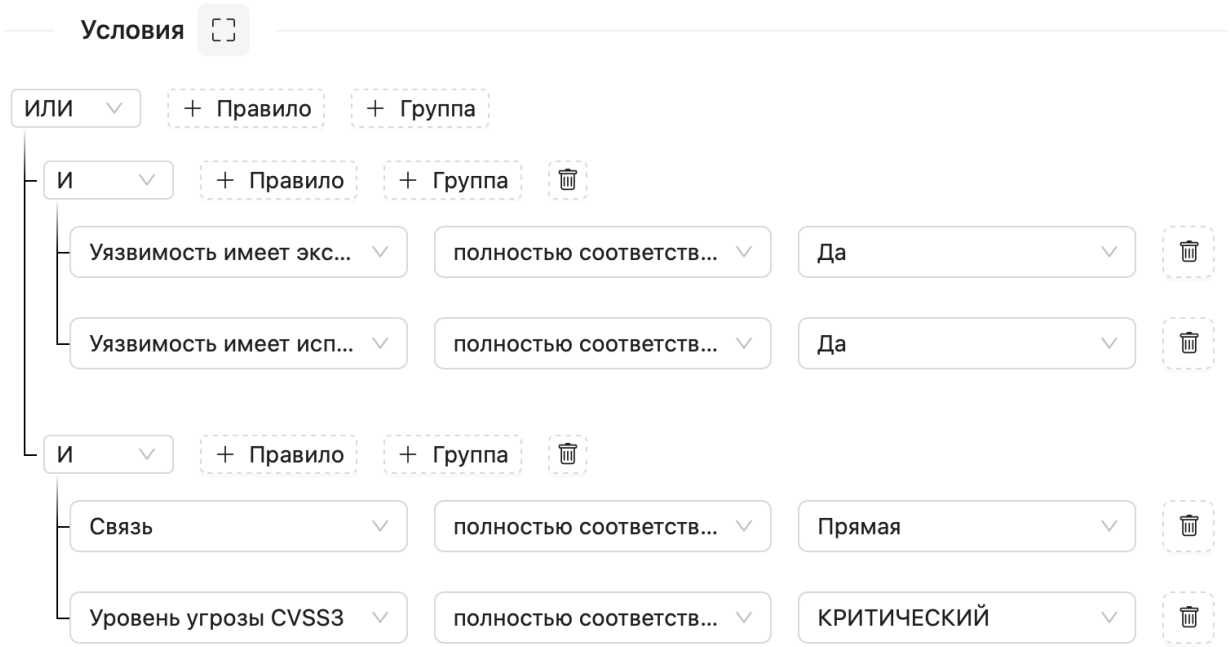
В случае создания копии политики выполняется создание новой политики с тем же описанием, условиями и связанными с ней действиями.

#Пример политики

Условия политики можно объединять в группы с помощью логических выражений **И/ИЛИ**. Группы не имеют ограничений по уровню вложенности и количеству условий.

Например, можно задать условия политики для следующего сценария – либо зависимость содержит уязвимость с эксплойтом и рекомендацию по исправлению, либо зависимость является директивной и содержит критическую уязвимость по стандарту CVSS 3.

Для создания такой политики необходимо добавить две группы, объединенные выражением **ИЛИ**. Это значит, что политика сработает при соответствии любой из перечисленных групп условий. Внутри группы задаются условия, объединенные выражением **И**.



Политика становится активной сразу после создания по нажатию кнопки **Создать**. Для созданной политики можно настроить действия при ее срабатывании: уведомление на почту или создание задачи в Jira или Kaiten.

Важно: политики срабатывают во время анализа, поэтому важно их создать до запуска анализа.

Рекомендация: если оставить поля Подразделения, Группы и Проекты пустыми, политика будет применяться для всех активных проектов в системе.

#Игнорирование политик

Созданные политики можно временно или навсегда проигнорировать при анализе. Условие игнорирования позволит оставить политики в системе, при этом не получая алертов о ее срабатывании, например в случае если уязвимость в компоненте не применима для конкретного проекта.

#Создание игноров

Создание и настройка условий игнорирования происходит в разделе `Настройки -> Игноры политики`.

Для создания условия игнорирования для одной или нескольких политик необходимо нажать на кнопку **Создать** и заполнить следующие поля:

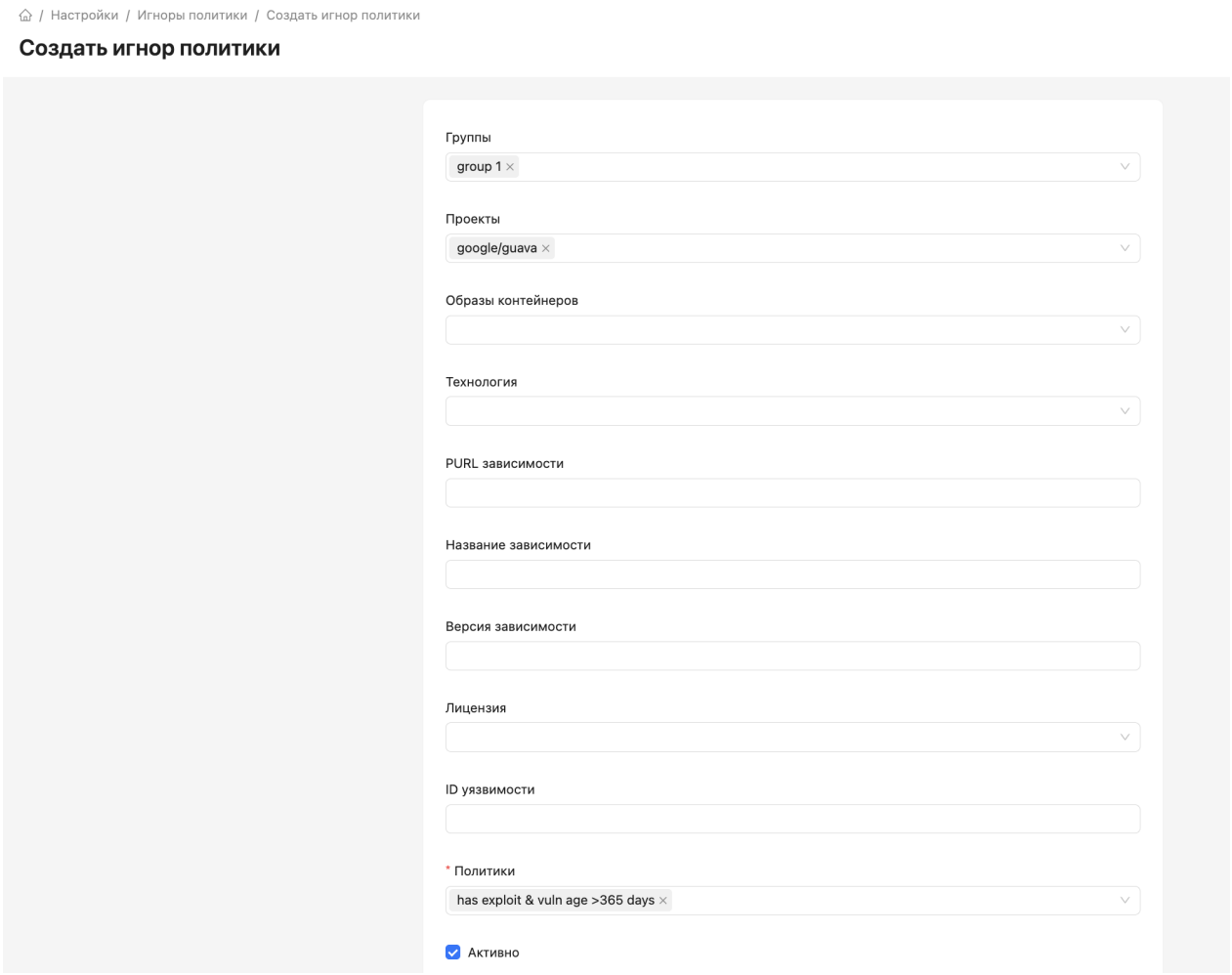
- **Группы** – группы проектов, на которые применяется игнор;
- **Проекты** — проекты, на которые применяется игнор;
- **Образы контейнеров** – образы в реестрах, на которые применяется игнор;
- **Технология** - язык программирования или экосистема;
- **PURL зависимости** — package URL, идентификатор компонента;
- **Название зависимости**;
- **Версия зависимости**;
- **Лицензия**;
- **ID уязвимости** - идентификатор уязвимости;
- **Политики**;
- **Активно** - состояние игнора;
- **Дата активации** - дата, с которой игнор начнет работать;
- **Дата окончания** - дата, после которой игнор закончит работать;
- **Заметка**.

Игнорирование по частичному совпадению

Игнорирование алерта по PURL происходит методом частичного совпадения: если в указанном PURL не хватает какого-то компонента, сравнение будет считаться положительным.

Например, алерт по зависимости с PURL
`pkg:conda/setuptools@65.1.0?build=py310h2ec42d9_0&channel=conda-forge&subdir=osx-64&type=tar.bz2`
при игноре по PURL `pkg:conda/setuptools@65.1.0` будет успешно проигнорирован.

При этом, для `pkg:conda/setuptools@65.1.0?subdir=foo` указанный PURL зависимости не будет учтен как подходящий, потому что квалификатор `subdir` не совпадает.



#Результаты игнорирования

Политики, на которые был распространено условие игнорирование, отображаются на вкладке "Игнорированные" раздела Алерты.

Сработавшие активные политики также можно быстро проигнорировать из вкладки "Активные", используя кнопку **Игнорировать**.

Доступ

Пользователь может просматривать только игноры, которые связанны с доступными ему проектами.

#Алерты политик

#Раздел «Алерты»

Результаты работы политик отображаются в разделе Алерты. Раздел имеет три вкладки:

- **Активные** – список алертов по результатам последнего анализа (проекта, сборки или компонента в прокси-репозитории);
- **Игнорированные** – список проигнорированных алертов;

- **Решенные** – список алертов, которые были решены после последнего анализа (условие политики больше не актуально).

Причина срабатывания политики отображается в поле **Условия политики**, включая заданные условия и найденные данные о компоненте. Например, значение `django@4.2.2 has CVE-2024-38875, CVSS3 Score 7.5 >= 7.00` подразумевает, что политика блокировки компонентов с CVSS3 равным или выше 7.00 сработала на компоненте django версии 4.2.2 с оценкой уязвимости 7.5.

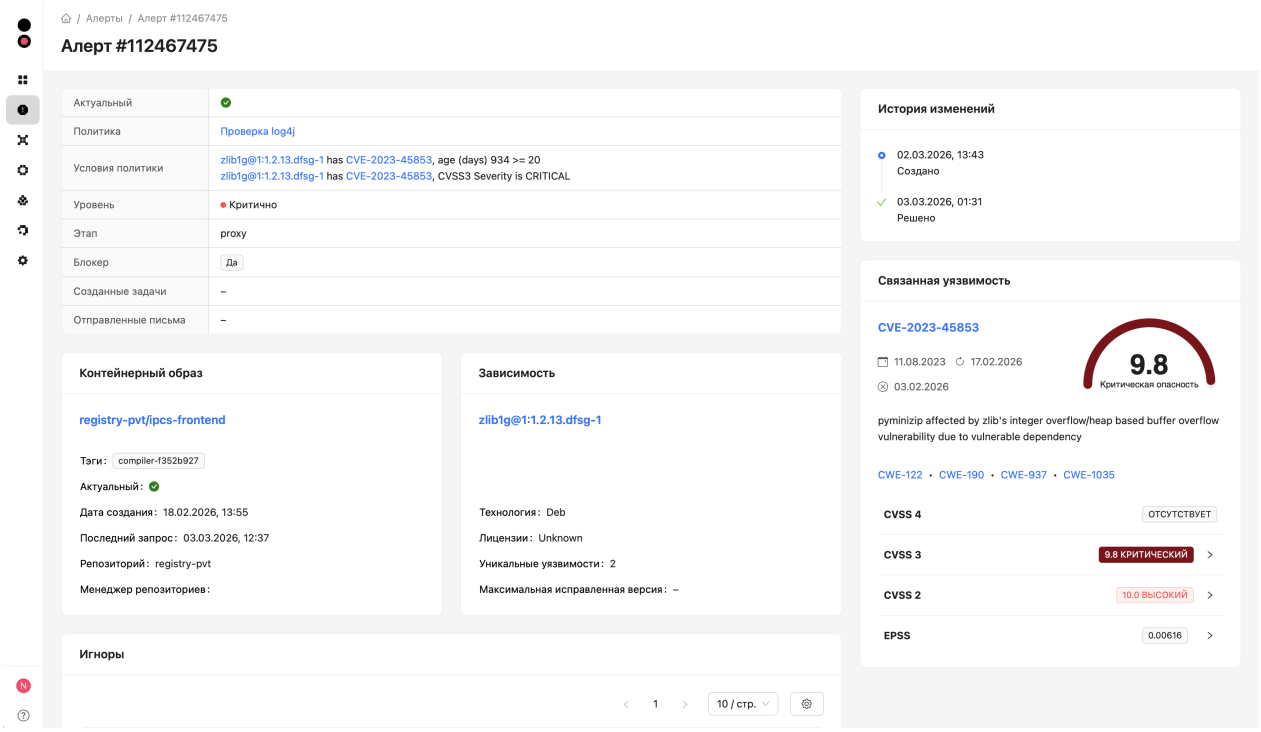
#Карточка алерта

Карточка алерта открывается по клику на алерт в списке. В верхней части страницы отображаются ключевые поля:

- **Актуальный** – актуальность алерта;
- **Политика** – политика, по которой сработал алерт (ссылка на карточку политики);
- **Условия политики** – совпавшие критерии, из-за которых сработала политика;
- **Уровень** – уровень критичности политики;
- **Этап** – стадия, на которой сработала политика;
- **Блокер** – признак блокирующей политики;
- **Созданные задачи** – список созданных задач (например, Jira);
- **Отправленные письма** – список получателей отправленных email-уведомлений.

Ниже показывается информация о связанных сущностях и детали, в зависимости от условий, с которыми сработал алерт:

- **Проект**/**Контейнерный образ**/**Пакет**/**Зависимость** – информация о компоненте и источнике (репозиторий, теги, лицензии);
- **Связанная уязвимость** – краткое описание, CVE, оценки CVSS и EPSS;
- **История** – события по алерту (время создания, время игнора, время решения).

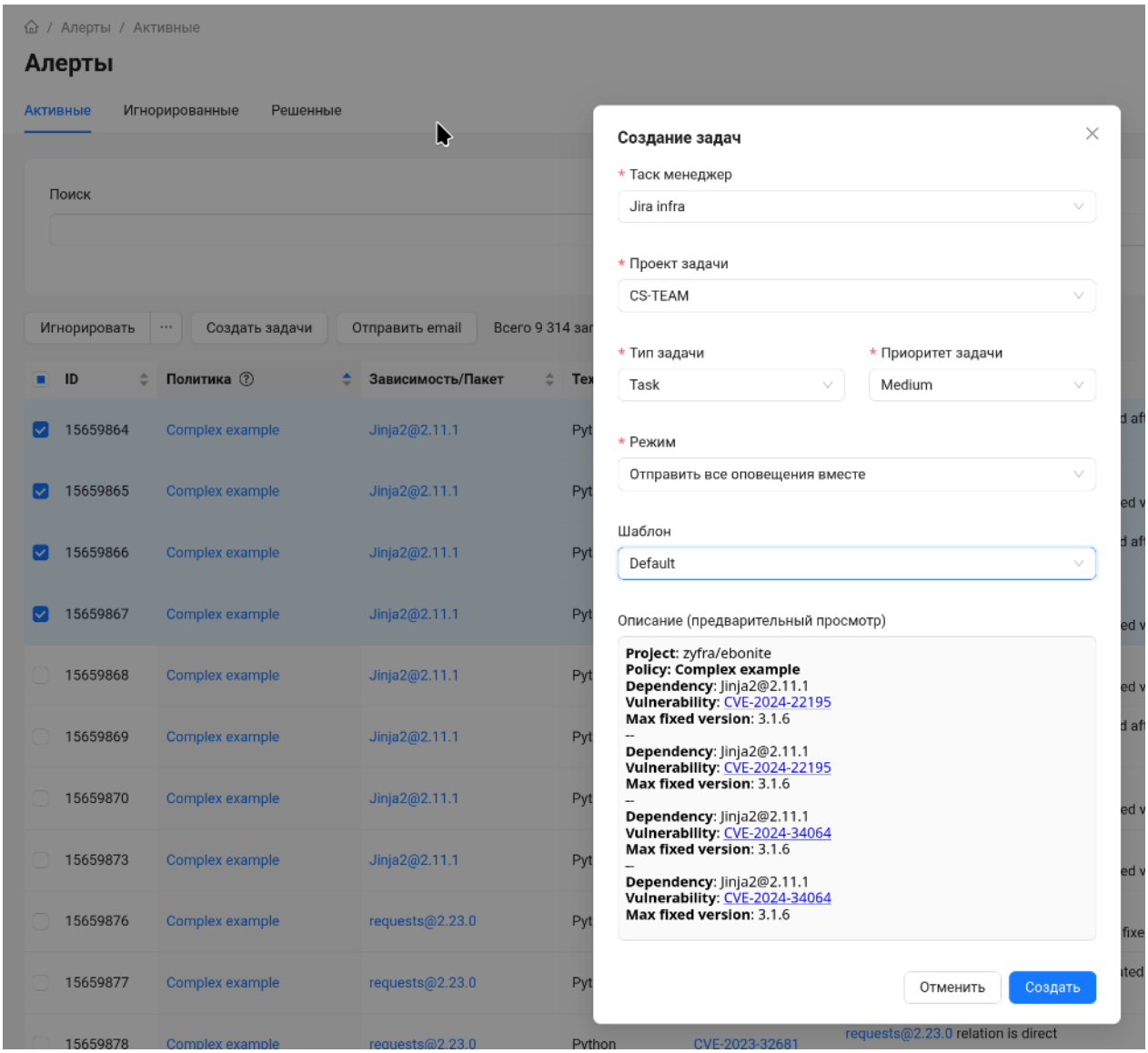


#Действия с алертами

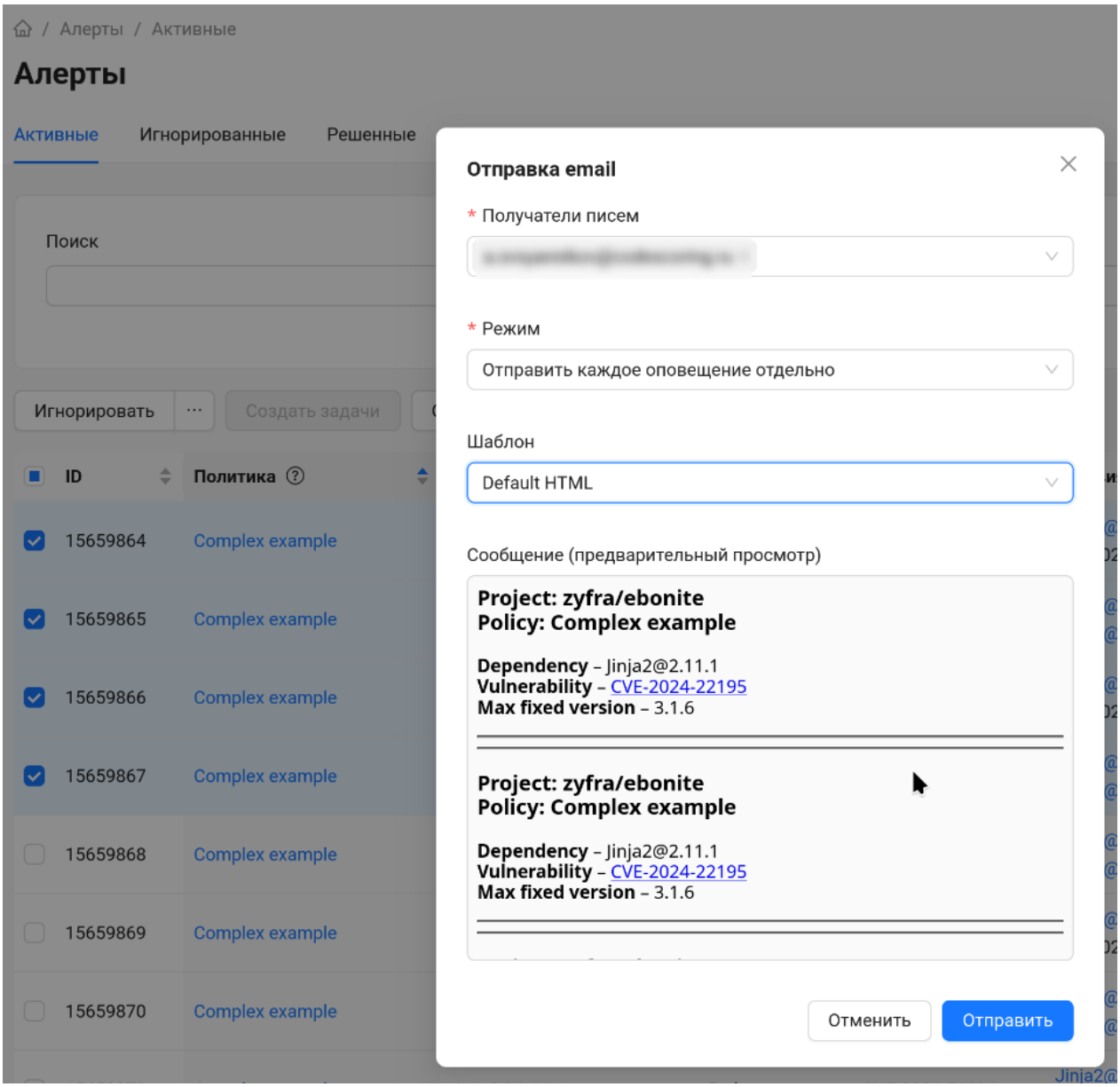
Для создания задачи или отправки email из списка алертов необходимо выбрать один или несколько алертов и нажать соответствующую кнопку.

Например, так:

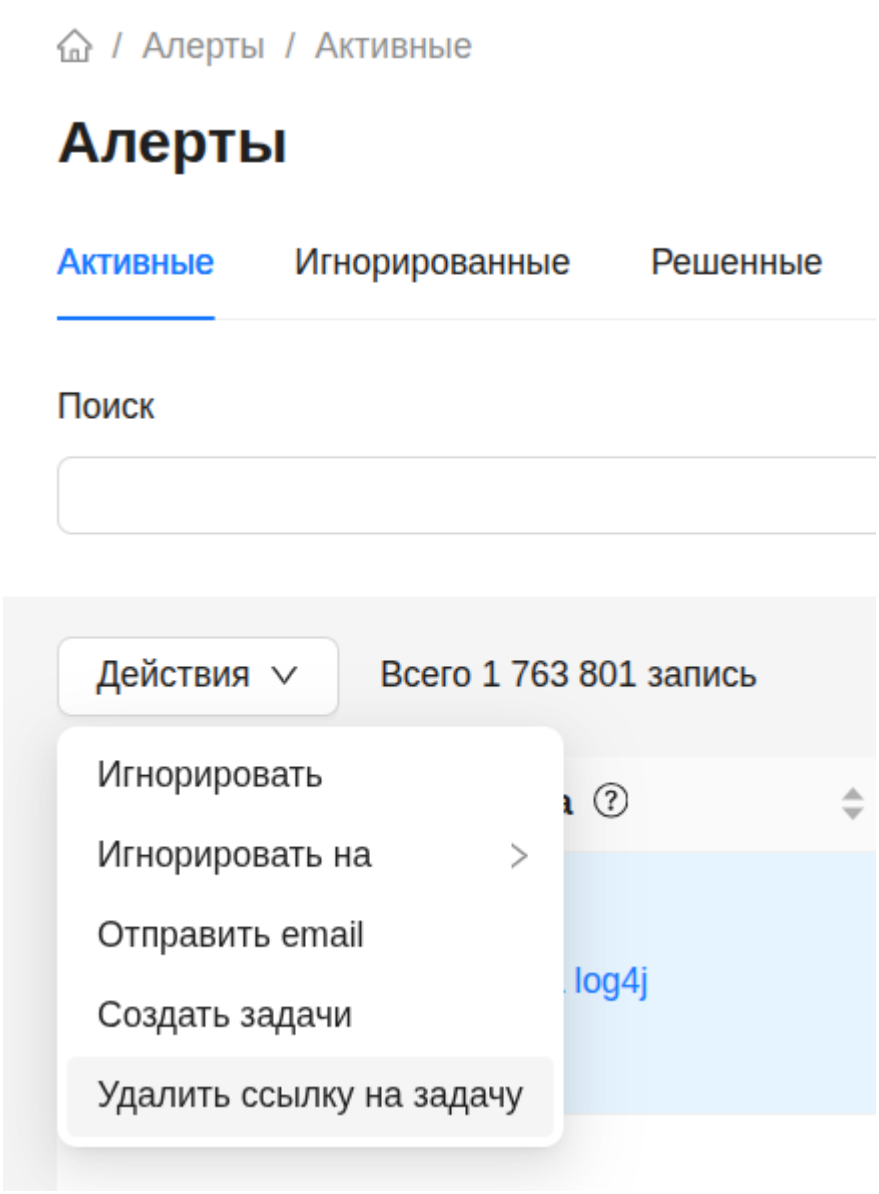
- создание задач



- отправка почтовых сообщений



Связанные задачи Jira можно отвязать, используя массовое действие Удалить ссылку на задачу.



#Подключение системы контроля версий

Для добавления в систему проектов (git-репозиториев) на анализ необходимо предварительно создать подключение к системе контроля версий (VCS). CodeScoring поддерживает следующие платформы:

- GitLab
- GitHub
- BitBucket (только Data Center и Server)
- Azure DevOps Git
- GitFlic
- Другие платформы, использующие Git

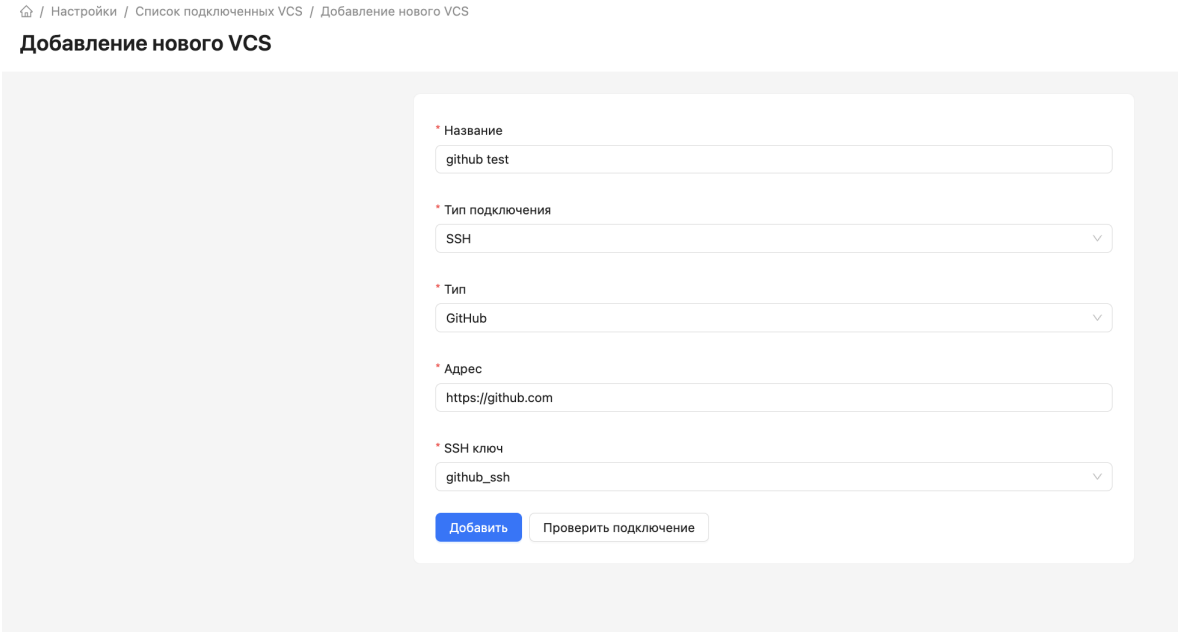
Интеграция происходит через два возможных механизма: Personal Access Token или ключ SSH. Для механизма Personal Access Token возможно подключение нескольких систем контроля версий с одним адресом, но разными токенами.

Важно: После подключения системы контроля версий невозможно поменять ее тип и адрес подключения.

Примечание: в случае, если для обращения к системе контроля версий используются приватные NS сервера, их необходимо указать в настройках платформы. Для получения соответствующих шаблонов необходимо обратиться к вендору.

#Добавление SSH-ключа

1. Скопировать существующий или сгенерировать новый приватный SSH ключ в системе контроля версий по инструкции для:
    - Gitlab
    - Github
    - BitBucket
    - Azure DevOps Git
2. В интерфейсе CodeScoring перейти в раздел `Настройки -> SSH ключи`
3. Нажать **Добавить** в правом верхнем углу.
4. Ввести название ключа в поле **Название** и сам приватный SSH ключ в поле **Приватный ключ**, после чего завершить добавление ключа.
5. Перейти в раздел `Настройки -> VCS`.
6. Нажать **Добавить** в правом верхнем углу.
7. Заполнить форму, как показано на скриншоте. SSH ключ выбирается из списка в поле **SSH key**.

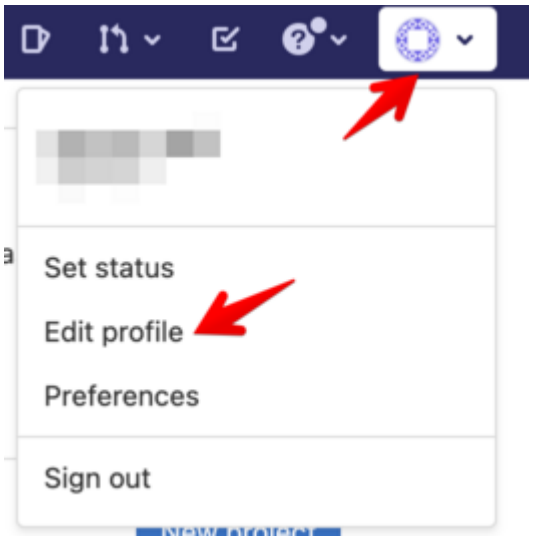


8. Проверить подключение можно по кнопке **Проверить подключение**. Для создания подключения необходимо нажать на кнопку **Добавить**.

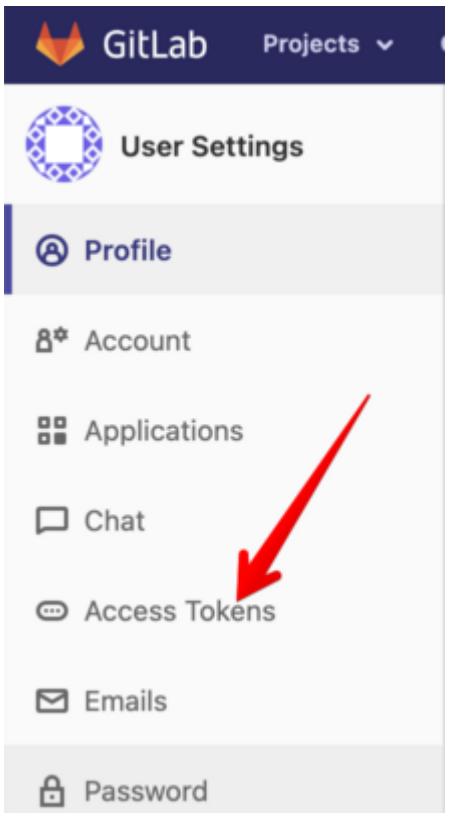
#Добавление токена для GitLab

Оригинальная инструкция для генерации токена на английском: https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html#create-a-personal-access-token

1. Войти в свой аккаунт в GitLab.
2. Через меню пользователя в правом верхнем углу перейти в раздел **Edit profile**.

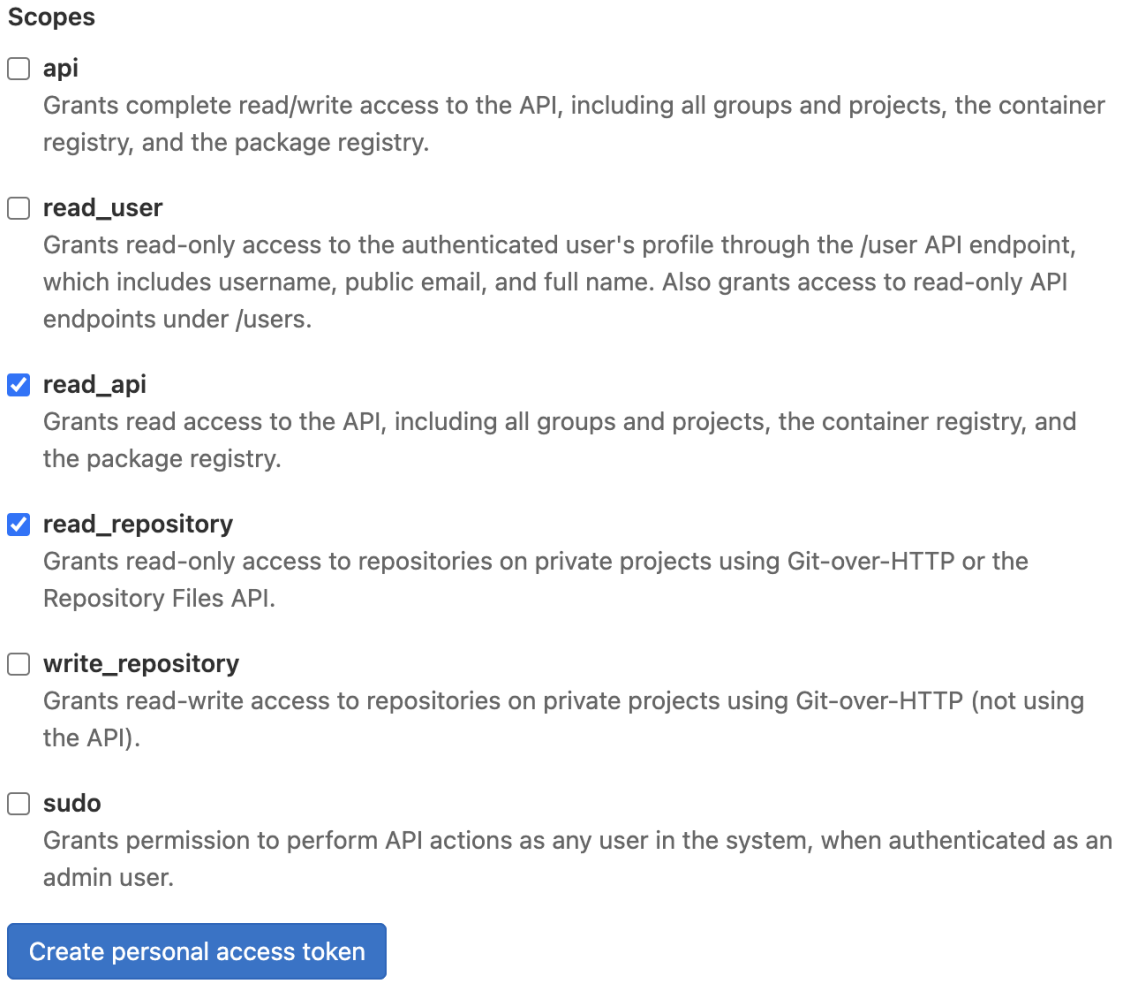


3. Далее в левом меню выбрать раздел **Access Tokens**.

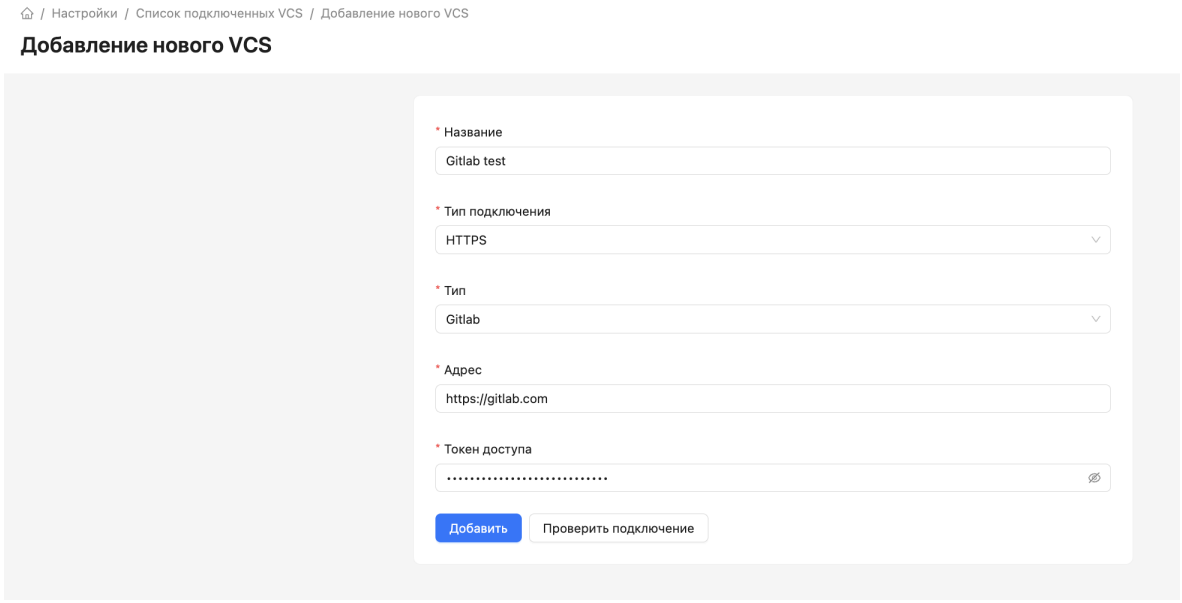


4. Задать название токену, например, "*codescoring-demo*", дату можно оставить пустой

5. В секции *scopes* выбрать **read_api** и **read_repository**.



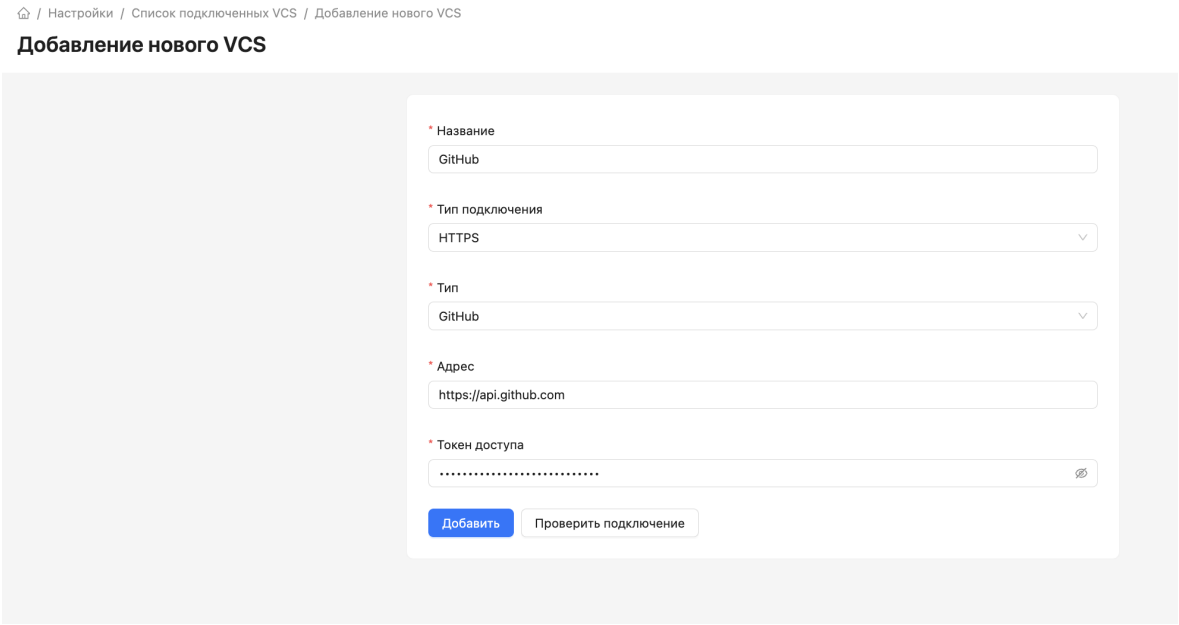
6. Нажать кнопку **Create personal access token**.
7. Скопировать сгенерированный токен.
8. В интерфейсе CodeScoring перейти в раздел `Настройки -> VCS`.
9. Нажать **Добавить** в правом верхнем углу.
10. Заполнить форму, как показано на скриншоте. Токен вставляется в поле **Токен доступа**.



#Добавление токена для GitHub

Оригинальная инструкция для генерации токена на английском: https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token

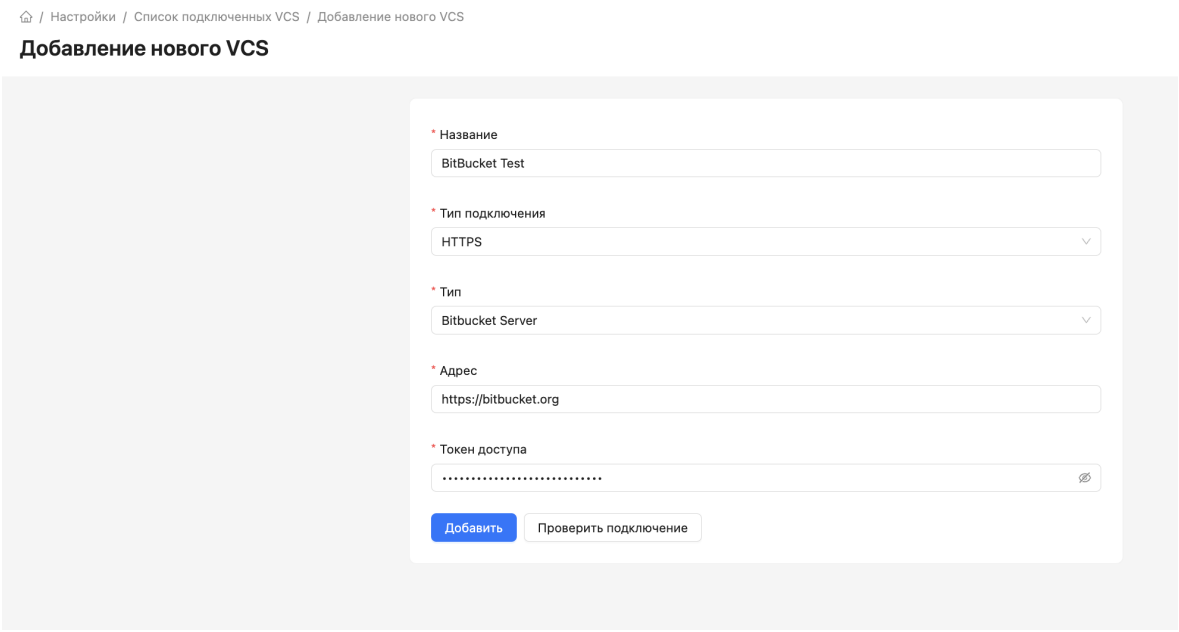
1. Войти в свой аккаунт в GitHub, для облачной версии логин доступен по ссылке https://github.com/login.
2. Если у аккаунта не верифицирован email, обязательно это сделать по инструкции.
3. Через меню пользователя в правом верхнем углу перейти в раздел **Settings**.
4. Далее в левом меню выбрать раздел **Developer settings**.
5. В левом меню выбрать раздел **Personal access tokens**.
6. Задать название токену, например, "codescoring-demo".
7. В секции Select scopes выбрать все опции списка *repos*.
8. Нажать кнопку **Generate token**.
9. Скопировать сгенерированный токен.
10. В интерфейсе CodeScoring перейти в раздел Настройки -> VCS.
11. Нажать **Добавить** в правом верхнем углу.
12. Заполнить форму, как показано на скриншоте. Токен вставляется в поле **Токен доступа**.



#Добавление токена для BitBucket Data Center и Server

Оригинальная инструкция для генерации токена на английском: https://confluence.atlassian.com/bitbucketserver072/personal-access-tokens-1005335924.html

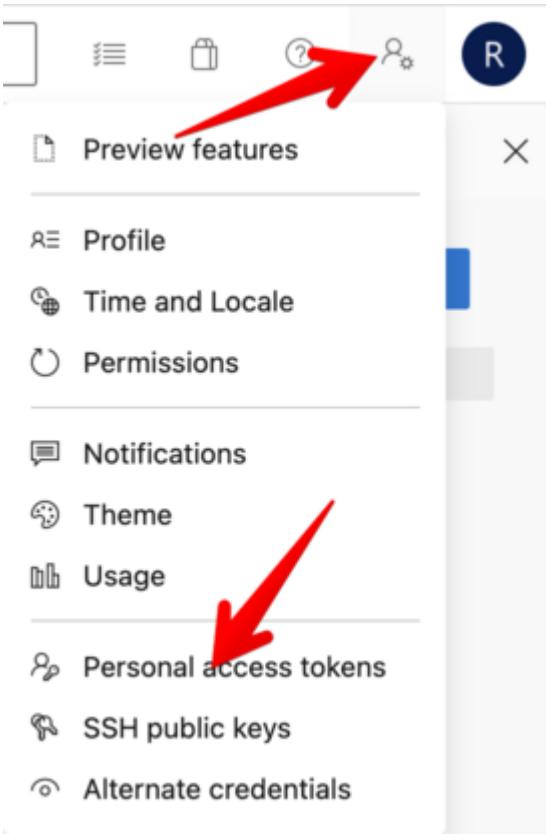
1. Войти в свой аккаунт в BitBucket.
2. Через меню пользователя в правом верхнем углу перейти в раздел **Manage account**.
3. Далее в левом меню выбрать раздел **Personal access tokens**.
4. Нажать на **Create token**.
5. Задать название токену, например, "codescoring-demo".
6. В секции **Permissions** дать права на чтение для проектов и репозиториев.
7. В секции **Expiry** при желании задать срок жизни токена.
8. Нажать кнопку **Create**.
9. Скопировать сгенерированный токен.
10. В интерфейсе CodeScoring перейти в раздел `Настройки -> VCS`.
11. Нажать **Добавить** в правом верхнем углу.
12. Заполнить форму, как показано на скриншоте. Токен вставляется в поле **Токен доступа**.



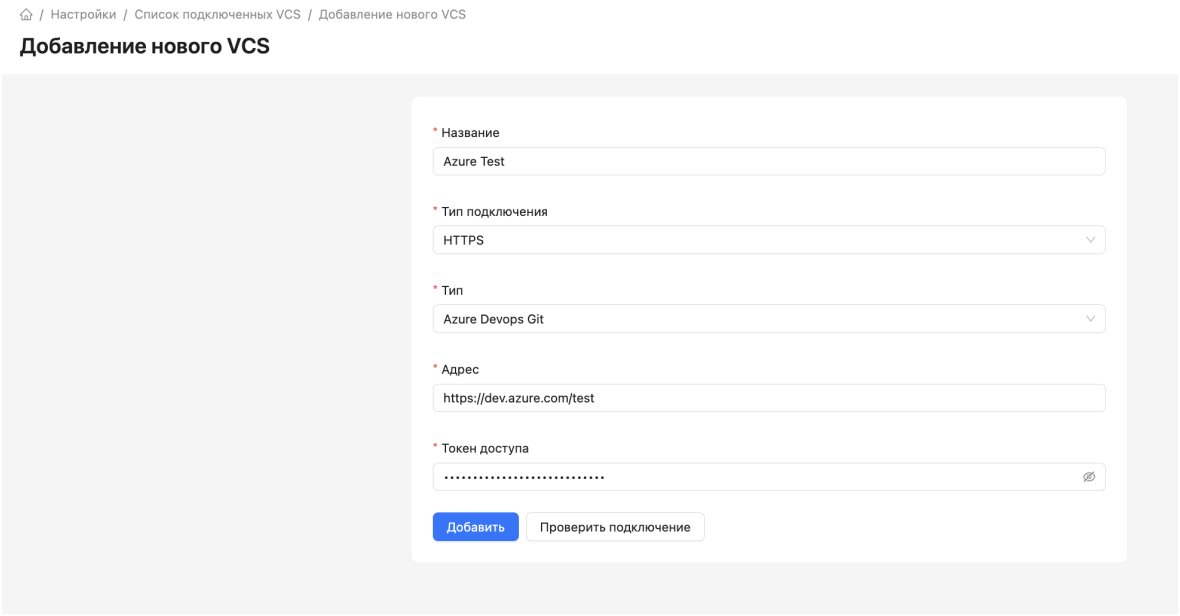
#Добавление токена для Azure DevOps Git

Оригинальная инструкция для генерации токена на английском: https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate?view=azure-devops

1. Войти в свой аккаунт в Azure DevOps.
2. Через меню пользователя в правом верхнем углу перейти в раздел **Personal access tokens**.



3. Далее нажать кнопку **New token**.
4. Задать название токену, например, "codescoring-demo", и срок действия токена.
5. В секции *Scopes* обязательно отметить доступ на **Read** для сущностей **Code** и **Identity**.
6. Нажать кнопку **Create**.
7. Скопировать сгенерированный токен.
8. В интерфейсе CodeScoring перейти в раздел `Настройки -> VCS`.
9. Нажать **Добавить** в правом верхнем углу.
10. Заполнить форму, как показано на скриншоте. Токен вставляется в поле **Токен доступа**.



#Подключение GitFlic через ssh

1. Создайте новую пару ключей ssh без passphase с типом **RSA_SHA512**,**SSH_ED25519** или **ECDSA-SHA2-nistp256**.
   `bash ssh-keygen -t ed25519 -N "" -q -f /tmp/id25519`
2. Перейдите в веб-интерфейс GitFlic в раздел `Настройки -> Ключи` и добавьте **публичный** ключ в пул ключей GitFlic.

   Посмотреть содержимое публичного ключа можно при помощи следующей команды: `bash cat /tmp/id25519.pub`
3. Перейдите в интерфейс CodeScoring в раздел `Настройки -> SSH ключи`, откройте форму добавления ключа по кнопке **Добавить** и добавьте **приватный** ключ в пул ключей CodeScoring.

Посмотреть содержимое приватного ключа можно при помощи следующей команды: `bash cat /tmp/id25519`

4. Перейдите в раздел `Настройки -> VCS`, откройте форму добавления VCS по кнопке **Добавить** и заполните поля создания VCS инстанса:
    - **Название** – название VCS подключения;
    - **Тип подключения** – тип клонирования репозитория (выбрать тип SSH);
    - **Тип** – тип инстанса VCS (выбрать тип GitFlic);
    - **Адрес** – адрес, по которому доступен VCS;
    - **SSH key** – ключ SSH, созданный в пункте 3.
5. Перейдите в раздел `Настройки -> Проекты`, откройте форму создания нового проекта по кнопке **Создать** и заполните поля создания проекта:
    - **Репозиторий** – ссылка на репозиторий в GitFlic;
    - **VCS** – название инстанса VCS, созданного в пункте 6;
    - **Название** – название проекта в CodeScoring.
6. Сохраните проект по кнопке **Создать**.

#Синхронизация данных из VCS

CodeScoring автоматически синхронизирует данные из систем контроля версий при добавлении проекта, запуске анализа и в рамках фоново выполняемых задач. Ниже описаны все механизмы получения и обновления данных из `git`:

1. При добавлении VCS-проекта, CodeScoring создаёт локальную копию репозитория в режиме `bare` и сохраняет её в каталоге проекта.
2. При запуске SCA, TQI или Secrets анализа система загружает актуальное состояние ветки, указанной в настройках проекта.
3. Раз в месяц CodeScoring инициирует фоновую задачу, которая синхронизирует все подключённые VCS-проекты с удалёнными репозиториями. Это профилактическое обновление, которое не заменяет синхронизацию при запуске анализа, а лишь ускоряет его на проектах, где анализ давно не выполнялся.
    - Выполняется `git fetch` для всех VCS-проектов
    - Задачи распределяются с помощью инструмента Huey так, чтобы равномерно распределить нагрузку по оставшимся дням месяца

Примечание: при необходимости досрочной синхронизации репозитория можно использовать ручной запуск обновления кода на странице проекта в разделе `Настройки -> Проекты`.

#Управление проектами

Проект в CodeScoring – это часть анализируемой кодовой базы. В системе возможно создать два типа проектов:

- **VCS-проект** – связан с репозиторием в системе контроля версий;
- **CLI-проект** – не имеет привязки к репозиторию и позволяет сохранять результаты сканирования от консольного агента johnny или загрузить готовый SBOM-файл.

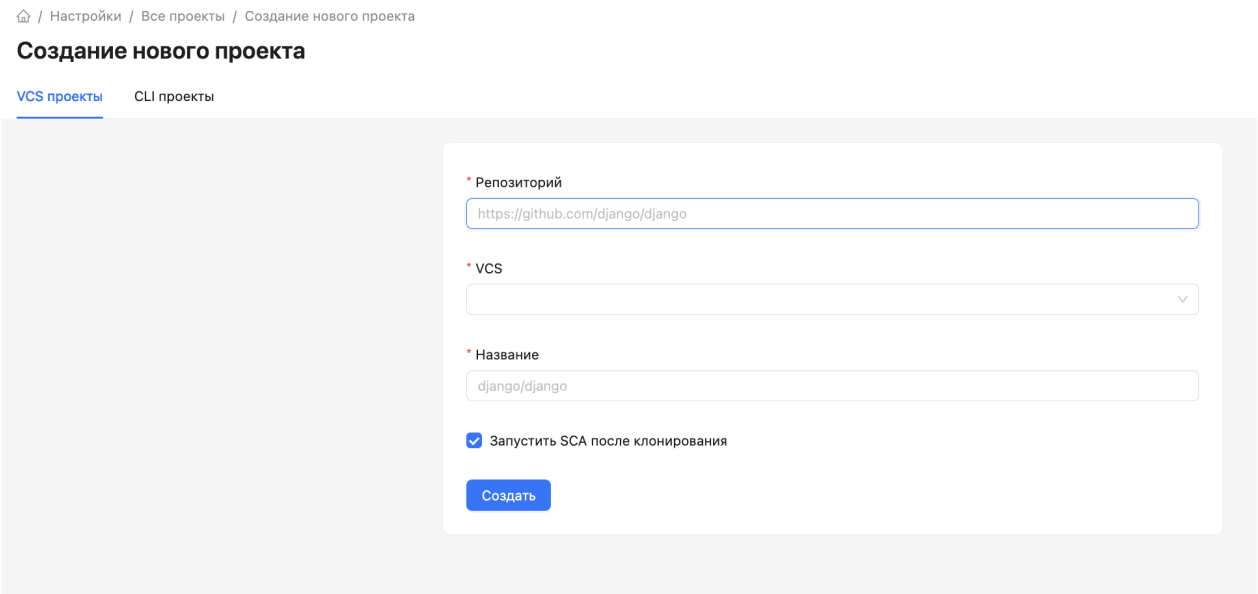
Управление проектами происходит в разделе `Настройки -> Проекты`.

#Создание VCS-проекта

Важно

Создать VCS-проект получится только после создания подключения к системе контроля версий.

Для перехода на форму создания VCS-проекта необходимо нажать на кнопку **Создать** и выбрать вкладку **VCS проекты**. Выбранный при создании тип проекта нельзя перевести в другой.



1. Для добавления проекта необходимо добавить в форме ссылку на репозиторий, выбрать соответствующую систему контроля версий из списка, задать название проекта и опцию запуска SCA анализа сразу после клонирования.
2. После добавления происходит первоначальное клонирование проекта, время которого будет зависеть от размера репозитория.
3. После клонирования проект будет доступен для редактирования и новых анализов.

Система принимает ссылки на репозитории в следующих форматах:

- GitLab
  - `<http/https>://<GitLab Server URL>/<group>/<project>`
- GitHub
  - `https://github.com/<username/organisation>/<project>`
- BitBucket
  - `<http/https>://<Bitbucket Server URL>/scm/<group>/<project>`
- Azure DevOps Git
  - `https://<organisation>.visualstudio.com/<group>/_git/<project>`
  - `https://dev.azure.com/<organisation>/<group>/_git/<project>`
  - `<http/https>://<Azure Server URL>/<organisation>/<group>/_git/<project>`

После добавления репозитория на странице настроек VCS-проекта доступно ручное обновление кода проекта по кнопке **Обновить код проекта**.

Внимание! Если анализируемый проект коммерческий, то рекомендуется указать для него категорию лицензии **Commercial License** для корректной работы политики по лицензионной совместимости компонентов.

#Создание CLI-проекта

Для перехода на форму создания CLI-проекта необходимо нажать на кнопку **Создать** и выбрать вкладку CLI проекты.

Для добавления проекта достаточно заполнить его название в поле **Название**.

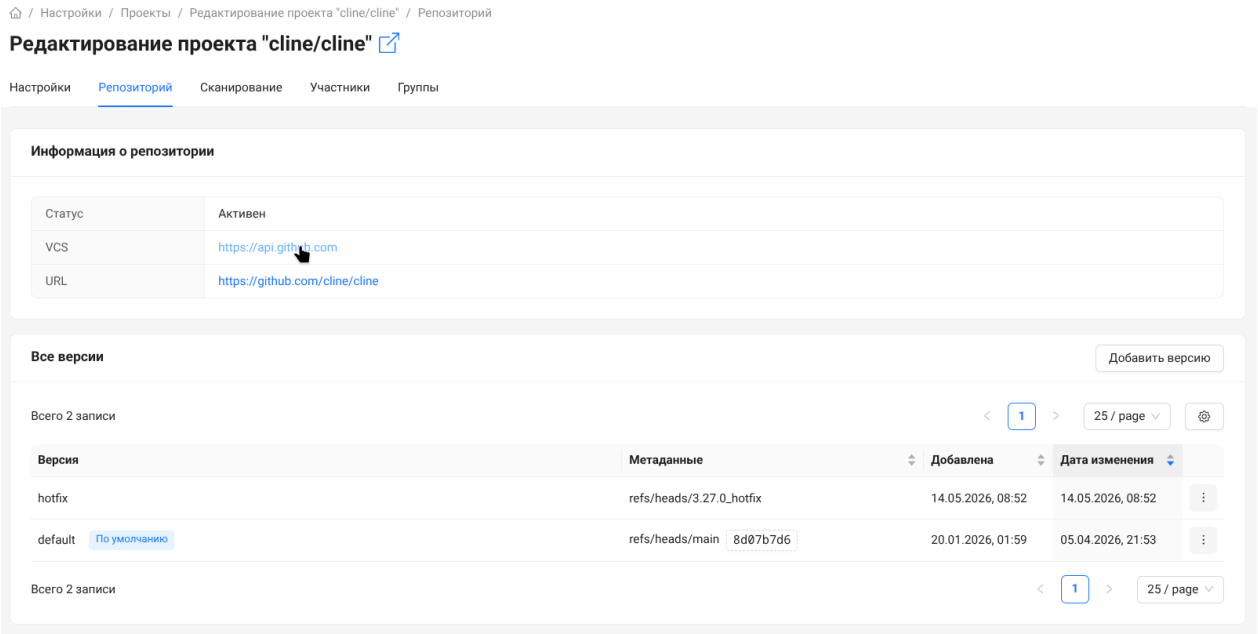
#Создание категорий проектов

Категории используются для группировки проектов системы по смысловым группам.

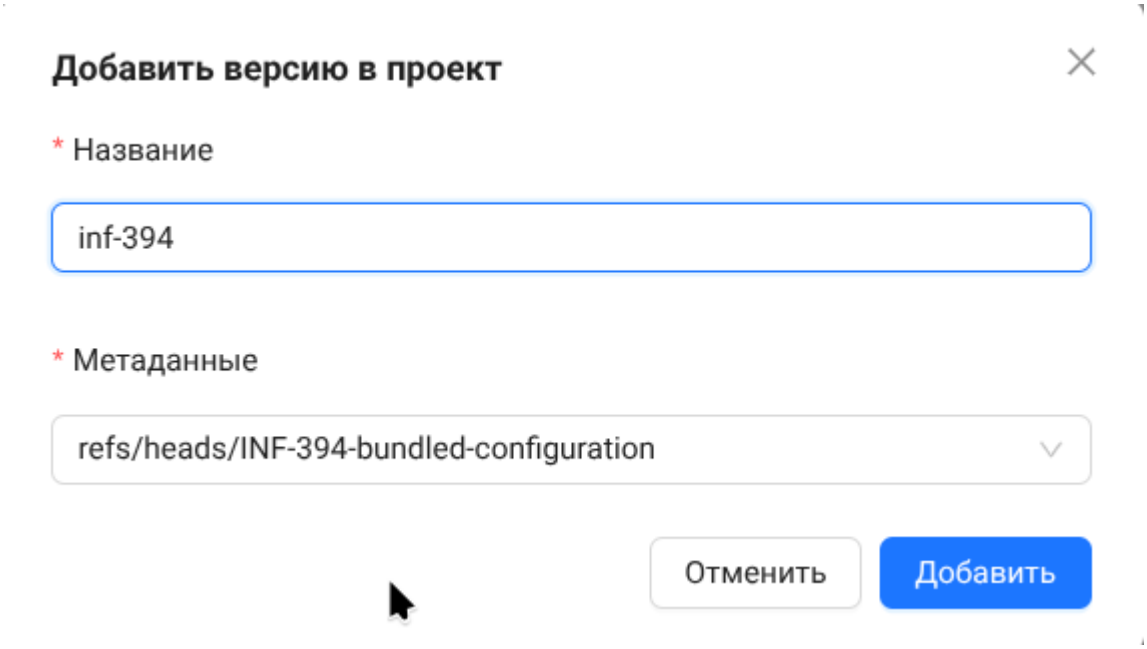
Управление категориями происходит в разделе `Настройки -> Категории`. Перейти на форму создания категории можно по кнопке **Создать**. Для создания категории достаточно задать ей название.

#Управление версиями

Управление версиями происходит в настройках проекта, в разделе `Репозиторий`.

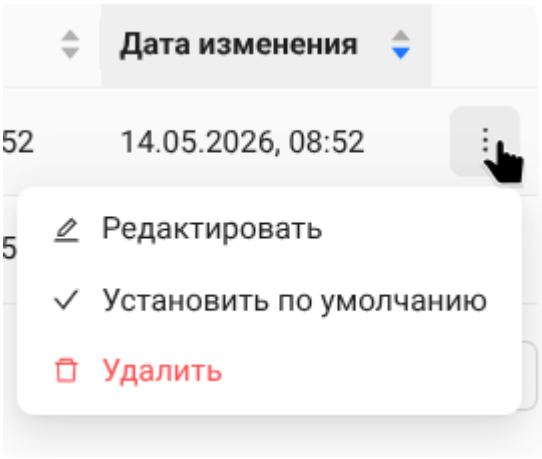


Добавить новую версию можно по кнопке **Добавить версию**. Для создания версии в CLI проекте достаточно задать ей название, а для VCS проекта необходимо указать метаданные тега или вертки репозитория.



Добавить новую версию также можно при запуске сканирования проекта, воспользовавшись соответствующим меню.

Установить версию по умолчанию можно в контекстном меню по кнопке **Установить по умолчанию**.



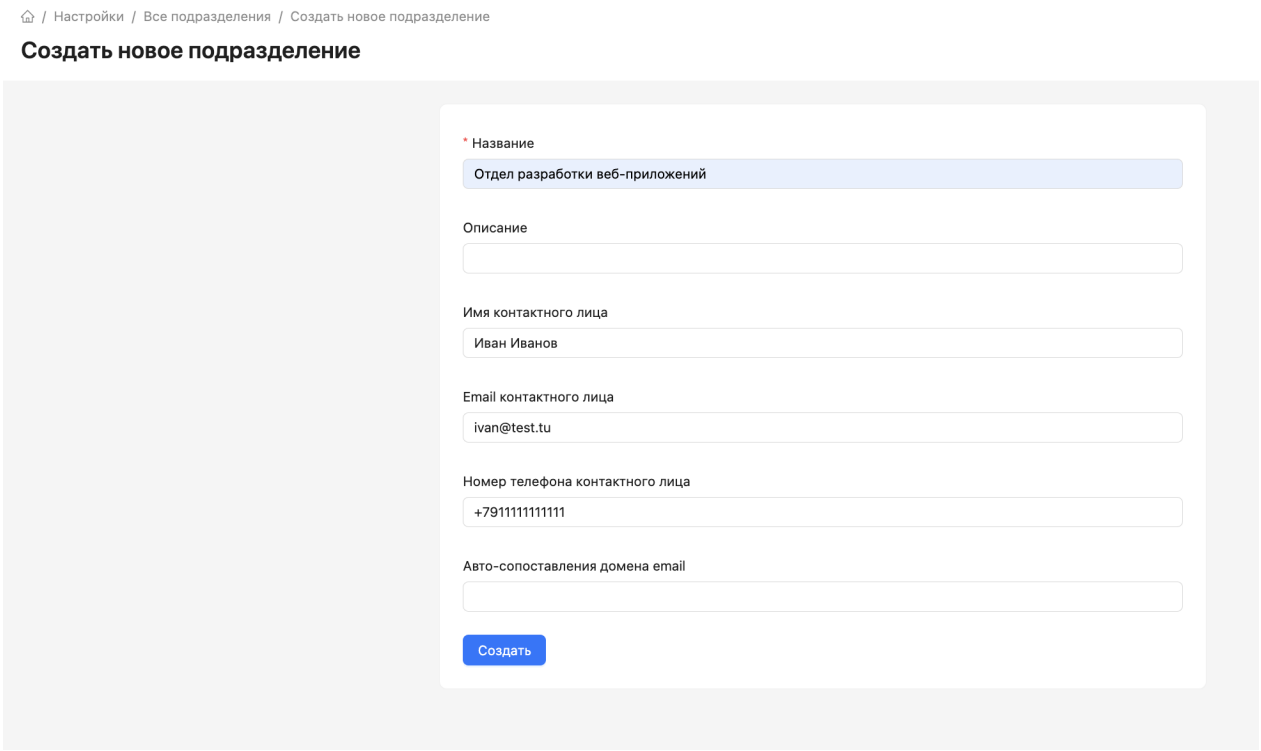
Удалить версию можно в контекстном меню записи версии по кнопке **Удалить**. Удалить версию по умолчанию нельзя.

При редактирование версии можно изменить название и метаданные.

#Настройка подразделений

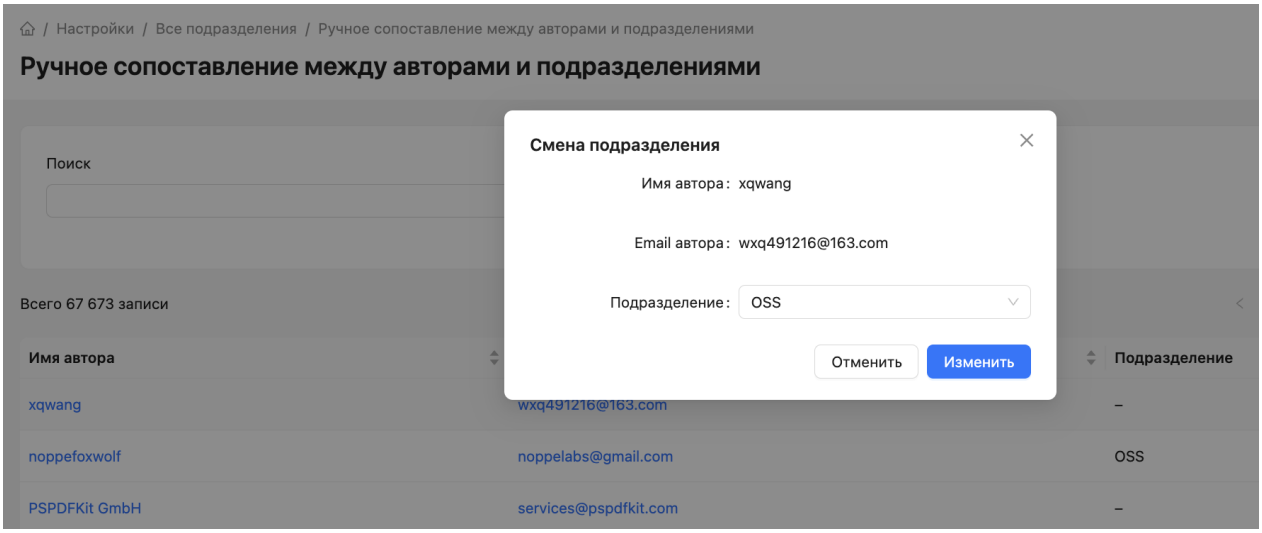
Подразделения — это абстрактные сущности в рамках организации. Они упрощают группировку проектов по принадлежности к разным группам ответственных пользователей.

Управление подразделениями происходит в разделе `Настройки -> Подразделения`. Перейти на форму создания подразделения можно по кнопке **Создать**. В форме необходимо обязательно заполнить поле названия подразделения **Название** и опциональные поля по желанию.



#Привязка авторов

В рамках системы имеется возможность создать связь между автором кода и конкретным подразделением. Для этого необходимо перейти на список авторов по кнопке **Изменить сопоставление авторов** и выбрать нужное подразделение в поле **Подразделение**.



#Настройка уведомлений

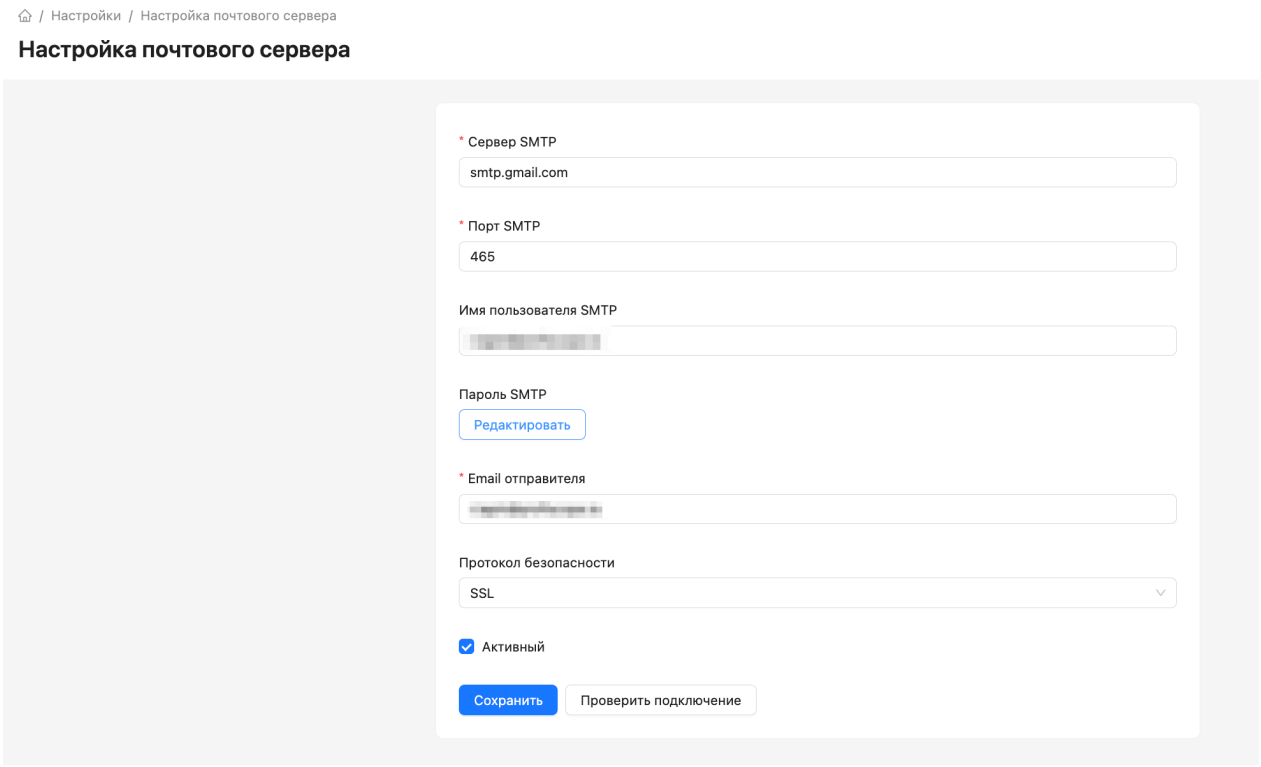
Для каждой политики можно настроить дополнительные уведомления о срабатывании политик, помимо просмотра результатов в разделе Алерты. На данный момент доступно три способа оповещения: через **email** и через таск-менеджеры **Jira** и **Kaiten**.

#Уведомления через email

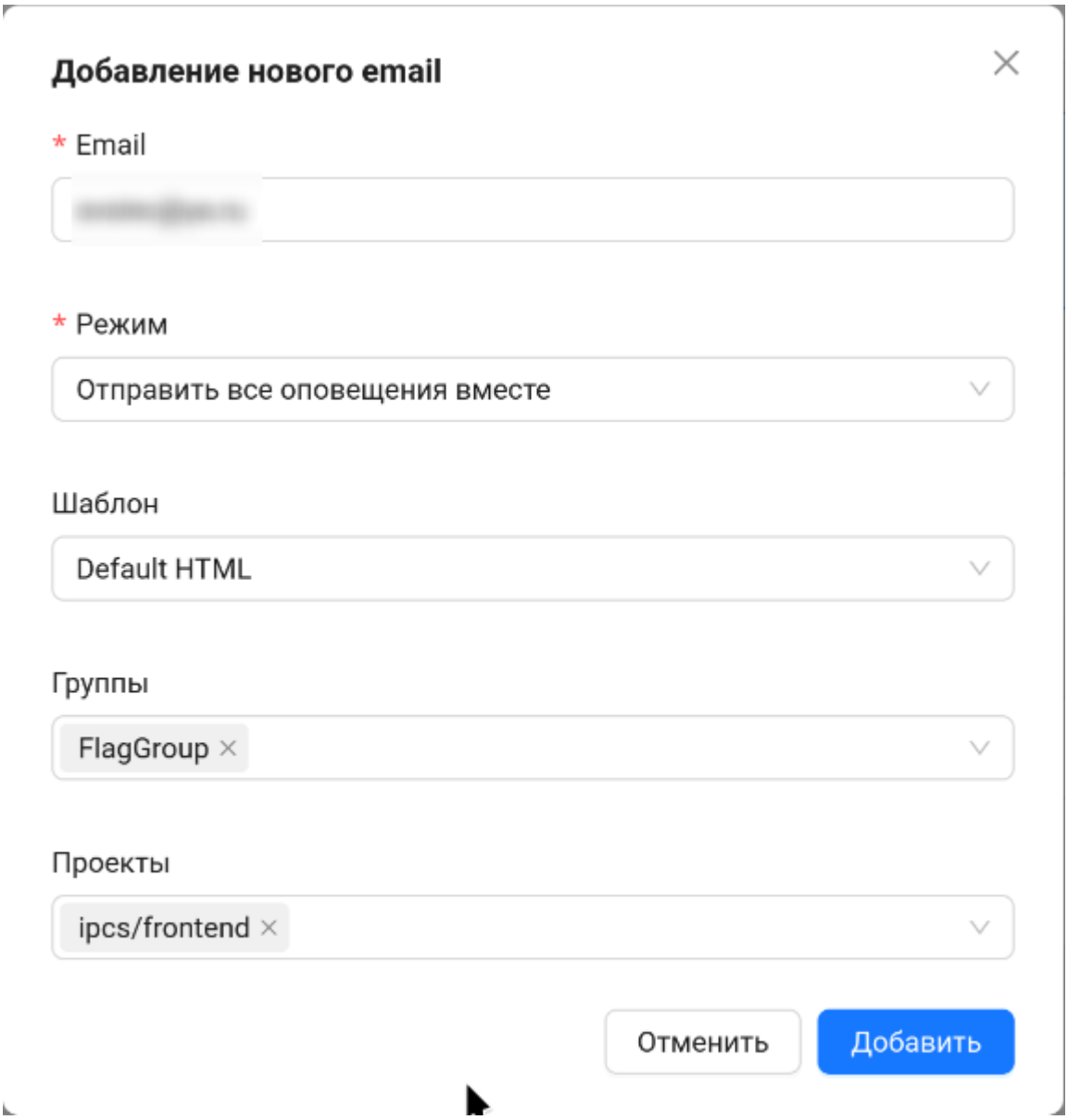
Отправка email оповещений осуществляется через интеграцию по протоколу SMTP.

Для отправки уведомлений через email необходимо предварительно настроить почтовый сервер в разделе
`Настройки -> Уведомления -> Email`. Для этого нужно заполнить все обязательные параметры и установить чек-бокс **Активный**.

Проверить правильность конфигурации можно по кнопке **Проверить подключение**.



После настройки почтового сервера на вкладке Действия на странице политики можно добавить email адрес, на который будет осуществляться рассылка писем с результатами работы политики:



- **Email** — почтовый адрес;
- **Режим** — режим отправки писем:
  - Отправить каждое оповещение отдельно;
  - Отправить все оповещения вместе;
- **Шаблон** - название шаблона. Если не указано, будет использован стандартный шаблон;
- **Группы** — группы проектов, на которые делается оповещение. Если не указано, подразумеваются все группы;
- **Проекты** — конкретные проекты, на которые делается оповещение. Если не указано, подразумеваются все проекты.

Если указаны и группы, и проекты, то оповещения будут включать в себя информацию по всем проектам из указанных групп и по всем указанным проектам.

Письмо с результатами работы политики отправляется **по завершении сканирования проекта**. Содержимое письма зависит от выбранного шаблона.

#Интеграция с таск-менеджерами

CodeScoring поддерживает интеграцию с таск-менеджерами Jira и Kaiten для формирования задач по сработавшим политикам. Настройка интеграции происходит в разделе
`Настройки -> Уведомления -> Менеджеры задач`.

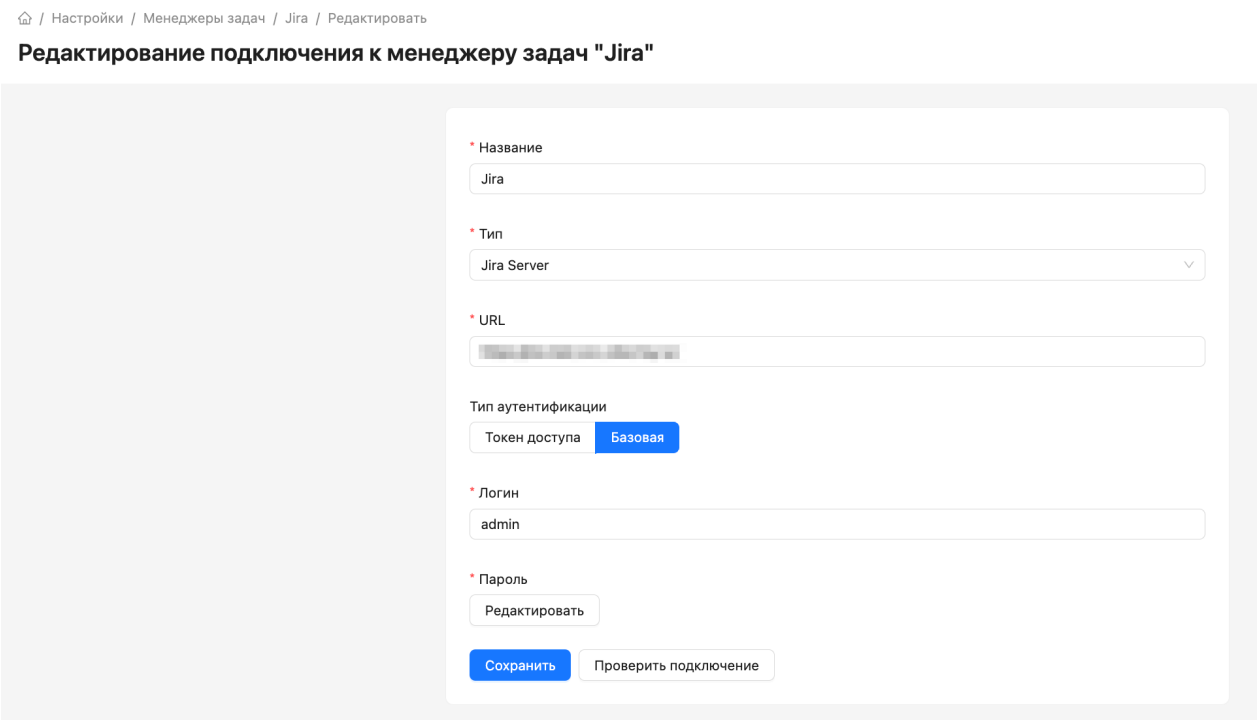
Для создания новой интеграции используется форма по кнопке **Добавить**.

- **Название** - название интеграции;
- **Тип** - тип таск-менеджера;
- **URL** - адрес, по которому доступен таск-менеджер;
- **Тип аутентификации** - аутентификация через токен доступа или логин и пароль.

Тип аутентификации для Kaiten

Kaiten поддерживает аутентификацию только через токен доступа.

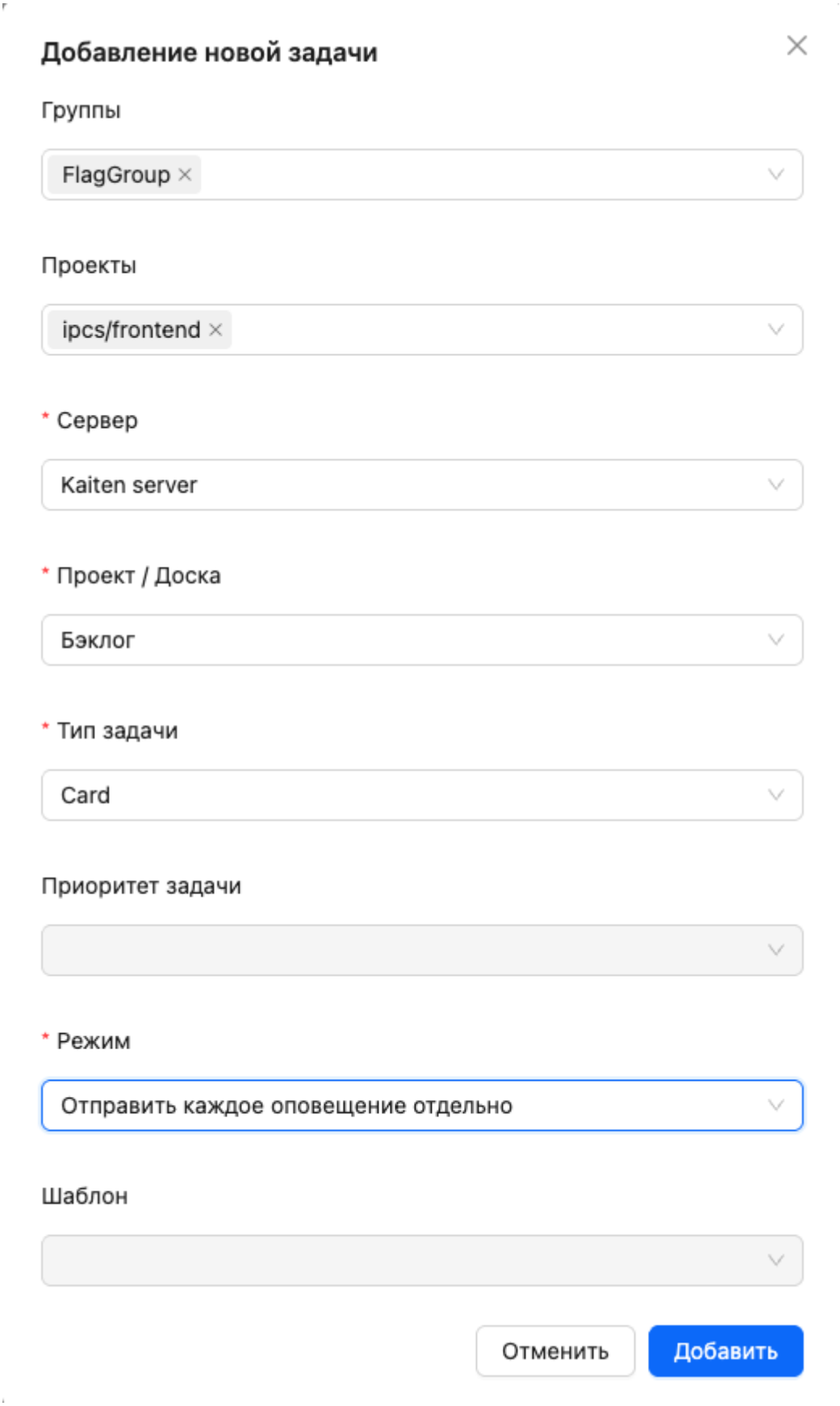
После заполнения полей можно проверить соединение с сервером по кнопке **Проверить подключение**, или завершить создание по кнопке **Добавить**.



#Создание задач в таск-менеджерах

После настройки интеграции на вкладке Действия на странице политики можно добавить сервер Jira или Kaiten, на котором будет создаваться задача с результатами работы политики:

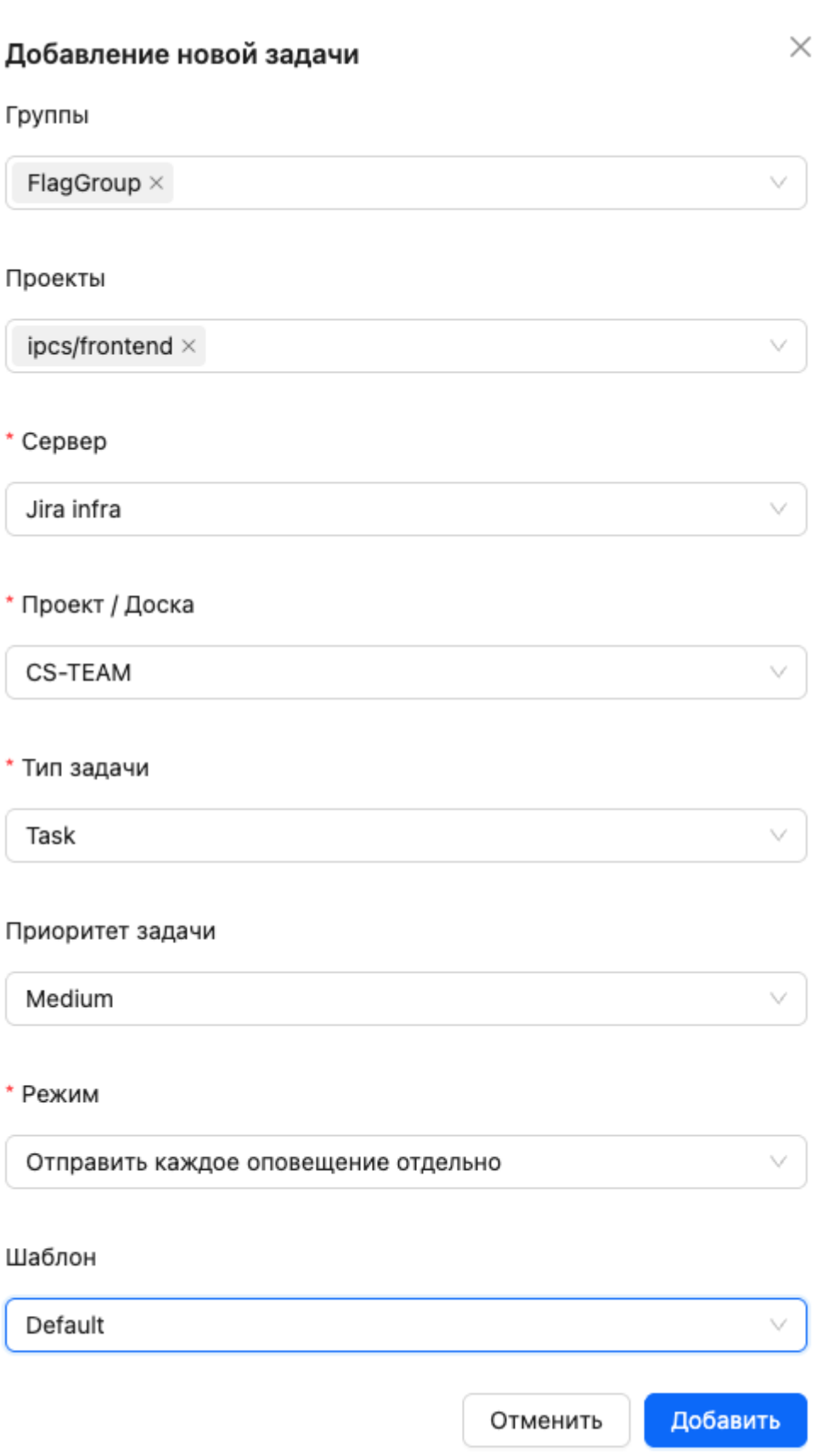
#Создание задач в Kaiten



- **Режим** — режим отправки:
  - Отправить каждое оповещение отдельно;

    - Отправить все оповещения вместе.
- **Группы** — группы проектов, на которые делается оповещение. Если не указано, подразумеваются все группы;
- **Проекты** — конкретные проекты, на которые делается оповещение. Если не указано, подразумеваются все проекты;
- **Сервер** — таск-менеджер (в данном случае Kaiten);
- **Проект/Доска** — доска в Kaiten;
- **Тип задачи** — тип карточки доступный в Kaiten;

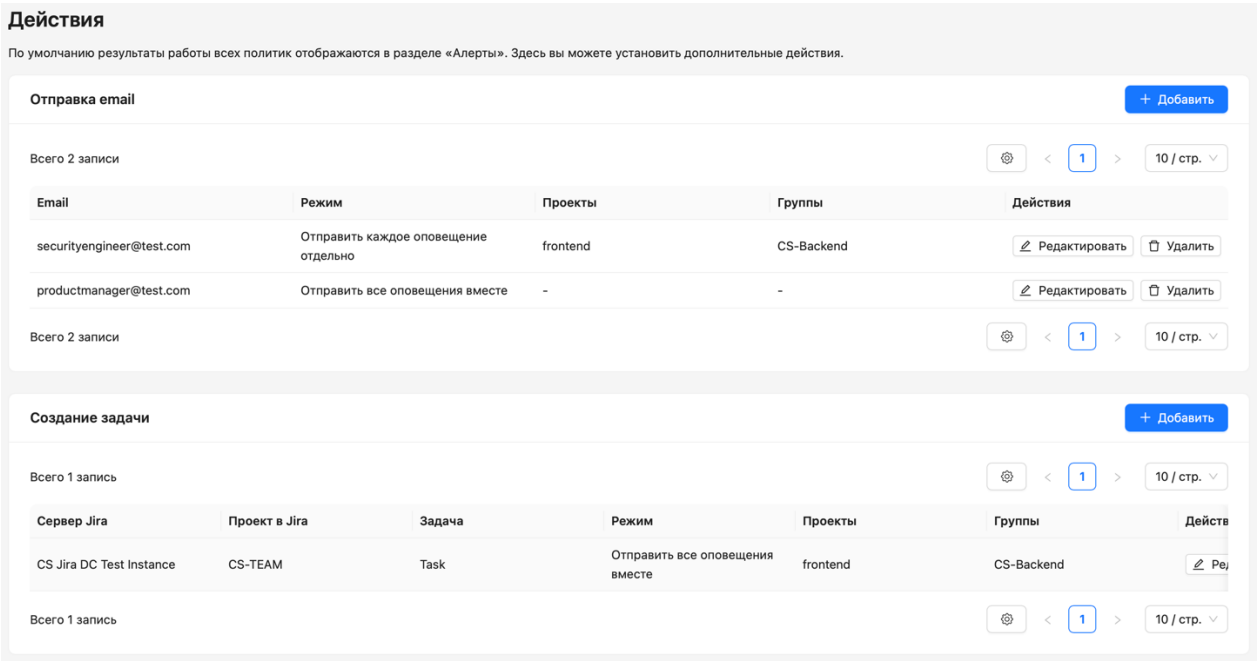
#Создание задач в Jira



- **Режим** — режим отправки:
  - Отправить каждое оповещение отдельно;

    - Отправить все оповещения вместе.
- **Группы** — группы проектов, на которые делается оповещение. Если не указано, подразумеваются все группы;
- **Проекты** — конкретные проекты, на которые делается оповещение. Если не указано, подразумеваются все проекты;
- **Сервер** — таск-менеджер (в данном случае Jira);
- **Проект/Доска** — проект в Jira;
- **Тип задачи** — тип карточки: *Task*, *Story* или *Bug*;
- **Приоритет задачи** - приоритет карточки. Если не указан, будет использован приоритет по умолчанию на стороне Jira;
- **Шаблон** - название шаблона. Если не указано, будет использован стандартный шаблон.

Если указаны и группы, и проекты, то оповещения будут включать в себя информацию по всем проектам из указанных групп и по всем указанным проектам.



#Управление шаблонами

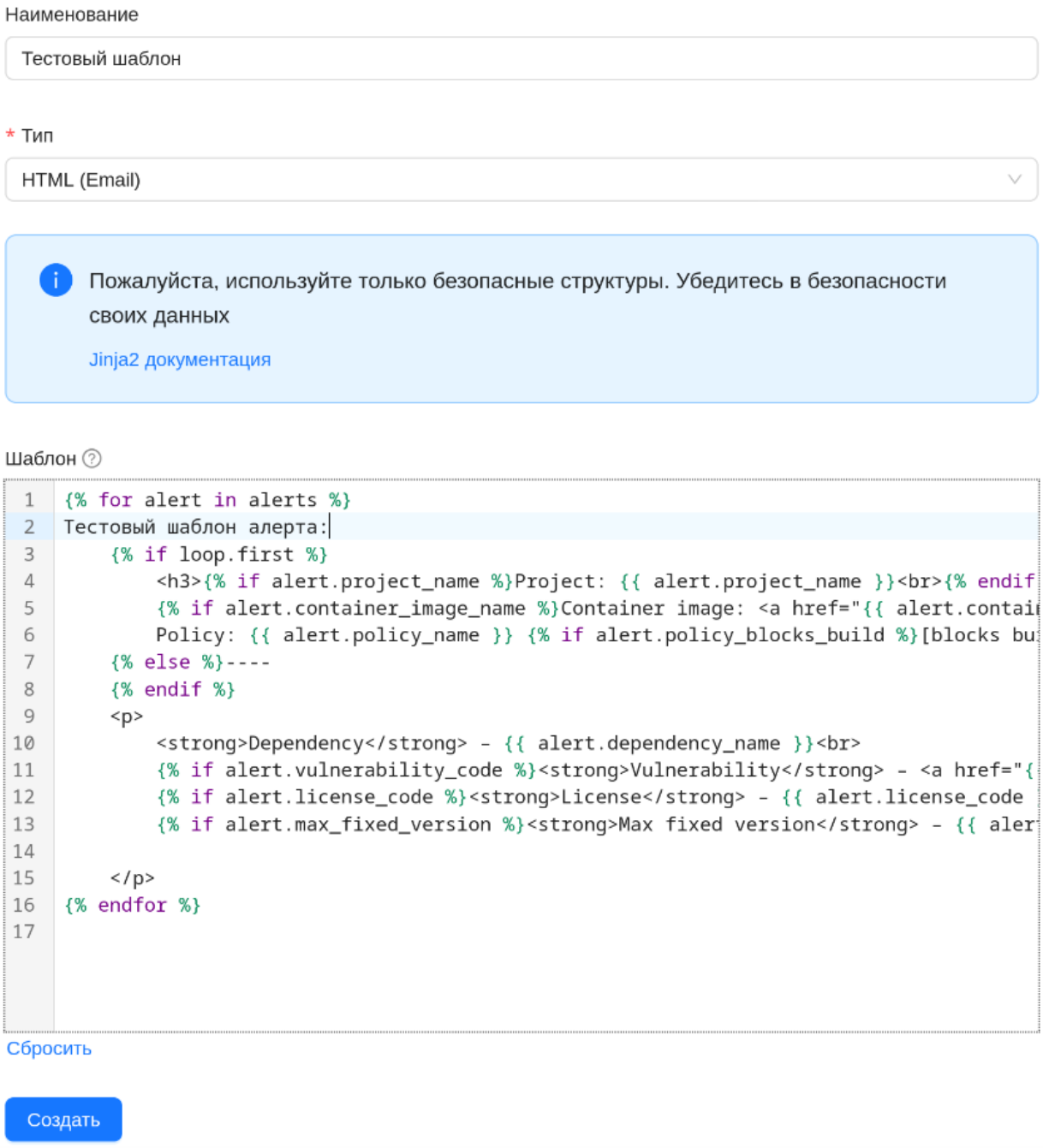
В CodeScoring поддерживается возможность использования собственных шаблонов для уведомлений по email или создания Jira-задач. Управление шаблонами доступно в разделе `Настройки -> Уведомления -> Шаблоны`.

Для создания нового шаблона используется форма со следующими полями:

- Наименование;
- Тип - шаблоны в зависимости от использования разделяются по типам Markdown для Jira-задач и HTML для email-оповещений;
- Шаблон - содержимое шаблона в формате jinja2.

Важно

Используйте только безопасные структуры. ::: Прежде чем завершить создание шаблона убедитесь в безопасности своих данных.



При заполнении поля **Шаблон** формы, необходимо учитывать, что шаблон может использоваться как для режима отправки "Отправить каждое оповещение отдельно", так и для "Отправить все оповещения вместе".

Содержимое email или jira-задачи формируется на основании шаблона и контекста алертов. В контексте предоставляется коллекция алертов (в режиме раздельной отправки в коллекции будет один алерт).

Для каждого алерта можно использовать следующие переменные:

- policy_alert_level: str - уровень критичности алерта;
- policy_alert_stage: str - стадия цикла разработки;
- policy_alert_matched_criteria_list: list[str] - список условий политики;
- policy_name: str - название политики;
- policy_blocks_build: bool - блокировка сборки или загрузки компонента из прокси-репозитория;
- policy_block_delay: int - задержка (в днях) от первого срабатывания политики до блокировки;
- policy_is_block_delayed: bool - использование отложенной блокировки;
- dependency_name: str - название зависимости;
- dependency_link: str - ссылка на зависимость;
- dependency_technology: str - язык программирования или экосистема;
- vulnerability_code: Optional[str] - идентификатор уязвимости из внешней базы;
- vulnerability_link: Optional[str] - ссылка на уязвимость;
- max_fixed_version: Optional[str] - максимальная исправленная версия;
- license_code: Optional[str] - лицензия;
- project_name: Optional[str] - название проекта;
- container_image_name: Optional[str] - название образа контейнера;
- container_image_link: Optional[str] - ссылка на образ контейнера.

:::note Примечание Все ссылки ведут на ту платформу, на которой были сформированы данные для email или задачи в Jira.

#Работа с аудит-логом

Аудит-лог – это журнал событий в системе **CodeScoring**. Он находится в разделе `Настройки -> Аудит лог`.

В аудит-логе фиксируются события системы, действия пользователей и произошедшие ошибки. Каждое событие содержит следующие данные:

- **Время события** – дата и время события;
- **Инициатор** – имя пользователя или `system` для действий системы;
- **Сообщение** – сообщение с деталями события;
- **Длительность** – продолжительность событий по этапам анализа и работе с внешними источниками.

Журнал можно отфильтровать по периоду или инициатору, а также найти в нем конкретное событие, используя поле Поиск. Также доступен экспорт журнала в формате CSV.

События в аудит-логе разделяются на несколько категорий. Ниже приведен полный список возможных событий по каждой из категорий с расшифровками.

#Активация лицензии

Текст события	Расшифровка
Object <ActivationKey: Owner: {owner.name}. Expired: YYYY-MM-DD HH:MM:SS+HH:MM (timezone). Authors limit: {limit.number}> created	Активация лицензии на ПО с указанием владельца ключа и ограничения по количеству авторов
No activation key	Отсутствует ключ активации
Problem with activation key: {status.lower()}	Проблема с ключом активации

#Аутентификация пользователя

Текст события	Расшифровка
User logged in	Пользователь успешно аутентифицировался в системе
User logged out	Пользователь вышел из системы
Failed login attempt {username}	Введен неверный пароль при попытке аутентификации

#Управление объектами

Текст события	Расшифровка
Object {instance!r} created	Создание любого объекта в системе пользователем через интерфейс
Object {instance!r} updated	Обновление любого объекта в системе пользователем через интерфейс
Object {instance!r} deleted	Удаление любого объекта в системе пользователем через интерфейс

#Запуск SCA анализа

Текст события	Расшифровка
[SCA][{analysis_run.project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Analysis started	SCA анализ проекта запущен
[SCA][{analysis_run.project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Analysis finished	SCA анализ проекта окончен
[SCA][{analysis_run.project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Analysis failed. Check server logs.	SCA анализ проекта завершился с ошибкой. Необходимо проверить логи сервера.
Failed to clone for repository {repository.name}.	Клонирование репозитория не удалось
Failed to detect branch for repository "{repository.name}	Не удалось обнаружить ветку для репозитория
[SCA][{project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Clone source code	Клонирование исходного кода проекта запущено
SCA][{project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Collect files data	Поиск данных о файлах проекта запущен
[SCA][{project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Collect manifests	Поиск манифестов проекта запущен
[SCA][{project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Create pipeline	Создание pipeline для SCA анализа проекта запущено
[SCA][{project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Check policies	Проверка политик SCA анализа проекта запущена
[SCA][{project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Update project metrics	Обновление метрик проекта для SCA анализа проекта запущено
[SCA][{project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Analyze dependencies	Анализ зависимостей проекта для SCA анализа проекта запущен

Текст события	Расшифровка
[[SCA][{project.name}] Analysis didn't start (Reason: {err.message})]	SCA анализ проекта не запустился из-за ошибки
Overall SCA run started for {len(projects)} project(s)	Запущен общий SCA анализ для проектов

Каждое событие SCA анализа содержит последовательный номер анализа в проекте и **UUID** запуска.

#Запуск TQI анализа

Текст события	Расшифровка
Rebuild author {primary_email}	Обновление информации по автору на основании правил объединения по основному email
Generate authors merge rules	Создание правил слияния авторов
(Run #{run_id}) Collect commits data for project {project.name}	Сбор данных коммитов для проекта запущен
(Run #{run_id}) Create authors	Создание авторов
(Run #{run_id}) Load authors OSS contributions	Загрузка вклада авторов в OSS
(Run #{run_id}) Authors analysis started	Начался анализ авторов
(Run #{run_id}) Authors analysis completed	Анализ авторов успешно завершен
(Run #{run_id}) Authors analysis failed. Check server logs.	Анализ авторов завершен с ошибкой. Необходимо проверить логи сервера
(Run #{run_id}) Authors analysis cancelled	Анализ авторов отменен
(Run #{run_id}) Update project {project.name}	Обновление проекта
(Run #{run_id}) Clones analysis started	Запущен анализ клонированного кода запущен

Текст события	Расшифровка
(Run #{run_id}) Clones analysis completed	Завершен анализ клонированного кода
(Run #{run_id}) Clones analysis failed. Check server logs.	Анализ клонированного кода завершен с ошибкой. Необходимо проверить логи сервера
(Run #{run_id}) Clones analysis cancelled	Анализ клонированного кода отменен
(Run #{run_id}) Clone source code for project {project.name}	Клонирование репозитория исходного кода проекта запущено

Каждое событие TQI анализа содержит **UUID** запуска.

#Управление политиками

Текст события	Расшифровка
Policy ignore {policy_ignore} created	Создание правила игнорирования политики
Policy ignore {policy_ignore} activated	Активация правила игнорирования политики
To Policy ignore {policy_ignore} added Policy {policy_alert.policy}	Добавлена политика в существующее правило игнорирования
Policy "{policy.name}" (id: {policy.pk}) skipped. Reason: {err!r}	Политика пропущена по причине ошибки

#Анализ секретов

Текст события	Расшифровка
[Secrets][{analysis_run.analysis_object}] Analysis started	Запущен анализ секретов
[Secrets] Training run started	Запущено обучение пользовательской модели на основе результатов разметки

Текст события	Расшифровка
[Secrets][{analysis_run.analysis_object}] Analysis finished	Завершен анализ секретов
[Secrets] Training run finished	Завершено обучение пользовательской модели на основе результатов разметки
[Secrets][{analysis_run.analysis_object}] Analysis failed. Check server logs.	Ошибка анализа секретов
[Secrets] Training run failed. Check server logs.	Ошибка обучения пользовательской модели

#Анализ контейнерных образов

Текст события	Расшифровка
In container image {container_image} dependency {dep_name_and_version} was changed	В контейнерном образе изменена зависимость
Updating images list for registry {container_registry} triggered via update button.	Запущено принудительное обновление списка образов из реестра

#Работа с LDAP

Текст события	Расшифровка
Applying all LDAP group mapping rules triggered	Запуск применения всех правил сопоставления групп LDAP
{message} While processing, failed to apply some of rules related to following LDAP servers: {', '.join(ldap_servers_mapping_failed_for)}. Check server logs.	Ошибка применения правил сопоставления групп для указанных LDAP-серверов

#Прочее

Текст события	Расшифровка
(Run #{task.id}) Analysis started via API	Анализ запущен через API
Some tasks in analysis failed	Некоторые задачи при анализе не выполнены
Could not connect to OSS Index, reason: {err}	Не удалось подключиться к индексу OSS из-за ошибки
Could not connect to OSS Index, reason: {err}	Ошибка подключения к OSS Index
There is already running analysis	Анализ уже запущен
Another analysis in progress. Parallel execution forbidden.	Запрещен параллельный запуск анализа
Repo path for {project} does not exist, setting status to Not cloned	Путь к репозиторию не найден, статус установлен как "Не клонирован"
Failed to clone for repository {project.repo_name} because project was deleted	Не удалось клонировать репозиторий, так как проект был удален
Updating {update_type} since {updated_after} / Updating all {update_type}	Информация об обновлении лицензий, уязвимостей

#Настройка метрик

CodeScoring хранит метрики в формате, поддерживаемом инструментом мониторинга **Prometheus**. На данный момент доступны метрики с платформе CodeScoring и плагинов в прокси-репозиториях.

#Сбор метрик платформы

Метрики платформы доступны в **CodeScoring API** по адресу `{platform-url}/api/metrics`. Для того, чтобы настроить отслеживание метрик в Prometheus, необходимо выполнить следующие шаги:

1. Открыть файл конфигурации `prometheus.yml` и добавить параметры для мониторинга метрик. Ниже приведен пример:

```
global:
  scrape_interval: 15s

  scrape_configs:
  - job_name: 'demo-codescoring'
    metrics_path: '/api/metrics'
    static_configs:
      - targets: ['{platform-url}'] # Адрес хоста
платформе
  - job_name: 'osa'
    metrics_path: '/api/osa/metrics'
    static_configs:
      - targets: ['{platform-url}'] # Адрес хоста
платформы
```

2. Перезапустить Prometheus, чтобы изменения вступили в силу.

3. Открыть интерфейс Prometheus и перейти на страницу **Graph**. В поле запроса ввести название одной из метрик:

#Метрики очередей

- **codescoring_tasks_queue_size_total** – общее количество задач, ожидающих выполнения;
- **codescoring_tasks_schedule_queue_size_total** – общее количество запланированных задач, ожидающих выполнения;
- **codescoring_tasks_running_tasks_total** – общее количество задач, выполняющихся в данный момент;
- **codescoring_celery_queue_size_total** – общее количество задач в Celery-очередях;
- **codescoring_celery_running_tasks_total** – общее количество выполняющихся задач в Celery-очередях.

Данные метрики можно отфильтровать по лейблу queue со следующими возможными значениями:

#Основные очереди:

- **ipcs** - основная очередь для исполнения асинхронных задач;
- **osa-container-image-scan** - очередь для сканирования контейнерных образов;
- **osa-package-scan** - очередь для сканирования пакетов, запрошенных через модуль OSA;
- **policy** - очередь для расчёта политик;
- **tqi** - очередь для задач в рамках анализов модуля TQI;
- **sca-external-scan** - очередь для SCA, запущенного с помощью консольного агента Johnny;
- **secrets** - очередь для задач в рамках анализов модуля Secrets.

#Очереди Celery:

- **default** - основная очередь для исполнения асинхронных задач;
- **webhooks** - очередь для работы с вебхукам;
- **secrets-external-scan** - очередь для анализов секретов, запущенных с помощью консольного агента Johnny;
- **osa-maintenance** - очередь для периодических задач модуля OSA;
- **osa-background-update** - очередь для фонового обновления сущностей модуля OSA;
- **oss-index** - очередь для загрузки данных из OSS Index, используется в случае, если интеграция инсталляции с ним включена;
- **migration-background-tasks** - очередь для фонового выполнения задач, связанных с перерасчётом данных после миграций схемы БД;
- **media** - очередь для асинхронной генерации отчётов;
- **media-cleaner** - очередь для периодического удаления сгенерированных отчётов;
- **sca-tasks** - очередь для анализов в рамках модуля SCA.

#Метрики анализа

codescoring_running_analyses_total – общее количество выполняющихся анализов.

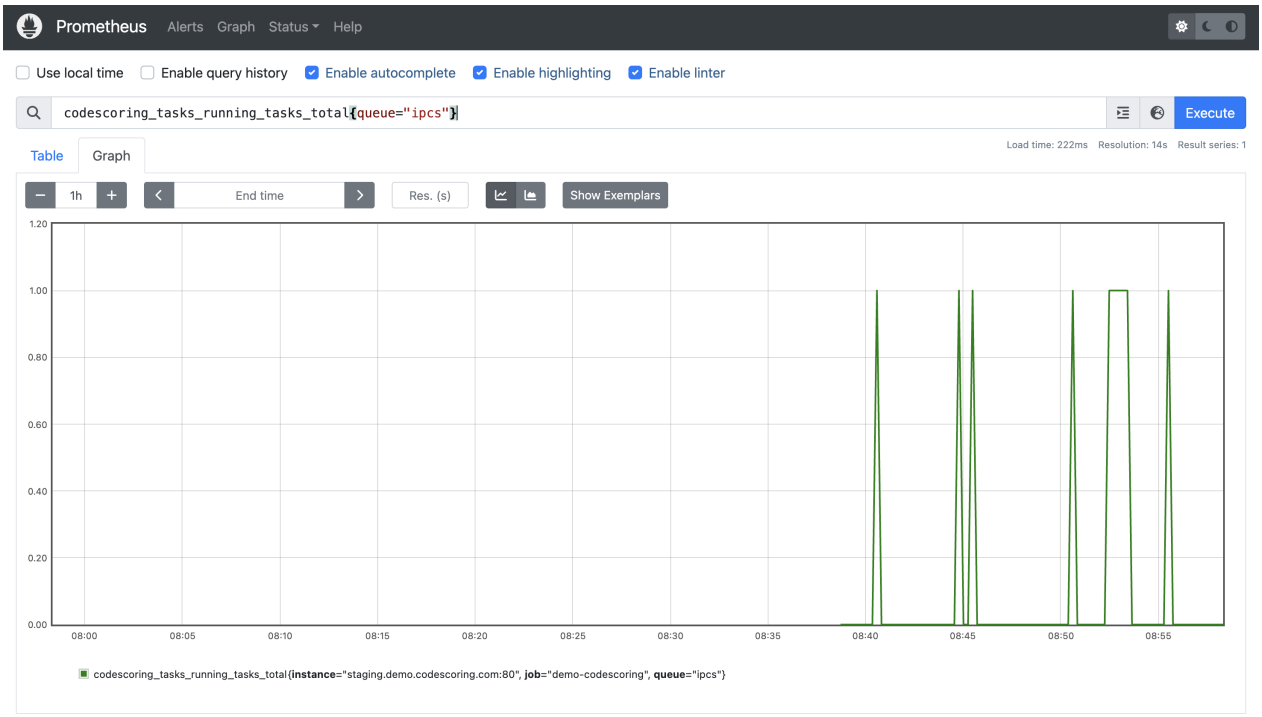
Данную метрику можно отфильтровать по лейблу `analysis_type` со следующими возможными значениями:

- `sca` – анализ зависимостей;
- `authors` – анализ авторов кода;
- `clones` – поиск клонов кода.

#Метрики запросов от плагинов OSA

- **codescoring_registration_packages_queue_size** – очередь регистрации пакетов из OSA;
- **codescoring_registration_container_images_queue_size** – очередь регистрации контейнерных образов из OSA.

Пример визуализации метрик:



#Сбор метрик OSA

Метрики OSA содержат информацию о состоянии соединений, количестве и времени запросов, а также о статусах сканирования и блокировки компонентов в менеджерах репозиториев. Метрики доступны в **CodeScoring API** по адресу `{platform-url}/api/osa/metrics`.

#Метрики соединений

codescoring_osa_api_db_connection_pool – состояние пула соединений к базе данных.

Метрику можно отфильтровать по лейблу `measure` со следующими возможными значениями:

- `pool_min` – минимальный размер пула;
- `pool_max` – максимальный размер пула;
- `pool_size` – текущий размер пула;
- `pool_available` – доступные соединения;
- `requests_waiting` – количество ожидающих запросов;
- `pool_used` – используемые соединения.

codescoring_osa_api_redis_connection_pool – состояние пула соединений к Redis.

Метрику можно отфильтровать по лейблу `measure` со следующими возможными значениями:

- `available_connections` – доступные соединения;
- `in_use_connections` – используемые соединения;
- `total_connections` – общее количество соединений.

#Метрики запросов компонентов

codescoring_osa_api_http_request_duration_seconds – продолжительность HTTP-запросов в секундах.

Метрика представлена в виде гистограммы и включает:

- `codescoring_osa_api_http_request_duration_seconds_sum` – общее время запросов;
- `codescoring_osa_api_http_request_duration_seconds_bucket` – распределение времени запросов по временным интервалам;
- `codescoring_osa_api_http_request_duration_seconds_count` – общее количество запросов.

Метрику можно отфильтровать по лейблам:

- `handler` – обработчик запроса (например, `/api/osa/packages/`);
- `method` – метод запроса (например, `POST`);
- `status` – HTTP-статус ответа (например, 2xx, 4xx, 5xx).

Пример:

```
codescoring_osa_api_http_request_duration_seconds_bucket{handl
er="/api/osa/packages/",le="0.01",method="POST",status="2xx"}
0.0
```

codescoring_osa_api_http_requests_total – общее количество HTTP-запросов.

Метрику можно отфильтровать по тем же лейблам, что и `codescoring_osa_api_http_request_duration_seconds`.

#Метрики статусов компонентов

codescoring_osa_api_requested_component_block_status_total – количество компонентов в различных статусах блокировки.

Метрику можно отфильтровать по лейблам:

- `block_status` – статус блокировки (например, `not_blocked`, `blocked_by_policies`, `blocked_scan_failed`);
- `object_type` – тип объекта (например, `package`, `container_image`).

Пример:

```
codescoring_osa_api_requested_component_block_status_total{blo
ck_status="blocked_by_policies",object_type="package"} 1.0
```

codescoring_osa_api_requested_component_scan_status_total – количество компонентов в различных статусах сканирования.

Метрику можно отфильтровать по лейблам:

- `scan_status` – статус сканирования (например, `not_scanned`, `scanned`);
- `object_type` – тип объекта (например, `package`, `container_image`).

Пример:

```
codescoring_osa_api_requested_component_scan_status_total{obje
ct_type="container_image",scan_status="not_scanned"} 0.0
```

#Метрики соединения с Index API

index_api_failure_rate - количество неуспешных запросов к Index API подряд.

#Подключение вебхуков

CodeScoring поддерживает систему оповещений на базе вебхуков. При возникновении событий отправляется P0ST HTTP-запрос на указанный URL.

#Добавление нового вебхука

Для добавления нового вебхука на платформе необходимо выполнить следующие действия:

1. Перейти в раздел `Настройки -> Вебхуки`.
2. Нажать на кнопку **Добавить**.
3. Заполнить поля в форме:
    - **Название** – название в системе CodeScoring;
    - **URL** – адрес с указанием протокола. Например: `http://webhook.com/`;
    - **Проекты** - список проектов, для которых будут применяться вебхуки. Если поле пустое, будут учитываться все проекты;
    - **События** – список событий, на которые сработает оповещение;
    - **Токен** – токен, который передается в HTTP-запросе (заголовок `X-CodeScoring-Authentication`);
4. Проверить подключение после заполнения данных по кнопке **Проверить подключение**. При тестировании используется триггер `test` с пустым payload.

После создания нового подключения по кнопке **Добавить** вебхук отобразится в списке раздела с возможностью посмотреть детальную информацию, отредактировать или удалить его.

#Структура тела запроса

```
{
  "events": [
    {
      "created_at": "datetime_in_iso_format",
      "trigger": "trigger_name",
      "payload": {
        ...
      }
    },
    ...
  ]
}
```

#Механизм отправления запросов

Запросы обрабатываются каждые 5 секунд, согласно периодической задаче, выполняемой системой.

Вебхук обычно отправляет одно событие за запрос. Однако в следующих случаях может быть отправлен массив событий:

- **Отправка накопленных событий:** если за единицу времени (по умолчанию – 5 секунд) произошло несколько событий, они будут отправлены одним запросом;
- **Ретраи (повторные попытки):** если сервер не ответил или вернул ошибку на предыдущую попытку, все неуспешные события отправляются повторно единым запросом.

Коды HTTP в диапазоне [200; 299] считаются успешными. Если ответ сервера не входит в этот диапазон, событие считается неуспешным, накапливается и отправляется повторно по следующему расписанию: 1 минута → 5 минут → 30 минут → 3 часа → 12 часов → 24 часа → 48 часов. Если хотя бы один запрос остается безуспешным через 48 часов, вебхук отключается, и события больше не отправляются.

#Управление вебхуками через API

Для управления вебхуками предоставляется API по эндпоинту `/api/settings/webhooks` со следующими командами:

- `GET /api/settings/webhooks/` — получить список всех вебхуков;
- `POST /api/settings/webhooks/` — создать новый вебхук;
- `GET /api/settings/webhooks/{id}/` — получить информацию по ID;
- `PUT /api/settings/webhooks/{id}/` — полностью обновить данные;
- `PATCH /api/settings/webhooks/{id}/` — частично обновить данные;
- `DELETE /api/settings/webhooks/{id}/` — удалить вебхук по ID;
- `POST /api/settings/webhooks/{id}/refresh_availability_status/` — обновить статус доступности;

- `POST /api/settings/webhooks/test/` — протестировать подключение;
- `GET /api/settings/webhooks/triggers/` — получить список доступных триггеров.

#Доступные события

Название события	Описание события	Значение trigger в запросе	Сх
SCA Policy has been triggered	Сработала политика при SCA-анализе	sca_policy_has_been_triggered	
OSA Policy has been triggered	Сработала политика при OSA-анализе	osa_policy_has_been_triggered	
Project SCA analysis started	SCA-анализ проекта был запущен	project_sca_analysis_started	
Project SCA analysis finished	SCA-анализ проекта был завершён	project_sca_analysis_finished	

Название события	Описание события	Значение trigger в запросе	Сх
Project SCA analysis cancelled	SCA-анализ проекта был отменён	project_sca_analysis_cancelled	
Project SCA analysis failed	SCA-анализ проекта был завершён с ошибкой	project_sca_analysis_failed	
Container Image SCA analysis started	SCA-анализ контейнерного образа был запущен	container_image_sca_analysis_started	
Container Image SCA analysis finished	SCA-анализ контейнерного образа был завершён	container_image_sca_analysis_finished	
Container Image SCA analysis cancelled	SCA-анализ контейнерного образа был отменён	container_image_sca_analysis_cancelled	
Container Image SCA analysis failed	SCA-анализ контейнерного образа был завершён с ошибкой	container_image_sca_analysis_failed	
		clones_analysis_started	

Название события	Описание события	Значение trigger в запросе	Сх
Clone analysis started	Анализ дубликатов проекта был запущен		
Clone analysis finished	Анализ дубликатов проекта был завершён	clones_analysis_finished	
Clone analysis cancelled	Анализ дубликатов проекта был отменён	clones_analysis_cancelled	
Clone analysis failed	Анализ дубликатов проекта был завершён с ошибкой	clones_analysis_failed	
Authors analysis started	Анализ авторов проекта был запущен	authors_analysis_started	
Authors analysis finished	Анализ авторов проекта был завершён	authors_analysis_finished	
Authors analysis failed	Анализ авторов проекта был завершён с ошибкой	authors_analysis_failed	

Название события	Описание события	Значение trigger в запросе	Сх
Authors analysis cancelled	Анализ авторов проекта был отменён	authors_analysis_cancelled	
Cloning of repository started	Клонирование репозитория было запущено	cloning_of_repository_started	
Cloning of repository finished	Клонирование репозитория было завершено	cloning_of_repository_finished	
Cloning of repository failed	Клонирование репозитория было завершено с ошибкой	cloning_of_repository_failed	
Project Secrets analysis started	Начался анализ секретов проекта	project_secrets_analysis_started	
Project Secrets analysis finished		project_secrets_analysis_finished	

Название события	Описание события	Значение trigger в запросе	Сх
	Завершился анализ секретов проекта		
Project Secrets analysis cancelled	Анализ секретов проекта был отменен	project_secrets_analysis_cancelled	
Project Secrets analysis failed	Анализ секретов проекта был завершен с ошибкой	project_secrets_analysis_failed	
Secrets ML model user training run started	Началось обучение ML модели для модуля Секретов	ml_model_user_secrets_training_run_started	
Secrets ML model user training run finished	Обучение ML модели для модуля Секретов завершено	ml_model_user_secrets_training_run_finished	
Secrets ML model user training run cancelled	Обучение ML модели для модуля Секретов отменено	ml_model_user_secrets_training_run_cancelled	
Secrets ML model user training run failed	Обучение ML модели для модуля Секретов завершено с ошибкой	ml_model_user_secrets_training_run_failed	

Название события	Описание события	Значение trigger в запросе	Сх
Secrets ML model user accepted	ML модель для модуля Секретов принята	ml_model_user_secrets_accepted	
Secrets ML model user purged	ML модель для модуля Секретов очищена	ml_model_user_secrets_purged	

Пример простого приложения на Flask

```
import json

from flask import Flask, request


app = Flask(__name__)

events = []


@app.get('/')
def show_events():
    return f'<pre>{json.dumps(events, indent=2)}</pre>'


@app.post('/')
def handle_events():
    received_events = request.json.get('events', [])
    events.extend(received_events)
    return 'OK'
```

#Работа с API

CodeScoring имеет открытый API, который позволяет программно взаимодействовать с системой. Для описания команд API используется инструмент Swagger, доступный по ссылке **[platform-url]/api/swagger**.

#Начало работы

Чтобы начать работу с CodeScoring API, необходим токен для аутентификации запросов. Получить токен можно в разделе Профиль по нажатию кнопки **Сгенерировать** в поле API токен.

Для аутентификации запросов вне Swagger необходимо прописать токен в header следующим образом:

`Authorization: Token <YOUR_TOKEN>`

#Структура API

Открытый API предоставляет ряд эндпоинтов, которые позволяют выполнять основные операции в системе. Эндпоинты объединены в разделы, соответствующие объектам в системе CodeScoring — зависимостям, лицензиям, уязвимостям, авторам и т.д.

#Пагинация

Для постраничного вывода результатов используется параметр page для указания номера страницы и параметр per_page для указания количества элементов на странице.

Значение параметра per_page регулируется переменной окружения CODESCORING_API_MAX_PAGE_SIZE, по умолчанию равно 100.

Рекомендация по значению переменной CODESCORING_API_MAX_PAGE_SIZE

Установка значения выше 100 может ухудшить производительность системы.

Некоторые эндпоинты работают **только** в режиме постраничного вывода, подробности можно получить на странице документации API **[platform-url]/api/swagger**.

#Примеры использования

- Запустить анализ всех проектов:

```
curl -X 'POST' \
  '[platform_url]/api/analyses/overall_sca/start/' \
  -H 'accept: application/json' \
  -H 'Authorization: token <YOUR_TOKEN>'
```

- Добавить политику:

```
curl -X 'POST' \
  '[platform_url]/api/policies/' \
  -H 'accept: application/json' \
```

```
  -H 'Authorization: token <YOUR_TOKEN>' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "string",
  "stages": [
    "dev"
  ],
  "level": "info",
  "proprietors": [
    0
  ],
  "projects": [
    0
  ],
  "conditions": {
    "additionalProp1": "string",
    "additionalProp2": "string",
    "additionalProp3": "string"
  },
  "conditions_connector": "and",
  "is_active": true,
  "is_blocks_build": true,
  "description": "string"
}'
```

- Получить информацию об отдельном проекте:

```
curl -X 'GET' \
  '[platform_url]/api/projects/340/' \
  -H 'accept: application/json' \
  -H 'Authorization: token <YOUR_TOKEN>'
```

- Получить список доступных лицензий:

```
curl -X 'GET' \
  '[platform_url]/api/licenses/' \
  -H 'accept: application/json' \
  -H 'Authorization: token <YOUR_TOKEN>'
```

Примечание

Команды создания и изменения основных сущностей в системе, таких как проекты, находятся в разделах с приставкой **settings >**.

#Потоки данных

Для эффективного поиска угроз в open source компонентах CodeScoring интегрирует данные из более **20** источников знаний (фидов). Записи из

всех источников дедуплицируются и объединяются под универсальными идентификаторами в системе.

Источники знаний об угрозах дополняют единую базу данных **CodeScoring Index** с информацией об опубликованных компонентах, включая собственный фид protestware.

Данный раздел содержит детальное описание отдельных фидов и процесса работы с ними.

Ниже приведена таблица с источниками данных и частотой их обновления.

Источник	Ссылка	Частота обновления
CodeScoring Protestware Feed	-	раз в сутки
Банк данных угроз безопасности информации ФСТЭК России	https://bdu.fstec.ru/	каждые 30 минут
Kaspersky Open Source Software Data Feed	https://www.kaspersky.ru/open-source-feed	каждые 30 минут
Astra Linux Security Advisories	https://astra.ru	каждые 4 часа
ALT Linux Security Tracker CVE	https://packages.altlinux.org	каждые 4 часа
Red OS Security Advisories	https://redos.red-soft.ru	раз в сутки
CVE.org (MITRE CVE Program)	https://www.cve.org	раз в сутки
National Vulnerability Database (NVD, NIST)	https://nvd.nist.gov/	раз в час
Open Source Vulnerabilities (OSV)	https://osv.dev	каждые 20 минут
GitHub Advisory Database	https://github.com/advisories	каждые 10 минут
GitHub Security Repositories (Более 2000 репозиториев)	https://github.com	раз в час
	https://openssf.org	раз в час

Источник	Ссылка	Частота обновления
OpenSSF Malware Database (OSSF Malicious Packages)		
Go Vulnerability Database (Golang)	https://vuln.go.dev	каждые 30 минут
GitLab Security Advisory Database	https://advisories.gitlab.com	раз в сутки
Packagist PHP Package Security Advisories	https://packagist.org	каждые 30 минут
PyPI Security Advisories (PySec)	https://pypi.org	каждые 30 минут
Ubuntu Security Notices	https://ubuntu.com/security/notices	каждые 30 минут
Alpine Linux Security Advisories	https://alpinelinux.org	каждые 30 минут
Debian Security Tracker	https://security-tracker.debian.org	каждые 30 минут
Red Hat Security Advisory	https://access.redhat.com	каждые 30 минут

#Работа с фидом protestware

С апреля 2022 года команда CodeScoring начала отслеживать случаи включения protestware в открытые программные компоненты. В дополнение к открытым данным, дополнительные механизмы идентификации подобных пакетов были внедрены в версии **CodeScoring 2022.49.0**.

Начиная с версии 2025.21.0, платформа поддерживает собственный централизованный фид на уровне базы знаний **CodeScoring Index** для обнаружения protestware. Этот фид регулярно обновляется и позволяет автоматически выявлять подобные компоненты в рамках модулей CodeScoring.OSA и CodeScoring.SCA.

#Что такое protestware?

Protestware – это компоненты, содержащие компрометирующие противоправные конструкции, представленные в исходном коде или сопутствующих данных. Такое ПО может изменять свое поведение или

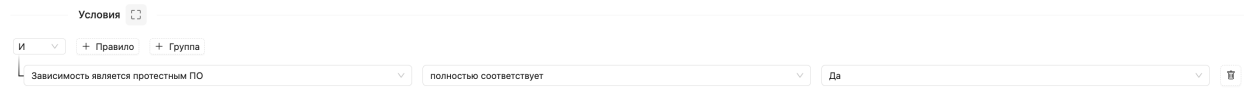
являться предпосылкой к появлению недекларированных возможностей.

Open Source Initiative рассматривает protestware как угрозу нейтральности и воспроизводимости открытого ПО.

#Настройка политики

Для проверки protestware в сторонних компонентах CodeScoring имеет встроенную политику безопасности.

Для ее активации необходимо зайти на форму в разделе `Настройки -> Политики` и указать условие **Зависимость является протестным ПО**



При необходимости можно установить признак **Блокер**, чтобы сделать политику блокирующей – при срабатывании такого условия сборка ПО и загрузка компонента из прокси-репозитория будут прерваны.

Важно

Применение блокирующего признака рекомендуется только после предварительной оценки влияния (проведения инвентаризации компонентов), так как это может повлиять на процесс разработки.

#Результаты анализа

Угрозы, связанные с protestware, помечаются идентификатором **CSPW** в разделе `Уязвимости`. Перейдя на страницу отдельной записи, можно увидеть детальную информацию о компоненте, характере угрозы и затронутой части кодовой базы.

Если настроена соответствующая политика безопасности, срабатывания по ней фиксируются в разделе Алерты. В поле **Условия политики** отображается конкретное основание для срабатывания, например:

```
es5-ext@0.10.64 is protestware
```

#Использование Kaspersky Open Source Software Threats Data Feed

CodeScoring интегрирует фид Kaspersky Open Source Software Threats Data Feed (Kaspersky OSSTDF) в модули CodeScoring.SCA и CodeScoring.OSA, предоставляя доступ к информации об угрозах в обнаруженных open source компонентах. Фид полезен не только как

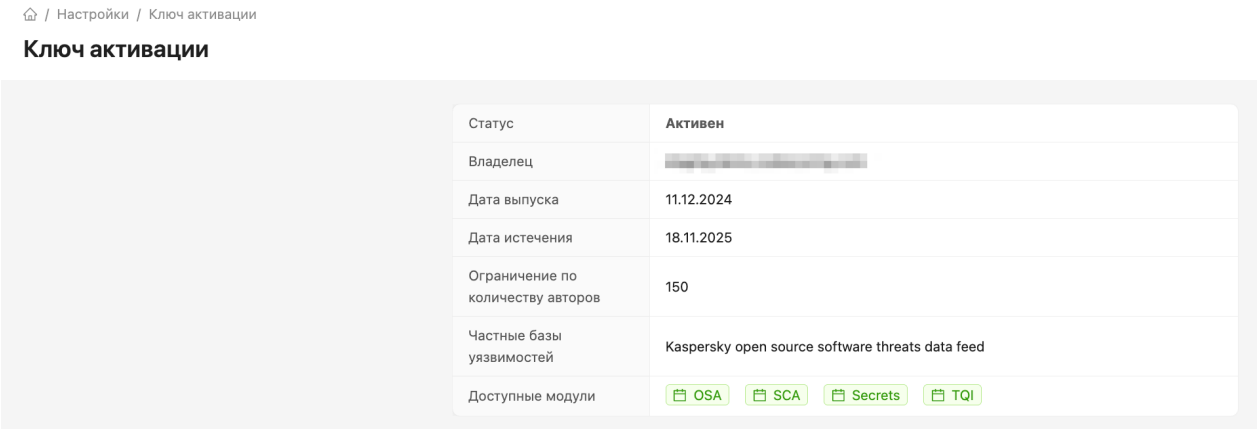
источник данных о новых уязвимостях, но и как инструмент обогащения уже известных записей из других источников.

Фид интегрирован на уровне базы знаний CodeScoring Index, что обеспечивает дедупликацию обнаруженных уязвимостей и единое представление данных для пользователя.

#Проверка подключения

Ключ активации CodeScoring содержит в себе данные о доступности частных фидов.

При успешном подключении Kaspersky OSSTDF, в разделе `Настройки -> Активационный ключ` поле "Частные базы уязвимостей" имеет значение “Kaspersky open source software threats data feed”.



#Результаты анализа

CodeScoring не только интегрирует фид с точки зрения данных, но и предлагает контекстные расширения функциональности. В частности, с подключением фида в списке уязвимостей и на их страницах появляется поле **Импакт**, на которое можно настроить отдельную политику безопасности. Значение поля обозначает то влияние на систему, которое оказывает угроза.

Например, аббревиатура **RLF** означает Read Local Files – такая уязвимость позволяет получить доступ к чтению файлов на устройстве пользователя. Эта информация помогает понять контекст уязвимости и определить направление потенциальной атаки.

Список возможных значений переменной:

- Исполнение произвольного кода (ACE);
- Инъекция кода (CI);
- Отказ в обслуживании (DoS);
- Утрата целостности (LoI);
- Перезапись произвольных файлов (OAF);
- Получение чувствительной информации (OSI);
- Эскалация привилегий (PE);
- Чтение локальных файлов (RLF);

- Обход безопасности (SB);
- Подмена пользовательского интерфейса (SUI);
- Запись локальных файлов (WLF);
- Межсайтовый скриптинг (XSS/CSS).

В случае скомпрометированных пакетов это поле принимает значение `Other`.

Для пакетов с вредоносным кодом возможны в том числе следующие значения:

- Вредоносное ПО (Malware);
- Средства взлома (Hacktool).

Более подробно о доступных полях можно прочитать в документации OSSTDF.

#Данные об уязвимости

На странице уязвимости, полученной из фида Kaspersky OSSTDF появляется дополнительная информация. В таблице с основными данными содержится поле Kaspersky, по которому можно определить идентификатор уязвимости, и поле **Импакт (Kaspersky)**.

#Настройка политик

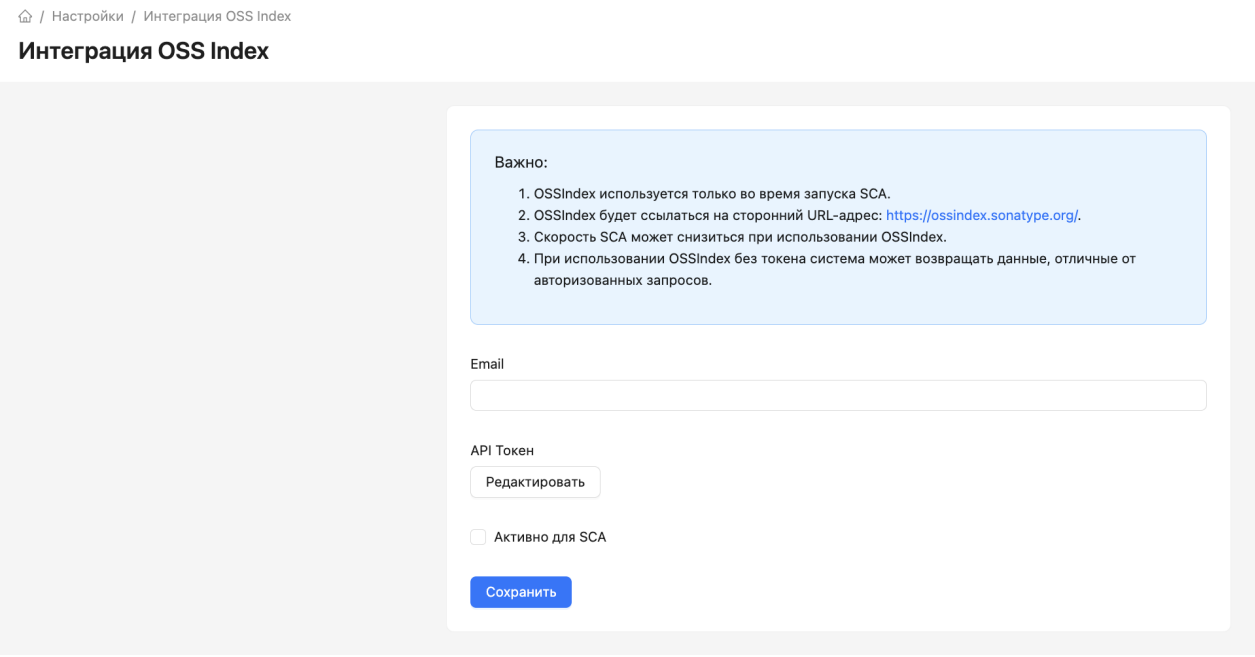
Данные из фида можно использовать для создания политик безопасности как для анализа текущей кодовой базы, так и для проверки поступающих компонентов.

Поле Импакт позволяет точно указать те типы уязвимостей, которые требуют внимания. Например, для настройки политики, блокирующей использование компонентов с вредоносным кодом, необходимо составить условие
`Импакт (Kaspersky) -> полностью соответствует -> Вредоносное ПО`.

#Интеграция OSS Index

В дополнение к внутренней базе данных **CodeScoring Index** для расширенного анализа можно также подключить сторонний фид Sonatype OSS Index.

Интеграция осуществляется в разделе `Настройки -> OSS Index`. Для подключения необходимо заполнить Email и API токен, полученный при регистрации пользователя в Sonatype OSS Index.



Дисклеймер по качеству данных

Интеграция OSS Index использует сторонний фид Sonatype OSS Index. Данные этого источника могут содержать ложноположительные срабатывания и неточности, поэтому рекомендуется использовать его как дополнительный источник и отдельно проверять результаты перед принятием блокирующих решений.

Важно:

1. OSS Index используется только во время запуска SCA.
2. OSS Index будет ссылаться на сторонний URL-адрес: https://ossindex.sonatype.org/.
3. Скорость SCA может снизиться при использовании OSS Index.
4. При использовании OSS Index без токена система может возвращать данные, отличные от авторизованных запросов.

#CodeScoring.Save

#Общее описание

CodeScoring.Save — менеджер репозиториев артефактов для Maven, npm, NuGet, PyPI, Go, Docker / OCI и raw-файлов. Сервис разворачивается в Kubernetes и интегрируется с другими модулями платформы CodeScoring.

Сервис выполняет три задачи:

- **Прокси и кэш** для внешних артефактов с возможностью применения политик безопасности при маршрутизации загрузок через OSA Proxy.

- **Hosted-хранилище** для внутренних библиотек, образов и произвольных файлов организации.
- **Унифицированный контур доступа** ко всем артефактам через выделенный сервис аутентификации и авторизации **cs-auth**.

#Поддерживаемые форматы

Maven, npm, NuGet, PyPI, Go, Docker / OCI, raw. Описание возможностей модуля приведено на странице Функциональные характеристики.

#Безопасность

Для контроля доступа используется выделенный сервис аутентификации и авторизации `cs-auth`.

#Дополнительные разделы

- Функциональные характеристики — возможности платформы и отдельных модулей.
- Установка и эксплуатация CodeScoring.Save — архитектура, требования, развертывание, обновление и настройка SSL/TLS.
- Управление репозиториями — практическое руководство по работе с proxy- и hosted-репозиториями, аутентификацией и политиками очистки.
- Интеграция с CodeScoring.OSA — настройка проверок безопасности через OSA Proxy.

#Управление репозиториями и артефактами

#Контекст

CodeScoring.Save хранит и раздаёт артефакты для команды разработки: сборщик публикует пакет, а IDE, CI-агенты и пользователи скачивают его по тому же URL, не выходя за периметр компании.

Все артефакты лежат в **репозиториях**, репозитории сгруппированы в **проекты**. Это та же иерархия, что и в интерфейсе:

```
›
Project(например, backend-team)
  ›
  Repository(например, maven-central)
    ›
    Artifact(например, commons-lang3:3.12.0)
```

Проект — это контейнер, в котором живут репозитории команды. Репозиторий — конкретное хранилище одного формата (Maven, npm,

Docker и т.д.) одного из двух типов. Артефакт — то, что в нём лежит и что качают клиенты.

#Два типа репозиториев

CodeScoring.Save поддерживает только два типа репозиториев, и каждый решает свою задачу.

Proxy-репозиторий — кэширующий прокси к внешнему источнику (например, Maven Central или npmjs.com). Когда клиент впервые запрашивает артефакт, Save качает его из upstream, кэширует и отдаёт. Все последующие запросы идут уже из кэша — это ускоряет сборки и снижает зависимость от внешних сервисов. На proxy-репозитории дополнительно работают политики безопасности OSA Proxy (если он настроен), которые могут заблокировать скачивание небезопасных компонентов.

Hosted-репозиторий — собственное хранилище, в которое команда публикует свои артефакты. Сюда попадают внутренние библиотеки, проверенные сторонние компоненты и build-artifacts.

В одном проекте может быть произвольное количество репозиториев обоих типов.

#URL-схема доступа

Внешний клиент (Maven, npm, Docker и т.д.) обращается к репозиторию по URL вида:

```
https://save.example.com/<format>/<project>/<repository>/
<artifact-path>
```

Формат	Шаблон
Maven	`https://save.example.com/maven/<project>/<repository>/<group>/<artifact>/<version>/<file>`
npm	`https://save.example.com/npm/<project>/<repository>/<package>`
NuGet	`https://save.example.com/nuget/<project>/<repository>/v3/index.json`
PyPI	`https://save.example.com/pypi/<project>/<repository>/simple/`
Go	`https://save.example.com/go/<project>/<repository>` (как GOPROXY)

Формат	Шаблон
Raw	`https://save.example.com/raw/<project>/<repository>/<filepath>`
OCI / Docker	`https://save.example.com/v2/<project>/<repository>/<image>/...`

Для OCI / Docker используется стандартный префикс `/v2/`, как требует OCI Distribution Spec. Дополнительно поддерживается Nexus-совместимая маршрутизация плоских URL — см. Работа с OCI / docker.

#Структура веб-интерфейса

Слева расположена вертикальная навигация со следующими разделами:

- **Projects** — основной раздел: список проектов, внутри каждого — список репозиториев, внутри каждого репозитория — дерево артефактов;
- **Cleanup** — политики автоматической очистки ненужных артефактов;
- **Settings** — управление учётными записями, ролями, сервисными аккаунтами, конфигурацией и журналом аудита.

В верхней части интерфейса находится **глобальный поиск** — открывается по клавише `/` и ищет одновременно по проектам, репозиториям и артефактам с подсказками в реальном времени.

#Базовый сценарий: подключить свой первый репозиторий

#Шаг 1. Создайте проект

Проект — это контейнер, в котором будут жить репозитории. Удобно делить проекты по командам, продуктам или окружениям.

1. В боковом меню выберите **Projects**.
2. Нажмите кнопку **Create project** в правом верхнем углу.
3. Заполните форму:
    - **Name** — человеко-читаемое название проекта (например, `Backend Team`);
    - **Color** — цвет для визуального различения проектов в списке;
    - **Key** — URL-safe идентификатор проекта (например, `backend-team`). Если оставить пустым, будет сгенерирован автоматически из Name, даже если Name не использует латиницу. **Изменить Key после создания нельзя**;
    - **Description** — необязательное описание проекта;

    - **Cleanup policy** — раздел двухпанельного селектора с уже созданными политиками очистки. Можно оставить пустым и привязать политики позже.
4. Нажмите **Create project**.

После создания откроется страница проекта со списком репозиториев (пока пустым) и его метаданными в шапке.

Key и Name

Name отображается в интерфейсе и может быть изменено в любой момент. Key участвует во всех URL и API-путях, поэтому его смена потребовала бы переподписки клиентов.

#Шаг 2. Создайте репозиторий в проекте

Внутри проекта создаётся репозиторий — конкретное хранилище для артефактов одного формата.

1. Откройте только что созданный проект.
2. Нажмите **Create repository** в правом верхнем углу страницы проекта.
3. Заполните общие поля:
    - **Name** — человеко-читаемое название (например, `Maven Central (proxy)`);
    - **Color** — цвет для удобства;
    - **Key** — URL-safe идентификатор (например, `maven-central`). Изменить после создания нельзя;
    - **Description** — необязательное описание.
4. Выберите **Format** — один из поддерживаемых пакетных форматов: `maven`, `npm`, `docker` (OCI), `nuget`, `pypi`, `go`, `raw`. После создания формат поменять нельзя.
5. Выберите **Type**:
    - **Proxy** — для проксирования внешнего реестра;
    - **Hosted** — для собственного хранилища.
6. Если выбран тип **Proxy**, появятся дополнительные поля:
    - **Proxy URL** — адрес upstream-репозитория (например, `https://repo1.maven.org/maven2/`);
    - **Cache TTL, seconds** — время жизни кэша метаданных в секундах. Для типичных внешних реестров достаточно `86400` (сутки).
7. При необходимости привяжите **Cleanup policy** — список уже созданных политик автоматической очистки.
8. Нажмите **Create repository**.

После создания откроется страница репозитория. В шапке отображается статус (Enabled/Disabled), формат, тип, даты создания и обновления, а для proxy-репозитория — кликабельная ссылка **Remote URL**, ведущая на upstream.

Изменение статуса репозитория

Статус (Enabled/Disabled) задаётся на форме редактирования: откройте репозиторий, нажмите **Edit repository** в шапке, переключите верхний радио-переключатель **Status**, нажмите **Save**. Disabled-репозиторий не отдаёт и не принимает запросы, но не удаляется и не теряет своих артефактов.

#Шаг 3. Получите готовые сниппеты для подключения клиентов

На странице репозитория в правой части шапки находится кнопка со значком «звено цепи». По клику она открывает popover **Useful snippets** — набор готовых фрагментов конфигурации и команд, который зависит от формата репозитория.

1. Нажмите кнопку **звено цепи** в шапке репозитория.
2. В появившейся карточке прокрутите список сниппетов:
   - каждый сниппет состоит из **заголовка** (например, `settings.xml`, `.npmrc`, `pip.conf`, `docker login`), блока кода с подсветкой синтаксиса и краткого описания;
   - набор сниппетов формируется на стороне сервера по формату и типу репозитория. Для Maven обычно показывается фрагмент `<mirror>` для `settings.xml` и `<repository>` для `pom.xml`; для Docker — `docker login` и `docker pull`; для npm — строка для `.npmrc`; и так далее.
3. Нажмите кнопку **Copy** справа от нужного сниппета — он скопируется в буфер обмена.
4. Вставьте сниппет в конфигурационный файл клиента или выполните команду в терминале.

#Шаг 4. Опубликуйте и скачайте первый артефакт

Дальше всё происходит уже на стороне клиента — Save принимает стандартные запросы пакетного менеджера. Для hosted-репозитория обычный путь — публикация через `mvn deploy`, `npm publish`, `twine upload` или `docker push`. Для proxy-репозитория — обычный pull, после которого артефакт окажется в кэше Save.

При первой загрузке может потребоваться аутентификация — см. раздел Подключение клиентов и аутентификация.

После того, как клиент впервые что-то загрузит или скачает, артефакт появится в дереве на странице репозитория.

#Работа с артефактами в репозитории

Страница репозитория состоит из двух колонок: слева — дерево артефактов с поиском и сортировкой, справа — детали выбранного артефакта.

#Просмотр дерева

Дерево показывает структуру репозитория: папки и файлы с иконками. Файлы со значком «замок» — это «защищённые» (locked/release) артефакты, которые нельзя удалить или перезаписать без дополнительных шагов. Раскрытие папок происходит лениво: дочерние элементы подгружаются по клику.

При клике на файл его детали открываются справа: имя, размер, путь, контрольная сумма, метаданные формата (например, Maven `groupId`/`artifactId`/`version` или Docker tags).

#Поиск, сортировка и фильтры

Над деревом расположено поле **Search artifacts**: введите подстроку и нажмите Enter — появится плоский список найденных артефактов с подсветкой контекста. Клик по строке откроет деталки соответствующего артефакта.

Рядом с поиском расположены две кнопки.

Кнопка **Sort** управляет сортировкой дерева:

- **Sort by** — `Name`, `Date added` или `Date modified`;
- **Sort direction** — `A to Z` или `Z to A`.

Кнопка-фильтр (иконка воронки) для proxy-репозиториев открывает дополнительный переключатель **Show uncached artifacts**. По умолчанию дерево показывает только то, что уже лежит в локальном кэше. С включённым флагом Save также подгружает в дерево артефакты, известные upstream, но ещё не скачанные — это удобно, чтобы предварительно посмотреть, что доступно в проксируемом источнике, не запуская реальный pull. *Это доступно **не** для всех форматов репозиториев* из-за различия протоколов.

#Действия над одним артефактом

После выбора артефакта в правой панели доступны:

- **Download** — скачивание файла локально;
- **Lock artifact** — пометить артефакт как `release`. После lock артефакт нельзя удалить или перезаписать обычным `upload` — это нужно, чтобы выпущенные версии не могли быть подменены;
- **Delete artifact** — удалить файл из репозитория (для hosted) или из кэша (для proxy).

Lock — необратимая операция в UI

Снять флаг release можно только через API:
`PUT /api/v1/artifacts/release?id=<artifact-id>` с `{"is_release": false}`.

#Групповые действия

Слева от каждой строки дерева есть чекбокс. После выделения нескольких файлов в верхней части появляется панель с действиями:

- **Lock** — массово зафиксировать выделенные артефакты;
- **Delete** — массово удалить.

Обе операции атомарны для всей выборки.

#Подключение клиентов и аутентификация

Save поддерживает несколько способов аутентификации, чтобы покрыть как интерактивные клиенты, так и CI-конвейеры. Конкретный способ выбирается клиентом самостоятельно через заголовок `Authorization` (или специальные заголовки формата).

Способ	Заголовок	Тип в cs-auth	Когда применять
Basic Auth (пользователь)	`Authorization: Basic base64(<username>:<password>)`	`basic`	Интерактивная работа из IDE, ручные `curl` и `mvn`/`pip`/`npm` от имени конкретного пользователя
Basic Auth (robot-аккаунт)	`Authorization: Basic base64(sa$<robot-name>:<api-key>)`	`api_key`	**CI/CD-пайплайны**, runners сборки, любые сервисные интеграции
Bearer JWT	`Authorization: Bearer <jwt>`	`bearer_jwt`	Сессионные запросы из веб-интерфейса и других сервисов после логина через cs-auth
Bearer (opaque token)	`Authorization: Bearer <opaque-token>`	`npm_token`	Любой не-JWT bearer-токен; на практике — npm-клиент после `npm adduser` / `npm login`
`X-NuGet-ApiKey`	`X-NuGet-ApiKey: <api-key>`	`nuget_key`	Только NuGet-формат, для

Способ	Заголовок	Тип в cs-auth	Когда применять
			совместимости с `dotnet nuget push`
Docker v2 token	`Authorization: Bearer <docker-jwt>`	`docker_token`	OCI-клиенты после прохождения цикла `401 → retry → 200`

Bearer-токен с тремя сегментами через точку классифицируется как JWT, остальные — как opaque npm-токен. Префикс sa$ в Basic Auth — это маркер service-аккаунта (robot).

#Robot-аккаунты для CI/CD

Для интеграции с CI/CD не рекомендуется использовать личные пароли — они привязаны к сотруднику и быстро устаревают. Вместо этого создаётся **robot-аккаунт** — служебная учётная запись с отдельной ролью и долгоживущим API-ключом.

#Создание через веб-интерфейс

1. В боковом меню откройте `Settings -> Robot accounts`.
2. Нажмите **Create robot account** в правом верхнем углу.
3. Заполните поля:
    - **Login** — служебное имя для аутентификации (например, `ci-builder`). Save сам добавит к введённому значению префикс для service-аккаунта;
    - **Name** — человеко-читаемое название (например, `CI Builder Bot`);
    - **Description** — необязательное описание;
    - **Expires in** — дата истечения ключа. Под полем доступны быстрые пресеты `Week`, `Month`, `Year`. Если ключ должен жить вечно, поставьте флаг **Never** справа от выбора даты — поле выбора даты автоматически блокируется;
    - **Global permissions** — глобальные разрешения для робота;
    - **Scoped permissions** — разрешения, ограниченные одним, несколькими (или всеми сразу) проектами или репозиториями. Переключатель **Scope** выбирает уровень (Projects / Repositories), затем для каждого scope добавляются записи `Add project permissions` / `Add repository permissions`.
4. Нажмите **Create robot account**.

После успешного создания на экране автоматически открывается модальное окно с API-ключом — это пароль для Basic Auth.

#Окно с API-ключом

После нажатия **Create robot account** Save открывает окно с API-ключом. В тем доступны:

- **API key** — поле в режиме «только для чтения» с самим значением ключа и кнопкой **Copy**, копирующей его в буфер обмена;
- **Key prefix** — короткий префикс ключа. По нему можно опознать ключ в журнале аудита, не раскрывая полный секрет;
- **Expires at** — дата истечения (если она задана);
- **Warning** — текст с сервера, например `Store this API key securely — it cannot be retrieved again.`

Под секретом находится кнопка **I have copied the key**, которая закрывает окно и переводит на страницу деталей созданного robot-аккаунта.

API-ключ показывается только один раз

После закрытия окна посмотреть или восстановить тот же ключ нельзя — Save хранит только хэш. Скопируйте ключ и сохраните его в secret-storage CI (GitLab CI variables, GitHub Actions secrets, HashiCorp Vault) до того, как закроете окно. Если ключ утерян, перевыпустите его (вручную в форме редактирования robot-аккаунта или через API `POST /admin/robots/<id>/rotate-key`) — после ротации старое значение становится недействительным.

#Создание через API

```
curl -X POST https://save.example.com/api/v1/admin/robots \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <admin-jwt>" \
  -d '{
    "username": "ci-builder",
    "display_name": "CI Builder Bot",
    "description": "Robot account for CI/CD pipelines",
    "permissions": {
      "project": {
        "backend-team": ["project_artifacts.download",
"project_artifacts.upload"]
      }
    },
    "key_expires_in_seconds": 7776000
  }'
```

В ответе вернётся `api_key`, который выдаётся **только один раз** при создании — сохраните его в secret-storage CI.

#Использование в клиенте

`username = sa$<robot-name>`, `password = <api-key>` — обычный Basic Auth:

```
curl -u "sa\$ci-builder:<api-key>" https://save.example.com/...
```

Экранирование

$ в shell В bash символ $ в одинарных кавычках сохраняется как есть, а в двойных — экранируется как \$. В YAML / TOML / XML конфигах экранирование не требуется.

API-ключ NuGet — особенность реализации

Заголовок `X-NuGet-ApiKey` поддерживается напрямую: cs-auth классифицирует его как тип `nuget_key` и валидирует по 12-символьному префиксу ключа. Для NuGet работают оба варианта, но мы рекомендуем Basic Auth с robot-аккаунтом — это единообразно с остальными форматами, и в журнале аудита явно отражается личность robot'а.

#Анонимный доступ на чтение

Если включён глобальный флаг `AllowAnonymousRead`, неаутентифицированным запросам выдаётся `PermissionSet` с правами `projects.view`, `project_repos.view`, `project_artifacts.view`, `project_artifacts.download` для всех проектов. Для Docker / OCI этот режим всё равно требует цикла `401 → /v2/token → 200`, иначе Docker-клиент не сможет работать (см. Работа с OCI / docker).

Текущее значение флага видно в `Settings -> Configuration`. Изменение значения выполняется через API:

```
curl -X PUT https://save.example.com/api/v1/admin/config \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <admin-jwt>" \
  -d '{"key": "AllowAnonymousRead", "value": "true"}'
```

#Политики очистки

Чтобы хранилище не разрасталось бесконечно, на репозитории привязываются **политики очистки** — правила автоматического удаления (или удержания) артефактов. Save поддерживает пять типов политик; для большинства задач достаточно простых, для сложных AND/OR-условий используется тип `expression`.

#Создание политики

1. В боковом меню откройте **Cleanup**.
2. Нажмите **Create policy**.

3. Заполните общие поля: **Display name**, **Description**, при необходимости — **Schedule** (cron-выражение из пяти полей: минуты, часы, день, месяц, день недели).
4. Выберите **Policy type** — определяет основное правило очистки:

Тип политики	Назначение	Параметр `value`
`delete-snapshots-older-than`	Удалять SNAPSHOT/dev-версии старше N дней	количество дней, например `30`
`keep-latest-versions`	Оставлять только N последних версий каждого артефакта, остальное удалять	N, например 5
`delete-by-age`	Удалять любые артефакты старше N дней	количество дней
`delete-by-size`	Удалять самые старые артефакты при превышении лимита размера	размер в гигабайтах целым числом, например `50`
`expression`	Гибкое DSL-правило с AND/OR-комбинацией критериев	вместо `value` передаётся объект `expression`

1. Укажите **Formats**, к которым применяется политика. Если оставить пустым, политика будет применяться ко всем форматам.
2. Поставьте флаг **Active**, чтобы политика начала работать сразу.
3. Для типа `expression` дополнительно соберите дерево критериев — каждый критерий это сочетание `type`, оператора сравнения (`eq`, `ne`, `gt`, `lt`, `matches`, `contains`) и значения; критерии можно объединять группами с логическим оператором `AND` или `OR`.
4. Нажмите **Create policy**.

#Доступные критерии для `expression`

Полный список — `GET /api/v1/enums/cleanup-criterion-types`.

Группа	`type`	Назначение
Возраст	`created_before`	Возраст артефакта (дней)
	`last_downloaded_days`	Дней с момента последней загрузки
	`not_downloaded_since`	

Группа	type	Назначение
		Никогда не скачивался или не скачивался с указанной даты
Версия	`version_pattern`	Glob-паттерн по версии
	`is_snapshot`, `is_prerelease`, `is_release`	Маркеры snapshot / pre-release / release
Имя/путь	`name_pattern`, `path_pattern`	Glob-паттерны
Размер	`size_greater_than`, `size_less_than`	Размер артефакта
Docker	`docker_tag`, `docker_untagged`	Тег / отсутствие тега
Maven	`maven_classifier`, `maven_packaging`	Classifier / packaging
npm	`npm_scope`	Скоуп пакета
PyPI	`pypi_package_type`	`sdist`, `bdist_wheel`, ...
NuGet	`nuget_is_prerelease`	NuGet-флаг pre-release
OCI	`oci_artifact_type`	Тип OCI-артефакта (Helm, Cosign, SBOM, ...)
Retention	`keep_latest`	Оставлять N последних версий

#Привязка политики к репозиторию

Привязка происходит на форме создания или редактирования репозитория (или проекта — тогда политика каскадно применяется ко всем его репозиториям).

1. Откройте репозиторий и нажмите **Edit** в шапке.
2. Прокрутите до раздела **Cleanup policy**.
3. В левой колонке (**Available**) — все существующие политики, в правой (**Applied**) — уже привязанные. Перенесите нужные политики стрелкой между колонками.
4. Сохраните изменения.

#Ручной запуск и предварительный просмотр через API

```
## Сухой прогон — посмотреть, что было бы удалено, без
сохранения политики
curl -X POST https://save.example.com/api/v1/cleanup/preview \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "repository_id": 42,
    "policy_type": "delete-snapshots-older-than",
    "value": "30"
  }'
```

```
## Запуск уже сохранённых политик (с опциональным dry_run)
curl -X POST "https://save.example.com/api/v1/cleanup/execute?
project=backend-team" \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "policies": ["delete-old-snapshots", "keep-latest-5"],
    "dry_run": true
  }'
```

#Управление доступом

Управление пользователями, ролями и правами на проекты и репозитории выполняется отдельным сервисом **cs-auth**. API проксируется через Save под префиксами `/api/v1/auth/*` и `/api/v1/admin/*`. Способы аутентификации описаны выше в разделе Подключение клиентов и аутентификация.

В веб-интерфейсе нужные разделы находятся в `Settings`:

- **Roles** — описания ролей с набором разрешений;
- **Users** — учётные записи людей;
- **Robot accounts** — служебные учётные записи (см. выше).

API cs-auth

Конкретные эндпоинты для управления пользователями, ролями и project membership — это поверхность API сервиса cs-auth. Save прозрачно их форвардит, но в своём swagger не описывает.

#Уровни доступа

Права в CodeScoring.Save гранулярны, в формате `<resource>.<action>`. Роли в cs-auth определяются пользователем и представляют собой набор таких разрешений.

На уровне репозитория (артефакты внутри репозитория):

Permission	Что разрешает
`artifacts.view`	Просмотр списка и метаданных артефактов
`artifacts.download`	Скачивание артефактов
`artifacts.upload`	Публикация артефактов (актуально для hosted)
`artifacts.delete`	Удаление артефактов
`artifacts.lock`	Lock/unlock одного или всех артефактов

На уровне репозитория (сам репозиторий):

Permission	Что разрешает
`repos.view`	Просмотр настроек репозитория и статистики
`repos.edit`	Изменение настроек репозитория
`repos.delete`	Удаление репозитория

На уровне проекта (cascade-permissions, распространяются на все репозитории и артефакты проекта):

Permission	Что разрешает
`projects.view`, `projects.create`, `projects.edit`, `projects.delete`	Управление проектами
`project_repos.view`, `project_repos.create`, `project_repos.delete`, `project_repos.edit`	Управление всеми репозиториями проекта
`project_artifacts.view`, `project_artifacts.upload`, `project_artifacts.download`, `project_artifacts.delete`, `project_artifacts.lock`, `project_artifacts.unlock`	Все операции с артефактами в любом репозитории проекта

Глобальные (только через global scope роли):

Permission	Что разрешает
`cleanup.view`, `cleanup.edit`, `cleanup.delete`	Управление политиками очистки
`roles.create`, `roles.delete`	Управление ролями
`users.create`, `users.delete`	Управление пользователями
`config.view`, `config.manage`	Просмотр и редактирование конфигурации
`audit.view`	Просмотр журнала аудита

Cascade-логика

Project-scope permissions автоматически cascade на repository-scope: пользователь с `project_artifacts.download` на проекте получит право `artifacts.download` на любой репозиторий внутри этого проекта. Для wildcard-доступа ко всем проектам / репозиториям используется ключ *.

#Привязка пользователя или robot-аккаунта к проекту

Привязка участника к проекту — это назначение конкретной роли конкретному пользователю или robot-аккаунту в рамках одного проекта. Назначенный участник получает все разрешения роли, а в шапке проекта он отображается как member.

#Через веб-интерфейс

Раздел в разработке

Раздел **Members** на странице проекта пока находится в разработке. Backend поддерживает все необходимые операции (`/api/v1/admin/projects/<project>/members`), но в UI вкладки **Members** на странице проекта ещё нет.

До появления вкладки выполните операции через API ниже.

#Через API

Добавление участника:

```
curl -X POST https://save.example.com/api/v1/admin/projects/
backend-team/members \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <admin-jwt>" \
  -d '{
```

```
    "user_id": 5,
    "role_id": 2
  }'
```

Для пользователей `role_id` обязателен, для robot-аккаунтов (флаг `is_service=true`) — может быть опущен.

Просмотр списка участников:

```
curl -u "<admin-username>:<admin-password>" \
  "https://save.example.com/api/v1/admin/projects/backend-team/
members?limit=50"
```

Смена роли существующему участнику:

```
curl -X PUT https://save.example.com/api/v1/admin/projects/
backend-team/members/5 \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer <admin-jwt>" \
  -d '{"role_id": 3}'
```

Удаление участника из проекта:

```
curl -X DELETE https://save.example.com/api/v1/admin/projects/
backend-team/members/5 \
  -H "Authorization: Bearer <admin-jwt>"
```

#Те же действия через API

Каждое действие, описанное выше для интерфейса, доступно и через REST API. Это полезно для скриптов первоначальной настройки, инфраструктуры как кода (Terraform / Ansible) и интеграционных тестов.

#Создание проекта

```
curl -X POST https://save.example.com/api/v1/projects \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "name": "backend-team",
    "display_name": "Backend Team",
    "description": "Backend services and microservices",
    "color": "#4A90D9"
  }'
```

#Создание Proxy-репозитория

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "backend-team",
    "name": "maven-central",
    "display_name": "Maven Central (proxy)",
    "format": "maven",
    "repository_type": "proxy",
    "remote_url": "https://repo1.maven.org/maven2/",
    "cache_ttl": 86400
  }'
```

#Создание Hosted-репозитория

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "backend-team",
    "name": "internal-maven",
    "display_name": "Internal Maven",
    "format": "maven",
    "repository_type": "hosted"
  }'
```

#Назначение политики очистки

```
curl -X POST "https://save.example.com/api/v1/cleanup/
assignments?project=backend-team" \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "policies": [
      {"policy_name": "delete-old-snapshots", "priority": 10,
"enabled": true}
    ]
  }'
```

#Создание политики очистки

```
curl -X POST https://save.example.com/api/v1/cleanup/policies \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
```

```
    "name": "delete-old-snapshots",
    "display_name": "Delete Old Maven Snapshots",
    "description": "Удалять SNAPSHOT-версии Maven старше 30
дней",
    "policy_type": "delete-snapshots-older-than",
    "value": "30",
    "formats": ["maven"],
    "enabled": true,
    "schedule": "0 2 * * *"
  }'
```

#Создание политики на основе выражения

```
curl -X POST https://save.example.com/api/v1/cleanup/policies \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "name": "clean-old-prereleases",
    "display_name": "Clean Old Pre-releases",
    "policy_type": "expression",
    "formats": ["maven", "npm"],
    "expression": {
      "action": "delete",
      "logical_operator": "and",
      "criteria": [
        {"type": "is_prerelease", "value": "true"},
        {"type": "created_before", "value": "14"}
      ]
    },
    "enabled": true
  }'
```

#Мониторинг

#Журнал аудита

Журнал аудита фиксирует все изменения в проектах, репозиториях, артефактах, ролях и robot-аккаунтах. В UI он доступен:

- **Глобально** — `Settings -> Audit log`;
- **По проекту** — на странице проекта в шапке справа есть иконка журнала, ведущая на отфильтрованный по проекту журнал.

Каждая запись содержит время события, инициатора, тип ресурса и его идентификатор, тип действия и набор полей с подробностями (имя загруженного артефакта, изменённые параметры репозитория и т.п.). Записи группируются по дням и отображаются в виде таймлайна, а ссылки внутри записи кликабельны (открывается соответствующий проект, репозиторий, политика, пользователь или роль).

Журнал можно выгрузить за период через кнопку **Export** в шапке (форматы JSON или HTML).

```
## Аудит-журнал по конкретному репозиторию через API
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=maven-central&limit=50"
```

#Статистика репозитория

Текущие агрегированные показатели по репозиторию доступны через API:

```
curl https://save.example.com/api/v1/repos/<repository_id>/
stats \
  -u "<username>:<password>"
```

Ответ:

```
{
  "id": 42,
  "name": "maven-central",
  "project_name": "backend-team",
  "artifact_count": 1523,
  "total_size": 16413032448,
  "cache_hit_ratio": 0.85,
  "last_activity": "2026-03-17T10:30:00Z"
}
```

Поля статистики

Эндпоинт возвращает только агрегированные показатели. Поле `cache_hit_ratio` присутствует только для proxy-репозиториев.

#Состояние сервиса

```
curl https://save.example.com/health
```

Глобальный health-check сервиса. Отдельных health-эндпоинтов на репозиторий нет: доступность upstream проверяется фоновыми воркерами и отражается через метрики Prometheus.

#Логи

Логи централизованные, в формате JSON. Фильтрация по конкретному репозиторию — через стандартные средства log-aggregator'a.

#Работа с Go

CodeScoring.Save реализует **Go Module Proxy Protocol** с префиксом `/go/<project>/<repository>/`. Совместим со стандартным Go toolchain (`go mod`, `go build`, `go install`).

#Proxy-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "backend",
    "name": "go-proxy",
    "format": "go",
    "repository_type": "proxy",
    "remote_url": "https://proxy.golang.org",
    "cache_ttl": 86400
  }'
```

#Hosted-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "backend",
    "name": "go-hosted",
    "format": "go",
    "repository_type": "hosted"
  }'
```

Hosted-репозиторий принимает PUT-загрузку модулей по URL вида:

```
PUT /go/<project>/go-hosted/<module>/@v/<version>.zip
```

Module paths используют case-encoding согласно Go Module Proxy spec: заглавные буквы заменяются на `!` + соответствующую строчную (например, `Acme` → `!acme`).

#Настройка клиента

```
## Установка GOPROXY
export GOPROXY="https://save.example.com/go/<project>/go-proxy"

## Если используется Basic Auth — credentials передаются
через .netrc:
```

```
cat >> ~/.netrc << EOF
machine save.example.com
login <username>
password <password>
EOF
chmod 600 ~/.netrc
```

Цепочка прокси

GOPROXY может содержать несколько URL через запятую: `GOPROXY="https://save.example.com/go/<project>/go-proxy,direct"`. Go toolchain попробует каждый прокси по очереди; `direct` означает обращение напрямую к VCS.

Robot-аккаунты в CI

Для CI/CD используйте robot-аккаунт: `login = sa$<robot-name>`, `password = <api-key>` в `~/.netrc`. Подробнее — в общем разделе Аутентификация.

GOPRIVATE для внутренних модулей

Если в репозитории хранятся внутренние модули, отсутствующие в публичном sumdb, добавьте их паттерны в `GOPRIVATE`.

```
export GOPRIVATE="git.example.com/*"
## При необходимости можно полностью отключить sumdb:
## export GOSUMDB=off
```

#Миграция URL репозитория

Сценарий использования: миграция Go-прокси с Nexus / Artifactory на CodeScoring.Save.

Источник	GOPROXY до миграции	GOPROXY после миграции
Nexus	`https://nexus.host.ru/repository/go-remote`	`https://save.example.com/go/<project>/go-proxy`
Artifactory	`https://jfrog.host.ru/artifactory/api/go/go-virt`	`https://save.example.com/go/<project>/go-proxy`
Официальный proxy	`https://proxy.golang.org`	`https://save.example.com/go/<project>/go-proxy`

#Устранение неполадок

#Проверка списка версий

```
curl -u "<username>:<password>" \
  https://save.example.com/go/<project>/go-proxy/github.com/
gin-gonic/gin/@v/list
```

#Состояние сервиса

```
curl https://save.example.com/health
```

#Аудит по репозиторию

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=go-proxy&limit=50"
```

#Работа с Maven

CodeScoring.Save поддерживает Maven 2 layout и совместим с любым JVM-инструментарием поверх него: `mvn`, `gradle`, `ant + Ivy` / `maven-ant-tasks`, `sbt` и т. д.

#Proxy-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "backend",
    "name": "maven-central-proxy",
    "format": "maven",
    "repository_type": "proxy",
    "remote_url": "https://repo1.maven.org/maven2/"
  }'
```

#Hosted-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "backend",
```

```
    "name": "internal-maven",
    "format": "maven",
    "repository_type": "hosted"
  }'
```

#Настройка клиента

#mvn

В `~/.m2/settings.xml`:

```
<settings>
  <mirrors>
    <mirror>
      <id>save-maven</id>
      <name>CodeScoring Save Maven</name>
      <url>https://save.example.com/maven/<project>/maven-
central-proxy/</url>
      <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>

  <servers>
    <server>
      <id>save-maven</id>
      <username><username></username>
      <password><password></password>
    </server>
  </servers>
</settings>
```

Публикация артефактов (deploy в hosted) — добавьте в `pom.xml`:

```
<distributionManagement>
  <repository>
    <id>save-maven</id>
    <url>https://save.example.com/maven/<project>/internal-
maven/</url>
  </repository>
</distributionManagement>
```

```
mvn deploy
```

#gradle

В `build.gradle`:

```
repositories {
    maven {
        url 'https://save.example.com/maven/<project>/maven-
central-proxy/'
        credentials {
            username = project.findProperty('saveUser') ?:
System.getenv('SAVE_USER')
            password = project.findProperty('savePassword') ?:
System.getenv('SAVE_PASSWORD')
        }
    }
}

publishing {
    repositories {
        maven {
            name = 'codescoring-save'
            url = 'https://save.example.com/maven/<project>/
internal-maven/'
            credentials {
                username = project.findProperty('saveUser') ?:
System.getenv('SAVE_USER')
                password = project.findProperty('savePassword')
 ?: System.getenv('SAVE_PASSWORD')
            }
        }
    }
}
```

Или в Kotlin DSL (`build.gradle.kts`):

```
repositories {
    maven {
        url = uri("https://save.example.com/maven/<project>/
maven-central-proxy/")
        credentials {
            username = (findProperty("saveUser") ?: System.get
env("SAVE_USER")) as String
            password = (findProperty("savePassword") ?:
System.getenv("SAVE_PASSWORD")) as String
        }
    }
}
```

Credentials удобно хранить в `~/.gradle/gradle.properties`:

```
saveUser=<username>
savePassword=<password>
```

Robot-аккаунты в CI

Для CI/CD используйте robot-аккаунт: `username = sa$<robot-name>`, `password = <api-key>`. Структура `settings.xml` и `gradle.properties` не меняется — отличаются только значения. Подробнее — в общем разделе Аутентификация.

#ant

Подход 1 — через `maven-ant-tasks`:

```
<project xmlns:artifact="urn:maven-artifact-ant">
  <typedef resource="org/apache/maven/artifact/ant/antlib.xml"
           uri="urn:maven-artifact-ant"
           classpathref="maven-ant-tasks.classpath" />

  <artifact:remoteRepository id="codescoring"
                             url="https://save.example.com/
maven/<project>/maven-central-proxy/">
    <authentication username="<username>"
password="<password>" />
  </artifact:remoteRepository>

  <artifact:dependencies pathId="dependency.classpath">
    <remoteRepository refid="codescoring" />
    <dependency groupId="org.apache.commons" artifactId="commo
ns-lang3" version="3.12.0" />
  </artifact:dependencies>
</project>
```

Подход 2 — через Apache Ivy с `ivysettings.xml`:

```
<ivysettings>
  <settings defaultResolver="codescoring" />
  <credentials host="save.example.com"
               realm="Sonatype Nexus Repository Manager"
               username="<username>"
               passwd="<password>" />
  <resolvers>
    <ibiblio name="codescoring"
             m2compatible="true"
             root="https://save.example.com/maven/<project>/
maven-central-proxy/" />
  </resolvers>
</ivysettings>
```

#sbt

В `~/.sbt/1.0/global.sbt`:

```
resolvers += "codescoring" at "https://save.example.com/maven/
<project>/maven-central-proxy/"
credentials += Credentials(
  "CodeScoring Save",
  "save.example.com",
  sys.env.getOrElse("SAVE_USER", ""),
  sys.env.getOrElse("SAVE_PASSWORD", "")
)
```

#Миграция URL репозитория

Сценарий использования: миграция Maven-репозитория с Nexus / Artifactory на CodeScoring.Save.

Источник	URL в settings.xml до миграции	URL в settings.xml после миграции
Nexus	`https://nexus.host.ru/repository/maven-remote`	`https://save.example.com/maven/<project>/maven-central-proxy/`
Artifactory	`https://jfrog.host.ru/artifactory/maven-remote`	`https://save.example.com/maven/<project>/maven-central-proxy/`
Официальный репозиторий	`https://repo.maven.apache.org/maven2`	`https://save.example.com/maven/<project>/maven-central-proxy/`

Параметры аутентификации (имя пользователя, пароль) и <servers>-блок переносятся без изменений.

#Устранение неполадок

#Проверка метаданных пакета

```
curl -u "<username>:<password>" \
  https://save.example.com/maven/<project>/maven-central-proxy/
org/apache/commons/commons-lang3/maven-metadata.xml
```

#Состояние сервиса

```
curl https://save.example.com/health
```

#Аудит по репозиторию

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=maven-central-proxy&limit=50"
```

#Работа с NPM

CodeScoring.Save реализует npm Registry API с префиксом `/npm/<project>/<repository>/`. Совместим со стандартными клиентами `npm`, `yarn` и `pnpm`.

#Proxy-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "frontend",
    "name": "npmjs-proxy",
    "format": "npm",
    "repository_type": "proxy",
    "remote_url": "https://registry.npmjs.org",
    "cache_ttl": 3600
  }'
```

#Hosted-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "frontend",
    "name": "npm-hosted",
    "format": "npm",
    "repository_type": "hosted"
  }'
```

#Настройка клиента

#npm

```
## Установка registry в текущем проекте
echo "registry=https://save.example.com/npm/<project>/npmjs-
proxy/" > .npmrc
```

```
## Аутентификация (npm 7+, _auth — base64(username:password))
NPM_AUTH=$(echo -n "<username>:<password>" | base64)
cat >> .npmrc << EOF
//save.example.com/npm/<project>/npmjs-proxy/:_auth=${NPM_AUTH}
//save.example.com/npm/<project>/npmjs-proxy/:always-auth=true
EOF

## Установка зависимости
npm install lodash
```

Robot-аккаунты в CI

Для CI/CD используйте robot-аккаунт: `username = sa$<robot-name>`, `password = <api-key>`. Структура `.npmrc` не меняется — отличается только значение `_auth` (base64 от `sa$<robot-name>:<api-key>`). Подробнее — в общем разделе Аутентификация.

Opaque bearer-токены

npm-клиент после `npm adduser` / `npm login` сохраняет полученный opaque-токен и далее использует его как `Authorization: Bearer <token>`. cs-auth классифицирует любой не-JWT bearer-токен (Bearer-значение без трёх сегментов через точку) как `npm_token` и валидирует через сервис-аккаунты, привязанные к API-ключам.

Публикация в hosted-репозиторий:

```
cat > .npmrc << EOF
registry=https://save.example.com/npm/<project>/npm-hosted/
//save.example.com/npm/<project>/npm-hosted/:_auth=${NPM_AUTH}
//save.example.com/npm/<project>/npm-hosted/:always-auth=true
EOF

npm publish
```

#yarn

#Yarn Classic (1.x)

Yarn 1.x использует тот же `.npmrc`, что и npm:

```
echo "registry=https://save.example.com/npm/<project>/npmjs-
proxy/" > .npmrc
yarn install
```

#Yarn 2+ (Berry)

В `.yarnrc.yml`:

```
npmRegistryServer: "https://save.example.com/npm/<project>/
npmjs-proxy/"

npmRegistries:
  "https://save.example.com/npm/<project>/npmjs-proxy/":
    npmAlwaysAuth: true
    npmAuthIdent: "<username>:<password>"
```

#pnpm

pnpm читает тот же `.npmrc`:

```
echo "registry=https://save.example.com/npm/<project>/npmjs-
proxy/" > .npmrc
pnpm install
```

Для scoped-пакетов можно настроить отдельный registry:

```
echo "@mycompany:registry=https://save.example.com/npm/
<project>/npm-hosted/" >> .npmrc
```

#Миграция URL репозитория

Сценарий использования: миграция npm-репозитория с Nexus / Artifactory на CodeScoring.Save.

Источник	`.npmrc` `registry=` до миграции	`.npmrc` `registry=` после миграции
Nexus	`https://nexus.host.ru/repository/npm-proxy`	`https://save.example.com/npm/<project>/npmjs-proxy/`
Artifactory	`https://jfrog.host.ru/artifactory/api/npm/npm-remote`	`https://save.example.com/npm/<project>/npmjs-proxy/`
Официальный репозиторий	`https://registry.npmjs.org`	`https://save.example.com/npm/<project>/npmjs-proxy/`

При миграции достаточно заменить URL в `.npmrc`/`.yarnrc.yml`. Существующие credentials Nexus/Artifactory можно переиспользовать как Basic Auth.

#Устранение неполадок

#Проверка метаданных пакета

```
curl -u "<username>:<password>" \
  https://save.example.com/npm/<project>/npmjs-proxy/lodash |
jq .
```

#Whoami / ping

```
curl -u "<username>:<password>" \
  https://save.example.com/npm/<project>/npmjs-proxy/-/whoami

curl -u "<username>:<password>" \
  https://save.example.com/npm/<project>/npmjs-proxy/-/ping
```

#Состояние сервиса

```
curl https://save.example.com/health
```

#Аудит по репозиторию

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=npmjs-proxy&limit=50"
```

#Работа с NuGet

CodeScoring.Save реализует **NuGet v3 API** с префиксом `/nuget/<project>/<repository>/`. Совместим со стандартными клиентами `dotnet`, `nuget.exe`, Visual Studio и Rider.

#Proxy-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "dotnet",
    "name": "nuget-proxy",
```

```
    "format": "nuget",
    "repository_type": "proxy",
    "remote_url": "https://api.nuget.org/v3/index.json"
  }'
```

URL для remote_url

Указывайте полный URL индекса сервиса — для официального nuget.org это `https://api.nuget.org/v3/index.json`. Save сам разрешит вложенные ресурсы (flatcontainer, registration, search) по этому индексу.

#Hosted-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "dotnet",
    "name": "nuget-hosted",
    "format": "nuget",
    "repository_type": "hosted"
  }'
```

После создания индекс сервиса доступен по URL `https://save.example.com/nuget/<project>/<repository>/v3/index.json` .

#Настройка клиента

#dotnet CLI

В `nuget.config` или `NuGet.Config` (имя файла зависит от ОС):

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="codescoring"
         value="https://save.example.com/nuget/<project>/nuget-
proxy/v3/index.json"
         protocolVersion="3" />
  </packageSources>
  <packageSourceCredentials>
    <codescoring>
      <add key="Username" value="<username>" />
      <add key="ClearTextPassword" value="<password>" />
    </codescoring>
```

```
  </packageSourceCredentials>
</configuration>
```

Использование:

```
dotnet restore
dotnet add package Newtonsoft.Json
```

#nuget.exe

```
nuget sources add \
  -Name codescoring \
  -Source https://save.example.com/nuget/<project>/nuget-proxy/
v3/index.json \
  -UserName <username> \
  -Password <password>

nuget install Newtonsoft.Json -Source codescoring
```

#Публикация (hosted)

Push пакетов в hosted-репозиторий через стандартные клиенты:

```
## dotnet CLI
dotnet pack -c Release
dotnet nuget push bin/Release/MyPackage.1.0.0.nupkg \
  --source https://save.example.com/nuget/<project>/nuget-
hosted/v3/index.json \
  --api-key <api-key-or-password>

## nuget.exe
nuget push MyPackage.1.0.0.nupkg \
  -Source https://save.example.com/nuget/<project>/nuget-
hosted/v3/index.json \
  -ApiKey <api-key-or-password>
```

X-NuGet-ApiKey vs Basic Auth для CI

`dotnet nuget push` поддерживает либо API key (заголовок `X-NuGet-ApiKey`), либо Basic Auth. CodeScoring.Save поддерживает оба способа: заголовок `X-NuGet-ApiKey` классифицируется как тип `nuget_key`. Тем не менее для CI/CD рекомендуется Basic Auth с robot-аккаунтом — это единообразно с остальными форматами, и в журнале аудита явно отражается имя robot'а.

#Миграция URL репозитория

Сценарий использования: миграция NuGet-репозитория с Nexus / Artifactory на CodeScoring.Save.

Источник	URL в NuGet.Config до миграции	URL в NuGet.Config после миграции
Nexus	`https://nexus.host.ru/repository/nuget.org-proxy/index.json`	`https://save.example.com/nuget/<project>/nuget-proxy/v3/index.json`
Artifactory	`https://jfrog.host.ru/artifactory/api/nuget/v3/nuget-remote`	`https://save.example.com/nuget/<project>/nuget-proxy/v3/index.json`
Официальный репозиторий	`https://api.nuget.org/v3/index.json`	`https://save.example.com/nuget/<project>/nuget-proxy/v3/index.json`

При миграции `<packageSourceCredentials>` сохраняется без изменений.

#Устранение неполадок

#Проверка индекса сервиса

```
curl -u "<username>:<password>" \
  https://save.example.com/nuget/<project>/nuget-proxy/v3/
index.json | jq .
```

В ответе должен быть массив `resources` с ресурсами `PackageBaseAddress`, `RegistrationsBaseUrl`, `SearchQueryService` и т. д.

#Список версий пакета

```
curl -u "<username>:<password>" \
  https://save.example.com/nuget/<project>/nuget-proxy/v3-
flatcontainer/newtonsoft.json/index.json
```

#Состояние сервиса

```
curl https://save.example.com/health
```

#Аудит по репозиторию

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=nuget-proxy&limit=50"
```

#Работа с OCI / Docker

CodeScoring.Save реализует стандарт **OCI Distribution Spec** на префиксе `/v2/`, поддерживая Docker, Helm OCI-чарты и любые другие OCI-артефакты (например, `oras`).

#Proxy-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "common",
    "name": "dockerhub-proxy",
    "format": "docker",
    "repository_type": "proxy",
    "remote_url": "https://registry-1.docker.io",
    "cache_ttl": 86400
  }'
```

Для образов с одним именем (без слешей) применяется стандартная нормализация Docker Hub `library/<image>`.

#Hosted-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "common",
    "name": "docker-hosted",
    "format": "docker",
    "repository_type": "hosted"
  }'
```

#URL-схема

Базовая схема — путевая, как требует Docker / OCI:

```
<host>/<project>/<repository>/<image>:<tag>
```

Дополнительно поддерживается **Nexus-совместимая маршрутизация плоских URL** (flat-alias). Для совместимых репозиториев `<project>/<repository>` префикс заменяется коротким алиасом или вовсе опускается.

#Настройка клиента

#Docker / Podman

```
## Логин
docker login save.example.com

## Pull через proxy
docker pull save.example.com/common/dockerhub-proxy/library/
nginx:latest

## Push в hosted
docker tag myapp:latest save.example.com/common/docker-hosted/
myapp:1.0.0
docker push save.example.com/common/docker-hosted/myapp:1.0.0
```

Anonymous read и

`docker login` Endpoint `/v2/` всегда отвечает 401 с challenge-заголовком `WWW-Authenticate: Bearer realm=...`, даже если репозиторий допускает анонимное чтение. Это нужно, чтобы Docker-клиент корректно выполнил cycle `401 → retry → 200` и затем при необходимости отправил Basic Auth. Это требование OCI Distribution Spec.

Robot-аккаунты в CI

Для CI/CD используйте robot-аккаунт:
`docker login -u 'sa$<robot-name>' -p '<api-key>' save.example.com`.
То же самое работает для `helm registry login` и `oras login`.
Подробнее — в общем разделе Аутентификация.

#Helm

CodeScoring.Save принимает Helm-чарты как обычные OCI-артефакты. Push/pull выполняется стандартными командами `helm`:

```
## Логин (Helm 3.8+)
helm registry login save.example.com -u <username>

## Push
helm package ./
```

```
mychart                                                  # →
mychart-0.1.0.tgz
helm push mychart-0.1.0.tgz oci://save.example.com/common/
docker-hosted

## Pull
helm pull oci://save.example.com/common/docker-hosted/mychart
--version 0.1.0
```

Plain HTTP для тестовых стендов

Для тестовых стендов без TLS используйте флаг `--plain-http` (Helm 3.13+):

```
helm push mychart-0.1.0.tgz oci://save.example.com/common/
docker-hosted --plain-http
```

#oras

oras работает с любым OCI-совместимым артефактом — конфигами, политиками, SBOM, бинарями и т. д.:

```
## Push произвольного файла как OCI-артефакта
oras push save.example.com/common/docker-hosted/configs/app:v1
\
  -u <username> -p <password> \
  ./config.yaml:application/vnd.example.config

## Pull
oras pull save.example.com/common/docker-hosted/configs/app:v1
\
  -u <username> -p <password>
```

#containerd / nerdctl

```
nerdctl login save.example.com
nerdctl pull save.example.com/common/dockerhub-proxy/library/
alpine:latest
```

#Маршрутизация плоских URL

Save поддерживает **Nexus-совместимое плоское пространство имён** для бесшовной миграции с Nexus / Artifactory: репозиторий можно сконфигурировать с префиксом-алиасом, чтобы он отвечал на запросы без `<project>/<repository>` в пути.

Применяется longest-prefix matching:

- `docker pull save.example.com/common/external/golang:1.24.1` — попадёт в репозиторий, чей flat-alias = `common/external/`.
- `docker pull save.example.com/johnny-depp:v1` — попадёт в catch-all hosted-репозиторий (если он сконфигурирован).

При создании репозитория с flat-alias указываются параметры `flat_alias_prefix` и/или `is_catch_all`. Префиксы валидируются на уникальность.

#Миграция URL репозитория

Сценарий использования: миграция Docker-реестра из Nexus / Artifactory на CodeScoring.Save.

Источник	URL до миграции	URL после миграции (с проектом)	URL после миграции (flat-alias)
Nexus	`nexus.host.ru:5000/library/nginx:latest`	`save.example.com/common/dockerhub-proxy/library/nginx:latest`	`save.example.com/library/nginx:latest`
Artifactory	`jfrog.host.ru/docker-remote/library/nginx`	`save.example.com/common/dockerhub-proxy/library/nginx`	`save.example.com/library/nginx`
Docker Hub	`docker.io/library/postgres:16`	`save.example.com/common/dockerhub-proxy/library/postgres:16`	`save.example.com/library/postgres:16`

Flat-alias-вариант ничего не требует от клиента — старые скрипты, в которых жёстко прописано `nexus.host.ru/library/nginx`, начинают работать после простой замены хоста.

#Устранение неполадок

#Проверка `/v2/`

```
curl -i https://save.example.com/v2/
## Ожидается: HTTP/1.1 401 Unauthorized с Www-Authenticate:
Bearer realm=...
```

#Проверка эндпоинта токена

```
curl -u "<username>:<password>" \
  "https://save.example.com/v2/token?
service=save.example.com&scope=repository:common/dockerhub-
proxy/library/nginx:pull"
```

В ответе — JSON с полем token, содержащим короткоживущий Docker v2 JWT.

#Список тегов

```
curl -u "<username>:<password>" \
  https://save.example.com/v2/<project>/<repository>/<image>/
tags/list
```

#Состояние сервиса

```
curl https://save.example.com/health
```

#Аудит по репозиторию

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=dockerhub-proxy&limit=50"
```

#Работа с PyPI

#Proxy-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "backend",
    "name": "pypi-proxy",
    "format": "pypi",
    "repository_type": "proxy",
    "remote_url": "https://pypi.org/",
    "cache_ttl": 3600
  }'
```

#Hosted-репозиторий

```
curl -X POST https://save.example.com/api/v1/repos \
  -H "Content-Type: application/json" \
  -u "<username>:<password>" \
  -d '{
    "project": "backend",
    "name": "pypi-hosted",
    "format": "pypi",
    "repository_type": "hosted"
  }'
```

После создания репозиторий доступен по URL `https://save.example.com/pypi/<project>/<repository>/`. Simple-индекс — по `/simple/`, загрузка пакета (twine-совместимый POST) — на корень репозитория.

#Настройка клиента

#pip

```
## Разовая установка через флаг --index-url
pip install --index-url https://save.example.com/pypi/
<project>/pypi-proxy/simple/ <package>

## Постоянная конфигурация через pip.conf
cat > ~/.config/pip/pip.conf << EOF
[global]
index-url = https://USER:PASS@save.example.com/pypi/<project>/
pypi-proxy/simple/
trusted-host = save.example.com
EOF
```

Расположение pip.conf

- Linux/macOS: `~/.config/pip/pip.conf` или `~/.pip/pip.conf`
- Windows: `%APPDATA%\pip\pip.ini`
- В virtualenv: `$VIRTUAL_ENV/pip.conf`

Robot-аккаунты в CI

Для CI/CD используйте robot-аккаунт: `username = sa$<robot-name>`, `password = <api-key>`. Конфиг `pip.conf` не меняется — отличается только значение `username`. Подробнее — в общем разделе Аутентификация.

#poetry

В `pyproject.toml`:

```
[[tool.poetry.source]]
name = "codescoring"
url = "https://save.example.com/pypi/<project>/pypi-proxy/
simple/"
priority = "primary"
```

Credentials передаются через `poetry config`:

```
poetry config http-basic.codescoring <username> <password>
```

#pipenv

В `Pipfile`:

```
[[source]]
url = "https://USER:PASS@save.example.com/pypi/<project>/pypi-
proxy/simple/"
verify_ssl = true
name = "codescoring"
```

#uv

```
uv pip install --index-url https://save.example.com/pypi/<proj
ect>/pypi-proxy/simple/ <package>
```

Или в `pyproject.toml`:

```
[[tool.uv.index]]
name = "codescoring"
url = "https://save.example.com/pypi/<project>/pypi-proxy/
simple/"
default = true
```

#Публикация (twine, hosted)

Загрузка пакетов в hosted-репозиторий — стандартным `twine upload`. URL-адрес публикации — **корень репозитория** (без `/simple/`).

```
## ~/.pypirc
[distutils]
index-servers = codescoring

[codescoring]
repository = https://save.example.com/pypi/<project>/pypi-
hosted/
```

```
username = <username>
password = <password>
```

```
python -m build               # сгенерировать .whl и sdist в
dist/
twine upload -r codescoring dist/*
```

#Миграция URL репозитория

Сценарий использования: миграция репозитория PyPI с Nexus / Artifactory на CodeScoring.Save.

Источник	URL в pip.conf до миграции	URL в pip.conf после миграции
Nexus	`https://nexus.host.ru/repository/pypi-proxy/simple`	`https://save.example.com/pypi/<project>/pypi-proxy/simple/`
Artifactory	`https://jfrog.host.ru/artifactory/api/pypi/pypi-remote/simple`	`https://save.example.com/pypi/<project>/pypi-proxy/simple/`
Официальный репозиторий	`https://pypi.org/simple`	`https://save.example.com/pypi/<project>/pypi-proxy/simple/`

Параметры аутентификации (имя пользователя / пароль) и формат `pip.conf` сохраняются без изменений.

#Устранение неполадок

#Проверка простого индекса

```
curl -u "<username>:<password>" \
  https://save.example.com/pypi/<project>/pypi-proxy/simple/<p
ackage>/
```

Ответ — HTML-страница со списком ссылок на файлы пакета. Если страница пустая или возвращается 404 — пакет не закэширован и upstream его не вернул.

#Состояние сервиса

```
curl https://save.example.com/health
```

#Аудит по репозиторию

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=pypi-proxy&limit=50"
```

#Интеграция с CodeScoring.OSA

CodeScoring.Save можно использовать как upstream-репозиторий для OSA Proxy. В такой схеме пакетные менеджеры обращаются к OSA Proxy, OSA Proxy проверяет запрашиваемые компоненты через CodeScoring.OSA и при необходимости перенаправляет разрешенные запросы в proxy-репозиторий Save.

CodeScoring.Save не подключается к OSA Proxy через отдельный API интеграции. Для проверки загружаемых компонентов пакетные менеджеры должны быть настроены на работу через **OSA Proxy**, а сам OSA Proxy должен быть настроен на соответствующий upstream-репозиторий.

Если upstream-репозиторием выступает proxy-репозиторий CodeScoring.Save, его URL указывается в поле `registry` репозитория OSA Proxy. В этом сценарии трафик проходит по цепочке:

```
Пакетный менеджер -> OSA Proxy -> CodeScoring.Save proxy-
репозиторий -> внешний upstream
```

OSA Proxy выполняет сканирование пакетов и манифестов, обращается к CodeScoring API для проверки политик и блокирует небезопасные компоненты в зависимости от выбранного `work-mode`.

#Что настраивается в OSA Proxy

Конфигурация выполняется в сервисе OSA Proxy. Для каждого поддерживаемого формата включается соответствующая секция и задаётся список репозиториев:

```
maven:
  enabled: true
  repository:
    - name: save-maven
      scan-manifest: true
      scan-package: true
```

```
      work-mode: strict_wait
      url-encoded-config: true
      registry: https://save.example.com/maven/<project>/maven-
central-proxy

npm:
  enabled: true
  repository:
    - name: save-npm
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
      registry: https://save.example.com/npm/<project>/npmjs-
proxy

codescoring:
  url: https://codescoring.example.com
  token: "<API_TOKEN>"
  work-mode: strict_wait
  osa-proxy-url: https://osa-proxy.example.com
  block-on-codescoring-errors: true
  remove-blocked-versions: true
  block-status-code: 403
```

Для PyPI дополнительно указывается `packages-registry`, для Go — `sumdb-registry`, для Docker — `auth-token-url`, если upstream registry требует отдельный token service.

#Режимы работы

Поведение проверки управляется параметром `work-mode`. Его можно задать глобально в секции `codescoring` или переопределить на уровне отдельного репозитория.

Поддерживаются режимы:

- `warmup` — загрузка данных в кэш CodeScoring без блокировки компонентов;
- `spectator` — загрузка данных в кэш CodeScoring без блокировки компонентов, с сохранением результатов запросов компонентов в платформе;
- `moderate` — блокировка компонентов, не прошедших проверку политик; загрузка непросканированных компонентов разрешена;
- `strict` — блокировка компонентов, не прошедших проверку политик; загрузка непросканированных компонентов запрещена;
- `strict_wait` — блокировка компонентов, не прошедших проверку политик; для непросканированных компонентов выполняется ожидание проверки.

#Ограничения

Проверка через OSA Proxy применяется к трафику, который проходит через OSA Proxy. Для hosted-репозиториев Save проверку нужно выполнять до публикации артефакта или выстраивать отдельный процесс контроля.

Политики безопасности настраиваются в CodeScoring. OSA Proxy использует CodeScoring API для сканирования компонентов, получения информации о пакетах и проверки политик, но не создаёт политики внутри Save.

#См. также

- Общее описание OSA Proxy
- Настройка сервиса OSA Proxy
- Поддерживаемые протоколы OSA Proxy
- Настройка Redis и кэширования OSA Proxy

#CodeScoring.OSA

Модуль CodeScoring.OSA реализует защиту цепочки поставок через набор интеграционных компонентов, которые совместно обеспечивают автоматическое сканирование загружаемых артефактов и блокировку небезопасных пакетов. В состав CodeScoring.OSA входят как плагины для менеджеров репозиториев, так и прокси-сервис — все эти компоненты работают согласованно и дополняют друг друга:

- плагины встраиваются в обработку *request|response* на стороне менеджера репозиториев (например, Sonatype Nexus и JFrog Artifactory) и блокируют загрузку нежелательных компонентов на уровне хранилища;
- прокси-сервис перехватывает запросы пакетных менеджеров к удалённым репозиториям, выполняет сканирование и при необходимости модифицирует ответы — это удобный вариант для централизованного контроля или когда установка плагина невозможна.

Способы интеграции:

- Sonatype Nexus Repository — плагин для Nexus (встраивается в request|response-пайплайн репозитория);
- JFrog Artifactory — плагин для Artifactory (аналогичная интеграция);
- Сфера — плагин для платформы Сфера;
- OSA Proxy — прокси-сервис, перехватывающий запросы от пакетных менеджеров к upstream-репозиториям, выполняющий автоматическое сканирование, модификацию ответов и управление доступом к компонентам в соответствии с политиками безопасности;

- OSA Proxy Go — Go-версия прокси-сервиса с поддержкой расширенного набора экосистем.

#Плагин для Nexus

Поддерживаемые типы репозиториев: Alpine, CocoaPods, Composer, Conan, Conda, Debian, Docker, Go, Maven, NPM, NuGet, PyPI, RPM, RubyGems.

#Установка плагина

Плагин **CodeScoring.OSA** поставляется в виде JAR-файла и поддерживает следующие версии Sonatype Nexus Repository (NXRM):

- `nexus-codescoring-plugin-{release}.jar` - для Nexus Repository Community Edition версий с **3.71** по **3.77** и Nexus Repository Pro версий с **3.33.1-01** по **3.77** (поддерживает H2 и PostgreSQL);
- `nexus-codescoring-plugin-legacy-{release}.jar` - для Nexus Repository OSS версий с **3.33.1-01** по **3.70.X** (поддерживает OrientDB).

Для добавления плагина в **NXRM** необходимо:

1. Скопировать полученный от вендора файл `nexus-codescoring-plugin.jar` в директорию `/opt/sonatype/nexus/deploy`:

```
cp nexus-codescoring-plugin.jar /opt/sonatype/nexus/deploy/
nexus-codescoring-plugin.jar
```

Если **NXRM** запущен в Docker-контейнере:

```
docker cp nexus-codescoring-plugin.jar nexus:/opt/sonatype/
nexus/deploy/nexus-codescoring-plugin.jar
```

1. Выдать права для пользователя и группы `nexus`:

```
chown nexus:nexus /opt/sonatype/nexus/deploy/nexus-codescoring-
plugin.jar
```

Если **NXRM** запущен в Docker-контейнере:

```
docker exec -it -u 0 nexus chown nexus:nexus /opt/sonatype/
nexus/deploy/nexus-codescoring-plugin.jar
```

1. Проверить, что у пользователя есть минимальный набор привилегий для корректной работы с плагином:

```
nx-repository-view-*-*-{read,browse}
```

После выполненных операций, необходимо произвести перезапуск **NXRM**.

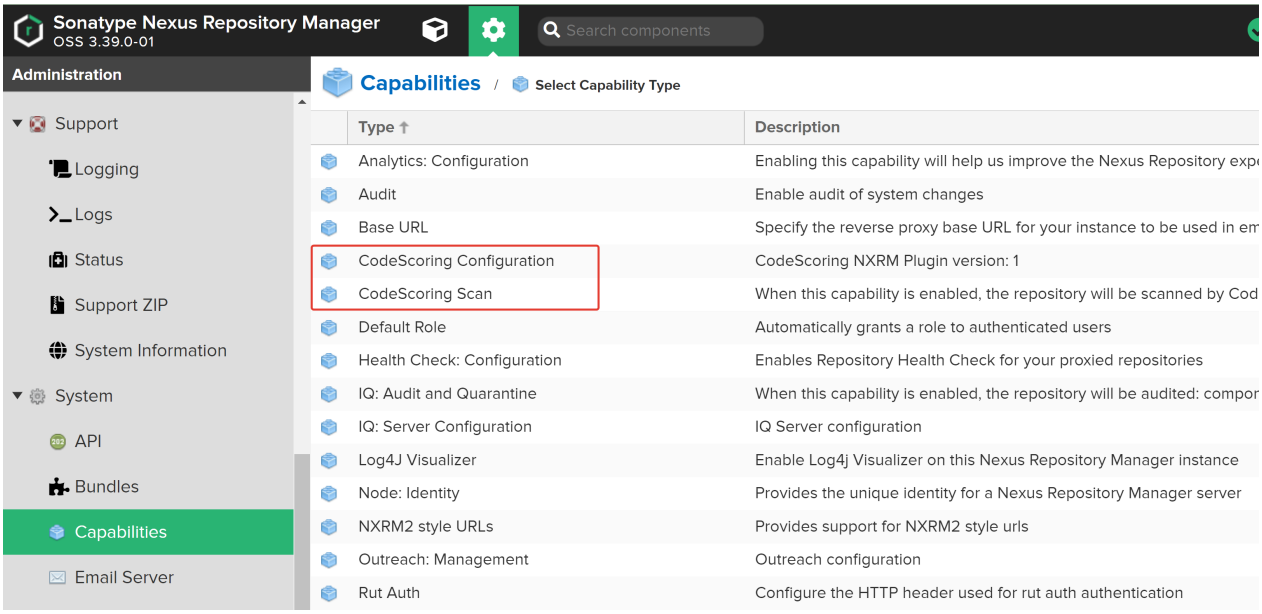
#Настройка плагина

Для применения плагина **CodeScoring.OSA** в дальнейшей работе, необходимо использовать механизм **Capability**, предоставляемый **NXRM**. **Capability** – это набор API и компонентов UI для встраивания в **NXRM**, позволяющий расширять его функциональность.

Плагин **CodeScoring.OSA** предоставляет пять новых **Capability**:

- **CodeScoring Configuration** — настройка взаимодействия с **on-premise** платформой **CodeScoring**;
- **CodeScoring Scan** — настройка сканирования для отдельно выбранного прокси-репозитория;
- **CodeScoring Docker Repository Scan** – настройка сканирования для отдельно выбранного hosted, proxy docker или virtual репозитория;
- **CodeScoring Repository Mask Scan** – настройка сканирования репозиториев, названия которых соответствуют regex-маске;
- **CodeScoring All Repositories Scan** – настройка сканирования для всех репозиториев.

После установки плагина **CodeScoring OSA** в разделе `System -> Capabilities` появится возможность создания **Capability** через элемент (`+ Create capability`) интерфейса.



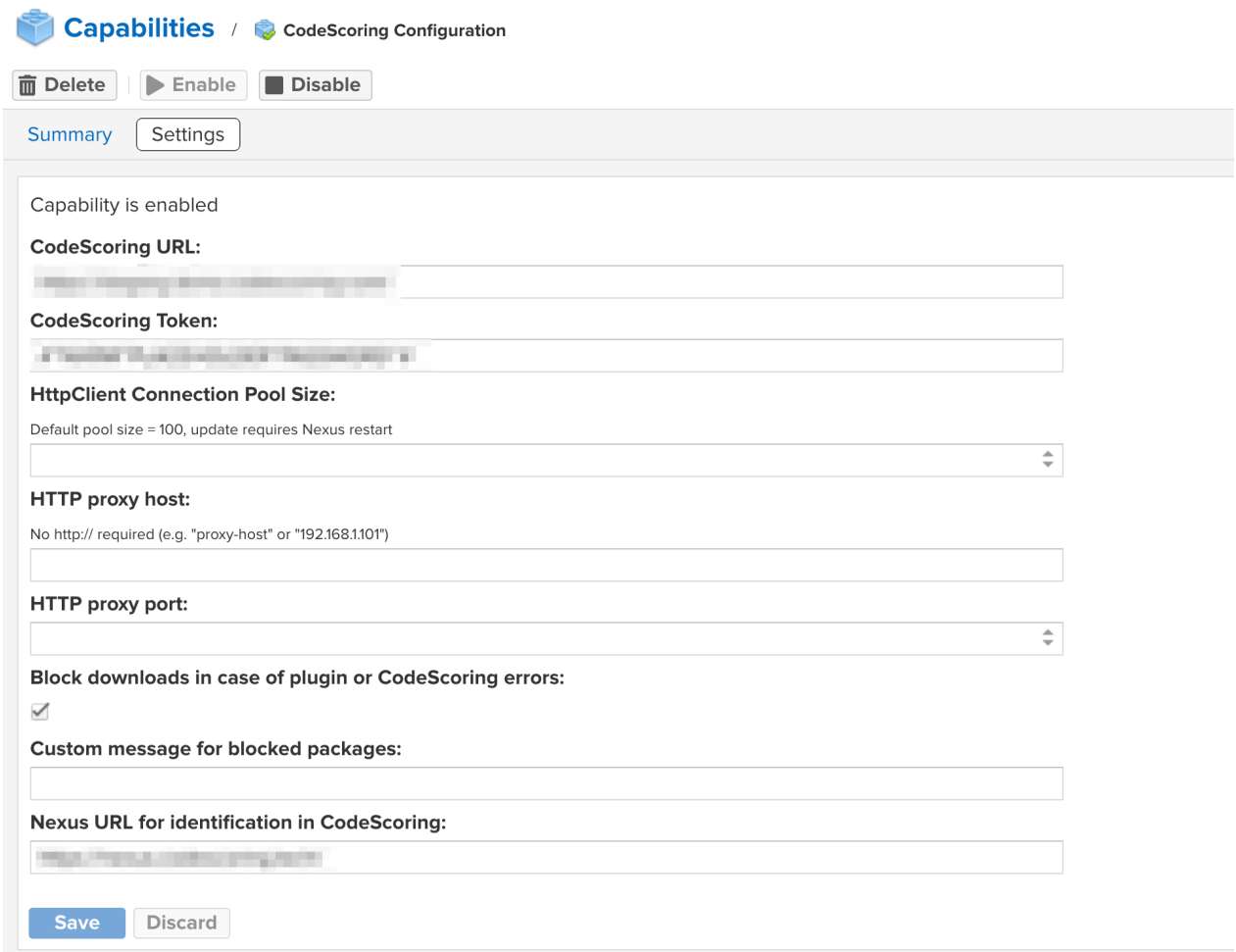
#CodeScoring Configuration

Расширение позволяет задать общие настройки плагина для работы с платформой **CodeScoring**:

- **CodeScoring URL** – адрес **on-premise** платформе **CodeScoring**;
- **CodeScoring Token** – ключ для авторизации вызовов API (Создается из раздела Профиль);
- **HttpClient Connection Pool Size** – количество доступных соединений. Параметр позволяет управлять количеством параллельных запросов, чтобы ускорить сканирование;
- **Timeout for CodeScoring requests in seconds** – время ожидания ответа от платформы (в секундах, по умолчанию значение **1800**);
- **HTTP Proxy Host** – адрес прокси-сервера. Используется в случае, если нет возможности наладить прямое соединение между NXRM и CodeScoring;
- **HTTP Proxy Port** – порт прокси-сервера;
- **Block downloads in case of plugin or CodeScoring errors** – блокировка загрузки компонента при наличии ошибок от плагина или CodeScoring API.
- **Custom message for blocked packages** – сообщение для пользователя при блокировке компонентов;
- **Append block URL to custom message** – добавление ссылки на страницу блокировки в кастомное сообщение;
- **Nexus URL for identification in CodeScoring** – адрес Nexus Repository Manager с протоколом для отображения результатов в платформе.

Обязательные поля

Поля **CodeScoring URL**, **CodeScoring Token**, **HttpClient Connection Pool Size**, **Timeout for CodeScoring requests in seconds** и **Nexus URL for identification in CodeScoring** являются обязательными к заполнению.

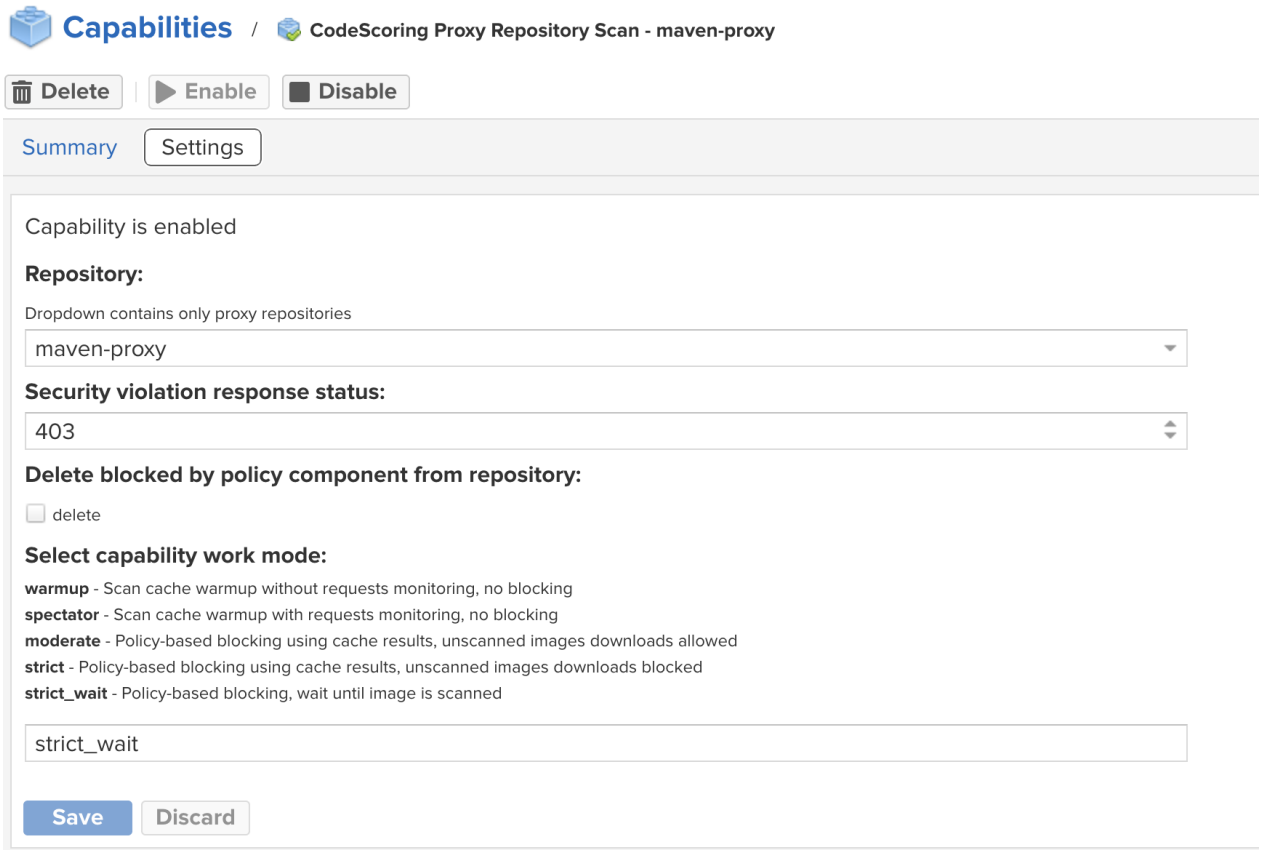


Внимание: указанные настройки будут использовать все экземпляры осуществляющие проверку прокси репозиториев.

#CodeScoring Proxy Repository Scan

Расширение позволяет включить проверку компонентов на выбранный прокси репозиторий со следующими параметрами:

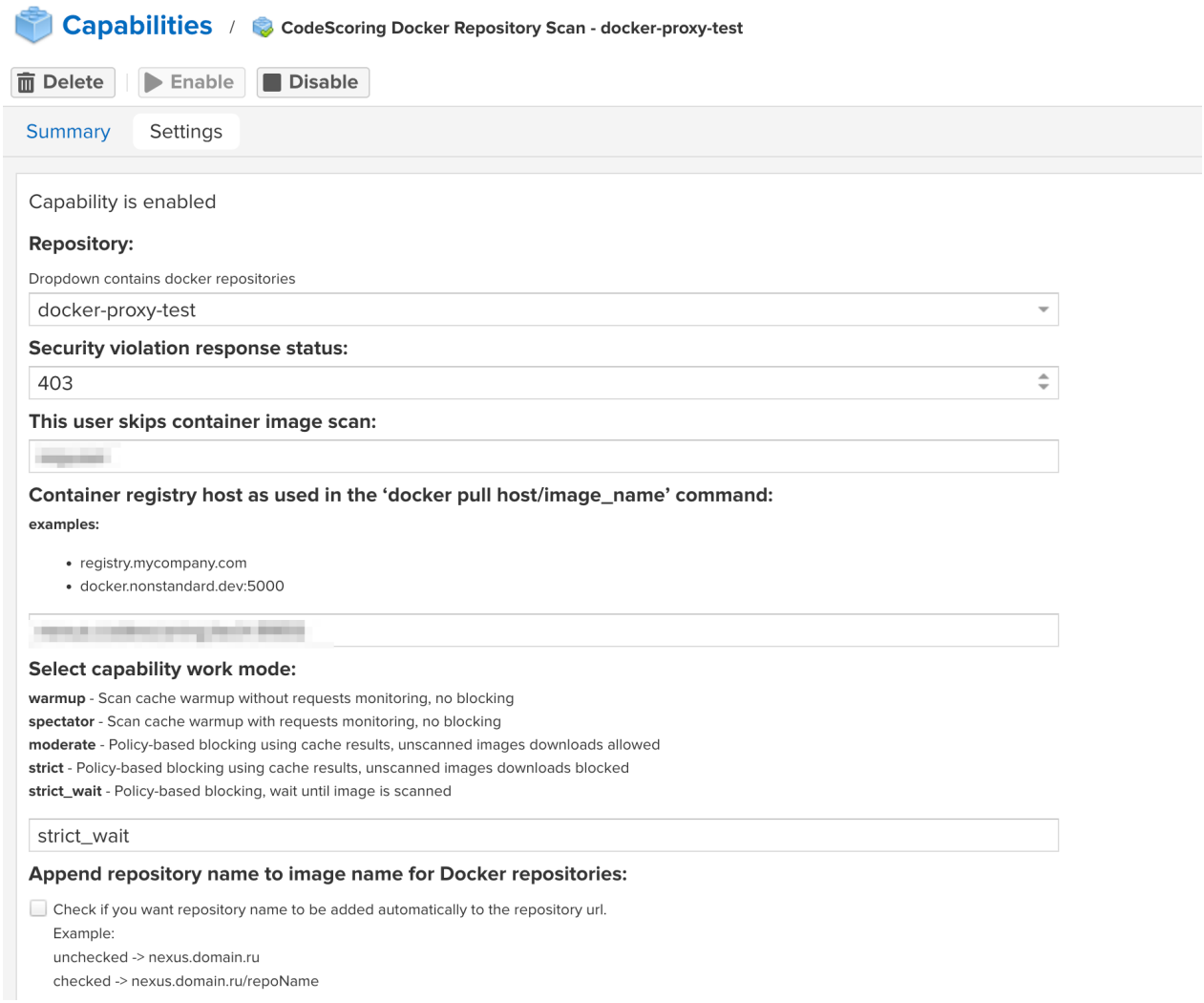
- **Repository** – выбор репозитория, для которого будет применена функция экранирования;
- **Security violation response status** – код ошибки, возвращаемый при срабатывании политик безопасности;
- **Delete blocked by policy component from repository** – принудительное удаление блокируемых компонентов из репозитория (*создание* "*стерильного*" *репозитория*);
- **Select capability work mode** – режим работы плагина.



#CodeScoring Docker Repository Scan

Расширение позволяет включить проверку компонентов на выбранный hosted или proxy docker репозиторий со следующими параметрами:

- **Repository** – выбор репозитория, для которого будет применена функция экранирования;
- **Security violation response status** – код ошибки, возвращаемый при срабатывании политик безопасности;
- **This user skips container image scan** – имя пользователя, для которого не применяется сканирование образов. Используется при загрузке и проверке компонентов консольным агентом;
- **Container registry host as used in the `docker pull host/image_name` command** – адрес (без указания протокола) и порт, через которые будут загружаться образы для сканирования. Используется для связи Nexus с репозиторием через Docker;
- **Select capability work mode** – режим работы плагина. Режимы работы описаны в секции ниже;
- **Append repository name to image name for Docker repositories** – добавление названия репозитория в PURL для корректной работы в режиме **RepoPath** (в случае обращения за компонентом через команду `docker pull registry/repository/image_name`).

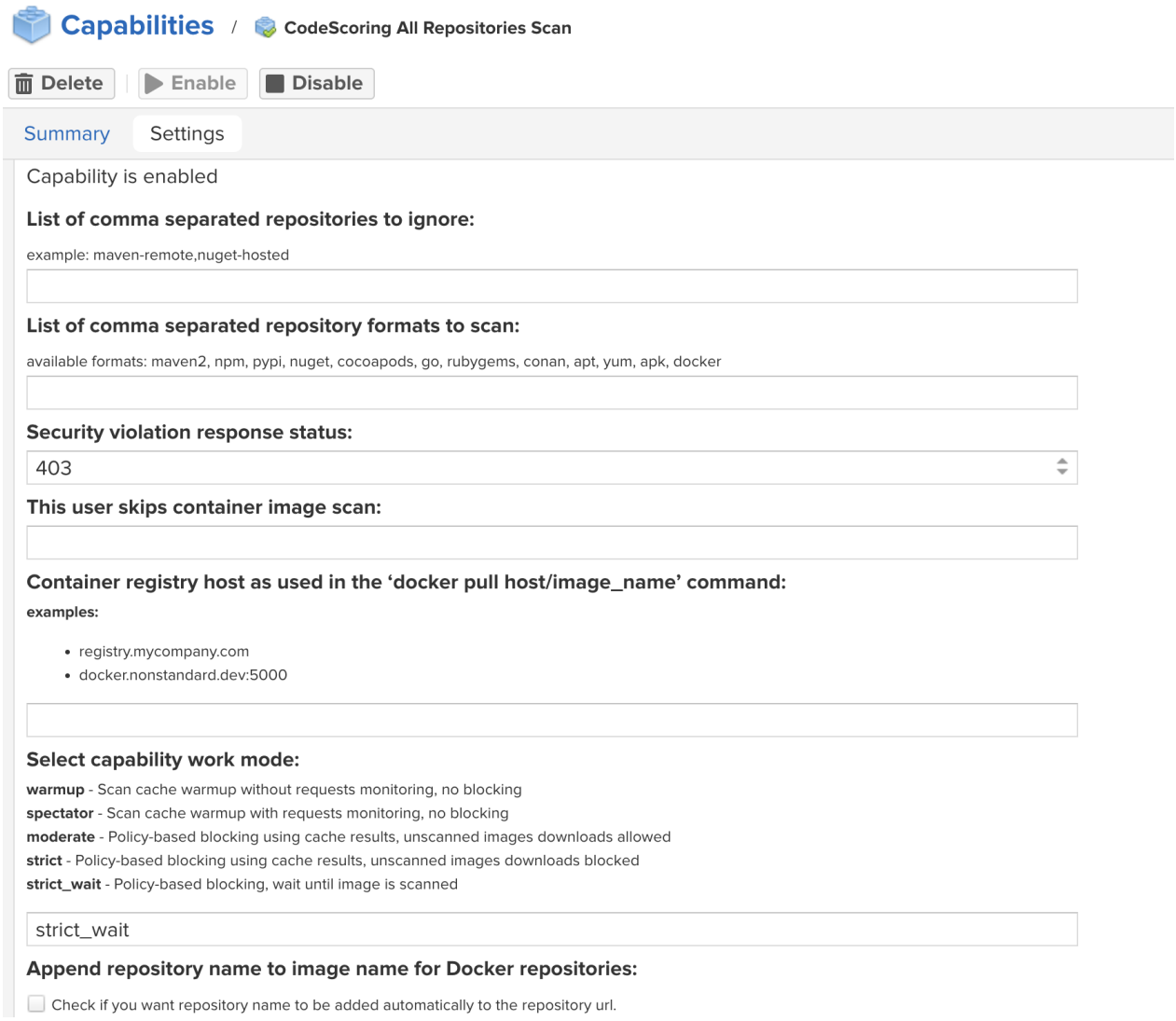


#CodeScoring All Repositories Scan

Расширение позволяет включить проверку компонентов на все репозитории в рамках Sonatype Nexus Repository Manager со следующими параметрами:

- **List of comma separated repositories to ignore** – список репозиториев, которые не будут сканироваться;
- **List of comma separated repository formats to scan** – список форматов репозиториев, которые будут сканироваться. Доступные форматы: `maven2`, `npm`, `pypi`, `nuget`, `cocoapods`, `go`, `rubygems`, `conan`, `apt`, `yum`, `apk`, `docker`, `conda`;
- **Repository exclude masks (regex, comma-separated)** – список regex-масок для репозиториев, которые всегда исключаются из сканирования. Маски имеют приоритет над остальными настройками;
- **Security violation response status** – код ошибки, возвращаемый при срабатывании политик безопасности;
- **This user skips container image scan** – имя пользователя, для которого не применяется сканирование образов. Используется при загрузке и проверке компонентов консольным агентом;

- **Container registry host as used in the `docker pull host/image_name` command** – адрес (без указания протокола) и порт, через которые будут загружаться образы для сканирования. Используется для связи Nexus с репозиторием через Docker;
- **Select capability work mode** – режим работы плагина. Режимы работы описаны в секции ниже;
- **Append repository name to image name for Docker repositories** – добавление названия репозитория в PURL для корректной работы в режиме **RepoPath** (в случае обращения за компонентом через команду `docker pull registry/repository/image_name`).



#CodeScoring Repository Mask Scan

Расширение позволяет включить проверку компонентов для репозиториев, названия которых соответствуют Java regex-паттерну. Для Capability доступны следующие параметры:

- **Repository name pattern (regex)** – Java regex-паттерн для сопоставления названий репозиториев. Например, `npm-.*`, `.*-remote` или `maven-(release|snapshot)-.*`;

- **Security violation response status** – код ошибки, возвращаемый при срабатывании политик безопасности;
- **Delete blocked by policy component from repository** – принудительное удаление блокируемых компонентов из репозитория (*создание* "*стерильного*" *репозитория*);
- **Select capability work mode** – режим работы плагина. Режимы работы описаны в секции ниже.

#Настройка режима работы плагина

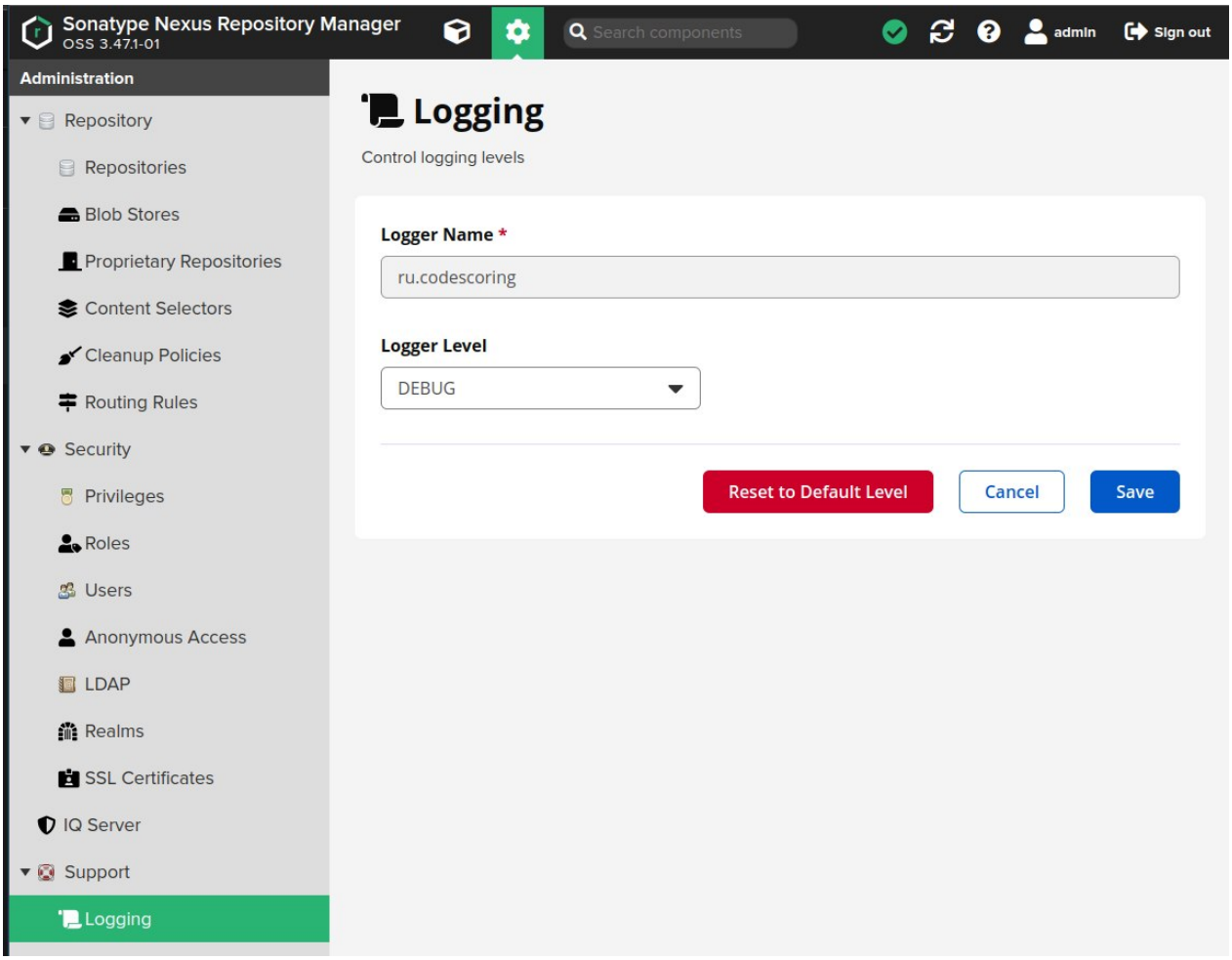
Режим работы плагина необходимо определить текстовой строкой в соответствующем поле настроек Capability **CodeScoring Proxy Repository Scan**, **CodeScoring Docker Repository Scan**, **CodeScoring All Repositories Scan** или **CodeScoring Repository Mask Scan**.

Плагин имеет 5 режимов работы, определяющих строгость проверки компонентов перед загрузкой.

- **warmup** – загрузка данных в кэш CodeScoring без блокировки компонентов;
- **spectator** – загрузка данных в кэш CodeScoring без блокировки компонентов, сохранение результатов запросов компонентов на платформе;
- **moderate** – блокировка компонентов, не прошедших проверку политик. Разрешена загрузка непросканированных компонентов;
- **strict** – блокировка компонентов, не прошедших проверку политик. Запрещена загрузка непросканированных компонентов;
- **strict_wait** – блокировка компонентов, не прошедших проверку политик. Ожидание проверки для непросканированных компонентов.

#Настройка логирования

Для настройки логирования событий плагина необходимо зайти в раздел `Support -> Logging` и добавить логгер с названием **ru.codescoring** и уровнем логирования **DEBUG**.



Результаты логирования событий доступны в разделе `Support -> Logs`.

#Блокировка компонента

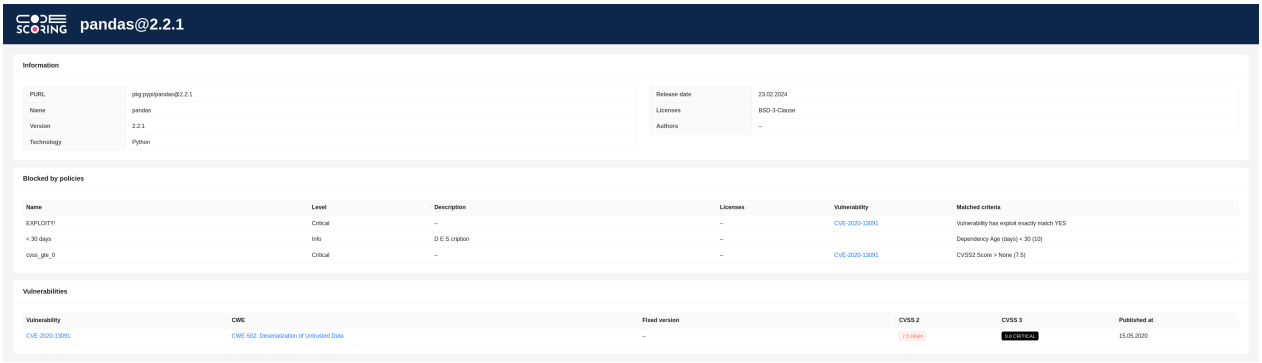
При блокировании загрузки компонента в консоли пользователя отображается одна из следующих причин блокировки:

- **"The download has been blocked in accordance with the policies configured in CodeScoring"** – блокировка компонента согласно настроенным на платформе политикам;
- **"The component has not yet been scanned by CodeScoring, it is scheduled to be scanned shortly. The download is blocked according to the plugin settings"** – блокировка непросканированного компонента с последующим запуском сканирования. Используется в режиме `strict`;
- **"The download has been blocked due to the failure of the scan of the component in CodeScoring"** – не удалось просканировать компонент;
- **"The download has been blocked due to the wrong mode of the plugin"** – используется некорректный режим работы плагина;
- **"The download has been blocked due to the timeout of the scan of the component in CodeScoring"** – истекло время ожидания сканирования компонента. Используется в режиме `strict_wait`;

- **"The download has been blocked, because registry is not configured in CodeScoring"** – отсутствует соответствующий Registry на платформе.

При использовании политики с отложенной блокировкой компонент получает статус `delayed_block`. Загрузка компонента в данном статусе не прерывается.

Ответ также содержит ссылку на страницу компонента в CodeScoring с информацией о сработавших политиках безопасности и найденных уязвимостях. При использовании кастомного сообщения добавление этой ссылки управляется настройкой **Append block URL to custom message**:



#Настройка SSL-соединения

Для настройки SSL-соединения между плагином и платформой необходимо выполнить импорт сертификатов в Java Truststore.

#Определение местоположения установки Java

Чтобы импортировать сертификаты в Java Truststore, сначала необходимо найти установку Java. Это можно сделать одним из следующих способов:

1. **Использование переменной окружения `JAVA_HOME`**:

   ```
   echo $JAVA_HOME
   ```

2. **Использование команды `java` с параметром `-XshowSettings:properties`**:

   ```
   java -XshowSettings:properties 2>&1 > /dev/null | grep
   'java.home'
   ```

3. **Использование команды `readlink` с путем к `java`**:

   ```
   readlink -f $(which java)
   ```

Дальнейшие действия подразумевают, что переменная $JAVA_HOME установлена.

#Загрузка сертификата

Скачать сертификат можно следующей командой:

```
openssl s_client -connect <codescoring.domain.ru>:443 2>/dev/
null | openssl x509 > codescoring_ca.pem
```

Убедитесь, что вы заменили <codescoring.domain.ru> на соответствующий адрес вашей платформе.

#Импорт сертификата

После загрузки сертификата его можно импортировать в Java Truststore с помощью следующей команды:

```
keytool -import -alias <mycert> -keystore $JAVA_HOME/lib/
security/cacerts -file <codescoring_ca.pem>
```

Примечания:

- Замените <mycert> на уникальное имя для вашего сертификата.
- Замените <codescoring_ca.pem> на фактическое имя вашего файла сертификата.
- Вас могут попросить ввести пароль для Truststore. Стандартный пароль: changeit.

#Проверка импорта

Чтобы убедиться, что сертификат был успешно импортирован, используйте команду keytool для отображения списка сертификатов в Truststore:

```
keytool -list -keystore $JAVA_HOME/lib/security/cacerts
```

Примечание:

- Для фильтрации результатов по вашему алиасу сертификата можно использовать команду grep:

  ```
  keytool -list -keystore $JAVA_HOME/lib/security/cacerts |
  grep mycert
  ```

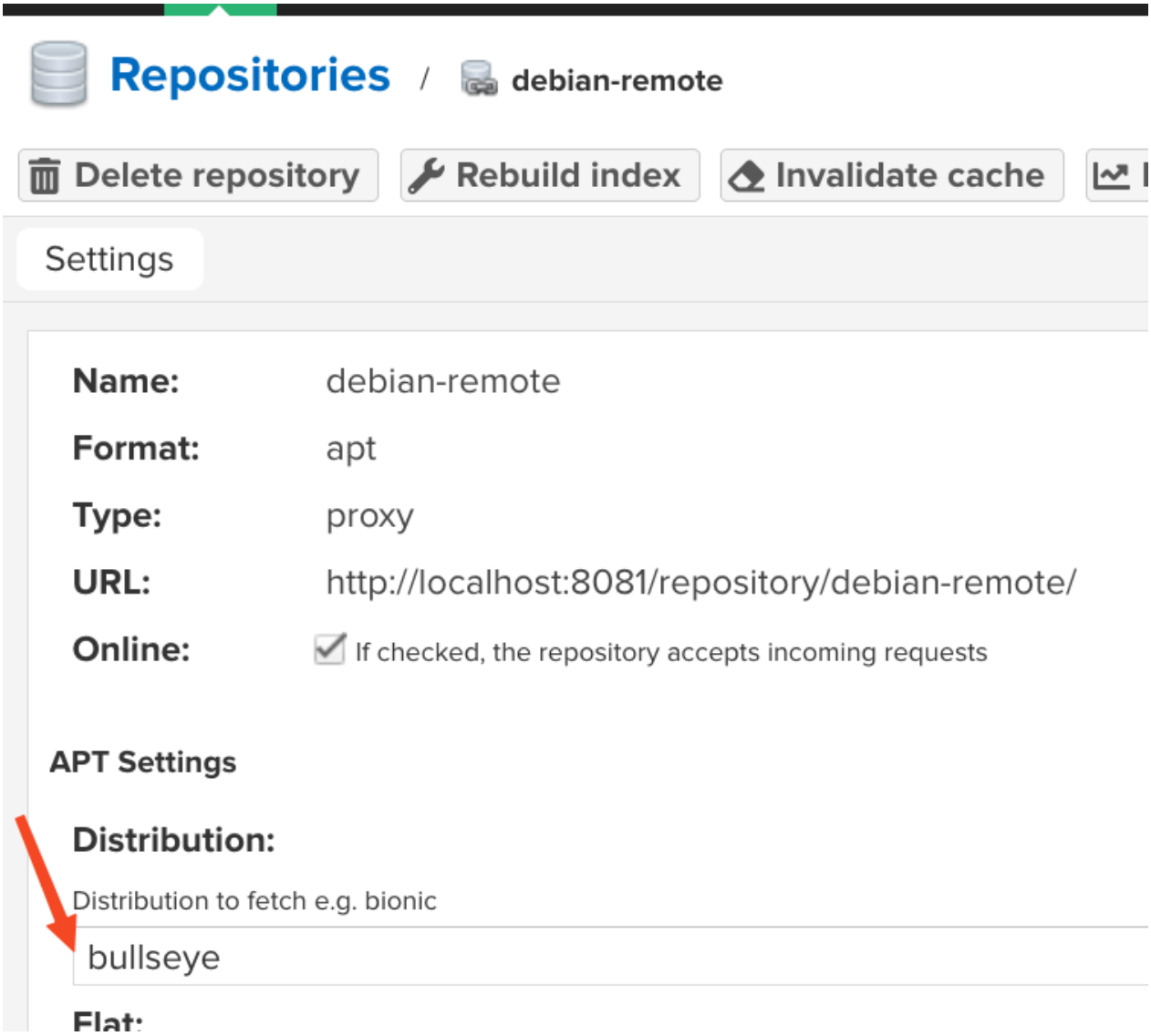
#Работа с системными пакетами

#Настройка репозиториев

Для корректной работы плагина с системными пакетами некоторых экосистем необходимо произвести дополнительные действия.

Для корректной работы плагина необходимо указать название (codename) дистрибутива из удалённого репозитория, например "bullseye" для Debian. Это название используется в PURL (Package URL) для повышения точности анализа пакета.

Название должно быть в нижнем регистре и без лишних символов.



Список поддерживаемых дистрибутивов Debian:

- **Debian 2.0** – *hamm*
- **Debian 2.1** – *slink*
- **Debian 2.2** – *potato*
- **Debian 3.0** – *woody*
- **Debian 3.1** – *sarge*

- **Debian 4** – *etch*
- **Debian 5** – *lenny*
- **Debian 6** – *squeeze*
- **Debian 7** – *wheezy*
- **Debian 8** – *jessie*
- **Debian 9** – *stretch*
- **Debian 10** – *buster*
- **Debian 11** – *bullseye*
- **Debian 12** – *bookworm*
- **Debian 13** – *trixie*
- **Debian 14** – *forky*

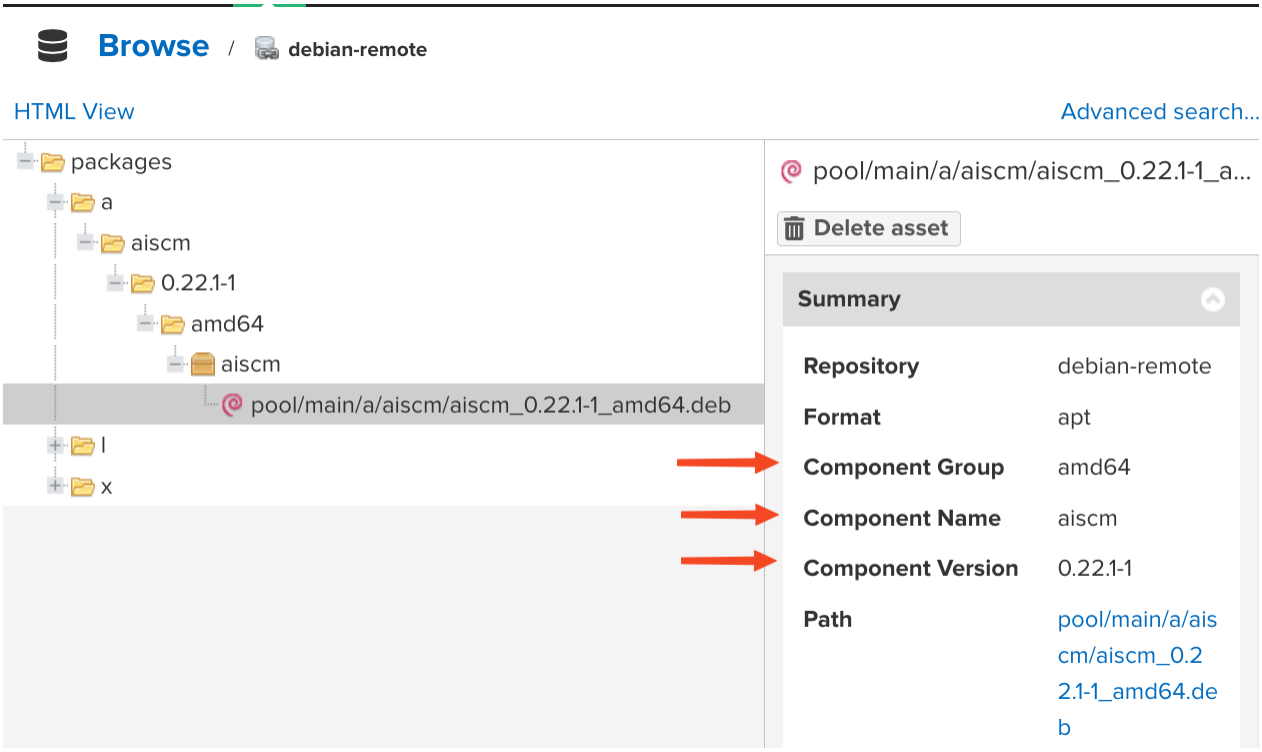
Список поддерживаемых дистрибутивов Ubuntu:

- **Ubuntu 4.10** – *warty*
- **Ubuntu 5.04** – *hoary*
- **Ubuntu 5.10** – *breezy*
- **Ubuntu 6.06** – *dapper*
- **Ubuntu 6.10** – *edgy*
- **Ubuntu 7.04** – *feisty*
- **Ubuntu 7.10** – *gutsy*
- **Ubuntu 8.04** – *hardy*
- **Ubuntu 8.10** – *intrepid*
- **Ubuntu 9.04** – *jaunty*
- **Ubuntu 9.10** – *karmic*
- **Ubuntu 10.04** – *lucid*
- **Ubuntu 10.10** – *maverick*
- **Ubuntu 11.04** – *natty*
- **Ubuntu 11.10** – *oneiric*
- **Ubuntu 12.04** – *precise*
- **Ubuntu 12.10** – *quantal*
- **Ubuntu 13.04** – *raring*
- **Ubuntu 13.10** – *saucy*
- **Ubuntu 14.04** – *trusty*
- **Ubuntu 14.10** – *utopic*
- **Ubuntu 15.04** – *vivid*
- **Ubuntu 15.10** – *wily*
- **Ubuntu 16.04** – *xenial*
- **Ubuntu 16.10** – *yakkety*
- **Ubuntu 17.04** – *zesty*
- **Ubuntu 17.10** – *artful*
- **Ubuntu 18.04** – *bionic*
- **Ubuntu 18.10** – *cosmic*
- **Ubuntu 19.04** – *disco*
- **Ubuntu 19.10** – *eoan*
- **Ubuntu 20.04** – *focal*
- **Ubuntu 20.10** – *groovy*
- **Ubuntu 21.04** – *hirsute*
- **Ubuntu 21.10** – *impish*
- **Ubuntu 22.04** – *jammy*
- **Ubuntu 22.10** – *kinetic*
- **Ubuntu 23.04** – *lunar*

- **Ubuntu 23.10** – *mantic*
- **Ubuntu 24.04** – *noble*
- **Ubuntu 24.10** – *oracular*
- **Ubuntu 25.04** – *plucky*

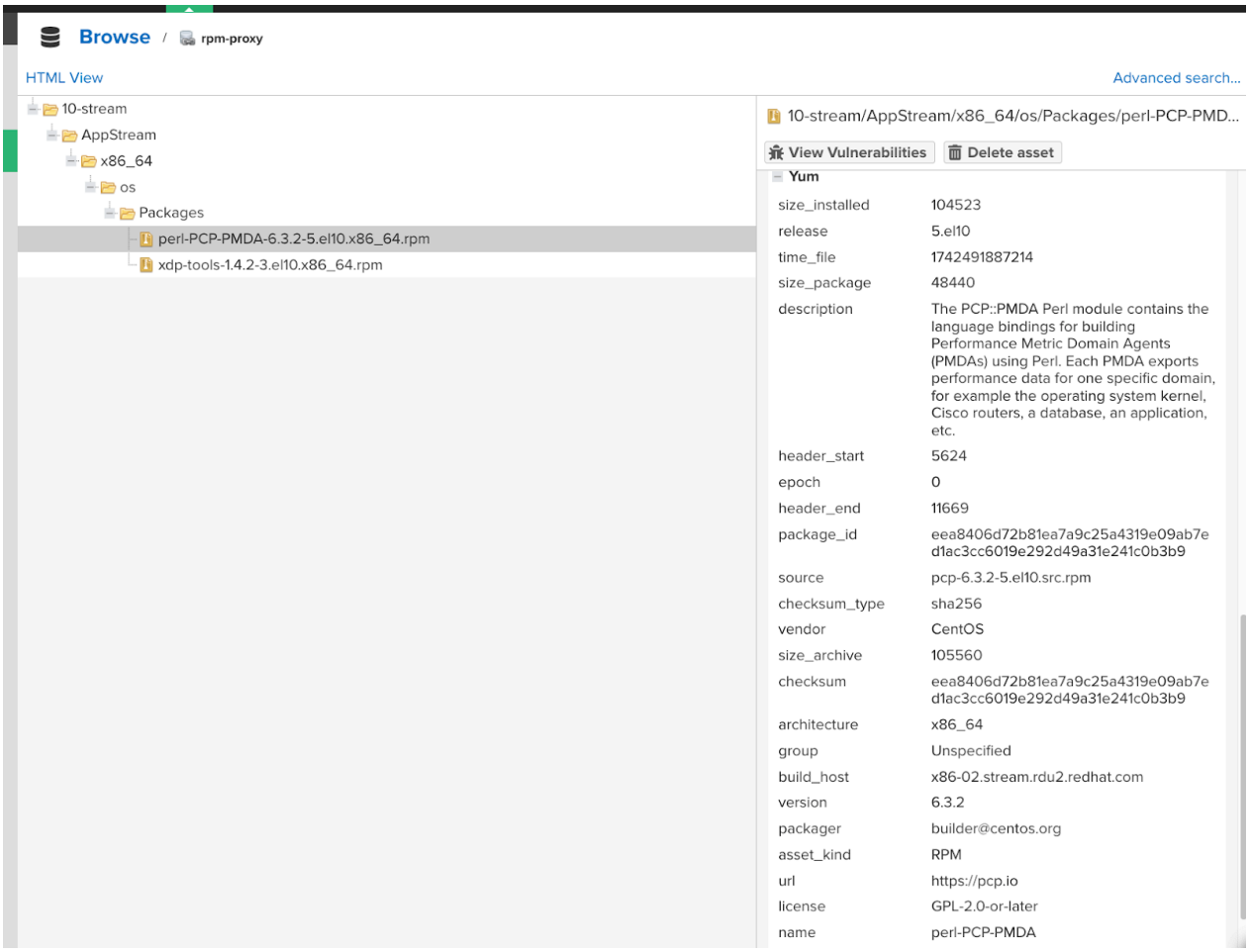
#Просмотр информации о пакете Debian

Плагин извлекает информацию о пакете из различных источников. В частности, он получает название пакета, версию и архитектуру из поля Summary asset'a. Если Summary отсутствует, данные для PURL парсятся из Path.



#Просмотр информации о пакете RPM

Для пакетов RPM плагин извлекает данные из атрибутов asset'a. Это включает название пакета, версию и архитектуру.



#Плагин для JFrog

Поддерживаемые типы репозиториев: Alpine, Cargo, CocoaPods, Composer, Conan, Conda, Debian, Docker, Go, Maven, NPM, NuGet, PyPI, RPM, RubyGems, Swift.

#Установка плагина

Плагин **CodeScoring.OSA** поддерживает версии JFrog Artifactory Pro **7.43** и выше.

Плагин поставляется в виде архива со следующей структурой:

CHANGELOG.md

codescoring.groovy

codescoring.yaml

›

lib

codescoring-plugin-jfrog.jar

Для добавления плагина в **JFrog** необходимо:

1. Распаковать полученный архив в директорию `$JFROG_HOME/artifactory/var/etc/artifactory/plugins`.
2. Создать в директории файл для настройки `codescoring.yaml`. Пример содержания находится в поставляемом архиве.
3. Вызвать **API JFrog Pro** для загрузки плагина `POST /api/plugins/reload`:

```
curl -X POST https://[JFROG_URL]/artifactory/api/plugins/reload
```

#Проверка установки плагина

Для проверки установки плагина в системе необходимо проверить логи сервиса. При успешной загрузке и инициализации в логах появится сообщение следующего содержания:

```
2023-08-08T09:41:35.105Z [jfrt ] [INFO ] [70be801ff583b741]
[r.c.p.codescoring:16          ] [art-init            ] -
CodeScoring: Initialization of CodeScoringPlugin completed
```

#Обновление плагина

В случае обновления архива с плагином, для вступления обновлений в силу необходимо использовать следующую команду API:

```
curl -X POST https://[JFROG_URL]/artifactory/api/plugins/reload
```

В случае обновления конфигурации плагина в файле `codescoring.yaml`, необходимо использовать следующую команду API:

```
curl -X POST https://[JFROG_URL]/api/plugins/execute/
codeScoringReload
```

#Настройка плагина

Для настройки плагина используется файл `codescoring.yaml`.

Пример содержания файла:

```
## true/false
disablePlugin: false

codeScoringAPI:
  # The base URL for all CodeScoring API endpoints.
  # Required.
  # Example: https://host:port or https://host
```

```
  url:

  # Your CodeScoring API Token for authentication.
  # Required.
  token:

  # Http client connection pool size to CodeScoring BE service.
  # By default, value is 200 since it correlates with the
default artifactory thread pool size for tomcat.
  # If you tuned your instance of the artifactory https://
jfrog.com/help/r/how-do-i-tune-artifactory-for-heavy-loads
  # you should scale this value for better performance maximum
up to tomcat.connector.maxThreads value.
  connectionPoolSize: 200

  # By default, if CodeScoring API hasn't responded within a
duration of 60 seconds, the request will be cancelled.
  # This property lets you customize the timeout duration in
seconds.
  timeout: 60

  # If you are using a proxy, you must provide both Hostname/IP
and port.
  proxy:
    host:
    port:

## Artifactory's response status code for blocked packages.
blockedBuildResponseCode: 403

## If set to 'false' allows artifact downloads regardless of
errors from CodeScoringAPI or plugin
blockOnErrors: true

## If set to 'true', the plugin will scan all supported
repositories
## except specified in the "excludeRepositories" section.
scanAllRepositories: false

## Store scan date and blocking reasons in the artifact
properties.
storeScanProperties: false

## Default settings for all repositories. Can be overridden by
repositories.repo-name settings
defaults:
  dockerRegistryUrl: jfrog.my.domain

  # warmup |  Scan cache warmup without requests monitoring, no
blocking
  # spectator | Scan cache warmup with requests monitoring, no
blocking
```

```
  # moderate | Policy-based blocking using cache results, not
scanned component downloads allowed
  # strict | Policy-based blocking using cache results, not
scanned component downloads blocked
  # strict_wait | Policy-based blocking, wait until component
is scanned
  # default value is strict_wait if not specified in default or
repository settings or in case of a typo
  workMode: strict_wait

  # Allows this user to skip scan
  skipScanUser: codescoring

  # Set to 'true' if you use Docker Access Method 'Sub domain'
(repo-name.jfrog.my.domain) or 'Port' (jfrog.my.domain:25000)
  stripRepoNameInDockerImageName: false

  # Artifactory url for CodeScoring to apply policies.
  # Value MUST BE equal to Repository Manager URL in
CodeScoring
  # Example: https://jfrog.my.domain
  repositoryManagerUrl:

  # Delete artifact from the repository if it is blocked by the
policies
  deleteBlocked: false

## Settings per repository
## Example:
## repositories:
##   docker-remote:
##   docker-local:
##     dockerRegistryUrl: another-jfrog.my.domain
##     skipScanUser: codescoring
##     workMode: spectator
##   pypi-remote:
##     workMode: warmup
repositories:

## Pattern-based repository matching using regex. Settings are
applied to
## repositories whose names match the given pattern.
## Same settings as in 'repositories' section, plus a 'pattern'
field.
## Lookup order: exact match in 'repositories' → first matching
mask → 'defaults'
## When scanAllRepositories=false, matching masks also act as
inclusion criteria.
## Example:
## repositoryMasks:
##   - pattern: "npm-.*"
##     workMode: moderate
```

```
##     skipScanUser: codescoring
##   - pattern: "pypi-.*-remote"
##     workMode: warmup
##   - pattern: ".*-staging"
##     workMode: spectator
##     deleteBlocked: true
repositoryMasks:

## Regex patterns for excluding repositories from scanning.
## Exclude masks always take priority over all other matching
(scanAllRepositories, exact, masks).
## Example:
## excludeRepositoryMasks:
##   - pattern: ".*-snapshot"
##   - pattern: "temp-.*"
excludeRepositoryMasks:

## List of the excluded repositories (exact names). Used, if
scanAllRepositories=true
## Example:
## excludeRepositories:
##   - npm-remote
##   - maven-local
excludeRepositories:

## List of repository types to scan. Used, if
scanAllRepositories=true
## Supported values are: maven, npm, pypi, nuget, cocoapods,
go, gems, debian, yum, alpine, docker, composer, cargo, conda,
conan, swift
## Example:
## repositoryTypes:
##   - npm
##   - go
repositoryTypes:
```

#Описание параметров

- **disablePlugin** – отключение плагина;
- **codeScoringAPI** - настройки параметров взаимодействия плагина с платформой CodeScoring;
  - **url** – адрес платформе CodeScoring (обязательно указание протокола);
  - **token** – ключ для авторизации вызовов API (*Создается из CodeScoring раздела `Profile -> Home`*);
  - **connectionPoolSize** – размер пула соединений с платформой CodeScoring;
  - **timeout** - время ожидания ответа (в секундах). По умолчанию, если CodeScoring API не отвечает в течение 60 секунд, запрос будет отменен;

        - **proxy** - настройки прокси-сервера;
            - **host** - хост/IP;
            - **port** - порт;
- **blockedBuildResponseCode** – код ошибки, возвращаемый при срабатывании политик безопасности;
- **blockOnErrors** - блокирование загрузки компонентов в случае ошибки при взаимодействии с платформой CodeScoring;
- **scanAllRepositories** - подключение всех поддерживаемых репозиториев за исключением указанных в параметре **excludeRepositories**;
- **storeScanProperties** - сохранение причины блокировки и отметки о времени сканирования в свойства артефакта;
- **defaults** – настройки сканирования по умолчанию для всех подключенных репозиториев;
    - **dockerRegistryUrl** – адрес docker registry;
    - **workMode** – режим работы плагина. Условия каждого режима работы описаны в секции ниже;
    - **skipScanUser** – пользователь Артифактори, для которого пропускается сканирование компонентов. Необходимо для того, чтобы CodeScoring мог самостоятельно забрать компонент для сканирования. Пользователя необходимо указывать аналогичного тому, что был указан в интеграции Менеджеры репозиториев инсталляции;
    - **stripRepoNameInDockerImageName** – убирать название репозитория из имени образа. Используется в подходе Repository Path при работе с docker registry. По умолчанию название репозитория добавляется к имени образа;
    - **repositoryManagerUrl** - URL Artifactory. Тот же URL должен быть указан в CodeScoring для применения политик по репозиториям.
    - **deleteBlocked** - удалять заблокированный политиками артефакт;
- **repositories** – список репозиториев, для которых работает сканирование компонентов. Для каждого репозитория можно отдельно указать параметры, как в параметре **defaults**;
- **repositoryMasks** – список масок для сопоставления репозиториев по регулярным выражениям. Для каждой маски доступны те же параметры, что и в **defaults**, плюс поле **pattern**. Подробнее в разделе «Настройка масок репозиториев»;
- **excludeRepositoryMasks** – список масок для исключения репозиториев из сканирования по регулярным выражениям. Имеют наивысший приоритет;
- **excludeRepositories** - список точных названий репозиториев, исключенных из обработки плагином.

Важно: для generic и VCS репозиториев обязательно указать один из следующих типов репозитория в поле Internal Description:

- maven
- npm
- pypi
- nuget

- cocoapods
- go
- gems
- debian
- yum
- alpine
- docker
- composer
- cargo
- conda
- conan
- swift

#Настройка масок репозиториев

Маски позволяют задавать параметры сканирования для групп репозиториев без перечисления каждого из них явно, используя регулярные выражения.

#repositoryMasks

Параметр `repositoryMasks` задаёт список масок с regex-паттернами. Для каждой маски доступны те же настройки, что и в секции `defaults` (режим работы, пользователь-исключение и т.д.), плюс обязательное поле `pattern`.

Порядок поиска настроек для репозитория:

1. Точное совпадение в `repositories`
2. Первая подходящая маска в `repositoryMasks`
3. Настройки по умолчанию из `defaults`

При `scanAllRepositories: false` совпадение с маской также является критерием включения репозитория в сканирование.

Пример:

```
repositoryMasks:
  - pattern: "npm-.*"
    workMode: moderate
    skipScanUser: codescoring
  - pattern: "pypi-.*-remote"
    workMode: warmup
  - pattern: ".*-staging"
    workMode: spectator
    deleteBlocked: true
```

#excludeRepositoryMasks

Параметр `excludeRepositoryMasks` задаёт список regex-паттернов для исключения репозиториев из сканирования. Маски исключения имеют

наивысший приоритет и применяются до проверки repositories, repositoryMasks и scanAllRepositories.

Пример:

```
excludeRepositoryMasks:
  - pattern: ".*-snapshot"
  - pattern: "temp-.*"
```

#Настройка режимов работы

Режим работы плагина определяется переменной **workMode** в файле codescoring.yaml.

Плагин имеет 6 режимов работы, определяющих строгость проверки компонентов перед загрузкой.

- **off** – сканирование компонентов отключено;
- **warmup** – загрузка данных в кэш CodeScoring без блокировки компонентов;
- **spectator** – загрузка данных в кэш CodeScoring без блокировки компонентов, сохранение результатов запросов компонентов на платформе;
- **moderate** – блокировка компонентов, не прошедших проверку политик. Разрешена загрузка непросканированных компонентов;
- **strict** – блокировка компонентов, не прошедших проверку политик. Запрещена загрузка непросканированных компонентов;
- **strict_wait** – блокировка компонентов, не прошедших проверку политик. Ожидание проверки для непросканированных компонентов.

Важно: выбранный режим работы будет влиять на **все** репозитории, указанные в переменной repositories.

#Настройка логирования

Файл с настройками логирования находится по пути $JFROG_HOME/artifactory/var/etc/artifactory/logback.xml.

Для настроек логирования событий плагина необходимо добавить в файл logback.xml следующее содержание:

```
<appender name="FILE_CODESCORING"
class="ch.qos.logback.core.rolling.RollingFileAppender">
    <File>${log.dir}/codescoring.log</File>
    <rollingPolicy
class="org.jfrog.common.logging.logback.rolling.FixedWindowWit
hDateRollingPolicy">
        <FileNamePattern>${log.dir.archived}/codescoring.
%i.log.gz</FileNamePattern>
    </rollingPolicy>
```

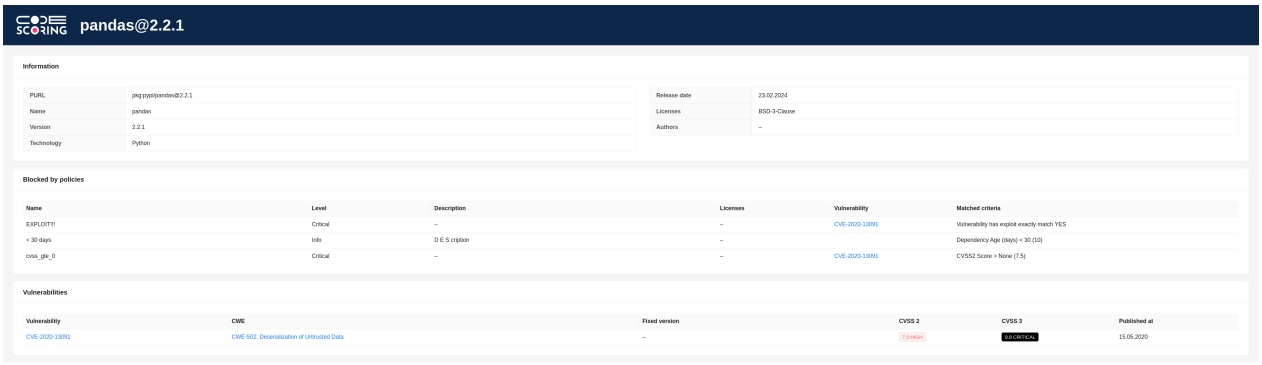
```
    <triggeringPolicy
class="org.jfrog.common.logging.logback.triggering.SizeAndInte
rvalTriggeringPolicy">
        <MaxFileSize>25MB</MaxFileSize>
    </triggeringPolicy>
    <encoder>
        <charset>UTF-8</charset>
        <pattern>%date{yyyy-MM-dd'T'HH:mm:ss.SSS, UTC+3}Z
[%-5p] [%-16X{uber-trace-id}] [%-30.30(%c{3}:%L)]
[%-20.20thread] - %m%n</pattern>
    </encoder>
</appender>
<logger name="ru.codescoring" additivity="false">
    <level value="debug" />
    <appender-ref ref="FILE_CODESCORING" />
</logger>
```

#Блокировка компонента

При блокировании загрузки компонента в консоли пользователя отображается одна из следующих причин блокировки:

- **"The download has been blocked in accordance with the policies configured in CodeScoring"** – блокировка компонента согласно настроенным на платформе политикам;
- **"The component has not yet been scanned by CodeScoring, it is scheduled to be scanned shortly. The download is blocked according to the plugin settings"** – блокировка непросканированного компонента с последующим запуском сканирования. Используется в режиме `strict`;
- **"The download has been blocked due to the failure of the scan of the component in CodeScoring"** – не удалось просканировать компонент;
- **"The download has been blocked due to the wrong mode of the plugin"** – используется некорректный режим работы плагина;
- **"The download has been blocked due to the timeout of the scan of the component in CodeScoring"** – истекло время ожидания сканирования компонента. Используется в режиме `strict_wait`;
- **"The download has been blocked, because registry is not configured in CodeScoring"** – отсутствует соответствующий Registry в платформе.

Ответ также содержит ссылку на страницу компонента в CodeScoring с информацией о сработавших политиках безопасности и найденных уязвимостях:



Важно: если компонент не содержит версию, то он не отправляется на анализ в CodeScoring и, соответственно, не блокируется плагином.

#Работа с системными пакетами

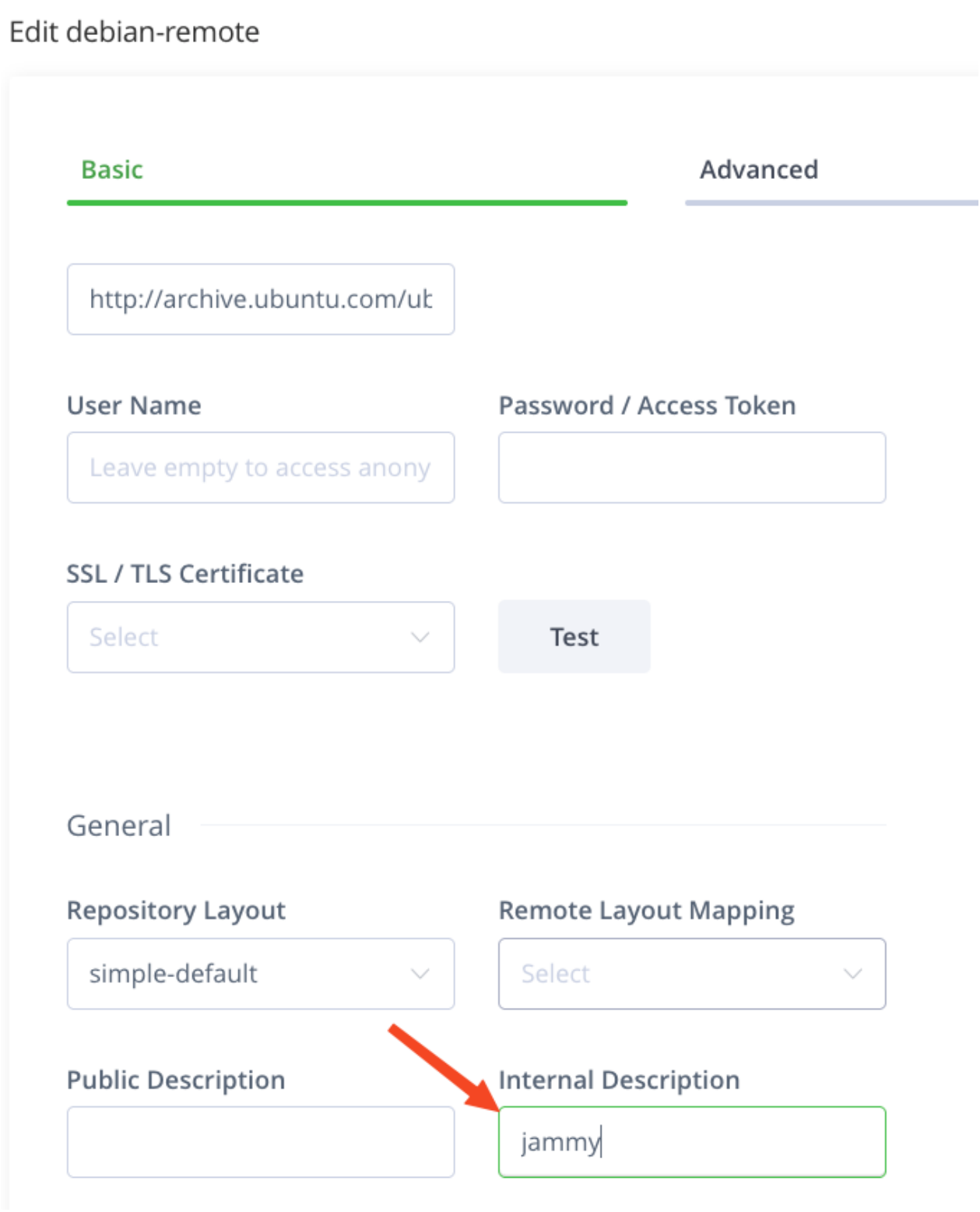
#Настройка репозиториев

Для корректной работы плагина с системными пакетами некоторых экосистем необходимо произвести дополнительные действия.

#Настройка репозитория Debian

Для корректного анализа пакетов необходимо указать название (codename) дистрибутива из удалённого репозитория, например "bullseye" для Debian. Это название вписывается в поле **Internal Description**. Оно используется в PURL (Package URL) для повышения точности анализа пакета.

Название должно быть в нижнем регистре и без лишних символов.



Список поддерживаемых дистрибутивов Debian:

- **Debian 2.0** – *hamm*
- **Debian 2.1** – *slink*
- **Debian 2.2** – *potato*
- **Debian 3.0** – *woody*
- **Debian 3.1** – *sarge*
- **Debian 4** – *etch*
- **Debian 5** – *lenny*

- **Debian 6** – *squeeze*
- **Debian 7** – *wheezy*
- **Debian 8** – *jessie*
- **Debian 9** – *stretch*
- **Debian 10** – *buster*
- **Debian 11** – *bullseye*
- **Debian 12** – *bookworm*
- **Debian 13** – *trixie*
- **Debian 14** – *forky*

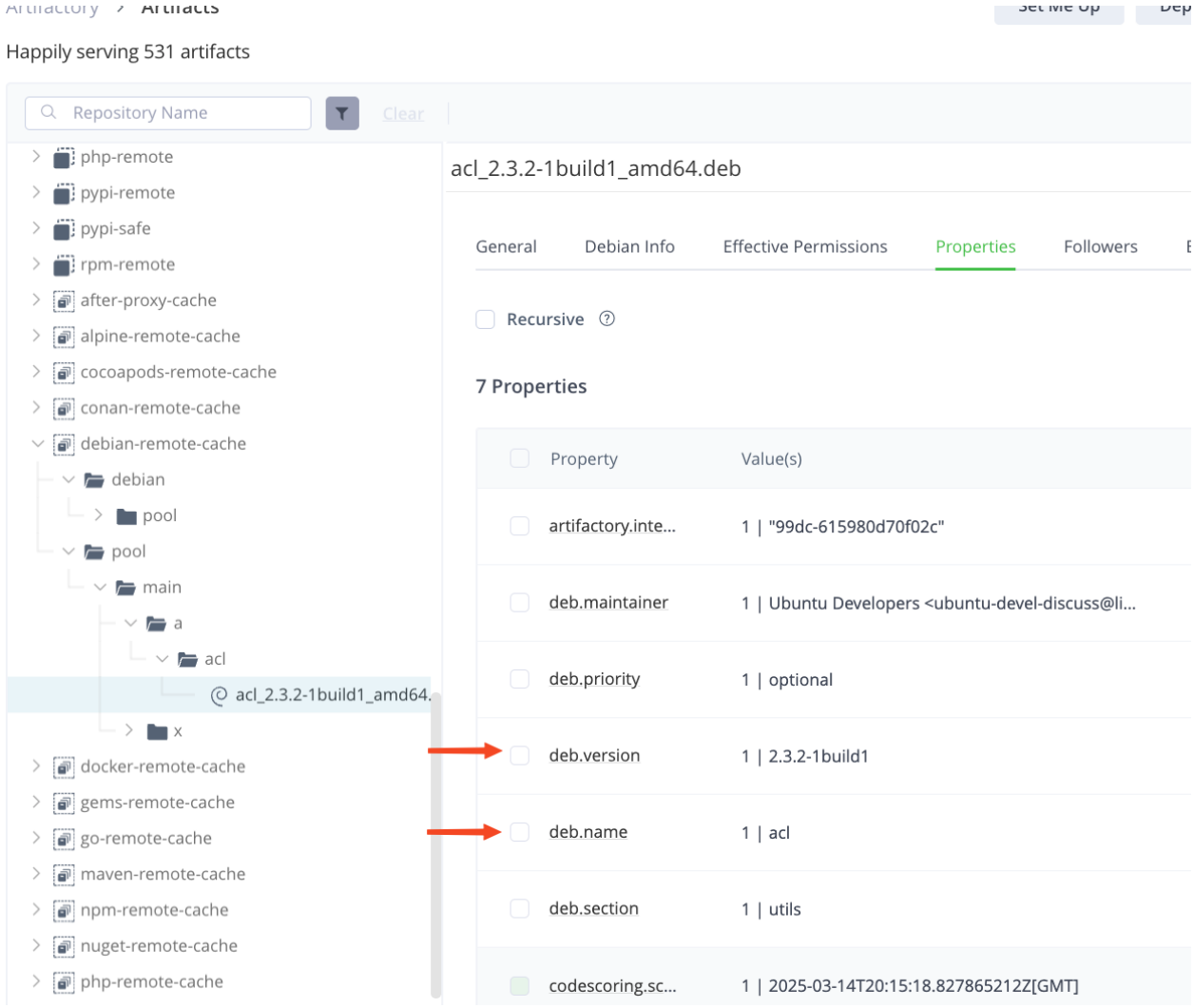
Список поддерживаемых дистрибутивов Ubuntu:

- **Ubuntu 4.10** – *warty*
- **Ubuntu 5.04** – *hoary*
- **Ubuntu 5.10** – *breezy*
- **Ubuntu 6.06** – *dapper*
- **Ubuntu 6.10** – *edgy*
- **Ubuntu 7.04** – *feisty*
- **Ubuntu 7.10** – *gutsy*
- **Ubuntu 8.04** – *hardy*
- **Ubuntu 8.10** – *intrepid*
- **Ubuntu 9.04** – *jaunty*
- **Ubuntu 9.10** – *karmic*
- **Ubuntu 10.04** – *lucid*
- **Ubuntu 10.10** – *maverick*
- **Ubuntu 11.04** – *natty*
- **Ubuntu 11.10** – *oneiric*
- **Ubuntu 12.04** – *precise*
- **Ubuntu 12.10** – *quantal*
- **Ubuntu 13.04** – *raring*
- **Ubuntu 13.10** – *saucy*
- **Ubuntu 14.04** – *trusty*
- **Ubuntu 14.10** – *utopic*
- **Ubuntu 15.04** – *vivid*
- **Ubuntu 15.10** – *wily*
- **Ubuntu 16.04** – *xenial*
- **Ubuntu 16.10** – *yakkety*
- **Ubuntu 17.04** – *zesty*
- **Ubuntu 17.10** – *artful*
- **Ubuntu 18.04** – *bionic*
- **Ubuntu 18.10** – *cosmic*
- **Ubuntu 19.04** – *disco*
- **Ubuntu 19.10** – *eoan*
- **Ubuntu 20.04** – *focal*
- **Ubuntu 20.10** – *groovy*
- **Ubuntu 21.04** – *hirsute*
- **Ubuntu 21.10** – *impish*
- **Ubuntu 22.04** – *jammy*
- **Ubuntu 22.10** – *kinetic*
- **Ubuntu 23.04** – *lunar*
- **Ubuntu 23.10** – *mantic*
- **Ubuntu 24.04** – *noble*

- **Ubuntu 24.10** – *oracular*
- **Ubuntu 25.04** – *plucky*
- **Ubuntu 26.04** – *resolute*

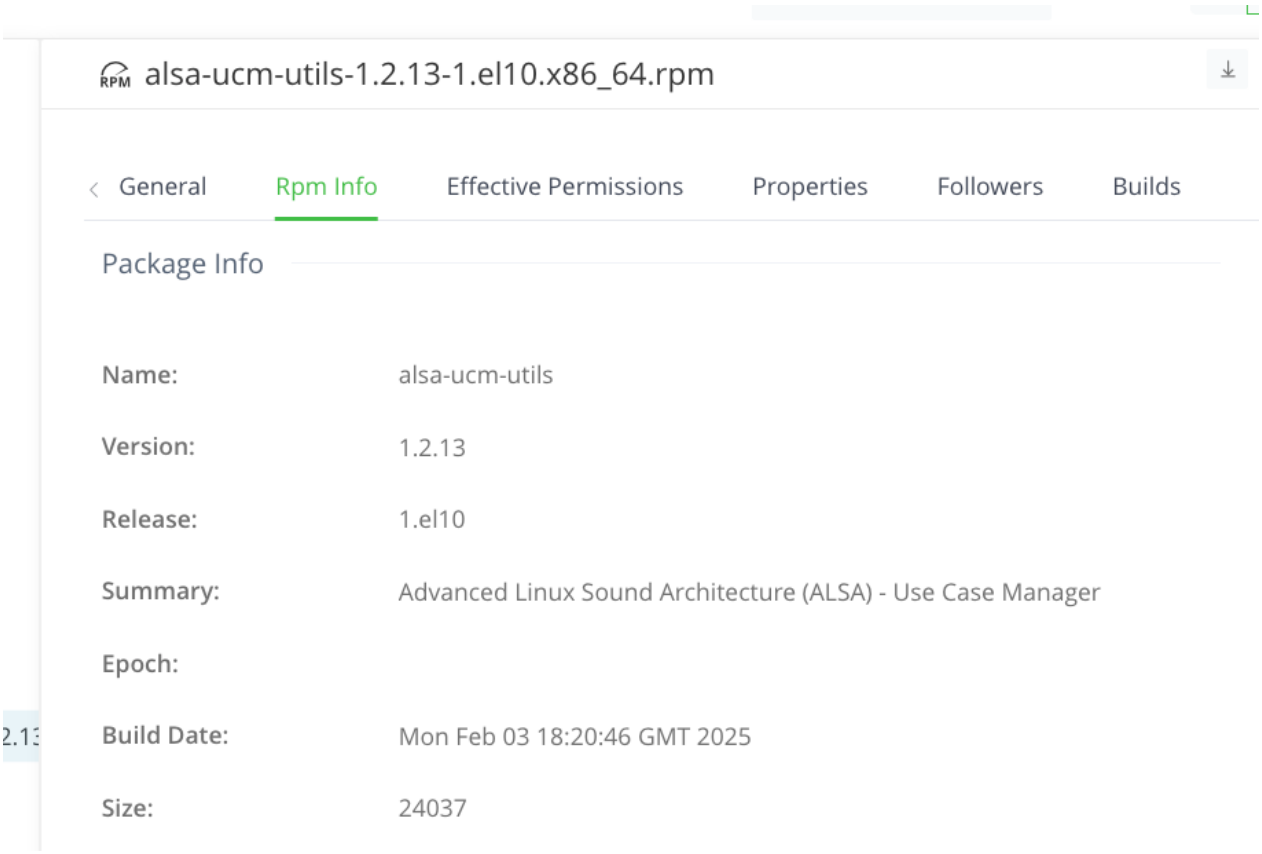
#Просмотр информации о пакете Debian

Плагин извлекает информацию о пакете из нескольких источников. В первую очередь, он получает название пакета, версию и архитектуру из **Properties** артефакта. Если Properties отсутствуют, данные для PURL парсятся из **Repository Path**.



#Просмотр информации о пакете RPM

Для пакетов RPM плагин получает название, версию и архитектуру, анализируя **Repository Path**.



#Плагин для Сфера.Дистрибутивы и лицензии

#Установка плагина

Плагин поставляется в виде jar-файла.

Для установки плагина в **Сфера** необходимо:

1. Поместить jar-файл плагина и файл конфигурации `codescoring.yaml` в папку `plugins`, находящуюся в рабочей директории PPDL приложения.
2. (*Опционально*) Для управления включением/отключением плагинов можно создать в папке `plugins` файлы `enabled.txt` или `disabled.txt`.
    - В файле должны быть перечислены имена включаемых/отключаемых плагинов.
    - Логика включения/выключения:
        - плагин в `disabled.txt` - отключен;
        - `enabled.txt` не пуст и при этом не содержит плагин - отключен;
        - в остальных случаях плагин включен.

#Настройка плагина

Для настройки плагина используется файл `codescoring.yaml`.

Пример содержания файла:

```
codeScoringAPI:
  # Базовый URL для всех эндпоинтов CodeScoring API.
  # Обязательный параметр.
  # Пример: https://host:port или https://host
  url:

  # Ваш API токен CodeScoring для аутентификации.
  # Обязательный параметр.
  token:

  # Размер пула соединений HTTP-клиента к сервису CodeScoring
BE.
  connectionPoolSize: 50

  # По умолчанию, если CodeScoring API не ответил в течение 60
секунд, запрос будет отменен.
  # Этот параметр позволяет настроить длительность таймаута в
секундах.
  timeout: 60

  # Если вы используете прокси, необходимо указать Hostname/IP
и порт.
  proxy:
    host:
    port:

## Если установлено значение 'false', разрешает загрузку
артефактов независимо от ошибок CodeScoringAPI или плагина
blockOnErrors: true

## Если установлено значение 'true', плагин будет сканировать
все поддерживаемые репозитории,
## за исключением указанных в разделе "excludeRepositories".
scanAllRepositories: false

## Настройки по умолчанию для всех репозиториев. Могут быть
переопределены в настройках конкретных репозиториев
(repositories.repo-name)
defaults:
  dockerRegistryUrl: registry.my.domain

  # warmup | Прогрев кэша сканирования без мониторинга
запросов, без блокировки
  # spectator | Прогрев кэша сканирования с мониторингом
запросов, без блокировки
  # moderate | Блокировка на основе политик с использованием
результатов кэша, разрешена загрузка непросканированных
компонентов
```

```
  # strict | Блокировка на основе политик с использованием
результатов кэша, загрузка непросканированных компонентов
заблокирована
  # strict_wait | Блокировка на основе политик, ожидание
завершения сканирования компонента
  # значение по умолчанию — strict_wait, если оно не указано в
настройках или в случае опечатки
  workMode: strict_wait

  # Позволяет этому пользователю пропускать сканирование
  skipScanUser: codescoring

  # URL менеджера репозиториев для применения политик
CodeScoring.
  # Значение ДОЛЖНО быть равно Repository Manager URL в
CodeScoring
  # Пример: https://sfera.my.domain.ru
  repositoryManagerUrl:

## Настройки для конкретных репозиториев
## Пример:
## repositories:
##   docker-remote:
##   docker-local:
##     dockerRegistryUrl: another-registry.my.domain
##     skipScanUser: anotheruser
##     workMode: spectator
##   pypi-remote:
##     workMode: warmup
repositories:

## Список исключенных репозиториев. Используется, если
scanAllRepositories=true
## Пример:
## excludeRepositories:
##   - npm-remote
##   - maven-local
excludeRepositories:

## Список типов репозиториев для сканирования. Используется,
если scanAllRepositories=true
## Поддерживаемые значения: maven, npm, pypi, nuget, go, gems,
debian, yum, alpine, docker
## Пример:
## repositoryTypes:
##   - npm
##   - go
repositoryTypes:
```

#Описание параметров

- **codeScoringAPI** - настройки параметров взаимодействия плагина с платформой CodeScoring;
  - **url** – адрес платформы CodeScoring (обязательно указание протокола);
  - **token** – ключ для авторизации вызовов API (*Создается из CodeScoring раздела `Profile -> Home`*);
  - **connectionPoolSize** – размер пула соединений с платформой CodeScoring;
  - **timeout** - время ожидания ответа (в секундах). По умолчанию, если CodeScoring API не отвечает в течение 60 секунд, запрос будет отменен;
  - **proxy** - настройки прокси-сервера;
    - **host** - хост/IP;
    - **port** - порт;
- **blockOnErrors** - блокирование загрузки компонентов в случае ошибки при взаимодействии с платформой CodeScoring;
- **scanAllRepositories** - подключение всех поддерживаемых репозиториев за исключением указанных в параметре **excludeRepositories**;
- **defaults** – настройки сканирования по умолчанию для всех подключенных репозиториев;
  - **dockerRegistryUrl** – адрес docker registry;
  - **workMode** – режим работы плагина. Условия каждого режима работы описаны в секции ниже;
  - **skipScanUser** – пользователь Сфера, для которого пропускается сканирование компонентов. Необходимо для того, чтобы CodeScoring мог самостоятельно забрать компонент для сканирования. Пользователя необходимо указывать аналогичного тому, что был указан в интеграции Менеджеры репозиториев инсталляции;
  - **repositoryManagerUrl** - URL Sfera. Тот же URL должен быть указан в CodeScoring для применения политик по репозиториям.
- **repositories** – список репозиториев, для которых работает сканирование компонентов. Для каждого репозитория можно отдельно указать параметры, как в параметре **defaults**;
- **excludeRepositories** - список названий репозиториев, исключенных из обработки плагином.

#Настройка режимов работы {: #work-mode-configuration }

Режим работы плагина определяется переменной **workMode** в файле `codescoring.yaml`.

Плагин имеет 6 режимов работы, определяющих строгость проверки компонентов перед загрузкой.

- **warmup** – загрузка данных в кэш CodeScoring без блокировки компонентов;

- **spectator** – загрузка данных в кэш CodeScoring без блокировки компонентов, сохранение результатов запросов компонентов на платформе;
- **moderate** – блокировка компонентов, не прошедших проверку политик. Разрешена загрузка непросканированных компонентов;
- **strict** – блокировка компонентов, не прошедших проверку политик. Запрещена загрузка непросканированных компонентов;
- **strict_wait** – блокировка компонентов, не прошедших проверку политик. Ожидание проверки для непросканированных компонентов.

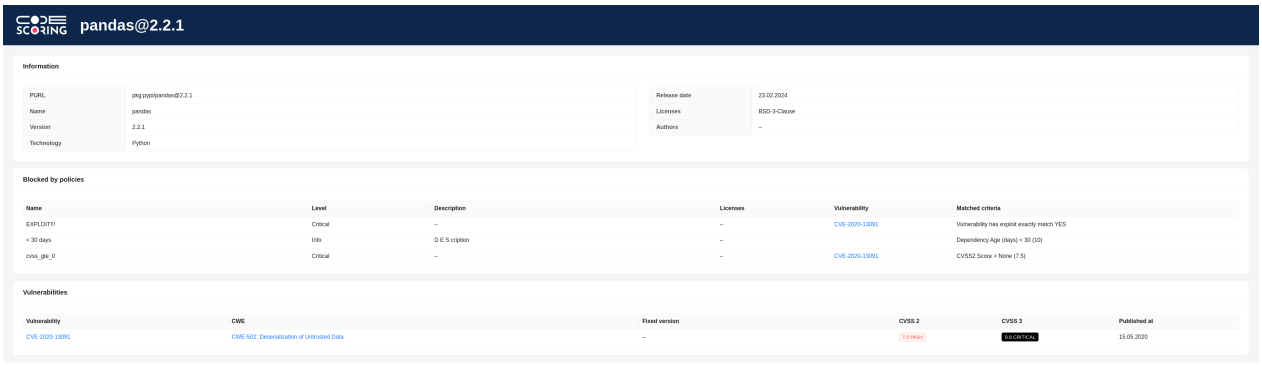
Важно: выбранный режим работы будет влиять на **все** репозитории, указанные в переменной `repositories`.

#Блокировка компонента

При блокировании загрузки компонента в консоли пользователя отображается одна из следующих причин блокировки:

- **"The download has been blocked in accordance with the policies configured in CodeScoring"** – блокировка компонента согласно настроенным на платформе политикам;
- **"The component has not yet been scanned by CodeScoring, it is scheduled to be scanned shortly. The download is blocked according to the plugin settings"** – блокировка непросканированного компонента с последующим запуском сканирования. Используется в режиме `strict`;
- **"The download has been blocked due to the failure of the scan of the component in CodeScoring"** – не удалось просканировать компонент;
- **"The download has been blocked due to the wrong mode of the plugin"** – используется некорректный режим работы плагина;
- **"The download has been blocked due to the timeout of the scan of the component in CodeScoring"** – истекло время ожидания сканирования компонента. Используется в режиме `strict_wait`;
- **"The download has been blocked, because registry is not configured in CodeScoring"** – отсутствует соответствующий Registry в платформе.

Ответ также содержит ссылку на страницу компонента в CodeScoring с информацией о сработавших политиках безопасности и найденных уязвимостях:



#Подключение менеджера репозиториев

CodeScoring.OSA осуществляет проверку артефактов в менеджерах репозиториев **Sonatype Nexus Repository** и **JFrog Artifactory** через плагины OSA. Для более удобной работы с артефактами в платформе можно предварительно настроить подключение к менеджеру репозиториев.

Для добавления нового менеджера репозиториев в платформе необходимо выполнить следующие действия:

1. Перейти в раздел `Настройки -> Менеджеры репозиториев`.
2. Нажать на кнопку **Добавить**.
3. Заполнить поля в форме:
    - **Название** – название в системе CodeScoring;
    - **Тип** – Sonatype Nexus Repository или JFrog Artifactory;
    - **Активно** – признак действующего менеджера репозиториев;
    - **URL** – адрес с указанием протокола. Например: `https://jfrog.example.com`;
    - **Имя пользователя** – имя пользователя с доступом к менеджеру репозиториев;
    - **Пароль**.
4. Проверить подключение после заполнения данных по кнопке **Проверить подключение**.

После создания нового подключения по кнопке **Добавить** менеджер репозиториев отобразится в списке раздела, с возможностью редактирования и удаления.

На странице просмотра можно увидеть список репозиториев со следующими параметрами:

- **Название** – название в рамках менеджера репозиториев;
- **Тип** – тип репозитория (поддерживаются hosted, proxy и virtual);
- **Экосистема** – тип содержащихся артефактов (NPM, PyPI и другие);
- **Активно** – наличие запросов на проверку компонентов за последние 24 часа;

- **Доступно** – доступность репозитория в API (проверяется раз в час);
- **Дата последнего запроса** – дата и время последнего запроса на проверку компонентов.

Процесс обновления данных менеджеров репозиториев

На **17-й минуте каждого часа** система выполняет проверку всех подключённых менеджеров репозиториев. Если подключение успешно, для каждого доступного менеджера выполняется загрузка и обновление списка репозиториев. Альтернативно, список можно обновить вручную.

Увидеть список пакетов и образов из подключенных репозиториев можно в разделах `OSA -> Пакеты` и `OSA -> Образы контейнеров`, а список запросов на проверку в разделе `OSA -> Запросы`.

В рамках разделов доступна фильтрация по отдельным компонентам или следующим полям:

- **Репозиторий** – репозиторий в рамках хранилища артефактов;
- **Технология** – язык программирования или операционная система;
- **Лицензия** – лицензия распространения компонента;
- **Статус блокировки** – признак заблокированного запроса компонента;
- **Последний запрос** – даты последнего запроса компонента;
- **Менеджер репозиториев** – названия подключенного менеджера в CodeScoring;
- **Экосистема репозитория** – тип хранящихся артефактов (PyPI, NPM и другие);
- **Содержит уязвимости** – наличие уязвимостей в запрашиваемых компонентах;
- **Актуальный** – признак обновляемого компонента. Более подробно об этом признаке можно прочитать на странице Обновление данных о компонентах.

Помимо этого, после подключения менеджера появляется возможность настроить политики безопасности для отдельных репозиториев.

#Подключение реестра контейнерных образов

#Поддерживаемые реестры контейнерных образов

CodeScoring поддерживает интеграцию с реестрами образов в следующих инструментах:

- Sonatype Nexus Repository;
- JFrog Artifactory;
- GitLab;
- Harbor;
- GitFlic;

- Иные, использующие протокол Docker Registry V2 API.

#Механизм загрузки контейнерных образов

CodeScoring загружает информацию об образах контейнеров, которые находятся в реестре, следующим образом:

1. Производит листинг названий образов;
2. Для каждого названия образа, производит листинг тэгов;
3. Для каждой пары название-тэг запрашивает манифест;
4. Берёт информацию об архитектуре и sha256 дайджесте из манифеста;
5. Сохраняет информацию о контейнерных образах, находящихся в реесстре на данный момент.

Важно: CodeScoring считает образом уникальную комбинацию следующих данных:

- Реестр, из которого загружена информация об образе;
- Название образа;
- Дайджест sha256 образа.

#Конфигурирование интеграции с реестром контейнерных образов

Для работы с образами необходимо предварительно подключить registry (реестр с образами) в разделе `Настройки -> Реестры`.

Переход на форму создания нового подключения осуществляется по кнопке **Добавить**. В форме необходимо заполнить следующие поля:

- **Название** – название реестра;
- **Тип** – тип менеджера репозитория (Sonatype Nexus Repository, JFrog Artifactory, JFrog Artifactory Repository Path или другой);
- **Активно** – признак действующего реестра. Для недействующих реестров не будет обновляться список доступных образов;
- **Тип авторизации** – тип авторизации (Basic, Bearer или Auto);
- **Адрес** – адрес реестра с указанием протокола. Например: `https://jfrog.example.com`;
- **Максимальное количество одновременных соединений** - максимальное количество соединений, которые будут одновременно открыты при загрузке информации об образах в реестре;
- **Максимальное количество соединений в режиме keep-alive** - максимальное количество соединений, которые будут сохранены для дальнейшего переиспользования при загрузке информации об образах в реестре;
- **Максимальное время жизни соединения в режиме keep-alive, в секундах** - максимальное время в секундах, в течение которого соединение в режиме keep-alive будет существовать для того, чтобы быть переиспользованным при загрузке информации об образах в реестре;

- **Размер страницы при пагинации** - количество записей, которое будет запрашиваться во время пагинированного листинга сущностей в реестре;
- **Таймаут подключения, в секундах** - сколько секунд система будет ожидать создания соединения перед тем, как произойдёт ошибка;
- **Таймаут получения соединения из пула, в секундах** - сколько секунд система будет ожидать получиния соединения из пула соединений. Это ожидание часто происходит, когда настроено существенное ограничение RPS или количества одновременных соединений, поэтому, в этих случаях необходимо выставлять высокое значение;
- **Таймаут на чтение, в секундах** - сколько секунд система будет ожидать записи в соединение перед тем, как произойдёт ошибка;
- **Таймаут на запись, в секундах** - сколько секунд система будет ожидать чтения из соединения перед тем, как произойдёт ошибка;
- **Максимальное количество запросов в секунду** - ограничение на максимальное количества запросов в секунду при загрузке информации об образах в реестре;
- **Пропустить проверку TLS?** – пропустить проверку сертификатов для TLS/SSL соединений;
- **Имя пользователя** – имя пользователя с доступом к реестру;
- **Пароль** – пароль для доступа к реестру (для типа менеджера GitFlic следует указывать транспортный токен);
- **Загрузить полный список образов?** - регулярно выгружать из реестра данные о присутствующих образах;
- **Репозитории для загрузки** - репозитории, из которых будут загружаться образы (доступно для типа менеджера JFrog Artifactory Repository Path).

Важные заметки об интеграции с реестром, реализуемом GitLab:

- при подключении GitLab Container Registry необходимо выбрать тип авторизации **Bearer**;
- Согласно Документации GitLab, листинг всех названий образов на эндпоинте `/v2/_catalog/` требует данных об администраторской УЗ.

Проверить подключение после заполнения данных можно по кнопке **Проверить подключение**.

#Просмотр информации об интеграции

После создания нового подключения по кнопке **Добавить** реестр отобразится в списке раздела, с возможностью просмотреть информацию о нем (**Просмотр**), изменить параметры подключения (**Редактировать**), или удалить подключение (**Удалить **).

Для обновления списка доступных образов необходимо нажать на кнопку **Обновить списки образов** на странице просмотра. Также на странице просмотра доступна проверка подключения по кнопке **Обновить статус**.

#Настройка политик OSA

Для принятия решения о блокировании или разрешении загрузки компонентов плагин использует механизм политик CodeScoring.

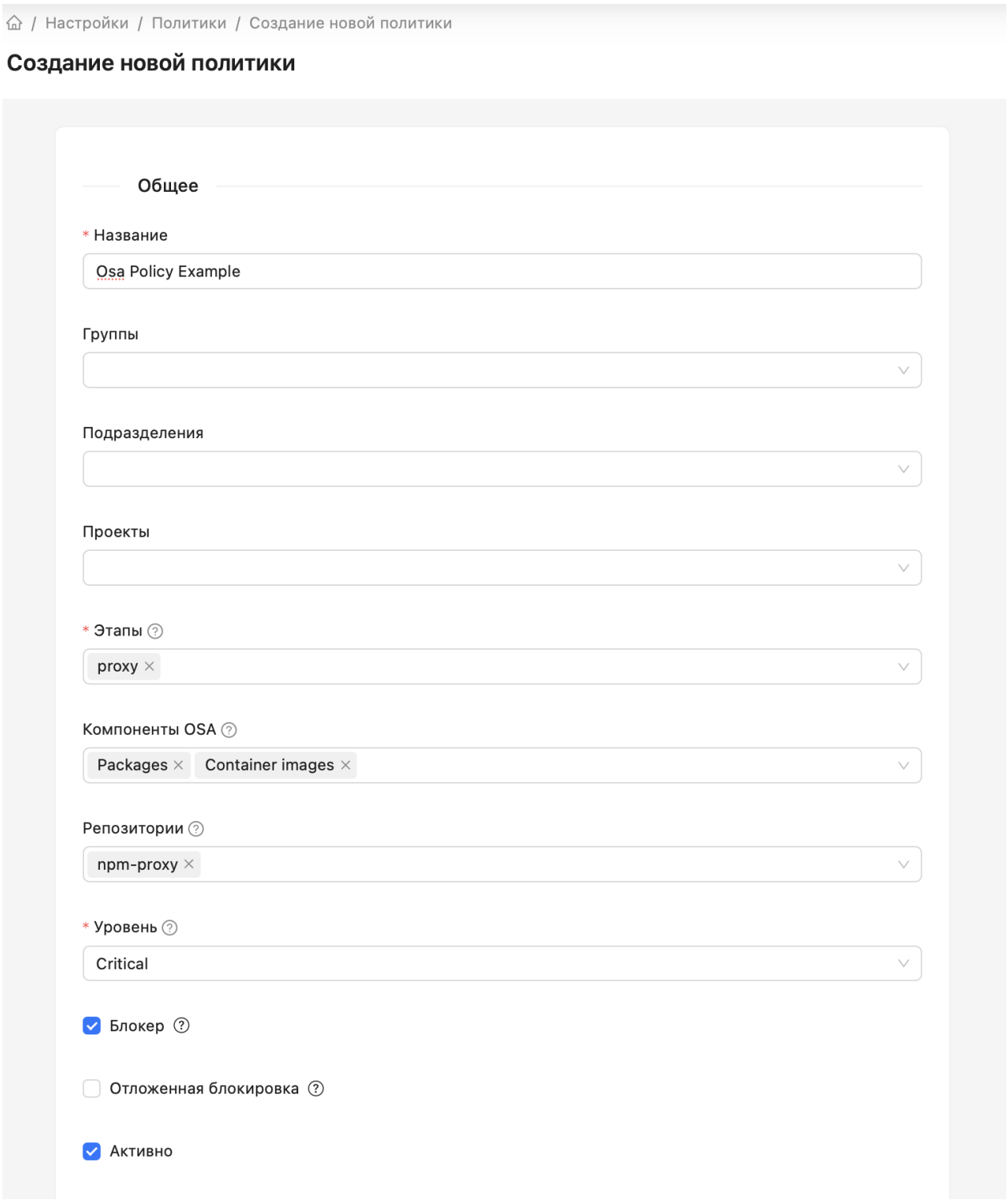
Для того чтобы политика применялась при вызове от плагина, необходимо задать следующие настройки выбранной политики в разделе `Policies`:

- **Этапы** – указать значение **proxy**;
- если необходимо, чтобы политика блокировала запрос компонента, установить признак **Блокер**;
- установить признак **Активно**.

Помимо этого, можно уточнить область применения политики по двум параметрам:

- **Компоненты OSA** – тип компонентов, на которые будет распространяться политика (пакеты или контейнерные образы);
- **Репозиториев** – список репозиториев в рамках хранилища артефактов.

Важно: для выбора репозитория необходимо сперва подключить менеджер репозиториев в платформе.



#Работа с компонентами OSA в платформе

Компоненты, которые прошли проверку плагином, отображаются в разделе 0SA интерфейса CodeScoring.

#Просмотр списка пакетов

Список просканированных пакетов можно посмотреть в подразделе 0SA -> Пакеты. Таблица в данном разделе содержит **все** пакеты,

которые проходили проверку за время работы OSA плагина, со следующей информацией:

- **Пакет** – название пакета (со ссылкой на его индивидуальную страницу);
- **Технология** – технология (язык программирования или инструмент сборки);
- **Лицензии** – лицензии;
- **Авторы** – авторы;
- **Уязвимости** – количество найденных уязвимостей в пакете;
- **Статус блокировки** – статус блокировки компонента на момент последнего запроса;
- **Выпущено** – дата и время публикации версии пакета;
- **Последний запрос** – дата и время последнего запроса пакета.

Таблицу с пакетами можно отфильтровать по технологии, лицензии, статусу блокировки или времени последнего запроса компонента.

По нажатию на название пакета осуществляется переход на его индивидуальную страницу, где отображается информация о найденных уязвимостях и сработавших политиках.

#Индивидуальная страница пакета

На индивидуальной странице пакета отображаются основные атрибуты компонента:

- **PURL** – уникальный идентификатор пакета;
- **Актуальный** – статус актуальности компонента;
- **Технология** – технология пакета;
- **Лицензии** – лицензии пакета;
- **Версия** – версия пакета;
- **Авторы** – авторы пакета;
- **Домашняя страница** – ссылка на страницу проекта;
- **Index URL** – ссылка на страницу пакета в пакетном индексе;
- **Выпущено** – дата и время публикации версии;
- **Последний запрос** – дата и время последнего запроса;
- **Первый запрос** – дата и время первого запроса;
- **Политики обновлены** – дата и время последнего обновления политик для компонента в соответствии с механизмом обновления;
- **Статус** - статус отзыва пакета;
- **Репозиторий** – имя репозитория, из которого получен пакет;
- **Менеджер репозиториев** – название подключенного менеджера репозиториев;
- **Ссылка на пакет** – ссылка на пакет в менеджере репозиториев.

#Просмотр списка контейнерных образов

Контейнерные образы отображаются в подразделе `OSA -> Образы контейнеров` после подключения соответствующего реестра. Каждая запись в списке содержит следующую информацию:

- **Название** – название образа;
- **Реестр контейнеров** – название реестра, в котором содержится образ;
- **Зависимости** – количество найденных зависимостей;
- **Уязвимости** - количество найденных уязвимостей;
- **Статус сканирования** – статус сканирования образа;
- **Статус блокировки** – статус блокировки компонента на момент последнего запроса (для репозиториев с плагином OSA);
- **Последнее сканирование** – дата и время последнего сканирования;
- **Последний запрос** – дата и время последнего запроса.

Таблицу с образами можно отфильтровать по названию реестра, а также статусу сканирования и блокировки.

По умолчанию добавленный образ не является просканированным. Для проведения анализа необходимо перейти на страницу образа и нажать на кнопку **Запустить SCA**. По результатам сканирования на странице образа появится информация о найденных зависимостях и уязвимостях, а также список сработавших политик безопасности и список слоёв данного образа.

Для образа доступно скачивание SBOM и PDF-отчета по данным последнего успешного сканирования.

#Индивидуальная страница контейнерного образа

На индивидуальной странице контейнерного образа отображаются основные атрибуты компонента:

- **PURL** – уникальный идентификатор образа;
- **Дайджест** – дайджест образа;
- **Теги** – теги образа;
- **Актуальный** – статус актуальности компонента;
- **Политики обновлены** – дата и время последнего обновления политик для компонента в соответствии с механизмом обновления;
- **Последний запрос** – дата и время последнего запроса;
- **Первый запрос** – дата и время первого запроса;
- **Последнее сканирование** – дата и время последнего сканирования;
- **Статус блокировки** – статус блокировки компонента на момент последнего запроса;
- **Реестр контейнеров** – имя реестра, в котором размещен образ;
- **Ссылки на образ в реестре** – ссылки на образ в подключенных реестрах;

- **Название базового образа** – название образа, на основе которого собран данный образ;
- **Версия базового образа** – версия образа, на основе которого собран данный образ;
- **Репозиторий** – имя репозитория, из которого получен образ;
- **Менеджер репозиториев** – название подключенного менеджера репозиториев.

#Просмотр списка слоёв образов

Слои контейнерных образов отображаются в подразделе OSA -> Слои образов после сканирования образов через SCA проект или в разделе OSA -> Образы контейнеров. Список помогает оценивать состав и размер слоёв, а также то, сколькими образами используется каждый слой.

Таблицу с фильтрами можно отфильтровать по id слоя, типу медиа и образу, в котором содержатся слои.

Каждая запись в списке содержит следующую информацию:

- **ID** – идентификатор слоя (дайджест) и краткое описание инструкции сборки, сформировавшей слой, по клику на которое можно посмотреть полную команду;
- **Тип медиа** – MIME-тип содержимого слоя;
- **Размер, МБ** – размер слоя в мегабайтах;
- **Количество образов** – число образов, в которых встречается данный слой.

Как и в других разделах со списками, доступны постраничный просмотр, выбор числа записей на странице, настройка отображения таблицы и сортировка по колонкам.

#Просмотр запросов компонентов

Список запросов компонентов из прокси-репозиториев с подключенным плагином можно посмотреть в подразделе OSA -> Запросы.

Запросы пакетов отображаются на вкладке **Пакеты** и по умолчанию содержат следующую информацию:

- **Пакет** – название пакета (со ссылкой на его индивидуальную страницу);
- **Технология** – технология;
- **Режим плагина** – режим работы плагина;
- **Статус блокировки** – статус блокировки;
- **Дата запроса** – дата и время запроса;
- **Инициатор запроса** - имя инициатора запроса в менеджере репозиториев.

Запросы контейнерных образов отображаются на вкладке **Образы контейнеров** и по умолчанию содержат следующую информацию:

- **Контейнерный образ** – название образа (со ссылкой на его индивидуальную страницу);
- **Реестр** – название реестра, в котором содержится образ;
- **Режим плагина** – режим работы плагина;
- **Статус блокировки** – статус блокировки;
- **Дата запроса** – дата и время запроса;
- **Инициатор запроса** - имя инициатора запроса в менеджере репозиториев.

Также для обеих таблиц можно отобразить колонки **Пользователь CodeScoring** (пользователь CodeScoring, настроенный в плагине OSA) и **Запрашиваемый PURL** (идентификатор запрашиваемого компонента).

#Обновление данных о компонентах

CodeScoring.OSA автоматически в фоне обновляет информацию о ранее запрошенных компонентах для обеспечения актуальности данных и должного уровня быстродействия.

#Механизм обновления

Пакеты и образы, которые успешно запрашивались хотя бы один раз за последние 14 дней, считаются актуальными, срок можно поменять в настройках. Система обновляет по таким пакетам данные каждые 2 часа, включая сведения по уязвимостям, лицензиям и мета-информацию по пакетам.

Если компонент не запрашивался в течение 14 дней, он автоматически переводится в статус **архивного**. Такие компоненты больше не обновляются до следующего запроса.

#Архивирование и удаление компонентов

По умолчанию данные об архивных компонентах сохраняются в системе, однако возможно включить их автоматическое удаление. Это поведение регулируется параметрами в конфигурации приложения (файл `app.env`):

- `OSA_ARCHIVE_THRESHOLD_DAYS` — через сколько дней без запросов компонент считается архивным (по умолчанию: 14);
- `OSA_ARCHIVE_AUTO_CLEANUP_ENABLED` — включение удаления данных об архивных компонентов (по умолчанию: `False`);
- `OSA_ARCHIVE_RETENTION_PERIOD_DAYS` — срок хранения данных об архивных компонентов перед удалением (по умолчанию: `30`);
- `OSA_ARCHIVE_CHUNK_SIZE` — размер порции для пакетной обработки компонентов при архивации и удалении (по умолчанию: `1000`).

#Фильтрация по актуальности

В разделах OSA → Пакеты, OSA → Образы и Алерты доступен фильтр **Актуальный**, позволяющий управлять отображением компонентов:

- **Да** — отображаются только актуальные (обновляемые) компоненты;
- **Нет** — отображаются только архивные (не обновляемые) компоненты.

По умолчанию отображаются все компоненты.

#Развертывание

После настройки файла application.yml приложение может быть либо развернуто и выполнено в среде контейнера Docker, либо оркестрировано с помощью Helm-чарта в Kubernetes.

#Развертывание в контейнере Docker

Чтобы запустить приложение как контейнер Docker, выполните следующую команду:

```
docker run -d \
-p 8080:8080 \
-e SPRING_CONFIG_ADDITIONAL_LOCATION=file:/app/config/ \
-v /path/to/your/config/application.yml:/app/config/
application.yml \
--name cs-proxy \
<registry-address>/cs-proxy:<tag>
```

#Развертывание в Kubernetes (Helm Chart)

Для сред Kubernetes приложение может быть развернуто с использованием предоставленного Helm-чарта, доступного по адресу https://{REGISTRY_URL}/repository/helm.

Порядок установки:

1. Создать namespace.

   ```
   kubectl create namespace cs-proxy
   ```

2. Создать secret для доступа к приватному реестру Docker-образов, используя адрес (REGISTRY_URL), логин (USERNAME) и пароль (PASSWORD), полученные от вендора.

```
kubectl create secret docker-registry codescoring-regcred
--docker-server=REGISTRY_URL --docker-username=USERNAME --
docker-password=PASSWORD -n cs-proxy
```

3. Установить Helm предпочтительным способом.

4. Выполнить следующие команды для добавления актуального Helm-репозитория на локальную машину:

```
helm repo add codescoring-org https://{REGISTRY_URL}/
repository/helm/ --username USERNAME --password PASSWORD
helm repo update
```

5. Создать файл `values.yaml` со следующим содержимым:

```
env:
  javaOpts: "-Xmx4g" # Опционально: настройка параметров
JVM

config: |
  # Данное поле необходимо заполнить текстом
конфигурационного файла application.yml

# Существует возможность создания ресурса Ingress
ingress:
  enabled: true
  className: ""
  annotations: {}
  hosts:
    - host: cs-proxy.example.com
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: cs-proxy
              port:
                number: 8080
  tls:
    - secretName: cs-proxy-tls
      hosts:
        - cs-proxy.example.com
```

6. Выполнить команду для установки чарта

```
helm install cs-proxy codescoring-org/cs-proxy -n cs-proxy
-f values.yaml --create-namespace --atomic --version
CHART_VERSION
```

#Настройка сервиса

#Основные параметры

Конфигурация **OSA Proxy** осуществляется через файл `application.yml`:

Пример конфигурационного файла

```
## Параметры CodeScoring
codescoring:
  host: URL-адрес сервера CodeScoring
  token: токен авторизации (с уровнем доступа User и выше)
  work-mode: рабочий режим (применяется только к сканированию
пакетов)
              # warmup | Разогрев кэша сканирования без
мониторинга запросов, без блокировки
              # spectator | Разогрев кэша сканирования с
мониторингом запросов, без блокировки
              # moderate | Блокировка на основе политик с
использованием результатов кэша, загрузка непроверенных
компонентов разрешена
              # strict | Блокировка на основе политик с
использованием результатов кэша, загрузка непроверенных
компонентов заблокирована
              # strict_wait | Блокировка на основе политик,
ожидание, пока компонент не будет отсканирован
  proxy-manager-host: хост прокси-сервера
  enable-status-
line: true/false (добавляет сообщение о причине блокировки в
строку состояния)
  block-status-code: статус код для блокировки загрузки пакетов
  block-on-codescoring-errors: блокирует загрузку пакета при
5xx status, ошибках сканирования (scan_failed)
  override-block-url: true/false (заменяет URL в ссылке на
причину блокировки на указанный в codescoring.host)
  remove-blocked-versions: true/false (по умолчанию true; при
true — заблокированные версии удаляются из манифеста, при false
— помечаются как устаревшие)
## Настройки PyPI
pypi:
  enabled: true
  repository:
    - name: internet-pypi
      scan-manifest: true
      scan-package: true
      url-encoded-config: true
      registry: https://pypi.org
      packages-registry: https://files.pythonhosted.org
    - name: arti-pypi
      scan-manifest: true
      scan-package: true
```

```
      registry: http://localhost:8081/artifactory/api/pypi/
pypi-remote
      packages-registry: http://localhost:8081/artifactory/api/
pypi/pypi-remote/packages
    - name: nexus-pypi
      scan-manifest: true
      scan-package: true
      registry: https://localhost:8081/repository/pypi-proxy
      packages-registry: https://localhost:8081/repository/
pypi-proxy/packages
## Настройки Maven
maven:
  enabled: true
  repository:
    - name: internet-mvn
      scan-manifest: true
      scan-package: true
      url-encoded-config: true
      registry: https://repo1.maven.org/maven2
    - name: arti-mvn
      scan-manifest: false
      scan-package: true
      registry: http://localhost:8081/artifactory/maven-remote
    - name: nexus-mvn
      scan-manifest: false
      scan-package: true
      registry: http://localhost:8081/repository/maven-proxy
## Настройки NPM
npm:
  enabled: true
  repository:
    - name: internet-npm
      scan-package: true
      scan-manifest: true
      url-encoded-config: true
      registry: https://registry.npmjs.org
    - name: arti-npm
      scan-package: true
      scan-manifest: true
      registry: http://localhost:8081/artifactory/api/npm/npm-
remote
    - name: nexus-npm
      scan-package: true
      scan-manifest: true
      registry: http://localhost:8081/repository/npm-proxy
## Настройки NuGet
nuget:
  enabled: true
  repository:
    - name: codescoring-nuget
      scan-package: true
      url-encoded-config: true
```

```
      registry: https://api.nuget.org
    - name: arti-nuget
      scan-package: true
      registry: http://localhost:8081/artifactory/api/nuget/v3/
nuget-remote
    - name: nexus-nuget
      scan-package: true
      scan-manifest: true
      registry: http://localhost:8081/repository/nuget-v3-proxy
## Настройки GO
go:
   enabled: true
   repository:
     - name: codescoring-go
       scan-manifest: true
       scan-package: true
       url-encoded-config: true
       registry: https://proxy.golang.org/
       sumdb-registry: https://sum.golang.org
     - name: arti-go
       scan-package: true
       scan-manifest: true
       url-encoded-config: true
       registry: http://localhost:8081/artifactory/api/go/go-
virt
     - name: nexus-go
       scan-package: true
       scan-manifest: true
       url-encoded-config: true
       registry: http://localhost:8081/repository/go-proxy/
## Настройка Debian
debian:
  enabled: true
  repository:
    - name: codescoring-debian
      scan-package: true
      url-encoded-config: true
      registry: https://ports.ubuntu.com/ubuntu-ports/
      distro: plucky
    - name: arti-debian
      scan-package: true
      url-encoded-config: true
      registry: http://localhost:8081/artifactory/debian-remote
      distro: plucky
    - name: nexus-debian
      scan-package: true
      url-encoded-config: true
      registry: http://localhost:8081/repository/debian11
      distro: bullseye
## Настройка Alpine (APK)
alpine:
  enabled: true
```

```
  repository:
    - name: codescoring-alpine
      scan-package: true
      registry: https://dl-cdn.alpinelinux.org/alpine
    - name: arti-alpine
      scan-package: true
      registry: http://localhost:8081/artifactory/alpine-remote
## Настройка RPM
rpm:
  enabled: true
  repository:
    - name: codescoring-rpm
      scan-package: true
      registry: https://repo.almalinux.org/almalinux
    - name: arti-rpm
      scan-package: true
      registry: http://localhost:8081/artifactory/rpm-remote
## Настройка Docker Registry
docker:
  enabled: true
  repository:
    - name: codescoring-docker
      registry: https://registry-1.docker.io
      auth-token-url: https://auth.docker.io
    - name: arti-docker
      registry: http://localhost:8081/artifactory/docker-remote
      auth-token-url: http://localhost:8081
```

Особенности работы в Nexus Repository и JFrog Artifactory

- Для JFrog Artifactory рекомендуется выставить `Custom Base URL` и использовать его в поле `registry` для корректной замены ссылок на пакеты внутри манифестов;
- В конфигурации `пакетный менеджер -> jfrog -> OSA proxy -> internet`, в дополнительных настройках репозитория JFrog необходимо выставить флаг `Bypass HEAD requests`.
- Для Nexus Repository идентичного функционала нет, в манифестах будет использован хост и порт (если указан) из запроса. При наличии `reverse proxy` рекомендуется использовать ссылку на него. Например:
  `registry: https://nexushost.ru/repository/pypi-proxy`.

#Дополнительные настройки

#Настройки уровня логирования

Пример настройки логирования

```
logging:
  level:
```

```
    ru:
      codescoring: info
```

#Просмотр заблокированных пакетов в логах

Чтобы найти заблокированные пакеты в логах приложения, убедитесь, что уровень логирования для `ru.codescoring` установлен на `info` или ниже. Компонент `PolicyLogger` выводит информацию о заблокированных пакетах в следующих форматах:

- Для пакетов, заблокированных политиками:
  `Policy '<policy_name>' blocked package '<package_name>' versions: [<versions>]`
- Для пакетов OSA, заблокированных платформой:
  `Policy blocked package '<purl>' for endpoint '<endpoint>': <reason>`

#Логирование внешних запросов

Внешние запросы в сторонние реестры можно логировать с помощью логгера `ru.codescoring.proxy.logging.RegistryRequestResponseLogger`. Для этого необходимо установить уровень логирования `trace` для данного компонента.

Пример настройки логирования внешних запросов

```
logging:
  level:

ru.codescoring.proxy.logging.RegistryRequestResponseLogger:
trace
```

#Режим обработки заблокированных версий в манифестах

Параметр `codescoring.remove-blocked-versions` управляет тем, как заблокированные версии пакетов отображаются в манифестах npm, PyPI и NuGet:

- `true` (по умолчанию) — заблокированные версии **полностью удаляются** из манифеста. Пакетный менеджер не видит их и не предлагает пользователю.
- `false` — заблокированные версии **остаются в манифесте**, но помечаются как устаревшие с указанием имени сработавшей политики:
  - **npm** — поле `deprecated` версии содержит имя политики;
  - **PyPI** — атрибут `data-yanked` ссылки на пакет содержит имя политики;
  - **NuGet** — поле `deprecation.message` записи содержит имя политики, `listed` устанавливается в `false`.

Пример настройки

```
codescoring:
  remove-blocked-versions: false
```

#Размер буфера для обработки больших манифестов

Пример настройки размера буфера

```
spring:
  http:
    codecs:
      max-in-memory-size: 150MB (это настройка по умолчанию,
уже включенная в приложение, увеличьте ее, если вы столкнулись
с очень большими манифестами)
```

#Политики повторных попыток и circuit breaker для запросов к платформе:

#Настройка повторных попыток

Эта конфигурация определяет политику повторных попыток для сервиса `codeScoringApi`. Она настроена на обработку временных сбоев путем повторной попытки запроса до 3 раз.

Повторные попытки используют стратегию экспоненциального отступления, начиная с задержки в 1 секунду и удваивая ее с каждой попыткой. Эта политика применяется только к определенным исключениям, таким как `WebClientRequestException`.

#Настройка Circuit Breaker

Circuit breaker (автоматический выключатель) для `codeScoringApi` действует как механизм быстрого отказа. Он отслеживает частоту сбоев и, если она достигает 50% (рассчитывается по последним 20 вызовам), он «открывается» и предотвращает дальнейшие запросы в течение 30 секунд. Это дает нижестоящему сервису время на восстановление. После периода ожидания он переходит в «полуоткрытое» состояние, позволяя пройти 5 пробным вызовам, чтобы определить, восстановился ли сервис.

Конфигурация Retry и Circuit Breaker может быть переопределена путем установки следующих свойств, например, для `codeScoringApi`.

#Добавление truststore сертификатов

Пример добавления truststore сертификатов в application.yml

```
spring:
  cloud:
    gateway:
      server:
        webflux:
          httpclient:
            ssl:
              trustedX509Certificates:
                - /usr/local/share/ca-certificates/
codescoring.crt
                - /etc/ssl/certs/ca-certificates.crt
```

#Добавление http proxy

Пример настройки http proxy

```
spring:
  cloud:
    gateway:
      httpclient:
        proxy:
          host: proxy.host.ru
          username: 'username'
          port: 9091
          password: 'password'
          non-proxy-hosts-pattern: '(localhost|
127.0.0.1|.*\.internal\.com)'
```

#Настройка Redis и кэширования

Для повышения производительности и снижения нагрузки на платформу CodeScoring поддерживается кэширование результатов работы политик (вердиктов сервиса Judge). Для работы кэширования требуется подключение к Redis.

Настройки Redis и кэширования

```
spring:
  data:
    redis:
      host: localhost
      port: 6379
      database: 0         # Номер базы данных (опционально)
      password: password  # Опционально
      timeout: 2000ms

cache:
  judge:
```

```
    enabled: true  # Включение кэширования (по умолчанию false)
    ttl: 24h       # Время жизни записи в кэше
    refresh-after: 30m  # Время, после которого запись
считается устаревшей и требует обновления (но все еще может
быть отдана из кэша)
    proactive-refresh-enabled: true # Включение проактивного
(фонового) обновления кэша
    proactive-refresh-interval:
2h  # Интервал запуска фонового обновления
    key-prefix: "cs:judge:" # Префикс для ключей в Redis
```

Особенности продления времени жизни (TTL) в кэше

Проактивное обновление не продлевает TTL (время жизни) записи в кэше. TTL продлевается только при чтении данных из кэша реальными запросами пользователей. Это позволяет автоматически удалять из Redis редко запрашиваемые пакеты и хранить только востребованные данные.

#Swagger UI

OSA Proxy предоставляет Swagger UI для просмотра документации API и управления кэшем.

- **URL:** `http://<osa-proxy-host>:<port>/api/swagger`
- **Доступные операции:**
  - Очистка кэша по PURL
  - Очистка кэша по типу пакета

#Поддерживаемые протоколы

Данный раздел содержит форматы данных и правила модификации ответов для каждого поддерживаемого пакетного менеджера в OSA Proxy.

#Maven

#Обрабатываемые файлы

- `maven-metadata.xml` - манифест с информацией о версиях
- `.jar`, `.war`, `.ear` - файлы пакетов

#Модификация полей в maven-metadata.xml

```
<metadata>
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <versioning>
    <latest>обновляется на последнюю незаблокированную</latest>
```

```
    <release>обновляется на последнюю незаблокированную</relea
se>
    <versions>
      <version>удаляются заблокированные версии</version>
    </versions>
  </versioning>
</metadata>
```

#NPM

#Обрабатываемые файлы

- JSON манифест пакета (путь `/{repository}/*`)
- `.tgz` - архивы пакетов

#Модификация полей в NPM манифесте

```
{
  "name": "package-name",
  "dist-tags": {
    "latest": "обновляется на последнюю незаблокированную
версию"
  },
  "versions": {
    "1.0.0": "удаляются заблокированные версии"
  },
  "time": {
    "1.0.0": "удаляются записи для заблокированных версий"
  }
}
```

#PyPI

#Обрабатываемые файлы

- HTML страницы Simple API (путь `/{repository}/simple/*`)
- `.zip`, `.tar`, `.tgz`, `.tar.gz`, `.tar.bz2`, `.egg`, `.whl` - файлы пакетов

#Модификация HTML страниц

- Удаляются ссылки для заблокированных версий
- Перезаписываются URL для скачивания через прокси

```
<!DOCTYPE html>
<html>
  <body>
    <a href="https://files.pythonhosted.org/packages/
example-1.0.0.tar.gz">example-1.0.0.tar.gz</a>
```

```
    <a href="https://files.pythonhosted.org/packages/
example-2.0.0.tar.gz">example-2.0.0.tar.gz</a>
  </body>
</html>
```

#NuGet

#Обрабатываемые файлы

- `index.json` - сервисный индекс
- Registration index JSON
- `.nupkg` - файлы пакетов

#Модификация registration индекса

```
{
  "version": "3.0.0",
  "items": [
    {
      "@id": "https://api.nuget.org/v3/registration5-gz-
semver2/package/index.json",
      "items": [
        {
          "catalogEntry": {
            "id": "Package",
            "version": "1.0.0"
          }
        },
        {
          "catalogEntry": {
            "id": "Package",
            "version": "2.0.0"
          }
        }
      ]
    }
  ]
}
```

#Go

#Обрабатываемые файлы

- Список версий (`/@v/list`)
- `.zip` — архивы модулей

#Модификация списка версий

- Из списка версий удаляются заблокированные версии.

#Debian

#Обрабатываемые файлы

- `.deb` — файлы пакетов

Особенности сканирования Debian

Для Debian поддерживается только сканирование пакетов. Модификация манифестов (файлов Packages) не производится.

#Alpine

#Обрабатываемые файлы

- `.apk` — файлы пакетов

Особенности сканирования Alpine

Для Alpine поддерживается сканирование пакетов. Модификация индексов (APKINDEX) не производится.

#RPM

#Обрабатываемые файлы

- `.rpm` — файлы пакетов

Особенности сканирования RPM

Для RPM поддерживается сканирование пакетов. Модификация метаданных (repodata) не производится.

#Docker

#Обрабатываемые файлы

- Manifests (v2 API)
- Слои образов (Blobs)

#Модификация манифестов

- Из мультиархитектурных манифестов (Manifest Lists) удаляются дайджесты заблокированных образов.

#Поведение при полной блокировке пакета

В случае, когда все доступные версии запрашиваемого пакета заблокированы политиками безопасности, OSA Proxy возвращает сообщение о блокировке всех версий.

Поскольку некоторые клиенты пакетных менеджеров могут не отображать это специфическое сообщение о блокировке в пользовательском интерфейсе, рекомендуется использовать утилиту `curl` для прямой диагностики статуса пакета. Ниже представлены примеры запросов с использованием `curl` для проверки статуса блокировки для различных типов пакетов:

#Pip

```
curl http://localhost:8080/codescoring-pypi/simple/имя_пакета
```

#Maven

```
curl http://localhost:8080/codescoring-maven/groupid/
artifactid/maven-metadata.xml
```

#npm

```
curl http://localhost:8080/codescoring-npm/имя_пакета
```

#NuGet

Хотя NuGet-клиент может выводить причину блокировки всех пакетов в консоли, прямой запрос через curl также позволяет получить подтверждение статуса:

```
curl http://localhost:8080/codescoring-nuget/nuget-api/v3/
registration5-gz-semver2/newtonsoft.json/index.json
```

#Go

```
curl http://localhost:8080/codescoring-go/имя_модуля/@v/list
```

#Сбор метрик

Метрики доступны в **OSA Proxy** по адресу `{osa-proxy-url}/actuator/metrics` в формате JSON, а также в формате для prometheus `{platform-url}/actuator/prometheus`.

Эти метрики собираются для каждого типа репозитория (`maven`, `pypi`, `nuget`, `npm`, `go`, `debian`, `alpine`, `rpm`, `docker`) и позволяют детально отслеживать входящие запросы к прокси-репозиториям.

#Доступные метрики

- `gateway_route_<package-type>_requests_seconds_count` – общее количество обработанных запросов;
- `gateway_route_<package-type>_requests_seconds_sum` – суммарное время обработки запросов, используется для расчета среднего времени ответа;
- `gateway_route_<package-type>_requests_seconds_max` – максимальное время обработки запроса;
- `gateway_route_<package-type>_requests_seconds_bucket` – SLO (Service Level Objective) метрики времени ответа с бакетами: 10ms, 25ms, 50ms, 100ms, 250ms, 500ms, 1s, 2s, 5s.

В рамках сбора метрик `<package-type>` заменяется на соответствующий тип репозитория: `maven`, `pypi`, `nuget`, `npm`, `debian`, `alpine`, `rpm`, `docker`. Например, для Maven-репозитория метрика будет называться `gateway_route_maven_requests_total`.

Данные метрики можно отфильтровать по следующим лейблам:

- **`operation`** – тип операции, выполняемой с пакетом;
  - `scan_package` – сканирование пакета;
  - `scan_manifest` – сканирование манифеста;
  - `other` – другие операции (например передача файлов не подпадающих под анализ).
- **`method`** – HTTP-метод запроса (`GET`, `POST`, `PUT`, и т.д.);
- **`repository`** – имя репозитория, к которому был выполнен запрос;
- **`status`** – код статуса HTTP-ответа (например, `200`, `403`, `500`);
- **`outcome`** – результат обработки запроса;
  - `success` – запрос успешно обработан;
  - `error` – произошла ошибка при обработке (статус 400 и выше, кроме кода блокировки);
  - `blocked_by_policies` – запрос был заблокирован политиками безопасности.

#Метрики обращений в CodeScoring

Для мониторинга взаимодействия с платформой CodeScoring доступны следующие метрики:

- `codescoring_api_requests_seconds_count` – общее количество запросов к API CodeScoring;
- `codescoring_api_requests_seconds_sum` – суммарное время выполнения запросов к API;
- `codescoring_api_requests_seconds_max` – максимальное время выполнения запроса к API;
- `codescoring_api_requests_seconds_bucket` – SLO метрики времени ответа API с бакетами: 10ms, 25ms, 50ms, 100ms, 250ms, 500ms, 1s, 2s, 5s.

Данные метрики позволяют отслеживать:

- Производительность взаимодействия с платформой CodeScoring
- Количество запросов на сканирование компонентов
- Время отклика API для выявления проблем связи
- Нагрузку на платформу со стороны OSA Proxy

#Параметры в URL в формате Base64

#Применение параметров в URL в формате Base64 для `osa-proxy`

Взаимодействие с `osa-proxy` в некоторых сценариях требует явного указания дополнительных параметров в пути URL. Это достигается путём кодирования требуемой информации в формате Base64 (URL-safe).

Основная цель использования Base64-кодированных параметров — предоставление `osa-proxy` необходимого контекста для корректного применения политик безопасности, особенно когда `osa-proxy` выступает в роли посредника для внешних репозиториев.

#Автоматическое определение контекста

Когда `osa-proxy` размещён между клиентом (пакетным менеджером) и внутренним менеджером репозиториев (например, JFrog Artifactory или Nexus Repository Manager), `osa-proxy` может автоматически извлечь информацию о хосте и имени репозитория из настроек конечного репозитория.

Пример конфигурации, где контекст определяется автоматически:

```
npm:
  repository:
    - name: codescoring-npm
      # ...
      registry: https://nexus.test.ru/repository/npm-proxy
```

#Явное указание контекста через Base64-параметры

В случаях, когда `osa-proxy` напрямую взаимодействует с внешними, общедоступными репозиториями (например, `https://registry.npmjs.org`), он не имеет возможности самостоятельно получить информацию о внутреннем хосте и имени репозитория. В такой ситуации для `osa-proxy` критически важно получить эти данные для применения привязанных политик безопасности и правильной обработки запроса.

Для этого используется строка, закодированная в Base64, которая содержит JSON-объект с параметрами, такими как `repoManagerHost` и `repoName`. Эта строка встраивается непосредственно в URL запроса, позволяя `osa-proxy` получить необходимый контекст.

Пример конфигурации, требующей явного указания контекста:

```
npm:
  repository:
    - name: codescoring-npm
      # ...
      registry: https://registry.npmjs.org # Здесь нужна
передача параметров
```

#Механизм работы:

Закодированная строка параметров в формате Base64 размещается в пути URL сразу после имени репозитория. `osa-proxy` декодирует эту строку, извлекает параметры и использует их для выполнения своих функций, включая применение политик безопасности, ассоциированных с конкретным внутренним репозиторием.

Общая структура URL:
`https://<osaproxy-host>/<repository-name>/<base64-parameters>/<rest-of-path>`

#Передача контекста для корректной работы политик безопасности, привязаных к репозиториям Nexus и Artifactory

`Клиент разработчика` -> `Nexus / Artifactory` -> `osa-proxy` -> `Интернет`

Для передачи контекстной информации, включающей хост и имя репозитория вашего менеджера репозиториев, эти данные следует интегрировать в Base64-кодированную строку параметров. Важно строго соблюдать правило, согласно которому данная Base64-строка должна располагаться непосредственно после имени репозитория в URL-адресе.

#Обновление конфигурации

Необходимо пометить репозиторий, как совместимый с Base64 параметрами `url-encoded-config: true`

```
npm:
  repository:
    - name: codescoring-npm
      url-encoded-config: true
```

```
      # ...
      registry: https://registry.npmjs.org
```

#Nexus

1. Перейдите в **Server Administration** -> **Repositories**.
2. Выберите желаемый тип (например, `maven2 (proxy)`).
3. В поле **Remote storage** введите URL вашего экземпляра `osa-proxy`, включая имя репозитория и параметры, закодированные в Base64.

Пример для прокси-репозитория Maven:
`https://osaproxy.example.com/internet-maven/eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL3JlcG8xLm1hdmVuLm9yZy9tYXZlbjIiLCJyZXBvTmFtZSI6ImludGVybmV0LW1hdmVuIn0/maven2`

#Artifactory

1. Перейдите в **Administration** -> **Repositories** -> **Remote**.
2. В конфигурации установите поле **URL** на URL `osa-proxy`. Этот URL должен включать имя репозитория и строку, закодированную в Base64.

Пример для удаленного репозитория PyPI:
`https://osaproxy.example.com/internet-pypi/eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL3B5cGkub3JnL3NpbXBsZSIsInJlcG9OYW1lIjoiaW50ZXJuZXQtcHlwaSJ9`

#Правило

Закодированная строка параметров в формате Base64 должна быть размещена в пути URL сразу после имени репозитория.

Общая структура URL выглядит следующим образом:
`https://<osaproxy-host>/<repository-name>/<base64-parameters>/<rest-of-path>`

Где:

- `<osaproxy-host>`: Имя хоста экземпляра `osa-proxy`.
- `<repository-name>`: Имя репозитория, к которому осуществляется доступ.
- `<base64-parameters>`: Закодированная в URL-safe Base64 JSON-строка, содержащая параметры.
- `<rest-of-path>`: Оставшаяся часть пути из настроек пакетного менеджера.

#Пример

Например, нужно передать следующие параметры в виде JSON-объекта.

```
{"repoManagerHost":"https://nexus.test.ru","repoName":"npm-proxy"}
```

Для этого следует:

1. **Преобразовать JSON-объект в строку.**
2. **Закодировать строку с использованием URL-safe Base64.**

Результат кодирования JSON-объекта выше в Base64:
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZXBvTmFtZSI6Im5wbS1wcm94eSJ9

#Настройка менеджеров пакетов

Чтобы постоянно использовать URL с параметрами в формате Base64 для всех запросов, необходимо обновить конфигурационный файл вашего менеджера пакетов.

#NPM

Для NPM нужно отредактировать файл `.npmrc` и установить ключ `registry`.

URL должен включать имя репозитория и строку, закодированную в Base64.

```
registry=https://osaproxy.example.com/npm-proxy/eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZXBvTmFtZSI6Im5wbS1wcm94eSJ9
```

#Maven

Для Maven нужно отредактировать файл `settings.xml`. Вы можете добавить новое `<mirror>` в секцию `<mirrors>`.

Тег `<url>` должен содержать полный URL, включая имя репозитория и строку, закодированную в Base64.

```
<settings>
  ...
  <mirrors>
    <mirror>
      <id>osa-proxy-mirror</id>
      <mirrorOf>*</mirrorOf>
      <url>https://osaproxy.example.com/my-maven-repo/eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZXBvTmFtZSI6Im5wbS1wcm94eSJ9/maven2</url>
    </mirror>
  </mirrors>
```

```
  ...
</settings>
```

Убедитесь, что значение <mirrorOf> соответствует репозиториям, которые вы хотите проксировать.

#Go

Для Go установите переменную окружения GOPROXY, чтобы она включала имя репозитория и строку, закодированную в Base64.

```
export GOPROXY="https://osaproxy.example.com/go-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6ImdvLXJlcG8ifQ"
```

#Debian

Для Debian нужно отредактировать файл /etc/apt/sources.list или файл в /etc/apt/sources.list.d/. Обновите поле URIs.

```
Types: deb
URIs: https://osaproxy.example.com/debian-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6ImRlYmlhbi1yZXBvIn0=
Suites: stable
Components: main
Signed-By: /path/to/key.gpg
```

#NuGet

Для NuGet отредактируйте файл NuGet.config и добавьте новый источник пакетов. Атрибут value тега <add> должен содержать полный URL.

```
<configuration>
  <packageSources>
    <add key="osa-proxy" value="https://osaproxy.example.com/
nuget-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6Im51Z2V0LXJlcG8ifQ/index.json" />
  </packageSources>
  ...
</configuration>
```

#PyPI

Для PyPI отредактируйте файл `pip.conf` (Linux/macOS) или `pip.ini` (Windows) и установите `index-url`.

```
[global]
index-url = https://osaproxy.example.com/pypi-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6InB5cGktcmVwbyJ9/simple
```

#Конфигурация Maven

#Миграция URL репозитория

Сценарий использования: миграция репозитория Maven с Artifactory на OSA Proxy.

Следующая таблица содержит сводку по перенаправлению URL репозиториев для Maven. Параметры аутентификации и другие настройки, такие как имя пользователя и пароль, остаются без изменений.

Источник	URL в settings.xml до миграции	URL в settings.xml после миграции	application.yml maven.repository.registry
Nexus	`https://nexus.host.ru/repository/maven-remote`	`https://{osa-proxy-url}/nexus-mvn`	`https://nexus.host.ru/repository/maven-remote`
Artifactory	`https://jfrog.host.ru/artifactory/maven-remote`	`https://{osa-proxy-url}/jfrog-mvn`	`https://jfrog.host.ru/artifactory/maven-remote`
Официальный репозиторий	`https://repo.maven.apache.org/maven2`	`https://{osa-proxy-url}/inet-mvn`	`https://repo.maven.apache.org/maven2`

#Миграция Maven репозитория

Исходный файл `.m2/settings.xml`:

```
<settings>
  <mirrors>
```

```
    <mirror>
      <id>artifactory</id>
      <mirrorOf>*</mirrorOf>
      <url>https://jfrog.host.ru/artifactory/maven-remote</url>
    </mirror>
  </mirrors>
  <servers>
    <server>
      <id>artifactory</id>
      <username>your-username</username>
      <password>your-password</password>
    </server>
  </servers>
</settings>
```

Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл `application.yml`) в секцию maven. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле `application.yml`

```
maven:
  enabled: true
  repository:
    - name: jfrog-mvn
      scan-manifest: true
      scan-package: true
      registry: https://jfrog.host.ru/artifactory/maven-remote
```

Обновлённый файл `.m2/settings.xml`:

```
<settings>
  <mirrors>
    <mirror>
      <id>cs-proxy</id>
      <mirrorOf>*</mirrorOf>
      <url>https://{osa-proxy-url}/jfrog-mvn</url>
    </mirror>
  </mirrors>
  <servers>
    <server>
      <id>cs-proxy</id>
      <username>your-username</username>
      <password>your-password</password>
    </server>
  </servers>
</settings>
```

#Конфигурация NPM

#Миграция URL репозитория

Сценарий использования: миграция репозитория npm с Artifactory на OSA Proxy.

Следующая таблица содержит сводку по перенаправлению URL репозиториев для NPM. Параметры аутентификации и другие настройки, такие как имя пользователя и пароль, остаются без изменений.

Источник	.npmrc registry: до миграции	.npmrc registry: после миграции	application.yml npm.repository.registry
Nexus	`https://nexus.host.ru/repository/npm-proxy`	`https://{osa-proxy-url}/nexus-npm`	`https://nexus.host.ru/repository/npm-proxy`
Artifactory	`https://jfrog.host.ru/artifactory/api/npm/npm-remote`	`https://{osa-proxy-url}/jfrog-npm`	`https://jfrog.host.ru/artifactory/api/npm/npm-remote`
Официальный репозиторий	`https://registry.npmjs.org`	`https://{osa-proxy-url}/inet-npm`	`https://registry.npmjs.org`

#Миграция NPM репозитория

Исходный файл `.npmrc`:

```
registry=https://artifactory.domain.ru/artifactory/api/npm/npm-remote/
//artifactory.domain.ru/artifactory/api/npm/npm-remote/:_password=1NHTGVrUnJQ
//artifactory.domain.ru/artifactory/api/npm/npm-remote/:username=asdf
//artifactory.domain.ru/artifactory/api/npm/npm-remote/:email=asdf@domain.ru
//artifactory.domain.ru/artifactory/api/npm/npm-remote/:always-auth=true
```

Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл `application.yml`) в секцию npm. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле `application.yml`

```
npm:
  enabled: true
  repository:
    - name: arti-npm
      scan-package: true
      scan-manifest: true
      registry: https://artifactory.domain.ru/artifactory/api/
npm/npm-remote/
```

Обновлённый файл .npmrc:

```
registry=https://{osa-proxy-url}/arti-npm
//{osa-proxy-url}/arti-npm/:_password=1NHTGVrUnJQ
//{osa-proxy-url}/arti-npm/:username=asdf
//{osa-proxy-url}/arti-npm/:email=asdf@domain.ru
//{osa-proxy-url}/arti-npm/:always-auth=true
```

#Конфигурация NuGet

#Миграция URL репозитория

Сценарий использования: миграция репозитория NuGet с Artifactory на OSA Proxy.

Следующая таблица содержит сводку по перенаправлению URL репозиториев для NuGet. Параметры аутентификации и другие настройки, такие как имя пользователя и пароль, остаются без изменений.

Источник	URL в NuGet.config до миграции	URL в NuGet.config после миграции	application.yml nuget.repository.registry
Nexus	`https://nexus.host.ru/repository/nuget.org-proxy/index.json`	`https://{osa-proxy-url}/nexus-nuget/nuget-api/index.json`	`https://nexus.host.ru/repository/nuget.org-proxy`
Artifactory			

Источник	URL в NuGet.config до миграции	URL в NuGet.config после миграции	application.yml nuget.repository.registry
	https://jfrog.host.ru/artifactory/api/nuget/v3/nuget-safe	https://{osa-proxy-url}/arti-nuget/nuget-api	https://jfrog.host.ru/artifactory/api/nuget/v3/nuget-safe
Официальный репозиторий	https://api.nuget.org/v3/index.json	https://{osa-proxy-url}/inet-nuget/nuget-api/v3/index.json	https://api.nuget.org

#Миграция NuGet репозитория

Исходный файл NuGet.config:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="Artifactory" value="https://jfrog.host.ru/
artifactory/api/nuget/v3/nuget-safe" />
  </packageSources>
  <packageSourceCredentials>
    <Artifactory>
      <add key="Username" value="your-username" />
      <add key="ClearTextPassword" value="your-password" />
    </Artifactory>
  </packageSourceCredentials>
</configuration>
```

Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл application.yml) в секцию nuget. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле application.yml

```
nuget:
  enabled: true
  repository:
    - name: arti-nuget
      scan-package: true
      registry: https://jfrog.host.ru/artifactory/api/nuget/v3/
nuget-safe
```

Обновлённый файл `NuGet.config`:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="OSA-Proxy" value="https://{osa-proxy-url}/arti-
nuget/nuget-api" />
  </packageSources>
  <packageSourceCredentials>
    <OSA-Proxy>
      <add key="Username" value="your-username" />
      <add key="ClearTextPassword" value="your-password" />
    </OSA-Proxy>
  </packageSourceCredentials>
</configuration>
```

#Конфигурация PyPI

#Миграция URL репозитория

Сценарий использования: миграция репозитория PyPI с Artifactory на OSA Proxy.

Следующая таблица содержит сводку по перенаправлению URL репозиториев для PyPI. Параметры аутентификации и другие настройки, такие как имя пользователя и пароль, остаются без изменений.

Источник	**URL в pip.conf / pip.ini до миграции**	**URL в pip.conf / pip.ini после миграции**	**application.yml pypi.repository.registry**
Nexus	`https://nexus.host.ru/repository/pip-remote/simple`	`https://{osa-proxy-url}/nexus-pypi/simple`	`https://nexus.host.ru/repository/pip-remote`
Artifactory	`https://jfrog.host.ru/artifactory/api/pypi/pypi-remote/simple`	`https://{osa-proxy-url}/jfrog-pypi/simple`	`https://jfrog.host.ru/artifactory/api/pypi/pypi-remote`
Официальный репозиторий	`https://pypi.org/simple`	`https://{osa-proxy-url}/inet-pypi/simple`	`https://pypi.org`

#Миграция PyPI репозитория

Исходный файл `pip.conf` (Linux/macOS) или `pip.ini` (Windows):

```
[global]
index-url = https://jfrog.host.ru/artifactory/api/pypi/pypi-
remote/simple
trusted-host = jfrog.host.ru
```

Или с аутентификацией:

```
[global]
index-url = https://username:password@jfrog.host.ru/
artifactory/api/pypi/pypi-remote/simple
trusted-host = jfrog.host.ru
```

Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл `application.yml`) в секцию pypi. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле `application.yml`

```
pypi:
  enabled: true
  repository:
    - name: jfrog-pypi
      scan-manifest: true
      scan-package: true
      registry: https://jfrog.host.ru/artifactory/api/pypi/
pypi-remote
      packages-registry: https://jfrog.host.ru/artifactory/api/
pypi/pypi-remote/packages
```

Пример настройки для GitLab в `application.yml`:

```
pypi:
  enabled: true
  repository:
    - name: python-sdk
      scan-manifest: true
      scan-package: true
      registry: https://gitlab.example.com/api/v4/projects/337/
packages/pypi
      packages-registry: https://gitlab.example.com/api/v4/
projects/337/packages/pypi/files
```

Обновлённый файл `pip.conf` (Linux/macOS) или `pip.ini` (Windows):

```
[global]
index-url = https://{osa-proxy-url}/jfrog-pypi
trusted-host = {osa-proxy-url}
```

Или с аутентификацией:

```
[global]
index-url = https://username:password@{osa-proxy-url}/jfrog-
pypi
trusted-host = {osa-proxy-url}
```

#Настройка нескольких реестров пакетов

Некоторые PyPI-репозитории могут отдавать пакеты с нескольких хостов. Например, индекс `download.pytorch.org` содержит ссылки как на собственные CDN-хосты, так и на стандартный `files.pythonhosted.org`.

Для корректного проксирования таких репозиториев используется параметр `additional-packages-registries` — словарь, где ключ задаёт хост источника, а значение — URL реестра пакетов, на который нужно перенаправлять запросы.

Пример настройки для репозитория PyTorch:

```
pypi:
  enabled: true
  repository:
    - name: pytorch-pypi
      scan-manifest: true
      scan-package: true
      registry: https://download.pytorch.org
      packages-registry: https://download.pytorch.org
      additional-packages-registries:
        download.pytorch.org: https://download.pytorch.org
        download-r2.pytorch.org: https://download-
r2.pytorch.org
        files.pythonhosted.org: https://files.pythonhosted.org
```

#Расположение конфигурационных файлов

- **Linux/macOS**: `~/.config/pip/pip.conf` или `~/.pip/pip.conf`
- **Windows**: `%APPDATA%\pip\pip.ini` или `%HOME%\pip\pip.ini`
- **Для virtualenv**: `$VIRTUAL_ENV/pip.conf`

#Конфигурация Go

#Миграция прокси для Go

Сценарий использования: миграция Go для использования OSA Proxy вместо прямого доступа или внешних публичных прокси.

Следующая таблица содержит сводку по перенаправлению URL для прокси Go. Параметры аутентификации и другие настройки (если применимы, например, для частных репозиториев, требующих специфических учетных данных) должны быть настроены отдельно в соответствии с вашими корпоративными политиками (например, через `.netrc` или SSH-ключи).

Источник модулей / Репозиторий	GOPROXY до миграции	GOPROXY после миграции
Nexus	`https://nexus.host.ru/repository/go-remote`	`https://{osa-proxy-url}/nexus-go`
Artifactor	`https://jfrog.host.ru/artifactory/api/go/go-virt`	`https://{osa-proxy-url}/arti-go`
Официальный прокси Go	`https://proxy.golang.org`	`https://{osa-proxy-url}/inet-go`

Checksum Database (sum.golang.org)

Checksum DB не является отдельным GOPROXY-эндпоинтом. Вместо этого он настраивается через переменную GOSUMDB. Подробнее — в разделе ниже.

#Детали миграции прокси Go

#Настройка окружения до миграции

До миграции ваш GOPROXY мог быть установлен на публичный прокси Go (`https://proxy.golang.org`) или не задан вовсе, что приводило к использованию `proxy.golang.org` по умолчанию. Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл `application.yml`) в секцию go. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле `application.yml`

```
go:
  enabled: true
  repository:
```

```
    - name: inet-go
      scan-package: true
      scan-manifest: true
      registry: https://proxy.golang.org
      sumdb-registry: https://sum.golang.org
```

Пример текущей конфигурации переменных окружения (например, в файле `.bashrc`, `.zshrc` или в CI/CD пайплайне):

```
export GOPROXY=https://{osa-proxy-url}/inet-go
```

#Настройка Checksum Database

Для проксирования запросов к `sum.golang.org` через OSA Proxy используется переменная `GOSUMDB`. Её значение задаётся в формате `<имя-базы> <url-прокси>`, где URL строится как `{osa-proxy-url}/{repo-name}/sumdb/sum.golang.org`:

```
export GOSUMDB="sum.golang.org https://{osa-proxy-url}/inet-go/
sumdb/sum.golang.org"
```

Полный пример запуска:

```
GOPROXY="https://{osa-proxy-url}/inet-go" \
GOSUMDB="sum.golang.org https://{osa-proxy-url}/inet-go/sumdb/
sum.golang.org" \
go get github.com/example/module@v1.0.0
```

#Конфигурация Debian пакетов

#Миграция URL репозитория

Сценарий использования: миграция репозиториев Debian с прямых источников на прокси-сервер OSA Proxy.

Следующая таблица содержит сводку по перенаправлению URL репозиториев для Debian. Обратите внимание, что формат строк репозитория deb или `deb-src` (дистрибутив, компоненты) остается без изменений, меняется только базовый URL репозитория.

Источник	URL в sources.list до миграции	URL в sources.list после миграции	application.yml apt.repository.registry
Nexus			

Источник	URL в sources.list до миграции	URL в sources.list после миграции	application.yml apt.repository.registry
	https://nexus.host.ru/repository/debian-group	https://{osa-proxy-url}/nexus-debian	https://nexus.host.ru/repository/debian-group
Artifactory	https://jfrog.host.ru/artifactory/debian-virtual	https://{osa-proxy-url}/jfrog-debian	https://jfrog.host.ru/artifactory/debian-virtual
Официальный Debian	https://deb.debian.org/debian/	https://{osa-proxy-url}/inet-debian	http://deb.debian.org/debian/

#Миграция APT репозитория

Исходный файл /etc/apt/sources.list или /etc/apt/sources.list.d/*.list:

```
Types: deb
URIs: https://deb.debian.org/debian
Suites: noble noble-updates noble-backports
Components: main universe restricted multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл application.yml) в секцию debian. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле application.yml

```
debian:
  enabled: true
  repository:
    - name: debian-apt
      scan-package: true
      distro: bullseye
      registry: http://deb.debian.org/debian/
```

После настройки прокси-сервера и добавления его в application.yml, ваш sources.list будет выглядеть так:

```
Types: deb
URIs: https://{osa-proxy-url}/codescoring-debian
```

```
Suites: noble noble-updates noble-backports
Components: main universe restricted multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

#Конфигурация Docker Registry

#Миграция URL реестра

Сценарий использования: миграция Docker реестров с прямых источников на прокси-сервер OSA Proxy.

Следующая таблица содержит сводку по перенаправлению URL для Docker.

Источник	URL до миграции	URL после миграции	application.yml docker.repository.registry
Nexus	`nexus.host.ru:5000`	`{osa-proxy-url}/nexus-docker`	`https://nexus.host.ru:5000`
Artifactory	`jfrog.host.ru/docker-remote`	`{osa-proxy-url}/jfrog-docker`	`https://jfrog.host.ru/docker-remote`
Docker Hub	`registry.hub.docker.com`	`{osa-proxy-url}/codescoring-docker`	`https://registry-1.docker.io`

#Миграция Docker клиента

Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл `application.yml`) в секцию docker. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле `application.yml`

```
docker:
  enabled: true
  repository:
    - name: codescoring-docker
      scan-package: true
      registry: https://registry-1.docker.io
      auth-token-url: https://auth.docker.io
```

После настройки прокси-сервера и добавления его в application.yml, команда для загрузки образа будет выглядеть так:

```
docker pull {osa-proxy-url}/library/alpine:latest
```

#Использование поддоменов для доступа

При использовании более одного Docker-репозитория необходимо включить поддержку поддоменов. Имена поддоменов должны соответствовать именам репозиториев из конфигурации `docker.repository`.

В этом случае команда для загрузки образа будет выглядеть так:

```
docker pull codescoring-docker.osaproxyhost.ru/library/postgres
```

Если настроен только один репозиторий, использование поддоменов не требуется — Docker-реестр будет доступен напрямую через хост OSA Proxy:

```
docker pull osaproxyhost.ru/library/postgres
```

#Конфигурация Alpine пакетов

#Миграция URL репозитория

Сценарий использования: миграция репозиториев Alpine с прямых источников на прокси-сервер OSA Proxy.

Следующая таблица содержит сводку по перенаправлению URL репозиториев для Alpine.

Источник	URL в repositories до миграции	URL в repositories после миграции	application.yml alpine.repository.registry
Nexus	`https://nexus.host.ru/repository/alpine-proxy`	`https://{osa-proxy-url}/nexus-alpine`	`https://nexus.host.ru/repository/alpine-proxy`
Artifactory	`https://jfrog.host.ru/artifactory/alpine-remote`	`https://{osa-proxy-url}/jfrog-alpine`	`https://jfrog.host.ru/artifactory/alpine-remote`

Источник	URL в repositories до миграции	URL в repositories после миграции	application.yml alpine.repository.registry
Официальный Alpine	`https://dl-cdn.alpinelinux.org/alpine`	`https://{osa-proxy-url}/inet-alpine`	`https://dl-cdn.alpinelinux.org/alpine`

#Миграция APK репозитория

Исходный файл `/etc/apk/repositories`:

```
https://dl-cdn.alpinelinux.org/alpine
```

Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл `application.yml`) в секцию alpine. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле `application.yml`

```
alpine:
  enabled: true
  repository:
    - name: codescoring-alpine
      scan-package: true
      registry: https://dl-cdn.alpinelinux.org/alpine
```

После настройки прокси-сервера и добавления его в application.yml, ваш файл репозиториев будет выглядеть так:

```
https://{osa-proxy-url}/codescoring-alpine
```

#Конфигурация RPM пакетов

#Миграция URL репозитория

Сценарий использования: миграция репозиториев RPM (YUM/DNF) с прямых источников на прокси-сервер OSA Proxy.

Следующая таблица содержит сводку по перенаправлению URL репозиториев для RPM.

Источник	baseurl в .repo до миграции	baseurl в .repo после миграции	application.yml rpm.repository.registry
Nexus	`https://nexus.host.ru/repository/rpm-proxy`	`https://{osa-proxy-url}/nexus-rpm`	`https://nexus.host.ru/repository/rpm-proxy`
Artifactory	`https://jfrog.host.ru/artifactory/rpm-remote`	`https://{osa-proxy-url}/jfrog-rpm`	`https://jfrog.host.ru/artifactory/rpm-remote`
Официальный Mirror	`https://repo.almalinux.org/almalinux`	`https://{osa-proxy-url}/inet-rpm`	`https://repo.almalinux.org/almalinux`

#Миграция YUM/DNF репозитория

Исходный файл `/etc/yum.repos.d/almalinux.repo`:

```
[baseos]
name=AlmaLinux $releasever - BaseOS
baseurl=https://repo.almalinux.org/almalinux/$releasever/
BaseOS/$basearch/os/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-AlmaLinux-9
```

Следующее определение репозитория необходимо добавить в YAML-конфигурацию сервиса (файл `application.yml`) в секцию rpm. Для применения изменений требуется перезапуск сервиса.

Конфигурация в файле `application.yml`

```
rpm:
  enabled: true
  repository:
    - name: codescoring-rpm
      scan-package: true
      registry: https://repo.almalinux.org/almalinux
```

После настройки прокси-сервера и добавления его в application.yml, конфигурация репозитория будет выглядеть так:

```
[baseos]
name=AlmaLinux $releasever - BaseOS
```

```
baseurl=https://{osa-proxy-url}/codescoring-rpm/$releasever/
BaseOS/$basearch/os/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-AlmaLinux-9
```

#OSA Proxy Go

OSA Proxy Go — это Go-версия прокси-сервиса для пакетных реестров с проверками CodeScoring OSA. Сервис принимает запросы пакетных менеджеров, проксирует их в upstream-реестры, проверяет манифесты и скачиваемые артефакты, блокирует запрещенные компоненты и отдает служебные endpoints для healthcheck, метрик, Swagger UI и управления кэшем.

#Поддерживаемые экосистемы

OSA Proxy Go поддерживает следующие типы реестров:

- npm;
- Composer;
- Maven;
- NuGet;
- PyPI;
- RubyGems;
- Go modules;
- Debian;
- Alpine;
- RPM;
- Docker Registry API v2.

::::note Альтернативные репозитории

OSA Proxy Go может работать не только с публичными реестрами, но и с менеджерами репозиториев, которые реализуют протоколы соответствующих экосистем, например Sonatype Nexus Repository, JFrog Artifactory или CodeScoring.Save. ::::

#Основные возможности

#Сканирование манифестов и пакетов

Для поддерживаемых экосистем доступны два уровня проверки:

- **сканирование манифестов** — анализ metadata/индексов пакетов и исключение заблокированных политиками версий из ответа пакетному менеджеру;
- **сканирование пакетов** — проверка скачиваемых архивов, бинарных пакетов или образов перед передачей клиенту.

Поддержка уровней зависит от экосистемы. Например, npm, Maven, NuGet, PyPI, Go, Composer и RubyGems поддерживают проверку metadata и пакетов, а Debian, Alpine и RPM — проверку скачиваемых пакетов без модификации системных индексов.

#Блокировка небезопасных компонентов

Если компонент нарушает политики безопасности, OSA Proxy Go может удалить небезопасные версии из metadata, заблокировать скачивание артефакта и вернуть настраиваемый HTTP-код блокировки.

#Модификация ответов

При включенном сканировании манифестов сервис модифицирует ответы upstream-реестров: удаляет заблокированные версии, обновляет ссылки на скачивание через прокси и сохраняет формат ответа, ожидаемый пакетным менеджером.

#Кэширование вердиктов

Для снижения нагрузки на CodeScoring и ускорения повторных запросов OSA Proxy Go поддерживает Redis-кэш результатов проверки Judge. Кэш выключен по умолчанию и настраивается в секции cache.

#Маршруты

Для всех экосистем, кроме Docker, имя маршрута берется из поля `name` в секции `repository` файла `osa-proxy.yml`.

Тип реестра	Форма маршрута
npm, Composer, Maven, NuGet, PyPI, Ruby, Go, Debian, Alpine, RPM	`GET /{repository-name}/{path...}`
Docker	`/v2/{path...}` и `GET /token`

Например, репозиторий npm с именем `npm` будет доступен по адресу:

```
https://osa-proxy.example.com/npm/
```

Docker-режим использует стандартные endpoints Docker Registry API v2 и не добавляет имя репозитория в путь:

```
docker pull osa-proxy.example.com/library/alpine:latest
```

#Служебные endpoints

Endpoint	Назначение
`GET /healthz`	Проверка, что процесс OSA Proxy Go запущен.
`GET /metrics`	Метрики в формате Prometheus.
`GET /api/v3/api-docs`	OpenAPI JSON.
`GET /api/swagger/`	Swagger UI.
`DELETE /api/cache/purls`	Удаление конкретных PURL из кэша вердиктов.
`DELETE /api/cache/packages/{packageType}`	Удаление записей кэша по типу пакета, имени пакета или repository context.

#Режимы работы

Поведение проверки задается параметром `work-mode`. Его можно указать глобально в `codescoring.work-mode` и переопределить для конкретного репозитория через `repository[*].work-mode`.

- `warmup` — разогрев кэша без блокировки компонентов;
- `spectator` — разогрев кэша и сохранение результатов запросов без блокировки;
- `moderate` — блокировка по политикам, загрузка непросканированных компонентов разрешена;
- `strict` — блокировка по политикам, загрузка непросканированных компонентов запрещена;
- `strict_wait` — блокировка по политикам с ожиданием проверки для непросканированных компонентов.

#Установка сервиса

OSA Proxy Go использует конфигурационный файл `osa-proxy.yml`. По умолчанию сервис ищет его в рабочей директории, но путь можно передать первым аргументом запуска или через переменную окружения `OSA_PROXY_CONFIG_PATH`.

#Docker

Пример запуска контейнера с внешним конфигурационным файлом:

```
docker run -d \
  --name osa-proxy \
  -p 8080:8080 \
  -e OSA_PROXY_CONFIG_PATH=/etc/osa-proxy/osa-proxy.yml \
  -v /path/to/osa-proxy.yml:/etc/osa-proxy/osa-proxy.yml:ro \
  <registry-url>/osa-proxy:<tag>
```

Проверка доступности:

```
curl http://localhost:8080/healthz
```

#Docker Compose

```
services:
  osa-proxy:
    image: <registry-url>/osa-proxy:<tag>
    container_name: osa-proxy
    ports:
      - "8080:8080"
    env_file:
      - .env
    environment:
      OSA_PROXY_CONFIG_PATH: /etc/osa-proxy/osa-proxy.yml
    volumes:
      - ./osa-proxy.yml:/etc/osa-proxy/osa-proxy.yml:ro
    healthcheck:
      test: ["CMD", "/app/osa-proxy", "healthcheck"]
      interval: 30s
      timeout: 3s
      retries: 3
      start_period: 5s
```

Если включен Redis-кэш вердиктов, добавьте Redis в Compose-файл и укажите его адрес в `cache.redis.address`.

#.env файл

При запуске через Docker Compose файл `.env` передается в контейнер только если он указан в `env_file`. Эти переменные можно использовать:

- напрямую как переменные окружения процесса;
- в `osa-proxy.yml` через плейсхолдеры вида `${VAR_NAME:default_value}`.

В примере Compose путь к конфигурации задается отдельно через `environment.OSA_PROXY_CONFIG_PATH`, поэтому не дублируйте его в `.env`.

Пример `.env`:

```
CODESCORING_URL=https://codescoring.example.com
CODESCORING_TOKEN=
WORK_MODE=strict_wait
OSA_PROXY_URL=https://osa-proxy.example.com
LOG_LEVEL=info

CACHE_ENABLED=false
REDIS_ADDRESS=redis:6379
REDIS_PASSWORD=
REDIS_DB=0
```

Пример использования переменных в `osa-proxy.yml`:

```
codescoring:
  url: ${CODESCORING_URL:https://codescoring.example.com}
  token: ${CODESCORING_TOKEN:}
  work-mode: ${WORK_MODE:strict_wait}
  osa-proxy-url: ${OSA_PROXY_URL:http://localhost:8080}

cache:
  judge:
    enabled: ${CACHE_ENABLED:false}
  redis:
    address: ${REDIS_ADDRESS:redis:6379}
    password: ${REDIS_PASSWORD:}
    db: ${REDIS_DB:0}

logging:
  level: ${LOG_LEVEL:info}
```

Для секретов используйте `.env`, а не literal-значения в `osa-proxy.yml`.

#Прокси для исходящих HTTP-запросов

OSA Proxy Go использует стандартные HTTP-клиенты Go. Они автоматически учитывают переменные окружения `HTTP_PROXY`, `HTTPS_PROXY` и `NO_PROXY`; также можно задавать lowercase-варианты `http_proxy`, `https_proxy`, `no_proxy`.

Пример `.env` для корпоративного proxy:

```
HTTP_PROXY=http://proxy.company.example:3128
HTTPS_PROXY=http://proxy.company.example:3128
NO_PROXY=localhost,127.0.0.1,::1,redis,codescoring.example.com,
.svc,.cluster.local
```

`NO_PROXY` должен включать адреса, к которым сервис должен ходить напрямую: локальные адреса, Redis, внутренние Kubernetes/Docker DNS-

имена, внутренние домены CodeScoring или package registry, если они не должны проходить через корпоративный proxy.

#Helm

Минимальный пример `values.yaml`:

```
image:
  repository: <registry-url>/osa-proxy
  tag: "<tag>"

service:
  type: ClusterIP
  port: 8080

probes:
  enabled: true
  path: /healthz

ingress:
  enabled: true
  className: nginx
  hosts:
    - host: osa-proxy.example.com
      paths:
        - path: /
          pathType: Prefix

config:
  create: true
  key: osa-proxy.yml
  mountPath: /etc/osa-proxy/osa-proxy.yml
  content: |
    codescoring:
      url: https://codescoring.example.com
      token: "<token>"
      work-mode: strict_wait
      osa-proxy-url: https://osa-proxy.example.com
      block-on-codescoring-errors: true
      remove-blocked-versions: true
      block-status-code: 403

    npm:
      enabled: true
      repository:
        - name: npm
          registry: https://registry.npmjs.org
          scan-manifest: true
          scan-package: true
          work-mode: strict_wait
          url-encoded-config: true
```

```
    logging:
      level: info
```

После установки проверьте endpoints:

```
curl https://osa-proxy.example.com/healthz
curl https://osa-proxy.example.com/metrics
```

#Настройка сервиса

Конфигурация OSA Proxy Go задается в файле `osa-proxy.yml`. Пример ниже показывает типовую рабочую конфигурацию с несколькими экосистемами, настройками CodeScoring, HTTP-клиента, Redis-кэша и логирования.

#Пример конфигурации

```
codescoring:
  url: https://codescoring.example.com
  token: "<token>"
  work-mode: strict_wait
  osa-proxy-url: https://osa-proxy.example.com
  enable-status-line: true
  block-on-codescoring-errors: true
  remove-blocked-versions: true
  legacy-judge: false
  stage: proxy
  block-status-code: 403
  judge-concurrency: 10
  resilience:
    retry:
      max-attempts: 3
      wait-duration: 1s
      exponential-backoff-multiplier: 2
    circuit-breaker:
      failure-rate-threshold: 50
      minimum-number-of-calls: 10
      sliding-window-size: 20
      wait-duration-in-open-state: 30s
      permitted-number-of-calls-in-half-open-state: 5

http:
  server:
    read-timeout: 2m
    read-header-timeout: 5s
    idle-timeout: 120s
    shutdown-timeout: 10s
```

```
  client:
    connection-timeout: 10s
    response-timeout: 30s
    max-manifest-body-size: 200mb
    max-idle-conns: 100
    max-idle-conns-per-host: 10
    idle-conn-timeout: 90s

pypi:
  enabled: true
  repository:
    - name: pypi
      registry: https://pypi.org
      packages-registry: https://files.pythonhosted.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
    - name: pytorch-pypi
      registry: https://download.pytorch.org
      packages-registry: https://download.pytorch.org
      additional-packages-registries:
        download.pytorch.org: https://download.pytorch.org
        download-r2.pytorch.org: https://download-
r2.pytorch.org
        files.pythonhosted.org: https://files.pythonhosted.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

maven:
  enabled: true
  repository:
    - name: maven
      registry: https://repo1.maven.org/maven2
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

nuget:
  enabled: true
  repository:
    - name: nuget
      registry: https://api.nuget.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

npm:
```

```
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

composer:
  enabled: true
  repository:
    - name: composer
      registry: https://repo.packagist.org
      packages-registry: https://api.github.com
      additional-packages-registries:
        github.com: https://github.com
        gitlab.com: https://gitlab.com
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

ruby:
  enabled: true
  repository:
    - name: ruby
      registry: https://rubygems.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

go:
  enabled: true
  repository:
    - name: go
      registry: https://proxy.golang.org
      sumdb-registry: https://sum.golang.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

debian:
  enabled: true
  repository:
    - name: debian
      registry: https://deb.debian.org/debian
      distro: bookworm
      scan-package: true
      work-mode: strict_wait
```

```
      url-encoded-config: true

alpine:
  enabled: true
  repository:
    - name: alpine
      registry: https://dl-cdn.alpinelinux.org/alpine
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

rpm:
  enabled: true
  repository:
    - name: rpm
      registry: https://mirror.stream.centos.org/10-stream/
AppStream/x86_64/os
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true

docker:
  enabled: true
  repository:
    - name: docker
      registry: https://registry-1.docker.io
      auth-token-url: https://auth.docker.io
      work-mode: strict_wait

cache:
  judge:
    enabled: false
    ttl: 24h
    refresh-after: 30m
    proactive-refresh-enabled: false
    proactive-refresh-interval: 2h
    proactive-refresh-workers: 10
    key-prefix: "cs:judge:"
  redis:
    address: redis:6379
    password: ""
    db: 0

logging:
  level: info
```

#Секция codescoring

Параметр	Назначение
`url`	URL платформы CodeScoring.
`token`	Токен доступа к CodeScoring.
`work-mode`	Глобальный режим работы, если он не переопределен на уровне репозитория.
`osa-proxy-url`	Внешний URL OSA Proxy Go, который используется при формировании ссылок и ответов.
`enable-status-line`	Добавляет причину блокировки в HTTP/1.1 status line, если клиент ее отображает.
`block-on-codescoring-errors`	Блокирует загрузку при ошибках CodeScoring или ошибках сканирования.
`remove-blocked-versions`	Удаляет заблокированные версии из манифестов. Если `false`, версии остаются с пометкой о блокировке, где это поддерживается форматом.
`block-status-code`	HTTP-код для блокировки. По умолчанию используется `403`.
`judge-concurrency`	Количество параллельных запросов к Judge. Используется, чтобы ограничить нагрузку на Judge при проверке больших списков версий и фоновом обновлении кэша.
`resilience.retry`	Настройки повторных запросов к CodeScoring.
`resilience.circuit-breaker`	Настройки circuit breaker для временной деградации внешних вызовов.

#Секции пакетных менеджеров

Каждая экосистема содержит флаг `enabled` и список `repository`. Имя репозитория становится частью URL OSA Proxy Go:

```
npm:
  enabled: true
  repository:
    - name: company-npm
      registry: https://registry.npmjs.org
      scan-manifest: true
```

```
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Такой репозиторий будет доступен по адресу:

```
https://osa-proxy.example.com/company-npm/
```

Поля `scan-manifest` и `scan-package` включают проверку манифестов и скачиваемых артефактов. Поддержка этих режимов зависит от экосистемы; подробнее см. Поддерживаемые протоколы. Параметр `work-mode` на уровне репозитория переопределяет глобальный `codescoring.work-mode`.

#Особенности экосистем

Для `composer` и `pypi` доступны `packages-registry` и `additional-packages-registries`, если артефакты загружаются с отдельных хостов. Для `go` указывается `sumdb-registry`, если нужно проксировать SumDB. Для Docker используется `auth-token-url`.

#Кэш вердиктов

По умолчанию Redis-кэш выключен:

```
cache:
  judge:
    enabled: false
  redis:
    address: redis:6379
```

Чтобы включить кэширование:

```
cache:
  judge:
    enabled: true
    ttl: 24h
    refresh-after: 30m
    proactive-refresh-enabled: false
    proactive-refresh-interval: 2h
    proactive-refresh-workers: 10
    key-prefix: "cs:judge:"
  redis:
    address: redis:6379
    password: ""
    db: 0
```

#Логирование

Уровень логирования задается в `logging.level`. Поддерживаются значения `debug`, `info`, `warn` и `error`.

```
logging:
  level: info
```

#Справочник параметров

#Корневые секции

Параметр	Назначение
`pypi`	Настройки PyPI-репозиториев.
`maven`	Настройки Maven-репозиториев.
`nuget`	Настройки NuGet-репозиториев.
`npm`	Настройки npm-репозиториев.
`composer`	Настройки Composer/Packagist-репозиториев.
`ruby`	Настройки RubyGems-репозиториев.
`go`	Настройки Go module proxy.
`debian`	Настройки Debian-репозиториев.
`alpine`	Настройки Alpine APK-репозиториев.
`rpm`	Настройки RPM/YUM/DNF-репозиториев.
`docker`	Настройки Docker Registry API v2.
`codescoring`	Подключение к CodeScoring и поведение проверок.
`http`	Таймауты и лимиты HTTP-сервера и HTTP-клиента.
`cache`	Redis-кэш вердиктов Judge.
`logging`	Уровень логирования сервиса.

#Общие параметры секций пакетных менеджеров

Параметр	Где доступен	Назначение
`enabled`	Все пакетные менеджеры	Включает регистрацию маршрутов для экосистемы. Если `false`, репозитории этой секции не обслуживаются.
`repository`	Все пакетные менеджеры	Список upstream-репозиториев для экосистемы.
`repository[*].name`	Все пакетные менеджеры	Имя репозитория. Для non-Docker экосистем становится первым сегментом URL: `/{name}/...`. Должно быть уникальным среди включенных маршрутов.
`repository[*].registry`	Все пакетные менеджеры	URL upstream-реестра, куда OSA Proxy Go проксирует запросы.
`repository[*].work-mode`	Все пакетные менеджеры	Режим работы для конкретного репозитория. Если пустой, используется `codescoring.work-mode`.
`repository[*].scan-manifest`	`npm`, `composer`, `maven`, `nuget`, `pypi`, `ruby`, `go`	Включает проверку и модификацию манифестов/metadata.
`repository[*].scan-package`	Все, кроме `docker`	Включает проверку скачиваемых файлов пакетов. Для `docker` сканирование образов включено логикой Docker Registry proxy.
`repository[*].url-encoded-config`	Все, кроме `docker`	Включает поддержку URL-safe Base64-контекста в пути для сценариев через Nexus/JFrog и применения политик к конкретному repository context.
`repository[*].file-type-filter`	Все, кроме `docker`	Ограничивает, какие файлы отправляются на пакетное сканирование, по расширениям. Если параметр не задан или выключен, фильтрация не применяется.

#Специфичные параметры репозиториев

Параметр	Где доступен	Назначение
`packages-registry`	`pypi`, `composer`	Базовый URL отдельного хоста, с которого скачиваются файлы пакетов, если он отличается от metadata registry.
`additional-packages-registries`	`pypi`, `composer`	Карта дополнительных host -> registry для пакетов, которые публикуют артефакты на нескольких доменах. Нужна для корректной маршрутизации ссылок на внешние package hosts.
`sumdb-registry`	`go`	URL Go checksum database, например `https://sum.golang.org`, если SumDB-запросы должны проходить через OSA Proxy Go.
`distro`	`debian`, `alpine`	Имя дистрибутива или ветки репозитория, которое используется при обработке metadata и путей пакетов.
`auth-token-url`	`docker`	URL token service Docker Registry. Для Docker Hub обычно `https://auth.docker.io`; для JFrog/Nexus указывается соответствующий endpoint менеджера репозиториев.

#file-type-filter

Параметр	Назначение
`additional-allowed-extensions`	YAML-массив строк. Наличие этого поля включает фильтр: после этого проходят только расширения из встроенного preset и из `additional-allowed-extensions`, а остальные неизвестные расширения блокируются. Можно указывать с точкой или без точки; значения нормализуются к нижнему регистру и форме с точкой.
`scanned-extensions`	YAML-массив строк. Включает фильтр и заставляет handler рассматривать файлы с этими расширениями как package artifacts для сканирования и

Параметр	Назначение
	краткоживущего кэша результата сканирования. Можно указывать с точкой или без точки.

Фильтр работает только для репозиториев не-Docker экосистем. Если секция `file-type-filter` отсутствует или задана как `{}`, фильтр выключен: OSA Proxy Go работает как без фильтра, то есть запросы проходят по обычным правилам handler'a и совместимость с прежним поведением сохраняется. Чтобы включить фильтр, добавьте хотя бы одно из полей `additional-allowed-extensions` / `scanned-extensions`. Учитывается именно наличие ключа: например, `additional-allowed-extensions: []` включает фильтр, но не добавляет расширений сверх встроенного preset.

При включенном фильтре OSA Proxy Go:

- пропускает metadata/manifest-запросы без проверки расширения;
- извлекает имя файла из URL path, декодирует URL-encoded символы и сравнивает расширение без учета регистра;
- разрешает файл, если его расширение входит во встроенный preset экосистемы или в `additional-allowed-extensions`;
- разрешает sidecar-файлы checksums и подписи (`.metadata`, `.sha256`, `.sha512`, `.sha1`, `.asc`, `.md5`), только если базовый артефакт тоже разрешен;
- сразу блокирует все остальные package file-запросы до обращения к upstream и CodeScoring.

Встроенные presets:

Экосистема	Разрешенные расширения
`npm`	`.tgz`
`composer`	`.zip`, `.tar`, `.tgz`, `.tar.gz`, `.tar.bz2`, `.tar.xz`
`pypi`	`.whl`, `.tar.gz`, `.tar.bz2`, `.tar.xz`, `.zip`, `.egg`
`nuget`	`.nupkg`, `.snupkg`
`ruby`	`.gem`
`go`	`.zip`
`alpine`	`.apk`
`rpm`	`.rpm`, `.drpm`

Экосистема	Разрешенные расширения
debian	.deb, .udeb, .dsc, .orig.tar.gz, .orig.tar.xz, .orig.tar.bz2, .debian.tar.gz, .debian.tar.xz, .debian.tar.bz2, .diff.gz
maven	.pom, .jar, .war, .ear, .rar, .dar, .zip, .tar.gz, .aar, .apk, .aab, .nar, .hpi, .jpi, .kar, .eba, .sar, .par, .car, .mar, .har, .obr, .module

Для Debian также разрешаются source tarballs вида `.orig-*.tar.gz`, `.orig-*.tar.xz` и `.orig-*.tar.bz2`.

`additional-allowed-extensions` расширяет только allow-list фильтра. Само наличие этого поля переводит репозиторий в режим allow-list: OSA Proxy Go разрешает встроенные расширения экосистемы и расширения из `additional-allowed-extensions`, а все остальные неизвестные package file-расширения блокирует. Параметр нужен, когда в репозитории есть допустимые файлы с нестандартными расширениями и их не нужно блокировать самим фильтром. Он не заставляет handler отправлять такие файлы на package scan: если стандартная стратегия экосистемы не считает расширение сканируемым, запрос пройдет дальше как обычный passthrough. Чтобы новый тип файла также участвовал в package scan, добавьте это расширение в `scanned-extensions`.

`scanned-extensions` используется для второго поведения: файлы с этими расширениями считаются сканируемыми package artifacts, даже если стандартная стратегия экосистемы их не распознает. Для таких расширений включается кэш результата сканирования на короткое время, чтобы родственные файлы с одной базой имени могли использовать один вердикт. Например, для Maven можно указать `scanned-extensions: [.jar, .pom]`, чтобы `demo-1.0.0.jar` и `demo-1.0.0.pom` группировались по базе `demo-1.0.0`.

Пример:

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-package: true
      file-type-filter:
        additional-allowed-extensions: [tgz, license]
        scanned-extensions: [tgz]
```

В этом примере `.tgz` разрешается preset'ом npm и участвует в package scan, а `.license` дополнительно разрешается фильтром, но не становится сканируемым артефактом.

#Пример поведения для npm

Без секции `file-type-filter` фильтр выключен. Npm handler работает по стандартной логике: package tarball `left-pad-1.0.0.tgz` отправляется на package scan, а остальные запросы обрабатываются как metadata или passthrough в зависимости от маршрута.

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-package: true
```

Пустая секция также оставляет фильтр выключенным:

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-package: true
      file-type-filter: {}
```

Чтобы включить фильтр без добавления новых расширений, можно задать пустой список. Тогда для npm разрешены только встроенный preset `.tgz` и sidecar-файлы к разрешенным артефактам. Запрос к `left-pad-1.0.0.tgz` пройдет и будет проверен, а запрос к `left-pad-1.0.0.exe` будет заблокирован до upstream и CodeScoring.

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-package: true
      file-type-filter:
        additional-allowed-extensions: []
```

Если нужно разрешить нестандартный файл, но не отправлять его на package scan, добавьте расширение только в `additional-allowed-extensions`:

```
npm:
  enabled: true
```

```
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-package: true
      file-type-filter:
        additional-allowed-extensions: [license]
```

В такой конфигурации `.tgz` будет сканироваться как npm package, `.license` пройдет фильтр как допустимый файл, а `.exe` будет заблокирован фильтром.

#codescoring

Параметр	Значение по умолчанию	Назначение
`url`	Обязательный параметр	URL платформы CodeScoring.
`token`	Обязательный параметр	Токен доступа к CodeScoring API.
`work-mode`	Не задан	Глобальный режим работы: `warmup`, `spectator`, `moderate`, `strict`, `strict_wait`.
`osa-proxy-url`	Не задан	Внешний URL OSA Proxy Go. Используется при генерации ссылок и подмене URL в ответах.
`enable-status-line`	`false`	Добавляет причину блокировки в HTTP/1.1 status line. Не влияет на HTTP/2 и HTTP/3; Docker-клиенты читают JSON body.
`block-on-codescoring-errors`	`false`	Блокирует скачивание, если CodeScoring вернул ошибку или пакет не удалось проверить.
`remove-blocked-versions`	`true`	Удаляет заблокированные версии из манифестов. При `false` версии остаются в metadata с пометкой о блокировке, если формат это поддерживает.
`legacy-judge`	`false`	Включает совместимость с версиями Judge до `2026.20.0`. Используйте только для инсталляций CodeScoring со старой версией сервиса Judge.

Параметр	Значение по умолчанию	Назначение
`stage`	Не задан	Значение stage/context, передаваемое в проверки CodeScoring. Обычно используется `proxy`.
`block-status-code`	`403`	HTTP-код ответа при блокировке пакета.
`judge-concurrency`	`10`	Ограничивает количество параллельных обращений к Judge. Чем ниже значение, тем меньше одновременных запросов OSA Proxy Go отправляет в Judge при проверке больших списков версий и фоновом обновлении кэша. Версии отправляются чанками до `900` PURL, поэтому параметр становится заметен, когда один пакет содержит больше `900` версий и проверка разбивается на несколько запросов.
`resilience`	См. ниже	Настройки устойчивости запросов к CodeScoring.

#codescoring.resilience.retry

Параметр	Значение по умолчанию	Назначение
`max-attempts`	`3`	Максимальное количество попыток запроса.
`wait-duration`	`1s`	Пауза между попытками.
`exponential-backoff-multiplier`	`2`	Множитель exponential backoff для увеличения паузы между повторами.

#codescoring.resilience.circuit-breaker

Параметр	Значение по умолчанию	Назначение
failure-rate-threshold	50	Процент ошибок, после которого circuit breaker открывается.
minimum-number-of-calls	10	Минимальное число вызовов для расчета error rate.
sliding-window-size	20	Размер окна, по которому считается статистика ошибок.
wait-duration-in-open-state	30s	Время ожидания перед переходом из open в half-open.
permitted-number-of-calls-in-half-open-state	5	Количество пробных запросов в half-open состоянии.

#http.server

Параметр	Значение по умолчанию	Назначение
read-timeout	2m	Максимальное время чтения всего входящего запроса.
read-header-timeout	5s	Максимальное время чтения HTTP-заголовков.
idle-timeout	120s	Время удержания idle keep-alive соединения.
shutdown-timeout	10s	Таймаут graceful shutdown.

#http.client

Параметр	Значение по умолчанию	Назначение
`connection-timeout`	`10s`	Таймаут установки соединения с upstream-реестрами и CodeScoring.
`response-timeout`	`30s`	Таймаут ожидания ответа.
`max-manifest-body-size`	`200mb`	Максимальный размер тела manifest/metadata, которое сервис готов обрабатывать. Поддерживаются значения вроде `200mb`.
`max-idle-conns`	`100`	Максимальное количество idle HTTP-соединений.
`max-idle-conns-per-host`	`10`	Максимальное количество idle HTTP-соединений на один host.
`idle-conn-timeout`	`90s`	Время жизни idle-соединения в HTTP-клиенте.

#cache.judge

Параметр	Значение по умолчанию	Назначение
`enabled`	`false`	Включает Redis-кэш результатов проверки Judge.
`ttl`	`24h`	Время жизни записи кэша.
`refresh-after`	`30m`	Возраст записи, после которого ее можно обновлять в фоне.
`proactive-refresh-enabled`	`false`	Включает фоновое обновление устаревающих записей.
`proactive-refresh-interval`	`2h`	Период запуска фонового обновления.
	`10`	

Параметр	Значение по умолчанию	Назначение
`proactive-refresh-workers`		Количество workers для фонового обновления.
`key-prefix`	Не задан	Префикс Redis-ключей, например `cs:judge:`.

#cache.redis

Параметр	Назначение
`address`	Адрес Redis в формате `host:port`.
`password`	Пароль Redis.
`db`	Номер Redis database.

#logging

Параметр	Значение по умолчанию	Назначение
`level`	`info`	Уровень логирования: `debug`, `info`, `warn`, `warning`, `error`. Неизвестное значение трактуется как `info`.

#Настройка Redis и кэширования

OSA Proxy Go поддерживает Redis-кэш вердиктов Judge, чтобы ускорять повторные запросы и снижать нагрузку на CodeScoring. Кэш выключен по умолчанию.

```
cache:
  judge:
    enabled: true
    ttl: 24h
    refresh-after: 30m
    proactive-refresh-enabled: false
    proactive-refresh-interval: 2h
    proactive-refresh-workers: 10
    key-prefix: "cs:judge:"
  redis:
```

```
    address: redis:6379
    password: ""
    db: 0
```

#Параметры

Параметр	Назначение
`cache.judge.enabled`	Включает Redis-кэш результатов проверки Judge.
`cache.judge.ttl`	Время жизни записи кэша. По умолчанию `24h`.
`cache.judge.refresh-after`	Возраст записи, после которого ее можно обновлять в фоне. По умолчанию `30m`.
`cache.judge.proactive-refresh-enabled`	Включает периодическое фоновое обновление устаревающих записей. По умолчанию `false`.
`cache.judge.proactive-refresh-interval`	Период фонового обновления. По умолчанию `2h`.
`cache.judge.proactive-refresh-workers`	Количество workers для фонового обновления. По умолчанию `10`.
`cache.judge.key-prefix`	Префикс Redis-ключей.
`cache.redis.address`	Адрес Redis в формате `host:port`.
`cache.redis.password`	Пароль Redis.
`cache.redis.db`	Номер базы Redis.

::::note TTL и фоновое обновление

Фоновое обновление не продлевает TTL записи само по себе. TTL продлевается при чтении данных из кэша реальными запросами, поэтому редко используемые записи со временем удаляются из Redis. ::::

#Управление кэшем

Служебный API доступен через Swagger UI:

```
https://osa-proxy.example.com/api/swagger/
```

Основные операции:

- `DELETE /api/cache/purls` — удалить конкретные PURL из кэша вердиктов;
- `DELETE /api/cache/packages/{packageType}` — удалить записи по типу пакета, имени пакета или repository context.

#Поддерживаемые протоколы

Раздел описывает, какие ресурсы OSA Proxy Go проверяет и какие ответы может модифицировать для каждой экосистемы.

#Сводная таблица

Экосистема	Сканирование манифестов	Сканирование пакетов	Модификация ответов
Maven	Да	Да	Удаление заблокированных версий из `maven-metadata.xml`, обновление `latest` и `release`.
npm	Да	Да	Удаление заблокированных версий из metadata, обновление `dist-tags` и ссылок на tarball.
PyPI	Да	Да	Удаление ссылок на заблокированные версии из Simple API, переписывание URL загрузки через прокси.
NuGet	Да	Да	Модификация service index и registration metadata, удаление заблокированных версий.
Go modules	Да	Да	Удаление заблокированных

Экосистема	Сканирование манифестов	Сканирование пакетов	Модификация ответов
			версий из `@v/list`, проксирование module zip и SumDB.
Composer	Да	Да	Модификация metadata Packagist/ Composer и переписывание dist URL через прокси.
RubyGems	Да	Да	Проверка metadata RubyGems и скачиваемых `.gem`-пакетов.
Debian	Нет	Да	Системные индексы Packages не модифицируются.
Alpine (APK)	Нет	Да	Индексы `APKINDEX` не модифицируются.
RPM	Нет	Да	Metadata `repodata` не модифицируется.
Docker	Да	Нет	Проверка image manifest; manifest list используется для определения image manifest и не отправляется на проверку как отдельный компонент. Blob/ layer-запросы проксируются без отдельной проверки слоев. Для нескольких Docker-репозиториев используются поддомены.

#Параметры scan-manifest и scan-package

`scan-manifest` включает проверку и модификацию metadata, из которой пакетный менеджер выбирает доступные версии. При срабатывании блокирующей политики небезопасные версии удаляются из ответа или помечаются как заблокированные, если формат это поддерживает.

`scan-package` включает проверку скачиваемого артефакта: архива, бинарного пакета, module zip, `.gem`, `.deb`, `.apk` или `.rpm`. Если политика блокирует компонент, скачивание прерывается с HTTP-кодом из `codescoring.block-status-code`.

Для Debian, Alpine и RPM используется только `scan-package`: системные индексы не изменяются, поэтому пакетный менеджер может видеть версию в индексе, но скачивание конкретного пакета будет заблокировано при нарушении политики.

Для Docker параметры `scan-manifest` и `scan-package` не используются в конфигурации репозитория. OSA Proxy Go проверяет Docker image manifest. Manifest list используется для определения image manifest и не отправляется на проверку как отдельный компонент; blob/layer-запросы проксируются в registry.

#Maven

- Metadata: `maven-metadata.xml`.
- Пакеты: `.jar`, `.war`, `.ear` и другие Maven-артефакты.
- При модификации metadata заблокированные версии удаляются из списка, а поля `latest` и `release` обновляются на последнюю разрешенную версию.

#npm

- Metadata: JSON-описание пакета.
- Пакеты: `.tgz`.
- Из metadata удаляются заблокированные версии, связанные записи `time`, а `dist-tags` пересчитываются на разрешенные версии.

#PyPI

- Metadata: страницы Simple API.
- Пакеты: `.whl`, `.tar.gz`, `.zip` и другие архивы Python-пакетов.
- Ссылки на заблокированные версии удаляются, URL загрузки переписываются так, чтобы скачивание проходило через OSA Proxy Go.

#NuGet

- Metadata: service index и registration index.
- Пакеты: `.nupkg`.

- Для клиента используется маршрут `/nuget-api/v3/index.json`; metadata переписывается на URL OSA Proxy Go.

#Go modules

- Metadata: список версий `@v/list`.
- Пакеты: module `.zip`.
- Заблокированные версии удаляются из списка версий. Для SumDB используется `sumdb-registry` и настройка `GOSUMDB`.

#Composer

- Metadata: Composer/Packagist metadata.
- Пакеты: dist-архивы `.zip`, `.tar`, `.tgz`, `.tar.gz`, `.tar.bz2`, `.tar.xz`.
- Dist URL переписываются на маршруты OSA Proxy Go. Для внешних dist-хостов используйте `packages-registry` и `additional-packages-registries`.

#RubyGems

- Metadata: индексы RubyGems.
- Пакеты: `.gem`.
- Проверяются metadata и скачиваемые gem-пакеты.

#Docker

- Metadata: Docker image manifest. Manifest list используется для определения image manifest.
- Пакеты: отдельного `scan-package` нет; blob/layer-запросы проксируются дальше в registry.
- Docker использует стандартные endpoints `/v2/...` и `/token`, поэтому несколько Docker-репозиториев разделяются по поддоменам, а не по первому сегменту пути.

#Диагностика блокировки

Если пакетный менеджер не показывает причину блокировки, проверьте соответствующий metadata endpoint напрямую:

```
curl https://osa-proxy.example.com/npm/lodash
curl https://osa-proxy.example.com/pypi/simple/requests/
curl https://osa-proxy.example.com/maven/org/apache/commons/
commons-lang3/maven-metadata.xml
curl https://osa-proxy.example.com/nuget/nuget-api/v3/
registration5-gz-semver2/newtonsoft.json/index.json
curl https://osa-proxy.example.com/go/github.com/gin-gonic/gin/
@v/list
```

#Настройка Base64 URL

Base64 URL используется, когда OSA Proxy Go должен получить контекст менеджера репозиториев из URL запроса. Это нужно для политик, привязанных к конкретному repository manager и имени репозитория, если upstream в `osa-proxy.yml` указывает напрямую на публичный реестр.

Чтобы включить такой режим для репозитория, задайте `url-encoded-config: true`:

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-manifest: true
      scan-package: true
      url-encoded-config: true
```

Base64-параметр размещается сразу после имени репозитория:

```
https://<osa-proxy-url>/<repository-name>/<base64-parameters>/
<path...>
```

JSON для кодирования содержит контекст репозитория:

```
{"repoManagerHost":"https://nexus.example.com","repoName":"npm-
proxy"}
```

Пример URL:

```
https://osa-proxy.example.com/npm/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLmV4YW1wbGUuY29tIi
wicmVwb05hbWUiOiJucG0tcHJveHkifQ/lodash
```

Для Docker этот механизм не используется в клиентском URL: Docker Registry API v2 работает через `/v2/...` и `GET /token`.

#Настройка Maven

```
maven:
  enabled: true
  repository:
    - name: maven
      registry: https://repo1.maven.org/maven2
```

```
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Пример `settings.xml`:

```
<settings>
  <mirrors>
    <mirror>
      <id>osa-proxy</id>
      <mirrorOf>*</mirrorOf>
      <url>https://osa-proxy.example.com/maven/</url>
    </mirror>
  </mirrors>
</settings>
```

#Настройка NPM

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Клиентский URL:

```
npm config set registry https://osa-proxy.example.com/npm/
npm view lodash version
```

Эквивалентная запись в `.npmrc`:

```
registry=https://osa-proxy.example.com/npm/
```

При миграции достаточно заменить `registry` в `.npmrc` с URL Nexus, Artifactory или `https://registry.npmjs.org` на `https://osa-proxy.example.com/npm/`. Учетные данные остаются в настройках npm.

#Настройка NuGet

```
nuget:
  enabled: true
  repository:
    - name: nuget
      registry: https://api.nuget.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Добавьте источник пакетов:

```
dotnet nuget add source https://osa-proxy.example.com/nuget/
nuget-api/v3/index.json --name osa-proxy
dotnet restore --source https://osa-proxy.example.com/nuget/
nuget-api/v3/index.json
```

Эквивалентный `NuGet.Config`:

```
<configuration>
  <packageSources>
    <clear />
    <add key="osa-proxy" value="https://osa-proxy.example.com/
nuget/nuget-api/v3/index.json" />
  </packageSources>
</configuration>
```

#Настройка PyPI

```
pypi:
  enabled: true
  repository:
    - name: pypi
      registry: https://pypi.org
      packages-registry: https://files.pythonhosted.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
    - name: pytorch-pypi
      registry: https://download.pytorch.org
      packages-registry: https://download.pytorch.org
      additional-packages-registries:
        download.pytorch.org: https://download.pytorch.org
```

```
        download-r2.pytorch.org: https://download-
r2.pytorch.org
        files.pythonhosted.org: https://files.pythonhosted.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Пример `pip.conf`:

```
[global]
index-url = https://osa-proxy.example.com/pypi/simple/
```

Постоянная настройка через `pip config`:

```
python -m pip config set global.index-url https://osa-
proxy.example.com/pypi/simple/
```

Разовые установки:

```
pip install --index-url https://osa-proxy.example.com/pypi/
simple/ requests
```

Для PyTorch используйте отдельный репозиторий:

```
pip install --index-url https://osa-proxy.example.com/pytorch-
pypi/whl/cu121 torch
```

#Настройка Go

```
go:
  enabled: true
  repository:
    - name: go
      registry: https://proxy.golang.org
      sumdb-registry: https://sum.golang.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Настройте Go toolchain. Для постоянной настройки используйте `go env -w`:

```
go env -w GOPROXY=https://osa-proxy.example.com/go
go env -w GOSUMDB="sum.golang.org https://osa-
proxy.example.com/go/sumdb/sum.golang.org"
go mod download
```

GOSUMDB нужен, если запросы к sum.golang.org также должны идти через OSA Proxy Go.

Для разового запуска можно задать переменные окружения:

```
GOPROXY=https://osa-proxy.example.com/go \
GOSUMDB="sum.golang.org https://osa-proxy.example.com/go/sumdb/
sum.golang.org" \
go get github.com/gin-gonic/gin@latest
```

#Настройка Composer

```
composer:
  enabled: true
  repository:
    - name: composer
      registry: https://repo.packagist.org
      packages-registry: https://api.github.com
      additional-packages-registries:
        github.com: https://github.com
        gitlab.com: https://gitlab.com
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Настройте репозиторий Composer в проекте:

```
composer config repositories.osa-proxy composer https://osa-
proxy.example.com/composer
composer config repo.packagist false
composer require monolog/monolog
```

Эквивалентная секция composer.json:

```
{
  "repositories": [
    {
      "packagist.org": false
    },
    {
```

```
      "type": "composer",
      "url": "https://osa-proxy.example.com/composer"
    }
  ]
}
```

additional-packages-registries нужен для dist-архивов, которые Composer metadata отдает с отдельных хостов, например GitHub или GitLab.

#Настройка RubyGems

```
ruby:
  enabled: true
  repository:
    - name: ruby
      registry: https://rubygems.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Для gem замените источник RubyGems:

```
gem sources --add https://osa-proxy.example.com/ruby/
gem sources --remove https://rubygems.org/
gem install rails
```

Для Bundler укажите источник в Gemfile:

```
source "https://osa-proxy.example.com/ruby/"

gem "rails"
```

#Настройка Debian

```
debian:
  enabled: true
  repository:
    - name: debian
      registry: https://deb.debian.org/debian
      distro: bookworm
      scan-package: true
```

```
      work-mode: strict_wait
      url-encoded-config: true
```

Пример `/etc/apt/sources.list.d/osa-proxy.sources`:

```
Types: deb
URIs: https://osa-proxy.example.com/debian
Suites: bookworm
Components: main
Signed-By: /usr/share/keyrings/debian-archive-keyring.gpg
```

```
apt update
apt install curl
```

#Настройка Docker

```
docker:
  enabled: true
  repository:
    - name: docker
      registry: https://registry-1.docker.io
      auth-token-url: https://auth.docker.io
      work-mode: strict_wait
```

Docker использует стандартные endpoints Registry API v2. Имя репозитория из конфигурации не добавляется в путь клиента.

```
docker pull osa-proxy.example.com/library/alpine:latest
```

Для Docker Hub можно настроить OSA Proxy Go как registry mirror в `/etc/docker/daemon.json`:

```
{
  "registry-mirrors": ["https://osa-proxy.example.com"]
}
```

После изменения перезапустите Docker daemon.

Если включено несколько Docker-репозиториев, используйте схему с поддоменами, где поддомен соответствует `repository[*].name`:

```
docker pull docker.osa-proxy.example.com/library/alpine:latest
```

Это требуется из-за особенностей Docker Registry API v2: клиент всегда обращается к фиксированным путям `/v2/...` и `/token`, поэтому имя

репозитория OSA Proxy Go нельзя добавить первым сегментом пути, как для npm, Maven или PyPI. Когда настроен один Docker-репозиторий, OSA Proxy Go может обслуживать его через основной host. Когда Docker-репозиториев несколько, сервис определяет нужную конфигурацию по host запроса.

Например, для конфигурации:

```
docker:
  enabled: true
  repository:
    - name: dockerhub
      registry: https://registry-1.docker.io
      auth-token-url: https://auth.docker.io
    - name: company
      registry: https://registry.company.example
      auth-token-url: https://registry.company.example
```

клиенты должны использовать разные hostnames:

```
docker pull dockerhub.osa-proxy.example.com/library/
alpine:latest
docker pull company.osa-proxy.example.com/team/image:latest
```

Для такой схемы настройте DNS wildcard или отдельные DNS-записи для поддоменов, TLS-сертификат с поддержкой этих имен и reverse proxy/load balancer, который передает запросы на OSA Proxy Go с исходным `Host`.

#Настройка Alpine (APK)

```
alpine:
  enabled: true
  repository:
    - name: alpine
      registry: https://dl-cdn.alpinelinux.org/alpine
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Пример `/etc/apk/repositories`:

```
https://osa-proxy.example.com/alpine/v3.20/main
https://osa-proxy.example.com/alpine/v3.20/community
```

```
apk update
apk add curl
```

#Настройка RPM

```
rpm:
  enabled: true
  repository:
    - name: rpm
      registry: https://mirror.stream.centos.org/10-stream/
AppStream/x86_64/os
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Пример `.repo` файла:

```
[osa-proxy]
name=OSA Proxy RPM
baseurl=https://osa-proxy.example.com/rpm/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
```

```
dnf makecache
dnf install curl
```

#Миграция на OSA Proxy Go

Этот раздел описывает перенос конфигурации со старой Java/Spring-версии OSA Proxy на OSA Proxy Go. Legacy-конфигурация использовала файл `application.yml`; Go-версия использует `osa-proxy.yml`.

#Основные изменения

Было в legacy OSA Proxy	Стало в OSA Proxy Go
`application.yml`	`osa-proxy.yml`
`codescoring.host`	`codescoring.url`
`codescoring.proxy-manager-host`	`codescoring.osa-proxy-url`

Было в legacy OSA Proxy	Стало в OSA Proxy Go
Spring Boot logging через `logging.level.ru.codescoring`	`logging.level: debug/info/warn/error`
Actuator endpoints	`/healthz`, `/metrics`, `/api/v3/api-docs`, `/api/swagger/`
Глобальный `codescoring.work-mode`	`codescoring.work-mode` плюс переопределение `repository[*].work-mode`

Поля `codescoring.enable-status-line`, `codescoring.block-status-code`, `codescoring.block-on-codescoring-errors` и `codescoring.remove-blocked-versions` сохраняют смысл, но должны быть перенесены в новую структуру `osa-proxy.yml`.

#Новые возможности

- Поддержка Composer и RubyGems.
- Переопределение режима работы для отдельного репозитория через `repository[*].work-mode`.
- Фильтрация типов файлов на уровне репозитория через `repository[*].file-type-filter`.
- Явная настройка HTTP-сервера и HTTP-клиента в `http.server` и `http.client` вместо Spring Boot / WebFlux properties.

#Пример миграции npm

Legacy-конфигурация:

```
codescoring:
  host: https://codescoring.example.com
  token: "<token>"
  work-mode: strict_wait
  proxy-manager-host: https://osa-proxy.example.com
  block-on-codescoring-errors: true
  remove-blocked-versions: true

npm:
  enabled: true
  repository:
    - name: internet-npm
      scan-package: true
      scan-manifest: true
      url-encoded-config: true
      registry: https://registry.npmjs.org
```

Конфигурация для OSA Proxy Go:

```
codescoring:
  url: https://codescoring.example.com
  token: "<token>"
  work-mode: strict_wait
  osa-proxy-url: https://osa-proxy.example.com
  block-on-codescoring-errors: true
  remove-blocked-versions: true
  block-status-code: 403

npm:
  enabled: true
  repository:
    - name: internet-npm
      scan-package: true
      scan-manifest: true
      work-mode: strict_wait
      url-encoded-config: true
      registry: https://registry.npmjs.org

logging:
  level: info
```

После миграции npm-клиент использует тот же route name:

```
npm config set registry https://osa-proxy.example.com/internet-
npm/
npm view lodash version
```

#Checklist миграции

1. Создайте новый файл `osa-proxy.yml`.
2. Перенесите URL платформы CodeScoring из `codescoring.host` в `codescoring.url`.
3. Перенесите токен в `codescoring.token`.
4. Перенесите внешний URL прокси из `codescoring.proxy-manager-host` в `codescoring.osa-proxy-url`.
5. Выберите глобальный `codescoring.work-mode` и при необходимости задайте `repository[*].work-mode` для отдельных репозиториев.
6. Перенесите секции пакетных менеджеров и проверьте `name`, `registry`, `scan-manifest`, `scan-package`, `url-encoded-config`.
7. Включайте `cache.judge.enabled` только если доступен Redis и нужно кэширование вердиктов.
8. Обновите конфигурацию package managers, чтобы они использовали URL OSA Proxy Go.
9. Проверьте `GET /healthz`, `GET /metrics` и один тестовый запрос к каждому включенному типу репозитория.

#Что удалить из старой конфигурации

Spring Boot параметры, JVM options и actuator-настройки не являются частью `osa-proxy.yml`. Для Go-версии не используются endpoints `/actuator/metrics` и `/actuator/prometheus`; метрики доступны напрямую по `/metrics`.

#Метрики

OSA Proxy Go отдает метрики Prometheus по адресу:

```
GET /metrics
```

Например:

```
curl http://localhost:8080/metrics
```

В Go-версии нет Spring Boot actuator endpoints. Для сбора метрик используйте `/metrics`, а не `/actuator/metrics` или `/actuator/prometheus`.

#Пример Prometheus scrape config

```
scrape_configs:
  - job_name: osa-proxy-go
    metrics_path: /metrics
    static_configs:
      - targets:
          - osa-proxy.example.com:8080
```

#Проверка после установки

1. Убедитесь, что сервис отвечает на healthcheck:

   ```
   curl http://localhost:8080/healthz
   ```

2. Убедитесь, что endpoint метрик возвращает данные в формате Prometheus:

   ```
   curl http://localhost:8080/metrics
   ```

3. Настройте сбор в Prometheus или ServiceMonitor Helm-чарта, если сервис развернут в Kubernetes.

#CodeScoring.SCA

#Общее описание

Модуль **CodeScoring.SCA** решает задачи инвентаризации ПО и поиска уязвимостей в компонентах с открытым исходным кодом. Основные функциональные возможности модуля включают:

- **Проверку на разных этапах разработки** с возможностью проверки проектов в системе контроля версий;
- **Интеграцию проверок в CI-конвейер** с блокирующими политиками безопасности с помощью консольного агента Johnny;
- **Построение SBOM** и визуализацию графа зависимостей;
- **Анализ на разных уровнях**: проверка манифестов, разрешение транзитивных зависимостей, перехват сборки для языков C и C++, сканирование Docker-образов;
- **Отслеживание истории сканирования** с возможностью выгрузки результатов для отчетности;
- Анализ достижимости уязвимостей.

#Настройка и запуск анализа

#Настройка анализа

Во время запуска анализа в платформе можно выбрать параметры для отдельных проектов.

Управление происходит на странице проекта в разделе `Настройки -> Проекты`.

- **Сканирование с хэшами** - сканирует проект с использованием хэш-сумм файлов, для поиска прямого включения зависимостей;
- **Исключения путей анализа** - список директорий, которые будут игнорироваться при сканировании;
- **Отключить рекурсивное сканирование** - отключает обход директории в глубину при сканировании. Будут найдены манифесты только в корневом каталоге;
- **Активировать облачный резолв** - включить разрешение зависимостей в облаке. Внимание! Использование облачного резолва может дать неточные результаты и увеличить время анализа;
- **Исключить из анализа SCA** - исключить данный проект из SCA анализа;

#Ручной запуск анализа

Композиционный анализ (SCA) запускается автоматически сразу при добавлении проекта. Для ручного запуска анализа по проекту необходимо использовать кнопку **Запустить SCA** на странице проекта.

При этом можно выбрать отдельную ветку или тэг для анализа, которая будет учитываться в истории сканирований.

Также анализ можно запустить на все проекты и на каждый тип анализа (SCA, Quality, Authors) отдельно. Управление запуском общего анализа происходит в разделе `Настройки -> Режим работы`.

Важно

Для получения корректных результатов нужно запустить анализ последовательно для каждого модуля, предварительно дождавшись завершения предыдущего запуска. Порядок запуска:

1. Software Composition Analysis (SCA)
2. Authors Analysis
3. Quality Analysis

Прогресс выполнения анализа можно отслеживать по сообщениям в разделе `Настройки -> Аудит лог`.

Первый запуск анализа авторов может выполняться заметное время, так как происходит траверс всей истории репозитория. Последующие запуски будут разбирать только разницу в коммитах по обновлениям с последнего запуска.

#Анализ по расписанию

Помимо ручного запуска, можно настроить анализ отдельных проектов по расписанию. Управление происходит на странице проекта в разделе `Настройки -> Проекты`.

По умолчанию параметр **Расписание сканирования SCA** имеет значение **Выкл.**. Для активации анализа по расписанию необходимо выбрать **Вкл.** и указать время и дни недели.

Примечание: Время сканирования будет учитываться по UTC +3.

#Запуск анализа Версии

При необходимости, можно запустить анализ выбранной версии. Для этого необходимо воспользоваться дополнительным действие кнопки запуска сканирования **Сканировать другую версию**.

В появившемся модальном окне можно выбрать версию для сканирования или создать новую версию и запустить сканирование.

Важно

Создать новую версию возможно только, если у пользователя есть полномочия на редактирование проекта.

#Обзор зависимостей

#Просмотр списка зависимостей

Список просканированных open source завимостей можно посмотреть в подразделе SCA -> Зависимости. Таблица в данном разделе содержит **все** зависимости, которые проходили проверку за время работы SCA модуля, со следующей информацией:

- **Зависимость** – название зависимости (со ссылкой на его индивидуальную страницу);
- **Технология** – технология (язык программирования или инструмент сборки);
- **Лицензии** – идентификатор лицензии, указанный в пакетном индексе;
- **Авторы** – разработчик компонента, указанный в пакетном индексе;
- **Уязвимости** – количество найденных уязвимостей в зависимости;
- **Найдено** – тип определения зависимости: по манифесту, облачный резолв или по содержимому (когда код компонента включен в кодовую базу проекта);
- **Связь** - тип зависимости (прямая или транзитивная);
- **Окружение** - окружение разработки;
- **Родительские зависимости** - связанные вышестоящие зависимости;
- **Проект** - проект, в котором используется зависимость;
- **Максимальная версия исправления** - версия зависимости, на которую необходимо выполнить обновление, чтобы закрыть обнаруженные в настоящий момент модулем SCA уязвимости, при этом учитываются только зависимости с указанной версией исправления уязвимости;
- **Дата выпуска** - дата и время релиза зависимости.

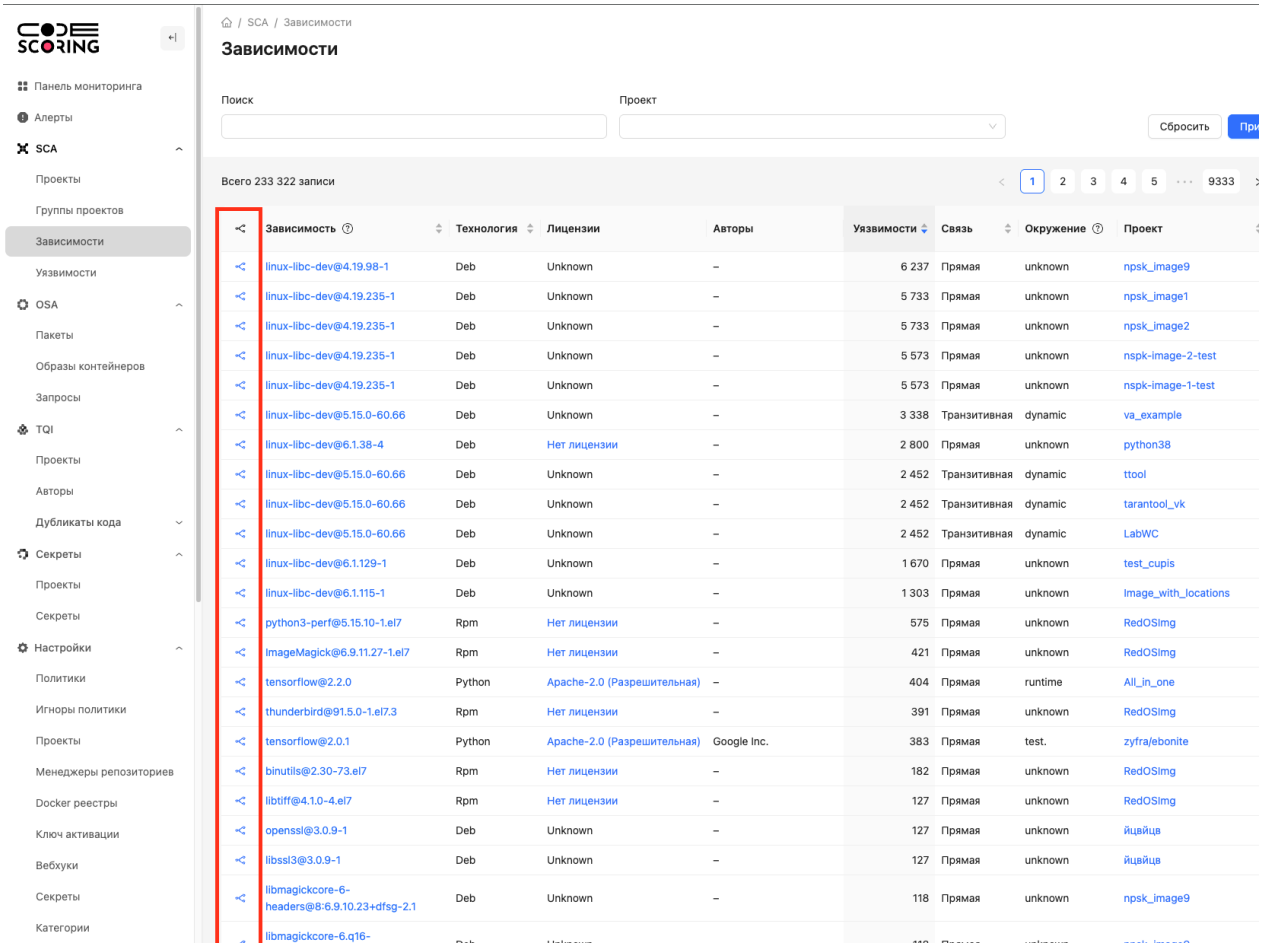
Таблицу с зависимостями можно отфильтровать по проекту, подразделению, категории проекта, группах проекта, технологии, лицензии, категории лицензии, как найдено, связи, родителям, окружению, временному периоду выпуска.

По нажатию на название зависимости осуществляется переход на его индивидуальную страницу, где отображается информация об его использовании в проектах и найденных уязвимостях.

#Работа с визуализацией графа зависимостей

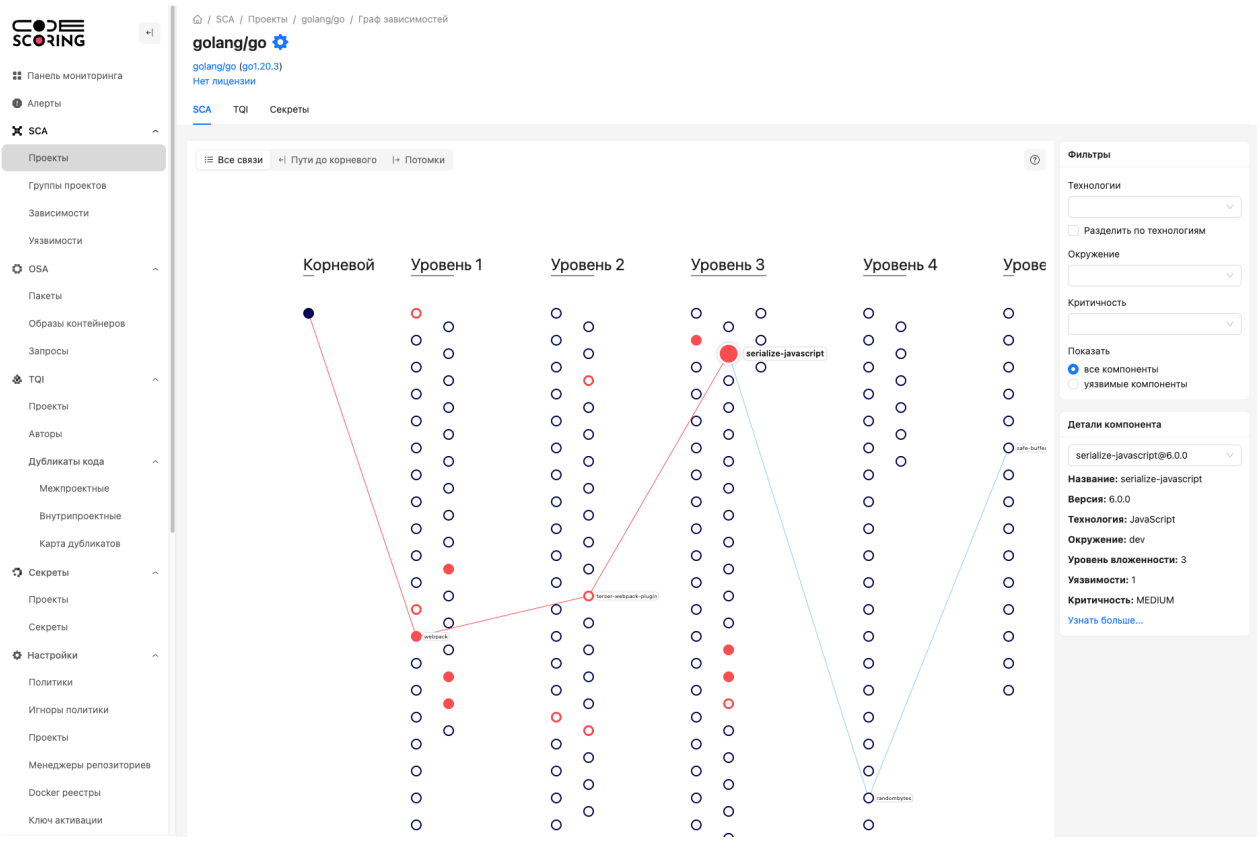
Open source зависимости программных проектов представляют собой граф — структуру, в которой отдельные компоненты выступают узлами, а связи между ними представлены в виде ребер.

Увидеть визуализацию графа зависимостей можно в разделе Зависимости или на странице проекта, нажав на соответствующую иконку в списке зависимостей.



На странице с интерактивной визуализацией представлены компоненты по уровням вложенности — от корневой зависимости до максимального уровня транзитивных зависимостей. По наведению курсора на объекты можно увидеть более подробную информацию о компоненте: версия, окружение, технология и количество уязвимостей.

Визуализация интерактивна и масштабируема. По нажатию на компоненту можно отследить путь ее попадания в проект. Компоненты с найденными уязвимостями обозначаются цветом.



Компоненты на полученном графе можно отфильтровать по следующим параметрам:

- технология;
- среда разработки;
- степень критичности уязвимости.

После выбора компоненты графа можно сконфигурировать отображаемые связи:

- направление (корень/потомки)
- уровень вложенности для потомков
- выбор конкретного пути до корня

#Работа с уязвимостями

#Просмотр списка уязвимостей

Список обнаруженных уязвимостей доступен в подразделе `SCA -> Уязвимости`. В этом разделе отображаются **все уязвимости**, выявленные модулями SCA и OSA за время их работы.

Таблица уязвимостей содержит следующую информацию:

- **Уязвимость** – идентификатор уязвимости (например, CVE) со ссылкой на ее индивидуальную страницу;
- **Зависимость** – компонент, в котором обнаружена уязвимость, с указанием версии;

- **Связь** — тип зависимости, в которой была обнаружена уязвимость (прямая или транзитивная);
- **Окружение** — среда использования зависимости (например, runtime, dev, main);
- **Проект** — проект, в котором зафиксировано использование уязвимой зависимости;
- **Статус** — статус триажа уязвимости;
- **CVSS 2** — оценка угрозы по шкале CVSS v2;
- **CVSS 3** — оценка угрозы по шкале CVSS v3;
- **CVSS 4** — оценка угрозы по шкале CVSS v4;
- **Эксплуатация** — текущее состояние эксплуатируемости уязвимости по оценке SSVC;
- **Автоматизируемо** — возможность автоматизировать эксплуатацию уязвимости по оценке SSVC;
- **Техническое влияние** — техническое влияние уязвимости по оценке SSVC;
- **EPSS** — вероятность экплуатации уязвимости по оценке EPSS;
- **CWE** — категории (Common Weakness Enumeration), к которым относится уязвимость;
- **Есть эксплойт** — признак наличия публично известного эксплойта;
- **Достижимо** — информация о достижимости уязвимости в контексте использования компонента;
- **Импакт** — тип потенциального воздействия уязвимости (например, XSS, DoS, RCE и др.);
- **Исправленная версия** — версия зависимости, в которой уязвимость устранена;
- **Найдено** — дата и время обнаружения уязвимости.

Для удобства анализа список уязвимостей можно отфильтровать по следующим параметрам:

- проект;
- подразделение;
- категория и группа проектов;
- временной период публикации уязвимости;
- дата обнаружения;
- рейтинг и уровень угрозы CVSS v2, CVSS v3 и CVSSv4;
- технология;
- окружение зависимости;
- тип связи зависимости (прямая или транзитивная);
- наличие эксплойта;
- достижимость;
- наличие исправления;
- наличие оценки SSVC;
- процент EPSS;
- классы CWE;
- импакт уязвимости;
- статус;
- обоснование;
- ответ.

Также доступен текстовый поиск по идентификатору уязвимости и связанным данным.

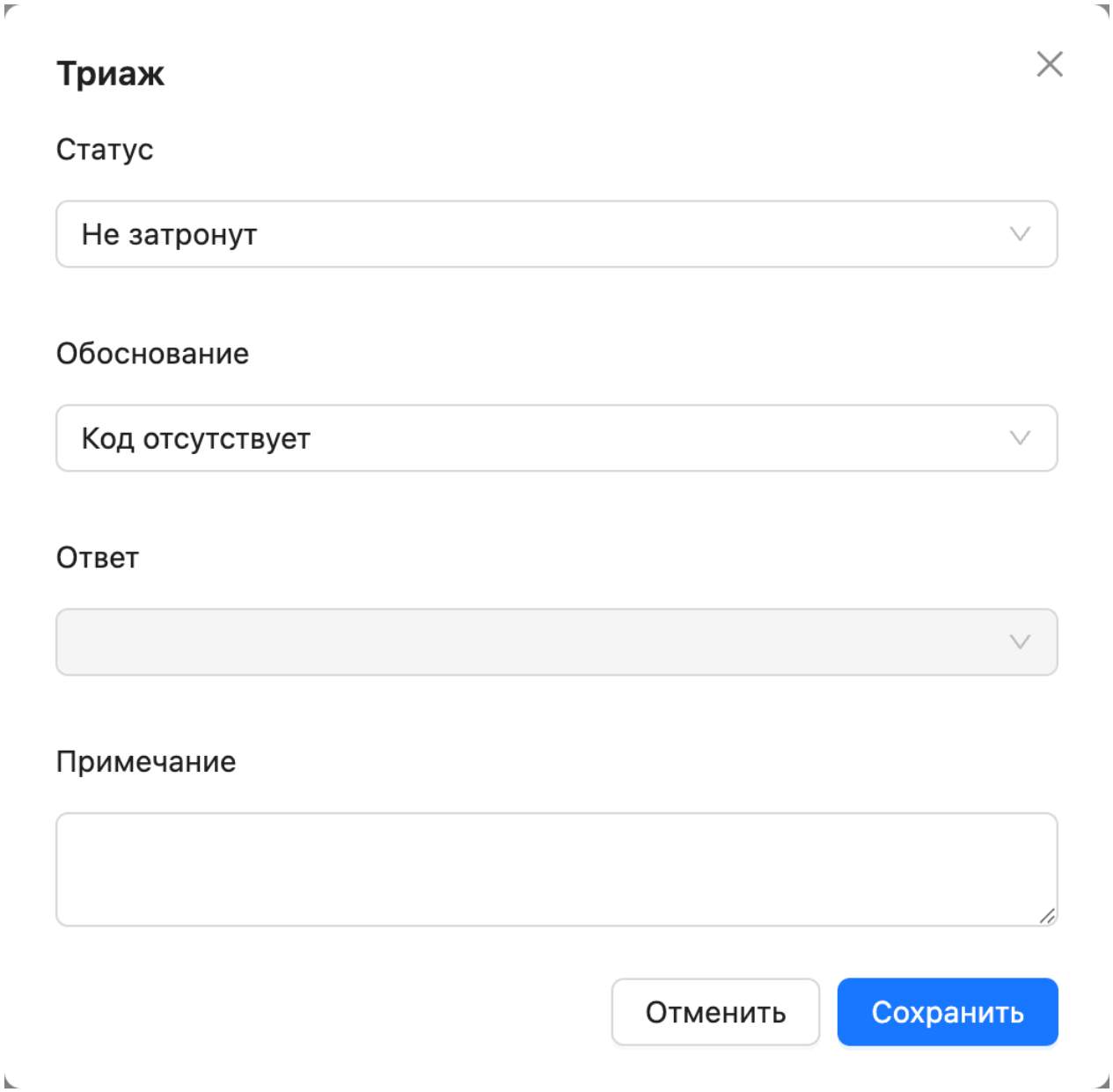
#Триаж

Триаж позволяет вручную задать статус уязвимости, отражающий её реальное влияние на проект и план действий команды. Функция реализована в соответствии со стандартом VEX (Vulnerability Exploitability eXchange).

Чтобы выполнить триаж, выберите одну или несколько уязвимостей в списке и нажмите кнопку **Триаж** над таблицей. Откроется модальное окно со следующими полями:

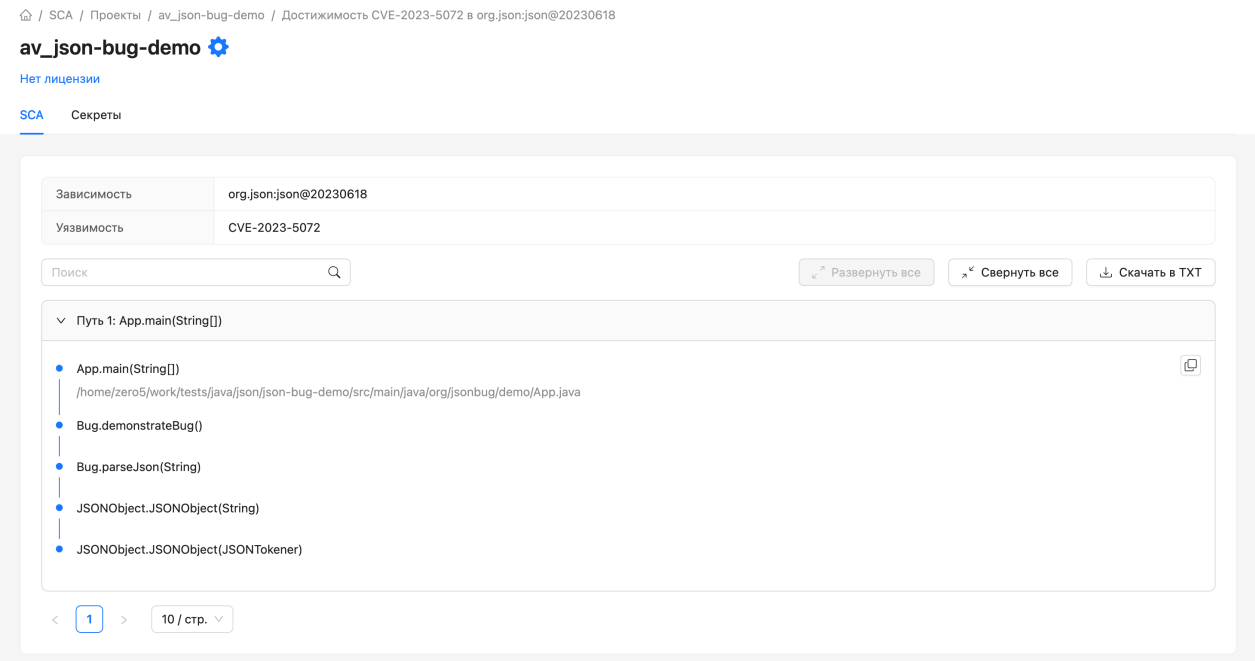
- **Статус** — статус уязвимости, отражающий её применимость к проекту (например, *Не затронут*, *Активен*, *Подтвержден*, *Ложноположительный*, *Без статуса*);
- **Обоснование** — причина присвоения статуса *Не затронут* (например, *Код отсутствует*, *Код недостижим*);
- **Ответ** — запланированный или реализованный ответ на уязвимость (например, *Невозможно исправить*, *Исправление не планируется*);
- **Детали** — текстовое поле для дополнительных комментариев или контекста.

После сохранения присвоенный статус отображается в колонке **Статус** списка уязвимостей и доступен для фильтрации.



#Достижимость

Для достижимых уязвимостей есть возможность просмотра визуализации путей и выгрузки их текстового представления



#Страница уязвимости

Индивидуальная страница уязвимости предназначена для детального анализа конкретной уязвимости и всей связанной с ней информации в рамках платформы.

Дедупликация уязвимостей

Страница отображает **единую дедуплицированную уязвимость**, даже если она была обнаружена несколькими источниками данных (например, NVD, GitHub Advisories, БДУ ФСТЭК и др.). При этом пользователь может просмотреть оригинальные данные каждого источника отдельно.

#Общая информация об уязвимости

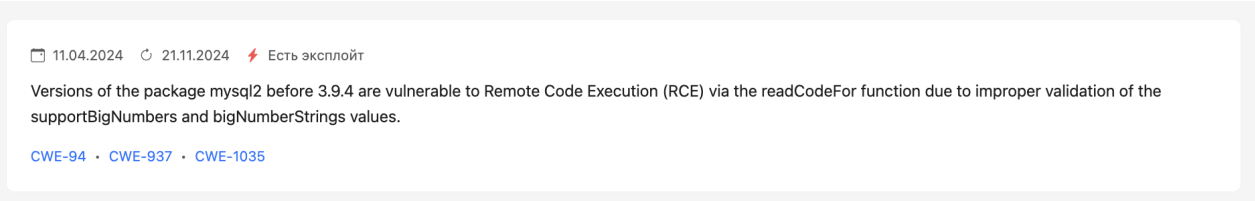
В верхней части страницы отображается сводная информация об уязвимости:

- идентификатор уязвимости (например, CVE);
- даты публикации, отзыва (если есть) и обновления в источнике данных;
- наличие публично известного эксплойта;
- является ли уязвимость протестной;
- использование уязвимости в ПО-вымогателе;
- краткое описание уязвимости;
- связанные категории CWE.

В качестве дат публикаций и отзыва отображаются самые ранние даты из всех источников. В качестве даты обновления - самая поздняя дата.

Идентификатор уязвимости и краткое описание отображаются из источников в порядке приоритета:

- CVE.ORG;
- GHSA;
- Kaspersky;
- BDU;
- Остальные источники в алфавитном порядке.

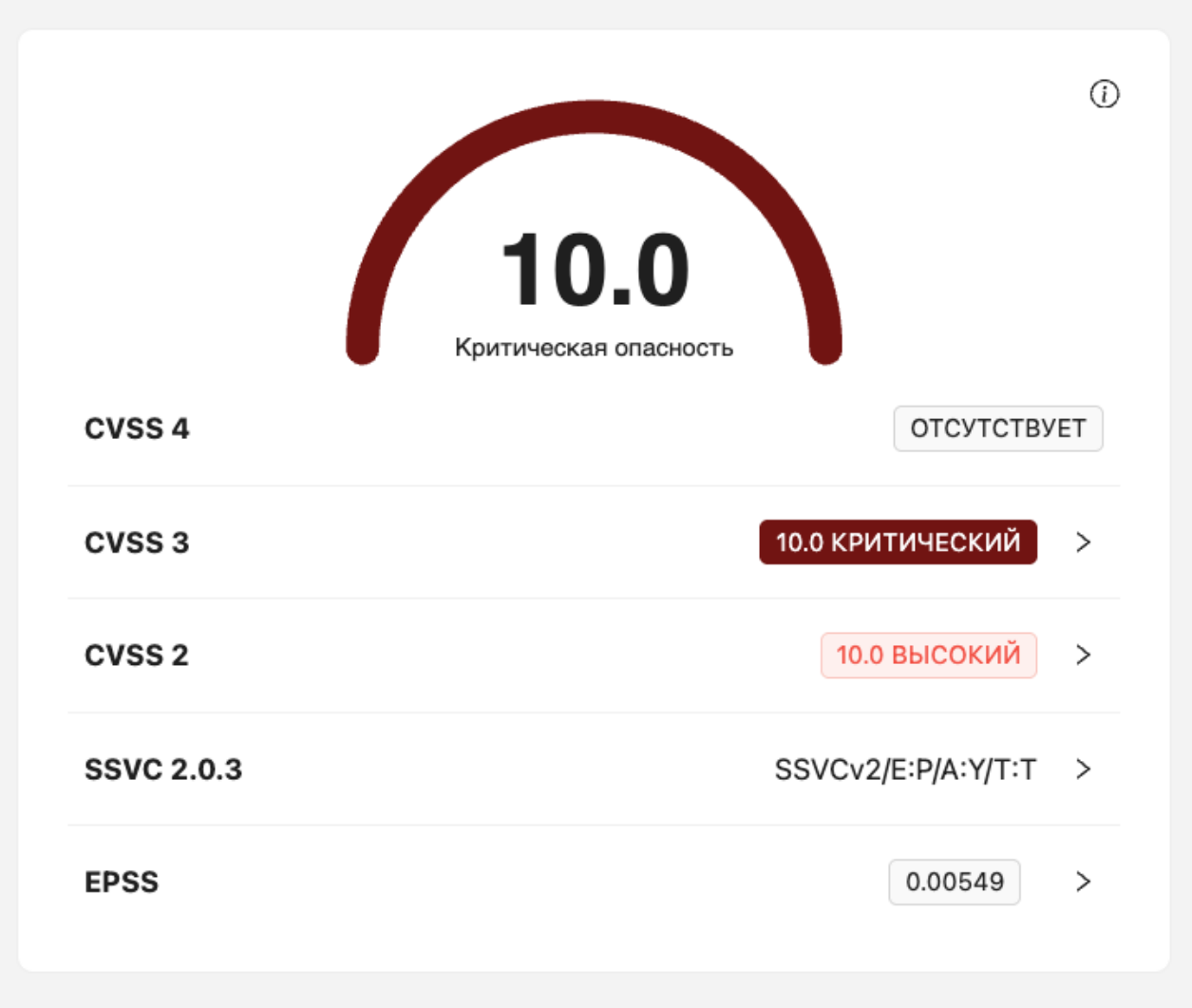


Справа отображается наивысшая оценка уровня угрозы для наиболее новой версии CVSS с учетом всех источников.

Оценка, уровень угрозы и остальные метрики отображаются из источника с наивысшей оценкой для данной версии CVSS.

Ниже также могут быть представлены:

- разделение по уровням CVSS с указанием версии и соответствующего уровня угрозы;
- вектор SSVC категоризации уязвимости;
- EPSS оценка вероятности эксплуатации;

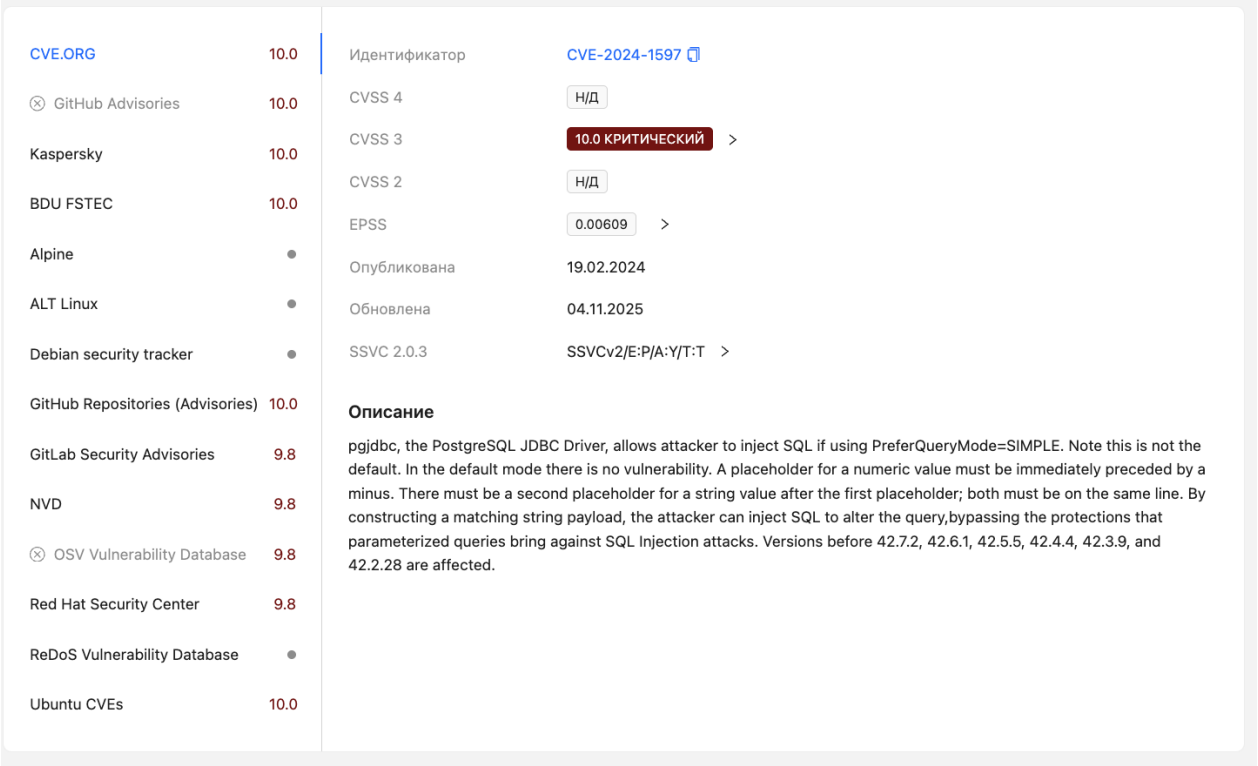


#Источники данных и оценки

Для уязвимости отображается список источников, в которых она была зафиксирована. Для каждого источника могут быть представлены:

- собственная оценка CVSS;
- версия CVSS;
- вектор SSVC категоризации уязвимости;
- EPSS оценка вероятности эксплуатации уязвимости;
- дополнительные атрибуты и метаданные источника.

Это позволяет сопоставлять данные из разных источников и учитывать расхождения в оценках при анализе рисков.

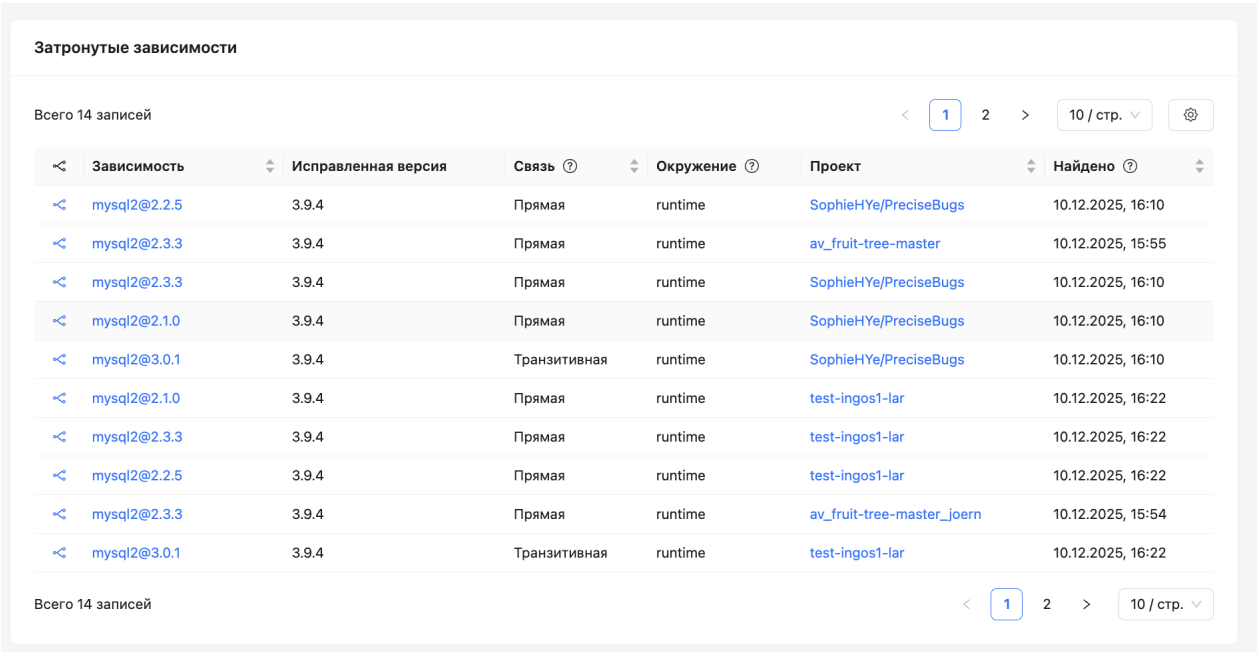


#Затронутые зависимости и образы

На странице отображаются списки затронутых компонентов:

- зависимости, обнаруженные в SCA-проектах;
- пакеты и образы, проверенные модулем CodeScoring.OSA.

Такое разделение упрощает анализ уязвимости в различных контекстах использования и помогает точнее оценить область её влияния.



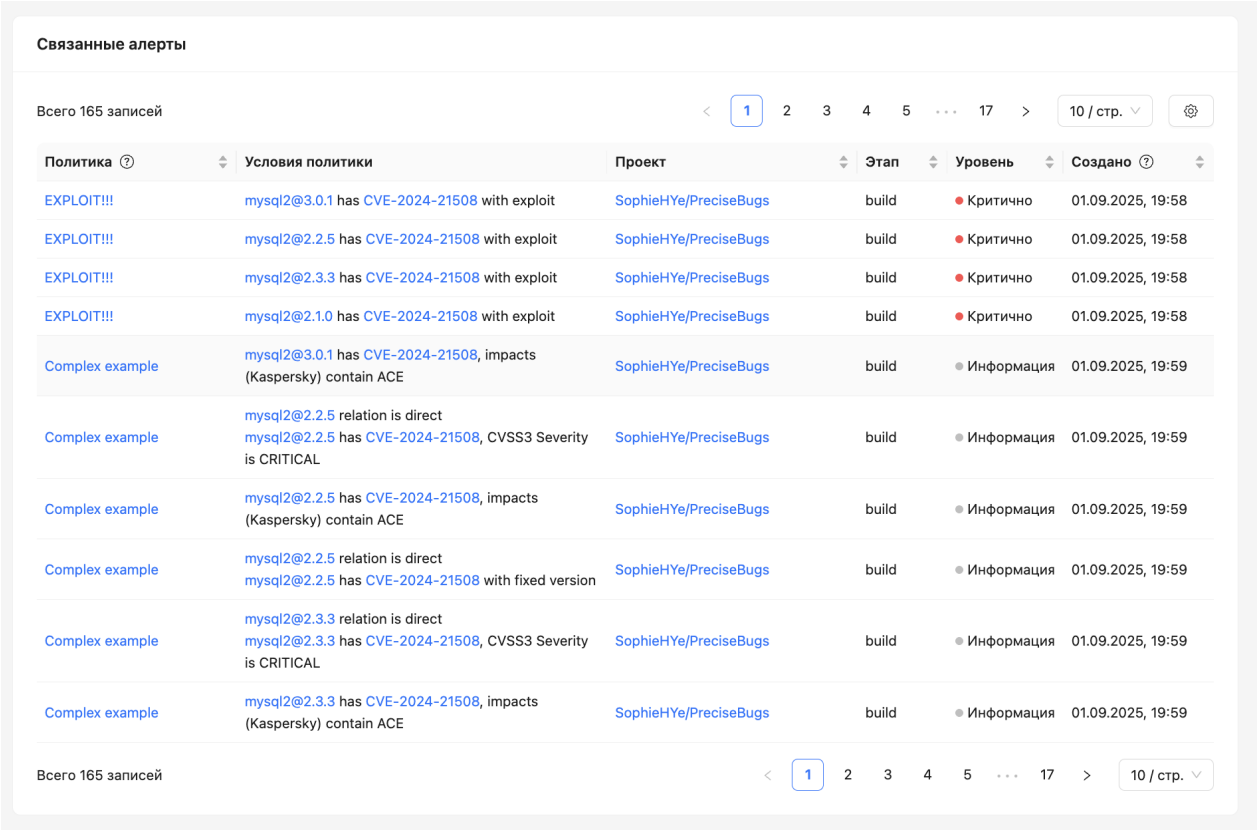
#Связанные алерты

В нижней части страницы представлен список связанных алертов, сгенерированных политиками безопасности.

Для каждого алерта отображается:

- политика, в рамках которой он был создан;
- условия срабатывания;
- проект и этап разработки;
- уровень критичности;
- дата и время создания.

Это позволяет быстро понять, **какие правила безопасности затрагивает уязвимость** и где именно она влияет на проект.



#Дополнительная информация

В правой части страницы также отображаются:

- ссылки на внешние ресурсы (NVD, CVE.org, GitHub и другие);
- внутренний идентификатор уязвимости в CodeScoring;
- дата последнего обновления данных.

Ссылки

https://github.com/pgjdbc/pgjdbc/commit/f4d0ed69c0b3aae8531d83d6af4c57f22312c813

x_refsource_MISC

https://github.com/pgjdbc/pgjdbc/security/advisories/GHSA-v7wg-cpwc-24m4

x_refsource_CONFIRM

[debian-lts-announce] 20220520 [SECURITY] [DLA 3018-1] libpgjava security update

mailing-list x_refsource_MLIST

FEDORA-2022-1151f65e9a

vendor-advisory x_refsource_FEDORA

https://security.netapp.com/advisory/ntap-20220311-0005/

x_refsource_CONFIRM

DSA-5196

vendor-advisory x_refsource_DEBIAN

https://github.com/advisories/GHSA-v7wg-cpwc-24m4

https://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/message/BVEO7BEFXPBVHSPYL3YKQWZI6DYXQLFS/

https://nvd.nist.gov/vuln/detail/CVE-2022-21724

https://security-tracker.debian.org/tracker/CVE-2022-21724

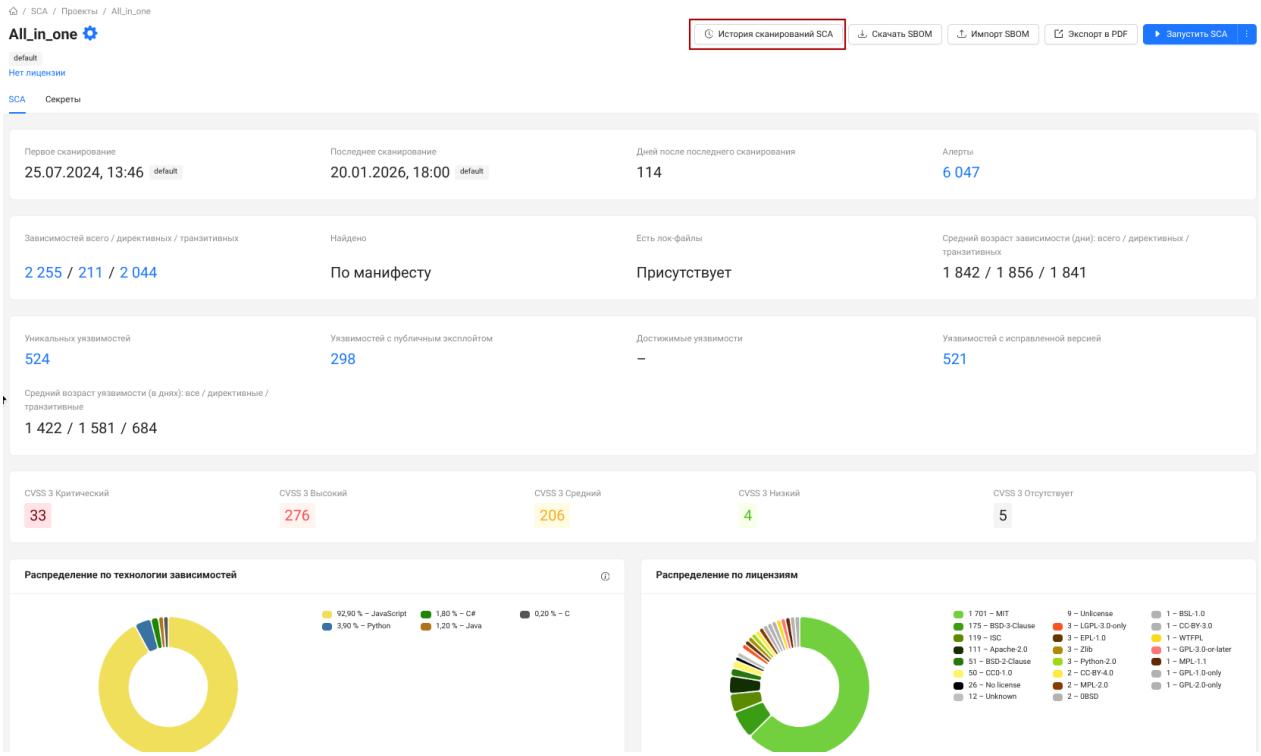
https://www.cve.org/CVERecord?id=CVE-2022-21724

https://поддержка.нппкт.рф/bin/view/ОСнова/Обновления/2.6/

CodeScoring ID UNCS-2022-9732

#Отслеживание истории результатов сканирования

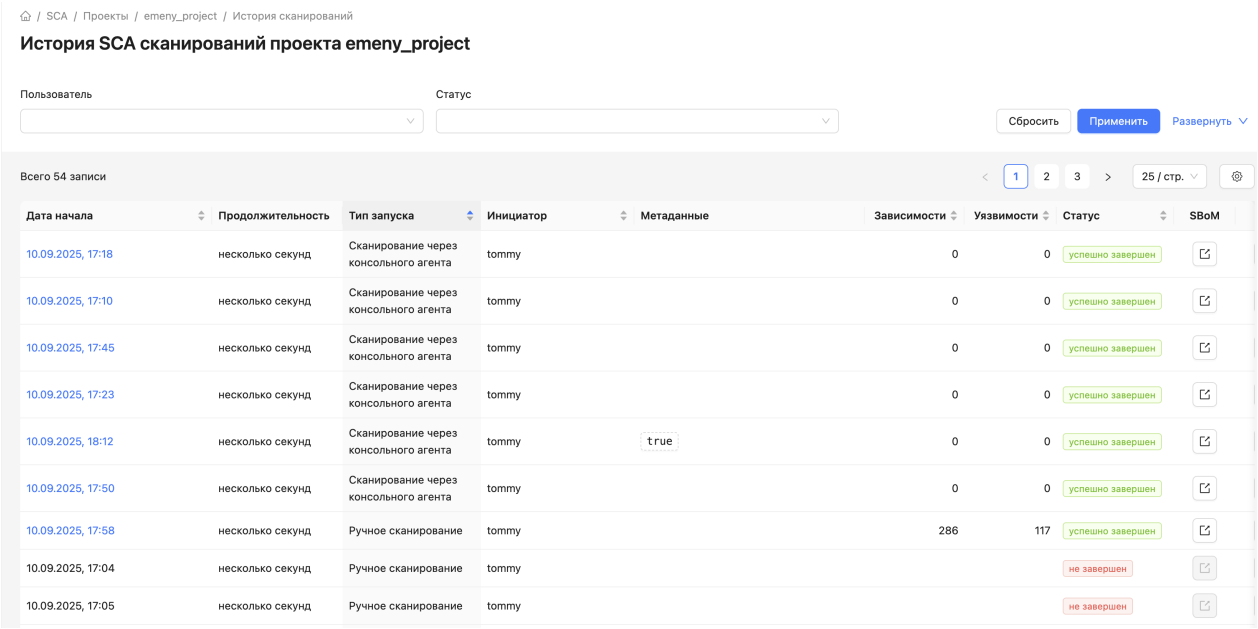
На каждый анализ проекта в рамках модуля SCA сохраняется снепшот результатов по найденным зависимостям и уязвимостям. Чтобы увидеть список доступных снепшотов, необходимо зайти на вкладку SCA на странице проекта и нажать на кнопку **SCA scan history**.



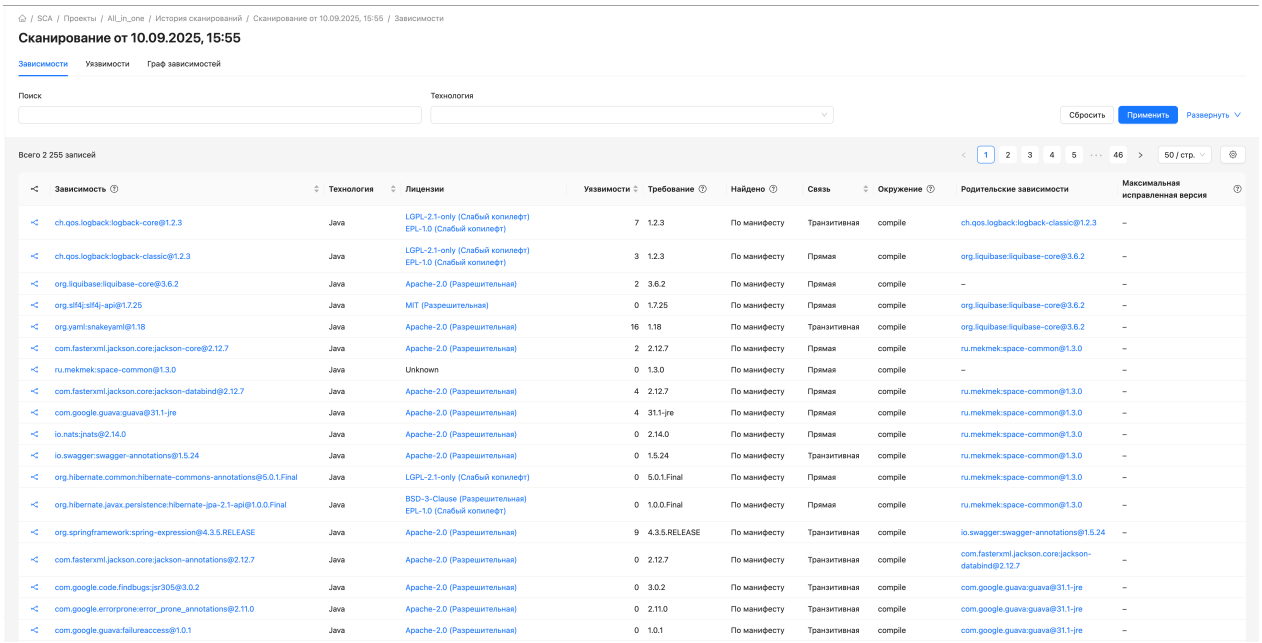
Снепшот содержит следующие данные:

- **Дата начала** – дата начала сканирования проекта. Дата последнего сканирования отмечается зеленым лейблом latest;
- **Продолжительность** – время, которое длился анализ;
- **Тип запуска** – тип запуска сканирования, ручной или по расписанию;
- **Инициатор** – пользователь, запустивший сканирование. Для запусков по расписанию указывается значение “system”;
- **Версия** – информация о версии, имя и метаданные git ветки (для VCS проектов);
- **Зависимости** – число найденных зависимостей;
- **Уязвимости** – число найденных уязвимостей;
- **Статус** – статус завершенного сканирования. Может иметь три возможных значения – success, failed или cancelled.

По кнопке в правой части списка доступно скачивание SBOM, сгенерированного во время проведения анализа, а также экспорт отчета в PDF.



Для того, чтобы посмотреть более подробную информацию по сканированию, необходимо нажать на ссылку с датой сканирования в первой колонке. На данной странице доступны список зависимостей, список уязвимостей и граф зависимостей проекта на момент выполнения анализа.



#Выгрузка результатов анализа

#Выгрузка в CSV

Каждую таблицу с результатами анализа в CodeScoring можно выгрузить в формате CSV, используя кнопку **Экспорт** в правом верхнем углу интерфейса.

CSV-таблица будет учитывать использованные на момент выгрузки фильтры.

#Формирование PDF-отчета по проекту

После проведения композиционного анализа на странице проекта становится доступным формирование PDF-отчета со сводной информацией по проекту.

Экспортировать PDF-отчет с данными последнего анализа можно на странице проекта по кнопке **Экспорт в PDF**. Экспортировать отчет по анализу за определенную дату можно на странице `История сканирований SCA`.

Отчет будет сформирован на том языке, который установлен в профиле пользователя.

Полученный отчет по умолчанию содержит следующие данные:

- общая информация по проекту (название, ветка VCS, время проведения последнего анализа, хэш коммита);
- распределение уязвимостей по CVSS;
- распределение уязвимостей по технологиям;
- таблица найденных зависимостей с разделением по технологиям и окружению разработки;
- таблица найденных уязвимостей с разделением по технологиям и окружению разработки;
- таблица активных алертов политик;
- таблица игноров политик;
- граф зависимостей в виде дерева.

Также есть возможность задать имя файла, выбрать необходимые блоки данных и отфильтровать данные перед генерацией отчета. По умолчанию в отчете будут перечислены только эффективные игноры политик. Эффективными являются те игноры, которые применены к политикам, которые активны в данный момент. Чтобы в списке отображались все игноры политик, нужно убрать галочку с чекбокса "Только эффективные игноры политик".

Если имя файла не указано, то оно автоматически сгенерируется по следующим правилам:

- Для проектов: `report_<название проекта>.pdf`
- Для контейнерных образов: `report_<название образа>_<первые 8 символов хэша>.pdf`

Экспорт PDF

Имя файла

* Содержимое

- [x] Сводка
- [x] Зависимости
- [x] Лицензии
- [x] Уязвимости
- [x] Алерты политик
- [x] Игноры политики
- [x] Дерево зависимостей
- [x] Примечание триажа

Экспортировать всё | Фильтровать данные

- [x] Только эффективные игноры политик

Окружение

Технология

Тип соответствия

Тип связи

#Работа со SBOM в рамках проекта

После проведения композиционного анализа проекта становится доступна выгрузка полученного перечня используемых компонентов (SBOM) в формате CycloneDX.

Выгрузить полученный SBOM можно на странице проекта в разделе Проекты по кнопке **Скачать SBOM**.

Экспорт SBOM поддерживается в следующих форматах:

- CycloneDX v1.4 JSON;
- CycloneDX v1.5 JSON;
- CycloneDX v1.6 JSON;
- CycloneDX v1.6 Ext JSON – расширенный формат CycloneDX, содержащий дополнительные свойства: `GOST:attack_surface`, `GOST:security_function`, `GOST:source_lang`, `GOST:provided_by`. Формат адаптирован под дополнительные требования к перечню программных компонентов от ФСТЭК России.
- CycloneDX v1.7 JSON;
- CycloneDX v1.7 Ext JSON – расширенный формат CycloneDX, содержащий дополнительные свойства: `GOST:attack_surface`, `GOST:security_function`, `GOST:source_lang`, `GOST:provided_by`. Формат адаптирован под дополнительные требования к перечню программных компонентов от ФСТЭК России.
- SPDX v2.3 JSON

Для выгрузки SBOM так же, как и для PDF, возможна дополнительная настройка. Правила автоматической генерации имен файлов SBOM следующие:

- Для проектов: `bom_<название проекта>_<формат SBOM>.json`
- Для контейнерных образов: `bom_<название образа>_<первые 8 символов хэша>_<формат SBOM>.json`

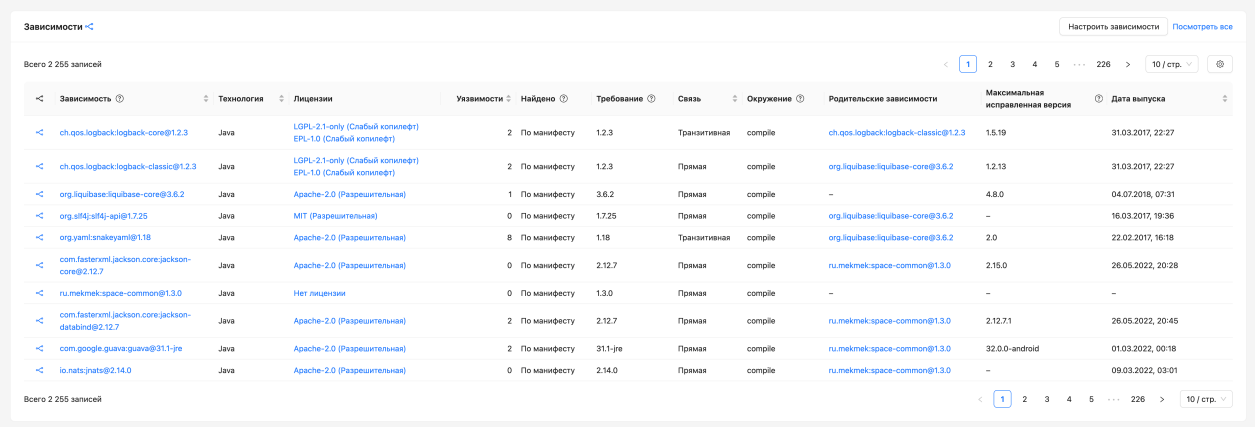
#Импорт SBOM

Для CLI-проектов также доступна загрузка SBOM через интерфейс по кнопке **Импорт SBOM**. Загружаемый SBOM должен быть в формате CycloneDX v1.4, 1.5, 1.6, 1.6_ext, 1.7 или 1.7_ext и иметь расширение `.json`.

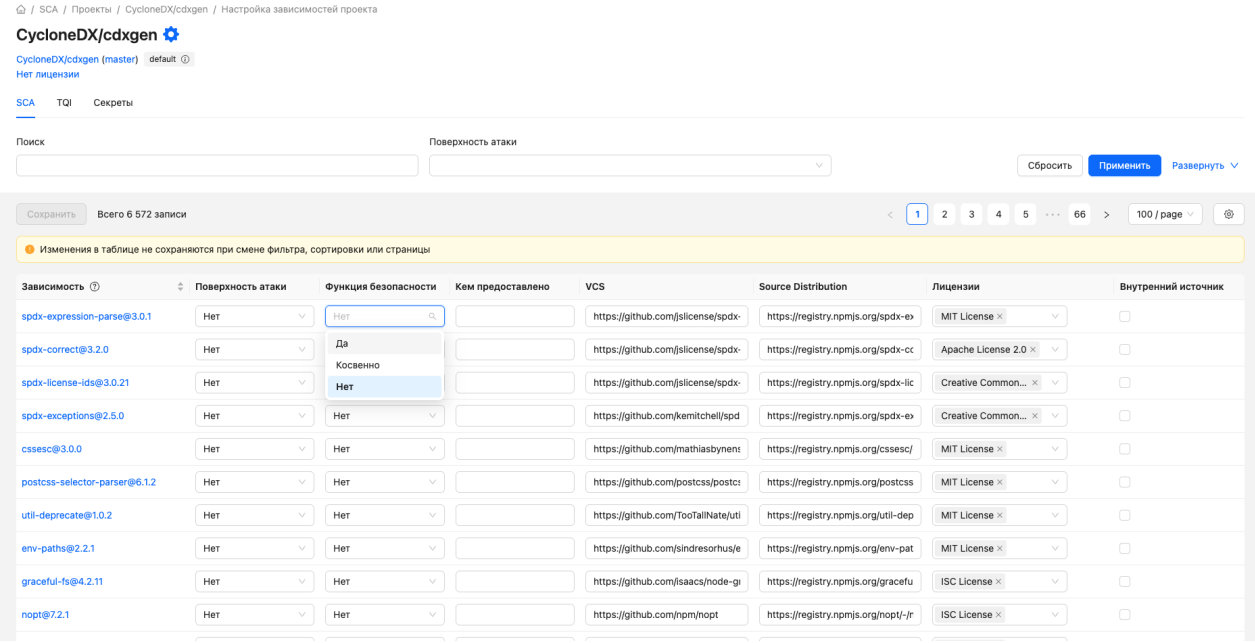
При импорте SBOM можно указать ветку или тег в качестве мета-информации.

#Настройка свойств зависимостей для выгрузки в SBOM

Для настройки свойств зависимостей необходимо перейти на страницу по кнопке Настроить зависимости в таблице зависимостей на странице проекта.



Страница позволяет указать поверхность атаки, функцию безопасности, ссылку на исходный код и лицензии для каждого компонента проекта.



Введенные значения учитываются:

- при экспорте SBOM со страницы проекта;
- при экспорте SBOM со страницы истории результатов сканирования (для самого последнего успешного SCA-сканирования);
- при последующих сканированиях проекта;
- при сканировании проекта через консольный агент Johnny;
- в дашборде проекта;
- на странице лицензии.

Важно: изменения значений не применяются к предыдущим сканированиям проекта и относятся только к SBOM текущего проекта, даже если зависимость используется в нескольких проектах.

#VCS

Поле **VCS** позволяет указать URL-адрес репозитория, в котором хранится код зависимости. При экспорте SBOM выбранное значение учитывается в поле externalReferences.

#Source Distribution

Поле **Source Distribution** содержит URL-адрес исходных кодов пакета. При экспорте SBOM выбранное значение учитывается в поле externalReferences.

#Поверхность атаки

Поле **Поверхность атаки** позволяет указать принадлежность компонента к поверхности атаки. Можно выбрать одно из следующих значений:

- Да — компонент входит в непосредственную поверхность атаки;
- Косвенно — компонент входит в косвенную поверхность атаки;
- Нет — иной случай (значение по умолчанию).

При экспорте SBOM в форматах `CycloneDX v1.6 Ext JSON` и `CycloneDX v1.7 Ext JSON` выбранное значение учитывается в свойстве `GOST:attack_surface` компонента.

#Функция безопасности

Поле **Функция безопасности** позволяет указать принадлежность компонента к функциям безопасности средства защиты информации. Можно выбрать одно из следующих значений:

- Да — если функции компонента непосредственно реализуют функции безопасности;
- Косвенно — если функции компонента участвуют в реализации функций безопасности, взаимодействуя с компонентами, реализующими функции безопасности;
- Нет — если функции компонента не участвуют в реализации функций безопасности (значение по умолчанию).

При экспорте SBOM в форматах `CycloneDX v1.6 Ext JSON` и `CycloneDX v1.7 Ext JSON` выбранное значение учитывается в свойстве `"GOST:security_function"` компонента.

#Кем предоставлено

Поле **Кем предоставлено** позволяет указать принадлежность компонента к средству защиты информации, из состава которого заимствован данный компонент. Можно указать произвольное значение в текстовом формате.

При экспорте SBOM в форматах `CycloneDX v1.6 Ext JSON` и `CycloneDX v1.7 Ext JSON` указанное значение учитывается в свойстве `"GOST:provided_by"` компонента.

#Лицензии

Поле **Лицензии** позволяет указать лицензии компонента.

При указании пустого списка выбираются значения, найденные при последнем SCA-анализе.

При экспорте SBOM выбранные значения учитываются в поле `licenses` компонента.

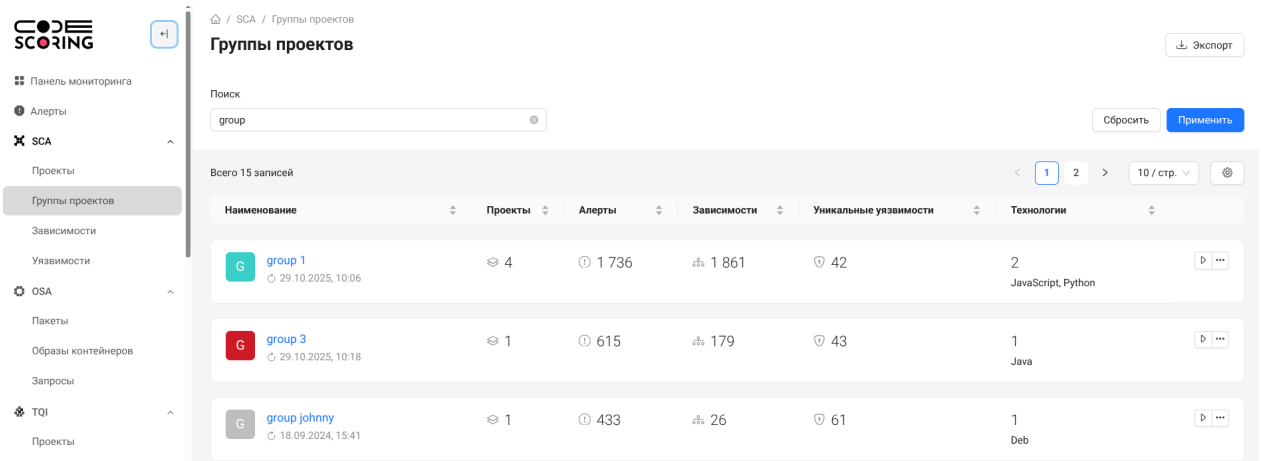
#Организация групп проектов

#Просмотр списка групп

Список групп можно посмотреть в разделе `SCA -> Группы проектов`. В данном разделе отображаются все группы, доступные для просмотра пользователю.

Для каждой группы отображается следующая информация:

- **Наименование** - название группы и дата последнего обновления настроек группы проектов;
- **Проекты** - количество проектов в составе группы;
- **Алерты** - общее количество алертов, связанных с проектами группы;
- **Зависимости** - общее количество зависимостей, связанных с проектами группы;
- **Уникальные уязвимости** - количество уникальных уязвимостей, связанных с проектами группы;
- **Технологии** - количество и список технологий, связанных с проектами группы.



Доступ

В группе отображаются только проекты входящие в группу и к которым пользователь имеет доступ.

Для каждой записи в списке проектов доступны следующие действия:

- **Запуск сканирования проектов группы** - выполняется запуск сканирования всех проектов, входящих в группу;
- **Редактирование группы** - открывается страница **Настройки -> Группы**;
- **Удаление группы** - удаляется группа с предварительным подтверждение операции у пользователя;
- **Экспорт данных в CSV** - агрегированные данные группы экспортируются в файл csv;
- **Скачивание SBOM** - выполняется выгрузка полученного после композиционного анализа перечня используемых компонентов (SBOM) для всех проектов, входящих в группу.

Доступ

Для осуществления действий требуется, чтобы у пользователя были соответствующие полномочия. Описание полномочий представлены в разделе Управление учетными записями.

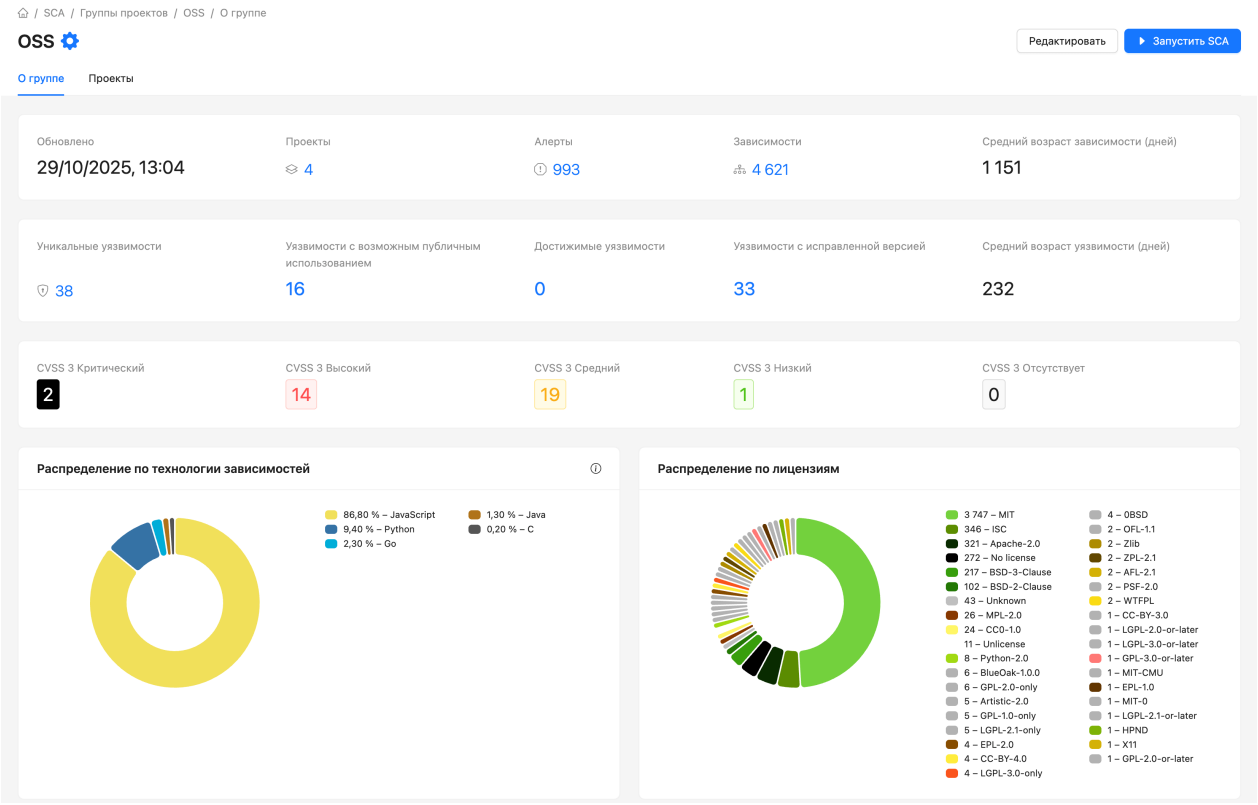
Скачивание SBOM

Для скачивания SBOM требуется, чтобы у каждого проекта был успешно завершен композиционный анализ.

#Страница группы проектов

#Общая статистика по группе

В разделе "О группе" предоставляется обобщенная информация.



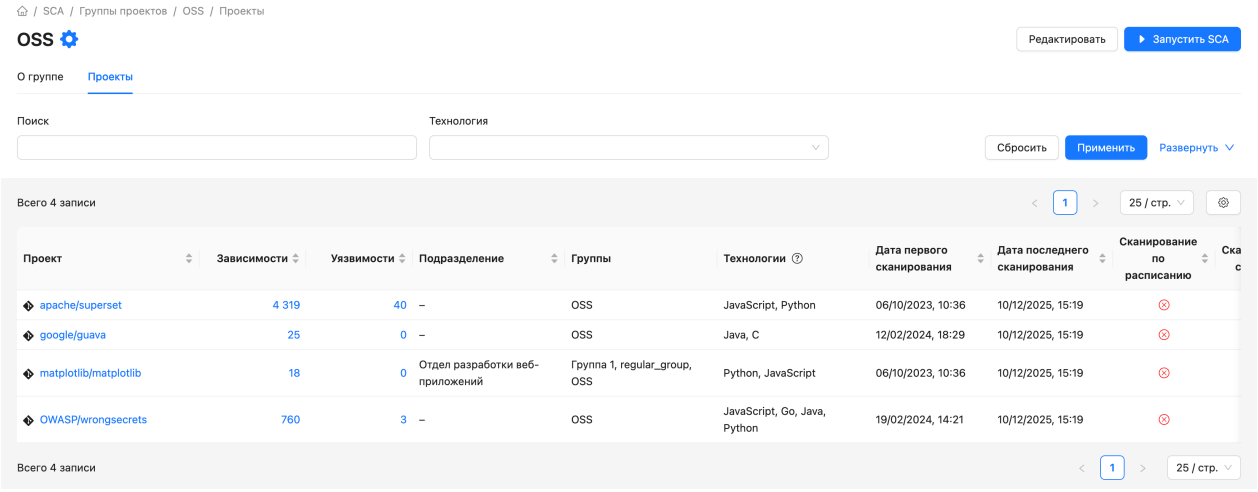
В разделе также представлены секции **Алерты**, **Уязвимости** и **Зависимости**, связанные с проектами данной группы.

Для удобства в каждой секции присутствуют стандартные фильтры.

#Состав группы

В разделе "Проекты" представлен список проектов, которые относятся к данной группе.

Для удобства работы представлена возможность фильтрации списка проектов.



#Работа с консольным агентом

Консольный агент **Johnny** предоставляется совместно с on-premise версией CodeScoring.

Агент — это исполняемый бинарный файл, осуществляющий парсинг манифестов известных пакетных менеджеров, сканирование Docker-образов, анализ сборки C и C++, разбор архивов и поиск прямых включений Open Source библиотек по хэшам. Агент работает в паре с платформой, получая от нее обогащенные данные об уязвимостях, лицензиях и настроенных политиках, а также сохраняя результаты сканирования в CLI проекты.

По умолчанию предоставляется сборка агента для Linux-совместимых систем. По запросу доступны сборки под Windows и MacOS.

Скачать исполняемый файл агента можно через платформу, используя следующие адреса:

- `[platform-url]/download/` – страница со списком доступных исполняемых файлов;
- `[platform-url]/download/johnny_version` – актуальная версия консольного агента;
- `[platform-url]/download/<file-name>` – загрузка исполняемого файла.

Для просмотра актуальной версии и загрузки файла необходима авторизация по API-токену.

#Принцип работы

При работе в режиме сканирования директорий с исходным кодом, агент рекурсивно обходит директорию, указанную в параметрах запуска, и осуществляет поиск и разбор манифестов известных пакетных менеджеров.

В режиме сканирования образов агент исследует файловую систему указанного образа, производя инвентаризацию компонентного состава.

По окончанию работы формируется **SBOM** файл, и в консоль выводится информация о найденных уязвимостях и сработавших политиках.

Пример вывода найденных уязвимостей:

Vulnerabilities:

Vulnerability		Affect				
Vulnerability	Has exploit	Fixed version	CVSSv2	CVSSv3	CWEs	Links
CVE-2021-21345	V	pkg:maven/com.thoughtworks.xstream/xstream@1.4.5				
		1.4.16	6.5 Medium	9.9 Critical	CWE-502 CWE-78 CWE-94	https://access.redhat.com/security/cve/CVE-2021-21345 https://bdu.fstec.ru/vul/2021-05946 https://github.com/advisories/GHSA-hwpc-8xqv-jvj4 https://nvd.nist.gov/vuln/detail/CVE-2021-21345 https://osv.dev/vulnerability/CVE-2021-21345 https://security-tracker.debian.org/tracker/CVE-2021-21345 https://ubuntu.com/security/CVE-2021-21345 https://www.cve.org/CVERecord?id=CVE-2021-21345
CVE-2019-19919		pkg:maven/com.github.jknack/handlebars@4.0.7				
			7.5 High	9.8 Critical	CWE-1321 CWE-74	https://bdu.fstec.ru/vul/2020-00795 https://github.com/advisories/GHSA-w457-6q6x-cgp9 https://nvd.nist.gov/vuln/detail/CVE-2019-19919 https://osv.dev/vulnerability/CVE-2019-19919 https://security-tracker.debian.org/tracker/CVE-2019-19919 https://ubuntu.com/security/CVE-2019-19919 https://www.cve.org/CVERecord?id=CVE-2019-19919
CVE-2021-21344	V	pkg:maven/com.thoughtworks.xstream/xstream@1.4.5				
		1.4.16	7.5 High	9.8 Critical	CWE-434 CWE-502	https://access.redhat.com/security/cve/CVE-2021-21344 https://bdu.fstec.ru/vul/2021-05499 https://github.com/advisories/GHSA-59jw-jqf4-3wq3 https://nvd.nist.gov/vuln/detail/CVE-2021-21344 https://osv.dev/vulnerability/CVE-2021-21344 https://security-tracker.debian.org/tracker/CVE-2021-21344 https://ubuntu.com/security/CVE-2021-21344 https://www.cve.org/CVERecord?id=CVE-2021-21344
CVE-2021-21346		pkg:maven/com.thoughtworks.xstream/xstream@1.4.5				
		1.4.16	7.5 High	9.8 Critical	CWE-434 CWE-502	https://access.redhat.com/security/cve/CVE-2021-21346 https://bdu.fstec.ru/vul/2021-05502 https://github.com/advisories/GHSA-4hrm-m67v-5cxr https://nvd.nist.gov/vuln/detail/CVE-2021-21346 https://osv.dev/vulnerability/CVE-2021-21346 https://security-tracker.debian.org/tracker/CVE-2021-21346 https://ubuntu.com/security/CVE-2021-21346 https://www.cve.org/CVERecord?id=CVE-2021-21346

Пример вывода сработавших политик:

Policy alerts:

Policy name	Blocks build	Level	Dependency	Vulnerability
Without version		Warning	pkg:maven/com.github.boscojared.disparse/disparse-core	
			pkg:maven/com.github.boscojared.disparse/disparse-smalld	
			pkg:maven/com.google.guava/listenablefuture	
Vuln block	V	Critical	pkg:maven/com.google.guava/guava@31.0.1-jre	CVE-2020-8908
				CVE-2023-2976
			pkg:maven/com.squareup.okhttp3/okhttp@3.12.10	CVE-2021-0341
			pkg:maven/org.apache.logging.log4j/log4j-core@2.16.0	CVE-2021-45105
				CVE-2021-44832
			pkg:maven/com.google.guava/guava@31.0.1-jre	CVE-2023-2976
			pkg:maven/com.squareup.okhttp3/okhttp@3.12.10	CVE-2021-0341
			pkg:maven/com.squareup.okio/okio@1.15.0	CVE-2023-3635
			pkg:maven/org.apache.logging.log4j/log4j-core@2.16.0	CVE-2021-45105
				CVE-2021-44832
EXPLOIT!!!			pkg:maven/com.google.guava/guava@31.0.1-jre	CVE-2020-8908
			pkg:maven/com.squareup.okio/okio@1.15.0	CVE-2023-3635

Пример вывода достижимых уязвимостей:

#Настройка через конфигурационный файл

Управлять параметрами консольного агента можно через добавление файла конфигурации `codescoring-johnny-config.yaml` в директорию с агентом. Ниже представлен список доступных параметров и пример конфиг-файла.

#Список параметров

#Параметры композиционного анализа

- **project** – название проекта в платформе CodeScoring;
- **save-results** – сохранение результатов в платформе CodeScoring. Используется в паре с названием проекта. Значение по умолчанию – `false`;
- **license** – лицензия анализируемого проекта, например `mit`;
- **stage** – этап разработки. Возможные значения: `build`, `dev`, `source`, `stage`, `test`, `prod`, `proxy`;
- **bom-path** – путь (с названием файла), по которому будет сохраняться сформированный файл `bom.json`;
- **bom-format** – формат формируемого SBOM. Возможные значения: `cyclonedx_v1_6_json`, `cyclonedx_v1_5_json`, `cyclonedx_v1_4_json`, `cyclonedx_v1_6_ext_json`, `cyclonedx_v1_7_json`. Значение по умолчанию: `cyclonedx_v1_6_json`;
- **timeout** – ограничение по времени ожидания анализа (в секундах);
- 2024.52.0 **branch-or-tag** – ссылка на ветку репозитория или тег, например `refs/tags/v1.0` (для команд `scan dir` и `scan file`);
- 2024.52.0 **commit** – хэш коммита в системе контроля версий (для команд `scan dir` и `scan file`);
- 2024.52.0 **hash** – хэш образа (для команды `scan image`);
- 2025.7.0 **cloud-resolve** – использование разрешения зависимостей в облаке. По умолчанию значение `false`.

#Общие параметры сканирования

- **ignore** – директории, которые будут игнорироваться при сканировании;
- **no-summary** – скрытие сводной информацию по проведенному сканированию в консоли. По умолчанию значение `false`;
- **only-hashes** – поиск **только** прямых включений Open Source библиотек по хэшам. По умолчанию значение `false`;
- **with-hashes** – поиск прямых включений Open Source библиотек по хэшам. По умолчанию значение `false`;
- **no-recursion** – выключение рекурсивного скана для команды `scan dir`. По умолчанию значение `false`;
- **block-on-empty-result** – блокирование сборки при получении пустого результата. При активации агент возвращает exit code **3** в случае отсутствия артефактов для анализа;
- 2026.20.0 **include-envs** – включение в результат только указанных окружений зависимостей, через запятую, например `compile,runtime`. Взаимоисключается с `exclude-envs`;
- 2026.20.0 **exclude-envs** – исключение указанных окружений зависимостей из результата, через запятую, например `test,dev`. Взаимоисключается с `include-envs`.

#Параметры сканирования Docker-образов

- **scan-files** – сканирование файловой системы внутри образа. По умолчанию значение `false`;
- **insecure-skip-tls-verify** – пропуск TLS верификации при подключении к реестру образов. По умолчанию значение `false`;
- **insecure-use-http** – использование протокола http при подключении к реестру образов. По умолчанию значение `false`;
- **registries** – список конфигураций для подключения к нескольким реестрам образов. Каждый элемент списка может содержать:
  - **authority** – URL реестра (например, `docker.io`, `localhost:5000`);
  - **login** – имя пользователя для подключения к реестру;
  - **password** – пароль для подключения к реестру;
  - **token** – токен для подключения к реестру. Взаимоисключается с параметрами `login` и `password`.

#Параметры сканирования сборки C и C++

- **build-result** – флаг, указывающий, что входными данными являются результаты предыдущей сборки, включая скомпилированные артефакты. По умолчанию значение `false`;
- **lib-versions** – путь к JSON-файлу со списком версий анализируемых библиотек;
- **output** – путь к файлу, в который будут сохранены результаты анализа сборки;
- **unresolved-file** – путь к файлу, в который будет сохранена информация о библиотеках с неразрешёнными версиями.

#Параметры парсинга для разных технологий

#Общие параметры

- **enabled** – включение парсеров для данной технологии;
- **parsers** – набор парсеров для манифестов.

#Параметры парсеров

- **enabled** – включение данного парсера;
- **match** – условие для определения подходящих манифестов, может быть по названию (`equal`) или расширению (`extension`);
- **properties** – дополнительные свойства для парсеров окружения, такие как путь к исполняемым файлам;
- **dotnet-path**, **maven-path**, **gradle-path**, **yarn-path**, **go-path**, **sbt-path**, **npm-path**, **pnpm-path**, **composer-path**, **pip-path**, **poetry-path**, **conda-lock-path** – пути к пакетным менеджерам для разрешения зависимостей в окружении;
- 2026.20.0 **pdm-path** – путь к `pdm` для разрешения зависимостей в окружении;
- 2025.45.0 **pipdeptree-path** – путь к `pipdeptree` для разрешения зависимостей в окружении;
- 2026.3.0 **bun-path** – путь к `bun` для разрешения зависимостей в окружении;
- 2025.45.0 **uv-path** – путь к `uv` для разрешения зависимостей в окружении;
- 2025.13.0 **swift-path** – путь к `swift` для разрешения зависимостей в окружении;
- **resolve-enabled** – разрешение зависимостей в окружении. По умолчанию значение `false`;
- **dotnet-args**, **gradle-args**, **maven-args**, **sbt-args**, **npm-args**, **yarn-args**, **pnpm-args**, **composer-args**, **pip-args**, **poetry-args**, **conda-args** – аргументы для передачи соответствующим пакетным менеджерам при разрешении зависимостей в окружении;
- 2026.20.0 **pdm-args** – аргументы для передачи `pdm` при разрешении зависимостей в окружении;
- 2025.45.0 **pipdeptree-args** – аргументы для передачи `pipdeptree` при разрешении зависимостей в окружении;
- 2026.3.0 **bun-args** – аргументы для передачи `bun` при разрешении зависимостей в окружении;
- 2025.45.0 **uv-args** – аргументы для передачи `uv` при разрешении зависимостей в окружении;
- 2025.13.0 **swift-args** – аргументы для передачи `swift` при разрешении зависимостей в окружении;
- **configuration** – конфигурация для парсера `gradle-dependency-tree_txt`;
- **depth** – глубина парсинга для парсера `jar`. По умолчанию значение `1`;
- **python-version** – версия Python, используемая для разрешения зависимостей в окружении.

#Параметры сканирования архивов

- **scan** – сканирование архивов. По умолчанию значение `false`;
- **depth** – глубина сканирования архивов. По умолчанию значение 1.

#Параметры вывода результатов

- 2023.48.0 **format** – формат вывода найденных уязвимостей. По умолчанию `coloredtable`. Возможна выгрузка в форматы `table`, `text`, `junit`, `sarif`, `csv`, `gl-dependency-scanning-report`, `gl-code-quality-report`;
- 2023.48.0 **group-vulnerabilities-by** – переменная для группировки уязвимостей в таблице;
- 2023.48.0 **sort-vulnerabilities-by** – порядок переменных для сортировки уязвимостей в таблице;
- 2025.29.0 **alerts-format** – формат вывода отчёта по срабатываниям политик. Поддерживаются форматы: `coloredtable`, `table`, `text`, `json`, `csv`. Значение по умолчанию – `coloredtable`.

#Параметры платформы

- **api_url** – адрес платформе;
- **api_token** – токен для доступа к платформе;
- 2026.3.0 **localization** — язык локализации вывода CLI. Возможные значения: en, ru. Значение по умолчанию — en.

#Параметры запуска поиска секретов

- 2025.13.0 **gitleaks-path** – путь к исполняемому файлу gitleaks, который будет использоваться при сканировании;
- 2025.13.0 **gl-secrets-report** – включение формирования отчета о найденных секретах в формате GitLab. По умолчанию `false`;
- 2025.13.0 **gl-secrets-report-filename** – имя формируемого файла для отчета в формате GitLab. По умолчанию `gl-secrets-report.json`.

#Параметры **инструмента поиска секретов Gitleaks**

- 2025.13.0 **baseline-path** – путь к baseline файлу отчета gitleaks. Все обнаруженные ранее секреты, зафиксированные в этом файле, будут проигнорированы при повторном сканировании;
- 2025.13.0 **enable-rule** – список ID правил, которые будут **включены** при сканировании;
- 2025.13.0 **gitleaks-ignore-path** – путь к файлу .gitleaksignore или директории, содержащей его. По умолчанию `.` (текущая директория);
- 2025.13.0 **ignore-gitleaks-allow** – игнорирование комментариев gitleaks:allow. По умолчанию `false`;
- 2025.13.0 **log-level** – уровень логирования. Возможные значения: `trace, debug, info, warn, error, fatal`. По умолчанию `info`;

- 2025.13.0 **max-decode-depth** – максимальная глубина рекурсивного декодирования. Значение `0` отключает декодирование;
- 2025.13.0 **max-target-megabytes** – максимальный размер файлов (в мегабайтах), которые будут обрабатываться. Файлы, превышающие этот размер, будут пропущены. По умолчанию 0 (ограничение отсутствует);
- 2025.13.0 **no-banner** – отключение баннера gitleaks при запуске. По умолчанию `false`;
- 2025.13.0 **no-color** – отключение цветного вывода для подробного (verbose) режима. По умолчанию `false`;
- 2025.13.0 **redact** – маскирование найденных секретов в логах и консоли. Значение 0 полностью отображает секреты, 100 – полностью скрывает. Можно задать промежуточное значение, например, 20 (маскирует 20% секрета). По умолчанию `0`;
- 2025.13.0 **verbose** – включение подробного (verbose) вывода при сканировании. По умолчанию `false`.

#Пример файла

```
## analysis options
analysis:
  #     Project name in CodeScoring
  project: ""
  # Save results to CodeScoring. Used only together with
project name
  save-results: false
  # Policy stage (build, dev, source, stage, test, prod, proxy)
  stage: build
  # License code
  license: mit
  # Path for save bom
  bom-path: "bom.json"
  # Format for bom
  bom-format: cyclonedx_v1_6_json
  # Timeout of analysis results waiting in seconds
  timeout: 3600
  # Reference to repository branch or tag (e.g. refs/tags/
v1.0). For scan dir and scan file commands
  branch-or-tag: ""
  # Commit. For scan dir and scan file commands
  commit: ""
  # Hash. For scan image command
  hash: ""
  # Use cloud resolve
  cloud-resolve: false
## scan options
scan:
  # general scan options
  general:
    # Ignore paths
```

```
    # - first
    # - /**/onem?re
    ignore:
      - .tmp
      - parsers
      - fixtures
      - .git
    # Do not print summary
    no-summary: false
    # Search only for direct inclusion of dependencies using
file hashes
    only-hashes: false
    # Search for direct inclusion of dependencies using file
hashes
    with-hashes: false
    # Block on empty result
    block-on-empty-result: true
    # Include only the listed dependency environments (scopes)
in the result.
    # Comma-separated string. Mutually exclusive with exclude-
envs.
    include-envs: ""
    # Exclude the listed dependency environments (scopes) from
the result.
    # Comma-separated string. Mutually exclusive with include-
envs.
    exclude-envs: ""
  # image scan options
  image:
    # scan files in image
    scan-files: false
    # skip TLS verification when communicating with the
registry
    insecure-skip-tls-verify: false
    # use http instead of https when connecting to the registry
    insecure-use-http: false
    # credentials for specific registries
    registries:
      - # the URL to the registry (e.g. "docker.io",
"localhost:5000", etc.)
        # same as JOHNNY_REGISTRY_AUTH_AUTHORITY env var
        authority: ""
        # same as JOHNNY_REGISTRY_AUTH_LOGIN env var
        login: ""
        # same as JOHNNY_REGISTRY_AUTH_PASSWORD env var
        password: ""
        # note: token and username/password are mutually
exclusive
        # same as JOHNNY_REGISTRY_AUTH_TOKEN env var
        token: ""
  # Directory scan options
  dir:
```

```
    # Prevents from recursively scan directories
    no-recursion: false
  # Scanning a build for C and C++ languages options
  build:
    # input is the result of the previous build process,
including compiled artifacts
    build-result: false
    # path to a JSON file with a list of versions of the
libraries being analyzed
    lib-versions: ""
    # path to a file where the build results will be saved
    output: ""
    # path to a file where information about libraries with
unresolved versions will be saved
    unresolved-file: UnresolvedLibs20241030_123655.json
  # Supported technologies
  technologies:
    # C
    clang:
      # Use C parsers
      enabled: true
      # C parsers
      parsers:
        # conan.lock parser
        conan_lock:
          # use parser
          enabled: true
          # matching criteria
          match: equal("conan.lock")
        # conanfile.py parser
        conanfile_py:
          # use parser
          enabled: true
          # matching criteria
          match: equal("conanfile.py")
        conanfile_txt:
          # use parser
          enabled: true
          # matching criteria
          match: equal("conanfile.txt")
    # C#
    csharp:
      # Use C# parsers
      enabled: true
      # C# parsers
      parsers:
        # .csporj parser
        csproj:
          # use parser
          enabled: true
          # matching criteria
          match: extension(".csproj")
```

```
        # dependencyReport.json parser
        dependencyreport_json:
          # use parser
          enabled: true
          # matching criteria
          match: equal("dependencyReport.json")
        # .csproj dotnet environment parser
        dotnet_csproj_env:
          # use parser
          enabled: false
          # matching criteria
          match: extension(".csproj")
          # parser properties
          properties:
            # path to dotnet for resolve
            dotnet-path: dotnet
            # pass args to dotnet tool
            dotnet-args: ""
        sln:
          # use parser
          enabled: true
          # matching criteria
          match: extension(".sln")
        sln_env:
          # use parser
          enabled: false
          # matching criteria
          match: extension(".sln")
        # .nuspec parser
        nuspec:
          # use parser
          enabled: true
          # matching criteria
          match: extension(".nuspec")
        # packages.config parser
        packages_config:
          # use parser
          enabled: true
          # matching criteria
          match: equal("packages.config")
        # packages.lock.json parser
        packages_lock_json:
          # use parser
          enabled: true
          # matching criteria
          match: equal("packages.lock.json")
        # paket.dependencies parser
        paket_dependencies:
          # use parser
          enabled: true
          # matching criteria
          match: equal("paket.dependencies")
```

```
      # paket.lock parser
      paket_lock:
        # use parser
        enabled: true
        # matching criteria
        match: equal("paket.lock")
      # project.assets.json parser
      project_assets_json:
        # use parser
        enabled: true
        # matching criteria
        match: equal("project.assets.json")
      # Project.json parser
      project_json:
        # use parser
        enabled: true
        # matching criteria
        match: equal("Project.json")
      # Project.lock.json parser
      project_lock_json:
        # use parser
        enabled: true
        # matching criteria
        match: equal("Project.lock.json")
  # Golang
  go:
    # Use Golang parsers
    enabled: true
    # Golang parsers
    parsers:
      # go.mod parser
      go_mod:
        # use parser
        enabled: true
        # matching criteria
        match: equal("go.mod")
      # go.mod environment parser
      go_mod_env:
        # use parser
        enabled: false
        # matching criteria
        match: equal("go.mod")
        # parser properties
        properties:
          # path to go for resolve
          go-path: go
      # go.sum parser
      go_sum:
        # use parser
        enabled: true
        # matching criteria
        match: equal("go.sum")
```

```
    # Java
    java:
      # Use Java parsers
      enabled: true
      # Java parsers
      parsers:
        # build.gradle, build.gradle.kts environment parser
        build_gradle_env:
          # use parser
          enabled: false
          # matching criteria
          match: extension("build.gradle") ||
extension("build.gradle.kts")
          # parser properties
          properties:
            # path to gradle for resolve
            gradle-path: ./gradlew
            # args to gradle tool
            gradle-args: ""
        # .gradle parser
        gradle:
          # use parser
          enabled: true
          # matching criteria
          match: extension(".gradle")
        # gradle dependency tree parser
        gradle-dependency-tree_txt:
          # use parser
          enabled: true
          # matching criteria
          match: equal("gradle-dependency-tree.txt") ||
equal("gradle-dependencies.txt")
          # parser properties
          properties:
            # configuration for parse
            configuration: ""
        # .gradle.kts parser
        gradle_kts:
          # use parser
          enabled: true
          # matching criteria
          match: extension(".gradle.kts")
        # gradle.lockfile parser
        gradle_lockfile:
          # use parser
          enabled: true
          # matching criteria
          match: extension("gradle.lockfile")
        # ivy.xml parser
        ivy_xml:
          # use parser
          enabled: true
```

```
          # matching criteria
          match: equal("ivy.xml")
        # jar parser
        jar:
          # use parser
          enabled: true
          # matching criteria
          match: extension(".jar") || extension(".war") ||
extension(".ear")
          # parser properties
          properties:
            # parse depth
            depth: 1
        # maven dependency tree parser
        maven-dependency-tree_txt:
          # use parser
          enabled: true
          # matching criteria
          match: equal("maven-dependency-tree.txt") ||
equal("mvn-dependency-tree.txt")
        # pom.xml maven environment parser
        maven_pom_xml_env:
          # use parser
          enabled: false
          # matching criteria
          match: equal("pom.xml")
          # parser properties
          properties:
            # path to maven for resolve
            maven-path: mvn
            # args to mvn tool
            maven-args: ""
        # pom.xml parser
        pom_xml:
          # use parser
          enabled: true
          # matching criteria
          match: equal("pom.xml")
        # scala dependency tree parser
        scala-dependency-tree_txt:
          # use parser
          enabled: true
          # matching criteria
          match: equal("scala-dependency-tree.txt") ||
equal("sbt-dependency-tree.txt")
        # build.sbt environment parser
        scala_build_sbt_env:
          # use parser
          enabled: false
          # matching criteria
          match: equal("build.sbt")
          # parser properties
```

```
        properties:
          # path to sbt for resolve
          sbt-path: sbt
          # args to sbt tool
          sbt-args: ""
    # JavaScript
    js:
      # Use JavaScript parsers
      enabled: true
      # JavsScript parsers
      parsers:
        # npm-shrinkwrap.json parser
        npm-shrinkwrap_json:
          # use parser
          enabled: true
          # matching criteria
          match: equal("npm-shrinkwrap.json")
        # package.json npm environment parser
        npm_package_json_env:
          # use parser
          enabled: false
          # matching criteria
          match: equal("package.json")
          # parser properties
          properties:
            # path to npm for resolve
            npm-path: npm
            # args for npm tool
            npm-args: ""
        # package-lock.json parser
        package-lock_json:
          # use parser
          enabled: true
          # matching criteria
          match: equal("package-lock.json")
        # package.json parser
        package_json:
          # use parser
          enabled: true
          # matching criteria
          match: equal("package.json")
        # yarn.lock parser
        yarn_lock:
          # use parser
          enabled: true
          # matching criteria
          match: equal("yarn.lock")
        # package.json yarn environment parser
        yarn_package_json_env:
          # use parser
          enabled: false
          # matching criteria
```

```
        match: equal("package.json")
        # parser properties
        properties:
          # path to yarn for resolve
          yarn-path: yarn
          # args for yarn tool
          yarn-args: ""
      # pnpm-lock.yaml parser
      pnpm_lock_yaml:
        # use parser
        enabled: true
        # matching criteria
        match: equal("pnpm-lock.yaml")
      # package.json pnpm environment parser
      pnpm_package_json_env:
        # use parser
        enabled: false
        # matching criteria
        match: equal("package.json")
        # parser properties
        properties:
          # path to npm for resolve
          pnpm-path: pnpm
          # args for pnpm tool
          pnpm-args: ""
      # bun.lock parser
      bun_lock:
        # use parser
        enabled: true
        # matching criteria
        match: equal("bun.lock")
      # package.json bun environment parser
      bun_env:
        # use parser
        enabled: false
        # matching criteria
        match: equal("package.json")
        # parser properties
        properties:
          # path to bun for resolve
          bun-path: bun
          # args for bun tool
          bun-args: ""
  # Objective-C
  objective_c:
    # Use Objective-C parsers
    enabled: true
    # Objective-C parsers
    parsers:
      # Podfile parser
      podfile:
        # use parser
```

```
        enabled: true
        # matching criteria
        match: equal("Podfile")
      # Podfile.lock parser
      podfile_lock:
        # use parser
        enabled: true
        # matching criteria
        match: equal("Podfile.lock")
      # .podspec parser
      podspec:
        # use parser
        enabled: true
        # matching criteria
        match: extension(".podspec")
  # PHP
  php:
    # Use PHP parsers
    enabled: true
    # PHP parsers
    parsers:
      # composer.json parser
      composer_json:
        # use parser
        enabled: true
        # matching criteria
        match: equal("composer.json")
      # composer.lock parser
      composer_lock:
        # use parser
        enabled: true
        # matching criteria
        match: equal("composer.lock")
      # composer environment parser
      composer_env:
        # use parser
        enabled: false
        # matching criteria
        match: equal("composer.json")
        # parser properties
        properties:
          # path to composer for resolve
          composer-path: composer
          # pass args to composer tool
          composer-args: ""
  # Python
  python:
    # Use Python parsers
    enabled: true
    # Python parsers
    parsers:
      # pip-resolved-dependencies.txt parser
```

```
        pip-resolved-dependencies_txt:
          # use parser
          enabled: true
          # matching criteria
          match: equal("pip-resolved-dependencies.txt")
        # pip environment parser
        pip_env:
          # use parser
          enabled: false
          # matching criteria
          match: equal("codescoring_pip_for_freeze")
          # parser properties
          properties:
            # path to pip for resolve
            pip-path: pip
            # args for pip tool
            pip-args: ""
        # pipdeptree parser
        pipdeptree:
          # use parser
          enabled: true
          # matching criteria
          match: equal("pipdeptree.txt")
        # pipdeptree environment parser
        pipdeptree_env:
          # use parser
          enabled: false
          # matching criteria
          match: equal("codescoring_pipdeptree")
          # parser properties
          properties:
            # path to pipdeptree for resolve
            pipdeptree-path: pip
            # args for pipdeptree tool
            pipdeptree-args: ""
        # Pipfile parser
        pipfile:
          # use parser
          enabled: true
          # matching criteria
          match: equal("Pipfile")
        # Pipfile.lock parser
        pipfile_lock:
          # use parser
          enabled: true
          # matching criteria
          match: equal("Pipfile.lock")
        # poetry.lock parser
        poetry_lock:
          # use parser
          enabled: true
          # matching criteria
```

```
    match: equal("poetry.lock")
  # pyproject.toml poetry environment parser
  poetry_pyproject_toml_env:
    # use parser
    enabled: false
    # matching criteria
    match: equal("pyproject.toml")
    # parser properties
    properties:
      # path to poetry for resolve
      poetry-path: poetry
      # args for poetry tool
      poetry-args: ""
  # uv.lock parser
  uv_lock:
    # use parser
    enabled: true
    # matching criteria
    match: equal("uv.lock")
  # pyproject.toml uv environment parser
  uv_env:
    # use parser
    enabled: false
    # matching criteria
    match: equal("pyproject.toml")
    # parser properties
    properties:
      # path to uv for resolve
      uv-path: uv
      # args for uv tool
      uv-args: ""
  # pdm.lock parser
  pdm_lock:
    # use parser
    enabled: true
    # matching criteria
    match: equal("pdm.lock")
  # pylock.toml parser (PEP 751)
  pylock_toml:
    # use parser
    enabled: true
    # matching criteria
    match: equal("pylock.toml")
  # pyproject.toml pdm environment parser
  pdm_env:
    # use parser
    enabled: false
    # matching criteria
    match: equal("pyproject.toml")
    # parser properties
    properties:
      # path to pdm for resolve
```

```
          pdm-path: pdm
          # args for pdm tool
          pdm-args: ""
      # pyproject.toml parser
      pyproject_toml:
        # use parser
        enabled: true
        # matching criteria
        match: equal("pyproject.toml")
      # requirements.txt parser
      requirements_txt:
        # use parser
        enabled: true
        # matching criteria
        match: match(".*require[^/]*(/)?[^/]*.(txt|pip)$")
      # setup.py parser
      setup_py:
        # use parser
        enabled: true
        # matching criteria
        match: equal("setup.py")
    # technology properties
    properties:
      # python version
      python-version: ""
  # Ruby
  ruby:
    # Use Ruby parsers
    enabled: true
    # Ruby parsers
    parsers:
      # Gemfile parser
      gemfile:
        # use parser
        enabled: true
        # matching criteria
        match: equal("Gemfile") || equal("gems.rb")
      # Gemfile.lock parser
      gemfile_lock:
        # use parser
        enabled: true
        # matching criteria
        match: equal("Gemfile.lock") || equal("gems.locked")
      # .gemspec parser
      gemspec:
        # use parser
        enabled: true
        # matching criteria
        match: extension(".gemspec")
  # Rust
  rust:
    # Use Rust parsers
```

```
      enabled: true
      # Rust parsers
      parsers:
        # cargo.lock parser
        cargo_lock:
          # use parser
          enabled: true
          # matching criteria
          match: equal("cargo.lock")
        # cargo.toml parser
        cargo_toml:
          # use parser
          enabled: true
          # matching criteria
          match: equal("cargo.toml")
    # conda
    conda:
      # Use Conda parsers
      enabled: true
      # Conda parsers
      parsers:
        # Conda-lock parser
        conda-lock_yml:
          # use parser
          enabled: true
          # matching criteria
          match: equal("conda-lock.yml")
        # Conda env parser
        conda_yml_env:
          # use parser
          enabled: false
          # matching criteria
          match: equal("environment.yml") ||
equal("environment.yaml") || equal("meta.yml") ||
equal("meta.yaml")
          # parser properties
          properties:
            # path to conda-lock for resolve
            conda-lock-path: conda-lock
            # args for conda tool
            conda-args: ""
    # swift
    swift:
      # Use swift parsers
      enabled: true
      # swift parsers
      parsers:
        # Package.resolved parser
        package_resolved:
          # use parser
          enabled: true
          # matching criteria
```

```
          match: equal("Package.resolved")
        # Package.swift parser
        package_swift:
          # use parser
          enabled: true
          # matching criteria
          match: equal("Package.swift")
        # Package.swift env parser
        package_swift_env:
          # use parser
          enabled: false
          # matching criteria
          match: equal("Package.swift")
          # parser properties
          properties:
            # path to swift for resolve
            swift-path: swift
            # args for swift tool
            swift-args: ""
  # scan secrets
  secrets:
    # gitleaks options
    gitleaks:
      # path to baseline with issues that can be ignored
      baseline-path: ""
      # only enable specific rules by id
      enable-rule: [ ]
      # path to .gitleaksignore file or folder containing one
      gitleaks-ignore-path: .
      # path to gitleaks binary to be used during scanning
      gitleaks-path: gitleaks
      # path to gitleaks config
      gitleaks-config: ""
      # ignore gitleaks:allow comments
      ignore-gitleaks-allow: false
      # log level (trace, debug, info, warn, error, fatal)
      log-level: info
      # allow recursive decoding up to this depth (default
\"0\", no decoding is done)
      max-decode-depth: 0
      # files larger than this will be skipped
      max-target-megabytes: 0
      # suppress banner
      no-banner: false
      # turn off color for verbose output
      no-color: false
      # redact secrets from logs and stdout. To redact only
parts of the secret just apply a percent value from 0..100. For
example --redact=20 (default 100%)
      redact: "0"
      # show verbose output from scan
      verbose: false
```

```
    # trufflehog options
    trufflehog:
      # path to trufflehog binary to be used during scanning
      trufflehog-path: trufflehog
      # path to trufflehog config to be used during scanning
      trufflehog-config: ""
      # number of concurrent workers
      concurrency: 10
      # don't verify the results
      no-verification: false
      # only output verified results
      only-verified: false
      # path to file with newline separated regexes for files
to include in scan
      include-paths: ""
      # path to file with newline separated regexes for files
to exclude in scan
      exclude-paths: ""
      # log level (debug, info, warn, error)
      trufflehog-log-level: info
    # output report in gitlab format
    gl-secrets-report: false
    # output file for report in gitlab format
    gl-secrets-report-filename: gl-secrets-report.json
    # git repository scanning options (used by 'secrets
gitleaks git' and 'secrets trufflehog git')
    git:
      # git branch, tag, or commit ref to scan (leave empty to
scan all refs)
      git-ref: ""
      # limit scan to this many commits from the tip (0 = no
limit)
      git-depth: 0
      # auth token for private repository access (passed via
environment, not CLI args)
  # scan archives options
  scan-archives:
    # scan archives
    scan: false
    # archive scanning depth
    depth: 1
## stats options
stats:
  # Report format. Supported formats: coloredtable, table,
text, junit, sarif, csv. Default output coloredtable to
console.
  format: coloredtable,junit>>junit.xml
  # Policy alerts report format. Supported formats:
coloredtable, table, text, json, csv. Default output
coloredtable to console.
  alerts-format: coloredtable
  # Group vulnerabilities by field
```

```
  group-vulnerabilities-by: vulnerability
  # Sort vulnerabilities by fields
  sort-vulnerabilities-by: -cvss3,-
cvss2,fixedversion,vulnerability,cwes,links,affect
## cli options
cli:
  # CodeScoring server url
  api_url: https://example_url
  # API token for integration with CodeScoring server
  api_token: example_token
  # Localization language (en|ru). Default: en
  localization: en
```

#Приоритет настроек

Поскольку параметры запуска агента можно настроить несколькими способами, при одновременном использовании двух и более способов агент будет принимать параметры в следующем порядке приоритетов:

1. Значение команды scan-technology (если она используется);
2. Значение флага команды;
3. Значение переменной окружения;
4. Значение из конфиг-файла.

#Настройка через переменные окружения

Параметры запуска консольного агента можно настроить через переменные окружения. Для настройки через переменные окружения используется структура конфигурационного файла.

#Формирование переменных окружения

1. **Префикс переменной**: Все переменные окружения начинаются с префикса `JOHNNY_`.
2. **Путь секций**: Переменная формируется на основе пути секций в конфигурационном файле. Разделители секций заменяются символом `_`.
3. **Замена символов**: Символы `"."` и `"-"` в именах секций также преобразуются в символ `_`.

#Пример

Рассмотрим пример настройки флага `block-on-empty-result` для блокирования сборки при получении пустого результата:

- **Путь в конфигурационном файле**: `scan.general.block-on-empty-result`;
- **Переменная окружения**: `JOHNNY_SCAN_GENERAL_BLOCK_ON_EMPTY_RESULT`;

Таким образом, для изменения значения этого параметра через переменные окружения, необходимо задать переменную `JOHNNY_SCAN_GENERAL_BLOCK_ON_EMPTY_RESULT` с нужным значением.

#Приоритет настроек

Поскольку параметры запуска агента можно настроить несколькими способами, при одновременном использовании двух и более способов агент будет принимать параметры в следующем порядке приоритетов:

1. Значение команды scan-technology (если она используется);
2. Значение флага команды;
3. Значение переменной окружения;
4. Значение из конфиг-файла.

#Команда сканирования

Запуск агента производится при помощи команды `scan` с возможными вариантами сканирования:

- `scan dir` – сканирование директории;
- `scan file` – сканирование файла;
- `scan image` – сканирование контейнерного образа;
- `scan bom` – сканирование SBOM;
- `scan <technology>` - сканирование директории с применением настроек для указанной технологии;
- `scan build` – сканирование сборки.

#Опции запуска

Доступные и необходимые опции запуска агента для сканирования можно посмотреть при помощи флага `help`.

```
$ ./johnny scan --help
johnny - CLI tool for dependency analysis for vulnerabilities
and license compliance issues. Works in connection with
CodeScoring SCA.
CodeScoring website: https://codescoring.ru
Documentation: https://docs.codescoring.ru

Exit codes:
- 0: successful run, no issues
- 1: some issues found, action required
- 2: run failure
- 3: successful run, no result
- 4: signing or verification failure
- 5: BOM validation error

Version: 2026.20.0
```

```
Usage:
   scan [command]

Scan Technologies Commands:
  clang       Scan clang
  conda       Scan conda
  csharp      Scan csharp
  go          Scan go
  java        Scan java
  js          Scan js
  objective_c Scan objective_c
  php         Scan php
  python      Scan python
  ruby        Scan ruby
  rust        Scan rust
  swift       Scan swift

Additional Commands:
  bom         Scan bom
  dir         Scan directory
  file        Scan file
  image       Scan image

Flags:
  -h, --help      help for scan
  -v, --version   version for scan

Global Flags:
      --alerts-format string              Alerts format.
Supported formats: coloredtable, table, text, csv, json.
Default output to console. Supports multiformat. Example:
'coloredtable,csv>>csv.csv'  (default "coloredtable")
      --api_token string                  API token for
integration with CodeScoring server (required if api_url is
set)
      --api_url string                    CodeScoring server
url (e.g. https://codescoring.mycompany.com) (required if
api_token is set)
      --block-on-empty-result             Block on empty result
      --bom-format string                 Bom format. Supported
formats:
cyclonedx_v1_4_json,cyclonedx_v1_5_json,cyclonedx_v1_6_ext_json
,cyclonedx_v1_6_json,cyclonedx_v1_7_json (default
"cyclonedx_v1_6_json")
      --bom-path string                   Path for save bom
file (default "bom.json")
      --bun-args string                   pass flags to 'bun
install --lockfile-only'
      --bun-path string                   Path to bun for
resolve (default "bun")
      --bun-resolve                       Enable resolve using
bun
```

```
      --cg-lang string                      Language to parse
call graph with. Supported languages:
csharp,go,java,kotlin,python,javascript
      --cg-path string                      Path to call graph
for vulnerability reachability analysis
      --cloud-resolve                       Activate cloud
resolve
      --composer-args string                pass flags to
'composer install'
      --composer-path string                Path to composer for
resolve (default "composer")
      --composer-resolve                    Enable resolve using
composer
      --conda-args string                   pass flags to conda
      --conda-lock-path string              Path to conda-lock
for resolve (default "conda-lock")
      --conda-resolve                       Enable resolve using
conda-lock
      --config string                       Config file (default
"codescoring-johnny-config.yaml")
      --create-project                      Create project in
CodeScoring if not exists
      --create-project-group                Create group in
CodeScoring if not exists
      --debug                               Output detailed log
      --dotnet-args string                  pass flags to 'dotnet
restore'
      --dotnet-path string                  Path to dotnet for
resolve (default "dotnet")
      --dotnet-resolve                      Enable resolve using
dotnet
      --exclude-envs string                 Exclude the listed
dependency environments (scopes) from the result, comma-
separated, e.g. --exclude-envs=test,dev
  -f, --format string                       Report format.
Supported formats: coloredtable, table, text, junit, sarif,
csv, gl-dependency-scanning-report, gl-code-quality-report.
Default output to console. Supports multiformat. Example:
'coloredtable,junit>>junit.xml'  (default "coloredtable")
      --gdt-match string                    Section in gradle
dependency tree for scan. By default - parse all sections
      --go-path string                      Path to go for
resolve (default "go")
      --go-resolve                          Enable resolve using
go
      --gradle-args string                  pass flags to 'gradle
dependencies'
      --gradle-path string                  Path to gradle for
resolve (default "./gradlew")
      --gradle-resolve                      Enable resolve using
gradle
  -g, --group-vulnerabilities-by string     Group vulnerabilities
```

```
by. Supported kinds 'vulnerability', 'affect' (default
"vulnerability")
      --ignore stringArray                Ignore paths (--
ignore first --ignore "/**/onem?re")
      --ignores-format string             Displays the ignores
of the specified project with formatting. Supported formats:
coloredtable, table, text, csv, json. Default output to
console. Supports multiformat. Example:
'coloredtable,csv>>csv.csv'  (default "coloredtable")
      --include-envs string               Include only the
listed dependency environments (scopes) in the result, comma-
separated, e.g. --include-envs=compile,runtime
      --license string                    Project license code
      --localization string               Localization language
(en|ru) (default "en")
      --maven-args string                 pass flags to 'mvn
dependency:tree'
      --maven-path string                 Path to mvn for
resolve (default "mvn")
      --maven-resolve                     Enable resolve using
mvn
      --no-summary                        Do not print summary
      --no-wait                           No wait analysis
results
      --npm-args string                   pass flags to 'npm
install'
      --npm-path string                   Path to npm for
resolve (default "npm")
      --npm-resolve                       Enable resolve using
npm
      --only-hashes                       Search only for
direct inclusion of dependencies using file hashes
      --pdm-args string                   pass flags to 'pdm
lock'
      --pdm-path string                   Path to pdm for
resolve (default "pdm")
      --pdm-resolve                       Enable resolve using
pdm
      --pip-args string                   pass flags to 'pip
freeze'
      --pip-path string                   Path to pip for
resolve (default "pip")
      --pip-resolve                       Enable resolve using
pip
      --pipdeptree-args string            pass flags to
'pipdeptree'
      --pipdeptree-path string            Path to pipdeptree
for resolve (default "pipdeptree")
      --pipdeptree-resolve                Enable resolve using
pipdeptree
      --pnpm-args string                  pass flags to 'pnpm
install'
```

```
      --pnpm-path string                    Path to pnpm for
resolve (default "pnpm")
      --pnpm-resolve                        Enable resolve using
pnpm
      --poetry-args string                  pass flags to 'poetry
debug resolve'
      --poetry-path string                  Path to poetry for
resolve (default "poetry")
      --poetry-resolve                      Enable resolve using
poetry
      --policy-ignores                      Displays the ignores
      --progress-bar string                 Progress bar formats:
spinner,text
      --project string                      Project name in
CodeScoring
      --project-categories string           Category names for
created project in CodeScoring (comma-separated list)
      --project-group string                Project group for
created or added project in CodeScoring
      --project-proprietor string           Proprietor for
created project in CodeScoring
      --python-version string               Python version
      --save-results                        Save results to
CodeScoring. Used just together with project name
      --sbt-args string                     pass flags to 'sbt
dependencyTree'
      --sbt-path string                     Path to sbt for
resolve (default "sbt")
      --sbt-resolve                         Enable resolve using
sbt
      --scan-archives                       Scan archives.
Supported types: '.jar', '.rar', '.tar', '.tar.bz2', '.tbz2',
'.tar.gz', '.tgz', '.tar.xz', '.txz', '.war', '.zip', '.aar',
'.egg', '.hpi', '.nupkg', '.whl'
      --scan-depth int                      Archive scanning
depth (default 1)
  -s, --sort-vulnerabilities-by string    Sort vulnerabilities
by. Comma separated field names. For DESC - write field name
with prefix '-'.
                                            FieldNames:
'vulnerability', 'fixedversion', 'cvss2', 'cvss3', 'cwes',
'links', 'affect' (default "-cvss3,-
cvss2,fixedversion,vulnerability,cwes,links,affect")
      --stage string                        Policy stage (build,
dev, source, stage, test, prod, proxy) (default "build")
      --swift-args string                   pass flags to 'swift
build
      --swift-path string                   Path to swift for
resolve (default "swift")
      --swift-resolve                       Enable resolve using
swift
  -t, --timeout uint16                      Timeout of analysis
```

```
results waiting in seconds (default 3600)
      --uv-args string                    pass flags to 'uv
lock'
      --uv-path string                    Path to uv for
resolve (default "uv")
      --uv-resolve                        Enable resolve using
uv
      --with-hashes                       Search for direct
inclusion of dependencies using file hashes
      --yarn-args string                  pass flags to 'yarn
install'
      --yarn-path string                  Path to yarn for
resolve (default "yarn")
      --yarn-resolve                      Enable resolve using
yarn

Use " scan [command] --help" for more information about a
command.
```

В параметре `--api_url` должен быть указан полный адрес on-premise платформы. Значение для `--api_token` можно взять в профиле пользователя платформы.

Указание параметра `--project` позволит при сканировании применить политики, относящиеся к выбранному проекту.

Для указания пути к файлу сохранения SBOM необходимо добавить параметр `--bom-path` в запрос или назначить переменную `bom-path` в config-файле. По умолчанию SBOM сохраняется в директории запуска в файл `bom.json`.

#Результаты работы

В зависимости от результата работы и параметров запуска агент возвращает соответствующий exit code:

- **0** – успешное сканирование, проблемы не были выявлены;
- **1** – в результате сканирования найдены проблемы, соответствующие настроенным политикам безопасности, необходимо действие пользователя;
- **2** – ошибка сканирования;
- **3** – пустой результат, не были найдены артефакты для анализа. Возвращается только если параметр `--block-on-empty-result` имеет значение `true`.

#Приоритет настроек

Поскольку параметры запуска агента можно настроить несколькими способами, при одновременном использовании двух и более способов агент будет принимать параметры в следующем порядке приоритетов:

1. Значение команды scan-technology (если она используется);

2. Значение флага команды;
3. Значение переменной окружения;
4. Значение из конфиг-файла.

#Запуск без участия платформы

Если параметры `--api_url` и `--api_token` не заданы, запуск сканирования будет производиться без взаимодействия с платформой CodeScoring. В результате сканирования будет сгенерирован файл SBOM, содержащий только список компонентов и их версий без обогащения дополнительной информацией.

#Сканирование директории

Сканирование директории производится при помощи субкоманды `scan dir`.

При запуске агент:

1. Рекурсивно проходит по всему содержимому указанной директории (если указан конкретный манифест, обрабатывает только его)
2. Идентифицирует файлы манифестов и разбирает их
3. Хеширует каждый файл (при запуске с `--with-hashes`)
4. Формирует запрос к платформе
5. После получения результата показывает суммарную информацию по найденным манифестам, зависимостям, уязвимостям, сработавшим политикам и более подробную информацию по каждой уязвимости и сработавшей политике
6. Дополнительно в текущей директории формируется файл `bom.json`, содержащий полный Software Bill of Materials в формате **CycloneDX**.

В зависимости от результата работы и параметров запуска агент возвращает соответствующий exit code.

По умолчанию агент проходит по содержимому директории рекурсивно (включая вложенные директории). Для нерекурсивного сканирования необходимо добавить параметр `--no-recursion` к команде `scan dir`.

Пример запуска команды

```
./johnny scan dir . \
--api_token <api_token> \
--api_url <api_url> \
--ignore .tmp --ignore fixtures --ignore .git
```

#Параметры команды

Команда **scan dir** имеет три уникальных параметра, помимо общих настроек команды сканирования:

- `--branch-or-tag` – ссылка на ветку или тег репозитория в формате `^refs/(heads|tags)/.+` (например, `refs/tags/v1.0`);
- `--commit` – указание хэша коммита;
- `--no-recursion` – отключение рекурсивного сканирования каталогов.

Для сводки доступных параметров команды и инструкции по использованию можно вызвать команду с флагом `-h, --help`.

#Сканирование файла

При необходимости сканирования отдельного манифеста внутри директории можно использовать команду `scan file`.

При запуске агент:

1. Идентифицирует формат указанного файла и производит разбор содержимого.
2. Формирует запрос к платформе для анализа содержимого.
3. После получения результатов отображает общую информацию о найденных манифестах, зависимостях, уязвимостях и сработавших политиках.
4. Дополнительно в текущей директории создается файл `bom.json`, содержащий полный Software Bill of Materials в формате **CycloneDX**.

В зависимости от параметров запуска агент возвращает соответствующий exit code:

- **0** – успешное сканирование, проблемы не были выявлены;
- **1** – в результате сканирования найдены проблемы, соответствующие настроенным политикам безопасности, необходимо действие пользователя;
- **2** – ошибка сканирования;
- **3** – пустой результат, не были найдены артефакты для анализа. Возвращается только если параметр `--block-on-empty-result` имеет значение `true`.

Пример запуска команды

Для сканирования только одного файла без обработки вложенных директорий или других манифестов, необходимо указать путь к файлу при запуске команды.

```
./johnny scan file path/to/file \
--api_token <api_token> \
--api_url <api_url>
```

#Параметры команды

Команда **scan file** имеет три уникальных параметра, помимо общих настроек команды сканирования:

- `--branch-or-tag` – ссылка на ветку или тег репозитория в формате `^refs/(heads|tags)/.+` (например, `refs/tags/v1.0`);
- `--commit` – указание хэша коммита;
- `--parser` – используемый парсер.

Для сводки доступных параметров команды и инструкции по использованию можно вызвать команду с флагом `-h, --help`.

#Доступные парсеры

Технология	Парсеры
Conda	`conda.conda-lock_yml`, `conda.conda_yml_env`
Ruby	`ruby.gemfile`, `ruby.gemfile_lock`, `ruby.gemspec`
C#	`csharp.packages_lock_json`, `csharp.project_json`, `csharp.project_lock_json`, `csharp.dependencyreport_json`, `csharp.paket_dependencies`, `csharp.nuspec`, `csharp.csproj`, `csharp.packages_config`, `csharp.dotnet_csproj_env`, `csharp.project_assets_json`, `csharp.paket_lock`
PHP	`php.composer_json`, `php.composer_lock`, `php.composer_env`
Python	`python.poetry_pyproject_toml_env`, `python.requirements_txt`, `python.pipfile`, `python.poetry_lock`, `python.pip-resolved-dependencies_txt`, `python.setup_py`, `python.pipfile_lock`, `python.pyproject_toml`, `python.pip_env`, `python.pipdeptree`, `python.uv_lock`
C	`clang.conan_lock`, `clang.conanfile_txt`, `clang.conanfile_py`
Go	`go.go_mod`, `go.go_sum`, `go.go_mod_env`

Технология	Парсеры
Objective-C	`objective_c.podfile`, `objective_c.podfile_lock`, `objective_c.podspec`
Rust	`rust.cargo_toml`, `rust.cargo_lock`
Java	`java.pom_xml`, `java.ivy_xml`, `java.maven-dependency-tree_txt`, `java.build_gradle_env`, `java.gradle-dependency-tree_txt`, `java.gradle_kts`, `java.gradle_lockfile`, `java.maven_pom_xml_env`, `java.jar`, `java.gradle`, `java.scala_build_sbt_env`, `java.scala-dependency-tree_txt`
JS	`js.package_json`, `js.yarn_package_json_env`, `js.yarn_lock`, `js.pnpm_package_json_env`, `js.pnpm_lock_yaml`, `js.npm_package_json_env`, `js.package-lock_json`, `js.npm-shrinkwrap_json`

#Сканирование архивов

Для сканирования архивов на предмет наличия манифестов используется флаг `--scan-archives`.

По умолчанию сканирование архивов работает только на один уровень вложенности. Для указания глубины сканирования необходимо добавить в команду параметр `--scan-depth` или указать в config-файле переменную `depth` в секции `scan-archives`.

Пример команды для сканирования архивов:

```
./johnny scan dir . \
--api_token <api_token> \
--api_url <api_url> \
--ignore .tmp --ignore fixtures --ignore .git \
--scan-archives \
--scan-depth 2
```

Поддерживаемые форматы архивов:

- `.jar`
- `.rar`
- `.tar`
- `.tar.bz2`
- `.tbz2`
- `.tar.gz`
- `.tgz`

- `.tar.xz`
- `.txz`
- `.war`
- `.zip`
- `.aar`
- `.egg`
- `.hpi`
- `.nupkg`
- `.whl`

#Сканирование контейнерных образов

Агент поддерживает функциональность сканирования образов в стандартах OCI и Docker и может быть запущен одним из перечисленных способов с указанием:

- пути до **tar**-архива созданного с использованием **docker save**:

```
./johnny scan image ./my_own.tar \
--api_url <api_url> \
--api_token <api_token>
```

- названия образа находящегося в демоне **Docker**, **Podman**:

```
./johnny scan image docker:python:3.9 \
--api_url <api_url> \
--api_token <api_token>
```

- названия образа из публичного **Docker HUB**:

```
./johnny scan image python:3.9 \
--api_url <api_url> \
--api_token <api_token>
```

- названия образа из приватного **registry**:

Перед работой с приватным репозиторием нужно выполнить команду `docker login`

```
./johnny scan image pvt_registry/johnny-depp:<version> \
--api_url <api_url> \
--api_token <api_token>
```

Альтернативно можно авторизоваться в приватном registry с помощью переменных окружения:

- `JOHNNY_REGISTRY_AUTH_AUTHORITY` - URL на registry (к примеру "docker.io", "localhost:5000" и т.д.);
- `JOHNNY_REGISTRY_AUTH_LOGIN` - логин;
- `JOHNNY_REGISTRY_AUTH_PASSWORD` - пароль;
- `JOHNNY_REGISTRY_AUTH_TOKEN` - токен;

или через аналогичные переменные в config-файле:

- `authority`;
- `login`;
- `password`;
- `token`.

Примечание: токен и логин с паролем взаимозаменяемы.

#Сканирование файловой системы внутри образа

Для выполнения сканирования файлов внутри образа необходимо добавить в команду параметр `--scan-files` или указать в config-файле переменную `scan-files` в секции `image`.

При сканировании файловой системы можно использовать параметр `--ignore` для исключения определенных файлов из анализа. Например:

```
./johnny scan image ./my_own.tar \
--api_url <api_url> \
--api_token <api_token> \
--scan-files \
--ignore "**/node_modules"
```

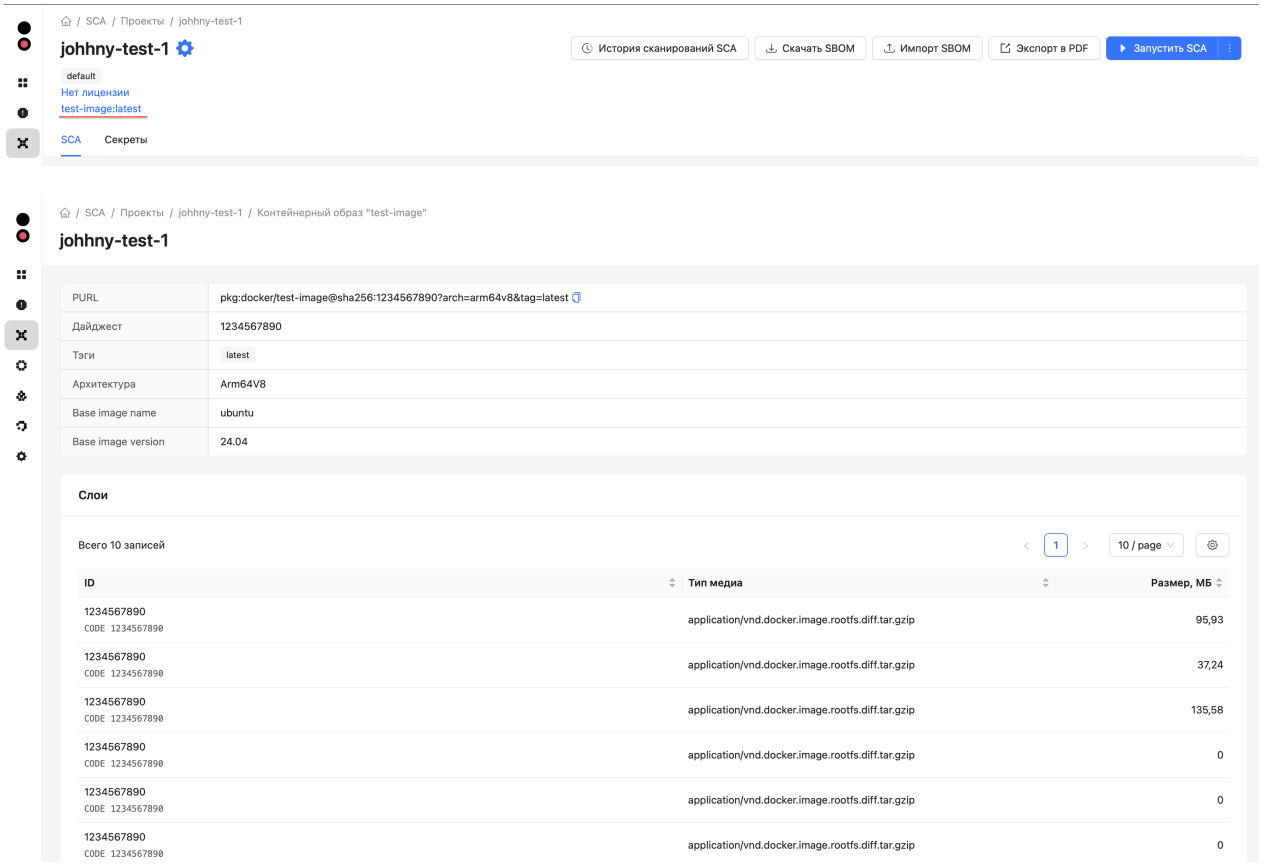
#Параметры команды

Команда **scan image** имеет четыре уникальных параметра, помимо общих настроек команды сканирования:

- `--hash` – указание хэша образа;
- `--scan-files` – сканирование файлов в образе.
- `--branch-or-tag` – ссылка на ветку или тег репозитория в формате `^refs/(heads|tags)/.+` (например, `refs/tags/v1.0`);
- `--commit` – указание хэша коммита;

Для сводки доступных параметров команды и инструкции по использованию можно вызвать команду с флагом `-h, --help`.

При сохранении результатов сканирования образа, в проекте можно посмотреть подробную информацию об образе и его слоях:



#Сканирование сборки для языков C и C++

В случае, если для сборки C/C++ проекта не используется пакетный менеджер Conan и соответствующие манифесты, для получения списка используемых библиотек Johnny можно запустить в специальном режиме для анализа вывода процесса сборки.

Johnny анализирует процесс сборки, используя флаги компилятора и выявляя использованные библиотеки. Далее с помощью системного кэша определяется местоположение библиотек и их источник.

Версия используемой библиотеки может быть найдена, если библиотека была подключена статически. В таком случае агент анализирует `.pc` файл, содержащий метаданные о компоненте.

#Пример работы

В проект добавляется JSON-файл `buildConfig`, описывающий последовательность команд для сборки. Например:

```
{
 //  Commands for execution
 "commands": [
    {
   // Command
   "command": "make",
```

```
    // Command execute parameters
    "flags_and_args": "clean"
   },
   {
    "command": "./configure"
   },
   {
    "command": "make",
    // analyze this command
    "do_analyze": true
   }
  ]
}
```

Далее вызывается команда анализа сборки и указывается путь до конфиг-файла:

```
./johnny scan build ./buildConfig.json
```

В результате работы агент отображает один из кодов:

- **0** – успешный анализ, не было выявлено проблем в сборке;
- **1** – в результате сканирования найдены проблемы, соответствующие настроенным политикам безопасности, необходимо действие пользователя;
- **2** – анализ не удался;
- **3** – пустой результат, не были найдены артефакты для анализа. Возвращается только если параметр `--block-on-empty-result` имеет значение `true`.

Важно: на данный момент команда поддерживает только Debian-based и RPM-based дистрибутивы Linux.

#Параметры команды

Команда **scan build** имеет шесть уникальных параметров, помимо общих настроек команды сканирования:

- `--build-result` – вводные данные представляют собой результат предыдущего процесса сборки, включая скомпилированные артефакты;
- `--lib-versions` – путь к JSON-файлу со списком версий анализируемых библиотек;
- `--output` – путь к файлу, в котором будут сохранены результаты сканирования;
- `--unresolved-file` – путь к файлу, где будут сохранены сведения о библиотеках с неразрешенными версиями.
- `--branch-or-tag` – ссылка на ветку или тег репозитория в формате `^refs/(heads|tags)/.+` (например, `refs/tags/v1.0`);
- `--commit` – указание хэша коммита.

Для сводки доступных параметров команды и инструкции по использованию можно вызвать команду с флагом `-h, --help`.

#Сканирование с использованием eBPF

eBPF (extended Berkeley Packet Filter) – это технология в ядре Linux, которая позволяет безопасно запускать пользовательский код в ответ на события в системе, такие как сетевой трафик, системные вызовы или действия процессов.

Особенность команды `scan build ebpf` состоит в том, что анализ выполняется не по протоколу сборочного инструмента, а путем мониторинга запускаемых процессов и их параметров через механизм eBPF.

Важно: для обеспечения работы с механизмом eBPF, запуск команды должен выполняться с правами **root-пользователя**.

#Пример работы

Параметры и результаты работы команды аналогичны команде `scan build`.

```
./johnny scan build ebpf ./buildConfig.json
```

#Сканирование SBOM

При необходимости сканирования существующего перечня программных компонентов (Software Bill of Materials, SBOM) в формате **CycloneDX** можно использовать команду `scan bom`.

При запуске агент:

1. Валидирует передаваемый SBOM на соответствие схеме указанной версии.
2. Производит разбор указанного SBOM.
3. Формирует запрос к платформе для анализа содержимого.
4. После завершения анализа отображает в консоли сводную информацию о результатах, а также таблицы с найденными уязвимостях и сработавшими политиками.
5. Дополнительно в текущей директории создается файл `bom.json`, содержащий дополненный SBOM.

В зависимости от параметров запуска агент возвращает соответствующий exit code:

- **0** – успешное сканирование, проблемы не были выявлены;
- **1** – в результате сканирования найдены проблемы, соответствующие настроенным политикам безопасности, необходимо действие пользователя;

- **2** – ошибка сканирования;
- **3** – пустой результат, не были найдены артефакты для анализа. Возвращается только если параметр `--block-on-empty-result` имеет значение `true`.
- **5** - ошибка валидации SBOM.

Пример запуска команды

Для сканирования SBOM необходимо указать путь к нему при запуске команды.

```
./johnny scan bom path/to/bom \
--api_token <api_token> \
--api_url <api_url>
```

#Параметры команды

Команда **scan bom** имеет два уникальных параметра, помимо общих настроек команды сканирования:

- `--branch-or-tag` – ссылка на ветку или тег репозитория в формате `^refs/(heads|tags)/.+` (например, `refs/tags/v1.0`);
- `--commit` – указание хэша коммита.

#Сканирование технологии

Для более удобной работы с различными экосистемами агент позволяет сканировать отдельные технологии с набором предварительно заданных настроек. Сканирование в таком случае производится при помощи субкоманды `scan <technology>`.

Поведение агента при сканировании аналогично поведению при выполнении команды `scan dir`, однако имеет следующие отличия:

1. Обход директории всегда выполняется нерекурсивно (как при использовании флага `--no-recursion` в команде `scan dir`);
2. Обрабатываются только манифесты, принадлежащие к выбранной технологии;
3. Используются все парсеры выбранной технологии, включая разрешение зависимостей в окружении. Прочие настройки при этом игнорируются;

#Список поддерживаемых технологий

Технологии указаны так же, как они используются в команде `scan <technology>`:

- clang
- conda
- csharp

- go
- java
- js
- objective_c
- php
- python
- ruby
- rust

Пример запуска команды

```
./johnny scan java . \
--api_token <api_token> \
--api_url <api_url>
```

Для сводки доступных параметров команды и инструкции по использованию можно вызвать команду с флагом `-h, --help`.

#Сканирование на наличие секретов

Важно

Для использования данного функционала платформа должна иметь активный модуль CodeScoring.Secrets.

Сканирование на наличие секретов выполняется с помощью следующих команд:

- `johnny secrets gitleaks dir` — сканирование файлов в указанной директории;
- `johnny secrets gitleaks git` — сканирование истории локального git-репозитория;
- `johnny secrets trufflehog dir` — сканирование файлов в указанной директории с помощью Trufflehog;
- `johnny secrets trufflehog git` — сканирование истории локального git-репозитория с помощью Trufflehog.

Важно: агент работает только с версией Gitleaks 8.19.0 и выше, и с версией Trufflehog 3.93.8 и выше.

При запуске агент:

1. Анализирует файлы в указанной директории или историю коммитов репозитория на наличие секретов (пароли, токены, ключи доступа и т. д.).

- исключает файлы и каталоги, указанные в `.gitleaksignore`;

- игнорирует секреты, зафиксированные в отчете Gitleaks, если задан `baseline-path`.

1. Формирует результаты по найденным секретами, при необходимости сохраняет их на платформе CodeScoring и создает отчет в формате GitLab.

#Сканирование git-репозитория

Режим `git` позволяет сканировать историю коммитов локального git-репозитория. В отличие от режима `dir`, который анализирует текущее состояние файлов, режим `git` проверяет секреты во всех коммитах репозитория или в ограниченном диапазоне.

#Пример запуска команды для Gitleaks

```
johnny secrets gitleaks git /path/to/repo \
--gitleaks-path <path-to-gitleaks> \
--api_token <api_token> \
--api_url <api_url> \
--save-results \
--project <project-name> \
--git-ref main \
--git-depth 100
```

#Пример запуска команды для Trufflehog

```
johnny secrets trufflehog git /path/to/repo \
--trufflehog-path <path-to-trufflehog> \
--api_token <api_token> \
--api_url <api_url> \
--save-results \
--project <project-name> \
--git-ref main \
--git-depth 100
```

#Параметры git-режима

Команды `johnny secrets gitleaks git` и `johnny secrets trufflehog git` поддерживают следующие дополнительные параметры:

- `--git-ref` – ветка, тег или коммит git, которые будут сканироваться. Если не задан, сканируются все ссылки. Примеры: `main`, `v1.0.0`, `a1b2c3d`;
- `--git-depth` – ограничение сканирования заданным количеством коммитов от вершины истории. Значение `0` означает отсутствие ограничения.

#Пример конфига для Gitleaks

Пример конфига, который расширяет стандартную конфигурацию, добавляя новое правило со своим регулярным выражением

```
title = “Custom gitleaks config”

[extend]
useDefault = true

[[rules]]
id = “custom-generic-password”
description = “Detected a Generic password”
regex = ‘’‘passw(?:or)d.+’‘’
entropy = 1
```

#Пример запуска команды

```
johnny secrets gitleaks dir . \
--gitleaks-path <path-to-gitleaks> \
--gitleaks-config <path-to-gitleaks-config> \
--api_token <api_token> \
--api_url <api_url> \
--save-results \
--create-project \
--project <project-name> \
--gitleaks-ignore-path .gitleaksignore \
--gl-secrets-report \
--gl-secrets-report-filename secrets-report.json
```

Данная команда запускает сканирование секретов в текущей директории, игнорируя файлы, перечисленные в `.gitleaksignore`, отправляет результаты в платформу CodeScoring, и формирует отчет в формате GitLab, записывая его в `secrets-report.json`.

#Параметры команды

Команды **johnny secrets gitleaks dir**, **johnny secrets gitleaks git**, **johnny secrets trufflehog dir** и **johnny secrets trufflehog git** имеют следующие уникальные параметры:

#Параметры запуска поиска секретов

- `--commit` – хэш коммита, который будет привязан к найденным секретам при сохранении результатов. Используется только для команд `dir`, если инструмент не определяет коммит самостоятельно (например: `--commit a1b2c3d`);
- `--gl-secrets-report` – включение формирования отчета о найденных секретах в формате GitLab;

- `--gl-secrets-report-filename` – имя выходного файла для отчета в формате GitLab (по умолчанию `gl-secrets-report.json`).

#Параметры Gitleaks

- `--gitleaks-path` – путь к исполняемому файлу Gitleaks, который будет использоваться при сканировании. Если не задан, будет выполняться вызов системной команды `gitleaks`;
- `--gitleaks-config` - путь к конфигурационному файлу Gitleaks;
- `--baseline-path` – путь к файлу отчета Gitleaks, который используется в качестве базовой линии для игнорирования ранее найденных секретов;
- `--enable-rule` – список ID правил, которые будут **включены** при сканировании;
- `--gitleaks-ignore-path` – путь к файлу `.gitleaksignore` или директории, содержащей его, для добавления fingerprint найденных ранее секретов;
- `--ignore-gitleaks-allow` – игнорирование комментариев `gitleaks:allow`, которые помечают строки как безопасные для игнорирования;
- `--log-level` – уровень логирования, который контролирует подробность выводимых сообщений. Возможные значения: `trace`, `debug`, `info`, `warn`, `error`, `fatal`;
- `--max-decode-depth` – максимальная глубина рекурсивного декодирования при поиске секретов. Значение `0` отключает декодирование;
- `--max-target-megabytes` – максимальный размер анализируемых файлов в мегабайтах. Файлы, превышающие этот размер, будут пропущены;
- `--no-banner` – отключение баннера Gitleaks, который отображается при запуске инструмента;
- `--no-color` – отключение цветного вывода для подробного режима (`verbose`);
- `--redact` – маскирование найденных секретов в логах. Можно задать промежуточные значения (например, `20` для скрытия 20% секрета);
- `--verbose` – включение подробного (`verbose`) вывода, предоставляющего больше информации о процессе сканирования.

#Параметры Trufflehog

- `--trufflehog-path` – путь к исполняемому файлу Trufflehog, который будет использоваться при сканировании. Если не задан, будет выполняться вызов системной команды `trufflehog`;
- `--trufflehog-config` – путь к конфигурационному файлу Trufflehog;
- `--concurrency` – количество параллельных воркеров при сканировании;
- `--no-verification` – отключить верификацию найденных секретов;
- `--only-verified` – выводить только верифицированные секреты;
- `--include-paths` – путь к файлу с regex-шаблонами (по одному на строку) для включения файлов в сканирование;

- `--exclude-paths` – путь к файлу с regex-шаблонами (по одному на строку) для исключения файлов из сканирования;
- `--trufflehog-log-level` – уровень логирования Trufflehog. Возможные значения: `debug`, `info`, `warn`, `error`.

Для сводки доступных параметров команды и инструкции по использованию можно вызвать команду с флагом `-h, --help`.

#Анализ достижимости уязвимостей

Что такое достижимость

Достижимость уязвимости — это проверка того, действительно ли потенциально уязвимый участок кода может быть выполнен при использовании приложения. Такой анализ позволяет отфильтровать «шум» и сосредоточиться на реально эксплуатируемых проблемах.

Консольный агент Johnny умеет анализировать уязвимости на достижимость из исходного кода. Для использования данной функции необходимо задать два параметра:

- `cg-path` — путь к файлу графа вызовов: для `java`, `python`, `go`, `kotlin` и `csharp` (C#) — формат Svace; для `javascript` — JSON, сформированный Joern (см. раздел ниже);
- `cg-lang` — язык программирования, для которого был построен граф вызовов. Поддерживаются значения `java`, `python`, `go`, `kotlin`, `csharp` (C#) и `javascript`.

#Построение графа вызовов

#С использованием инструмента Svace

1. Скачать модуль Svace `https://REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/#browse/browse:files:codescoring%2Fsvace-callgraph`

2. Получить токен пользователя в CodeScoring (по ссылке `{platform-url}/cabinet/profile`)

3. Запустить Svace на исходном коде проекта. Этот этап лучше делать в рамках или после шага сборки в конвейере.

    1. Инициализация

    ```
    svace init
    ```

    2. Контролируемая сборка

```
svace build <build command>
```

Пример для проектов на языке Java:

```
svace build mvn clean package
```

Пример на языке Go:

```
svace build go build -a main.go
```

Пример на языке Python:

```
svace build --python .
```

Пример на языке Kotlin:

```
svace build ./gradlew clean build
```

Пример на языке C#:

```
svace build dotnet build
```

3. Анализ результатов и построение графа вызовов

```
svace analyze --build-call-graph-only --license-
server-url "http(s)://<codescoring_host>" --license-
server-token "<токен из п.2>"
```

4. В случае успешного выполнения всех шагов в директории проекта появится файл `.svace-dir/analyze-res/call-graph-results/<project_name>-graph-order.json`, содержащий граф вызовов.

   Сохранение файла

   До версии Svace 5.0.260311 файл с графом вызовов сохранялся в папку `.svace-dir/analyze-res/call-graph`

5. Запустить сканирования с помощью Johnny, например:

```
johnny-linux-amd64 scan dir . --api_url "http(s)://
<codescoring_host>" --api_token "<токен из п.1>"  --cg-
path .svace-dir/analyze-res/call-graph-results/<project_n
ame>-graph-order.json  --cg-lang java
```

#С использованием Joern (JavaScript)

Для языка JavaScript граф вызовов для анализа достижимости строится с помощью Joern. В каталоге с исходным кодом проекта выполните:

```
joern-parse .
joern-slice usages cpg.bin
```

Команда `joern-parse` создаёт CPG в файле `cpg.bin`. Команда `joern-slice usages` записывает результат в JSON. По умолчанию файл сохраняется как `slices.json` в текущей папке. Путь к этому файлу передайте в параметре `--cg-path` при запуске Johnny. Имя выходного файла можно задать опцией `-o` у `joern-slice`; см. документацию Joern по CPG slicing.

Пример запуска сканирования:

```
johnny-linux-amd64 scan dir . --api_url "http(s)://
<codescoring_host>" --api_token "<токен из личного кабинета>"
--cg-path <путь_к_json_joern> --cg-lang javascript
```

#Удаленный анализ

В версии **Svace 5.0.260311** появилась возможность использования сервера удалённого анализа. Это позволяет вынести построение графа вызовов за пределы CI/CD пайплайна, снизить нагрузку на сборочные серверы и оптимизировать процесс непрерывной интеграции.

#Этапы настройки сервера

1. Указать переменные окружения

```
SVACE_LIC_SERVER_URL=http(s)://<codescoring_host>
SVACE_LIC_SERVER_TOKEN=<токен CodeScoring>
```

1. Инициализация

```
svace server init
```

1. Запустить сервер

```
svace server start
```

1. Отправить проект на анализ. После настройки сервера не нужно указывать параметры лицензии `--license-server-url` и `--license-server-token`.

```
svace remote --host <адрес сервера> analyze --build-call-graph-only
```

При необходимости можно добавить параметры `--port`, `--login`, `--pass` и т.д.

Подробнее с настройками сервера удалённого анализа можно ознакомиться в документации Svace.

#Получение результатов

В таблице уязвимостей с найденными достижимыми вызовами будет проставлена отметка в соответствующем столбце:

Vulnerabilities:

Vulnerability			Affect				
Vulnerability	Has exploit	Reachable	Fixed version	CVSSv2	CVSSv3	CWEs	Links
CVE-2023-5072	V	V	pkg:maven/org.json/json@20230618				
			20231013	None	7.5 High	CWE-1035 CWE-358 CWE-770 CWE-937	https://access.redhat.com/security/cve/CVE-2023-5072 https://github.com/advisories/GHSA-4jq9-2xhw-jpx7 https://nvd.nist.gov/vuln/detail/CVE-2023-5072 https://osv.dev/vulnerability/CVE-2023-5072 https://security-tracker.debian.org/tracker/CVE-2023-5072 https://ubuntu.com/security/CVE-2023-5072 https://www.cve.org/CVERecord?id=CVE-2023-5072
CVE-2023-50728			pkg:maven/org.json/json@20230618				
			20231013	None	7.5 High	CWE-1035 CWE-755 CWE-937	
KLA62110		V	pkg:maven/org.json/json@20230618				
			20231013	None	7.5 High	CWE-770	https://github.com/advisories/GHSA-rm7j-f5g5-27vv

В конце будет доступна ещё одна таблица с перечислением деревьев вызовов для уязвимостей:

Reachability paths:

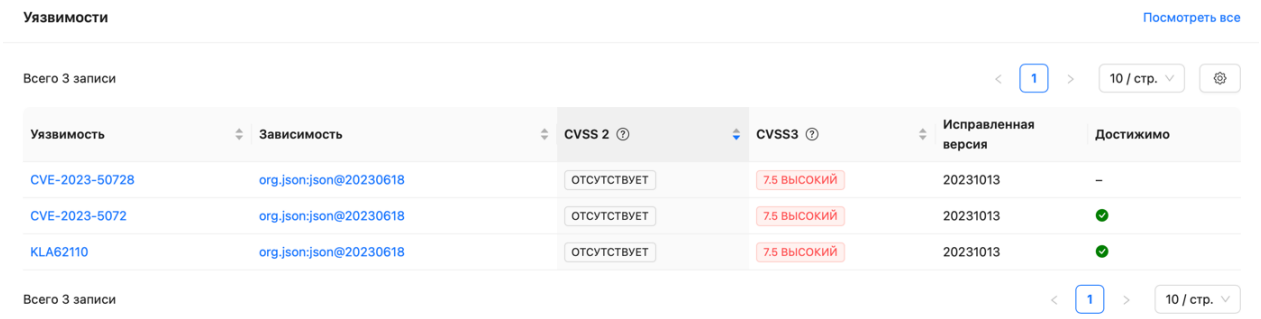
Package	Vulnerability	Paths
pkg:maven/org.json/json@20230618	CVE-2023-5072	Bug.main(String[]) (/json-bug/src/main/java/orgjsonbug/Bug.java) – JSONObject.JSONObject(String) – ! JSONObject.JSONObject(JSONTokener)
	KLA62110	

Пример для более крупного проекта:

Reachability paths:

Package	Vulnerability	Paths
pkg:maven/org.eclipse.jetty/jetty-xml@9.4.49.v20220914	KLA50723	VexResource.uploadVex(VexSubmitRequest) (/prj/src/main/java/org/dependencytrack/resources/v1/VexResource.java) – VexResource.process(QueryManager, Project, String) (/prj/src/main/java/org/dependencytrack/resources/v1/VexResource.java) – – BomResource.validate(byte[], Project) (/prj/src/main/java/org/dependencytrack/resources/v1/BomResource.java) – – – CycloneDxValidator.validate(byte[]) (/prj/src/main/java/org/dependencytrack/parser/cyclonedx/CycloneDxValidator.java) – – – ! XmlParser.XmlParser() VexResource.uploadVex(String, String, String, List) (/prj/src/main/java/org/dependencytrack/resources/v1/VexResource.java) – VexResource.process(QueryManager, Project, List) (/prj/src/main/java/org/dependencytrack/resources/v1/VexResource.java) – – BomResource.validate(byte[], Project) (/prj/src/main/java/org/dependencytrack/resources/v1/BomResource.java) – – – CycloneDxValidator.validate(byte[]) (/prj/src/main/java/org/dependencytrack/parser/cyclonedx/CycloneDxValidator.java) – – – ! XmlParser.XmlParser() BomResource.uploadBom(BomSubmitRequest) (/prj/src/main/java/org/dependencytrack/resources/v1/BomResource.java) – BomResource.process(QueryManager, Project, String, List) (/prj/src/main/java/org/dependencytrack/resources/v1/BomResource.java) – – BomResource.validate(byte[], Project) (/prj/src/main/java/org/dependencytrack/resources/v1/BomResource.java) – – – CycloneDxValidator.validate(byte[]) (/prj/src/main/java/org/dependencytrack/parser/cyclonedx/CycloneDxValidator.java) – – – ! XmlParser.XmlParser() BomResource.uploadBom(String, boolean, String, String, String, String, String, boolean, List) (/prj/src/main/java/org/dependencytrack/resources/v1/BomResource.java) – BomResource.process(QueryManager, Project, List, List) (/prj/src/main/java/org/dependencytrack/resources/v1/BomResource.java) – – BomResource.validate(byte[], Project) (/prj/src/main/java/org/dependencytrack/resources/v1/BomResource.java) – – – CycloneDxValidator.validate(byte[]) (/prj/src/main/java/org/dependencytrack/parser/cyclonedx/CycloneDxValidator.java) – – – ! XmlParser.XmlParser() BomUploadProcessingTask.inform(Event) (/prj/src/main/java/org/dependencytrack/tasks/BomUploadProcessingTask.java) – BomUploadProcessingTask.processEvent(BomUploadProcessingTask$Context, BomUploadEvent) (/prj/src/main/java/org/dependencytrack/tasks/BomUploadProcessingTask.java) – – BomParserFactory.createParser(byte[]) – – ! XmlParser.XmlParser() VexUploadProcessingTask.inform(Event) (/prj/src/main/java/org/dependencytrack/tasks/VexUploadProcessingTask.java) – BomParserFactory.createParser(byte[]) – ! XmlParser.XmlParser()

Если был указан флаг `--save-results`, то результаты достижимости будут в колонке "Достижимо" таблицы уязвимостей:



Уязвимости — Посмотреть все

Всего 3 записи — 1 — 10 / стр.

Уязвимость	Зависимость	CVSS 2	CVSS3	Исправленная версия	Достижимо
CVE-2023-50728	org.json:json@20230618	ОТСУТСТВУЕТ	7.5 ВЫСОКИЙ	20231013	–
CVE-2023-5072	org.json:json@20230618	ОТСУТСТВУЕТ	7.5 ВЫСОКИЙ	20231013	✓
KLA62110	org.json:json@20230618	ОТСУТСТВУЕТ	7.5 ВЫСОКИЙ	20231013	✓

Всего 3 записи — 1 — 10 / стр.

#Подпись и верификация SBOM

Для подтверждения целостности и подлинности SBOM-файлов поддерживается подпись и верификация с использованием цифровых подписей RSA SHA256. Функциональность доступна начиная с версии консольного агента **2025.29.0**.

Обе команды доступны без необходимости задания параметров `--api_url` и `--api_token`.

Важно: поддерживаются только RSA ключи, и они должны быть в формате PEM.

#Подпись SBOM файла

Для создания цифровой подписи SBOM файла используется команда `sign bom`. Она имеет следующие параметры:

- `--private-key <путь>` - путь к приватному ключу RSA в формате PEM (**обязательно**);
- `--include-public-key` - включить публичный ключ в SBOM файл (опционально).

Примеры запуска

```
## Подпись файла с указанием приватного ключа
./johnny sign bom <bom_json> \
--api_token <api_token> \
--api_url <api_url> \
--private-key <private_key_pem>

## Подпись файла с включением публичного ключа в SBOM
./johnny sign bom <bom_json> \
--api_token <api_token> \
--api_url <api_url> \
--private-key <private_key_pem> \
--include-public-key
```

#Верификация подписи SBOM файла

Для проверки подписи SBOM файла используется команда `verify bom`. Она имеет следующие параметры:

- `--public-key <путь>` - путь к публичному ключу RSA в формате PEM (опционально). Если этот параметр задан и файл SBOM содержит публичный ключ, будет использован ключ из файла SBOM.

Примеры запуска

```
## Верификация с использованием публичного ключа из файла
./johnny verify bom <bom_json> \
--api_token <api_token> \
--api_url <api_url> \
--public-key <public_key_pem>

## Верификация с использованием ключа из SBOM файла
./johnny verify bom <bom_json> \
--api_token <api_token> \
--api_url <api_url>
```

#Результаты работы

Агент возвращает следующие exit code:

- 0: успешное выполнение;
- 4: ошибка верификации подписи.

#Запуск с помощью Docker

Для работы запуска через Docker в данный момент нужна активная авторизация в registry с образами системы.

Пример вызова на текущей директории

```
docker run --rm \
    -v $(pwd):/code \
    -a stdout \
    <registry-address>/johnny-depp:<version> \
    scan dir . \
    --api_token <api_token> \
    --api_url <api_url> \
    --ignore .tmp --ignore fixtures --ignore .git
```

Параметр `-a stdout` необходим для корректного отображения таблиц **Vulnerabilties** и **Policy Alerts** при запуска агента через Docker.

#Добавление в Gitlab CI

Консольный агент поддерживает добавление в Gitlab CI с помощью файла `.gitlab-ci.yaml` и поставляется как в виде docker-образа, так и в виде бинарного файла.

#Docker-образ Johnny

Пример файла .gitlab-ci.yaml (Docker-образ)

`<version>` необходимо заменить на версию агента. Список актуальных версий с описанием доступен на странице Changelog.

```
stages:
  - test

sca:
  stage: test

  script:
    - docker pull REGISTRY_URL/johnny-depp:<version>
    - >
      docker run -v $(pwd):/code
      <registry-address>/johnny-depp
      --api_token $CODESCORING_API_TOKEN
      --api_url $CODESCORING_API_URL
      --ignore .git
      --ignore fixtures
      --ignore parsers
      .

  artifacts:
    paths:
      - bom.json
    when: always
    expire_in: 1 week
```

#Бинарный файл Johnny

Для использования бинарного файла консольного агента, необходимо предварительно выполнить следующие действия на машине gitlab-runner'a:

1. Скачать файл командой

```
wget -O /usr/local/bin/johnny https://
REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/repository/
files/codescoring/johnny-depp/JOHNNY_VERSION/johnny-linux-
amd64-JOHNNY_VERSION
```

или

```
curl -o /usr/local/bin/johnny https://
REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/repository/
files/codescoring/johnny-depp/JOHNNY_VERSION/johnny-linux-
amd64-JOHNNY_VERSION
```

JOHNNY_VERSION необходимо заменить на версию агента. Список актуальных версий с описанием доступен на странице Changelog. REGISTRY_URL, REGISTRY_USERNAME и REGISTRY_PASSWORD необходимо заменить на адрес, логин и пароль, полученные от вендора. 2. Разрешить исполнение файла

```
chmod +x /usr/local/bin/johnny
```

Пример вызова в .gitlab-ci.yaml (бинарный файл)

```
stages:
  - test

sca:
  stage: test

  script:
    - >
      johnny
      scan dir
      --api_token $JOHNNY_API_TOKEN
      --api_url $JOHNNY_API_URL
      --ignore .git
      --ignore fixtures
      --ignore parsers
      .

  artifacts:
    paths:
```

```
      - bom.json
    when: always
    expire_in: 1 week
```

#Добавление в Jenkins

Консольный агент поддерживает добавление в Jenkins двумя способами: через `Jenkinsfile` и специализированный плагин.

#Добавление агента в Jenkinsfile

#Использование Docker-образа

Пример добавления агента в pipeline (Docker-образ)

```
pipeline {
    agent any

  environment {
    CODESCORING_REGISTRY_URL='REGISTRY_URL'
    CODESCORING_AGENT_IMAGE='REGISTRY_URL/johnny-
depp:<version>'
    CODESCORING_REGISTRY_CREDENTIALS=credentials('cs-registry-
creds')
    CODESCORING_API_URL='https://localhost:8080'
  }

  stages {

    stage("Login to Codescoring docker registry") {
      steps {
        sh """
        docker login -u "$CODESCORING_REGISTRY_CREDENTIALS_USR"
"$CODESCORING_REGISTRY_URL" -p
"$CODESCORING_REGISTRY_CREDENTIALS_PSW"
          """
      }
    }

    stage('Run CodeScoring Agent') {
      steps {
        sh """
        docker run -v \$(pwd):/code --rm $
{CODESCORING_AGENT_IMAGE} --api_token ${CODESCORING_API_TOKEN}
--api_url ${CODESCORING_API_URL} --ignore .tmp --ignore
fixtures --ignore .git .
        """
      }
    }
```

```
  }
}
```

#Использование бинарного файла

Для использования бинарного файла консольного агента, необходимо предварительно выполнить следующие действия на машине с Jenkins:

1. Скачать файл командой

   ```
   wget -O /usr/local/bin/johnny https://
   REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/
   repository/files/codescoring/johnny-depp/JOHNNY_VERSION/
   johnny-linux-amd64-JOHNNY_VERSION
   ```

   или

   ```
   curl -o /usr/local/bin/johnny https://
   REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/
   repository/files/codescoring/johnny-depp/JOHNNY_VERSION/
   johnny-linux-amd64-JOHNNY_VERSION
   ```

   Переменную `JOHNNY_VERSION` необходимо заменить на версию агента. Список актуальных версий доступен в разделе Changelog.

   Переменные `REGISTRY_URL`, `REGISTRY_USERNAME` и `REGISTRY_PASSWORD` необходимо заменить на адрес, логин и пароль, полученные от вендора.

2. Разрешить исполнение файла

```
  chmod +x /usr/local/bin/johnny
```

Пример вызова агента в pipeline (бинарный файл)

```
pipeline {
    agent any

  environment {
    CODESCORING_API_URL='http://localhost:8001'
    CODESCORING_API_TOKEN='API_TOKEN'
  }

  stages {

    stage('Run CodeScoring Agent') {
      steps {
        sh """
        johnny scan dir --api_token ${CODESCORING_API_TOKEN} --
```

```
api_url ${CODESCORING_API_URL} --ignore .tmp --ignore fixtures
--ignore .git .
        """
      }
    }
  }
}
```

#Добавление в Gitflic CI

С помощью консольного агента johnny можно настроить сканирование компонентов в GitFlic CI. Поддерживаются типы раннера GitFlic shell и GitFlic docker.

#Использование агента с типом раннера GitFlic shell

Для использования консольного агента с типом раннера GitFlic Shell необходимо предварительно выполнить следующие действия:

1. Скачать файл командой

   ```
   wget -O /usr/local/bin/johnny https://
   REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/
   repository/files/codescoring/johnny-depp/JOHNNY_VERSION/
   johnny-linux-amd64-JOHNNY_VERSION
   ```

   или

   ```
   curl -o /usr/local/bin/johnny https://
   REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/
   repository/files/codescoring/johnny-depp/JOHNNY_VERSION/
   johnny-linux-amd64-JOHNNY_VERSION
   ```

   `JOHNNY_VERSION` необходимо заменить на версию агента. Список актуальных версий с описанием доступен в разделе Changelog. `REGISTRY_URL`, `REGISTRY_USERNAME` и `REGISTRY_PASSWORD` необходимо заменить на адрес, логин и пароль, полученные от вендора.

2. Разрешить исполнение файла

```
chmod +x /usr/local/bin/johnny
```

Пример вызова бинарного файла агента в `gitflic-ci.yaml`:

```
  stages:
    - test
```

```
  sca:
    stage: test

  script:
      - >
        johnny
        scan dir
        --api_token $JOHNNY_API_TOKEN
        --api_url $JOHNNY_API_URL
        --ignore .git
        --ignore fixtures
        --ignore parsers
        .

artifacts:
  reports:
  paths:
  dependency_scanning:  "bom.json"
```

Результатами выполненного сканирования можно управлять на вкладке **Безопасность** в интерфейсе проекта.

#Использование агента с типом раннера GitFlic docker

Для использования консольного агента с типом раннера GitFlic docker необходимо предварительно выполнить следующие действия на машине с агентом:

1. Скачать файл командой

   ```
   wget -O /usr/local/bin/johnny https://
   REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/
   repository/files/codescoring/johnny-depp/JOHNNY_VERSION/
   johnny-linux-amd64-JOHNNY_VERSION
   ```

   или

   ```
   curl -o /usr/local/bin/johnny https://
   REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/
   repository/files/codescoring/johnny-depp/JOHNNY_VERSION/
   johnny-linux-amd64-JOHNNY_VERSION
   ```

   `JOHNNY_VERSION` необходимо заменить на версию агента. Список актуальных версий с описанием доступен в разделе Changelog. `REGISTRY_URL`, `REGISTRY_USERNAME` и `REGISTRY_PASSWORD` необходимо заменить на адрес, логин и пароль, полученные от вендора.

2. Скопировать агента в контейнер, который планируется использовать в задаче

```
docker cp ./johnny CONTAINER:/usr/bin
```

3. Разрешить исполнение файла

```
docker exec CONTAINER chmod +x /usr/bin/johnny
```

4. Сохранить изменения в контейнере

```
docker commit <container name><repository>:<tag>
```

Важно: при необходимости сохраните контейнер в удаленном репозитории.

Пример вызова бинарного файла агента в `gitflic-ci.yaml`:

```
    stages:
      - test

    sca:
      stage: test
    image: <repository><tag>

    script:
        - >
          johnny
          scan dir
          --api_token $JOHNNY_API_TOKEN
          --api_url $JOHNNY_API_URL
          --ignore .git
          --ignore fixtures
          --ignore parsers
          .

  artifacts:
    reports:
    paths:
    dependency_scanning:  "bom.json"
```

Результатами выполненного сканирования можно управлять на вкладке **Безопасность** в интерфейсе проекта.

#Подключение к реестру и проверка образов

Пример выборочной проверки образа с помощью агента в `gitflic-ci.yaml`:

```
image: angelikade/mvn-npm-jdk:codescoring
stage: test-codescoring-image
```

```
when: manual
scripts:
  - ls -la
  - |
    /usr/bin/johnny scan image <registry>/<repository>/
<imagename>:<tag> \
    --api_token "${CS_TOKEN}" \
    --api_url "${CS_URL}"
```

Важно: доступ к файлу /v2/_catalog в GitFlic выключен из соображений безопасности. На текущий момент, рекуррентный проход по всем образам в реестре невозможен.

#Использование политик безопасности при сканировании

1. Настройте политики на платформе CodeScoring
2. Запустите конвейер, используя стандартные настройки сканирования

```
    stages:
      - test

    sca:
      stage: test

    script:
        - >
          johnny
          scan dir
          --api_token $JOHNNY_API_TOKEN
          --api_url $JOHNNY_API_URL
          --ignore .git
          --ignore fixtures
          --ignore parsers
          .

  artifacts:
    reports:
    paths:
    dependency_scanning:  "bom.json"
```

3. При срабатывании политик агент завершит работу с возвратом кода ошибки и раннер GitFlic автоматически остановит конвейер.

Важно: на текущий момент в GitFlic не реализован механизм получения артефактов при завершении задачи с ошибкой. Ввиду этого, просмотр отчета по артефакту, вызвавшему остановку конвейера, в веб-интерфейсе GitFlic невозможен.

#Сохранение результатов сканирования

Для сохранения результатов сканирования в on-premise платформе необходимо добавить в команду параметры `--save-results` и `--project` или указать в config-файле следующие переменные:

- `project` — имя CLI-проекта в системе, в который будут сохраняться результаты;
- `save-results` — флаг сохранения результатов, по умолчанию стоит значение **false**.

Если CLI-проект не создан в системе заранее, можно указать в команде вызова или в config-файле параметр `--create-project`. Для нового проекта можно указать следующие параметры:

- `--project-group` - имя группы. В случае если группа с таким именем не существует и передан флаг `--create-project-group`, перед добавлением проекта в группу она будет создана;
- `--project-proprietor` - подразделение;
- `--project-categories` - категории.

Пример команды сохранения результатов сканирования в новый проект

```
./johnny scan dir . \
--api_token <api_token> \
--api_url <api_url> \
--save-results \
--create-project \
--project "project-name" \
--project-group "group" \
--project-proprietor "proprietor"
```

#Экспорт результатов сканирования

Консольный агент Johnny поддерживает выгрузку результатов сканирования в различных форматах. Это позволяет адаптировать отчетность под разные нужды, включая интеграцию с системами управления уязвимостями. По умолчанию отчеты отображаются на английском, для переключения на русский нужно использовать флаг `--localization ru`.

#Отчет о найденных уязвимостях

#Доступные форматы

- **coloredtable** – цветная таблица в консоли. Формат по умолчанию;
- **table** – простая таблица;
- **text** – текстовый отчет;
- **junit** – используется в CI/CD (Jenkins, GitLab CI, GitHub Actions);

- **sarif** – выгружается в DefectDojo и другие системы управления уязвимостями;
- **csv** – применяется в BI-системах, Excel, Pandas, SQL;
- **gl-dependency-scanning-report** – формат отчета для GitLab Dependency Scanning;
- **gl-code-quality-report** – формат отчета для GitLab Code Quality;
- **gl-secrets-report** – формат отчета для GitLab Secret Detection.

#Пример использования

При необходимости можно указать несколько форматов, разделив их запятыми, например:

```
./johnny scan file path/to/file \
--api_token <api_token> \
--api_url <api_url> \
--format "coloredtable, junit>>junit.xml"
```

В этом примере вывод будет в формате `coloredtable` в консоль, а также сохранится в файл `junit.xml` в формате `junit`.

#Отчет о сработавших алертах

#Доступные форматы

- **coloredtable** – цветная таблица в консоли. Формат по умолчанию;
- **table** – простая таблица;
- **text** – текстовый отчет;
- **json** – структурированный формат на основе JavaScript Object Notation, удобен для машинной обработки данных;
- **csv** – текстовый формат файла, для хранения табличных данных.

#Пример использования

При необходимости можно указать несколько форматов, разделив их запятыми, например:

```
./johnny scan file path/to/file \
--api_token <api_token> \
--api_url <api_url> \
--alerts-format "coloredtable, json>>alerts.json"
```

В этом примере вывод будет в формате `coloredtable` в консоль, а также сохранится в файл `alerts.json` в формате `json`.

#Отчет об игнорах политик

Формирование отчета выполняется по флагу `--policy-ignores`. В отчет попадают игноры политик с указанным в `--project` проектом.

#Доступные форматы

- **coloredtable** – цветная таблица в консоли. Формат по умолчанию;
- **table** – простая таблица;
- **text** – текстовый отчет;
- **json** – структурированный формат на основе JavaScript Object Notation, удобен для машинной обработки данных;
- **csv** – текстовый формат файла, для хранения табличных данных.

#Пример использования

При необходимости можно указать несколько форматов, разделив их запятыми, например:

```
./johnny scan file path/to/file \
--api_token <api_token> \
--api_url <api_url> \
--project <project> \
--policy-ignores \
--ignores-format "coloredtable, json>>ignores.json"
```

В этом примере вывод будет в формате `coloredtable` в консоль, а также сохранится в файл `ignores.json` в формате `json`.

#Разрешение зависимостей в окружении сборки

Пакетные менеджеры некоторых экосистем по умолчанию не включают транзитивные зависимости в манифесты. Для качественного проведения композиционного анализа при работе с ними рекомендуется применять механизм разрешения зависимостей в окружении сборки.

При разрешении зависимостей в окружении агент проверяет отсутствие lock-файла. Если lock-файл обнаружен, резолв не выполняется даже при наличии соответствующих флагов, и результаты берутся из обнаруженного lock-файла. Исключение составляют технологии, где имя и расположение lock-файла не фиксировано пакетным менеджером и может быть передано как параметр в резолв.

#Настройка разрешения зависимостей

Параметры разрешения зависимостей в окружении, пути к пакетному менеджеру и параметры выполнения регулируются следующими параметрами в команде `scan`:

- `--dotnet-resolve` / `--dotnet-path` / `--dotnet-args`
- `--go-resolve` / `--go-path`
- `--gradle-resolve` / `--gradle-path` / `--gradle-args`

- `--maven-resolve` / `--maven-path` / `--maven-args`
- `--npm-resolve` / `--npm-path` / `--npm-args`
- `--poetry-resolve` / `--poetry-path` / `--poetry-args`
- `--sbt-resolve` / `--sbt-path` / `--sbt-args`
- `--yarn-resolve` / `--yarn-path` / `--yarn-args`
- `--pip-resolve` / `--pip-path` / `--pip-args`
- `--composer-resolve` / `--composer-path` / `--composer-args`
- `--pnpm-resolve` / `--pnpm-path` / `--pnpm-args`
- `--conda-resolve` / `--conda-lock-path` / `--conda-args`
- `--pipdeptree-resolve` / `--pipdeptree-path` / `--pipdeptree-args`
- `--uv-resolve` / `--uv-path` / `--uv-args`
- `--bun-resolve` / `--bun-path` / `--bun-args`

Пример команды:

```
./johnny \
scan dir . \
--api_token <api_token> \
--api_url <api_url> \
--dotnet-resolve \
--dotnet-path <path/to/dotnet>
```

При необходимости перечисленные параметры можно добавить в конфигурационный файл агента.

#Поддерживаемые экосистемы

#.NET

Базовый манифест, который берется за основу при разрешении зависимостей: `.csproj`.

Для проектов .NET агент выполняет команду:

```
dotnet restore
```

После чего анализируется файл `obj/project.assets.json`, содержащий полную информацию о зависимостях и их версиях. Выполняется в каталоге, в котором расположен `.csproj` файл.

В случае обнаружения `.sln` манифеста, разрешение зависимостей будет выполнено в его контексте. Условием для запуска команды разрешения является отсутствие `obj/project.assets.json` файла для одного и более компонентов решения.

#Go

Базовые манифесты, которые берутся за основу при разрешении зависимостей: `go.mod`, `go.sum`.

Агент использует данные из файлов `go.mod` и `go.sum`, добавляя записи из `go.sum`, которые отсутствуют в `go.mod` (только строки без постфикса `/go.mod`). Затем выполняется:

```
go mod graph
```

Полученный список пар `parent → child` используется для построения дерева зависимостей. При отсутствии указания родителя используется команда:

```
go mod why <package>
```

Если родительская связь не установлена, зависимость исключается с предупреждением.

#Gradle

Базовые манифесты, которые берутся за основу при разрешении зависимостей: `build.gradle`, `build.gradle.kts`.

Для разрешения зависимостей в Gradle по умолчанию необходимо задать следующее значение:

```
--gradle-path ./gradlew
```

С заданным значением сначала выполняется пользовательская задача:

```
./gradlew CodeScoring_All_Dependencies --configuration <конфиг
урация>
```

Если задача отсутствует, используется стандартная команда:

```
./gradlew dependencies --configuration <конфигурация>
```

Агент анализирует консольный вывод и формирует граф зависимостей.

#Дополнительная информация

При наличии в директории файла gradle-dependency-tree.txt будет использоваться существующий файл, новый создан не будет.

#Maven

Базовый манифест, который берется за основу при разрешении зависимостей: `pom.xml`.

Для проектов Maven используется команда:

```
mvn dependency:tree -f <pom.xml> -DoutputFile=<tmpdir/mdt.json>
```

Агент парсит файл `mdt.json`, содержащий полную структуру зависимостей. При наличии файла `maven-dependency-tree.txt` он будет обработан как самостоятельный lock-файл, и резолв не будет выполняться.

#npm

Базовый манифест, который берется за основу при разрешении зависимостей: `package.json`.

Агент выполняет:

```
npm install
```

Далее анализируется сформированный файл `package-lock.json`, фиксирующий дерево зависимостей и используемые версии.

#pnpm

Базовый манифест, который берется за основу при разрешении зависимостей: `package.json`.

Выполняется команда:

```
pnpm install
```

Анализируется lock-файл `pnpm-lock.yaml`, в котором содержатся данные обо всех зависимостях.

#yarn

Базовый манифест, который берется за основу при разрешении зависимостей: `package.json`.

Выполняется команда:

```
yarn install
```

Агент парсит файл `yarn.lock`, содержащий информацию о зависимостях и их версиях.

#bun

Базовый манифест, который берется за основу при разрешении зависимостей: `package.json`.

Выполняется команда:

```
bun install --lockfile-only
```

Агент парсит файл `bun.lock`, содержащий информацию о зависимостях и их версиях.

Наличие bun.lockb файла

Для того чтобы разрешить зависимости в проекте в котором есть `bun.lockb` агенту необходимо передать следующий флаг: `--bun-args '--save-text-lockfile --frozen-lockfile'`

Особенность работы пакетного менеджера

В связи с особенностями реализации механизма создания файла `bun.lock` пакетным менеджером bun между двумя запусками набор глубоких транзитивных зависимостей может меняться

#pip

Для Python-проектов используется команда:

```
pip freeze
```

Результат команды фиксирует список установленных зависимостей и их версии. В результатах указывается фиктивный файл `codescoring_pip_for_freeze`.

#pipdeptree

Для Python-проектов используется команда:

```
pipdeptree
```

Результат команды фиксирует список установленных зависимостей и их версии в дереве зависимостей проекта. Для проектов, использующих `requirements.txt` или `Pipfile`, зависимости будут ограничены только теми пакетами, которые перечислены в этих манифестах — остальные пакеты из среды, обнаруженные pipdeptree, в отображение не попадут. Для проектов с `pyproject.toml` фильтрация будет проведена по `project.name`. В результатах указывается фиктивный файл `codescoring_pipdeptree`.

Взаимодействие с другими манифестами

Для того чтобы зависимости основного манифеста проекта (например, `requirements.txt`) не отображались в результатах анализа вместе с результатом анализа pipdeptree рекомендуется исключить этот манифест из сканирования:

```
    johnny scan python . \
      --pipdeptree-resolve \
      --ignore "requirements.txt"
```

#Poetry

Базовый манифест, который берется за основу при разрешении зависимостей: `pyproject.toml`.

Выполняются две команды:

```
poetry debug resolve --tree
poetry debug resolve
```

Первый вывод содержит дерево с констрейнтами, второй — конкретные версии. Агент сопоставляет данные и формирует итоговый граф.

#uv

Базовый манифест, который берется за основу при разрешении зависимостей: `pyproject.toml`.

Выполняется команда:

```
uv lock
```

Агент парсит файл `uv.lock`, содержащий информацию о зависимостях и их версиях.

#pdm

Базовый манифест, который берется за основу при разрешении зависимостей: `pyproject.toml`.

Выполняется команда:

```
pdm lock
```

Агент парсит файл `pdm.lock`, содержащий информацию о зависимостях и их версиях. В случае использования альтернативного формата агент распарсит файл `pylock.toml`.

#sbt (Scala)

Базовый манифест, который берется за основу при разрешении зависимостей: `build.sbt`.

Для Scala используется команда:

```
sbt dependencyTree
```

Анализируется консольный вывод, содержащий структуру зависимостей проекта.

#Swift

Базовый манифест, который берется за основу при разрешении зависимостей: `Package.swift`.

Для Swift-экосистемы агент выполняет:

```
swift build --package-path <путь_к_Package.swift>
```

Далее парсится файл `Package.resolved`, содержащий зафиксированные версии пакетов.

#Composer (PHP)

Базовый манифест, который берется за основу при разрешении зависимостей: `composer.json`.

Выполняется команда:

```
composer install
```

Агент анализирует файл `composer.lock`, в котором зафиксировано состояние зависимостей.

#Conda

Базовые манифесты, которые берутся за основу при разрешении зависимостей: `environment.yml`, `environment.yaml`, `meta.yml`, `meta.yaml`.

Для проектов, использующих Conda, агент выполняет:

```
conda-lock -f <environment.yml> --filename <tmpdir/conda-
lock.yml>
```

После этого анализируется файл `conda-lock.yml`, в котором зафиксированы все зависимости проекта.

#Интеграция CodeScoring в среду разработки

#Общее описание

CodeScoring предоставляет анализ состава программного обеспечения (SCA) непосредственно в вашей среде разработки через специализированные плагины для IDE. Эти интеграции позволяют разработчикам выявлять и исправлять уязвимые зависимости, перенося безопасность на ранние этапы жизненного цикла разработки.

Интегрируя обнаружение уязвимых компонентов в IDE, разработчики могут:

- Обнаруживать проблемы безопасности при написании кода, а не после аудита
- Получать быструю и актуальную обратную связь об уязвимостях зависимостей
- Узнавать о более безопасных версиях и применять их в среде разработки
- Поддерживать безопасный код с самых ранних этапов разработки
- Следить за составом компонентов
- Просматривать список алертов для сработавших политик

#Доступные интеграции

CodeScoring предлагает плагины для самых популярных сред разработки:

- Расширение для Visual Studio Code
- Плагин для IntelliJ-based IDEs

Обе интеграции обеспечивают сканирование уязвимостей из среды, визуальные индикаторы в коде и возможности обновления уязвимых зависимостей в один клик, позволяя разработчикам поддерживать безопасные зависимости без нарушения их рабочего процесса.

#Расширение CodeScoring.SCA для Visual Studio Code

Расширение предоставляет возможности анализа состава программного обеспечения (SCA) для VS Code, подсвечивая уязвимые зависимости в файлах Вашего проекта и предоставляя подробную информацию об уязвимостях через интеграцию с Johnny CLI.

Расширение **CodeScoring.SCA** поддерживает версии Visual Studio Code **1.95.0** и выше.

#Поддерживаемые экосистемы

#Языки и менеджеры пакетов

Экосистема	Файлы манифестов	Сгенерированные файлы версий
JavaScript/Node	package.json	package-lock.json, npm-shrinkwrap.json, yarn.lock, pnpm-lock.yaml, npm-lock.yaml
Python	setup.py, pyproject.toml, Pipfile, *require*.txt, *require*.pip	Pipfile.lock, poetry.lock
Java	pom.xml, ivy.xml, *.gradle, *.gradle.kts	gradle.lockfile, maven-dependency-tree.txt, gradle-dependency-tree.txt
Ruby	Gemfile, gems.rb, *.gemspec	Gemfile.lock, gems.locked
Go	go.mod	go.sum
Rust	Cargo.toml	Cargo.lock
PHP	composer.json	composer.lock
C#/.NET	*.csproj, packages.config, Project.json, paket.dependencies, *.nuspec	packages.lock.json, Project.lock.json, paket.lock, project.assets.json, dependencyReport.json
Swift	Package.swift	Package.resolved
Objective-C	Podfile, *.podspec	Podfile.lock
Conda	environment.yml, environment.yaml, meta.yml, meta.yaml	conda-lock.yml
Conan (C/C++)	conanfile.txt, conanfile.py	conan.lock

#Обнаружение файлов по умолчанию

- **Автоматическое**: Сканирует все поддерживаемые файлы
- **Рекурсивное**: Ищет в подкаталогах

Точная настройка сканирования производится с помощью модификации config.yaml файла

#Начало работы

#Предварительные требования

Перед началом убедитесь, что у Вас есть:

- Visual Studio Code, установленный в Вашей системе
- Доступ к установке CodeScoring с активными учетными данными
- Дистрибутив codescoring-sca расширения для vscode (.vsix)

#Требуемые разрешения

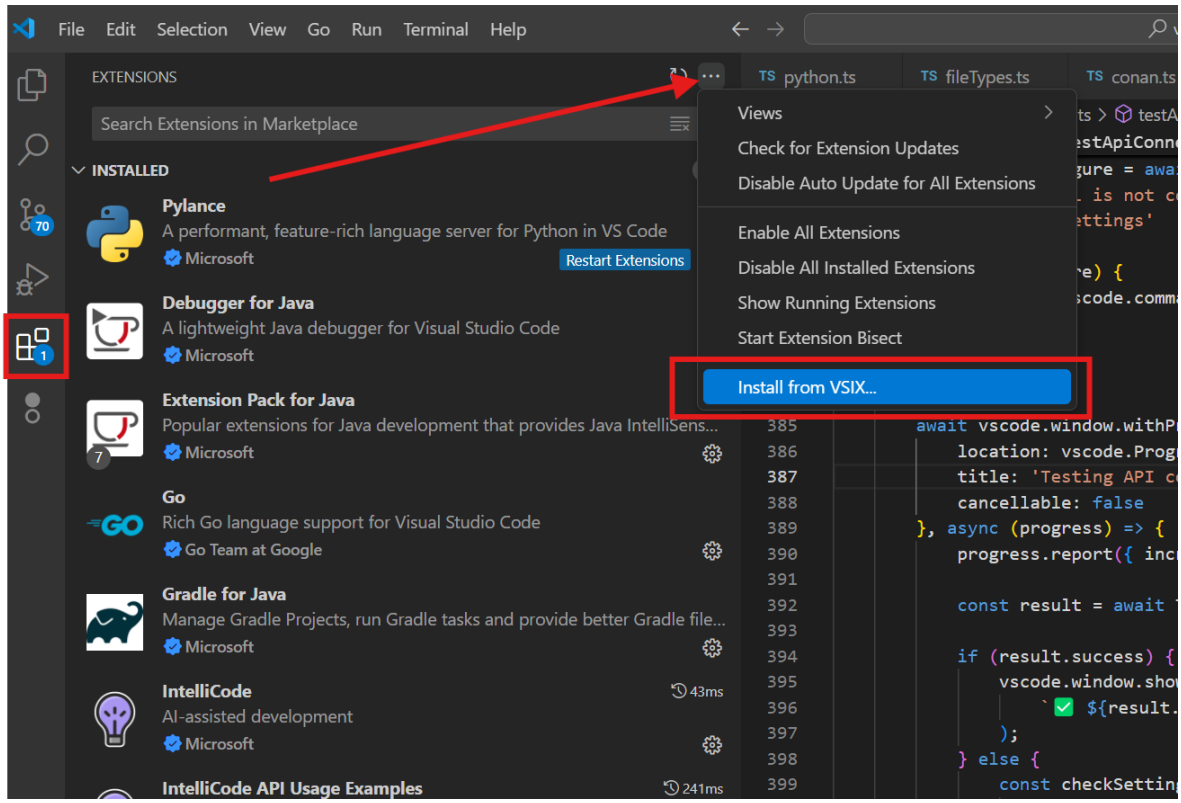
- **Файловая система**: Чтение файлов проекта, запись файлов .codescoring, загрузка исполняемого файла, выполнение загруженного CLI
- **Сеть**: Связь с CodeScoring API
- **API VS Code**: Интеграция с редактором

#Шаг 1: Загрузите расширение

Расширение поставляется в виде платформенно-независимого файла `codescoring-sca-<version>.vsix`.

#Шаг 2: Установите расширение из файла VSIX

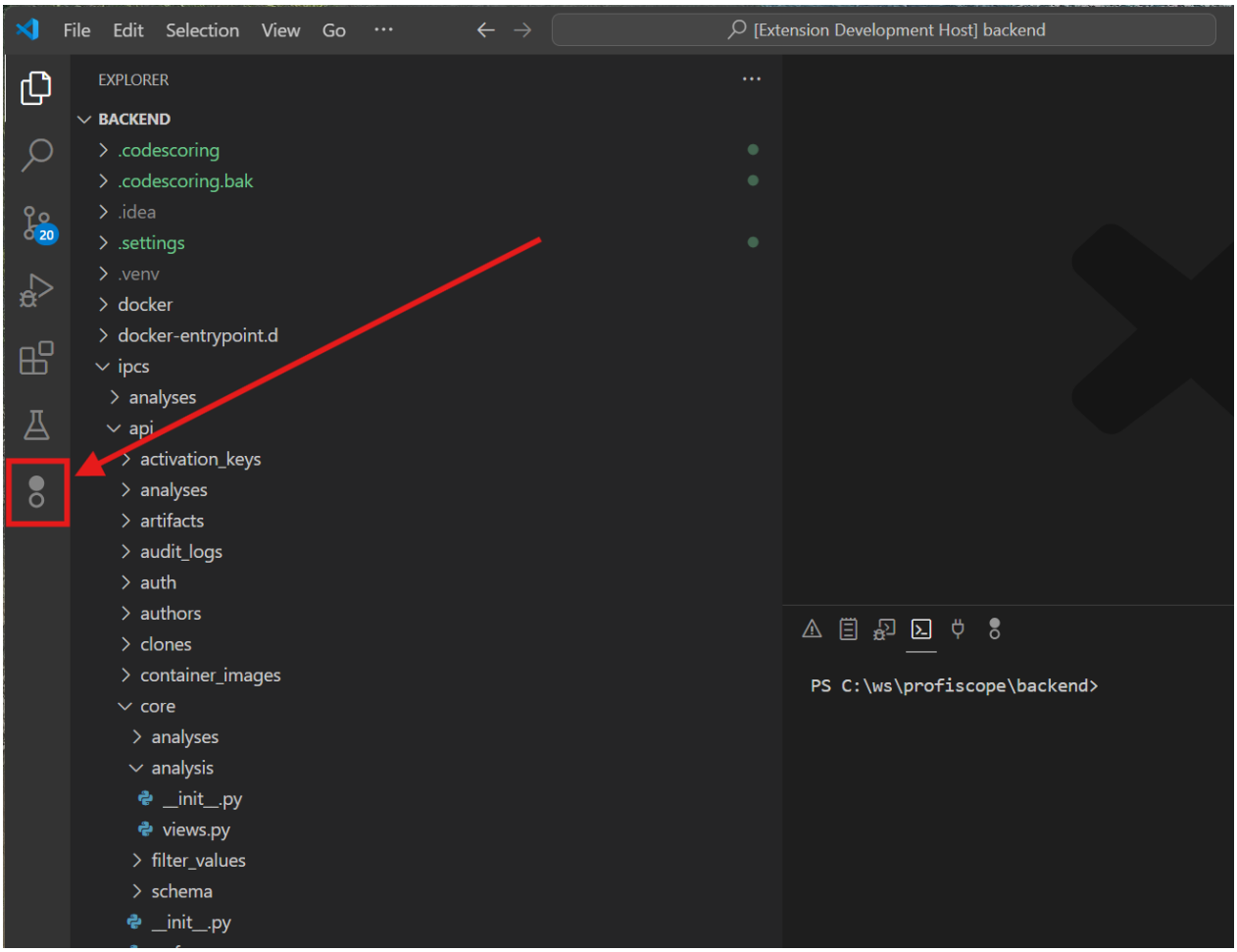
1. Откройте Visual Studio Code
2. Перейдите в **Файл → Настройки → Расширения** (или нажмите `Ctrl+Shift+X` или `Cmd+Shift+X` на MacOS)
3. Нажмите на меню с **тремя точками (...)** в правом верхнем углу панели расширений
4. Выберите **"Установить из VSIX..."** из выпадающего меню



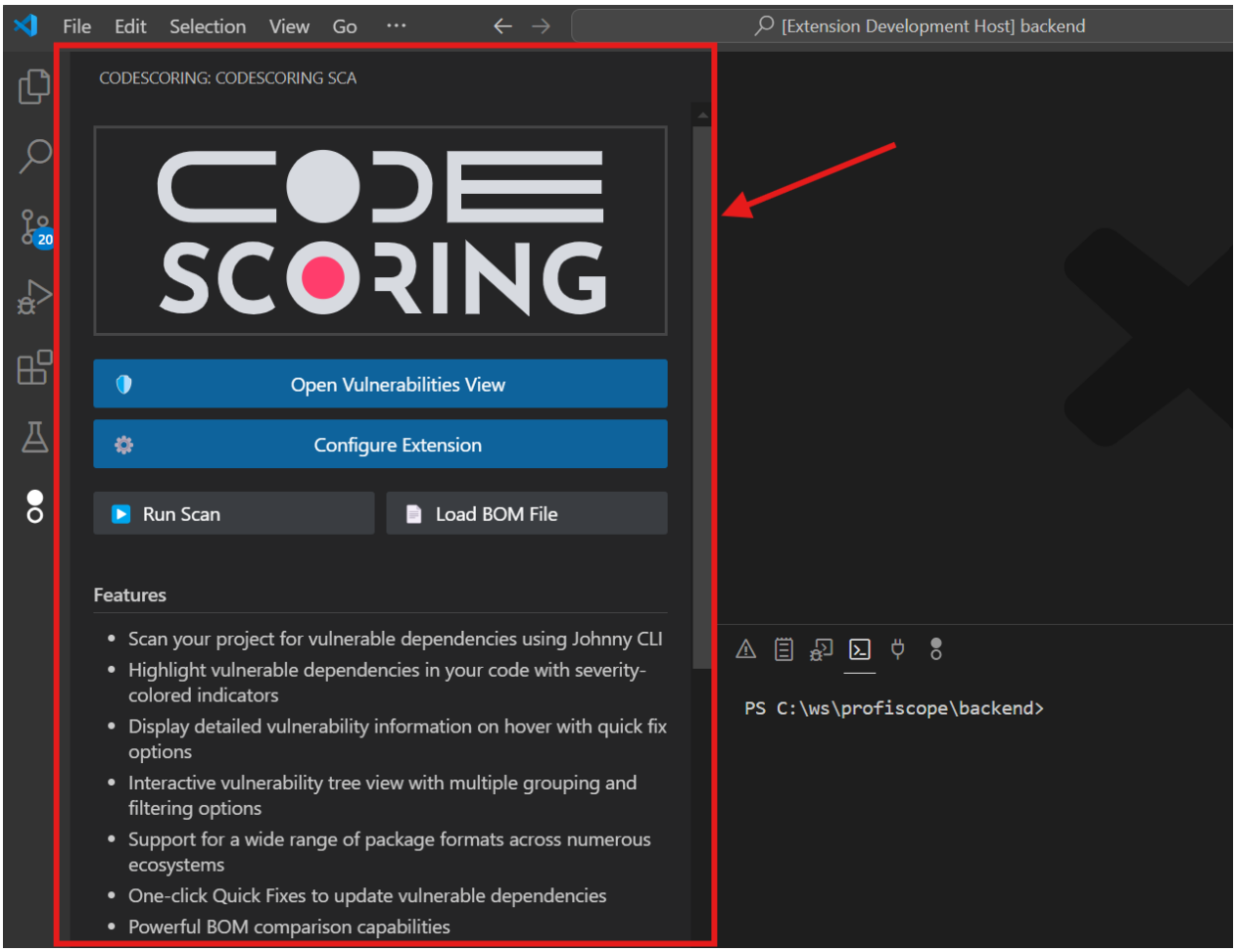
5. Перейдите к месту, куда вы сохранили файл `.vsix`
6. Выберите файл и нажмите **"Установить"**
7. Дождитесь завершения установки

#Шаг 3: Найдите расширение CodeScoring

После установки вы должны увидеть логотип CodeScoring в панели действий VS Code (боковая панель слева).

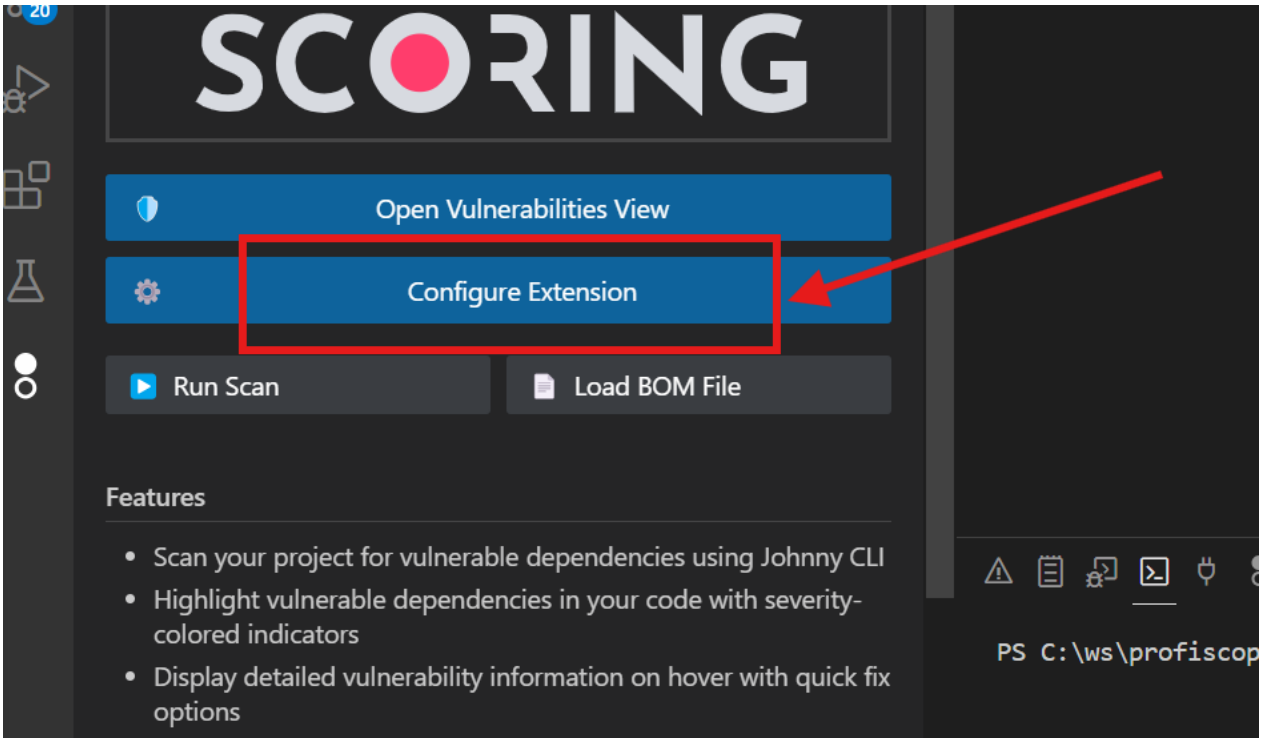


1. Нажмите на **иконку CodeScoring** в панели действий
2. Откроется боковая панель **"CODESCORING: CODESCORING SCA"**



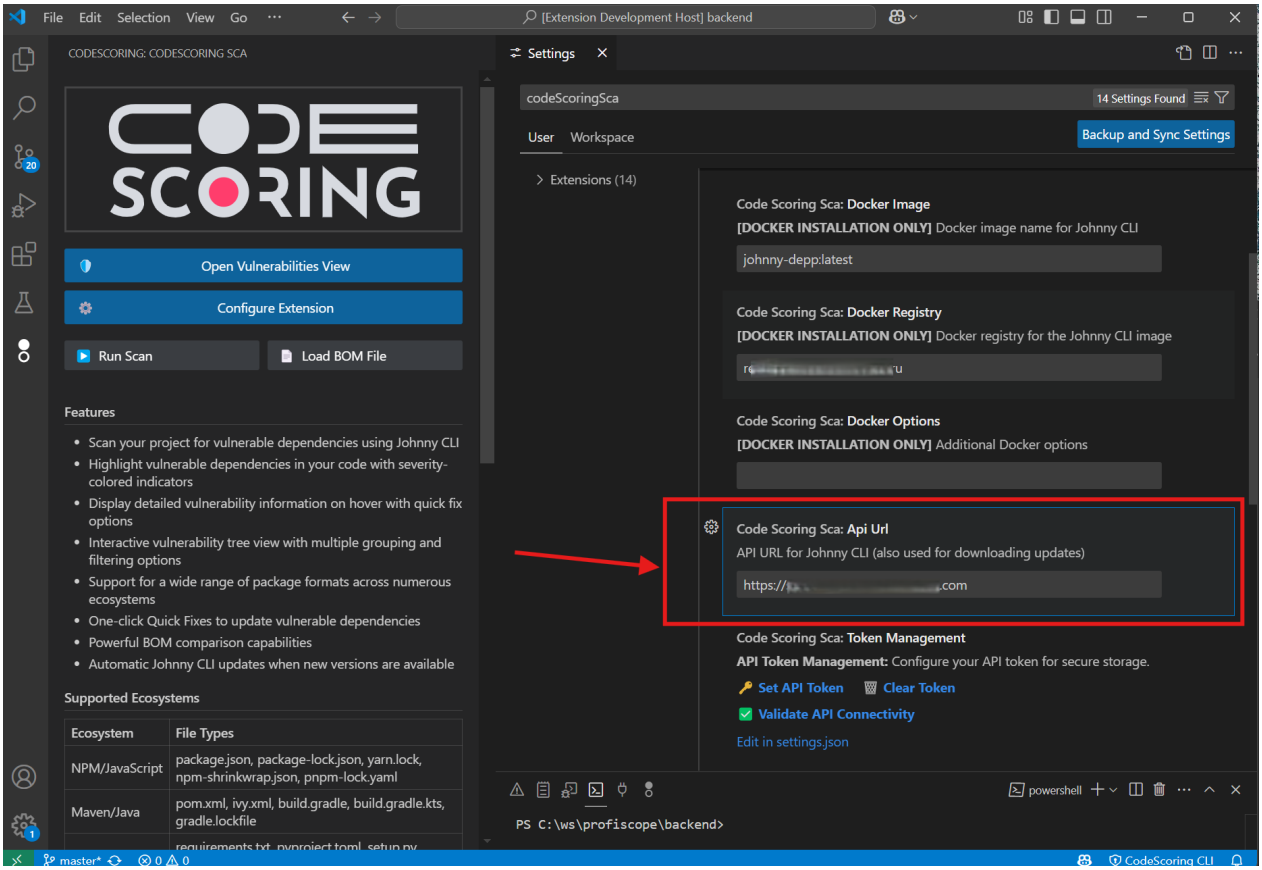
#Шаг 4: Настройте расширение

1. В боковой панели CodeScoring SCA нажмите кнопку **"Configure Extension"**
2. Откроется страница настроек CodeScoring SCA



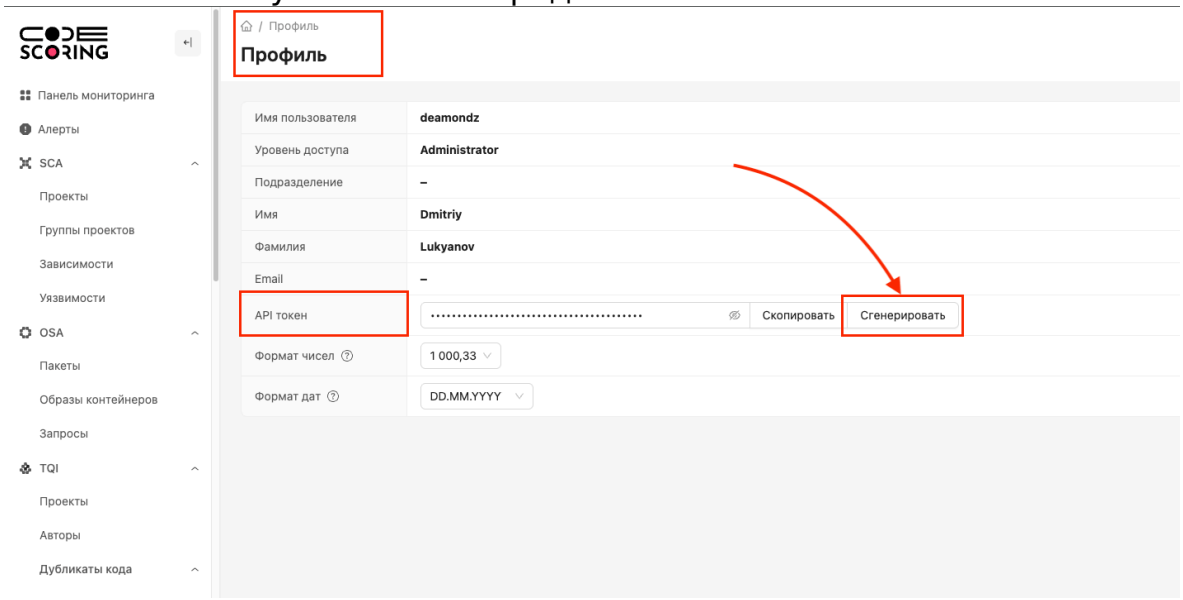
#4.1 Проверьте URL API

1. В настройках найдите поле **API URL**
2. Вам необходимо использовать URL CodeScoring, установленного в Вашей организации. При необходимости, обратитесь к администратору.

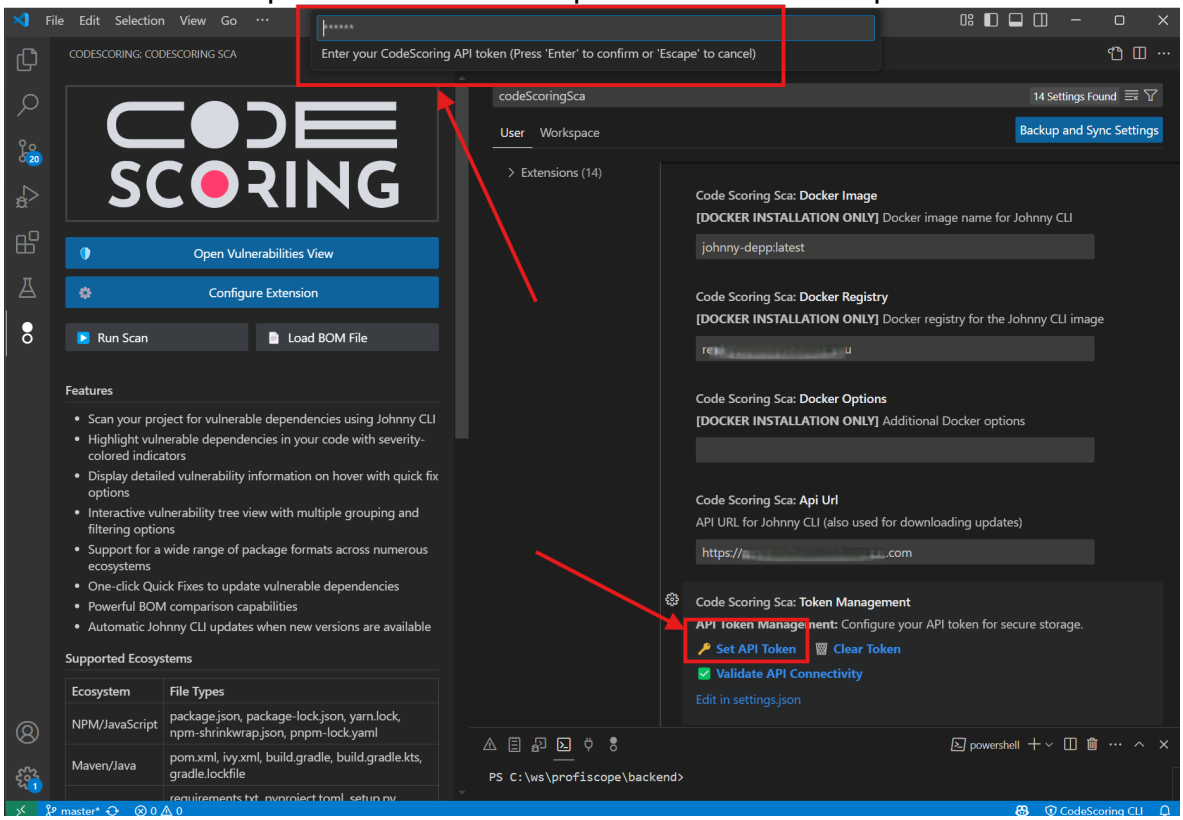


#4.2 Сгенерируйте и установите токен API

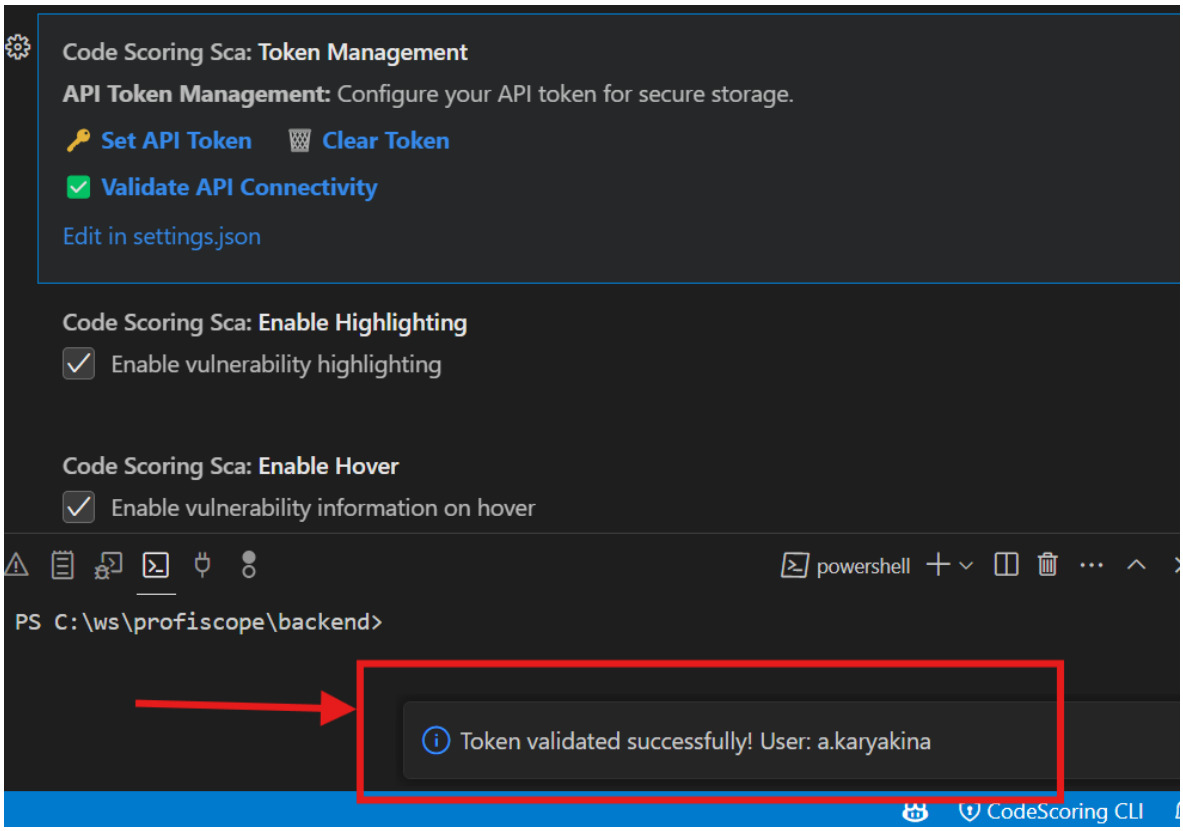
1. Откройте веб-браузер и перейдите по адресу: `<API URL>/cabinet/profile`
2. Войдите в свою учетную запись CodeScoring
3. Убедитесь, что вы находитесь на странице своего профиля
4. Найдите поле **"API token"**
5. Нажмите кнопку **"Generate"** рядом с полем токена API



6. Скопируйте значение сгенерированного токена API
7. Вернитесь к настройкам VS Code
8. Нажмите на ссылку **"Set API Token"** в настройках плагина
9. Вставьте скопированный токен при появлении запроса

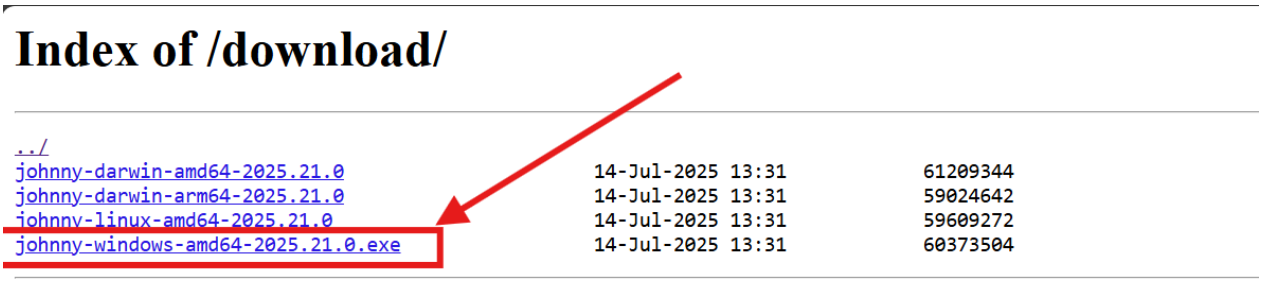


10. Плагин должен отобразить подтверждение, что токен действителен



#4.3 Загрузите Johnny CLI

1. Перейдите на страницу релизов: <API URL>/download/ (обратите внимание на завершающую косую черту)
2. Загрузите последний релиз исполняемого файла в зависимости от Вашей операционной системы



#4.4 Настройте Johnny CLI

Существует три способа получить Johnny CLI для анализа Ваших зависимостей с помощью нашего сервиса.

4.4.1 Локальная установка

Предварительные требования:

- Файл запуска Johnny CLI должен быть загружен и сделан исполняемым в системе
- Операционная система должна разрешать запуск исполняемого файла (для проверки запустите один раз файл в консоли с параметром --help вручную)

Шаги настройки:

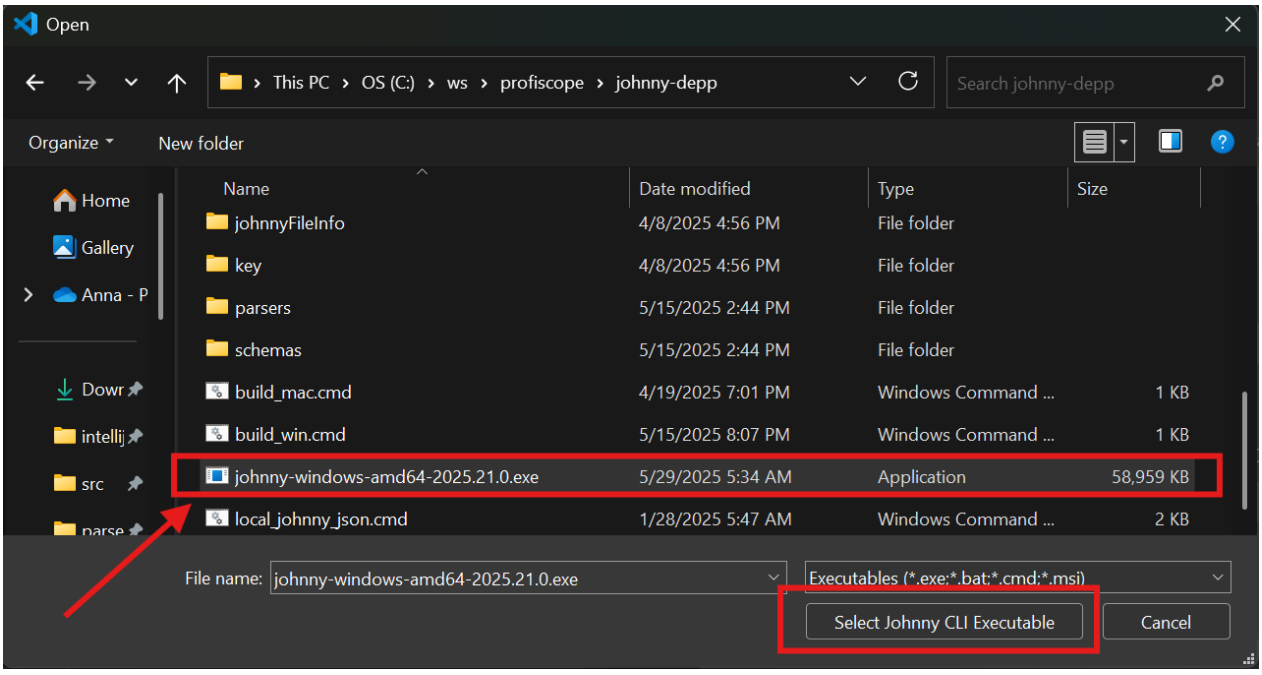
1. Установите тип установки на Local

- В настройках VS Code измените `platformType` на `local`
- Или в settings.json:

```
"codescoringSca.platformType": "local"
```

1. В настройках найдите поле `Codescoring Sca: Johnny Cli Path`, помеченное **[LOCAL platform ONLY]**
2. Нажмите на ссылку действия **"Browse..."**
3. Перейдите к месту, где вы ранее загрузили Johnny CLI
4. Выберите исполняемый файл Johnny CLI

Примечание: Если плагин спросит об изменении прав доступа к файлу (chmod +x), нажмите **"Yes"**, чтобы разрешить плагину сделать файл исполняемым.



4.4.2 Автоматическая загрузка клиента

Предварительные требования:

- Должен быть настроен API URL

- Должен быть настроен токен API и валидация должна пройти успешно

Шаги настройки:

1. Установите тип установки на Local

- В настройках VS Code измените `installationType` на `local`
- Или в settings.json:

```
"codescoringSca.installationType": "local"
```

1. В настройках найдите поле `Codescoring Sca: Johnny Cli Path`, помеченное **[LOCAL INSTALLATION ONLY]**
2. Очистите, если необходимо, значение в этом поле (пустое значение - это значение по умолчанию)
3. Теперь при первом запросе сканирования Johnny CLI будет загружен с API URL. Это позволит Вам автоматически получить обновление клиента, как только оно будет доступно. Загруженный клиент будет сохранен в следующем месте:
    - **Linux**: `~/.config/Code/User/globalStorage/CodeScoring.codescoring-sca/johnny`
    - **Windows**: `%APPDATA%\Code\User\globalStorage\CodeScoring.codescoring-sca\johnny.exe`
    - **MacOS**: `~/Library/Application Support/Code/User/globalStorage/CodeScoring.codescoring-sca/johnny`

4.4.3 Использование Docker

Установка Docker позволяет запускать Johnny CLI в изолированном контейнере, что полезно, когда вы не хотите устанавливать его непосредственно в Вашей системе.

Предварительные требования:

- Docker должен быть установлен и запущен в Вашей системе
- Ваш пользователь должен иметь права на выполнение команд Docker

Шаги настройки:

1. Установите тип установки на Docker

- В настройках VS Code измените `installationType` на `docker`
- Или в settings.json:

```
"codescoringSca.installationType": "docker"
```

1. Настройте Docker образ

- **Образ**: `johnny-depp:2025.29.0` (по умолчанию)
- **Реестр**: `<адрес-реестра-кодскоринг>`
- Пример полного пути к образу: `sample-codescoring-registry.com/johnny-depp:2025.29.0`

1. **Опционально: Дополнительные опции Docker** Добавьте пользовательские опции запуска Docker при необходимости:

   ```
   "codescoringSca.dockerOptions": "--memory=2g --cpus=2"
   ```

Как это работает:

- Клиент автоматически монтирует каталог Вашего проекта в контейнер
- Сканирование выполняется внутри контейнера, а результаты сохраняются в Вашем проекте
- Ручные команды Docker не требуются - расширение обрабатывает все автоматически

Пример конфигурации Docker:

```
{
  "codescoringSca.installationType": "docker",
  "codescoringSca.dockerImage": "johnny-depp:2025.29.0",
  "codescoringSca.dockerRegistry": "sample-codescoring-
registry.com",
  "codescoringSca.dockerOptions": ""
}
```

Устранение неполадок с установкой Docker:

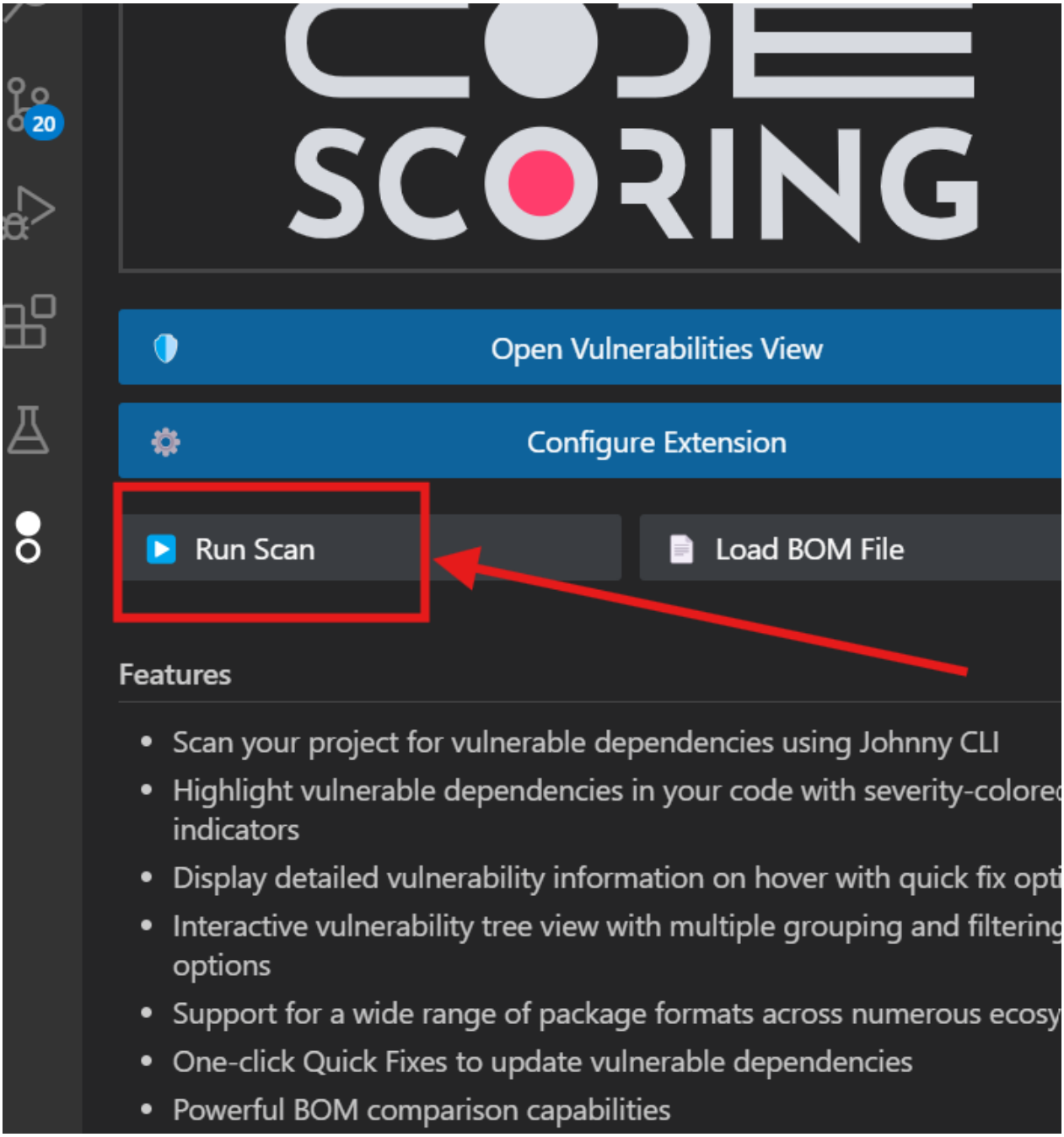
- **"Docker not found"**: Убедитесь, что Docker установлен и команда `docker` находится в Вашем PATH
- **Permission denied**: Добавьте Вашего пользователя в группу docker: `sudo usermod -aG docker $USER`
- **Image pull failed**: Проверьте учетные данные реестра и сетевое подключение
- **Container exits immediately**: Проверьте панель вывода VS Code для получения подробных сообщений об ошибках

#Шаг 5: Запустите первое сканирование

Теперь, когда расширение настроено, вы можете запустить первое сканирование уязвимых зависимостей:

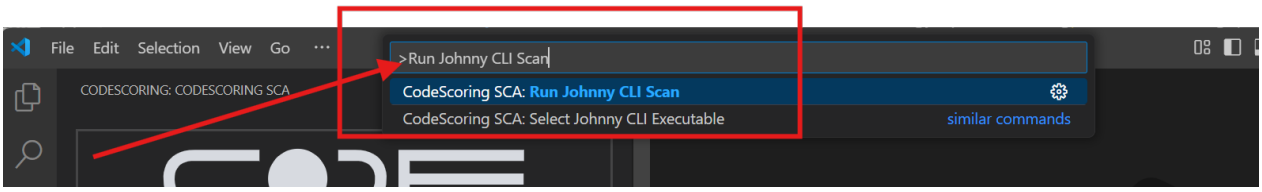
#Метод 1: Использование боковой панели

1. Откройте папку проекта в VS Code
2. Нажмите на иконку CodeScoring в панели действий
3. В боковой панели "CODESCORING: CODESCORING SCA" нажмите кнопку **"Run Scan"**



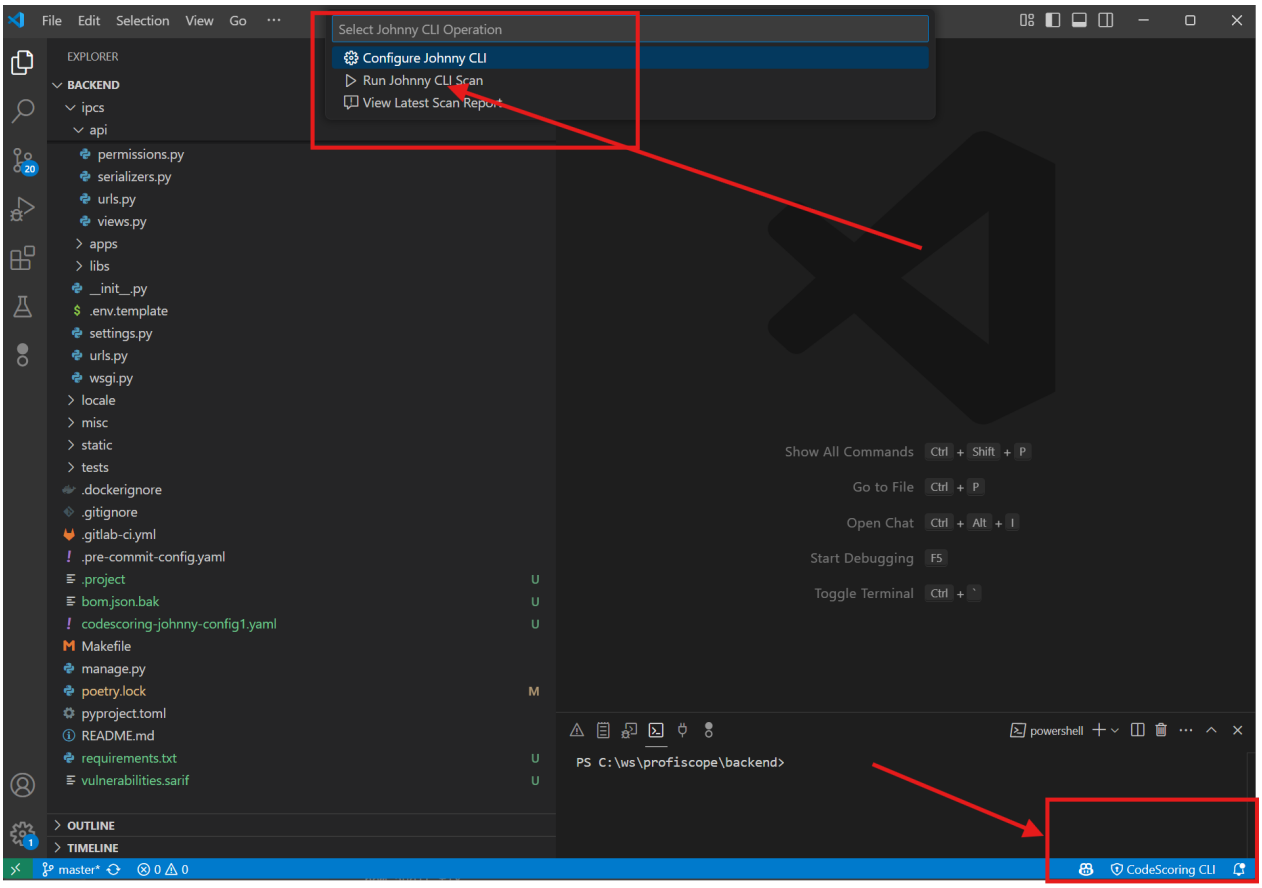
#Метод 2: Использование палитры команд

1. Нажмите `Ctrl+Shift+P` (или `Cmd+Shift+P` на MacOS), чтобы открыть палитру команд
2. Введите "Run Johnny CLI Scan"
3. Выберите **"CodeScoring SCA: Run Johnny CLI Scan"** из списка



#Метод 3: Использование строки состояния

1. Посмотрите на нижнюю строку состояния VS Code
2. Найдите индикатор **"CodeScoring CLI"**
3. Нажмите на него, чтобы открыть меню Johnny CLI
4. Выберите **"Run Johnny CLI Scan"** из списка



#Шаг 6: Тонкая настройка конфигурации сканирования

После завершения сканирования вы увидите новый каталог `.codescoring`, содержащий:

1. `config.yaml` - файл конфигурации для Johnny CLI, вы можете прочитать о нем в этом разделе. Вы можете менять этот файл, так как он никогда не будет перезаписан.
2. `report.html` - отчет, сгенерированный в формате html, содержащий вывод Johnny CLI в формате цветной таблицы. Перезаписывается во время каждого сканирования.
3. `bom.json` - файл результатов сканирования, созданный Johnny CLI в формате cyclone-dx 1.6, он будет загружен автоматически и

показан в панели уязвимостей для любой открытой в VSCode директории, в которой существует .codescoring/bom.json
4. `bom.json.N` - где N - это ревизия сканирования, т.е. 0 - предыдущее сканирование, а 5 (например) - самое первое сканирование, и bom.json.0 будет использоваться для сравнения с bom.json (и показан в дереве DIFF), если он существует на момент открытия новой папки

›
your-project
› .codescoring
config.yaml# Конфигурация сканирования
report.html# Последний отчет сканирования
bom.json# Текущие уязвимости
bom.json.0# Предыдущее сканирование (сравнение)
bom.json.1# Более старые сканирования...
package.json# Ваши зависимости
...Ваш код ...

#Пример использования конфигурации сканирования CodeScoring

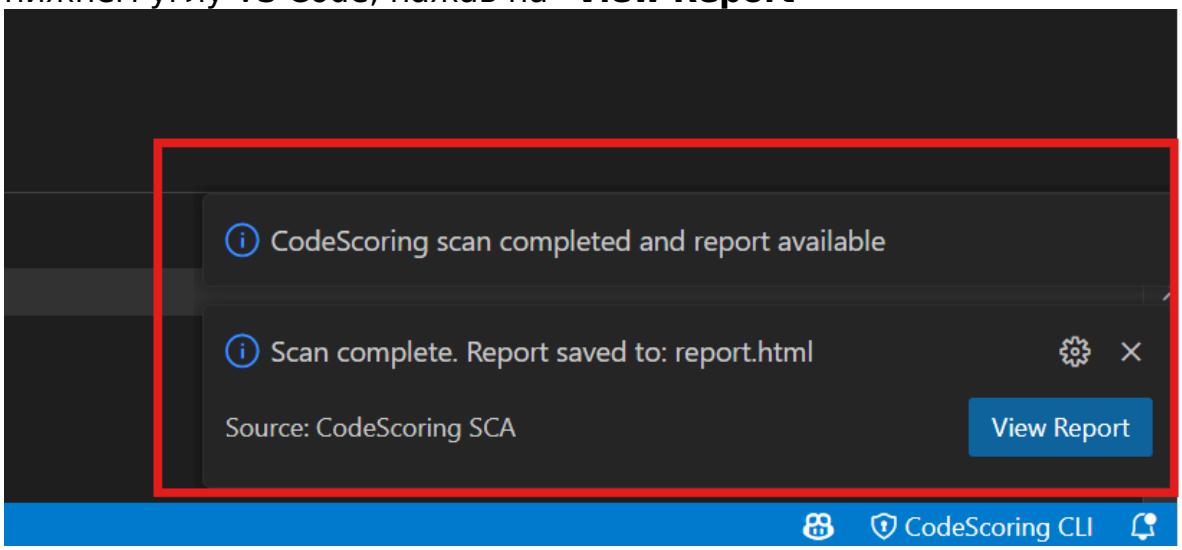
Отредактируйте `.codescoring/config.yaml` для настройки:

```
scan:
  general:
    ignore: # Директории для пропуска
      - node_modules
      - .git
      - test
    with-hashes: true # Включить хеши файлов для точного сопоставления
    only-hashes: false # Использовать только обнаружение на основе хешей
  dir:
    no-recursion: false # Предотвращает рекурсивное сканирование корневой директории
```

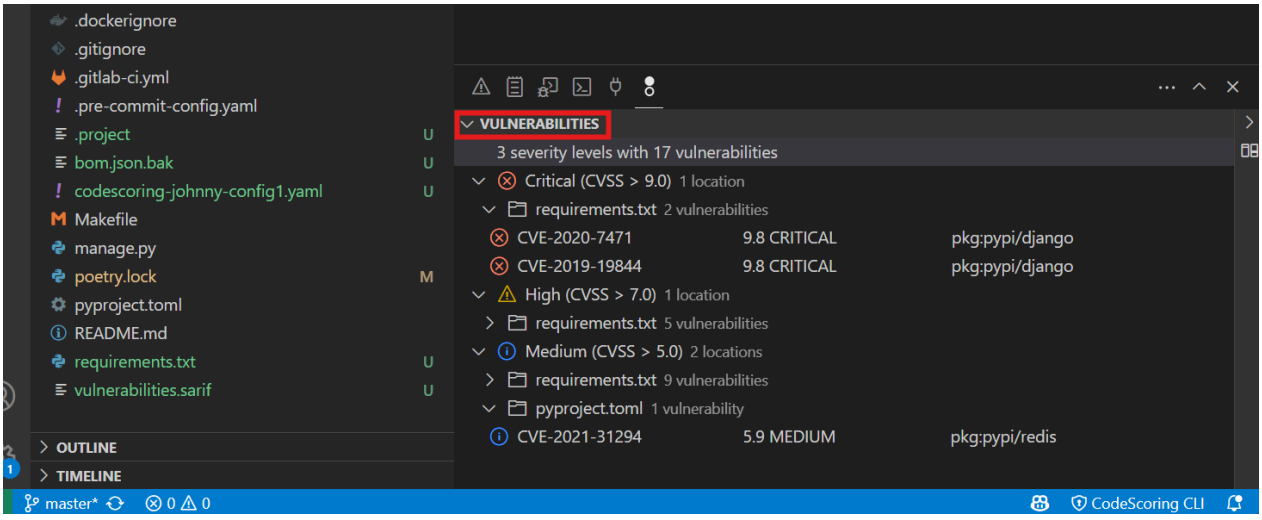
#Шаг 7: Просмотр результатов сканирования

После завершения сканирования:

1. **Проверьте уведомления**: Посмотрите на уведомления в правом нижнем углу VS Code, нажав на **"View Report"**



2. **Откройте дерево уязвимостей** (открывается само по умолчанию): Используйте кнопку **"Open Vulnerabilities View"** из боковой панели
3. **Просмотрите уязвимости**: Панель уязвимостей покажет все обнаруженные проблемы безопасности в Ваших зависимостях
4. **Изучите детали**: Вы можете нажимать на отдельные уязвимости, чтобы увидеть подробную информацию
5. **Примените исправления**: Используйте опции быстрого исправления для обновления уязвимых зависимостей
6. **Откройте панель алертов**: Используйте кнопку **Open Alerts View** в боковом меню плагина для открытия окна со списком алертов по сработавшим политикам



#7.1 Подсветка уязвимостей

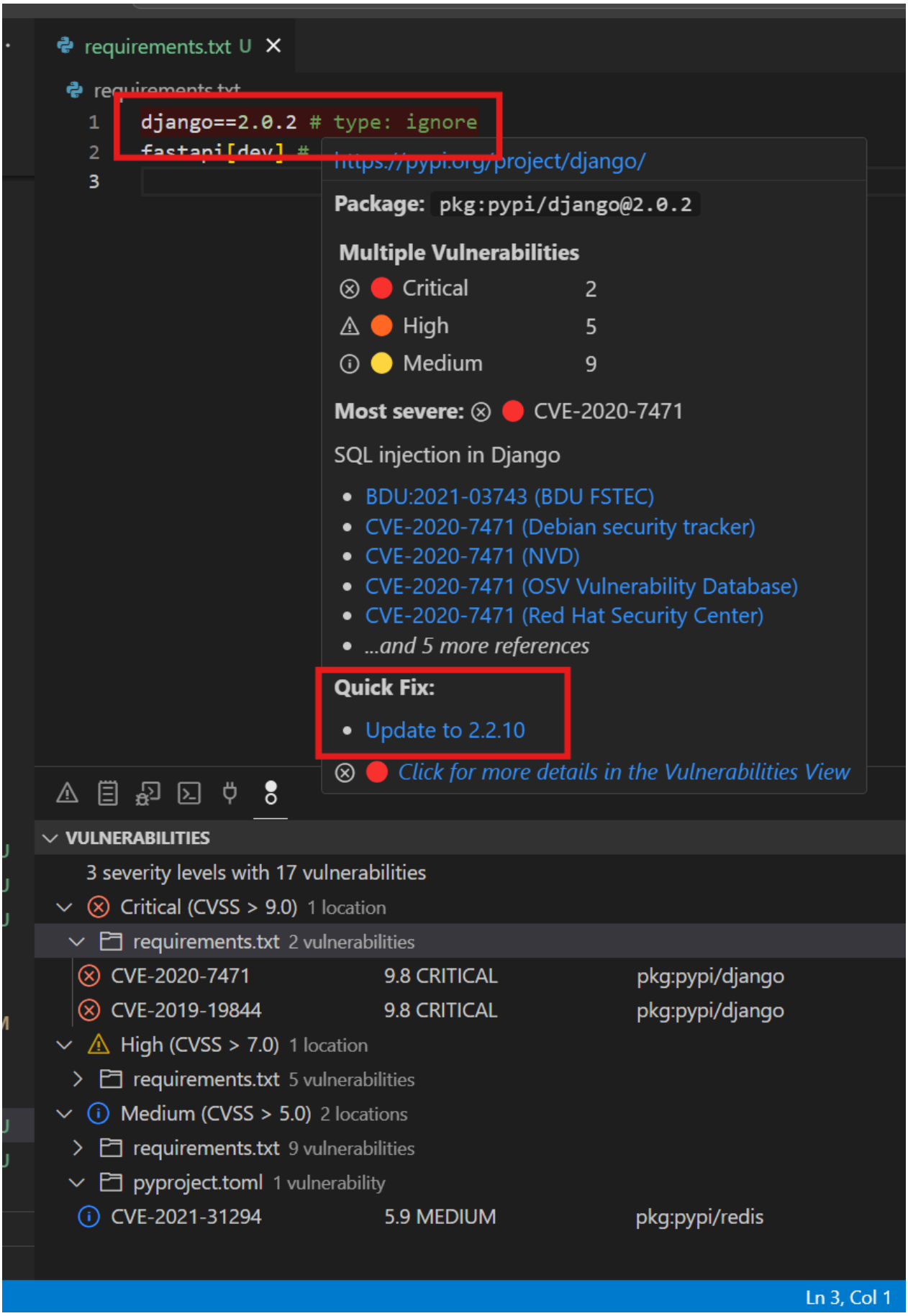
- **Подсветка в коде**: Уязвимые зависимости подсвечиваются по мере ввода

- **Цвета критичности**:
  - Критический (красный)
  - Высокий (оранжевый)
  - Средний (желтый)
  - Низкий (синий)
- **Поддержка нескольких файлов**: Работает со всеми поддерживаемыми типами файлов
- **Наведите курсор** на подсвеченные зависимости, чтобы увидеть детали уязвимостей

#7.2 Информация при наведении

При наведении курсора на подсвеченные зависимости отображается:

- **Идентификатор уязвимости**: Номер CVE со ссылкой
- **Критичность**: Оценка CVSSv3 и уровень
- **Описание**: Что затрагивает уязвимость
- **Ссылки на источники**: Информация об официальной регистрации уязвимости, патчах, эксплойтах и прочем
- **Быстрое исправление**: Опция обновления в один клик


requirements.txt U
requirements.txt
1 django==2.0.2 # type: ignore
2 fastapi[dev] #
3
https://pypi.org/project/django/
Package: pkg:pypi/django@2.0.2
Multiple Vulnerabilities
Critical 2
High 5
Medium 9
Most severe: CVE-2020-7471
SQL injection in Django
BDU:2021-03743 (BDU FSTEC)
CVE-2020-7471 (Debian security tracker)
CVE-2020-7471 (NVD)
CVE-2020-7471 (OSV Vulnerability Database)
CVE-2020-7471 (Red Hat Security Center)
...and 5 more references
Quick Fix:
Update to 2.2.10
Click for more details in the Vulnerabilities View
VULNERABILITIES
3 severity levels with 17 vulnerabilities
Critical (CVSS > 9.0) 1 location
requirements.txt 2 vulnerabilities
CVE-2020-7471 9.8 CRITICAL pkg:pypi/django
CVE-2019-19844 9.8 CRITICAL pkg:pypi/django
High (CVSS > 7.0) 1 location
requirements.txt 5 vulnerabilities
Medium (CVSS > 5.0) 2 locations
requirements.txt 9 vulnerabilities
pyproject.toml 1 vulnerability
CVE-2021-31294 5.9 MEDIUM pkg:pypi/redis
Ln 3, Col 1

#7.3 Панель уязвимостей

7.3.1 Структура древовидного представления

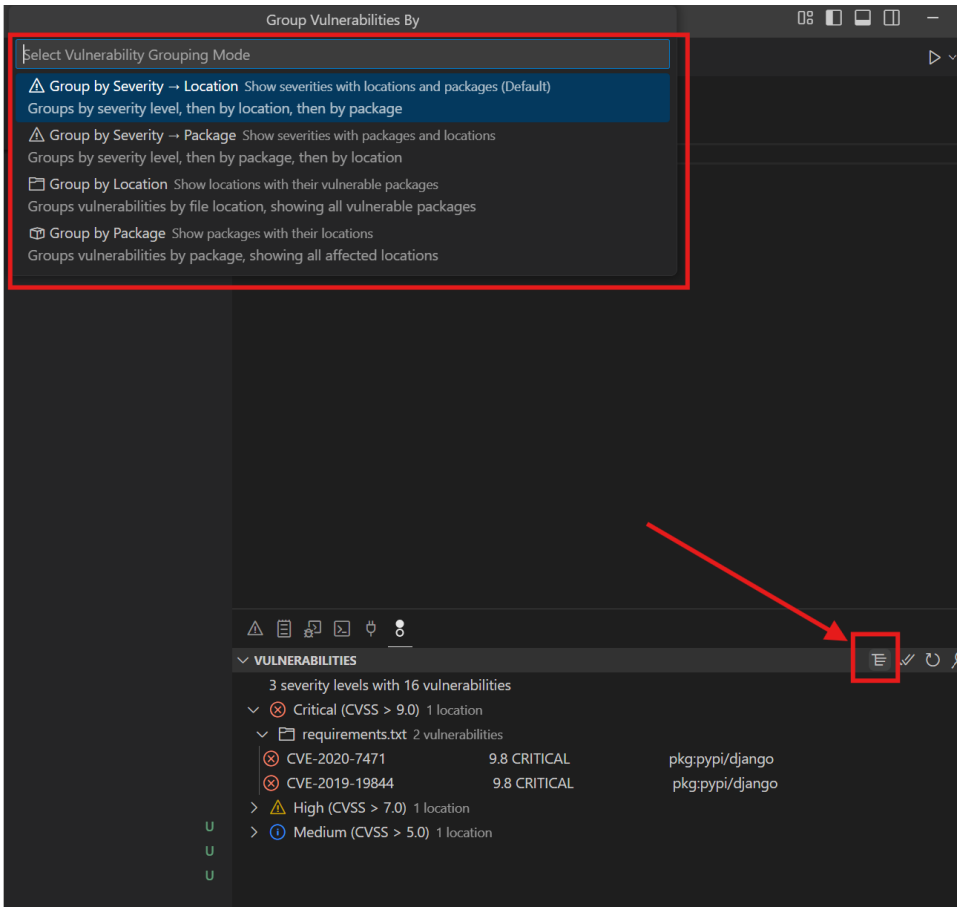
```
 Уязвимости (247)
├── 🔴 Критические (12)
│   ├── CVE-2023-1234 - Удаленное выполнение кода
│   │   ├── lodash@4.17.20 → 4.17.21
│   │   └── package.json:15
│   └── ...
├── 🟠 Высокие (45)
├── 🟡 Средние (89)
└── 🔵 Низкие (101)
```

7.3.2 Опции группировки

- Используйте панель уязвимостей для фильтрации по критичности, пакету или другим критериям
- Группируйте уязвимости по различным категориям для лучшей организации

Изменение группировки через кнопку панели инструментов или команду:

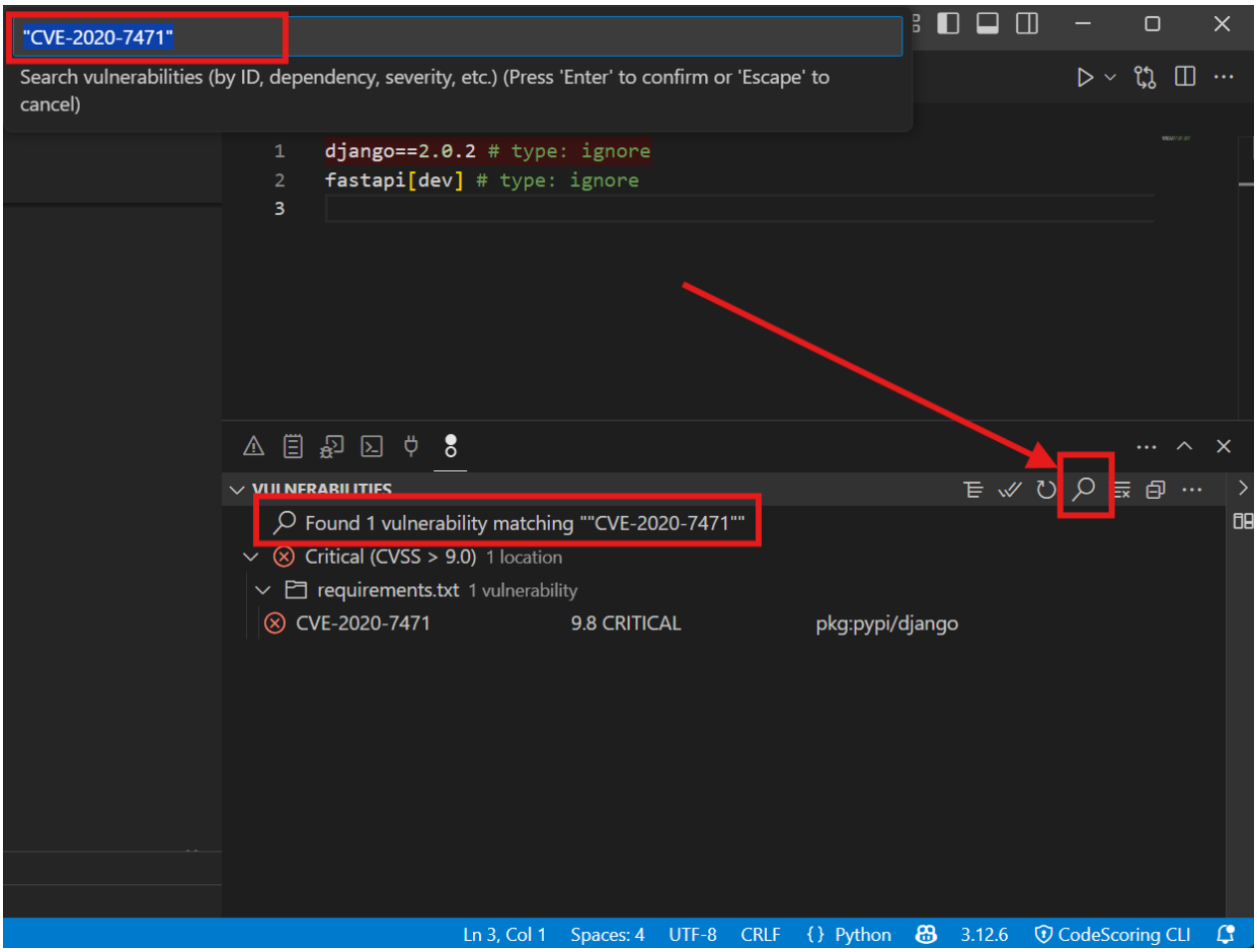
- **По критичности -> Расположению** (по умолчанию): По уровню критичности, подгруппировка по пакету
- **По критичности -> Пакету (PURL)**: По уровню критичности, подгруппировка по имени пакета в алфавитном порядке
- **По расположению**: Группировка по пути к файлу
- **По пакету (PURL)**: Группировка по пакету



#7.4 Поиск и фильтрация

Возможности поиска

- **Несколько полей**: Поиск по:
  - Имени пакета (например, "lodash")
  - Идентификатору CVE (например, "CVE-2023")
  - Пути к файлу (например, "frontend/")
  - Уровню критичности
- **Нечеткое сопоставление**: Находит частичные совпадения, если не используются кавычки
- **Точное совпадение**: Используйте кавычки
- **Без учета регистра**: Не требуется точный регистр



#7.5 Быстрые исправления

Быстрые исправления

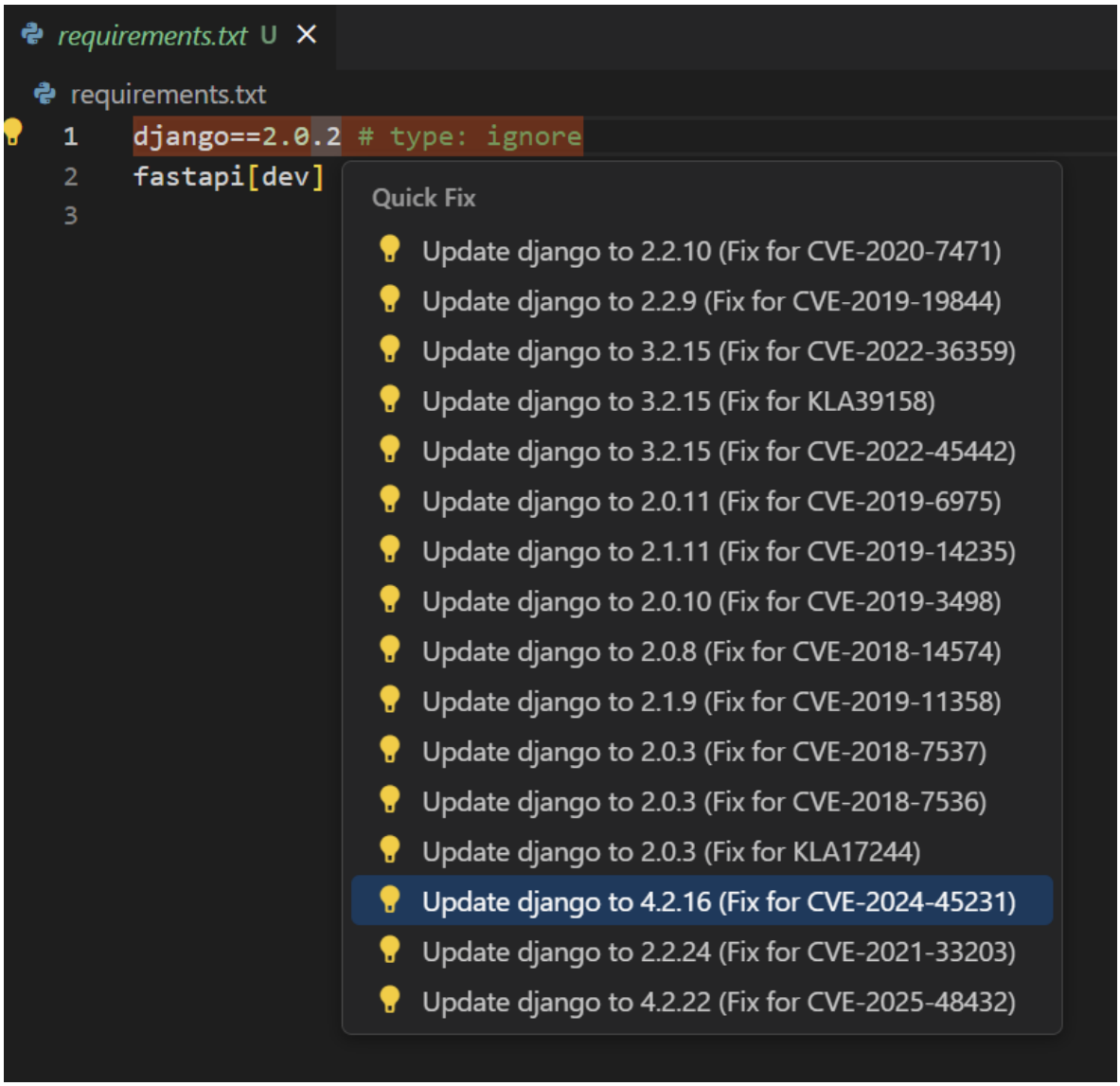
- Нажмите на предлагаемые обновления версий при наведении мышкой на уязвимый компонент, чтобы автоматически обновить на последнюю версию зависимости
- При нажатии `Ctrl+.` (`Cmd+.` на MacOS), когда курсор находится на уязвимом компоненте, Вам предложится выбрать конкретную версию для обновления (при наличии нескольких)

Индивидуальные исправления

- Наведите курсор мыши на уязвимую зависимость
- Нажмите на предлагаемую версию во всплывающей подсказке
- После обновления Вы увидите сообщение об успешном изменении

или

- Наведите курсор клавиатуры на уязвимую зависимость
- Нажмите `Ctrl+.` (`Cmd+.` на MacOS)
- Выберите подходящую версию из списка, если их несколько



или

- **Кнопка Fix Selected**: Выберите один уязвимый компонент или уязвимость, принадлежащую этому компоненту, и компонент будет обновлен до последней безопасной версии

Массовые исправления

- **Кнопка Fix All**: Обновляет все уязвимые компоненты с доступными исправлениями, видимые в данный момент в дереве, при этом учитывает примененные фильтры (установленные с помощью кнопки с лупой на панели инструментов)

#7.6 Работа с файлами BOM

Автозагрузка

Расширение автоматически загружает файлы BOM из:

1. `.codescoring/bom.json` (основной)
2. `bom.json` (корневой каталог)

Ручные операции

- **Загрузить BOM**: `Ctrl+Shift+P` (или `Cmd+Shift+P` на MacOS) → "Load BOM File"
- **Закрыть BOM**: Очищает все данные об уязвимостях и освобождает память

#7.7 Сравнение BOM

Сравнивается полный состав всех компонентов, а не только уязвимых. С помощью сравнения можно увидеть различия в полном перечне используемых компонентов и отследить изменения версий. При этом старые версии компонентов будут показаны как удаленные, а новые - как добавленные. Чтобы удобнее было отслеживать изменения в версиях компонентов, можно сгруппировать их по имени пакета (см ниже).

Автоматическое сравнение

При открытии проекта:

- Загружает текущий BOM (`bom.json`)
- Сравнивает с предыдущим (`bom.json.0`)
- Показывает уведомление об изменениях

Ручное сравнение при открытом BOM

1. Имеется загруженный BOM (целевой)
2. Выполните команду "Compare BOMs"
3. Выберите базовый файл BOM
4. Отобразится результат сравнения в панели DIFF

Ручное сравнение двух произвольных BOM

1. Закройте BOM при необходимости
2. Выполните команду "Compare BOMs"
3. Выберите базовый файл BOM для сравнения
4. Выберите целевой файл BOM для сравнения с базовым
5. Отобразится результат сравнения в панели DIFF

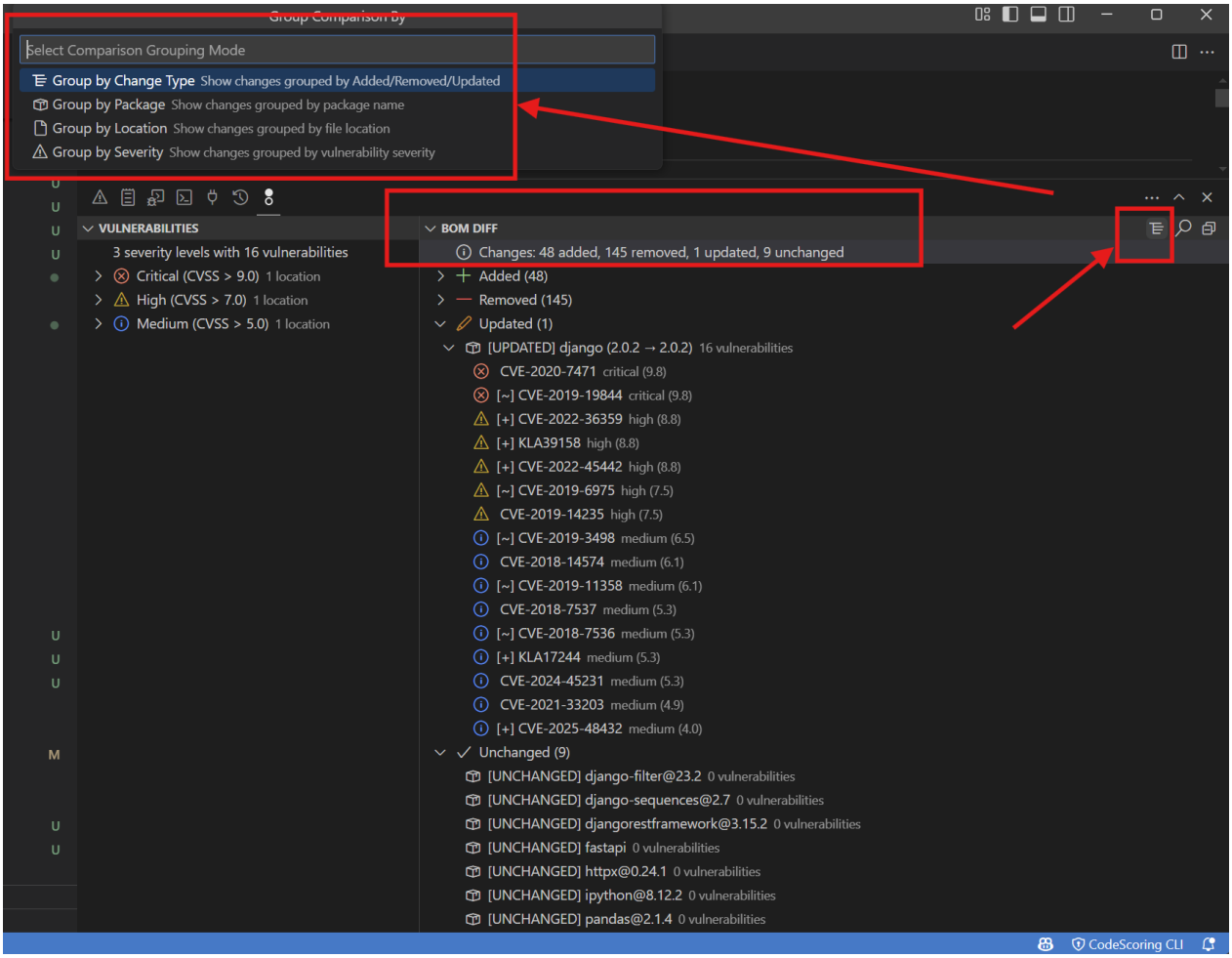
Представления сравнения

```
 BOM DIFF (Изменения: 23 добавлено, 15 удалено, 45 обновлено,
73 без изменений)
├── + Добавлено (23)
│   ├── [ADDED] react@18.1.3 0 уязвимостей
│   └── ...
├── − Удалено (15)
├──  Обновлено (45)
│   ├── [UPDATED] lodash: 4.17.20
```

```
│   └── ...
└── ✓ Без изменений (73)
```

Опции группировки сравнения

- **По типу изменения**: Добавлено/Удалено/Обновлено/Без изменений (примечание: обновлено означает, что количество уязвимостей было обновлено для компонента)
- **По пакету**: Алфавитная группировка пакетов без версии. Это наиболее полезная группировка для просмотра обновленных компонентов (добавленная версия и удаленная версия будут сгруппированы вместе)
- **По расположению**: Группировка по пути к файлу
- **По критичности**: Группировка по влиянию уязвимости



Фильтрация сравнения

Вы также можете искать/фильтровать дерево BOM DIFF, чтобы сосредоточиться на интересующих Вас компонентах. Чтобы очистить поиск, нажмите кнопку "Clear Comparison Search" в правом верхнем углу BOM DIFF.

#7.8 Отчеты

Отчеты сканирования

- **Автоматически генерируются**: Создаются после каждого сканирования
- **Расположение**: `.codescoring/report.html`
- **Формат**: Цветной HTML
- **Содержимое**:
  - Статус сканирования
  - Выполненная команда
  - Сводка результатов
  - Найденные уязвимости
  - Предупреждения от политик
  - Сообщения об ошибках

Просмотр отчетов

- **Команда**: "View Latest Scan Report"
- **Открывается в**: Предварительном просмотре HTML в VS Code
- Адаптируется к выбранной цветовой схеме VSCode в момент генерации

#7.9 Панель алертов

В панели **Alerts** представлена информация по алертам для сработавших политик по итогам анализа. Для каждого алерта показывается:

- Название политики
- Уровень алерта
- Статус блокировки
- Список пакетов с критериями

#Шаг 8: Настройки и кастомизация

Список доступных настроек

Настройка	Описание	По умолчанию
`apiUrl`	Ваш URL с установленной CodeScoring	
`apiToken`	API токен. Безопасно сохраняется	*(устанавливается командой)*
`platformType`	local или docker	`local`
`johnnyCliPath`	Путь к Johnny CLI (пустой для автозагрузки)	*(автозагрузка)*

Настройка	Описание	По умолчанию
`dockerImage`	Имя docker образа с Johnny CLI	`johnny-depp:2025.29.0`
`dockerRegistry`	Реестр с образом Johnny CLI	*(предоставляется заботой)*
`dockerOptions`	Дополнительные опции docker	
`enableHighlighting`	Показывать подсветку в коде	`true`
`enableHover`	Показывать всплывающие подсказки	`true`
`enableQuickFixes`	Разрешить исправления в один клик	`true`
`showVulnerabilityHeaders`	Отображать заголовки для колонок уязвимостей	`false`
`paginationSize`	Количество элементов на странице	`100`
`batchProcessingSize`	Количество элементов для обработки за раз	`100`
`severityColors`	Пользовательское сопоставление цветов	*(цвета по умолчанию)*

Горячие клавиши

Настройте в параметрах VS Code, например:

```
{
  "key": "ctrl+shift+s",
  "command": "codescoring-sca.runJohnnyCLI"
}
```

#Устранение неполадок

#Логи расширения

- Для подробного понимания шагов функциональности проверьте лог файл расширения, расположенный:
  - Windows: `%USERPROFILE%\.vscode\extensions\codescoring-sca-[version]\out\logs\extension.log`
  - macOS/Linux: `~/.vscode/extensions/codescoring-sca-[version]/out/logs/extension.log`

#Распространенные проблемы

Проблемы сканирования

Проблема	Решение
Сканирование зависает	Попробуйте запустить файл cli вручную, убедитесь, что операционная система и антивирус позволяют ему работать, проверьте интернет-соединение, проверьте токен API
Нет результатов	Убедитесь, что проект имеет файлы зависимостей, проверьте вывод report.html
Частичные результаты	Проверьте шаблоны игнорирования в конфигурации

Проблемы отображения

Проблема	Решение
Нет подсветки	Включите в настройках, перезагрузите окно
Неправильные цвета	Проверьте совместимость темы
Отсутствует панель	Вид → Открыть вид → Уязвимости
Не показываются всплывающие подсказки	Включите наведение в настройках

Проблемы исправления

Проблема	Решение
Исправление не удается	Проверьте права на запись
Неправильная версия	Вручную укажите в файле пакета
Ломает проект	Используйте контроль версий, откатитесь
Конфликты	Исправляйте по одному

Свяжитесь с отделом заботы: support@codescoring.ru

#Безопасность и конфиденциальность, обработка данных

- **Локальное сканирование**: Код не отправляется на серверы
- **Связь с API**: Передаются только метаданные (конфигурационные файлы Вашего пакетного менеджера)
- **Хранение токенов**: Безопасное хранилище учетных данных VS Code

#Рекомендации

#Рекомендации по рабочему процессу

1. **Начальная настройка**: Полное сканирование при запуске проекта
2. **Обзоры**: Сравнивайте BOM между версиями
3. **CI/CD**:

- **Перед коммитом**: Выполните полное сканирование
- **Поделитесь конфигурацией**: Закоммитьте `.codescoring/config.yaml`
- **Игнорируйте временные файлы**: Добавьте в `.gitignore`:

```
.codescoring/report.html
.codescoring/bom.json.*
```

- **Отслеживайте основной BOM**: Версионируйте `.codescoring/bom.json`
- **Стандартизируйте**: Стандартизируйте настройки расширения

#Интеграция с процессами разработки

1. **Code Review**:
    - Проверяйте изменения зависимостей
    - Требуйте исправления критических уязвимостей
    - Документируйте принятые риски

2. **Release Management**:
    - Генерируйте отчеты для каждого релиза
    - Отслеживайте улучшения безопасности
    - Планируйте обновления зависимостей
3. **Compliance**:
    - Экспортируйте BOM для аудита
    - Отслеживайте лицензии компонентов
    - Поддерживайте историю сканирований

#Плагин CodeScoring.SCA для IntelliJ based IDEs

Плагин предоставляет возможности анализа состава программного обеспечения (SCA) для IntelliJ IDEA и других IDE, основанных на IntelliJ движке, подсвечивая уязвимые зависимости в файлах вашего проекта и предоставляя подробную информацию об уязвимостях через интеграцию с Johnny CLI.

Плагин **CodeScoring.SCA** поддерживает версии IntelliJ IDEA **2024.1** и выше, а также все IDE на базе IntelliJ Platform (OpenIDE, GIGA IDE, PyCharm, WebStorm, PhpStorm, RubyMine, GoLand, CLion, Rider, Android Studio).

#Поддерживаемые экосистемы

#Языки и менеджеры пакетов

Экосистема	Файлы манифеста	Сгенерированные файлы версий	Особенности
Java/JVM	pom.xml, *.gradle, *.gradle.kts, ivy.xml	gradle.lockfile, gradle-dependency-tree.txt, maven-dependency-tree.txt	Полная поддержка Maven и Gradle
JavaScript/Node	package.json	package-lock.json, yarn.lock, npm-shrinkwrap.json, pnpm-lock.yaml	NPM, Yarn, PNPM
Python	setup.py, pyproject.toml, pipfile	requirements.txt, requirements.pip, Pipfile.lock, poetry.lock	Pip, Poetry, Pipenv
Ruby			

Экосистема	Файлы манифеста	Сгенерированные файлы версий	Особенности
	Gemfile, gems.rb, *.gemspec	Gemfile.lock, gems.locked	Bundler и RubyGems
Go	go.mod	go.sum	Go модули
Rust	Cargo.toml	Cargo.lock	Cargo
PHP	composer.json	composer.lock	Composer
C#/.NET	*.csproj, packages.config, *.nuspec, paket.dependencies	packages.lock.json, project.assets.json, paket.lock, project.lock.json	NuGet и Paket
Swift	Package.swift	Package.resolved	Swift Package Manager
Objective-C	Podfile, *.podspec	Podfile.lock	CocoaPods для iOS/macOS
C/C++	conanfile.txt, conanfile.py	conan.lock	Conan менеджер пакетов
Conda	environment.yml, meta.yml, environment.yaml, meta.yaml	conda-lock.yml	Conda окружения

#Обнаружение файлов по умолчанию

- **Автоматическое**: Сканирует все поддерживаемые файлы
- **Рекурсивное**: Ищет в подкаталогах проекта

Точная настройка сканирования производится с помощью модификации config.yaml файла

#Начало работы

#Предварительные требования

Перед началом убедитесь, что у вас есть:

- IntelliJ IDEA 2024.1 или новее (или любая совместимая поддерживаемая IDE на основе IntelliJ Platform)
- Доступ к установке CodeScoring с активными учетными данными
- Дистрибутив codescoring-intellij плагина (.zip файл)

#Требуемые разрешения

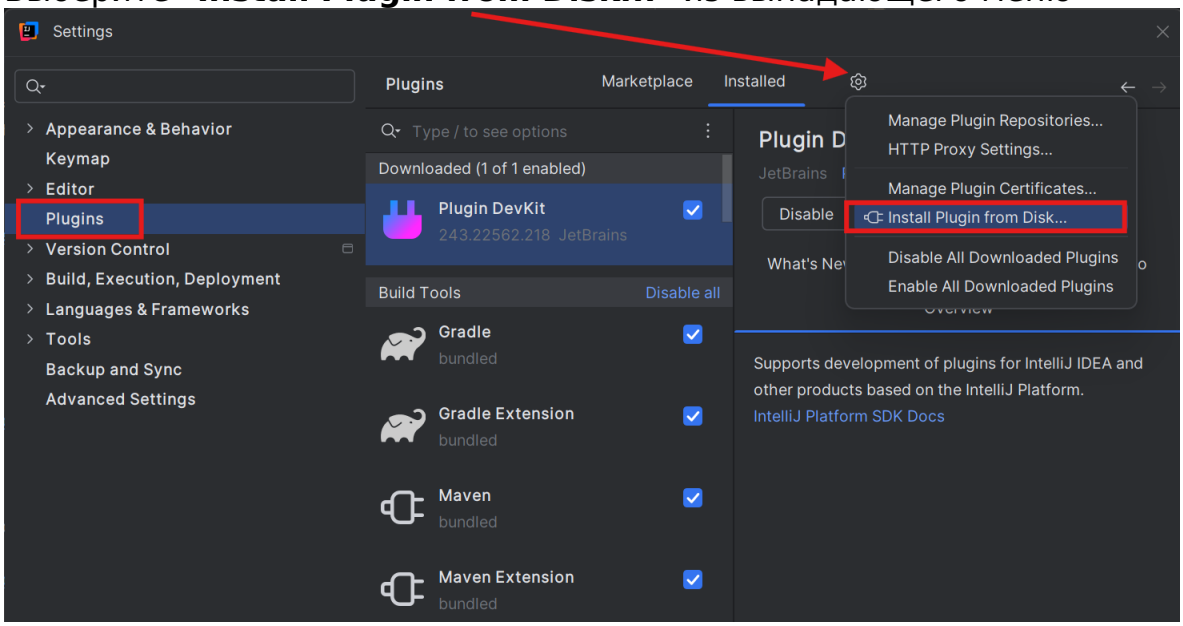
- **Файловая система**: Чтение файлов проекта, запись файлов .codescoring, загрузка исполняемого файла, выполнение загруженного CLI
- **Сеть**: Связь с CodeScoring API
- **API VS Code**: Интеграция с редактором

#Шаг 1: Загрузите плагин

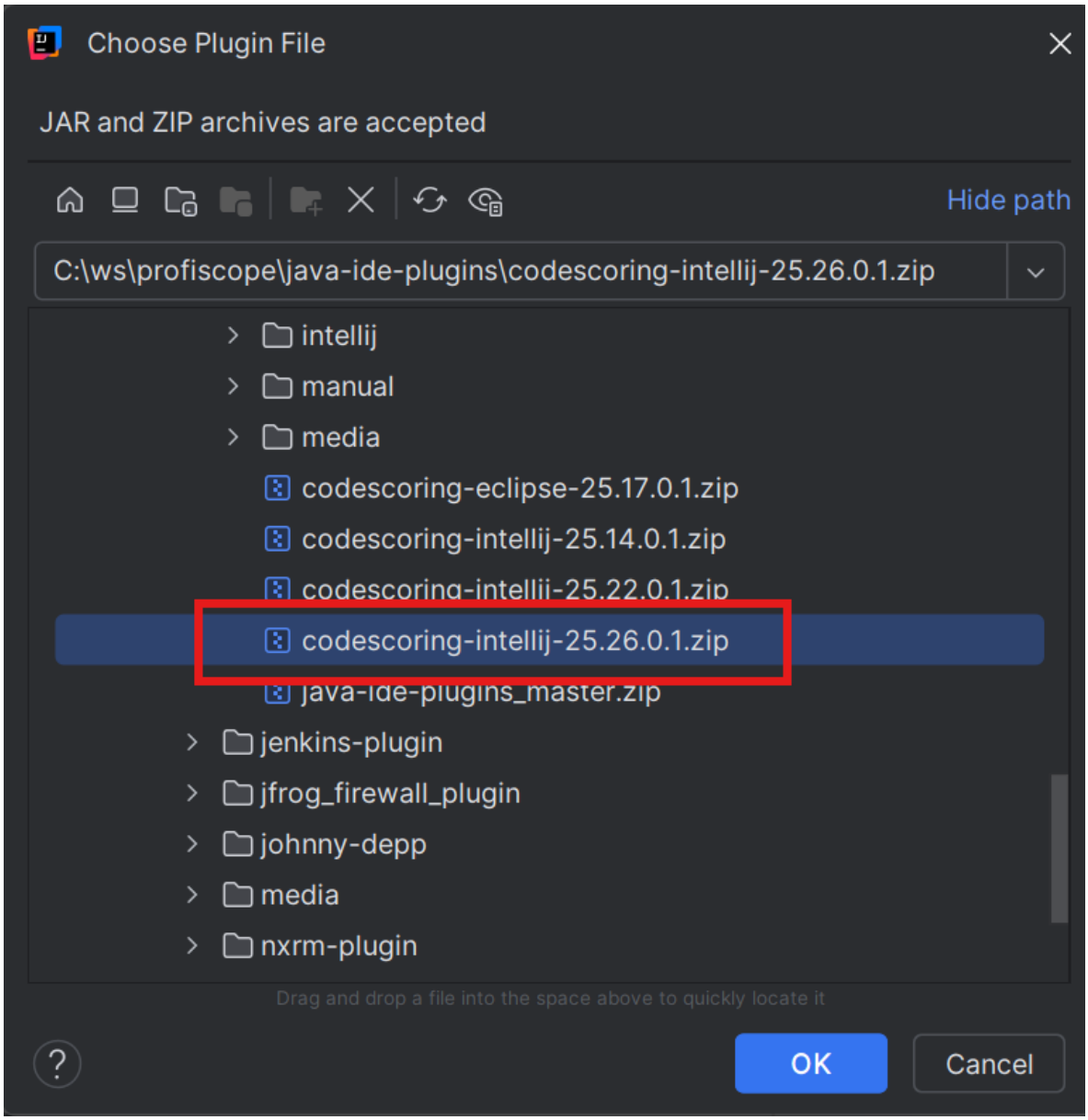
Плагин поставляется в виде файла `codescoring-intellij-<version>.zip`.

#Шаг 2: Установите плагин из ZIP файла

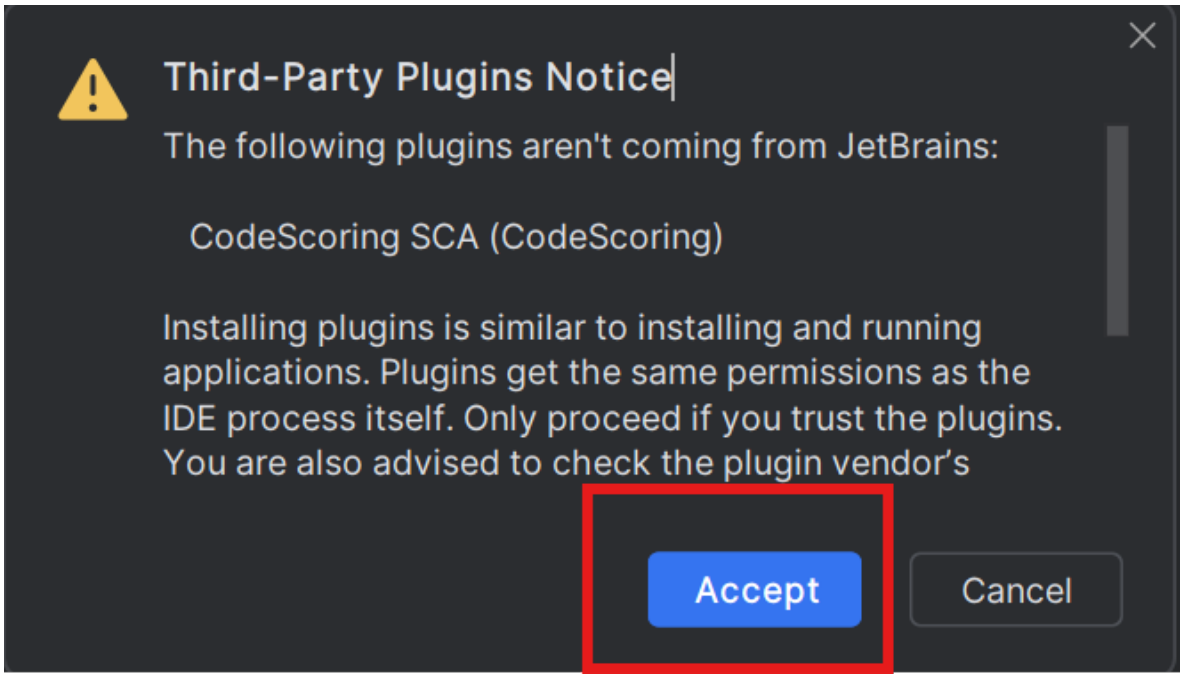
1. Откройте IntelliJ based IDE
2. Перейдите в **File → Settings** (или **<IDE Name> → Preferences** на macOS)
3. В диалоге настроек выберите **Plugins** в левой боковой панели
4. Нажмите на **значок шестеренки (⚙)** в верхней части панели плагинов
5. Выберите **"Install Plugin from Disk..."** из выпадающего меню



6. Найдите место, куда вы загрузили файл `.zip`
7. Выберите файл и нажмите **"OK"**



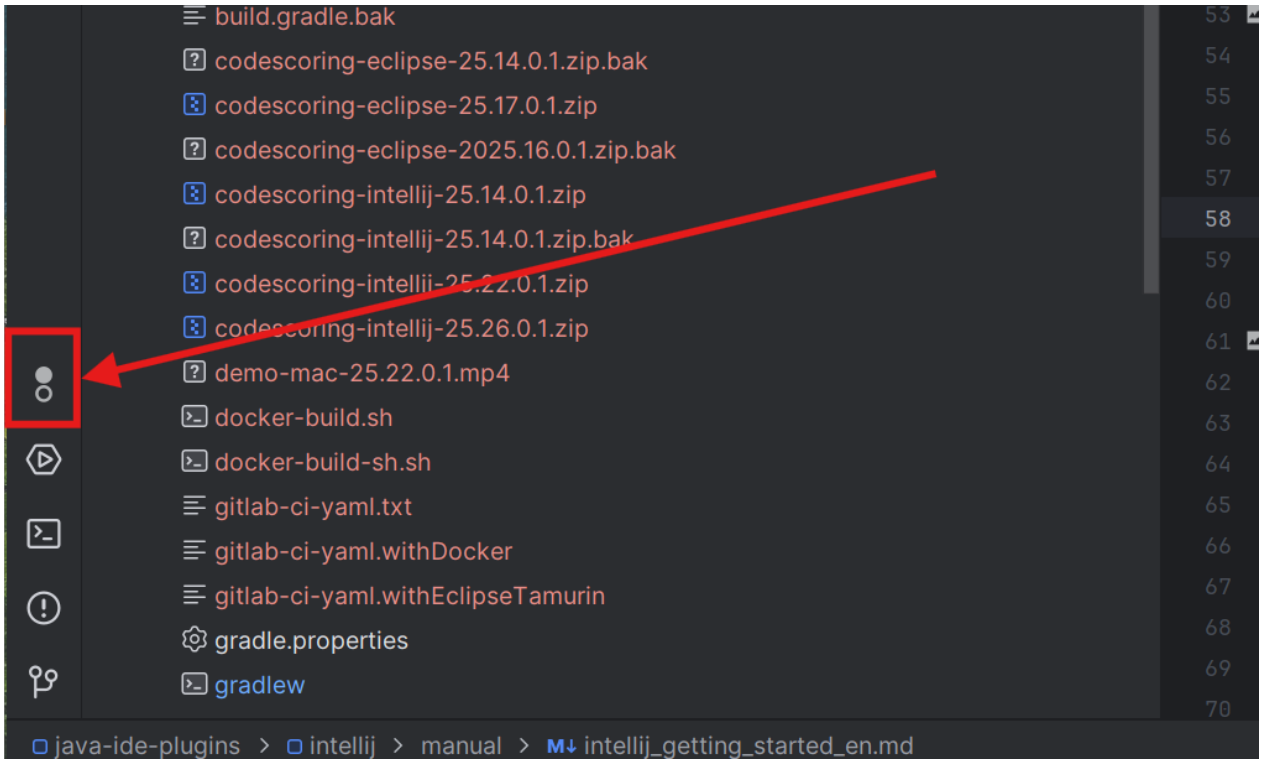
8. Дождитесь завершения установки
9. Подтвердите установку плагина от CodeScoring



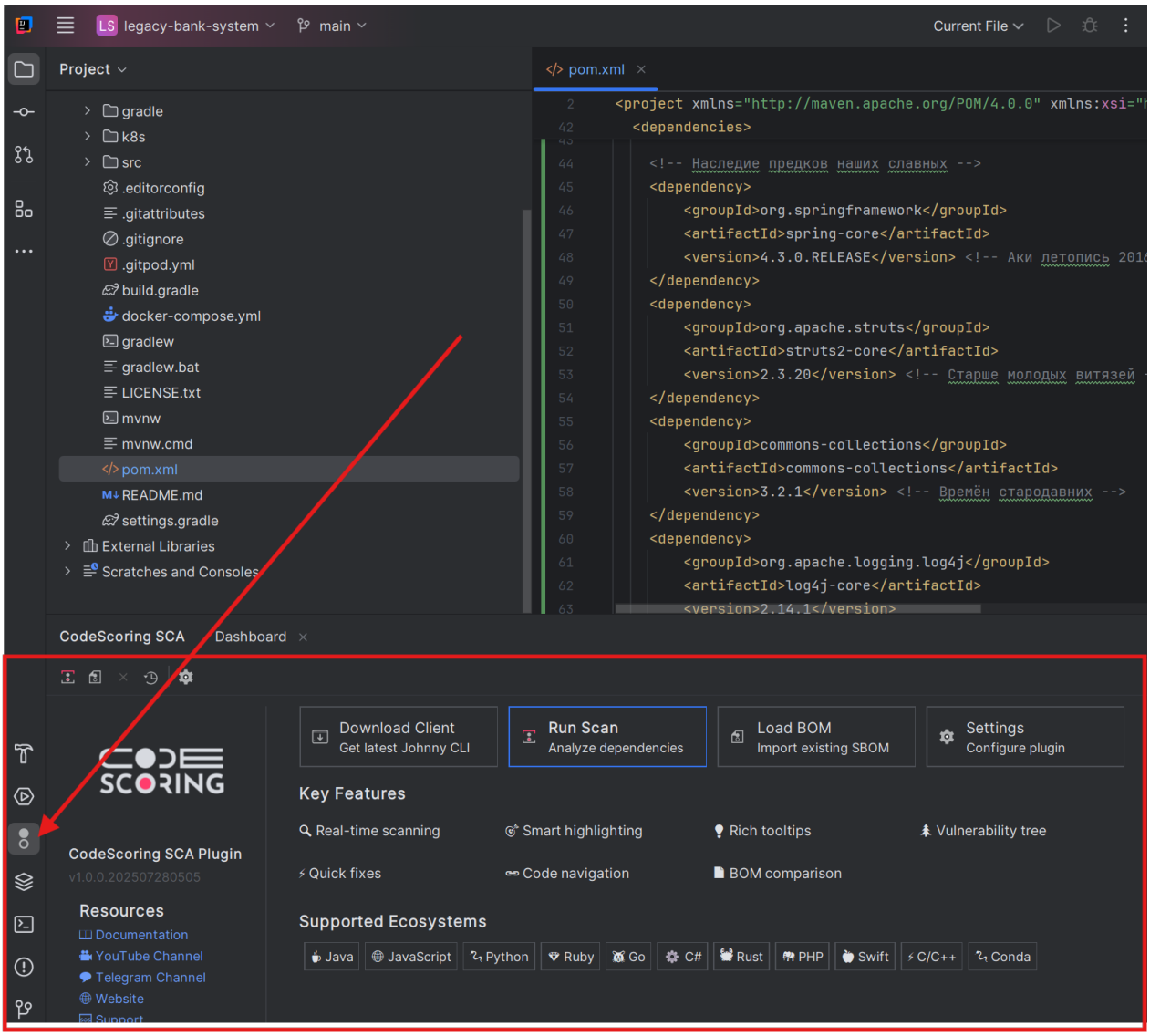
10. Перезапустите IntelliJ based IDE при появлении запроса

#Шаг 3: Найдите плагин CodeScoring

После установки и перезапуска вы должны увидеть окно инструментов CodeScoring.

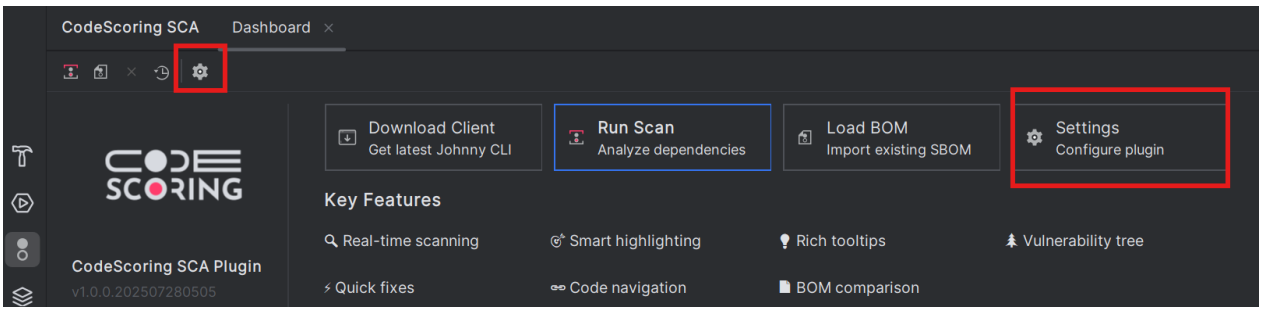


1. Найдите вкладку окна инструментов **"CodeScoring SCA"** (обычно внизу или слева в IDE)
2. Если не видно, перейдите в **View → Tool Windows → CodeScoring SCA**
3. Это откроет окно инструментов **CodeScoring SCA** с панелью Dashboard



#Шаг 4: Настройте плагин

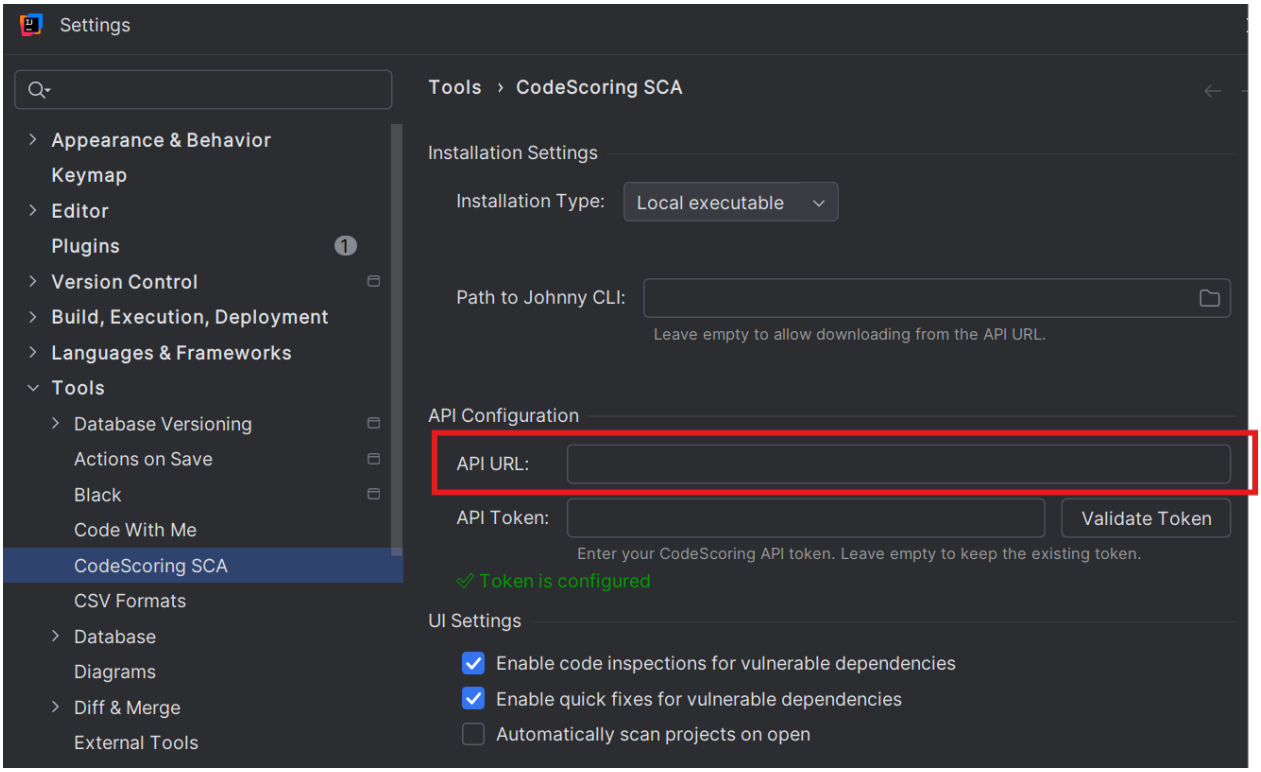
1. В окне инструментов CodeScoring SCA нажмите кнопку **"Settings"** (значок шестеренки) на панели инструментов
2. Откроется страница настроек CodeScoring SCA



#4.1 Проверьте URL API

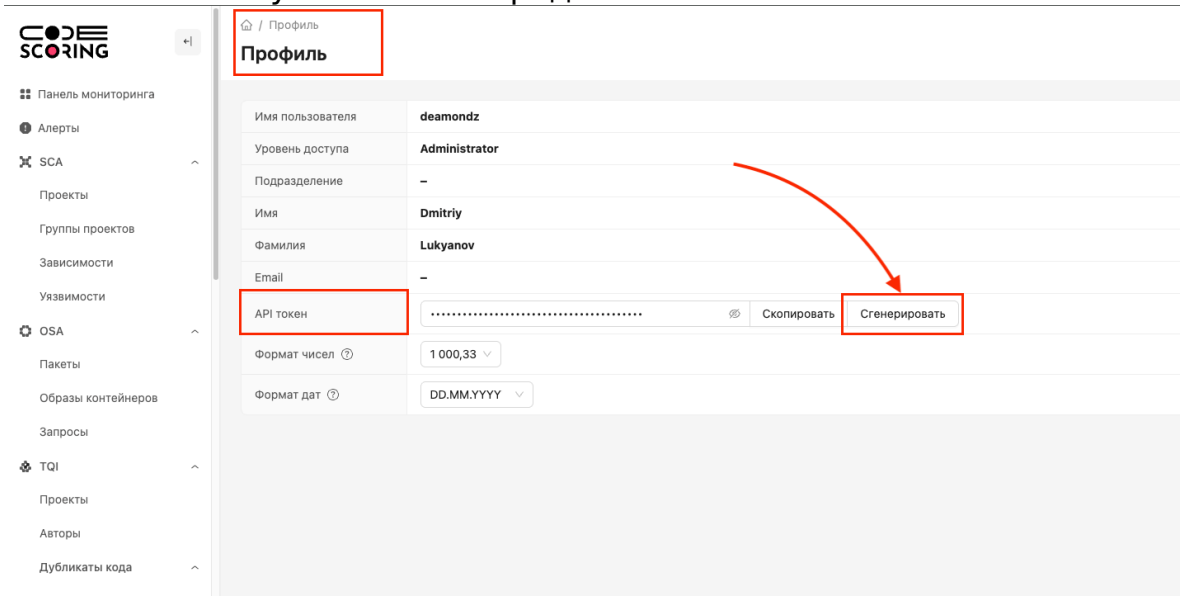
1. В настройках найдите поле **API URL**

2. Вам необходимо использовать URL CodeScoring, установленный в Вашей организации. При необходимости, обратитесь к администратору.



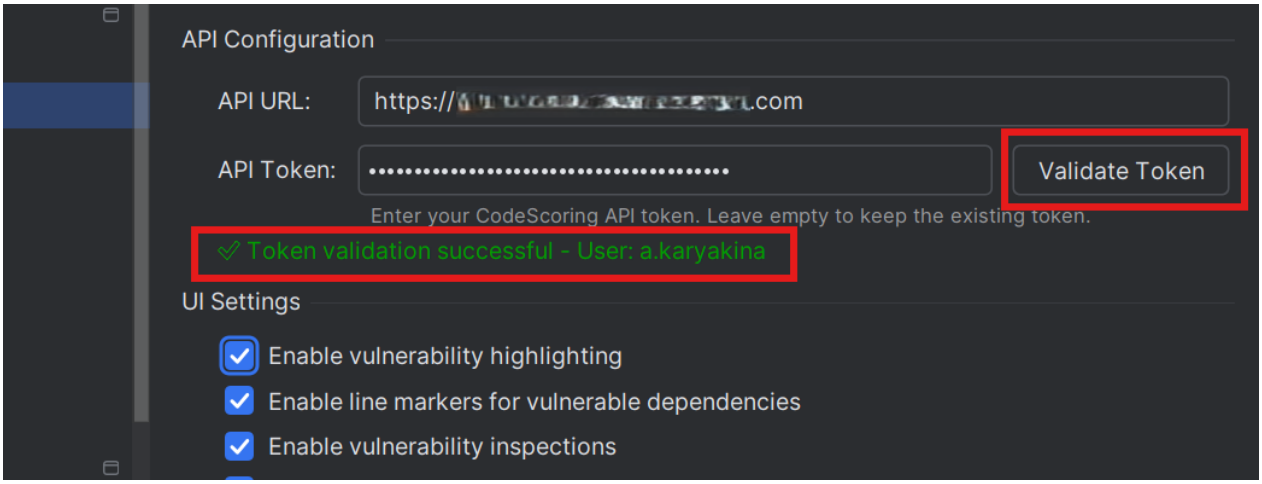
#4.2 Сгенерируйте и установите токен API

1. Откройте веб-браузер и перейдите по адресу: `<API URL>/cabinet/profile`
2. Войдите в свою учетную запись CodeScoring
3. Убедитесь, что вы находитесь на странице своего профиля
4. Найдите поле **"API token"**
5. Нажмите кнопку **"Generate"** рядом с полем токена API



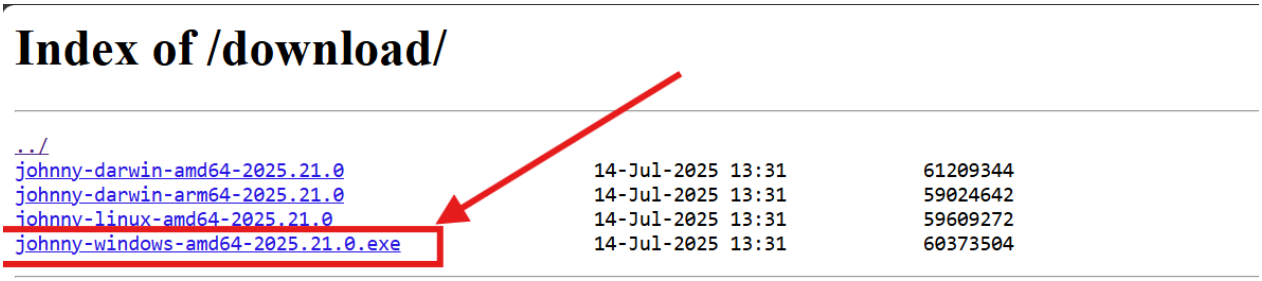
6. Скопируйте значение сгенерированного токена API
7. Вернитесь к настройкам IntelliJ-based IDE

8. Вставьте токен в поле **"API Token"**
9. Нажмите **"Validate Token"** для проверки работы токена
10. Плагин должен отобразить подтверждение того, что токен действителен



#4.3 Загрузите Johnny CLI (Опционально)

1. Перейдите на страницу релизов: `<API URL>/download/` (обратите внимание на завершающую косую черту)
2. Загрузите последний релиз исполняемого файла в зависимости от вашей операционной системы



#4.4 Настройте Johnny CLI

Существует три способа получить Johnny CLI для анализа ваших зависимостей с помощью нашего сервиса.

4.4.1 Локальная установка

Предварительные требования:

- Johnny CLI должен быть загружен и файл сделан исполняемым в системе

- Операционная система должна разрешать запуск исполняемого файла (для проверки запустите один раз файл в консоли с параметром --help вручную)

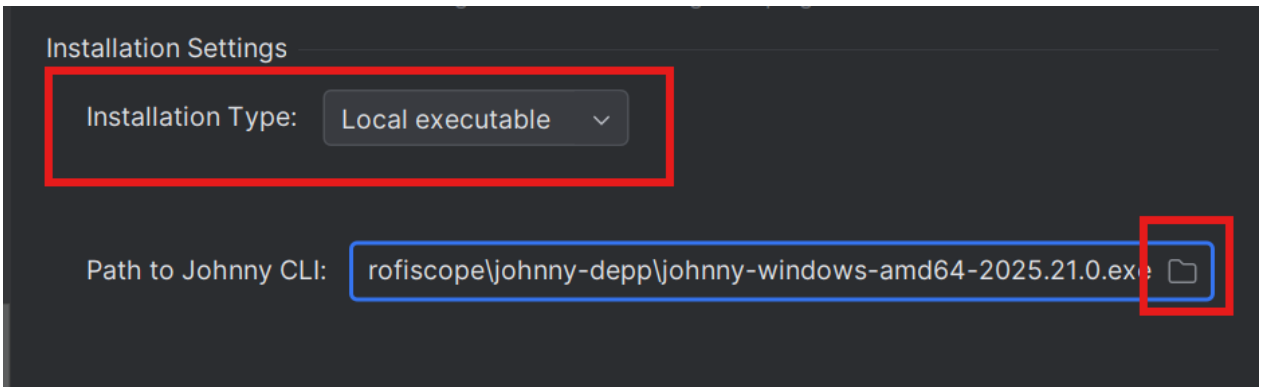
Шаги настройки:

1. Задайте тип установки Local

- В настройках выберите **"Local executable"** в выпадающем списке **Installation Type**

1. В поле **Johnny CLI Path** нажмите кнопку выбора папки
2. Перейдите к месту, куда вы ранее загрузили Johnny CLI
3. Выберите исполняемый файл Johnny CLI
4. Нажмите **"OK"** для сохранения настроек

Примечание: Если плагин спросит об изменении прав доступа к файлу (chmod +x), нажмите **"Да"**, чтобы разрешить плагину сделать файл исполняемым.



4.4.2 Автоматическая загрузка клиента

Предварительные требования:

- Должен быть настроен API URL
- Должен быть настроен токен API и валидация должна пройти успешно

Шаги настройки:

1. Установите тип установки на Local

- В настройках выберите **"Local executable"** в выпадающем списке **Installation Type**

1. Оставьте поле **Johnny CLI Path** пустым
2. Теперь при первом запросе сканирования Johnny CLI будет загружен с API URL. Это позволит Вам автоматически получить обновление клиента, как только оно будет доступно. Загруженный клиент будет сохранен в следующем месте:
    - **Linux/MacOS**: `~/.codescoring/johnny`
    - **Windows**: `%USERPROFILE%\.codescoring\johnny.exe`

4.4.3 Использование Docker

Установка Docker позволяет запускать Johnny CLI в изолированном контейнере, что полезно, когда вы не хотите устанавливать его непосредственно в вашей системе.

Предварительные требования:

- Docker должен быть установлен и запущен в вашей системе
- Ваш пользователь должен иметь права на выполнение команд Docker

Шаги настройки:

1. Установите тип установки на Docker

   - В настройках выберите **"Docker"** в выпадающем списке **Installation Type**

1. Настройте Docker образ

   - **Docker Image**: `johnny-depp:2025.29.0` (по умолчанию)
   - **Docker Registry**: `<адрес-реестра-кодскоринг>`
   - Пример полного пути к образу: `sample-codescoring-registry.com/johnny-depp:2025.29.0`

1. **Опционально: Дополнительные опции Docker** Добавьте пользовательские опции запуска Docker при необходимости в поле **Additional Docker Options**:

   ```
   --memory=2g --cpus=2
   ```

Как это работает:

- Клиент автоматически монтирует каталог вашего проекта в контейнер
- Сканирование выполняется внутри контейнера, а результаты сохраняются в вашем проекте
- Ручные команды Docker не требуются - плагин обрабатывает все автоматически

Устранение неполадок с установкой Docker:

- **"Docker not found"**: Убедитесь, что Docker установлен и команда `docker` находится в вашем PATH
- **Permission denied**: Добавьте вашего пользователя в группу docker: `sudo usermod -aG docker $USER`
- **Image pull failed**: Проверьте учетные данные реестра и сетевое подключение
- **Container exits immediately**: Проверьте Event Log для получения подробных сообщений об ошибках

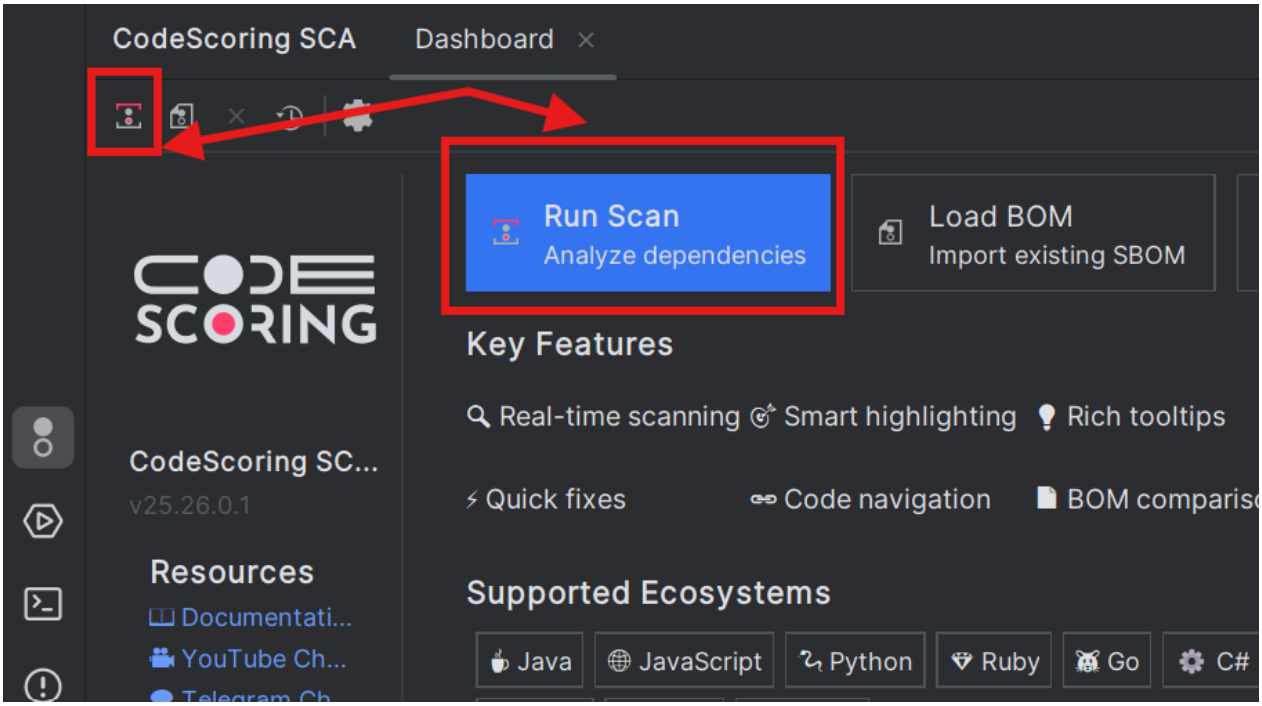
#Шаг 5: Запустите первое сканирование

Теперь, когда плагин настроен, вы можете запустить первое сканирование зависимостей:

#Метод 1: Использование dashboard

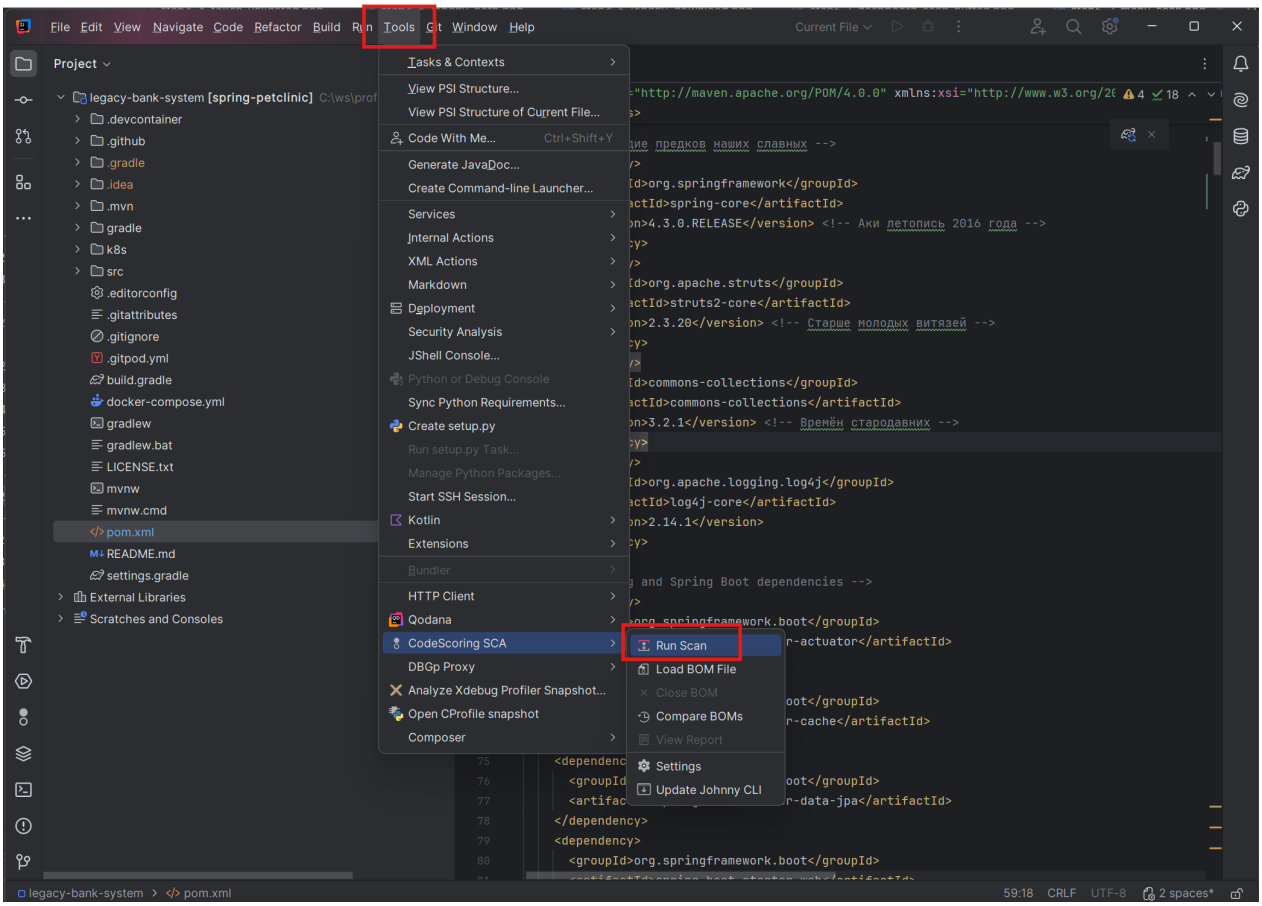
1. Откройте проект в IntelliJ-based IDE
2. Откройте окно инструментов CodeScoring SCA
3. В панели Dashboard нажмите кнопку **"Run Scan"**

!



#Метод 2: Использование главного меню

1. Перейдите в **Tools → CodeScoring SCA → Run Scan**



#Метод 3: Использование панели инструментов

1. Найдите вкладку плагина **CodeScoring SCA** и ее главную панель инструментов
2. Нажмите кнопку **"Run Scan"** (см скриншот из метода 1)

#Шаг 6: Тонкая настройка конфигурации сканирования

После завершения сканирования вы увидите новый каталог `.codescoring`, содержащий:

1. **config.yaml** - файл конфигурации для Johnny CLI, вы можете прочитать о нем в этом разделе. Вы можете менять этот файл, так как он никогда не будет перезаписан.
2. **donotfix.yaml** - файл конфигурации со списком шаблонов имен файлов, которые должны быть исключены из действий Quick Fix. Вы можете изменять этот файл.
3. **report.html** - отчет, сгенерированный в формате html, содержащий вывод Johnny CLI в формате цветной таблицы. Перезаписывается во время каждого сканирования.
4. **bom.json** - файл результатов сканирования, созданный Johnny CLI в формате cyclone-dx 1.6, он будет загружен автоматически и показан в панели уязвимостей для любого открытого в IntelliJ-based IDE проекта, в котором существует .codescoring/bom.json
5. **bom.json.N** - где N - это ревизия сканирования, т.е. 0 - предыдущее сканирование, а 5 (например) - самое первое сканирование, и

bom.json.0 будет использоваться для сравнения с bom.json (и показан в дереве DIFF), если он существует на момент открытия проекта

› your-project
› .codescoring
config.yaml# Конфигурация сканирования
donotfix.yaml# Конфигурация QuickFix
report.html# Последний отчет сканирования
bom.json# Текущие уязвимости
bom.json.0# Предыдущее сканирование (сравнение)
bom.json.1# Более старые сканирования...
pom.xml# Ваши зависимости
...ваш код ...

#Пример использования конфигурации сканирования CodeScoring

Отредактируйте `.codescoring/config.yaml` для настройки:

```
scan:
  general:
    ignore: # Директории для пропуска
      - target
      - build
      - .idea
    with-hashes: true # Включить хеши файлов для точного сопоставления
    only-hashes: false # Использовать только обнаружение на основе хешей
  dir:
    no-recursion: false # Предотвращает рекурсивное сканирование корневой директории
```

#Настройка исключений Quick Fix

Файл `.codescoring/donotfix.yml` контролирует, какие файлы не должны быть изменены действиями Quick Fix. Это особенно полезно для сгенерированных файлов (таких как lock-файлы), которые должны быть перегенерированы, а не исправлены вручную.

Пример использования конфигурации исключений QuickFix:

```
## Lock-файлы и сгенерированные файлы, которые не должны быть
изменены напрямую
patterns:
  - go.sum
  - package-lock.json
  - yarn.lock
  - Cargo.lock
  - composer.lock
  - "*.generated.*"
  - "**/generated/**"
```

Синтаксис шаблонов:

- Точное имя файла: `go.sum`
- Шаблоны с подстановочными знаками: `*.lock`, `*-lock.json`
- Шаблоны директорий: `**/node_modules/**`
- Несколько расширений: `*.{lock,generated}`

Файл создается автоматически при:

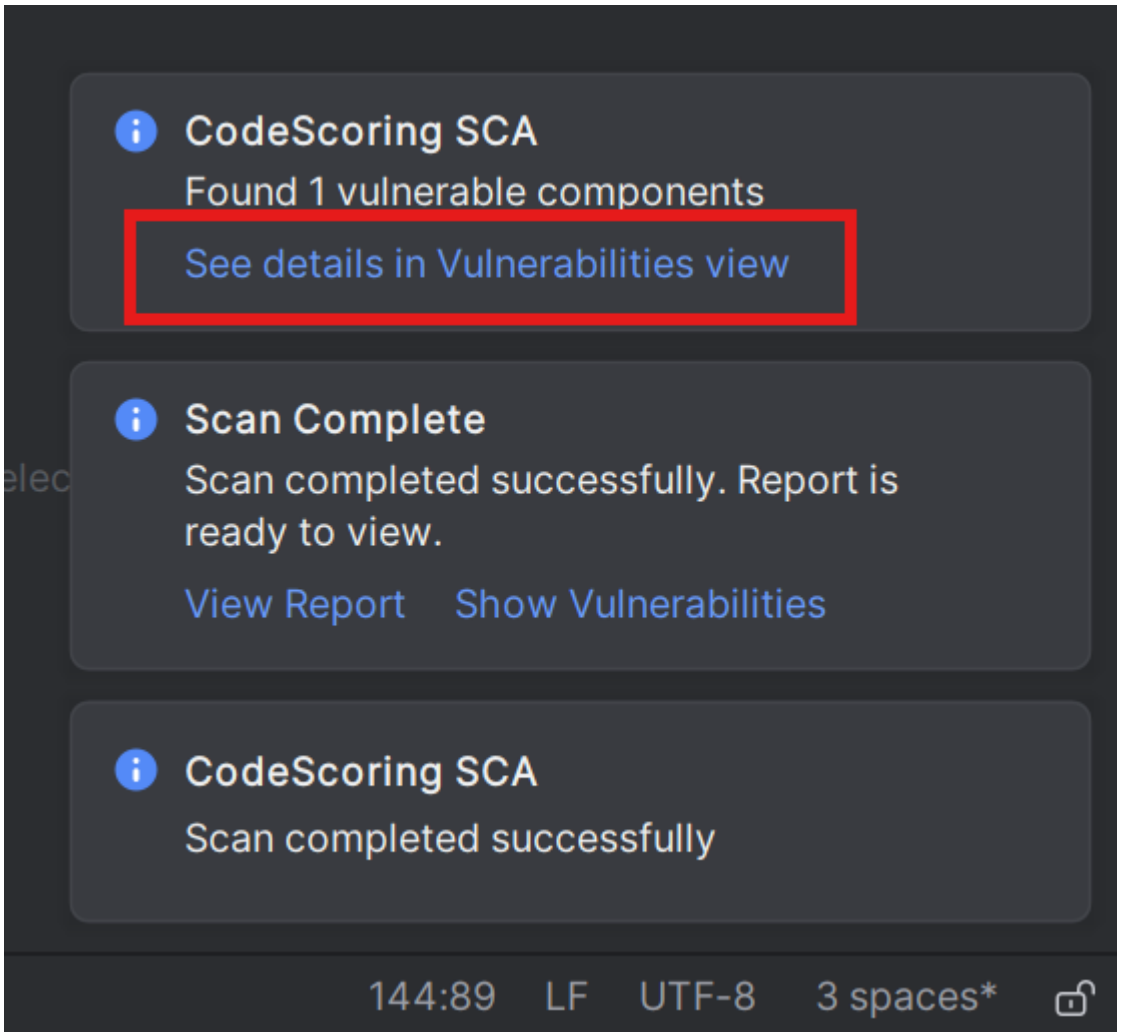
- Первом сканировании в проекте
- Открытии проекта с существующей директорией `.codescoring`, но без `donotfix.yml`

Примечание: Файлы, соответствующие этим шаблонам, все равно будут сканироваться на наличие уязвимостей и показаны в результатах, но действия Quick Fix (как индивидуальные, так и массовые) будут их пропускать. Пользователи должны перегенерировать эти файлы, используя команды их менеджера пакетов.

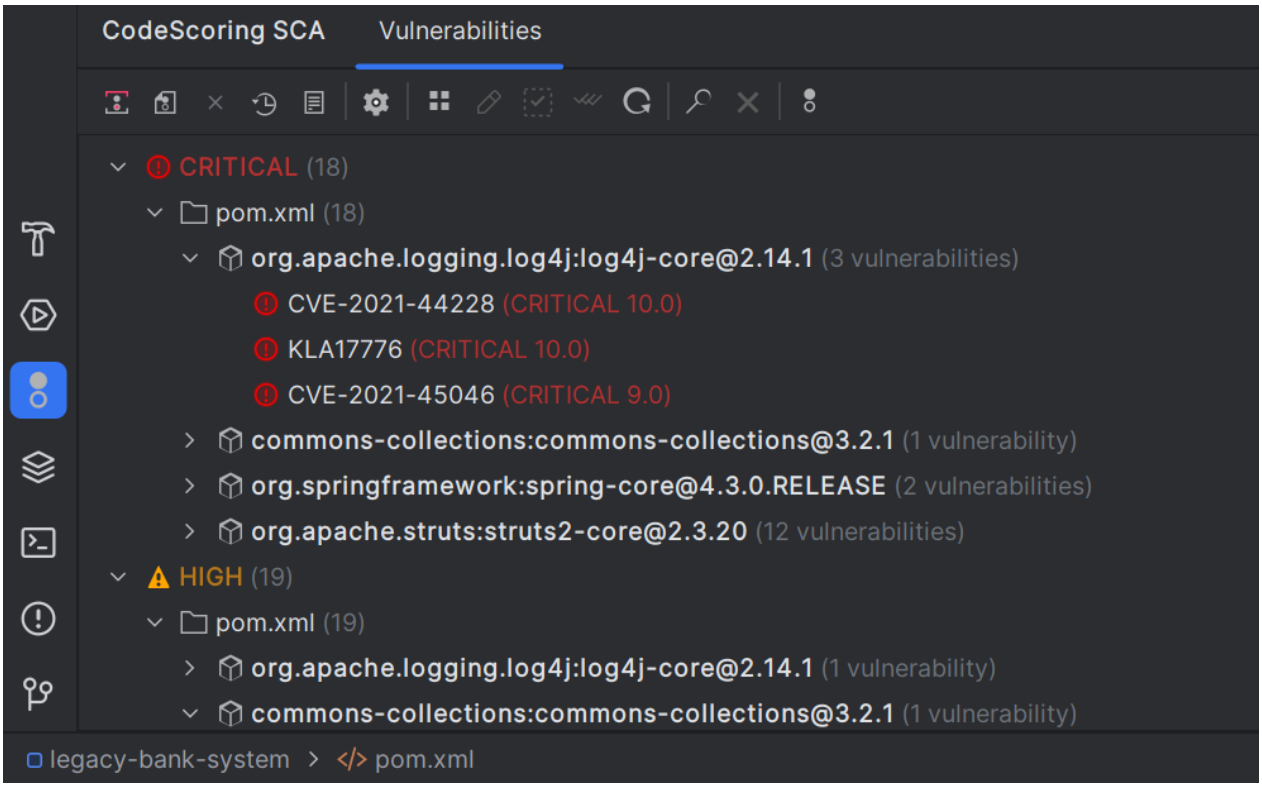
#Шаг 7: Просмотр результатов сканирования

После завершения сканирования:

1. **Проверьте уведомления**: Посмотрите на уведомления в правом нижнем углу IntelliJ-based IDE, нажав на **"View Report"** и **"See details in Vulnerabilities view""**

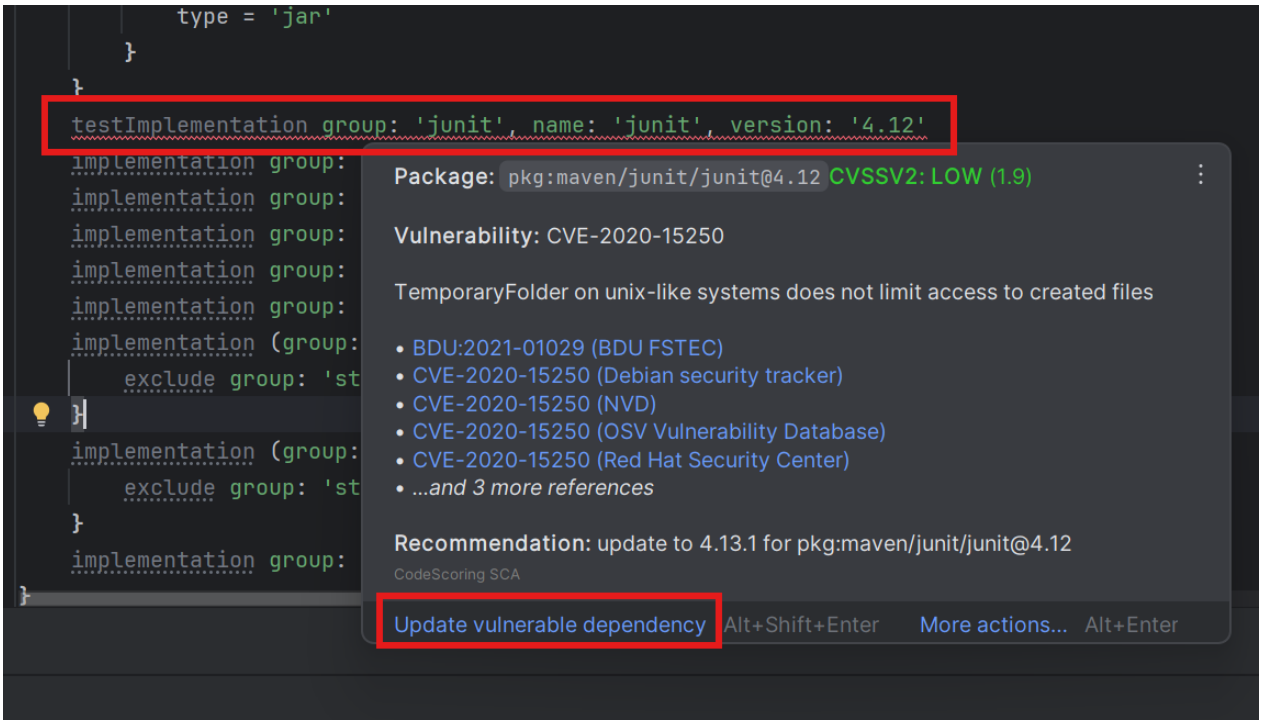


2. **Откройте дерево уязвимостей**: Окно плагина автоматически переключится на панель **Vulnerabilities**
3. **Просмотрите уязвимости**: Панель уязвимостей покажет все обнаруженные проблемы безопасности в Ваших зависимостях
4. **Изучите детали**: Вы можете нажимать на отдельные уязвимости, чтобы увидеть подробную информацию в панели деталей
5. **Примените исправления**: Используйте кнопки **"Fix All"** или **"Fix Selected"** или отдельные быстрые исправления для обновления уязвимых зависимостей
6. **Просмотр алертов**: Вы можете нажать на панель **Alerts** для просмотра алертов по сработавшим политикам



#7.1 Подсветка уязвимостей

- **Подсветка в коде**: Уязвимые зависимости подсвечиваются прямо в файлах кода (`build.gradle`, `pom.xml`, `package.json` и т.д.)
- **Цвета критичности**:
  - Критический (красный)
  - Высокий (оранжевый)
  - Средний (желтый)
  - Низкий (синий)
- **Поддержка нескольких файлов**: Работает со всеми поддерживаемыми типами файлов
- **Наведите курсор** на подсвеченные зависимости, чтобы увидеть детали уязвимостей



#7.2 Информация при наведении

При наведении курсора на подсвеченные зависимости отображается:

- **Идентификатор уязвимости**: Номер CVE со ссылкой
- **Критичность**: Оценка CVSSv3 и уровень
- **Описание**: Что делает уязвимость
- **Ссылки на источники**: Информация об официальной регистрации уязвимости
- **Рекомендации**: Предлагаемые версии для обновления
- **Быстрое исправление**: Опция обновления в один клик

#7.3 Панель уязвимостей

7.3.1 Структура дерева

```
 Уязвимости (247)
├── 🔴 Критические (12)
│   ├── CVE-2023-1234 - Удаленное выполнение кода
│   │   ├── lodash@4.17.20
│   │   └── pom.xml:15
│   └── ...
├── 🟠 Высокие (45)
├── 🟡 Средние (89)
└── 🟢 Низкие (101)
```

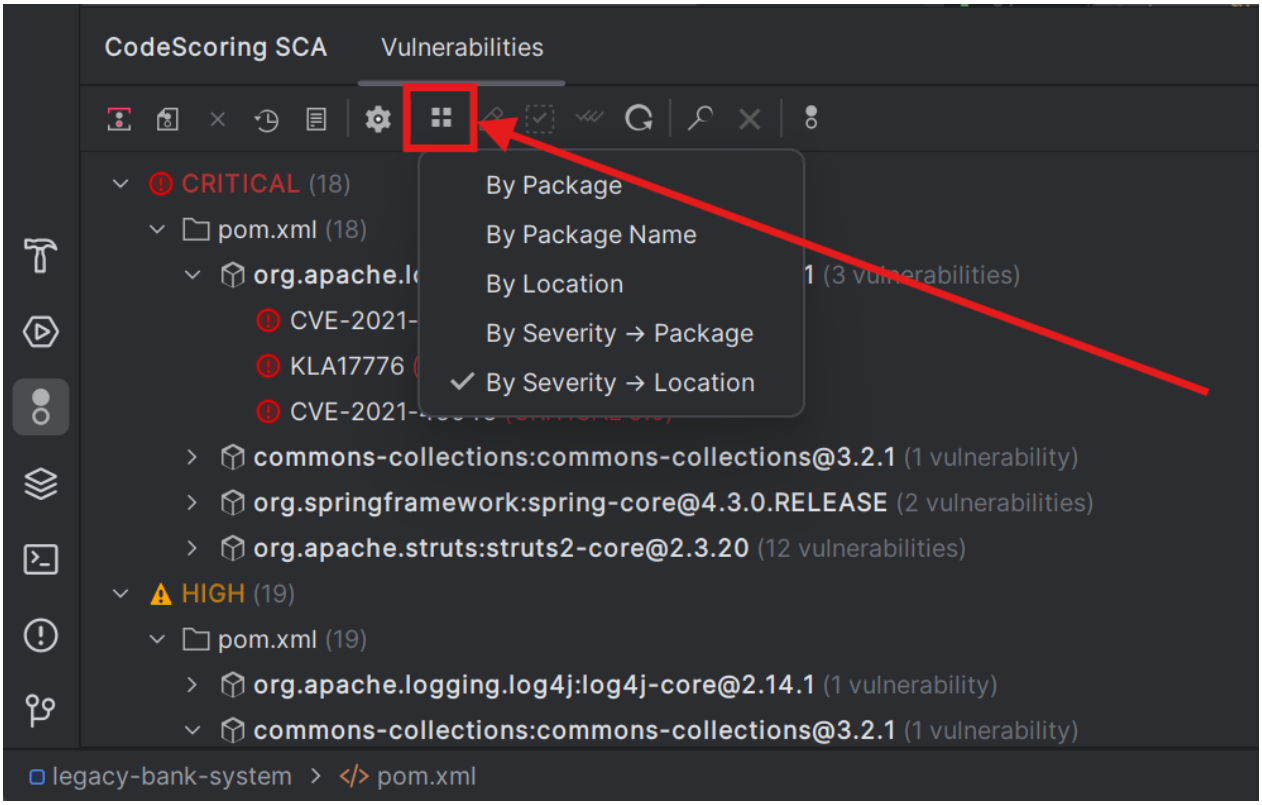
7.3.2 Опции группировки

- Используйте панель уязвимостей для фильтрации по критичности, пакету или другим критериям

- Группируйте уязвимости по различным категориям для лучшей организации

Изменение группировки через кнопку панели инструментов или команду:

- **По критичности → Местоположению → Компоненту** (по умолчанию): По уровню критичности, подгруппировка по файлу и затем по пакету
- **По критичности → Компоненту**: По уровню критичности, подгруппировка по имени пакета в алфавитном порядке
- **По местоположению → Компоненту**: Группировка по пути к файлу
- **По имени компонента → Компоненту**: Группировка разных версий одного и того же пакета вместе в алфавитном порядке
- **По компоненту**: По компоненту (с версией, если известна)



#7.4 Поиск и фильтрация

Возможности поиска

- **Несколько полей**: Поиск по:
  - Имени пакета (например, "lodash")
  - Идентификатору CVE (например, "CVE-2023")
  - Пути к файлу (например, "frontend/")
  - Уровню критичности
- **Нечеткое сопоставление**: Находит частичные совпадения
- **Точное совпадение**: Ищет точное совпадение слова в кавычках
- **Без учета регистра**: Не требуется точный регистр

Панель инструментов поиска Для поиска и фильтрации в плагине имеются:

- **Поле поиска**: Для ввода поисковых запросов
- **Кнопка очистки**: Сброс поиска
- **Индикатор результатов**: Показывает количество найденных элементов первым элементом дерева уязвимых компонентов

#7.5 Быстрые исправления

Быстрые исправления в коде

- **Лампочка IntelliJ**: Нажмите на лампочку рядом с уязвимой зависимостью

или

- **Alt+Enter** (⌥ + **Enter** на MacOS): Используйте горячую клавишу, когда курсор находится на уязвимой зависимости

или

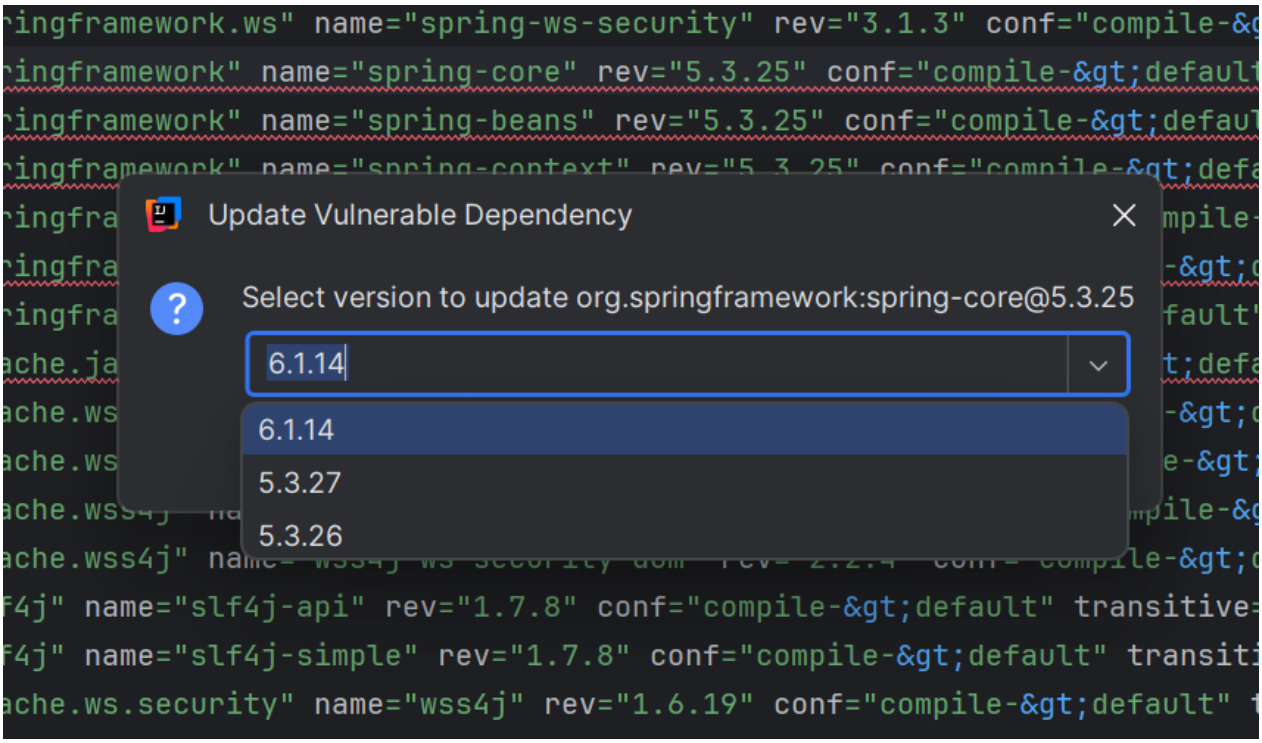
- **Update vulnerable dependency** внизу на всплывающей карточке уязвимого компонента

или

- кнопка **Fix Selected** на панели инструментов,

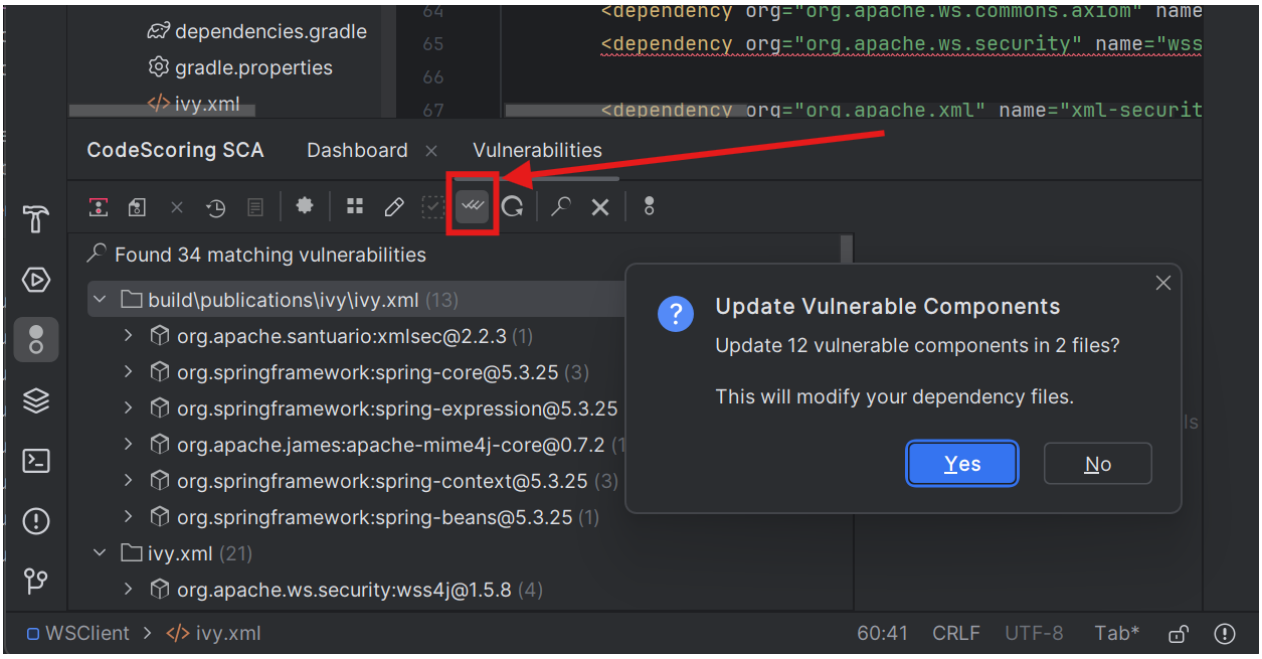
а затем

- **Выберите версию**: Если доступно несколько безопасных версий, выберите подходящую



Массовые исправления (не работает при группировках по критичности)

- **Кнопка Fix All**: Обновляет все уязвимые компоненты с доступными исправлениями
- **Интеллектуальное обновление**: Автоматически выбирает наиболее подходящую безопасную версию
- **Отчет об изменениях**: Показывает, сколько зависимостей были обновлены
- **Исключения**: Учитывает шаблоны, определенные в `.codescoring/donotfix.yml`



#7.6 Работа с файлами BOM

Автозагрузка Плагин автоматически загружает файлы BOM из:

1. `.codescoring/bom.json` (основной)
2. `bom.json` (корневой каталог проекта)

Ручные операции

- **Загрузить BOM**: **Tools → CodeScoring SCA → Load BOM File**
- **Закрыть BOM**: **Tools → CodeScoring SCA → Close BOM**

#7.7 Сравнение BOM

Сравнивается полный состав всех компонентов, а не только уязвимых. С помощью сравнения можно увидеть различия в полном перечне используемых компонентов и отследить изменения версий. При этом старые версии компонентов будут показаны как удаленные, а новые - как добавленные. Чтобы удобнее было отслеживать изменения в версиях компонентов, можно сгруппировать их по имени пакета (см ниже).

Автоматическое сравнение При открытии проекта:

- Загружает текущий BOM (`bom.json`)
- Сравнивает с предыдущим (`bom.json.0`)
- Показывает уведомление об изменениях

Ручное сравнение

1. Выберите **Tools → CodeScoring SCA → Compare BOMs**
2. Выберите базовый файл BOM
3. Если BOM уже загружен, он будет использован как целевой
4. Если BOM не загружен, выберите целевой файл
5. Отобразится результат сравнения в панели DIFF

Представления сравнения

```
 BOM DIFF (Изменения: 23 добавлено, 15 удалено, 45 обновлено,
73 без изменений)
├── + Добавлено (23)
│   ├── [ADDED] react@18.1.3 0 уязвимостей
│   └── ...
├── − Удалено (15)
├──  Обновлено (45)
│   ├── [UPDATED] lodash: 4.17.20
│   └── ...
└── ✓ Без изменений (73)
```

Опции группировки сравнения

- **По типу изменения**: Добавлено/Удалено/Обновлено/Без изменений (по умолчанию)
- **По пакету**: Алфавитная группировка пакетов, наиболее полезный вид группировки для отслеживания изменившихся версий пакетов
- **По расположению**: Группировка по пути к файлу
- **По критичности**: Группировка по влиянию уязвимости

Фильтрация сравнения

Используйте поле поиска для фильтрации результатов сравнения по:

- Имени пакета
- Типу изменения
- Пути к файлу

#7.8 Отчеты

Отчеты сканирования

- **Автоматически генерируются**: Создаются после каждого сканирования
- **Расположение**: `.codescoring/report.html`
- **Формат**: Цветной HTML с подробной информацией
- **Содержимое**:
  - Статус сканирования
  - Выполненная команда
  - Сводка результатов
  - Найденные уязвимости
  - Предупреждения от политик
  - Сообщения об ошибках

Просмотр отчетов

- **Команда**: **Tools → CodeScoring SCA → View Report**
- **Открывается в**: на выбор, внешний браузер, внутренний предпросмотр, редактор кода

#7.9 Панель алертов

В панели **Alerts** представлена информация по алертам для сработавших политик по итогам анализа. Для каждого алерта показывается:

- Название политики
- Уровень алерта
- Статус блокировки
- Список пакетов с критериями

#Шаг 8: Настройки и кастомизация

#Список доступных настроек

Настройка	Описание	По умолчанию
API Configuration		
`API URL`	URL вашей установки CodeScoring	
`API Token`	API токен. Безопасно сохраняется	*(устанавливается через UI)*
platform Settings		
`platform Type`	Local executable или Docker	`Local executable`
`Path to Johnny CLI`	Путь к Johnny CLI (пустой для автозагрузки)	*(автозагрузка)*
`Docker Image`	Имя Docker образа	`johnny-depp:2025.29.0`
`Docker Registry`	Реестр Docker	*(предоставляется заботой)*
`Additional Docker Options`	Дополнительные опции Docker	
UI Settings		
`Enable vulnerability inspections`	Включить инспекции кода	`true`
	Разрешить быстрые исправления	`true`

Настройка	Описание	По умолчанию
`Enable quick fixes for vulnerable dependencies`		
`Automatically scan projects on open`	Запускать сканирование при открытии проекта	`true`
Severity Colors		
`Critical Color`	Цвет для критических уязвимостей	*(красный)*
`High Color`	Цвет для высоких уязвимостей	*(оранжевый)*
`Medium Color`	Цвет для средних уязвимостей	*(желтый)*
`Low Color`	Цвет для низких уязвимостей	*(синий)*
`Unknown Color`	Цвет для неизвестной критичности	*(серый)*

#Устранение неполадок

#Логи плагина

- Для подробного понимания работы плагина проверьте логи IDE:
  - **Help → Show Log in Explorer/Finder**
  - Ищите записи с "CodeScoring" в `idea.log`

#Распространенные проблемы

Проблемы установки

Проблема	Решение
Плагин не виден после установки	Полностью перезапустите IntelliJ-based IDE, проверьте **Settings → Plugins**
Ошибка совместимости	Убедитесь, что версия IDE 2024.1 или новее, убедитесь в отсутствии проблем в лог файле
Установка зависает	Проверьте подключение к интернету, попробуйте установить заново

Проблемы конфигурации

Проблема	Решение
Поля настройки не отображаются	Возможно, несовместимая IDE, проверьте версию и тип IDE, загляните в лог-файлы IDE
Валидация токена не удается	Проверьте URL API, сгенерируйте новый токен, проверьте прокси/VPN
Johnny CLI не найден	Проверьте путь, права доступа, антивирус
Docker не работает	Убедитесь, что Docker запущен, проверьте права пользователя

Проблемы сканирования

Проблема	Решение
Сканирование зависает	Проверьте Event Log, попробуйте запустить Johnny CLI вручную.
Нет результатов	Убедитесь, что проект содержит файлы зависимостей
Частичные результаты	Проверьте конфигурацию в .codescoring/config.yaml
Ошибки токена	Проверьте срок действия токена, права доступа

Проблемы отображения

Проблема	Решение
Нет подсветки	Включите в настройках, перезагрузите файлы
Неправильные цвета	Проверьте настройки цветов критичности
Отсутствует панель	**View → Tool Windows → CodeScoring SCA**
Медленная работа	Уменьшите размер пагинации в настройках

Проблемы исправления

Проблема	Решение
Исправление не удается	Проверьте права на запись файлов
Исправление игнорируется	Проверьте логи плагина и .codescoring/donotfix.yml
Неправильная версия	Вручную укажите версию в файле
Конфликты версий	Исправляйте по одному компоненту
Откат изменений	Используйте систему контроля версий

#Получение помощи

1. **Проверьте Event Log**: **View → Tool Windows → Event Log**
2. **Включите debug логи**:
   - **Help → Diagnostic Tools → Debug Log Settings**
   - Добавьте `com.codescoring.intellij`
3. **Отчеты об ошибках**: Просмотрите `.codescoring/report.html` для деталей сканирования

Свяжитесь с отделом заботы: support@codescoring.ru

#Безопасность и конфиденциальность, обработка данных

- **Локальное сканирование**: Код не отправляется на серверы
- **Связь с API**: Передаются только метаданные (конфигурационные файлы Вашего пакетного менеджера)
- **Хранение токенов**: Безопасное хранилище учетных данных VS Code

#Рекомендации

#Рекомендации по рабочему процессу

1. **Начальная настройка**: Полное сканирование при запуске проекта
2. **Обзоры**: Сравнивайте BOM между версиями
3. **CI/CD**:
   - **Перед коммитом**: Выполните полное сканирование
   - **Поделитесь конфигурацией**: Закоммитьте `.codescoring/config.yaml` и `.codescoring/donotfix.yaml`
   - **Игнорируйте временные файлы**: Добавьте в `.gitignore`:

```
.codescoring/report.html
.codescoring/bom.json.*
```

```
- **Отслеживайте основной BOM**: Версионируйте `.codescoring/bom.json`
- **Стандартизируйте**: Стандартизируйте настройки плагина
```

#Интеграция с процессами разработки

1. **Code Review**:
    - Проверяйте изменения зависимостей
    - Требуйте исправления критических уязвимостей
    - Документируйте принятые риски
2. **Release Management**:
    - Генерируйте отчеты для каждого релиза
    - Отслеживайте улучшения безопасности
    - Планируйте обновления зависимостей
3. **Compliance**:
    - Экспортируйте BOM для аудита
    - Отслеживайте лицензии компонентов
    - Поддерживайте историю сканирований

#Работа с Lock-файлами

1. **Понимание Lock-файлов**:
    - Lock-файлы генерируются менеджерами пакетов
    - Они не должны редактироваться вручную
    - Изменения должны вноситься в файлы манифестов
2. **Поведение Quick Fix**:
    - Файлы, соответствующие шаблонам `donotfix.yml`, пропускаются
    - Обновите файлы манифестов, затем перегенерируйте lock-файлы
    - Используйте соответствующие команды менеджера пакетов, например:
        - **Go**: `go mod tidy`
        - **NPM**: `npm install`
        - **Yarn**: `yarn install`
        - **Cargo**: `cargo update`
3. **Рекомендации**:
    - Просмотрите `donotfix.yaml` и настройте шаблоны по необходимости
    - Документируйте ваш процесс перегенерации
    - Автоматизируйте обновления lock-файлов в CI/CD

#Работа с зависимостями

В отдельных случаях при работе с зависимостями требуются дополнительные действия для улучшения точности композиционного анализа.

В этом разделе собраны инструкции и лучшие практики по работе с зависимостями в разных экосистемах, а также описание работы CodeScoring в нестандартных сценариях обработки зависимостей.

#Работа с зависимостями в Java

#Apache Maven:

#Создание файла `maven-dependency-tree.txt`

```
mvn dependency:tree -DoutputFile=maven-dependency-tree.txt
```

#Gradle:

#Создание файла `gradle-dependency-tree.txt`

```
./gradlew dependencies > gradle-dependency-tree.txt
```

#Создание файла `gradle-dependency-tree.txt` для мульти-проектных сборок

Для анализа зависимостей в Gradle-проектах Johnny использует файл `gradle-dependency-tree.txt`. В обычных проектах он формируется автоматически. Однако в мульти-проектных сборках его корректное построение возможно только при наличии в проекте специальной задачи с ожидаемым именем.

Для получения всех зависимостей в таком случае необходимо произвести следующие действия:

#Связывание манифестов

- Если в директории находится build.gradle и gradle.lockfile без совпадения по имени они будут состыкованы;
- При наличии в одной директории всех трёх манифестов (build.gradle, gradle.lockfile, gradle-dependency-tree.txt) приоритет при связыванию отдаётся gradle-dependency-tree, gradle.lockfile в этом случае разбирается отдельно;
- При наличии в одной директории нескольких лок-файлов для одного build.gradle без совпадения по имени для связывания будет

использован любой из них. Остальные лок-файлы разбираются отдельно.

#Groovy

Добавить в файл `build.gradle` код:

```
subprojects {
    afterEvaluate { project ->
        project.tasks.register('CodeScoring_All_Dependencies',
DependencyReportTask)
    }
}

tasks.register('CodeScoring_All_Dependencies') {
    dependsOn subprojects.findAll {
        it.tasks.findByName('CodeScoring_All_Dependencies') !=
null
    }.collect {
        it.tasks.named('CodeScoring_All_Dependencies')
    }
}
```

#Kotlin

Добавить в файл `build.gradle.kts` код:

```
subprojects {
    afterEvaluate {

tasks.register<DependencyReportTask>("CodeScoring_All_Dependen
cies")
    }
}

tasks.register("CodeScoring_All_Dependencies") {
    dependsOn(subprojects.mapNotNull {
        it.tasks.findByName("CodeScoring_All_Dependencies")
    })
}
```

После этого выполнить команду:

```
./gradlew CodeScoring_All_Dependencies > gradle-dependency-
tree.txt
```

После создания артефактов необходимо применить команду консольного агента scan file для полученного результатов сканирования, например:

```
./johnny \
scan file ./gradle-dependency-tree.txt \
--api_token <api_token> \
--api_url <api_url>
```

#Работа с зависимостями в Scala

#sbt

#Создание файлов `scala-dependency-tree.txt` или `sbt-dependency-tree.txt`

1. **Настройка ширины графа зависимостей**

   Чтобы сгенерировать полный граф зависимостей добавьте следующую строку в файл `build.sbt`:

   ```
   ThisBuild / asciiGraphWidth := 999999999
   ```

   Альтернативно, можно установить значение `asciiGraphWidth` глобально.

2. **Генерация дерева зависимостей**

   Выполните следующую команду для генерации дерева зависимостей:

   ```
   sbt clean compile "dependencyTree::toFile target/tree.txt"
   ```

   Убедитесь, что файл сохранен с именем `scala-dependency-tree.txt` или `sbt-dependency-tree.txt`, так как только эти имена поддерживаются для корректного парсинга.

3. **Сканирование сгенерированного файла**

Опция консольного агента `--sbt-resolve` в команде сканирования в данном случае не нужна, поскольку выполняется сканирование уже сгенерированного дерева с полной структурой зависимостей.

#Работа с зависимостями в Go

#Go Modules

#Создание файла `go.sum`

1. Инициализируйте модуль:

```
go mod init <module_name>
```

2. Установите зависимости:

```
go get <package>
```

3. После установки зависимостей автоматически создаются и обновляются файлы `go.mod` и `go.sum`.

4. Закрепите версии зависимостей:

```
go mod tidy
```

#Работа с зависимостями в JavaScript

#NPM

#Создание файла `package-lock.json`

1. Инициализируйте проект:

```
npm init -y
```

2. Установите зависимости:

```
npm install
```

#Поддержка механизма NPM package alias

Механизм NPM package alias позволяет устанавливать пакеты под разными именами, что удобно для одновременного использования нескольких версий библиотеки, замены зависимости без изменения её имени в коде и работы с форками.

Вместо стандартного указания версии используется синтаксис, явно задающий, какой пакет и его версию установить под нужным именем. Это упрощает тестирование, обновления и совместимость зависимостей.

В `package.json` в секции dependencies может быть указана следующая запись:

```
"dependencies": {
  "@babel/legacy-core": "npm:@babel/core@=7.12.0"
}
```

Консольный агент Johnny корректно обрабатывает эту запись, распознавая, что **@babel/legacy-core** – это alias для **@babel/core** версии 7.12.0. В ходе анализа зависимостей учитывается оригинальный пакет, предотвращая ошибки, связанные с несуществующими именами.

#Поддержка механизма NPM overrides

Механизм NPM overrides позволяет изменить версии транзитивных зависимостей. Это полезно в случаях необходимости замены зависимости с известной уязвимостью, либо замены зависимости на форк.

В секции overrides файла `package.json` может быть указана следующая запись:

```
"overrides": {
  "foo": "1.0.0"
}
```

В `package-lock.json` будет единственная версия пакета `foo`:

```
"node_modules/foo": {
"version": "1.0.0",
```

Консольный агент Johnny корректно обрабатывает эту запись, распознавая пакет **foo** версии 1.0.0.

#Поддержка механизма NPM workspaces

Механизм NPM workspaces позволяет централизованно управлять несколькими пакетами.

В `package.json` в секции workspaces может быть указана следующая запись:

```
"workspaces": [ "packages/a", "packages/b" ]
```

В таком случае агент Johnny будет обрабатывать корневой `package.json` и все `package.json` всех пакетов из workspaces как единое целое.

#PNPM

#Создание файла `pnpm-lock.yaml`

1. Инициализируйте проект:

   ```
   pnpm init -y
   ```

2. Установите зависимости:

   ```
   pnpm install
   ```

#Поддержка механизма PNPM package alias

Механизм PNPM package alias позволяет устанавливать пакеты под разными именами, что удобно для одновременного использования нескольких версий библиотеки, замены зависимости без изменения её имени в коде и работы с форками.

Вместо стандартного указания версии используется синтаксис, явно задающий, какой пакет и его версию установить под нужным именем. Это упрощает тестирование, обновления и совместимость зависимостей.

В `package.json` в секции dependencies может быть указана следующая запись:

```
"dependencies": {
  "lodash-old": "npm:lodash@3.10.1"
}
```

Консольный агент Johnny корректно обрабатывает эту запись, распознавая, что **lodash-old** – это alias для **lodash** версии 3.10.1. В ходе анализа зависимостей учитывается оригинальный пакет, предотвращая ошибки, связанные с несуществующими именами.

#Поддержка механизма PNPM overrides

Механизм PNPM overrides позволяет изменить версии транзитивных зависимостей. Это полезно в случаях необходимости замены зависимости с известной уязвимостью, либо замены зависимости на форк.

В секции pnpm/overrides файла `package.json` может быть указана следующая запись:

```
"pnpm": {
  "overrides": {
```

```
    "example-package": "^1.3.0"
  }
}
```

В `pnpm-lock.yaml` версия пакета `example-package` будет не ниже 1.3.0:

```
example-package@1.3.0: {}
```

Консольный агент Johnny корректно обрабатывает эту запись, распознавая пакет **example-package** версии 1.3.0.

#Поддержка механизма PNPM workspaces

Механизм PNPM workspaces позволяет централизованно управлять несколькими пакетами.

В `pnpm-workspace.yaml` расположенном рядом с корневым `package.json` быть указана следующая запись:

```
packages:
- 'packages/*'
```

В таком случае агент Johnny будет обрабатывать корневой `package.json` и все `package.json` всех пакетов из workspaces как единое целое.

#Yarn

#Создание файла `yarn.lock`

1. Инициализируйте проект:

   ```
   yarn init -y
   ```

2. Установите зависимости:

   ```
   yarn install
   ```

#Поддержка механизма Yarn package alias

Механизм Yarn package alias позволяет устанавливать пакеты под разными именами, что удобно для одновременного использования нескольких версий библиотеки, замены зависимости без изменения её имени в коде и работы с форками.

Вместо стандартного указания версии используется синтаксис, явно задающий, какой пакет и его версию установить под нужным именем.

Это упрощает тестирование, обновления и совместимость зависимостей.

В `package.json` в секции dependencies может быть указана следующая запись:

```
"dependencies": {
  "lodash-old": "npm:lodash@3.10.1"
}
```

В `yarn.lock` формируется запись:

```
"lodash-old@npm:lodash@3.10.1":
```

Консольный агент Johnny корректно обрабатывает эту запись, распознавая, что **lodash-old** – это alias для **lodash** версии 3.10.1. В ходе анализа зависимостей учитывается оригинальный пакет, предотвращая ошибки, связанные с несуществующими именами.

#Поддержка механизма Yarn selective dependency resolution

Yarn поддерживает избирательное разрешение версий через поле `resolutions` в `package.json`, что позволяет задавать конкретные версии зависимостей без редактирования `yarn.lock`.

Этот механизм полезен, если вам нужно обновить подзависимость, которая не обновляется часто, исправить уязвимость в транзитивной зависимости или зафиксировать версию из-за проблемного обновления.

CodeScoring поддерживает обработку данного механизма в консольном агенте Johnny. Вот несколько сценариев его работы:

#Замена пакета

Для замены пакета через механизм resolutions, в `package.json` добавляется следующая запись. В данном примере пакет **parcel/watcher** заменяется на пакет **favware/skip-dependency**.

```
"resolutions": {
  "@parcel/watcher": "npm:@favware/skip-dependency@latest"
}
```

Соответствующая данному пакету запись в файле `yarn.lock` будет следующей:

```
dependencies:
  "@parcel/watcher": "npm:2.1.0"
```

При установке в сборке используется пакет **favware/skip-dependency** версии 1.2.2. Консольный агент корректно идентифицирует данный механизм и анализирует именно финальный пакет.

```
"@parcel/watcher@npm:@favware/skip-dependency@latest":
  version: 1.2.2
  resolution: "@favware/skip-dependency@npm:1.2.2"
```

#Фиксация версии транзитивной зависимости

Для фиксации версии через механизм resolutions в `package.json` добавляется следующая запись. В данном примере версия пакета **http-signature** фиксируется на **1.3.4**.

```
"resolutions": {
  "http-signature": "1.3.4"
}
```

Соответствующие данному пакету записи в файле `yarn.lock` будут следующими:

```
dependencies:
  http-signature "~1.2.0"
```

При установке в сборке будет использована версия **http-signature 1.3.4**. Консольный агент анализирует финальную версию пакета.

```
http-signature@1.3.4, http-signature@~1.2.0:
  version "1.3.4"
  resolved "https://registry.yarnpkg.com/http-signature/-/http-
signature-1.3.4.tgz#a65b41193110b222364e776fd1ac848655a0e2f0"
```

#Фиксация версии при множественных зависимостях

Для фиксации версии при наличии нескольких зависимостей через механизм resolutions в `package.json` добавляется следующая запись. В данном примере версия пакета **yaml** фиксируется на **2.2.2**.

```
"resolutions": {
  "yaml": "2.2.2"
}
```

Соответствующие данному пакету записи в файле `yarn.lock` будут следующими:

```
dependencies:
  yaml: ^1.10.0
```

```
  yaml: ^2.2.1
  yaml: ^1.7.2
  yaml: ^1.10.2
  yaml: ^2.3.4
  yaml: 2.3.1
  yaml: ^2.1.1
```

При установке в сборке будет использована версия **2.2.2**. Консольный агент анализирует только зафиксированную в `resolutions` версию пакета.

```
"yaml@npm:2.2.2":
  version: 2.2.2
  resolution: "yaml@npm:2.2.2"
```

#Bun

#Создание файла `bun.lock`

```
bun install
```

В случае, если в проекте уже используется бинарный формат `bun.lockb` для создания `bun.lock` нужно использовать следующую команду:

```
bun install --save-text-lockfile --frozen-lockfile --lockfile-only
```

#Работа с зависимостями в .NET

#NuGet

#Создание файла `packages.lock.json`

1. Включите поддержку lock-файла (для .NET 5 и выше):

   ```
   dotnet nuget locals all --clear
   ```

2. Установите зависимости:

   ```
   dotnet restore --use-lock-file
   ```

#Создание файла `paket.lock`

1. Создайте lock-file:

```
paket install
```

#Особенности работы с `sln` манифестом

При сканировании директории, в случае обнаружения *.sln манифеста, список анализируемых манифестов будет составлен из перечисленных в нем компонентов. Остальные компоненты не входящие в состав решения будут проигнорированы.

#Поддержка `Directory.Packages.props`

Агент автоматически обнаруживает файл `Directory.Packages.props` при сканировании `.csproj`-проектов и использует версии пакетов, указанные в нём. Поиск файла выполняется вверх по дереву директорий от расположения проекта до корня сканирования.

#Общая информация

- В рамках экосистемы .NET основным манифестом считается `*.csproj`, lock-файлом `packages.lock.json`;
- Манифест `deps.json` считается отдельным lock-файлом и не связывается с другим манифестом. В нем указаны зависимости необходимые для целевого рантайма.

#Работа с зависимостями в PHP

#Composer

#Создание файла `composer.lock`

1. Инициализируйте проект:

   ```
   composer init
   ```

2. Установите зависимости:

   ```
   composer install
   ```

   или создайте lock-file напрямую:

   ```
   composer update
   ```

#Работа с зависимостями в Python

#pip

#Создание файла `requirements.txt`

1. Установите зависимости и сохраните их в lock-файл:

```
pip freeze > requirements.txt
```

#pipenv

#Создание файла `Pipfile.lock`

1. Установите pipenv:

```
pip install pipenv
```

2. Создайте `Pipfile.lock`:

```
pipenv install
```

#poetry

#Создание файла `poetry.lock`

Если файл `poetry.lock` еще не существует, Poetry создаст его автоматически при установке зависимостей. Если файл уже существует, он будет обновлен. Для этого выполните команду:

```
poetry lock
```

Эта команда обновит зависимости, указанные в `pyproject.toml`, и создаст или обновит файл `poetry.lock`.

#pipdeptree

#Создание файла `pipdeptree.txt`

При обнаружении файла `pipdeptree.txt` агент проанализирует его содержимое как результат вывода утилиты pipdeptree в стандартном формате дерева зависимостей. Для создания файла можно использовать следующие команды:

```
pipdeptree > pipdeptree.txt
```

Для фильтрации вывода по конкретным пакетам окружения:

```
pipdeptree --packages "example1,example2" > pipdeptree.txt
```

Взаимодействие с другими манифестами

Для того чтобы зависимости основного манифеста проекта (например, `requirements.txt`) не отображались в результатах анализа вместе с результатом анализа pipdeptree рекомендуется исключить этот манифест из сканирования:

```
    johnny scan python . \
      --ignore "requirements.txt"
```

#uv

#Создание файла `uv.lock`

Если файл `uv.lock` еще не существует, uv создаст его автоматически при установке зависимостей. Если файл уже существует, он будет обновлен. Для этого выполните команду:

```
uv lock
```

Эта команда обновит зависимости, указанные в `pyproject.toml`, и создаст или обновит файл `uv.lock`.

#Поддержка механизма UV workspaces

Механизм UV workspaces позволяет централизованно управлять несколькими пакетами.

В `pyproject.toml` в секции workspaces может быть указана следующая запись:

```
[tool.uv.workspace]
members = [
    "packages/core",
    "packages/api"
]
```

В таком случае агент Johnny будет обрабатывать корневой `pyproject.toml` и все `pyproject.toml` всех пакетов из workspace как единое целое.

#pdm

#Создание файла `pdm.lock`

Если файл `pdm.lock` еще не существует, pdm создаст его автоматически при установке зависимостей. Если файл уже существует, он будет обновлен. Для этого выполните команду:

```
pdm lock
```

Эта команда обновит зависимости, указанные в `pyproject.toml`, и создаст или обновит файл `pdm.lock`.

#Создание файла `pylock.toml`

Помимо стандартного формата pdm позволяет сформировать lock-файл в формате `pylock.toml`. Для фиксирования зависимостей в этом формате перед выполнением `pdm lock` необходимо выполнить следующую команду:

```
pdm config lock.format pylock
```

#Работа с зависимостями в Ruby

#Bundler

#Создание файла `Gemfile.lock`

1. Инициализируйте проект:

   ```
   bundle init
   ```

2. Установите зависимости:

   ```
   bundle install
   ```

   или создайте lock-file напрямую:

   ```
   bundle lock
   ```

#CodeScoring.Secrets

#Общее описание

CodeScoring.Secrets – это модуль для поиска чувствительной информации в коде (пароли, API ключи, токены), который использует собственную модель машинного обучения для значительного снижения количества ложных срабатываний при сканировании.

Поиск секретов осуществляется через открытые инструменты анализа, на данный момент используется движок Gitleaks.

#Создание конфигурации для поиска секретов

1. Для начала работы с модулем Secrets необходимо предварительно создать VCS или CLI проект в разделе `Настройки -> Проекты`.

2. Задать конфигурацию движка секретов в разделе `Настройки -> Секреты`, открыв форму по кнопке **Добавить**.

3. В форме конфигурации необходимо указать имя, выбрать движок для поиска секретов в коде и прописать ему стандартную конфигурацию – она будет передана на вход движка при сканировании.

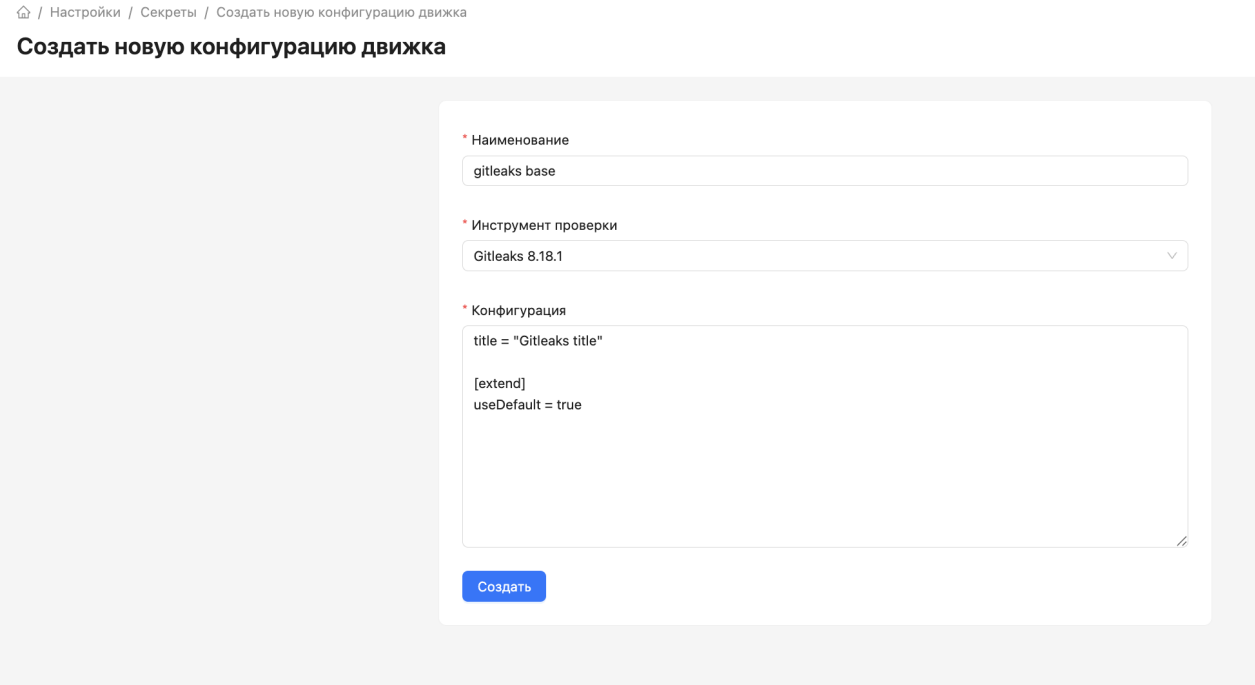
В поле **Инструмент проверки** можно выбрать один из поддерживаемых движков:

- **Gitleaks 8.27.0**;
- **TruffleHog 3.93.8**.

Пример конфигурации для Gitleaks:

```
title = "Gitleaks title"

[extend]
useDefault = true
```



Подробнее с конфигурированием движка Gitleaks можно ознакомиться в документации инструмента.

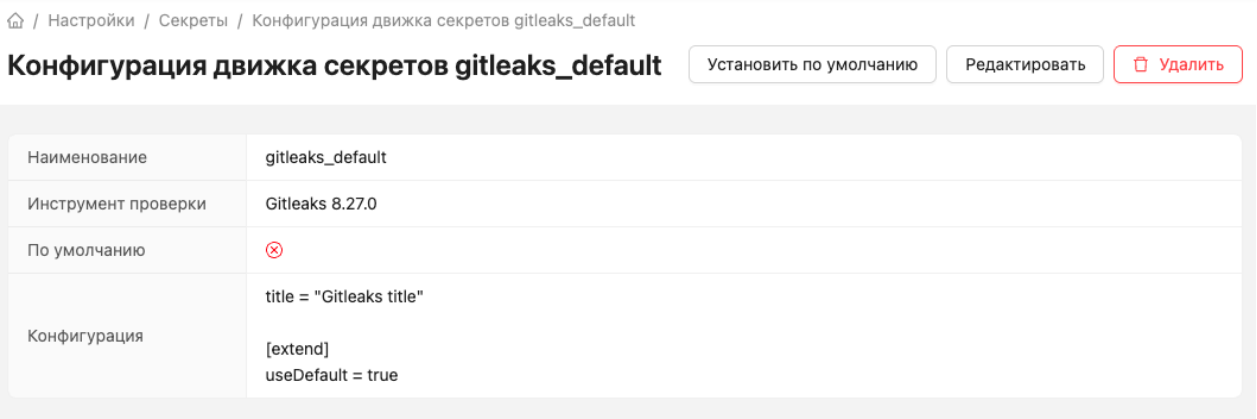
Пример конфигурации для TruffleHog:

```
detectors:
  - name: generic-api-key
    keywords:
      - key
      - api
      - token
      - secret
      - client
      - passwd
      - password
      - auth
      - access
    regex:
      # регулярное выражение generic-api-key из Gitleaks
      generic-api-key: "(?i)(?:key|api|token|secret|client|
passwd|password|auth|access)(?:[0-9a-z\\-_\\t .]{0,20})(?:[\
\s|']|[\\s|\"]){0,3}(?:=|>|:{1,3}=|\\|\\|:|<=|=>|:|\\?=)(?:'|
\"|\\s|=|\\x60){0,5}([0-9a-z\\-_.=]{10,150})(?:['|\"|\\n|\\r|\
\s|\\x60|;]|$)"
```

Подробнее с конфигурированием движка TruffleHog можно ознакомиться в документации инструмента.

#Создание конфигурации движка секретов по умолчанию

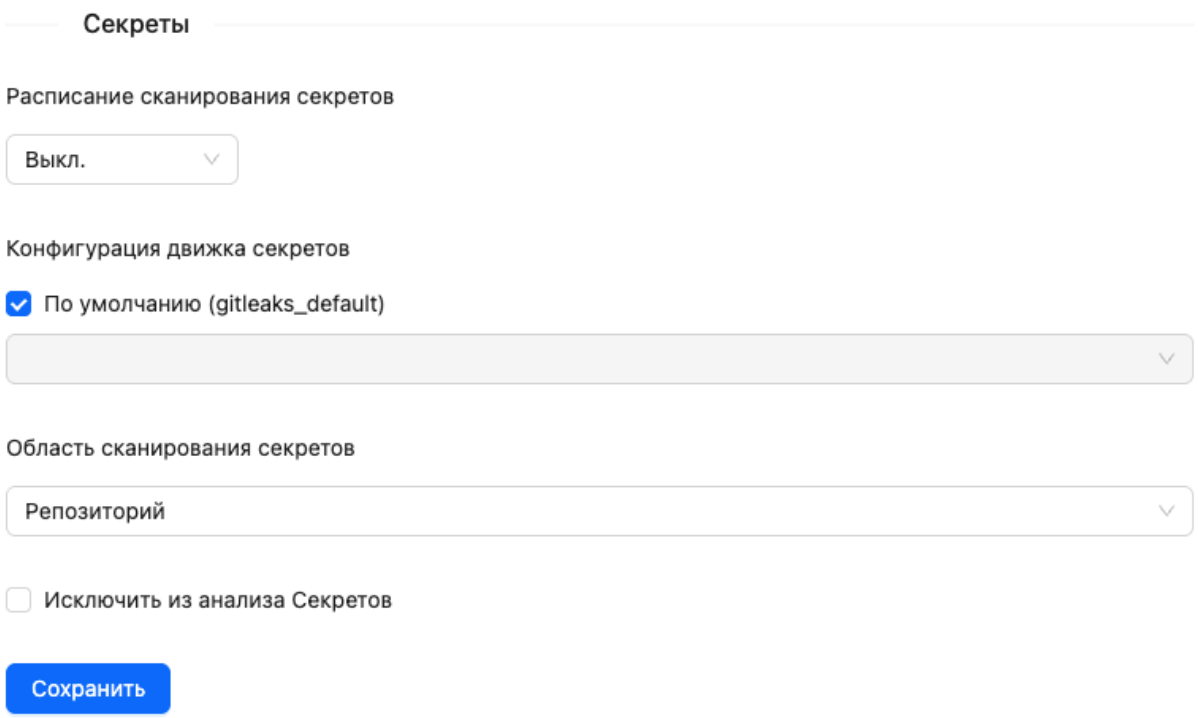
Для задания конфигурации по умолчанию необходимо в настройках конфигурации нажать на кнопку **Установить по умолчанию**.



Редактирование конфигурации по умолчанию

Нельзя установить более одной конфигурации по умолчанию, а также удалить используемую по умолчанию конфигурацию.

Для использования в проекте конфигурации по умолчанию необходимо в настройках проекта в разделе **Секреты** установить чек-бокс **По умолчанию**. В круглых скобках будет указана конфигурация, используемая в данный момент по умолчанию.



Изменение конфигурации по умолчанию

При установке новой конфигурации по умолчанию во всех проектах, где установлен чек-бокс **По умолчанию**, будет использована новая установленная конфигурация.

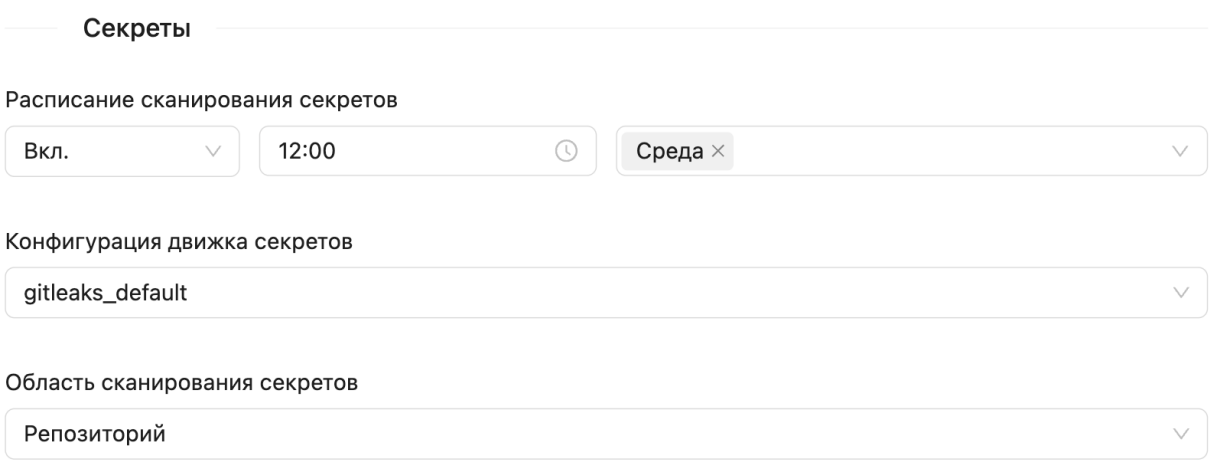
Конфигурация движка секретов у нового проекта

При создании нового проекта для сканирования секретов автоматически будет установлена конфигурация по умолчанию. Конфигурацию можно изменить в настройках проекта в разделе **Секреты**.

#Настройка VCS проекта для работы с секретами

Для работы модуля Secrets в рамках проекта необходимо задать параметры сканирования на странице настроек проекта в разделе `Настройки -> Проекты`:

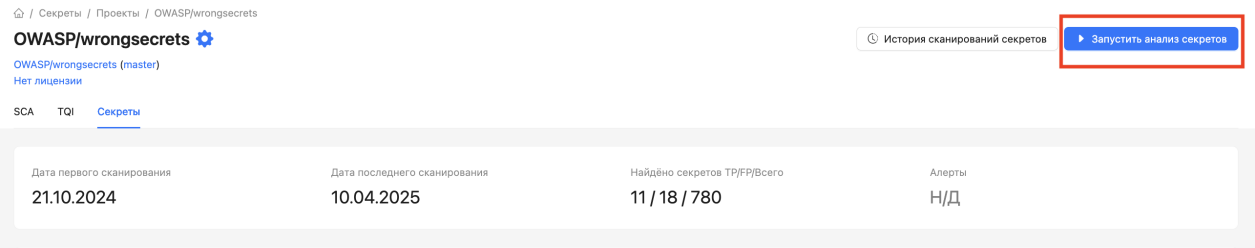
- **Расписание сканирования секретов** - график сканирования на наличие секретов (задается временем и днями недели);
- **Конфигурация движка секретов** - конфигурация движка секретов;
- **Область сканирования секретов** - область применения сканирования:
  - **Репозиторий** - для сканирования всех веток в рамках репозитория;
  - **Ветка или тег** - для сканирования ветки по умолчанию в настройках проекта;
- **Исключить из анализа Секретов** - исключить данный проект из анализа Секретов;



#Запуск поиска секретов

#Поиск секретов в отдельном проекте

Для запуска поиска секретов необходимо перейти на вкладку Секреты выбранного проекта и нажать на кнопку **Запустить анализ секретов**, после чего запустится поиск секретов в данном проекте.



В зависимости от величины проекта анализ может длиться от нескольких секунд до несколько минут.

#Поиск секретов по расписанию (VCS-проекты)

Помимо ручного запуска, можно настроить анализ отдельных проектов по расписанию.

Управление происходит на странице проекта в разделе `Настройки -> Проекты`.

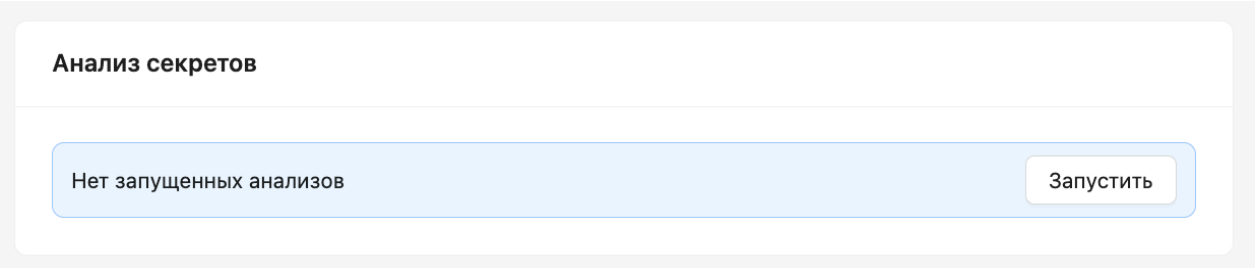
По умолчанию параметр **Расписание сканирования секретов** имеет значение **Выкл.**. Для активации анализа по расписанию необходимо выбрать **Вкл.** и указать время и дни недели.

Примечание: Время сканирования будет учитываться по UTC +3.

Примечание: Поиск не будет запущен, если у проекта не выбрана конфигурация поиска секретов.

#Поиск секретов во всех проектах

Для запуска поиска секретов по всем проектам необходимо перейти в раздел Режим работы и нажать на кнопку **Запустить** под заголовком **Анализ секретов**.



#Анализ по расписанию

Помимо ручного запуска, можно настроить анализ отдельных проектов по расписанию. Управление происходит на странице проекта в разделе `Настройки -> Проекты`.

По умолчанию параметр **Расписание сканирования секретов** имеет значение **Выкл.**. Для активации анализа по расписанию необходимо выбрать **Вкл.** и указать время и дни недели.

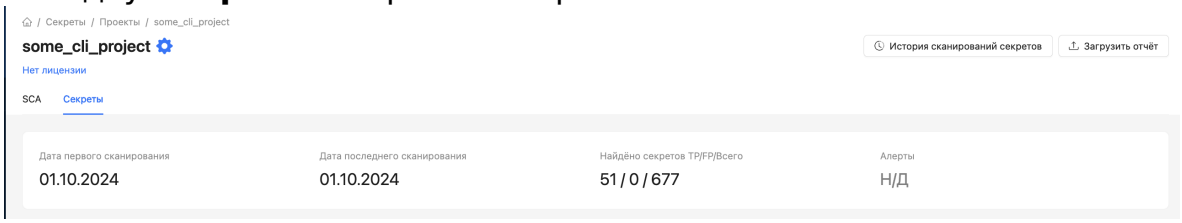
Примечание: Время сканирования будет учитываться по UTC +3.

#Поиск секретов в CLI проектах

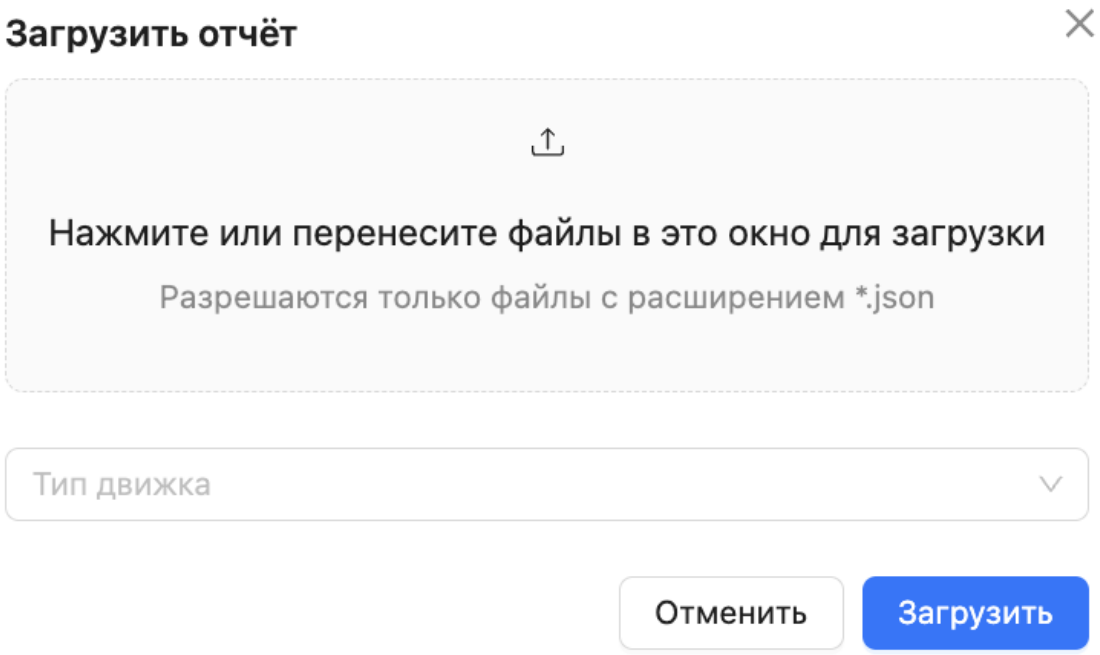
#Загрузка отчета в CLI проект

В CLI проект можно импортировать отдельно созданный отчет в формате `json`. Об использовании **Gitleaks** в командой строке можно прочесть в документации инструмента.

1. После получения отчета от инструмента необходимо перейти на вкладку **Секреты** выбранного проекта.



2. Нажать на кнопку **Загрузить отчёт**.



3. Выбрать файл с отчетом и указать каким инструментом он был создан в поле **Тип движка**, затем нажать на кнопку **Загрузить**.

#Сканирование CLI-проектов

Для сканирования секретов в CLI проектах и интеграции в CI/CD конвейер используется команда `secrets gitleaks dir` в консольном агенте Johnny.

Важно: агент работает только с версией Gitleaks 8.19.0 и выше.

Более подробно об использовании команды можно прочесть в документации консольного агента.

#Работа с найденными секретами

#Просмотр секретов в отдельном проекте

Для просмотра данных о секретах в разрезе проекта необходимо перейти на вкладку **Секреты** на странице проекта.

Страница отображает следующую сводную информацию:

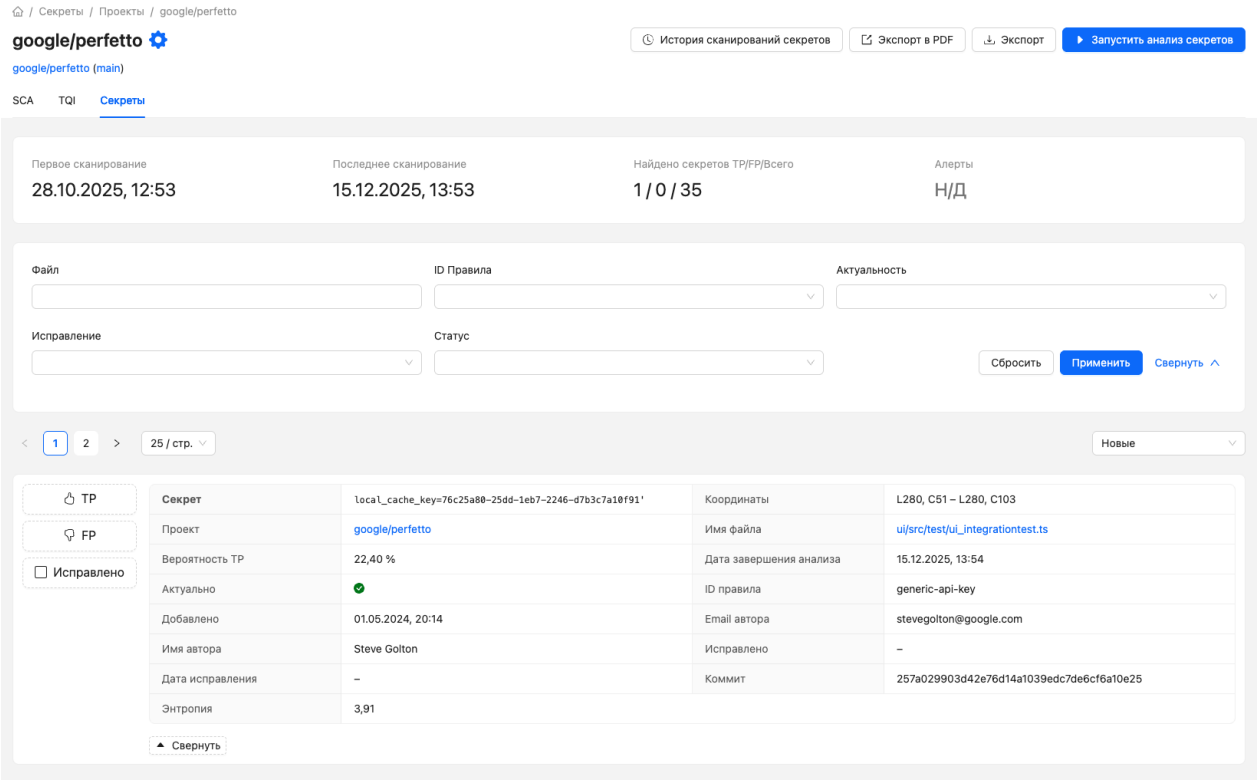
- Дата первого сканирования;
- Дата последнего сканирования;
- Количество найденных секретов по категориям (истинно-позитивные, ложно-позитивные, все).

Таблица с найденными секретами имеет следующие поля:

- **Секрет** – содержание секрета;
- **Проект** – название проекта, в котором был найден секрет;
- **Имя файла** – имя файла, в котором был найден секрет;
- **Координаты** - строка и столбец начала и конца секрета в файле;
- **Вероятность TP** – вероятность истинной находки;
- **Дата завершения анализа** – дата и время завершения сканирования;
- **Актуально** – был ли найден секрет при последнем сканировании;
- **ID правила** – идентификатор правила поиска секретов в рамках используемого движка конфигурации;
- **Добавлено** – дата и время добавления секрета в код;
- **Email автора** – почта автора, ответственного за добавление секрета;
- **Имя автора** – имя автора, ответственного за добавление секрета;
- **Исправлено** – имя пользователя, который пометил находку как исправленную;
- **Дата исправления** – дата исправления;
- **Коммит** – хэш коммита, в котором был добавлен секрет;
- **Энтропия** - энтропия найденного секрета.

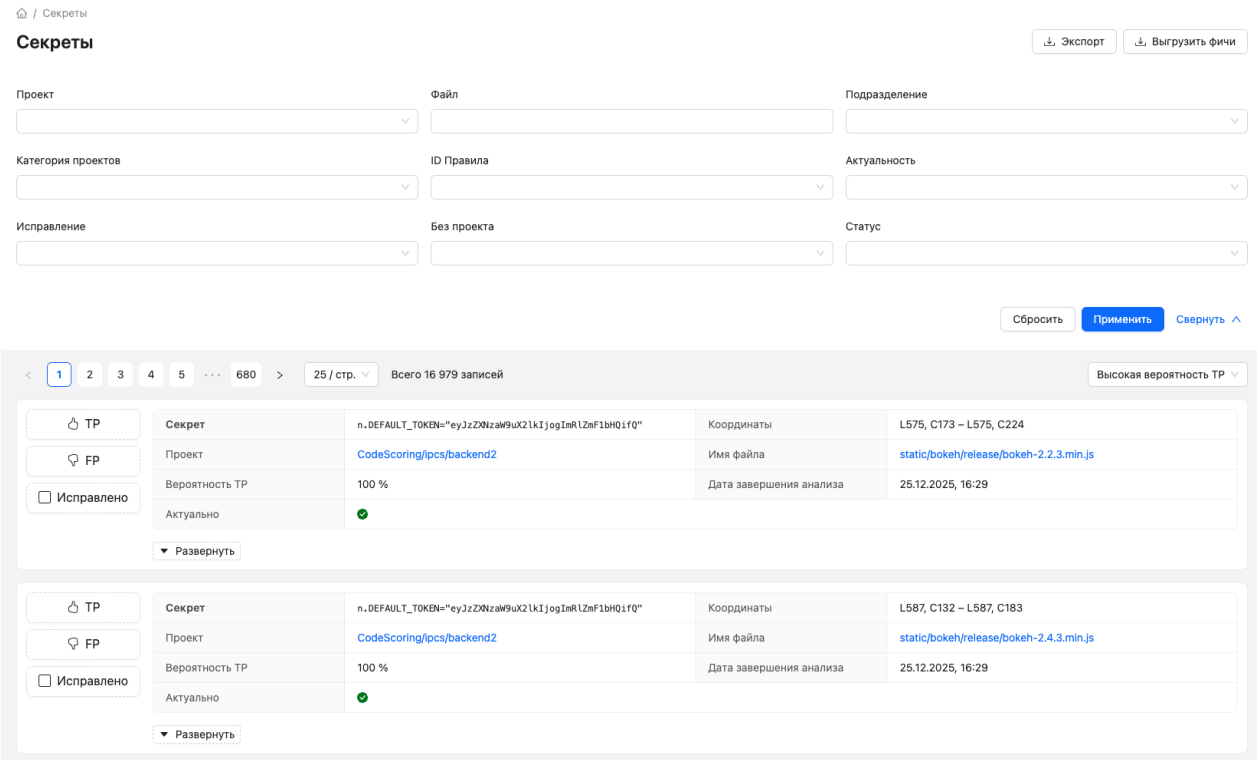
Энтропия

Данный параметр является энтропией Шеннона и может быть использован в правилах, как пороговое значение.



#Просмотр секретов

Чтобы изучить найденные секреты по всем проектам, необходимо перейти в раздел Секреты в меню системы.



Таблицу с секретами можно отфильтровать по критериям:

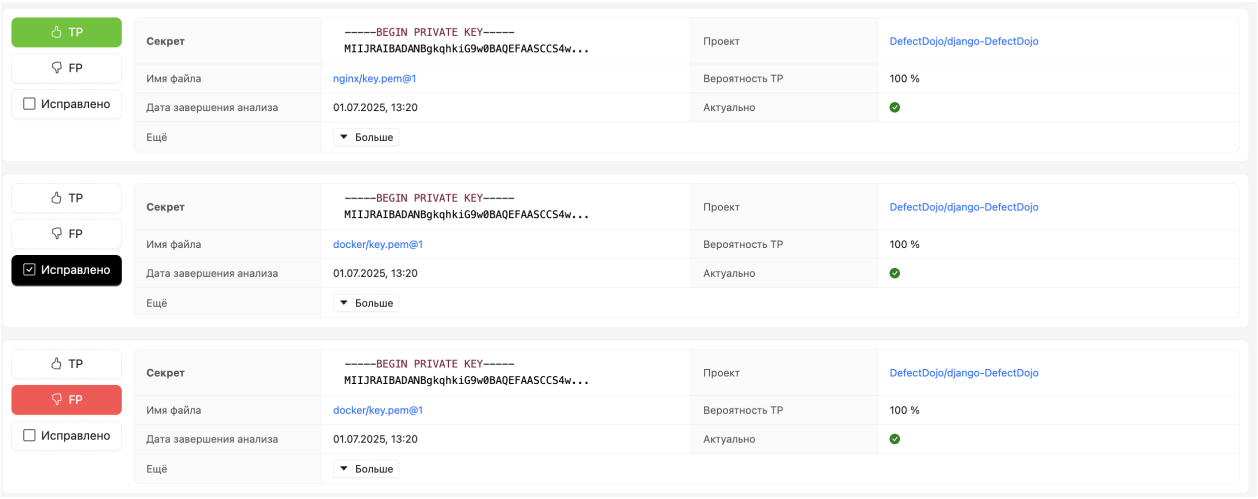
- **Проект** – название проекта, в котором был найден секрет;
- **Файл** – имя файла, в котором был найден секрет;

- **Подразделение** – подразделение организации, к которому принадлежит проект с найденным секретом;
- **Категория** – категория проекта с найденным секретом;
- **ID правила** – идентификатор правила поиска секретов в рамках используемого движка конфигурации;
- **Актуальный** – секрет найден при последнем сканировании;
- **Исправленный** – секрет исправлен;
- **Без проекта** – секрет не привязан к проекту;
- **Статус** - статус находки (истинно-положительный, ложно-положительный, без статуса).

#Разметка истинных и ложных срабатываний.

Каждый секрет можно обозначить как истинно-положительный, ложно-положительный или исправленный. Для этого используются кнопки **TP**, **FP** и **Исправлено** в таблице найденных секретов.

Ручная разметка будет использоваться для дообучения модели машинного обучения.



#Выгрузка данных по секретам

Чтобы получить выгрузку данных по найденным секретам можно воспользоваться кнопкой **Экспорт** в правом верхнем углу раздела.

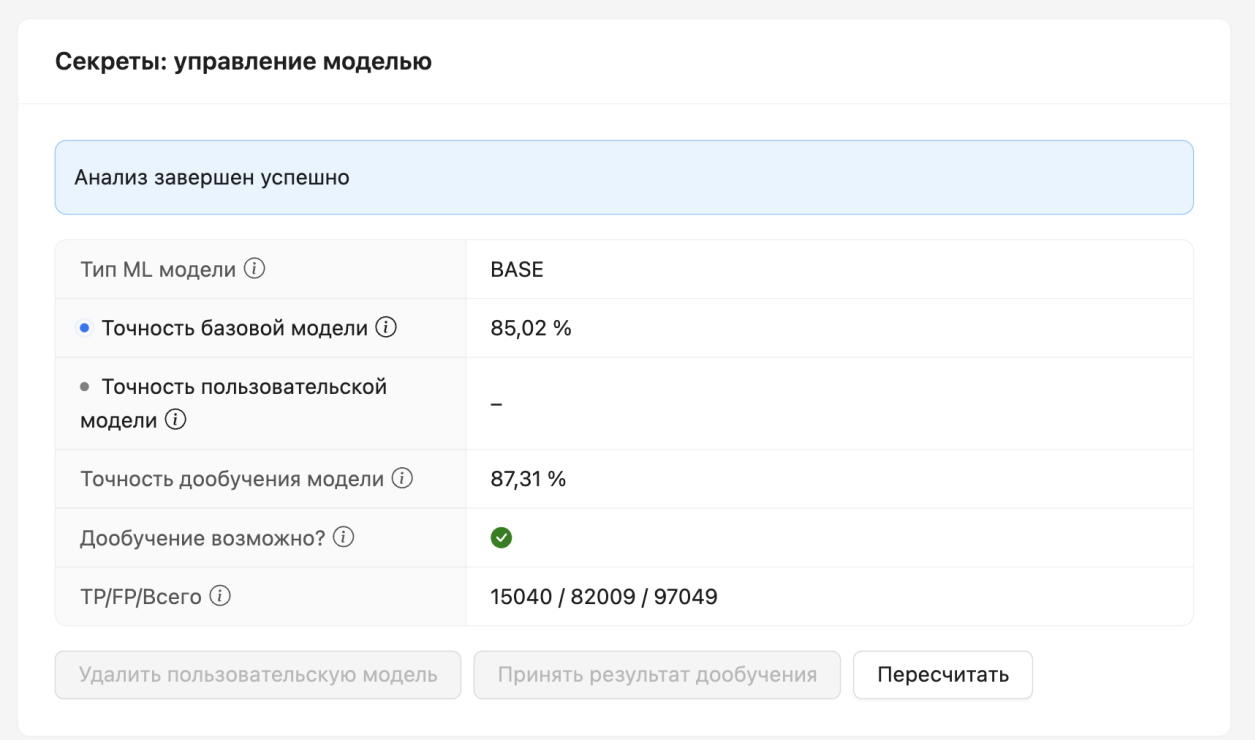
#Управление моделью машинного обучения

По умолчанию CodeScoring использует собственную модель машинного обучения чтобы снизить количество ложных срабатываний при поиске секретов. С помощью ручной разметки найденных секретов можно дообучить модель и улучшить результаты поиска на собственном исходном коде.

Для того, чтобы дообучить модель, необходимо перейти в раздел Настройки -> Режим работы и нажать на кнопку **Запустить** в секции **Секреты: управление моделью**. Для активации возможности

дообучения модели необходимо разметить минимум 1000 найденных секретов как истинно-положительные или ложно-положительные.

После дообучения можно сравнить результаты поиска секретов и на их основе либо принять пользовательскую модель (**Принять результат дообучения**), либо вернуться к базовой модели (**Удалить пользовательскую модель**).



В секции управления пользователю выводится информация о текущем состоянии модели:

- **Тип ML модели** – тип использованной модели (базовая или пользовательская);
- **Точность базовой модели** – точность поиска на основе размеченных находок. Истинно-положительные находки берутся за единицу, ложно-положительные — за ноль. Итоговая точность – это среднее значение всех результатов, представленное в процентах.
- **Точность пользовательской модели** – точность поиска с использованием пользовательской модели;
- **Точность дообучения модели** – точность поиска с использованием последнего дообучения;
- **Дообучение возможно?** – возможность дообучения модели на основе текущей разметки (с указанием причины в случае невозможности дообучения);
- **TP/FP/Всего** – истинно-положительные, ложно-положительные и все находки.

Важно: если дообучение модели невозможно – это значит, что разметка недостаточно полная. В таком случае необходимо обозначить большее количество находок как истинно-положительных или ложно-положительных.

#CodeScoring.TQI

#Общее описание

CodeScoring.TQI – это модуль для анализа качества собственного исходного кода организации, определяющий основные параметры технического долга.

Основные функциональные возможности модуля позволяют:

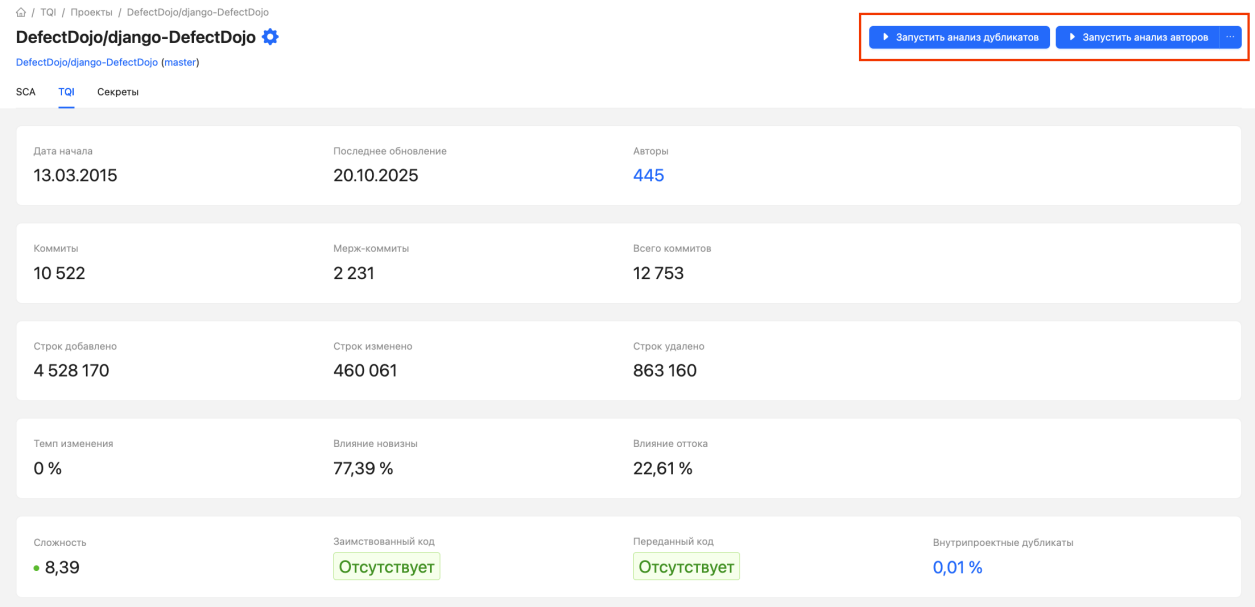
- Просматривать динамику проекта с подсчетом цикломатической сложности кода;
- Строить профили авторов с сравнением схожести;
- Отслеживать дубликаты кода внутри и между проектов.

#Запуск TQI анализа

#Анализ отдельного проекта

После успешного клонирования проекта из VCS на его странице в разделе `TQI -> Проекты` появляется возможность запустить два вида анализа:

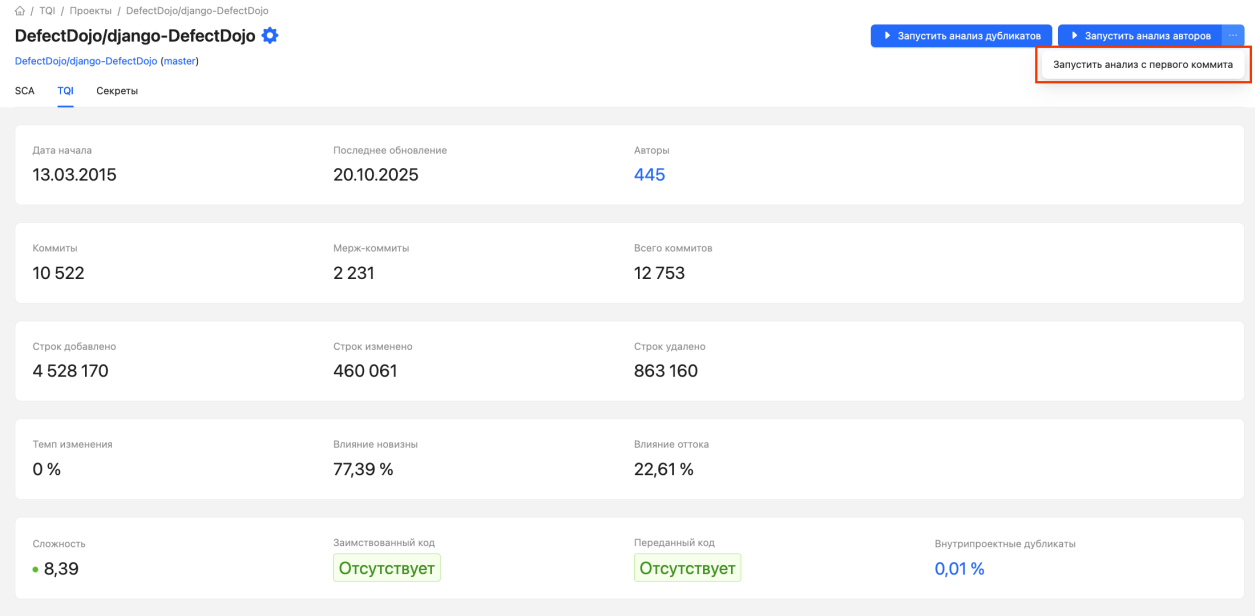
1. Анализ дубликатов
2. Анализ авторов



После запуска анализа процесс выполняется в фоновом режиме, и по его завершении результаты становятся доступны на странице проекта.

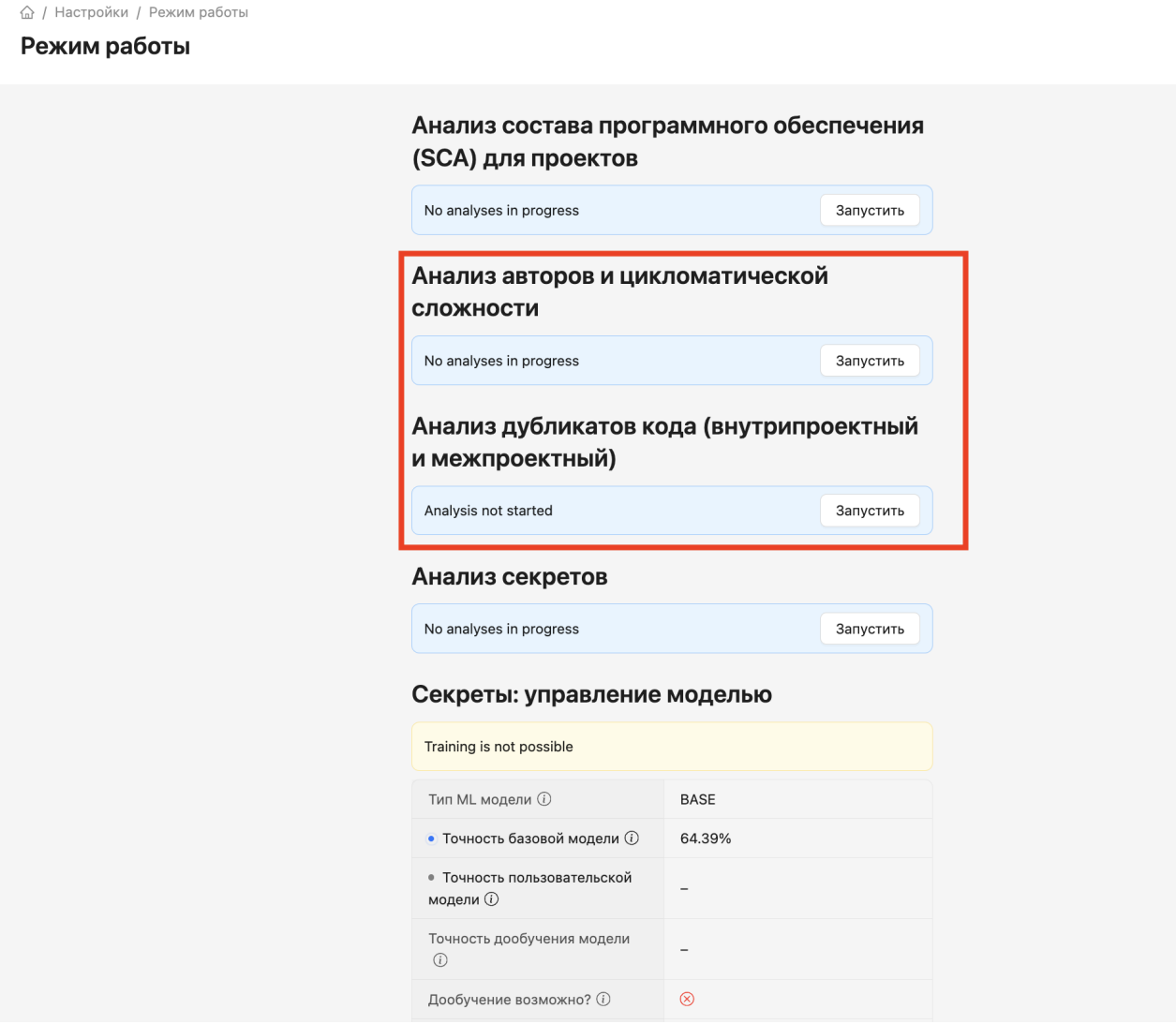
#Анализ проекта с первого коммита

Также можно запустить анализ авторов с первого коммита в проекте. Анализ обработает все коммиты в репозитории и актуализирует данные в CodeScoring.



#Анализ всех проектов

Для запуска анализа по всем VCS проектам в системе необходимо перейти в раздел Настройки -> Режим работы и запустить один из двух видов анализа.



#Анализ по расписанию

Помимо ручного запуска, можно настроить анализ отдельных проектов по расписанию. Управление происходит на странице проекта в разделе `Настройки -> Проекты`.

По умолчанию расписание не настроено и выключено. Для активации анализа по расписанию необходимо его включить и указать время и дни недели. Анализ по расписанию дубликатов и авторов можно настроить независимо.

Примечание: Время сканирования будет учитываться по UTC +3.

#Просмотр результатов TQI анализа

#Страница проекта

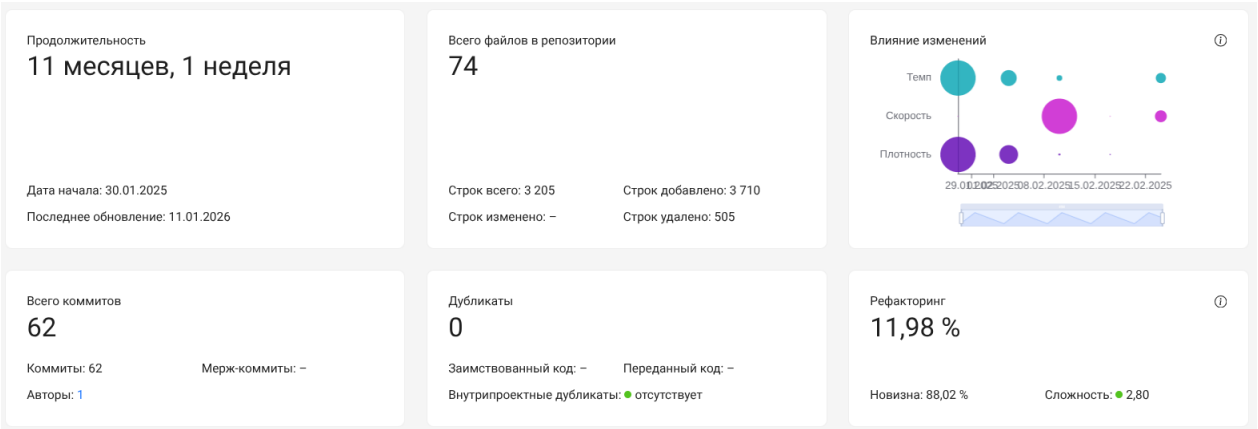
После завершения анализа на странице проекта в разделе `TQI -> Проекты` становится доступен детализированный отчет,

содержащий ключевые метрики, информацию об авторах, динамику изменений и список коммитов.

#Общая статистика по проекту

Начало отчета фиксирует ключевые показатели по результатам анализа:

- **Дата начала проекта** – фиксирует момент первого коммита в репозитории;
- **Продолжительность** - количество месяцев от времени первого коммита до времени последнего изменения;
- **Последнее обновление проекта** – время последнего зафиксированного изменения;
- **Авторы** – количество разработчиков, которые вносили изменения в кодовую базу;
- **Коммиты** - количество коммитов в репозитории;
- **Мерж-коммиты** - количество мерж-коммитов в репозитории
- **Всего коммитов** – общее число коммитов (коммиты + мерж-коммиты) в репозитории;
- **Строк добавлено** – количество добавленных строк кода;
- **Строк изменено** - количество измененных строк кода;
- **Строк удалено** – количество удаленных строк кода;
- **Темп изменения** - средний объем изменений в коммитах относительно общего количества кода в репозитории;
- **Новизна** - доля добавленных строк кода относительно общего количества строк кода в репозитории;
- **Рефакторинг** - доля измененных и удаленных строк кода относительно общего количества строк кода в репозитории;
- **Средняя цикломатическая сложность** – показатель сложности кода, основанный на количестве ветвлений в логике программы;
- **Наличие заимствованного кода** – выявляет участки кода, которые были скопированы из других проектов внутри организации;
- **Наличие переданного кода** – определяет фрагменты кода, переданные из других проектов внутри организации;
- **Наличие внутрипроектных дубликатов** – фиксирует повторяющиеся участки кода внутри проекта.



#Авторский состав

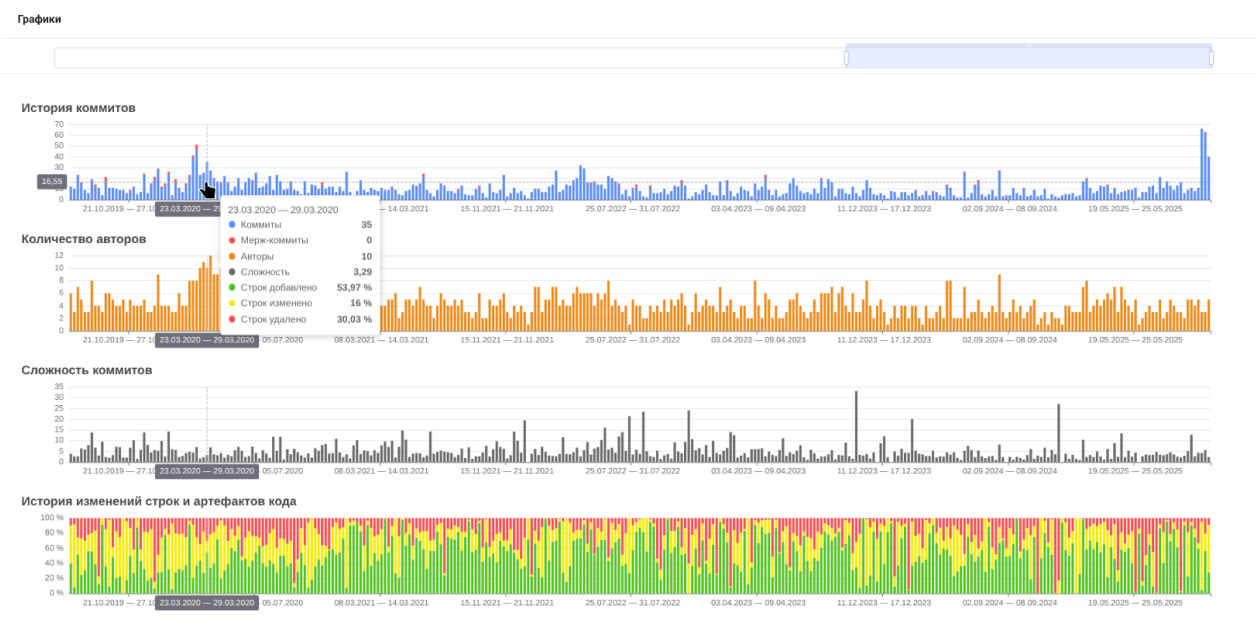
Список авторов можно посмотреть в виде таблицы, изменить отображение колонок и выгрузить в формате CSV.

- **Автор** – имя и почта автора;
- **Работает с** – дата первого коммита автора;
- **Последняя активность** – дата последнего коммита автора в проекте;
- **Активность, месяцы** – количество месяцев, в течение которых автор активно коммитил изменения;
- **Всего коммитов** – общее количество коммитов, сделанных автором в проекте. Доступна детализация при наведении курсора на значение;
- **Сложность** – средняя цикломатическая сложность по коммитам автора в проекте;
- **Дубликаты** – количество заимствованных фрагментов кода, сделанных автором;
- **Технологии** – языки программирования, с которыми работает автор (определяется по его коммитам).

#Динамика проекта

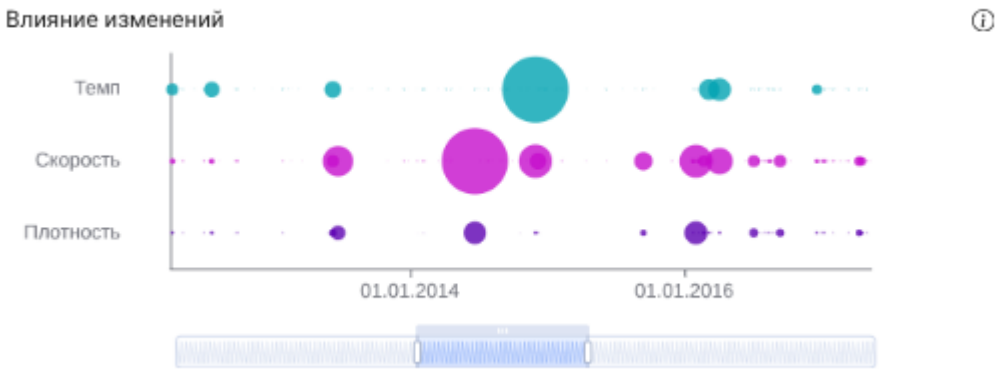
Историю проекта можно отследить по графикам, показывающим динамику проекта по следующим параметрам:

- История добавлений/изменений/удалений строк кода;
- История коммитов;
- Количество авторов;
- Сложность коммитов.



Кроме этого, оценить влияние изменений на проект можно по следующим параметрам:

- Темп;
- Скорость;
- Плотность.



Временной промежуток на графиках можно менять с помощью слайдера, выбирая интересующий период для анализа.

#Список коммитов с расчетом цикломатической сложности

Для каждого коммита рассчитывается цикломатическая сложность, а также показывается его контекст:

- **Хэш** – уникальный идентификатор коммита с ссылкой на систему контроля версий;
- **Сообщение коммита** – краткое описание внесенных изменений, указанное автором при коммите;
- **Дата коммита** – дата и время, когда было выполнено изменение;
- **Строк добавлено** – количество строк кода, добавленных в коммите;
- **Строк удалено** – количество строк, удаленных в коммите;
- **Сложность** – значение цикломатической сложности, рассчитываемое на основе внесенных изменений;
- **Автор** – имя разработчика, выполнившего коммит.

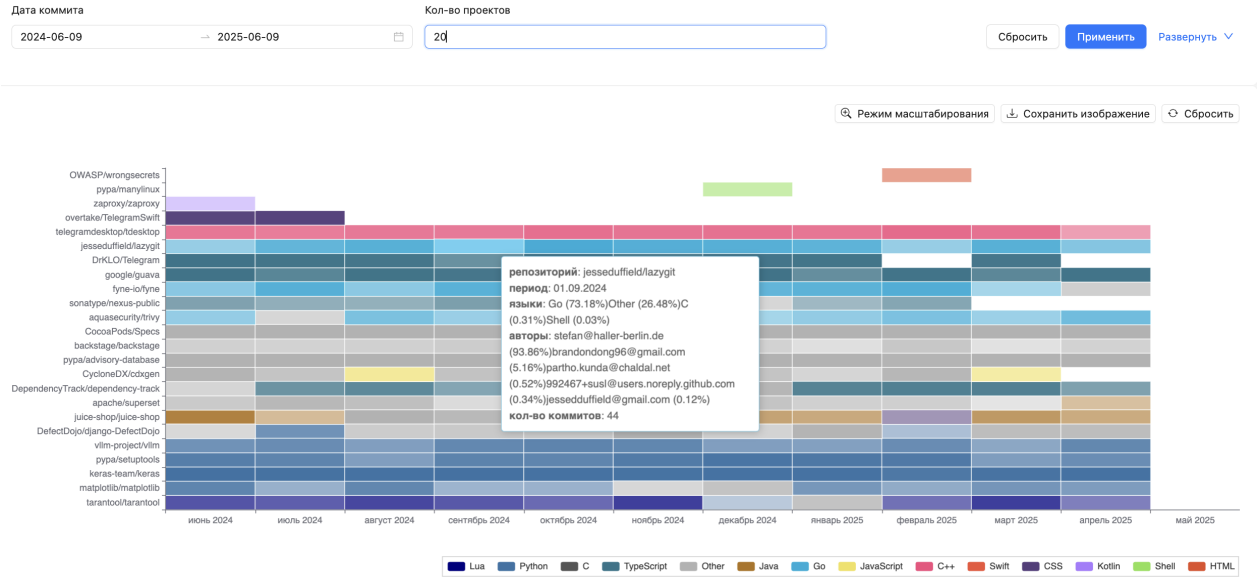
#Визуализация результатов

#Карта активности

Карта активности доступна в разделе `TQI –> Проекты` на вкладке **Карта активности**. Она отображает весь вклад авторов за выбранный промежуток времени по набору проектов, который можно отфильтровать по следующим параметрам:

- **Дата коммита** – период, в течение которого был совершен коммит в системе контроля версий;

- **Количество проектов** – общее количество проектов, отображаемое на карте;
- **Подразделение** – часть организации, которая управляет проектом;
- **Категория проекта** – категория проекта, назначенная в рамках системы CodeScoring;
- **Технологии** – языки программирования, используемые в проекте.

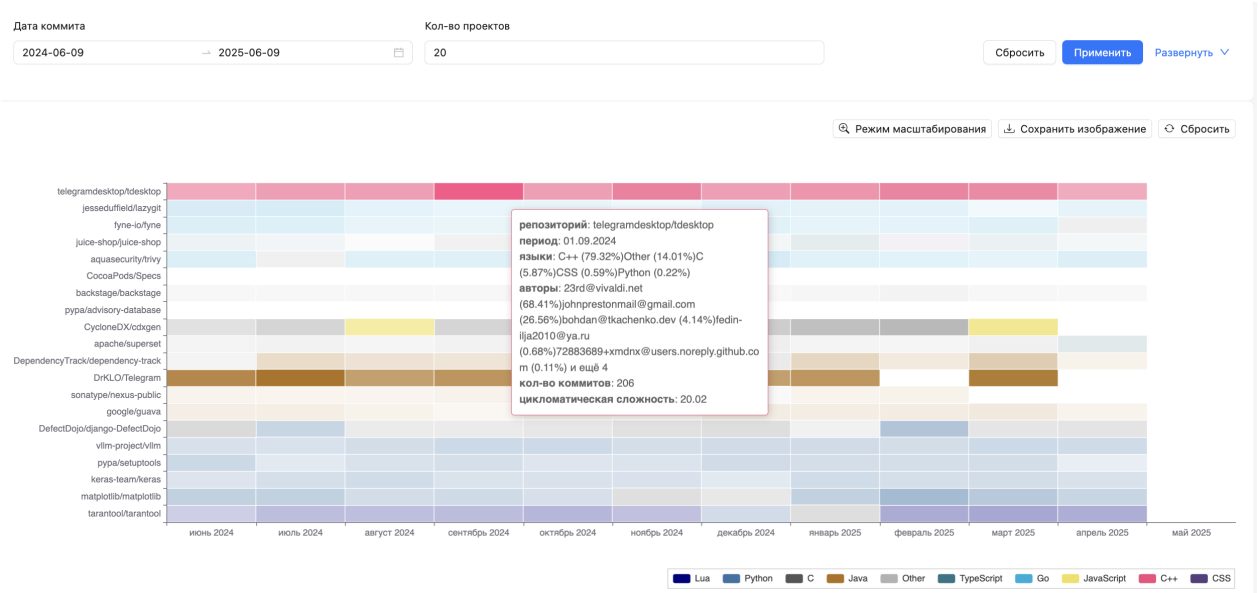


Карту можно также сохранить как PNG изображение.

#Карта сложности

Карта сложности доступна в разделе TQI –> Проекты на вкладке **Карта сложности**. Она отображает изменение сложности набора проектов, который можно отфильтровать по следующим параметрам:

- **Дата коммита** – период, в течение которого был совершен коммит в системе контроля версий;
- **Количество проектов** – общее количество проектов, отображаемое на карте;
- **Подразделение** – часть организации, которая управляет проектом;
- **Категория проектов** – категория, назначенная в рамках системы CodeScoring;
- **Технологии** – языки программирования, используемые в проекте.



Карту можно также сохранить как PNG изображение.

#Отслеживание дубликатов кода

CodeScoring.TQI позволяет отслеживать фрагменты кода, которые были скопированы из одного проекта организации в другой, или продублированы в рамках одного проекта.

#Межпроектные дубликаты

В разделе `TQI -> Дубликаты кода -> Межпроектные` отображается список проектов, между которыми было произведено копирование. При нажатии на количество дубликатов отобразится таблица с детальной информацией о произведенном копировании с указанием следующих полей:

- Выдержки скопированного кода с ссылкой на коммит в системе контроля версий;
- Направление копирования;
- Дата коммита;
- Автор;
- Количество скопированных строк;
- Уровень встречаемости дубликата (низкий, средний, высокий);
- Технология.

#Внутрипроектные дубликаты

В разделе `TQI -> Дубликаты кода -> Внутрипроектные` отображается список проектов, в которых копировались фрагменты кода, а также процент дубликатов от общего кода проекта и уровень встречаемости. При нажатии на количество дубликатов отобразится таблица с

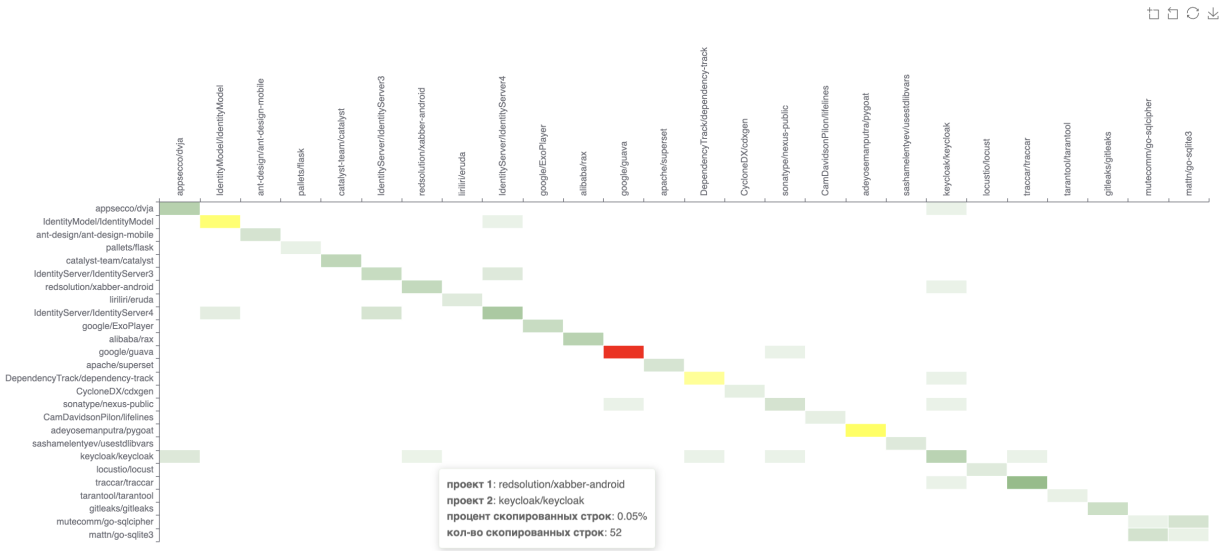
детальной информацией о произведенном копировании с указанием следующих полей:

- Выдержки скопированного кода с ссылкой на коммит в системе контроля версий;
- Направление копирования;
- Дата коммита;
- Автор;
- Количество скопированных строк;
- Уровень встречаемости дубликата (низкий, средний, высокий);
- Технология.

#Карта дубликатов

В разделе `TQI -> Дубликаты кода -> Карта дубликатов` можно увидеть визуализацию заимствований между проектами. Карту дубликатов можно отфильтровать по следующим полям:

- **Подразделение** – часть организации, к которой относится проект;
- **Категория проекта** – категория проекта в рамках системы CodeScoring.



При нажатии на пересечение между двумя проектами произойдет переход на страницу с детальным описанием дубликатов. Карту можно также сохранить как PNG изображение.

#Расчет метрик технического долга

CodeScoring.TQI отслеживает несколько метрик технического долга. Основные из них – **цикломатическая сложность**, **встречаемость дубликатов**, **темп изменений**.

#Расчет цикломатической сложности

Цикломатическая сложность – это показатель, который отражает количество независимых путей в коде. Чем выше это значение, тем сложнее поддерживать и тестировать код.

Расчет ведется по формуле:

```
M = E - N + 2P
```

Где:

- **M** – цикломатическая сложность;
- **E** – количество рёбер (переходов между операторами);
- **N** – количество узлов (операторов, условий);
- **P** – количество компонент связности (обычно 1).

Пример:

```
def is_even(x):
    print("Even" if x % 2 == 0 else "Odd")
```

Цикломатическая сложность:

- **N = 3** (вход, `if-else`, `print`).
- **E = 3** (вход -> `if-else`, `if-else` -> `print`, `print` -> выход).
- **P = 1** (функция).

Таким образом цикломатическая сложность составляет M = E - N + 2P = 3 - 3 + 2 = **2**

Уровни сложности:

- **Низкая**: < 10 (простой код, легко читаемый и поддерживаемый);
- **Средняя**: 10–20 (умеренно сложный код, требует внимания при изменениях);
- **Высокая**: > 20 (сложный код, возможны проблемы с тестированием и поддержкой).

#Расчет процента внутрипроектных дубликатов

Этот показатель отражает, какая часть кода проекта является дублируемой. Он рассчитывается по следующей формуле:

```
Процент дубликатов = (количество дублируемых строк кода / общее количество строк кода) * 100%
```

Чем выше этот процент, тем больше кода можно оптимизировать с помощью рефакторинга.

#Расчет встречаемости дубликатов

Этот показатель оценивает масштаб распространения дублированного кода по проекту.

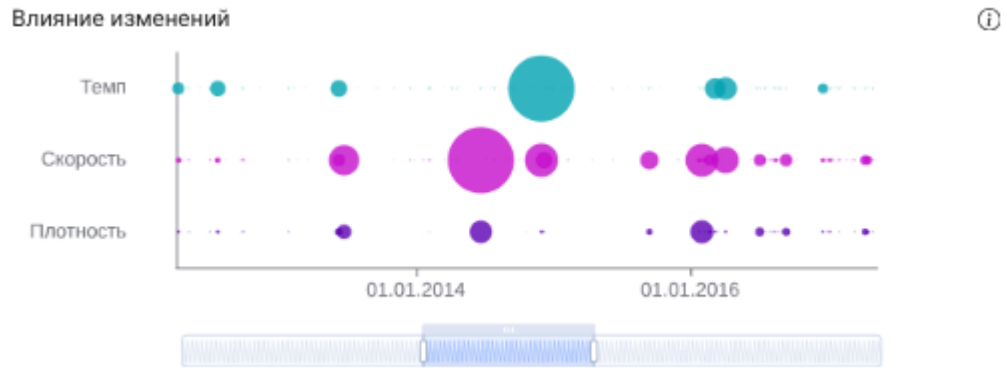
Категории:

- **Низкий уровень** – если дублируемых строк меньше 50;
- **Средний уровень** – если дублируемых строк от 50 до 300;
- **Высокий уровень** – если дублируемых строк больше 300.

#Графики влияния изменений

Оценить влияние изменений на проект можно по следующим параметрам на общем графике:

- Темп;
- Скорость;
- Плотность.



Размер элемента на графике позволяет оценить его влияние на выборку. Расчет выполняется за период. Минимальный период - неделя.

Размеры элементов в выборке нормированы. Нормирование выборки выполняется по формуле:

```
X’ = (X–Xmin)/(Xmax–Xmin)
```

Где:

- **Xmin** - минимальное значение в выборке;
- **Xmax** - максимальное значение в выборке;
- **X’** - нормализованное значение.

#Расчет темпа изменений

Темп изменений показывает объем изменения кода относительно общего объема строк кода. Высокий темп изменения кода указывает на

частые изменения требований или нестабильный ритм работы команды разработки.

Расчет ведется по формуле:

```
R = L / T
```

Где:

- **R** – темп изменений;
- **L** – количество внесенных (добавленных, удаленных, модифицированных) строк кода, за расчетный период;
- **T** – общее количество строк кода в репозитории проекта на начало расчетного периода.

#Расчет скорости изменений

Скорость изменений показывает объем изменений в коде относительно количества коммитов. При высоких значениях возрастает нагрузка при проведении код-ревью, тестирования и сопровождения.

Расчет ведется по формуле:

```
V = T / C
```

Где:

- **V** – плотность изменений;
- **L** – количество внесенных (добавленных, удаленных, модифицированных) строк кода, за расчетный период;
- **C** – количество коммитов.

#Расчет плотности изменений

Плотность изменений показывает какой объем изменений в коде был произведен относительно количества измененных файлов. Большое значение плотности свидетельствует об модификации большого количества модулей. Повышается нагрузка про проведении код-ревью, тестирования и сопровождения.

Расчет ведется по формуле:

```
D = T / F
```

Где:

- **D** – плотность изменений;
- **L** – количество внесенных (добавленных, удаленных, модифицированных) строк кода, за расчетный период;
- **F** – количество измененных файлов.

#Подключить проект и выполнить первый SCA-анализ

#Контекст

Первый практический шаг в CodeScoring.SCA — подключить репозиторий как VCS-проект и дождаться первого SCA-анализа. В этом сценарии для примера используется GitLab, но сам порядок работы такой же и для других поддерживаемых систем контроля версий: сначала создается подключение, затем проект, затем запускается первый анализ.

#Что получится

После выполнения сценария в CodeScoring появится подключенный проект с первым завершенным SCA-анализом. На странице проекта будут доступны результаты сканирования, а при необходимости — история запусков с метаданными по ветке и коммиту.

#Требования

Перед началом убедитесь, что у вас есть:

- доступ к репозиторию в GitLab;
- учетная запись GitLab, в которой можно создать `Personal Access Token`;
- доступ к CodeScoring с правами `VCS: добавление репозиториев` и `Projects: создание проектов`;
- лицензия CodeScoring, в которой включен модуль **SCA**.

#Шаги

#Шаг 1. Создайте токен доступа в GitLab

CodeScoring использует токен для чтения репозитория и проверки его содержимого при анализе.

1. Войдите в GitLab под своей учетной записью.
2. Откройте `Edit profile`.
3. В левом меню перейдите в раздел **Access Tokens**.
4. Укажите имя токена, например `codescoring-demo`.
5. В секции `scopes` включите `read_api` и `read_repository`.
6. Нажмите **Create personal access token**.
7. Скопируйте сгенерированный токен и сохраните его в безопасном месте.

Токен готов к использованию в настройках подключения GitLab в CodeScoring.

#Шаг 2. Добавьте подключение к GitLab в CodeScoring

На этом шаге GitLab становится доступен платформе как источник кода для будущих анализов.

1. В CodeScoring перейдите в `Настройки -> VCS`.
2. Нажмите **Добавить**.
3. Заполните форму подключения:
   - **Название** — понятное имя подключения, например `GitLab main`;
   - **Тип подключения** — `HTTPS`;
   - **Тип** — `Gitlab`;
   - **Адрес** — адрес GitLab, например `https://gitlab.com`;
   - **Токен доступа** — токен, созданный на предыдущем шаге.
4. Нажмите **Проверить подключение**.
5. Если проверка прошла успешно, нажмите **Добавить**.

После успешной проверки подключение можно выбрать при создании проекта.

#Шаг 3. Создайте VCS-проект

Теперь можно добавить сам репозиторий в CodeScoring и сразу запустить первый SCA-анализ после клонирования.

1. Перейдите в `Настройки -> Проекты`.
2. Нажмите **Создать** и выберите вкладку **VCS проекты**.
3. Заполните форму проекта:
   - **Репозиторий** — ссылка на репозиторий GitLab;
   - **VCS** — созданное на предыдущем шаге подключение GitLab;
   - **Название** — имя проекта в CodeScoring.
4. Оставьте включенной опцию **Запустить SCA после клонирования**.
5. Нажмите **Создать**.

После сохранения проекта начнется первоначальное клонирование репозитория, а затем автоматически запустится SCA-анализ.

#Шаг 4. Дождитесь завершения первого анализа

Во время первого запуска платформа получает исходный код из GitLab и строит первичный срез зависимостей и уязвимостей.

1. Откройте страницу созданного проекта.
2. При необходимости отслеживайте прогресс в разделе `Настройки -> Аудит лог`.
3. Дождитесь завершения анализа.

Когда анализ завершится, на странице проекта станут доступны результаты SCA.

#Шаг 5. Проверьте результаты анализа

После первого запуска важно убедиться, что проект действительно проанализирован и результаты можно использовать дальше.

1. На странице проекта откройте вкладку SCA.
2. Проверьте, что на вкладке отображаются результаты анализа проекта.
3. При необходимости откройте историю сканирований SCA, чтобы убедиться, что последний запуск завершился успешно.
4. В истории сканирования откройте последний запуск по дате и проверьте:
    - число найденных зависимостей;
    - число найденных уязвимостей;
    - метаданные VCS, включая ветку и SHA коммита.

На этом этапе проект уже подключен к платформе, а первые результаты анализа готовы к разбору.

#Результат

Сценарий можно считать завершенным, если:

- открыть `Настройки -> VCS` и убедиться, что подключение GitLab сохранено;
- открыть `Настройки -> Проекты` и убедиться, что проект создан;
- открыть страницу проекта и проверить, что результаты SCA уже отображаются;
- при необходимости открыть историю сканирований SCA и убедиться, что последний запуск завершился успешно.

После этого проект готов к дальнейшему разбору зависимостей, уязвимостей и настройке политик безопасности.

#Что дальше

После первого анализа можно перейти к следующим задачам:

- проверить состав зависимостей;
- разобрать найденные уязвимости;
- настроить политики безопасности и регулярный анализ.

#Настроить базовый набор политик безопасности

#Контекст

После первого анализа в CodeScoring.SCA обычно быстро появляется много данных о зависимостях и уязвимостях. Базовый набор правил нужен, чтобы сразу отделить самые приоритетные случаи от общего

потока результатов и быстрее понять, на что реагировать в первую очередь.

Для VCS-проектов на этапе `source` для старта удобно собрать три отдельные политики:

- для уязвимостей с публичным эксплойтом и исправлением;
- для зависимостей с критичными уязвимостями;
- для слишком молодых компонентов, опубликованных менее месяца назад.

Эти правила помогают быстрее расставить приоритеты, не перегружая команду шумными срабатываниями и не включая блокировки на первом этапе.

Почему именно такой стартовый набор

OpenSSF рекомендует выстраивать работу с зависимостями вокруг понятной приоритизации риска, а OWASP подчёркивает важность раздельной обработки самых опасных и самых управляемых случаев. Исследования CodeScoring показывают, что на старте полезно не смешивать все сигналы в одно правило, а сразу разделять их по разным очередям: отдельно следить за уязвимостями, которые уже эксплуатируются и уже исправимы; отдельно — за зависимостями с критичными уязвимостями; отдельно — за слишком молодыми компонентами, которые требуют дополнительной проверки перед широким использованием.

#Что получится

После прохождения сценария в CodeScoring будет три активные политики для VCS-проектов на этапе `source`:

- политика для уязвимостей с публичным эксплойтом и исправлением на этапе `source`;
- политика для зависимостей с критичными уязвимостями на этапе `source`;
- политика для слишком молодых компонентов на этапе `source`.

После повторного SCA-анализа станет видно, какие из этих правил уже дают полезные срабатывания.

#Требования

Перед началом убедитесь, что есть:

- доступ в CodeScoring с ролью `Administrator` или `Security Manager`;
- хотя бы один подключенный VCS-проект, для которого можно повторно запустить SCA-анализ;
- понимание, будет ли набор применяться сразу ко всем проектам или сначала к одному пилотному проекту.

#Шаги

#Шаг 1. Создайте политику для уязвимостей с эксплойтом и исправлением

Такое правило помогает быстро выделить не просто уязвимости, а те случаи, где атака уже практична и при этом есть понятный путь к исправлению.

1. Перейдите в `Настройки -> Политики`.
2. Нажмите **Создать**.
3. Заполните контекст политики:
   - **Название** — например, `Есть эксплойт и исправление`;
   - **Этапы** — `source`;
   - **Уровень** — выберите подходящий уровень критичности;
   - **Активно** — включите;
   - **Блокер** — оставьте выключенным.
4. Если набор настраивается сначала для пилотного проекта, укажите его в поле **Проекты**. Если нужно распространить правило на все активные проекты, оставьте поля **Подразделения**, **Группы** и **Проекты** пустыми.
5. В верхней группе условий с логическим выражением **И** добавьте:
   - **Уязвимость имеет эксплойт**;
   - **Уязвимость имеет исправление**.
6. Нажмите **Создать**.

Почему блокер лучше не включать сразу

Для стартового набора полезнее сначала проверить, какие реальные срабатывания дает правило и сколько в нем шума. Так проще настроить рабочий процесс команды, не останавливая анализ и не ломая привычный поток работы.

После сохранения в системе появится правило, которое будет создавать алерты по наиболее приоритетным и уже исправимым уязвимостям в VCS-проектах.

#Шаг 2. Создайте политику для зависимостей с критичными уязвимостями

Отдельное правило для критичных уязвимостей помогает вынести самые тяжёлые случаи в самостоятельный поток обработки. Для стартового набора это полезно, потому что такие находки легче отдельно контролировать и не смешивать с остальными уровнями риска.

1. Откройте политику, созданную на предыдущем шаге.
2. Нажмите **Создать копию**.
3. Измените основные поля:
   - **Название** — например, `Критичная уязвимость в зависимости`;
   - **Этапы** — оставьте `source`;
   - **Активно** — оставьте включенным;

    - **Блокер** — оставьте выключенным.
4. Удалите прежние условия.
5. В верхней группе условий выберите логическое выражение **ИЛИ** и добавьте:
    - **Уровень угрозы CVSS2** = `критический`;
    - **Уровень угрозы CVSS3** = `критический`;
    - **Уровень угрозы CVSS4** = `критический`.
6. Нажмите **Создать**.

Теперь в наборе есть отдельное правило для зависимостей, у которых есть хотя бы одна критичная оценка по одной из версий CVSS.

#Шаг 3. Создайте информирующую политику для слишком молодых компонентов

Такое правило помогает вынести в отдельный поток срабатываний зависимости, которые были опубликованы совсем недавно. Для стартового набора это полезнее, чем сразу делать правило блокирующим: команда получает отдельный сигнал для дополнительной проверки новых пакетов и версий, не смешивая его ни с уязвимостями, ни с обычным плановым обновлением.

Почему это правило лучше оставить информирующим

Для стартового набора полезнее сначала увидеть такие компоненты как отдельный сигнал и понять, как часто они появляются в проектах. Если сделать такую политику блокирующей слишком рано, можно остановить рабочие процессы из-за обычных обновлений библиотек ещё до того, как команда согласует правила проверки новых версий.

1. В разделе `Настройки -> Политики` снова нажмите **Создать**.
2. Заполните контекст политики:
    - **Название** — например, `Компонент младше 30 дней`;
    - **Этапы** — `source`;
    - **Активно** — включите;
    - **Блокер** — оставьте выключенным.
3. Если набор настраивается постепенно, при необходимости укажите пилотный проект в поле **Проекты**.
4. В верхней группе условий выберите логическое выражение **ИЛИ** и добавьте:
    - **Возраст зависимости (в днях)** < `30`;
    - при необходимости — условие на отсутствие информации о возрасте зависимости.
5. Нажмите **Создать**.

Если нужно расширить правило или выбрать другой порог риска, список доступных критериев описан в настройке политик.

Что важно знать про возраст зависимости

Платформа определяет дату публикации зависимости для поддерживаемых экосистем. Перед тем как распространять такое

правило на все проекты, его лучше сначала проверить на пилотном проекте и убедиться, что критерий отрабатывает так, как ожидается на ваших пакетах и источниках.

После этого в отдельные алерты начнут попадать компоненты, которые появились слишком недавно и поэтому требуют дополнительной проверки перед широким использованием в проектах.

#Шаг 4. Повторно запустите анализ и проверьте первые срабатывания

Политики начинают работать во время анализа, поэтому после настройки важно сразу проверить, что хотя бы правила этапа `source` реально участвуют в процессе.

1. Откройте один из проектов, к которому должны применяться новые политики.
2. На странице проекта нажмите **Запустить SCA**.
3. Дождитесь завершения анализа.
4. Откройте раздел Алерты.
5. Проверьте, появились ли новые срабатывания по политикам:
    - для уязвимостей с эксплойтом и исправлением;
    - для прямых зависимостей с критичными уязвимостями;
    - для слишком молодых компонентов.

На этом этапе набор уже работает: все три правила начинают давать алерты после анализа и помогают разнести по разным типам самые важные случаи для разбора.

#Результат

Сценарий можно считать завершенным, если:

- в Настройки -> Политики сохранены три активные политики из этого набора;
- у всех трех правил указан этап `source` и заданы нужные критерии;
- правило для слишком молодых компонентов использует условие **Возраст зависимости (в днях) < 30** и остается неблокирующим;
- после повторного анализа в разделе Алерты можно проверить первые срабатывания по каждому типу политики.

После этого в платформе уже есть минимальный набор защитных правил, который помогает отсечь наиболее рискованные компоненты и быстрее разбирать действительно важные находки.

#Что дальше

- разобрать срабатывания в алертах;
- настроить исключения для допустимых случаев;
- подключить уведомления по политикам.

#Автоматически проверять безопасность компонентов в сборке и формировать отчет

#Контекст

Если композиционный анализ запускается только вручную, уязвимости и нарушения политик легко заметить слишком поздно, уже после неудачной сборки или выпуска. Опытные команды встраивают такую проверку прямо в сборочный конвейер, чтобы каждый запуск сразу показывал состав компонентов, найденные уязвимости и срабатывания политик.

Для этого используется консольный агент Johnny: он анализирует зависимости, сохраняет результаты в CodeScoring и сохраняет SBOM и отчеты.

В сценарии ниже для примера используется GitLab CI, но тот же принцип подходит и для других конвейеров, где агент Johnny можно запускать как часть сборки.

#Что получится

После прохождения сценария в `.gitlab-ci.yml` появится отдельное задание `sca`, которое:

- запускает агент Johnny на содержимом репозитория;
- сохраняет результаты в проект CodeScoring;
- формирует `bom.json` и `report.sarif` как артефакты сборки.

#Требования

Перед началом убедитесь, что есть:

- GitLab-репозиторий с настроенным GitLab CI;
- переменные GitLab CI `JOHNNY_API_URL` и `JOHNNY_API_TOKEN`;
- лицензия CodeScoring, в которой включен модуль **SCA**;
- доступ к CodeScoring, в котором можно автоматически создать проект для результатов агента Johnny через `--create-project` или использовать существующий;
- понимание, какие политики должны применяться к запуску агента на этапе `build`.

#Шаги

#Шаг 1. Подготовьте бинарный файл агента в среде сборки

Перед добавлением задания в конвейер важно убедиться, что сам агент доступен в той среде, где будет выполняться сборка.

1. Откройте страницу `[platform-url]/download/` в своей инсталляции CodeScoring.
2. При необходимости проверьте актуальную версию агента по адресу `[platform-url]/download/johnny_version`.
3. Скачайте подходящий исполняемый файл агента в среду сборки, например в `/usr/local/bin/johnny`.
4. Разрешите исполнение файла:

```
chmod +x /usr/local/bin/johnny
```

Если требуется более подробный пример именно для GitLab CI, его можно взять из руководства по добавлению агента в GitLab CI.

После этого в конвейере можно вызывать `johnny` как обычную исполняемую команду.

#Шаг 2. Добавьте отдельное задание `sca` в `.gitlab-ci.yml`

Главная задача этого шага — вынести проверку зависимостей в самостоятельное задание и сразу определить, какие файлы останутся после выполнения сборки.

Пример задания в

```
stages:
  - test

sca:
  stage: test

  script:
    - >
      johnny scan dir .
      --api_token $JOHNNY_API_TOKEN
      --api_url $JOHNNY_API_URL
      --project "billing-service-cli"
      --save-results
      --create-project
      --stage build
      --localization ru
      --format "coloredtable,sarif>>report.sarif"
      --ignore .git
```

```
  artifacts:
    paths:
      - bom.json
      - report.sarif
    when: always
    expire_in: 1 week
```

Что важно в этом задании:

- `scan dir .` запускает анализ директории репозитория;
- `--project`, `--save-results` и `--create-project` сохраняют результаты в CodeScoring;
- `--stage build` применяет политики, относящиеся к запуску через агент Johnny;
- `--format "coloredtable,sarif>>report.sarif"` оставляет консольный вывод и одновременно пишет отчет в формате `sarif`;
- `bom.json` формируется агентом автоматически и сохраняется как артефакт вместе с `report.sarif`.
- полный список флагов и режимов запуска собран в руководстве по запуску агента Johnny.

После этого в конвейере появляется отдельное задание, которое не только запускает анализ, но и оставляет файлы, пригодные для отчётности и дальнейшей автоматизации.

#Шаг 3. Запустите конвейер с новым заданием

Теперь важно добиться первого реального прогона, чтобы проверить сразу три вещи: команда выполняется, артефакты создаются, а результаты отправляются в платформу.

1. Сохраните изменения в `.gitlab-ci.yml`.
2. Зафиксируйте их в Git и отправьте ветку в GitLab.
3. Дождитесь запуска задания `sca`.

Как трактовать код возврата агента

Код возврата 1 означает, что агент Johnny завершил анализ и нашёл проблемы, соответствующие политикам безопасности. Это не аварийное завершение. Подробно коды возврата и форматы отчетов разобраны в руководстве по запуску агента Johnny.

После первого запуска станет понятно, хватает ли текущих переменных и прав, чтобы задание реально дошло до анализа и выгрузки результатов.

#Шаг 4. Посмотрите, как выглядит типичный запуск агента Johnny

Ниже — интерактивное демо с примером задания и типичным выводом агента.

После такого запуска в сборке уже остаются SBOM и машинно-читаемый отчет в формате `sarif`.

#Шаг 5. Проверьте, что результаты действительно пригодны для работы дальше

Автоматическая проверка имеет смысл только тогда, когда ее результат можно сразу использовать в сборке, платформе и дальнейшей обработке.

Проверьте, что после выполнения задания:

1. В артефактах GitLab доступны файлы:
   - `bom.json`;
   - `report.sarif`.
2. В CodeScoring появился проект `billing-service-cli`, если он не существовал раньше.
3. В этом проекте сохранены результаты последнего запуска.

Зачем нужны оба файла

`bom.json` удобно использовать как SBOM-артефакт сборки, а `report.sarif` — как машинно-читаемый отчет для CI/CD-инструментов, систем безопасной разработки и последующей автоматизации проверок.

На этом этапе сборка уже не просто выполняет проверку, а оставляет после себя понятный набор данных для анализа и отчетности.

#Результат

Сценарий можно считать завершенным, если:

- в `.gitlab-ci.yml` появилось отдельное задание `sca` с запуском агента Johnny;
- задание формирует `bom.json` и `report.sarif` как артефакты;
- агент Johnny сохраняет результаты в проект CodeScoring через `--save-results`;
- команда понимает, что код возврата 1 означает найденные нарушения политик, а не сбой агента.

После этого безопасность компонентов проверяется автоматически на этапе сборки, а отчеты остаются доступны и в CI, и в платформе.

#Что дальше

- добавить дополнительные форматы выгрузки результатов;
- уточнить параметры запуска агента Johnny и состав флагов;
- масштабировать сценарий на другие проекты.

#Составить и выгрузить ППК (SBOM) с учетом требований ФСТЭК

#Контекст

Для сертификации и подготовки сопроводительных материалов недостаточно просто выгрузить обычный SBOM. Нужен перечень программных компонентов в машиночитаемой форме, который учитывает дополнительные требования ФСТЭК и при этом отражает полный состав проекта, включая зависимости, которые не всегда раскрываются в манифестах по умолчанию.

CodeScoring поддерживает выгрузку SBOM в расширенных форматах `CycloneDX v1.6 Ext JSON` и `CycloneDX v1.7 Ext JSON`, адаптированных под такие требования. Чтобы такой файл получился полезным, сначала важно собрать полный состав компонентов, а затем разметить в проекте свойства зависимостей, которые не определяются автоматически.

В примере ниже для локального запуска используется Python-проект и `pip`, потому что на этой связке проще наглядно показать, зачем для полного ППК иногда нужно разрешение зависимостей в локальной среде и явное указание пути к пакетному менеджеру. Тот же подход применяется и для других экосистем, где для полного инвентаря нужно разрешение зависимостей в окружении.

#Что получится

После прохождения сценария:

- в CodeScoring появится проект с полным составом компонентов, сохраненным после локального запуска агента;
- для ключевых зависимостей будут заполнены свойства, важные для выгрузки ППК;
- из интерфейса можно будет скачать `CycloneDX v1.6 Ext JSON` или `CycloneDX v1.7 Ext JSON` с учетом требований ФСТЭК.

#Требования

Перед началом убедитесь, что есть:

- доступ к on-premise инсталляции CodeScoring;
- лицензия CodeScoring, в которой включен модуль **SCA**;
- API-токен платформы для локального запуска агента;
- локальная копия проекта, для которого нужно сформировать ППК;
- установленный в локальной среде пакетный менеджер и известный путь к нему, например `/usr/local/bin/pip3`;
- права на просмотр проекта и выгрузку SBOM из интерфейса.

#Шаги

#Шаг 1. Подготовьте агент Johnny для локального запуска

Сначала нужно убедиться, что локальный запуск можно выполнить без дополнительных донастроек в последний момент.

1. Откройте страницу `[platform-url]/download/` в своей инсталляции CodeScoring.
2. При необходимости проверьте актуальную версию по адресу `[platform-url]/download/johnny_version`.
3. Скачайте исполняемый файл агента под свою систему.
4. Сделайте файл исполняемым:

```
chmod +x ./johnny
```

После этого агент можно использовать для локальной инвентаризации проекта и сохранения результатов в платформу.

#Шаг 2. Сначала получите полный состав компонентов локально

Чтобы получить полный граф зависимостей с учетом транзитивных, нужен либо lock-файл соответствующего пакетного менеджера, либо запуск с разрешением зависимостей в окружении сборки. В случае `pip` отдельный lock-файл обычно не используется, поэтому в этом примере показан второй вариант с явным указанием пути к пакетному менеджеру.

Пример локального запуска с сохранением результатов в CodeScoring

```
./johnny scan dir . \
  --api_token <api_token> \
  --api_url <api_url> \
  --project "ppk-fstec-demo" \
  --save-results \
  --create-project \
  --localization ru \
  --pip-resolve \
  --pip-path /usr/local/bin/pip3 \
  --bom-path bom-local.json
```

Что важно в этой команде:

- `--pip-resolve` включает разрешение зависимостей в окружении;
- `--pip-path` явно указывает, какой `pip` нужно использовать для локального запуска;
- `--save-results` и `--create-project` сохраняют результаты в отдельный CLI-проект CodeScoring;
- `--bom-path bom-local.json` оставляет локальную копию SBOM рядом с проектом;

- полный список флагов и режимов запуска собран в руководстве по запуску агента Johnny.

Когда

`resolve` действительно нужен Для некоторых экосистем пакетные менеджеры по умолчанию не раскрывают транзитивные зависимости в манифестах. В таких случаях CodeScoring рекомендует разрешение зависимостей в окружении. Если lock-файл уже существует, агент использует его и отдельный `resolve` не выполняет.

Ниже показано короткое интерактивное демо. Оно показывает разницу между запуском только по манифесту и запуском с `resolve`. Для `pip` такой режим стоит использовать только в изолированном окружении проекта: иначе в результат могут попасть лишние пакеты из локальной среды, которые не относятся к сканируемому приложению.

После такого запуска в платформе появляется CLI-проект с полным составом компонентов, а локально сохраняется `bom-local.json`.

#Шаг 3. Разметьте свойства зависимостей перед выгрузкой

Расширенный SBOM для ФСТЭК отличается от обычного тем, что в него добавляются дополнительные свойства компонентов. Часть из них требует ручной разметки в проекте.

1. Откройте созданный проект в CodeScoring.
2. Перейдите к таблице зависимостей.
3. Нажмите **Настроить зависимости**.
4. Для компонентов, которые нужно отразить в ППК более точно, заполните необходимые поля:
   - **VCS** — ссылка на репозиторий или архив с исходным кодом;
   - **Поверхность атаки** — Да, Косвенно или Нет;
   - **Функция безопасности** — Да, Косвенно или Нет;
   - **Кем предоставлено** — если компонент заимствован из другого продукта.
5. Сохраните изменения.

Что не заполняется автоматически

Поля, соответствующие поверхности атаки, функции безопасности и источнику заимствования, требуют экспертной оценки и не должны считаться полностью автоматическими. Именно поэтому их лучше проверить и заполнить перед выгрузкой.

Эти значения относятся только к текущему проекту и учитываются в следующих выгрузках из него.

После этого выбранные значения будут учитываться при последующих выгрузках SBOM из проекта.

#Шаг 4. Выгрузите ППК в расширенном формате CycloneDX

Теперь можно получить машиночитаемый файл, который уже учитывает разметку из проекта.

1. На странице проекта нажмите **Скачать SBOM**.
2. В списке форматов выберите:
    - `CycloneDX v1.6 Ext JSON`, или
    - `CycloneDX v1.7 Ext JSON`.
3. При необходимости задайте имя файла.
4. Подтвердите выгрузку.

Почему нужен именно формат Ext

Для требований ФСТЭК недостаточно обычного CycloneDX JSON. Нужен расширенный формат `Ext`, в котором дополнительные свойства компонентов включаются в `properties`.

После этого на локальной машине появится файл ППК в формате, который адаптирован под дополнительные требования ФСТЭК.

#Шаг 5. Проверьте содержимое выгруженного файла

Финальная проверка нужна, чтобы убедиться, что выгружен не просто SBOM, а именно тот файл, который содержит добавленные свойства и пригоден для дальнейшей передачи и проверки.

1. Откройте выгруженный JSON-файл в редакторе или просмотрщике JSON.
2. Найдите один из компонентов, который был размечен на предыдущем шаге.
3. Убедитесь, что:
    - у компонента есть блок `properties`;
    - в нем присутствуют значения для свойств, связанных с поверхностью атаки, функцией безопасности и источником заимствования;
    - ссылка на репозиторий или архив исходного кода попала в `externalReferences`.

После такой проверки можно быть уверенным, что файл содержит не только базовый перечень компонентов, но и дополнительную разметку, важную для ППК.

#Результат

Сценарий можно считать завершенным, если:

- локальный запуск агента сформировал полный состав компонентов и сохранил результаты в проект CodeScoring;
- для нужных зависимостей в проекте заполнены свойства, влияющие на выгрузку ППК;

- из интерфейса скачан `CycloneDX v1.6 Ext JSON` или `CycloneDX v1.7 Ext JSON`;
- в выгруженном файле видны дополнительные свойства компонентов и ссылка на исходный код там, где она была указана.

После этого ППК можно использовать как машиночитаемый результат для внутренней проверки и дальнейшей подготовки материалов.

#Что дальше

- уточнить параметры разрешения зависимостей для других экосистем;
- донастроить свойства зависимостей для следующих выгрузок;
- проверить файл через SBOM Checker ИСП РАН.

#Блокировать вредоносные компоненты через OSA Proxy

#Контекст

Если небезопасный пакет сначала попадает во внутренний менеджер репозиториев, а проверка срабатывает только потом, команде приходится отдельно разбирать инцидент и чистить кэш. Надежнее отсеивать такие компоненты ещё на входе в периметр, до того как они станут доступны разработчикам и сборкам.

OSA Proxy — это прокси-сервис CodeScoring, который перехватывает запросы пакетных менеджеров к удалённым репозиториям, проверяет компоненты и при необходимости блокирует их. В этом сценарии он разворачивается перед JFrog Artifactory, чтобы внутренний PyPI-репозиторий получал только те пакеты, которые прошли проверку по политикам безопасности.

#Что получится

После прохождения сценария:

- OSA Proxy будет развернут и настроен для PyPI в режиме `strict`;
- JFrog Artifactory начнет получать пакеты через OSA Proxy, а не напрямую из внешнего источника;
- блокирующая политика на этапе `proxy` будет останавливать вредоносные компоненты по условию **Зависимость опасна**;
- результат проверки можно будет увидеть в разделе `OSA -> Запросы`.

#Требования

Перед началом убедитесь, что есть:

- хост или стенд, на котором можно развернуть OSA Proxy;

- доступ к Docker или Kubernetes и образу OSA Proxy для выбранного способа установки;
- доступ к `application.yml` OSA Proxy и возможность перезапустить сервис после изменения конфигурации;
- JFrog Artifactory с удалённым PyPI-репозиторием, который можно перенастроить;
- лицензия CodeScoring, в которой включен модуль **OSA**;
- доступ в CodeScoring с правами на создание политик и просмотр данных OSA;
- тестовый пакетный поток, на котором можно проверить работу схемы до её включения для всех команд.

#Шаги

#Шаг 1. Разверните OSA Proxy

Если сервис ещё не установлен, сначала нужно поднять его в отдельном окружении. Для первой проверки достаточно варианта с Docker.

Пример запуска OSA Proxy в Docker

```
docker run -d \
-p 8080:8080 \
-e SPRING_CONFIG_ADDITIONAL_LOCATION=file:/app/config/ \
-v /path/to/your/config/application.yml:/app/config/
application.yml \
--name cs-proxy \
<registry-address>/cs-proxy:<tag>
```

Если используется Kubernetes, удобнее сразу развернуть сервис через Helm Chart. Оба варианта установки описаны в документации по развертыванию OSA Proxy.

После установки сервис должен быть доступен по своему URL и готов к загрузке конфигурации.

#Шаг 2. Настройте OSA Proxy для PyPI и включите строгий режим

Теперь нужно включить такой режим работы, при котором непросканированные и запрещённые компоненты не будут проходить дальше во внутренний менеджер репозиториев.

Пример конфигурации OSA Proxy для PyPI

```
codescoring:
  host: https://codescoring.example.com
  token: <token>
  work-mode: strict
  enable-status-line: true
```

```
  block-status-code: 403

pypi:
  enabled: true
  repository:
    - name: internet-pypi
      scan-manifest: true
      scan-package: true
      url-encoded-config: true
      registry: https://pypi.org
      packages-registry: https://files.pythonhosted.org
```

Что важно в этой конфигурации:

- `work-mode: strict` запрещает загрузку непросканированных компонентов;
- `scan-manifest: true` позволяет убирать запрещённые версии из ответа Simple API;
- `scan-package: true` включает проверку архивов пакетов;
- `enable-status-line: true` помогает быстрее понимать причину блокировки при диагностике;
- `url-encoded-config: true` нужен для сценария, где OSA Proxy получает контекст внутреннего репозитория через URL.

После сохранения `application.yml` перезапустите OSA Proxy.

Строгий режим лучше включать поэтапно

Новые запросы начнут фильтроваться сразу, но пакеты, которые уже успели попасть в кэш JFrog Artifactory раньше, сами не исчезнут. Поэтому безопаснее сначала проверить схему на отдельном удалённом репозитории и только потом переводить основной поток в `strict`.

#Шаг 3. Поставьте OSA Proxy перед JFrog Artifactory

Теперь нужно сделать так, чтобы удалённый PyPI-репозиторий в JFrog Artifactory больше не обращался напрямую во внешний источник и получал пакеты только через OSA Proxy.

1. Откройте в JFrog Artifactory раздел **Administration -> Repositories -> Remote**.

2. Найдите удалённый PyPI-репозиторий, через который команды получают внешние Python-пакеты.

3. В поле **URL** укажите адрес OSA Proxy вместо прямого адреса PyPI.

4. Если удалённый репозиторий в JFrog Artifactory обращается напрямую во внешний PyPI, передайте OSA Proxy контекст внутреннего репозитория через Base64-параметры и используйте URL такого вида:

```
https://osaproxy.example.com/internet-pypi/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL3B5cGkub3JnL3NpbXBsZ
SIsInJlcG9OYW1lIjoiaW50ZXJuZXQtcHlwaSJ9
```

В этом примере в строке Base64 закодирован такой JSON:

```
{"repoManagerHost":"https://pypi.org/simple","repoName":"
internet-pypi"}
```

OSA Proxy использует эти параметры, чтобы понять, к какому внутреннему репозиторию относится запрос и какие политики к нему применять.

5. В дополнительных настройках JFrog Artifactory включите флаг **Bypass HEAD requests**.
6. Сохраните изменения.

После этого новые запросы к внешнему PyPI начнут проходить через OSA Proxy.

#Шаг 4. Создайте блокирующую политику для вредоносных компонентов

Теперь нужно задать правило, которое будет срабатывать на этапе `proxy` и запрещать компонент, если он попадает под критерий **Зависимость опасна**.

1. Перейдите в `Настройки -> Политики`.
2. Нажмите **Создать**.
3. Заполните основные поля:
   - **Название** — например, `Опасная зависимость на этапе proxy`;
   - **Этапы** — `proxy`;
   - **Компоненты OSA** — `Пакеты`;
   - **Активно** — включите;
   - **Блокер** — включите.
4. Если правило должно действовать только для одного канала поставки, выберите нужный репозиторий. Если защита нужна для всех прокси-репозиториев, оставьте это поле пустым.
5. Добавьте условие:
   - **Зависимость опасна**.
6. Нажмите **Создать**.

Блокирующую политику лучше сначала проверять на пилотном репозитории

На этапе `proxy` блокер влияет не на отчет, а на сам факт получения компонента. Если правило сразу повесить на основной репозиторий без пилотной проверки, можно внезапно остановить привычные запросы команд разработки и сборок.

После сохранения OSA Proxy сможет применять это правило к новым запросам на получение пакетов.

#Шаг 5. Проверьте, что клиент больше не видит запрещённые версии

В этой проверке удобнее смотреть не на установку конкретного пакета, а на то, какой ответ получает клиент от индекса пакетов после фильтрации версий.

В этом примере клиент запрашивает список доступных версий `requests` через OSA Proxy и получает уже отфильтрованный ответ Simple API. Если опасной версии нет в ответе, пакетный менеджер не сможет выбрать её для установки.

#Шаг 6. Проверьте результат в `OSA -> Запросы`

Последний шаг нужен, чтобы убедиться, что платформа зафиксировала сам запрос и его статус, а не только изменила ответ для пакетного менеджера.

1. Перейдите в раздел `OSA -> Запросы`.
2. Откройте вкладку **Пакеты**.
3. Найдите запрос, выполненный через пилотный PyPI-репозиторий.
4. Проверьте, что в списке видны:
    - название пакета;
    - режим работы;
    - статус блокировки;
    - дата запроса;
    - инициатор запроса.

После этого уже можно проверить не только поведение клиента, но и то, что событие сохранилось в самой платформе.

#Результат

Сценарий можно считать завершённым, если:

- OSA Proxy развернут и работает в режиме `strict` для PyPI-источника;
- JFrog Artifactory получает пакеты через OSA Proxy;
- в CodeScoring создана активная блокирующая политика на этапе `proxy` с условием **Зависимость опасна**;
- клиент получает отфильтрованный ответ индекса, а в `OSA -> Запросы` видно зафиксированный запрос и его статус.

После этого вредоносные и запрещённые компоненты можно останавливать ещё до того, как они окажутся во внутреннем репозитории и станут доступны командам разработки.

#Что дальше

- разобрать детали запросов и статусы блокировки в OSA;
- уточнить режимы работы и параметры блокировки OSA Proxy;
- масштабировать схему на другие пакетные менеджеры.

#Найти актуальные секреты в коде проекта

#Контекст

В этом сценарии используется VCS-проект с модулем Secrets. Ниже показан короткий путь: как запустить анализ, перейти к списку находок и с помощью встроенной ML-модели быстрее отобрать секреты, которые с большей вероятностью требуют внимания.

#Требования

Перед началом убедитесь, что есть:

- лицензия CodeScoring, в которой включен модуль **Secrets**;
- VCS-проект, в котором можно запустить анализ секретов;
- доступ к проекту и к результатам анализа секретов в CodeScoring.

При просмотре видео полезно обратить внимание на поле **Вероятность TP** и на то, как модель помогает отфильтровать ложные срабатывания. Это позволяет быстрее перейти к находкам, которые действительно стоит проверить в первую очередь.

#Что дальше

- разобрать найденные секреты и их статусы;
- настроить параметры поиска секретов для VCS-проекта;
- включить регулярный запуск анализа секретов.

#Codescoring On-premise Changelog

#[2026.20.2] - 2026-06-02

#Изменено

- Обновлен образ Frontend `nginx` с `nginx:1.29.5-alpine` до `nginx:1.31.1-alpine`
- Secrets Обновлена базовая версия Go для Gitleaks с `golang:1.25.4` до `golang:1.26.3`
- Обновлен базовый образ Backend с `alpine:3.23.3` до `alpine:3.23.4`
- Обновлена базовая версия Go для сервиса обработки политик с `golang:1.25.1` до `golang:1.25.10`

- TQI Обновлен компонентный состав JS-библиотек JSCPD
- Обновлен базовый образ Index Proxy с `alpine:3.23.3` до `alpine:3.23.4`
- OSA Обновлен базовый образ OSA API с `alpine:3.23.3` до `alpine:3.23.4`
- Обновлен образ Redis с `7.4.6` до `7.4.9-alpine3.21`

#Исправлено

- Исправлено выполнение действий в алертах политик при указании определенного проекта в настройках
- Secrets Исправлена работа движка TruffleHog при работе с авторизацией в VCS

#[2026.20.1] - 2026-05-25

#Изменено

- SCA Изменены названия кнопок запуска SCA-сканирования

#Исправлено

- SCA Исправлено падение страницы уязвимостей при выборе нескольких фильтров
- Исправлена обработка политик для сервиса OSA Proxy
- TQI Исправлена ошибка при запуске анализа авторов нового проекта с первого коммита
- OSA Исправлено поведение, при котором OSA API не учитывал недавно обновлённые или созданные политики
- Исправлено отображение технологий проектов
- Снижена нагрузка на базу данных при миграции на новую модель версий проектов
- Исправлено сохранение шаблона при создании действия по отправке алертов в политиках

#[2026.20.0] - 14-05-2026

Минимальная версия PostgreSQL

Для корректной работы CodeScoring начиная с версии 2026.20.0 необходимо использовать PostgreSQL версии 15 или выше. Без обновления PostgreSQL корректность работы инсталляции не гарантируется. Процедура обновления описана в документации.

Сроки миграции

Миграция при обновлении на версию 2026.20.0 может занять дольше, чем обычно. Это ожидаемое поведение и вызвано необходимостью перевести проекты на новую модель управления версиями.

#Добавлено

- Data Добавлен источник уязвимостей CISA KEV Catalog
- SCA Добавлена информация из источника CISA KEV на страницу уязвимости
- SCA Добавлена возможность управления VEX-статусами уязвимостей
- Secrets Добавлена поддержка сканирования секретов движком TruffleHog
- Secrets Добавлен список движков, с помощью которых был найден секрет, на карточку секрета
- SCA OSA Добавлена обработка информации о слоях контейнерных образов от агента Johnny. Для контейнерных образов добавлены поля названия и версии базового образа
- SCA Добавлена информация об отсканированном образе в SCA-проекте
- SCA Добавлена страница образа, связанного со SCA-проектом
- SCA OSA Добавлена страница слоёв в разделе OSA, а также таблицы слоёв на странице отсканированного образа в проекте и странице контейнерного образа OSA
- OSA Добавлены команды создания слоёв контейнерных образов
- Добавлена проверка на наличие связанных политик и игноров политик при удалении проекта. При наличии связанных объектов удаление будет запрещено
- SCA Добавлены столбцы и фильтры по данным EPSS и SSVC в таблицы уязвимостей
- SCA Добавлена подсказка об источнике описания уязвимости на страницу уязвимости
- Добавлена колонка с информацией о времени последнего обновления в таблицу алертов. Значение изменяется при повторном срабатывании алерта
- Добавлена политика "Зависимость является ПО-вымогателем"
- Добавлена политика "Зависимость отозвана"
- OSA Добавлена поддержка ранее недоступных политик в модуле CodeScoring.OSA
- SCA Реализовано управление версиями в настройках проекта
- SCA Добавлен фильтр по зависимостям на странице уязвимостей
- SCA Добавлено поле `Inner source` ("Внутренний источник") для зависимостей
- SCA Добавлена фильтрация пакетов SBOM по полю `Inner source`
- SCA OSA Добавлен статус отзыва на страницы пакета и страницы зависимости
- SCA Добавлена возможность выбора GitFlic в качестве типа VCS
- Добавлены фильтры "Название", "Имя контактного лица" и "Email контактного лица" на странице владельцев проектов
- OSA API Добавлена возможность запускать эндпоинт `api/osa_components/packages/{id}/scan/` по идентификатору пакета
- SCA Добавлена выгрузка `source-distribution` в раздел `externalReferences` при формировании SBOM

- SCA Добавлен вывод ссылки на результат сканирования в раздел `metadata.tools.components.externalReferences` при формировании SBOM
- SCA Добавлены параметры `percent` и `percentile` для EPSS при экспорте SBOM в `vulnerabilities.ratings`
- SCA Добавлены параметры `is_dangerous` и `is_protestware` при экспорте SBOM в `components.properties`
- SCA Добавлено логирование невалидного токена доступа VCS-проекта в аудит лог и логи контейнера `tasks`
- Добавлена возможность убрать приоритет задачи на форме создания и редактирования задачи при редактировании политики
- SCA Добавлен экспорт SBOM для групп проектов
- Secrets Добавлен множественный выбор для фильтра "ID правила" во вкладке "Секреты" проекта

#Изменено

- TQI API Изменены права доступа, необходимые для получения ключа Svace через API. Теперь ключ может получить пользователь с ролью `Administrator`, `Security Manager` или `User` с правами `Owner` или `Developer` хотя бы в одном проекте
- SCA Изменено отображение пунктов EPSS и SSVC: они всегда показываются в блоке с оценкой уязвимости и в информации из фида CVE.ORG независимо от наличия данных
- SCA Изменена логика отображения отзыва уязвимости: в верхнем блоке дата отзыва отображается только если уязвимость отозвана во всех источниках. В источнике уязвимость считается отозванной, если там есть хотя бы одна отозванная запись
- Secrets Реализована дедупликация находок от разных движков секретов
- Изменены системные требования: минимальная версия PostgreSQL теперь 15+
- Изменено название условия политики "Список авторов зависимостей" на "Список email авторов зависимостей" в форме создания и редактирования политики
- OSA Изменено минимальное допустимое значение таймаутов при настройке реестра контейнеров на 1
- Ускорена загрузка данных дашборда с топ-5 политиками и количеством алертов
- Ускорена проверка политик безопасности в сервисе OSA Proxy

#Исправлено

- Data Исправлены некорректные даты публикации, отображавшиеся как 1970 год, для пакетов из NuGet и Packagist
- Data Исправлен сбор информации об авторах пакетов из crates.io
- Оптимизирована периодическая задача `check_vcs_availability`: убран лишний запрос к базе данных
- SCA Исправлена отправка уведомлений по электронной почте для SCA-анализа, если проект исключён из SCA-анализа

- SCA Исправлена выгрузка SBOM без лицензий: вручную изменённые лицензии больше не попадают в отчёт, если лицензии исключены из выгрузки
- SCA Исправлена выгрузка игноров в PDF-отчёт: чекбокс "Только активные игноры" переименован в "Только эффективные игноры" и теперь по умолчанию исключает игноры неактивных политик
- Secrets Исправлена работа фильтра "ID правила" при выгрузке PDF-отчёта по секретам
- API Исправлена Swagger-схема эндпоинтов для выгрузок PDF-отчётов
- Исправлена работа политики "Возраст уязвимости пустой"
- Исправлена работа политики "Версия зависимости пустая"
- OSA Исправлена проверка политик OSA для уязвимостей, которые отсутствовали в базе CodeScoring на момент сканирования
- SCA Исправлен счётчик уязвимых функций при сканировании из интерфейса
- Исправлена взаимная блокировка при обновлении списка репозиториев
- SCA Исправлены отображаемые названия и версии зависимостей в графах зависимостей проектов и на страницах детальной информации зависимостей
- OSA Исправлено дублирование тегов образов при сканировании посредством OSA
- API Исправлен ответ при ошибке валидации при добавлении проекта в группу: ответ теперь соответствует контракту API
- SCA Оптимизирована загрузка страницы истории сканирования проектов
- SCA Оптимизирована маркировка уязвимых узлов на графе зависимостей
- Исправлены права доступа для настроек групп проектов
- Исправлена ошибка логина по OIDC
- API Доступ к части эндпоинтов приведён в соответствие с ролевой моделью
- Secrets Исправлен некорректный статус пользовательской модели секретов после её удаления
- SCA Исправлена сортировка в SCA-проектах

#[2026.11.2] - 2026-04-03

#Добавлено

- OSA Добавлена переменная окружения `DATABASE_CONNECTION_POOL_TIMEOUT` в OSA API для настройки таймаута получения соединения из пула. Значение по умолчанию — `30` секунд
- OSA Расширены метрики пула соединений с базой данных в OSA API: добавлены `requests_queued`, `requests_wait_ms`, `requests_errors`, `connections_errors`, `connections_lost` и другие

#Изменено

- OSA Разделено ожидание результата сканирования и удержание соединений с базой данных в OSA API, поэтому соединения больше не блокируются во время сканирования

#Исправлено

- SCA Исправлено чтение лицензий компонентов в SBOM, сгенерированных сторонними инструментами
- OSA Исправлена проверка доступности Redis в OSA API
- OSA Исправлено использование таймаута сканирования SCAN_WAIT_TIMEOUT при ожидании результата сканирования в OSA API
- SCA Исправлена ошибка 403 при повторном сканировании SBOM в истории CLI-проекта для пользователей с ролью USER
- Исправлена ошибка, из-за которой пользователю с ролью user был недоступен раздел уязвимостей

#[2026.11.1] - 2026-03-24

#Исправлено

- Исправлена ошибка недоступности Index API для инсталляций, работающих через прокси
- Исправлен формат сообщения аудит-лога об обновлении уязвимостей, если предыдущая задача обновления ещё не завершилась
- Исправлено сохранение метрик SSVC при получении уязвимостей из Index API

#[2026.11.0] – 2026-03-13

#Добавлено

- Добавлена индивидуальная страница алерта
- Добавлена поддержка Kaiten в качестве менеджера задач
- SCA Добавлено отображение данных EPSS и соответствующие политики безопасности на страницу уязвимости
- SCA Добавлено отображение оценок CVSSv4 и соответствующие политики безопасности на страницу уязвимости
- SCA Добавлено отображение оценки SSVCv2.0.3 и соответствующие политики безопасности
- SCA Добавлено отображение категорий протестного ПО и соответствующие политики безопасности на страницу уязвимости
- SCA Добавлено отображение дат публикации, отзыва и обновления уязвимости в конкретном фиде
- SCA Добавлена возможность копирования идентификаторов уязвимостей на странице уязвимости
- SCA Добавлена подсказка для верхнего блока описания на странице уязвимости

- SCA Добавлена подсказка к полю "Сканировать с хэшами" в настройках проекта
- SCA Добавлено поле "Заметка игнора" в таблицу игноров политик PDF-отчёта
- SCA Добавлен столбец "Заметка" в CSV-файл, экспортируемый на странице алертов
- SCA Добавлена возможность сортировки по столбцу "Найдено" в разделе "Затронутые зависимости" при просмотре уязвимости
- SCA Добавлено отображение количества успешных SCA-анализов в группе проектов
- SCA Добавлено выделение полей "VCS" и "Лицензии" на странице настройки зависимостей проекта при ручных изменениях
- SCA Добавлен фильтр по хэшу образа на странице истории сканирований CLI-проекта
- SCA Добавлена колонка с лицензиями в проекте на странице зависимости
- SCA Добавлено поле "Статус блокировки" для алертов в PDF и CSV отчётах
- SCA Добавлена колонка со ссылкой на страницу алерта в таблице алертов на странице SCA проекта
- SCA Добавлена связь между фильтрами "Окружение" и "Технология" на странице Зависимости
- OSA Добавлена связь между фильтрами "Репозиторий" и "Менеджер репозиториев" на страницах Запросы и пакеты
- OSA Добавлен фильтр "Доступно" на страницу "Менеджеры репозиториев"
- OSA Добавлен фильтр по пользователю CodeScoring на странице запросов OSA
- OSA Добавлено поле для настройки таймаута проверки наличия образа в реестре перед анализом
- OSA Добавлена дата последнего сканирования на страницу контейнера
- OSA Добавлено логирование успешного завершения и статистики загрузки контейнерных образов в аудит лог
- OSA Добавлена простановка статуса “Неактуальный” для образа при ошибке отсутствия образа и очистка неактуальных образов в `periodic_cleanup_osa_components`
- OSA Добавлена переменная окружения `USE_JSON_LOG_FORMAT` в OSA API, управляющая форматом логов (по умолчанию `true`)
- API Добавлен фильтр `projects_isnull` в `/api/policies/`, `/api/policy_alerts_v2/`, `/api/settings/policy_ignores/`
- Запуск обновления лицензий и уязвимостей из Index API теперь сопровождается записью в аудит-логе

#Изменено

- Secrets Обновлена модель для оценки истинности секретов
- Secrets Изменена методика подбора выборки дообучения
- OSA Изменена логика загрузки контейнерных образов на основе явного указания необходимых mediaType
- SCA Переименовано поле в фильтрах PDF-отчёта

- SCA Изменено поле нумерации на идентификатор алерта в таблице "Игноры политики" PDF-отчёта
- SCA Изменён принцип отображения оценок, метрик и уровней опасности CVSS: используются данные из источника с максимальной оценкой для каждой версии CVSS
- SCA Изменён принцип отображения дат публикации, отзыва и обновления в описании уязвимости: показываются самые ранние даты публикации и отзыва и самая поздняя дата обновления из всех источников. Политики с условиями на эти свойства будут обновлены при следующем сканировании проекта, образа или пакета
- SCA Изменён приоритет провайдеров для метаданных уязвимости: CVE.ORG, GHSA, Kaspersky, BDU, затем остальные по алфавиту
- SCA Изменён цвет "критического" уровня опасности CVSS
- SCA Изменён текст поля "Импакт" на странице уязвимости для фида Касперского
- SCA Доработано отображение критичности на странице уязвимости
- SCA OSA Упрощено редактирование параметров SBOM в SCA проектах и контейнерных образах: настройки открываются сразу в режиме редактирования, сохранение выполняется одной кнопкой
- API Изменён формат поля `impacts` уязвимостей в соответствующих методах API на `pk-name`
- Изменена логика политики, связанной с датой релиза: учитывается только дата без времени, время не отображается в UI
- Обновлены базовые образы сервисов backend / tasks-*, judge, index-proxy, osa и frontend (Alpine 3.23.3, nginx 1.29.5)
- Убраны Scopes по умолчанию при настройке OIDC

#Исправлено

- OSA Исправлена ошибка при сканировании образов с отсутствующими алертами
- OSA Оптимизирована работа политик безопасности для запросов с количеством пакетов больше 100
- OSA Оптимизирована работа OSA API: стандартизированы проверки доступа и контекста запросов, улучшено кэширование токенов и ключей
- OSA Исправлен рассинхрон между сервисом политик и OSA при проверке политик зависимостей
- SCA Исправлено масштабирование графа зависимостей
- SCA Исправлено сохранение результатов достижимости уязвимостей для существующего проекта
- SCA Исправлены имена источников уязвимостей в выгрузке SBOM
- SCA Исправлено дублирование записей в таблице "Затронутые образы" на странице уязвимости
- SCA Исправлен подсчёт количества уязвимостей с отсутствующим уровнем опасности CVSS3 на странице SCA проектов и групп проектов
- SCA Исправлено отображение количества зависимостей, уязвимостей и алертов в группе проектов при отсутствии SCA-анализа

- SCA Исправлено отсутствие свойства "language" у зависимостей при выгрузке SBOM в форматах 1.6_ext или 1.7_ext
- SCA Исправлено взаимодействие с OSS Index в ходе SCA
- SCA Исправлено обновление поля "исправленная версия" для уязвимостей, у которых была отозвана последняя исправленная версия
- SCA Исправлен подсчёт количества уязвимостей у зависимости в истории сканирований SCA
- SCA Оптимизирована загрузка страницы зависимостей
- TQI Исправлена ошибка подсчёта авторов в аудит-отчётах при 404 от репозиториев
- TQI Оптимизирован алгоритм сравнения авторов, снижено количество ложных отрицаний
- Secrets Исправлен баг сериализации секретов при анализе проекта, подключенного через интеграцию "Other Git VCS"
- SCA Исправлено поведение фильтров экспорта SBOM, из-за которого файл выгружался полностью, независимо от выбранного содержимого
- OSA Исправлена ошибка сканирования образа через OSA API, если указанный менеджер репозиториев не был создан
- OSA Исправлено появление дубликатов тегов у образов при загрузке из реестра
- API Исправлена схема `/api/commits`: поле `author.id` сделано `nullable`
- Исправлено зависание файла отчёта в статусе "In progress" при ошибке запроса на сохранение
- Исправлен запрет создания правила групп при недоступном LDAP
- Исправлено неполное логирование действий с политиками в аудит логе
- В миграциях убрано использование схемы `public`, вызывающее ошибку при миграции в PostgreSQL 15+ со схемой, отличной от `public`

#Удалено

- Secrets При удалении проекта теперь удаляются найденные в нём секреты

#[2026.3.3] - 2026-03-12

#Добавлено

- Добавлены отдельные переменные окружения для таймаутов Index API: `INDEX_API_POOL_TIMEOUT`, `INDEX_API_CONNECT_TIMEOUT`, `INDEX_API_WRITE_TIMEOUT`, `INDEX_API_READ_TIMEOUT`

#Изменено

- Разделён единый таймаут запросов к Index API на pool, connect, write и read

#Устарело

- Устарела переменная окружения `INDEX_API_TIMEOUT`

#[2026.3.2] - 2026-02-20

#Исправлено

- Исправлена ошибка, из-за которой в выгружаемых SBOM не проставлялась метка `vulnerability_is_reachable` для уязвимостей
- Исправлена ошибка сканирования системных пакетов с одинаковыми названиями и версиями, но разными квалификаторами
- Исправлена ошибка анализов проектов с опцией поиска зависимостей по хэшам
- Оптимизирована работа раздела с группами проектов

#[2026.3.1] - 2026-01-30

#Добавлено

- Добавлена конфигурация для обновления PostgreSQL, процедура обновления описана в документации

Обновление до PostgreSQL 15

До конца 2025 года CodeScoring поставлялся с PostgreSQL мажорной версии 13. Финальный релиз PostgreSQL 13 (13.23) состоялся 13 ноября 2025 года. Для обеспечения безопасной и поддерживаемой работы CodeScoring необходимо выполнить обновление до PostgreSQL 15. Администраторам инсталляций, развёрнутых в Docker Compose, необходимо при обновлении указать значение переменной POSTGRES_IMAGE в файле `.env`.

#Исправлено

- SCA Исправлено поведение, при котором Johnny считал блокирующими политики, по которым были отложенные алерты
- Исправлено поведение, при котором по игнорированным алертам отправлялись email-уведомления и создавались задачи в Jira
- OSA Исправлена ошибка сканирования некоторых пакетов Maven
- Исправлено поведение, при котором подсчет уникальных авторов заканчивался с ошибкой через 60 секунд

#[2026.3.0] - 2026-01-16

#Добавлено

- SCA Добавлена поддержка CycloneDX версий 1.7 и 1.7 ext
- SCA Добавлена страница визуализации достижимости уязвимости

- SCA OSA Добавлено поле `GOST:provided_by` в таблицу редактирования зависимостей и в SBOM
- SCA Добавлена возможность указывать ветку или тег при импорте SBOM
- SCA Добавлена проверка формата и версии загружаемого SBOM-файла
- SCA Добавлен API-эндпоинт со списком актуальных спецификаций и версий SBOM
- SCA Добавлена подсветка уязвимых зависимостей в графе при экспорте PDF-отчёта проекта
- SCA Добавлены списки «Затронутые пакеты» и «Затронутые образы» на странице уязвимости
- SCA Добавлена колонка «Требование» в таблице раздела `SCA → Зависимости`
- TQI Добавлены фильтры по дате начала и дате последнего обновления в список проектов в разделе TQI
- TQI Добавлены метрики плотности и скорости кода для графиков на странице TQI-проекта
- TQI Добавлено отображение длительности проекта в месяцах, количества строк кода и количества файлов на странице TQI-проекта
- TQI Добавлена иконка merge-коммитов, фильтрация по типу коммитов и возможность сворачивать сообщение в таблице коммитов на странице TQI-проекта
- TQI Добавлена настройка расписания анализов авторов и клонов для проекта
- TQI Добавлено удаление неактуальных коммитов в проекте при запуске анализа авторов с первого коммита
- OSA Добавлена поддержка менеджеров репозиториев типа «Сфера.Дистрибутивы и Лицензии»
- OSA Добавлен тип `Virtual / Виртуальный` для репозиториев JFrog и Nexus
- OSA Добавлены дополнительные поля для контейнерных образов (время запросов, политики, статус блокировки, реестр, ссылка)
- OSA Добавлена возможность указывать Docker-репозитории для политик с компонентами OSA и передавать имя репозитория при запросе образа
- OSA Добавлена проверка уникальности при создании менеджеров репозиториев
- OSA Добавлено определение экосистем `cargo`, `composer` и `huggingface` в запросах Sonatype Nexus Repository
- OSA Добавлено определение экосистем `ai editor extensions`, `ansible`, `bazel modules`, `helm oci`, `huggingface`, `jetbrains plugins`, `nim model` и `oci` в запросах JFrog Artifactory
- OSA Добавлено вывод PURL на странице контейнерного образа
- OSA Добавлен фильтр по компонентам OSA в списке политик
- OSA Добавлена возможность передачи заголовков хоста для формирования ссылки на заблокированный компонент в OSA
- OSA Добавлена поддержка загрузки данных об образах из реестров JFrog Artifactory через API Repository Path
- OSA Добавлено обновление даты последнего запроса и статуса сканирования при сканировании образа или пакета через UI

- Secrets Добавлено сохранение координат секретов
- Secrets Добавлены координаты в ссылки на секреты
- Secrets Добавлено поле для настройки дефолтного движка секретов
- Data Добавлена левая граница диапазонов уязвимых версий из БДУ ФСТЭК по мажорной версии, которая устраняет перекрытие диапазонов и снижает False Positive
- Data Добавлен сбор исправленных версий для компонентов из блока "Возможные меры по устранению уязвимости" БДУ ФСТЭК
- Добавлено массовое действие для алертов: отвязывание Jira-задач
- Добавлены ссылки для перехода между настройками и просмотром группы проектов
- Добавлена возможность использовать списки в условиях политик
- Добавлены информационные сообщения об исключении проекта из анализа
- Добавлены поля затронутых компонентов OSA, групп и репозиториев на странице просмотра политики
- Добавлен вывод времени сканирований в метаинформации проекта
- Добавлен вывод версий компонентов системы в модальном окне «О системе»
- Добавлено экранирование спецсимволов при поиске в LDAP

#Изменено

- SCA Переработан интерфейс страницы уязвимости
- SCA Изменено отображение количества алертов и информации о зависимостях и уязвимостях на странице проекта, если SCA-анализ ещё не выполнялся
- SCA Изменён способ сравнения PURL при принятии решения об игноре алерта. Подробнее в разделе Игнорирование политик
- SCA Экспорт SBOM из интерфейса теперь выполняется асинхронно (аналогично CSV и PDF отчётам)
- SCA Изменены фильтры рейтинга CVSS: допустимые значения ограничены диапазоном [0,10]
- SCA Убрано отображение `has_exploit` в SBOM в случаях отсутствия эксплойта
- TQI Переработан интерфейс на страницах TQI-проекта и автора
- Изменён индикатор загрузки в UI платформы
- Изменён доступ пользователя с ролью `User` к действиям с алертами
- API В ответах API-фильтров строковые значения `True` / `False` приведены к булевым `true` / `false`
- Изменен HTTP-метод для запроса UserInfo в OIDC на GET
- Изменён базовый образ в сервисе judge: с Alpine 3.20 на Alpine 3.23.2
- Изменён базовый образ в сервисе index-proxy: с Alpine 3.20 на Alpine 3.23.2
- Изменён базовый образ в сервисе osa: с Alpine 3.20 на Alpine 3.23.2
- Изменён базовый образ в сервисе backend: / tasks-* с Alpine 3.20 на Alpine 3.23.2
- Изменён базовый образ в сервисе frontend: с nginx:1.26.3 (Alpine 3.20.6) на nginx:1.29.4

#Исправлено

- SCA Оптимизирован процесс формирования SBOM
- SCA OSA Сокращено время сохранения алертов по окончании анализа
- SCA OSA Оптимизирована работа анализа за счёт устранения лишних запросов при получении данных об уязвимостях и лицензиях
- SCA Оптимизирована выгрузка уязвимостей в CSV
- SCA Исправлено дублирование алертов при фильтрации по связи зависимости или окружению зависимости
- SCA Исправлена работа фильтра `Has VCS / Указан VCS` на странице редактирования зависимостей проекта
- SCA Исправлена некорректная выгрузка поля `match type` при экспорте отчёта зависимостей
- SCA Исправлен подсчёт уязвимостей на странице пакетов
- SCA Исправлено отображение severity при наличии CVSS Score
- SCA Сокращено потребление памяти при активном использовании SCA
- SCA Убраны алерты с отложенной блокировкой при сканировании через Johnny с флагом `--save-results`, если время блокировки ещё не наступило
- SCA OSA Оптимизирована работа API и загрузка страниц UI всех страниц раздела OSA, страницы списка алертов, а также групп проектов
- TQI Оптимизирована загрузка вкладки "Похожие авторы" на странице автора
- TQI Оптимизирована загрузка графиков на странице TQI-проекта
- TQI Исправлено возможное дублирование коммитов у TQI-проектов
- TQI Исправлена формула расчёта темпа изменения для графика на странице TQI-проекта
- TQI Исправлен подсчёт доли технологий в группах проектов
- OSA Исправлено отображение пакетов из удалённых менеджеров репозиториев в дашборде
- Secrets Исправлено сохранение одинаковых секретов с разными координатами в файле
- Secrets Исправлены вывод и запись в аудит-лог ошибки об отсутствии модели при запуске анализа секретов
- Data Исправлены некорректные VCS ссылки с `/sponsor` и github.io
- Data Исправлено отображение лицензий по версиям пакетов - устранены случаи, когда лицензии относились к другой версии
- Data Исправлены некорректные версии БДУ ФСТЭК с символом "X" и запятыми
- Data Исправлены некорректные VCS ссылки в npm с /issues на конце
- Исправлена разблокировка кнопки экспорта PDF после сканирования образа и проекта
- Исправлено форматирование чисел на графиках дашборда и TQI-проектов
- Исправлено отображение разделов платформы при истечении срока действия модуля в активационном ключе
- Скорректирован вывод статистики проекта, по которому ещё не было сканирований

#Устарело

- OSA API-эндпоинты истории сканирования контейнерных образов помечены как устаревшие

#Удалено

- OSA Удалена страница истории сканирования образов
- TQI Убран выбор времени в фильтрах по дате начала и дате последней активности в списке авторов и проектов организации
- TQI Удалены метрики «Темп изменений» со страниц TQI-проекта и автора
- Убрано сохранение истории переходов (в браузере) на внутренних вкладках

#[2025.45.7] - 2025-12-19

#Добавлено

- OSA Добавлены настройки `INDEX_API_TIMEOUT` и `LOG_LEVEL` в сервис Index Proxy

#Исправлено

- OSA Исправлена работа политик в OSA Proxy без указания `repository_id`
- SCA Исправлено формирование версии npm пакетов в SBOM-файлах

#[2025.45.6] - 2025-12-13

#Добавлено

- OSA Реализована работа политик с использованием кэша в сервисе OSA Proxy

#Исправлено

- SCA Исправлено формирование SBOM-файла для проектов с protestware

#[2025.45.5] - 2025-12-05

#Добавлено

- OSA Добавлен автоматический выбор типа авторизации для реестров контейнерных образов

#Исправлено

- SCA Оптимизирован запрос получения списка проектов на странице зависимости

#[2025.45.4] - 2025-12-02

#Исправлено

- SCA Оптимизирована выгрузка PDF-отчета
- SCA Оптимизировано формирование графа зависимостей

#[2025.45.3] - 2025-11-24

#Исправлено

- OSA Исправлено отображение пакетов из удалённых менеджеров репозиториев
- OSA Оптимизирован запрос загрузки списка пакетов
- Secrets Исправлен механизм обнаружение ML-модели в модуле Secrets

#[2025.45.2] - 2025-11-18

#Добавлено

- SCA Добавлено сохранение метаданных при пересканировании проекта из истории сканирования

#Исправлено

- OSA Оптимизирована скорость сканирования пакетов в модуле OSA
- Исправлена ошибка работы с игнорами при расчёте политиками OSA Proxy

#Изменено

- OSA Изменена детализация логирования в модуле OSA

#[2025.45.1] - 2025-11-11

#Исправлено

- Исправлена ошибка работы с самоподписанными SSL сертификатами
- OSA Исправлен некорректный вывод статуса блокировки пакета

#[2025.45.0] - 2025-11-07

#Добавлено

- SCA Добавлен новый раздел `SCA -> Группы проектов` с агрегированными метриками и возможностью запуска массового анализа
- SCA Реализована выгрузка в CSV агрегированных данных SCA-проектов в группе
- SCA OSA Реализован разбор компонентов экосистемы RedOS
- Добавлены новые возможности фильтрации для графа зависимостей: фильтрация связей по родителям и потомкам, разделение по технологиям
- TQI Добавлено отображение количества merge-коммитов и изменённых строк в проекте в таблицах проектов и авторов, на страницах проекта и автора в метриках и на графиках
- TQI Добавлена возможность запускать анализ авторов проекта с первого коммита
- TQI Добавлены метрики "Темп изменения", "Влияние новизны" и "Влияние оттока"
- TQI Добавлен график "Темп изменения" на страницу проекта
- Добавлен автофокус для форм: авторизация, настройки политик, игноры политик, менеджеры репозиториев, реестры, проекты
- Добавлено отображение индикатора применённых фильтров
- SCA Добавлен набор API для отображения данных проектов в разрезе SCA, агрегированных по группам, подробнее об изменениях можно узнать по адресу `/api/swagger`
- OSA Добавлены колонка и фильтр "Инициатор запроса" на страницу `OSA -> Запросы`
- Добавлены примеры запросов с реальными фильтрами для эндпоинтов `/bulk_actions/` в Swagger
- SCA Добавлена колонка "Требование" в списки зависимостей проекта и сканирования
- SCA В PDF-отчёт добавлено поле "Требование" с информацией о версии пакета
- SCA Добавлена колонка "Группы" на страницу списка проектов и в экспортируемый CSV
- SCA Добавлено поле "Максимальная исправленная версия" на странице зависимости
- Добавлены фильтры "Email-ы авторов" и "Merged with" на страницу правил объединения авторов
- Secrets Добавлен фильтр "Статус" на странице проекта
- Secrets Добавлен фильтр "Статус" на странице списка секретов
- Добавлена секция игноров политик в PDF-отчёт
- SCA Добавлен фильтр "Родители" на странице списка зависимостей
- Переработаны `/health` эндпоинты REST-сервисов и добавлены проверки для celery-beat и celery-worker
- Добавлены ссылки на конкретную уязвимость CSPW в отчётах Johnny
- SCA Добавлены фильтры в список уязвимостей на странице детального просмотра зависимости
- Добавлена ссылка на проект в уведомлении о создании проекта

- Добавлена возможность копировать созданную политику
- Добавлена возможность сворачивать боковое меню
- OSA Добавлены ссылки на политики на странице заблокированного компонента
- Добавлена колонка "Блокер" в таблицы алертов и значение задержанной блокировки в статус блокирования
- Добавлена кнопка "Развернуть/свернуть" в карточке секрета
- Добавлена обработка отсутствия слэша для страницы скачивания Johnny (`/download`)
- Добавлена возможность автоматически заполнять настройки OIDC по кнопке "Получить настройки". Подробнее можно узнать в разделе о настройке OpenID Connect

#Изменено

- Перенесена колонка с технологиями в списке проектов организации автора
- Переименовано поле `Host` в URL на странице создания и редактирования реестра контейнеров
- Изменено поведение панели фильтров: при переходе между разделами состояние не сохраняется; при применении фильтров панель остаётся открытой
- OSA Убраны начальные фильтры по актуальности в списках раздела OSA
- Убрана возможность менять пароль у пользователей, заведённых через LDAP или OIDC
- Secrets Убрана подсветка синтаксиса в секции Secrets
- Добавлена поддержка множественного выбора в фильтрах на страницах списка уязвимостей и списка алертов
- Изменены форматы отображения данных в колонках "Сервер LDAP" и "Сервер OIDC" на странице списка пользователей
- Исправлен content-type для ряда методов API в Swagger
- Изменены права доступа для запуска анализа и просмотра проектов в модулях CodeScoring.SCA и CodeScoring.Secrets (актуальные права можно увидеть в разделе управления учетными записями:
- TQI Улучшено отображение данных во всплывающем окне при наведении на графики метрик по проектам
- Изменён контент модального окна уведомления о новой версии
- Изменен расчёт задержки блокировки по политикам с расчёта по дням на расчёт по точному количеству часов
- Улучшен перевод на русский язык в системе
- SCA Очередь `sca-tasks` перенесена в celery worker
- Изменена структура и наполнение бокового меню
- SCA Убрано кодирование текста в unicode в файлах SBOM
- Обновлён Redis до версии 7.4.6

#Исправлено

- Оптимизировано получение значений фильтров с фиксированными значениями

- Оптимизирована загрузка карты активности авторов
- Исправлены ошибки в полях Тип и Тип авторизации на странице редактирования реестра контейнеров
- Исправлен формат таблиц зависимостей и уязвимостей в PDF-отчёте
- SCA Оптимизирована скорость SCA-анализа
- SCA Оптимизирована скорость формирования SBOM во время SCA-анализа
- OSA Исправлен дедлок при сканировании множества пакетов с одинаковым названием, но разными квалификаторами
- Исправлен сброс поля "Исходные файлы" при повторном сканировании проекта
- Исправлена ошибка при редактировании действий с политикой
- Secrets Исправлено отображение списка секретов
- Исправлено отображение уязвимых пакетов на графе зависимостей
- Исправлена фильтрация контейнерных образов со статусом блокировки
- Исправлена ссылка на активные политики на дашборде

#Устарело

- Эндпоинт `/api/tqi/projects/:id/by_complexity` помечен как устаревший. Используйте `/api/tqi/projects/:id/by_commits`
- Переменная окружения `PLATFORM_API_MAX_PAGE_SIZE` помечена как устаревшая. Используйте `CODESCORING_API_MAX_PAGE_SIZE`

#Удалено

- Удалена очередь high-priority
- Удалены проблемные ссылки со страницы детального просмотра автора, вызывавшие скачки UI

#[2025.37.5] - 2025-10-27

#Добавлено

- Максимальное количество элементов на странице в API теперь регулируется переменной окружения `PLATFORM_API_MAX_PAGE_SIZE`, по умолчанию значение 100. Превышение значения в запросе вызовет ошибку

#Исправлено

- OSA Оптимизирована загрузка образов из реестров
- SCA Оптимизирован API-запрос на получение зависимостей по проекту

#[2025.37.4] - 2025-10-15

#Исправлено

- Исправлена ошибка, из-за которой для политик с одним и тем же проектом в группе и в действиях политики дублировались уведомления на email и задачи в Jira
- Исправлена обработка условий политик по критерию CVSS3 Severity
- Исправлена ошибка 500 при определенных параметрах этапов политики
- Исправлена работа политик для условия "PURL полностью соответствует"
- Исправлена ошибка выставления приоритета задачи в Jira после анализа

#[2025.37.3] - 2025-10-07

#Исправлено

- SCA Исправлена ошибка определения путей до манифестов `.csproj` и `project.assets.json` при сканировании .NET проектов

#[2025.37.2] - 2025-10-03

#Исправлено

- Исправлена ошибка, из-за которой для политик без настроенных действий отправлялись ошибочные уведомления
- SCA Добавлена миграция базы данных для исправления признака архивности у алертов, созданных SCA-анализами

#[2025.37.1] - 2025-09-25

#Добавлено

- OSA Добавлена возможность деактивации менеджера репозиториев
- Добавлена очередь `media-cleaner` в сервис `tasks-media`
- Обновлена версия библиотеки `celery`

#Изменено

- Задачи, связанные с очисткой файловой системы от неактуальных медиа-файлов, вынесены в отдельную очередь `media-cleaner`

#Исправлено

- OSA Исправлена ошибка игнорирования алертов по OSA-пакетам
- Исправлена ошибка сервера Index Proxy при наличии кириллицы в имени владельца активационного ключа

- Оптимизировано удаление менеджера репозиториев
- SCA Исправлена ошибка в политике «PURL содержит»: если проверяемый элемент не являлся валидным PURL, политика не срабатывала
- SCA Исправлена ошибка при SCA-сканировании проектов, в которых присутствуют зависимости с одинаковыми name, version и language
- Исправлена ошибка включения неактивных политик
- Secrets Исправлена ошибка при выгрузке PDF-отчёта секретов
- Исправлена ошибка при отсутствии параметра error в ответе от OpenID Connect

#[2025.37.0] - 2025-09-12

#Добавлено

- SCA Добавлен анализ достижимости уязвимостей для Java, работает с использованием модуля построения колграфа Svace
- SCA Добавлена политика "Уязвимость достижима"
- SCA Добавлено отображение в UI признака достижимости для уязвимостей
- SCA Добавлена возможность указать данные `component manufacturer`, которые будут размещены в соответствующем разделе SBOM версий `CycloneDX 1.6` и `CycloneDX 1.6 ext`, на уровне платформы и для каждого проекта в отдельности
- Добавлены новые переменные окружения `DEFAULT_PROJECT_MANUFACTURER_NAME`, `DEFAULT_PROJECT_MANUFACTURER_EMAIL`, `DEFAULT_PROJECT_MANUFACTURER_HOMEPAGE`
- SCA Добавлено сохранение размеченных данных при импорте SBOM
- SCA Добавлена поддержка выгрузки SBOM в формате SPDX
- SCA Добавлены ссылки на подозрительные коммиты для уязвимостей с идентификаторами типа CSPW в SBOM
- Добавлена интеграция с внешними провайдерами идентичности, реализующими протокол OpenID Connect
- SCA Добавлена локализация PDF-отчетов в модуле SCA
- Добавлено сохранение связей для Jira-задач и писем, созданных автоматическими действиями с алертами
- SCA Добавлено поле "Требование", отображающее диапазон требуемых версий в манифесте, в раздел зависимостей проекта
- Добавлено необязательное поле "Приоритет" в форму действия по созданию задач в Jira
- Добавлена возможность создавать кастомные шаблоны для писем и задач Jira в рамках действий с алертами, по умолчанию используется встроенный шаблон
- Добавлен скрипт перешифровки чувствительных данных при смене токена SECRET_KEY
- SCA Добавлено поле "Дата выпуска" в список зависимостей проекта
- Добавлена группировка опций в выпадающем списке условий при настройке политики

- Добавлен перенос правил на странице настройки политики, а также изменён их вид
- Добавлена возможность автоматического обновления списка аудит-лога
- SCA Добавлен поиск по проектам и фильтр по проекту, связи, типу обнаружения и окружению для отдельной уязвимости на странице зависимости
- TQI Добавлена возможность рассчитать количество уникальных авторов в GitLab вне рамок анализа
- SCA Добавлена возможность запуска сканирования SBOM в истории сканов с целью проверки новых уязвимостей в исторических данных по компонентам
- SCA В раздел зависимостей PDF отчета проекта добавлено поле "Максимальная исправленная версия"
- TQI Добавлена возможность фильтрации по нескольким авторам на вкладках "Список" и "Карта активности" в разделе `TQI -> Авторы`
- TQI Добавлено всплывающее окно со списком проектов автора при наведении на количество проектов автора на странице `TQI -> Авторы`
- TQI Добавлен фильтр по авторам на странице проектов в разделе TQI
- TQI Добавлен график "Количество авторов" на странице проекта в разделе TQI
- TQI Добавлены графики "Коммиты автора", "Проекты автора" на странице автора в разделе TQI
- Добавлена возможность изменять названия проектов
- Добавлена возможность массовых действий над некоторыми сущностями в разделе Настройки
- Добавлена возможность остановки генерации отчетов
- Secrets Добавлена возможность экспорта CSV-отчета с секретами на вкладке проекта
- Secrets Добавлена возможность генерации PDF отчета с секретами на вкладке проекта

#Изменено

- SCA Улучшен разбор ссылок на VCS в CycloneDX SBOM
- TQI Улучшен механизм объединения авторов по почтам
- Приведен к единому виду интерфейс запуска действий с алертами
- SCA Хэш коммита в PDF-отчете проекта теперь выводится полностью
- Уменьшено максимальное количество элементов в результатах пагинируемых ответов API до 100
- Заблокирован запуск SCA и TQI анализов до окончания клонирования кода для VCS проектов
- Дополнены сообщения о невозможности запуска SCA анализа для VCS и CLI проектов
- OSA Изменен вывод детальной информации на странице пакета в модуле OSA
- TQI Исправлено отображение кнопки "Изменить сопоставление авторов" при отключенном модуле TQI

- Изменено поле `GOST:source_lang` на `GOST:source_langs` во всех экспортируемых файлах
- TQI В раздел TQI добавлена информация о количестве месяцев, когда автор делал коммиты
- TQI Параметр проекта "Продолжительность (в месяцах)" теперь отображает целое количество месяцев активности проекта
- Удалено поле `internal` из ответа метода API `/api/activation_keys/`
- Разделено API списка проектов по модулям (SCA, TQI, Secrets)
- Базовые образы Index Proxy, сервиса `frontend`, сервиса `backend` / `tasks-*` переведены с Debian bookworm на Alpine Linux
- Улучшена производительность истории сканирований
- SCA Оптимизирована загрузка списков зависимостей и уязвимостей
- Secrets Изменено отображение и содержание текста подсказок о невозможности обучения ML-модели в модуле Secrets
- Изменен тип фильтра "Задержка блокировки" на странице политик

#Исправлено

- SCA Исправлен формат таблицы с алертами в PDF-отчете проекта
- Исправлена ссылка на выборку проектов-кандидатов на рефакторинг в разделе "Панель мониторинга"
- Исправлен сброс выбранного количества элементов на странице в таблицах при изменении сортировки
- Исправлен сброс пагинации при переходе на ту же страницу
- SCA Исправлена обработка ссылок в CycloneDX SBOM
- SCA Реализована недостающая логика для работы алертов при сравнении значения PURL с учетом регистра
- Исправлены автоматически сгенерированные имена типов в схеме OpenAPI
- Исправлена форма глаголов условий в политиках в английской локали
- TQI Исправлены ссылки на проекты на вкладке "Проекты организации" на странице автора
- TQI Исправлены ссылки на дубликаты кода в карте дубликатов
- TQI Исправлена выгрузка проектов автора в CSV
- Исправлена долгая загрузка списка алертов, если выбран фильтр "Менеджер репозиториев = N/A"
- Исправлена схема Swagger для эндпоинта `/api/activation_keys/`
- SCA Исправлено отсутствие опций в фильтре по окружению зависимостей если параметр `USE_SMART_FILTERS` имеет значение `False`
- SCA Исправлена ошибка при сканировании проекта, в зависимостях которого есть пакеты с невалидным PURL
- Исправлен поиск в выпадающих списках в русскоязычной локали
- TQI Исправлены виджеты "Проекты-кандидаты на рефакторинг" и "Внутрипроектные дубликаты" на панели мониторинга
- Исправлен поиск внутри выпадающих списков условий на странице редактирования политики
- SCA Исправлена работа политики "Зависимость опасна" в модуле SCA

- SCA Исправлена ошибка работы политики с условием `PURL exactly_match`

#[2025.29.4] - 2025-08-22

#Добавлено

- Добавлены гранулярные настройки таймаутов для реестров контейнерных образов на подключение, получение соединения из пула, чтение и запись
- Добавлена настройка ограничения количества запросов в секунду для подключаемых реестров контейнерных образов

#Изменено

- Изменена конфигурация параметров загрузки контейнерных образов из реестров. Настройка больше не производится через переменные окружения, все настройки перенесены в UI

#[2025.29.3] - 2025-08-15

#Добавлено

- Добавлена настройка времени хранения результатов задач в Redis `TASK_RESULT_EXPIRATION_PERIOD`

#Изменено

- Изменена версия базового образа osa-api Alpine с 3.21 до 3.20 для поддержки мониторинга Dynatrace последних версий

#Исправлено

- Исправлено отсутствие алертов политик в выгрузке PDF-отчета по первому SCA-сканированию
- Оптимизированы запросы к базе данных при рассчете игноров политик
- Исправлено поведение архивации OSA пакетов. OSA Пакеты, которые загрузились, но никогда не запрашивались, теперь отправляются в архив
- Исправлено сохранение чекбокса "Может создавать CLI проекты через API" на форме настройки пользователя CodeScoring
- Оптимизировано регулярное обновления актуальных OSA пакетов

#[2025.29.2] - 2025-08-05

#Исправлено

- Исправлена ошибка фонового обновления пакетов в модуле OSA

- Исправлена ошибка дублирования уязвимостей в PDF-отчете по проекту
- Добавлена настройка таймаутов для соединений с Redis через переменные `REDIS_SOCKET_CONNECT_TIMEOUT`, `REDIS_SOCKET_TIMEOUT`, `REDIS_SOCKET_KEEPALIVE`

#[2025.29.1] - 2025-07-22

#Исправлено

- SCA Оптимизирована загрузка страницы зависимостей в модуле SCA
- Исправлена ошибка при изменении настроек LDAP

#[2025.29.0] - 2025-07-18

#Добавлено

- SCA Добавлена возможность посмотреть результаты SCA-анализа проекта из истории сканов
- Добавлена возможность кастомизации экспорта SBOM и PDF файлов
- Добавлена возможность ручного создания задачи в менеджере задач и отправки писем для выбранных алертов
- Добавлена возможность настройки отложенной блокировки в политиках
- Добавлена возможность применять игноры политик к группам проектов
- Добавлена новая роль Security Manager
- SCA Добавлено условие для политики `Зависимость является потомком` для поиска дочерних зависимостей нужного пакета на любом уровне графа зависимостей
- SCA Добавлена колонка `Максимальная исправленная версия` в таблице зависимостей
- SCA Добавлена возможность скачать актуальную версию консольного агента Johnny напрямую из платформы
- Добавлена колонка "Технология" в выгрузку списка алертов
- Добавлены всплывающие уведомления с результатом выполнения анализа при его завершении
- Добавлен поиск в выпадающем списке критериев в формах создания и редактировании политики
- Добавлен дублирующий блок кнопок после группы условий в формах создания и редактировании политики
- Добавлена возможность развернуть блок управления условиями политик
- Добавлены настройки формата вывода дат и чисел в интерфейсе системы
- SCA Добавлена легенда для графа зависимостей проекта
- SCA Добавлен идентификатор анализа в вебхуки, связанные с завершением SCA-анализа

- OSA Добавлен вывод признака архивности/активности для OSA-пакетов, контейнерных образов и алертов
- Добавлена проверка на наличие данных при экспорте PDF-отчета
- Добавлена обработка ошибки при попытке скачать файл, который был удален по истечению срока хранения

#Изменено

- Очередь `tasks-media` переведена в Celery. Количество воркеров управляется переменными `CELERY_MEDIA_WORKER_CONCURRENCY` (минимальное количество, по умолчанию равно 2) и `CELERY_MEDIA_WORKER_MAX_CONCURRENCY` (максимальное, по умолчанию равно 4). Переменная `HUEY_MEDIA_WORKERS` удалена
- OSA Оптимизирован механизм фонового обновления OSA-пакетов пакетов. Обновляются только актуальные пакеты. По умолчанию, актуальными считаются пакеты, запрошенные за последние 14 дней, параметр конфигурируется в настройках платформе
- Улучшены сообщения об ошибках при проверке доступности репозиториев через SSH
- Улучшено отображение списка событий в разделе "Вебхуки"
- Secrets Улучшена логика отображения раздела управления ML-моделью в модуле Secrets
- Оптимизирован алгоритм запуска перерасчета политик при обновлении уязвимостей: запуск происходит только при изменении данных, влияющих на политики
- Оптимизирована загрузка страниц с образами и алертами
- Изменен выбор защищенного протокола для подключения к почтовому серверу с чекбоксов на поле с выпадающим списком
- Унифицированы кнопки действий в разделах с таблицами сущностей
- Обновлена OpenAPI спецификация в части обработки ошибок
- OSA Изменён базовый образ в сервисе OSA API с Debian bookworm на Alpine
- Обновлен образ pgbouncer для перехода с libevent на c-ares в качестве DNS-бэкенда для поддержки ресурсных типов записей SOA и протокола EDNS0
- Secrets Обновлена версия gitleaks для модуля Secrets до 8.27.0
- Обновлен образ Redis с 7.0.12 до 7.4.3
- Обновлен образ PostgreSQL с 13.4 до 13.21
- SCA Обновлена версия Johnny на платформе до 2025.29.1

#Удалено

- Удалена переменная `HUEY_MEDIA_WORKERS`

#Исправлено

- Исправлен поиск на странице профиля автора в разделе "Проекты организации"

- SCA Исправлена ошибка валидации хэша образа при передаче значения через параметр `--hash` при сканировании образа через Johnny
- Исправлены некоторые неточности схемы API в Swagger
- OSA Исправлена ошибка обработки списка сущностей из Docker Registry при получении `null` вместо пустого списка
- Исправлена ошибка обработки файлов подписи образов с расширением `.sig` и `.att` при работе с реестрами контейнеров
- TQI Исправлена ошибка на вкладке "Схожие Авторы" на странице профиля автора в случае наличия авторов без определенных технологий
- Исправлена ошибка неполного вывода подключенных систем контроля версий при создании VCS-проекта
- Исправлен вывод длинного имени файла в списке экспортируемых файлов
- Исправлен вывод поля "До" в модальном окне игнора политики
- SCA Исправлен сброс состояния графа зависимостей при переходе на другую вкладку браузера
- Исправлены проблемы обновления данных на странице после редактирования некоторых сущностей
- OSA Исправлены взаимные блокировки при использовании нескольких экземпляров сервиса OSA Registration
- Исправлена ошибка при редактировании подключения к менеджеру задач Jira
- Исправлено некорректное сокрытие чувствительных данных в ошибке при клонировании проекта
- Исправлено отображение выбранных параметров в условии политики
- OSA Исправлен вывод списка авторов на странице с детальной информацией о блокировке пакета в модуле OSA
- Исправлено сохранение политик при изменении некоторых условий

#[2025.21.2] - 2025-06-18

- Оптимизирован процесс миграции данных при обновлении расписания сканирования проектов

#[2025.21.1] - 2025-06-04

- Оптимизирована выгрузка CSV-отчета в разделе "Зависимости"
- Оптимизирована утилизация памяти при выгрузке PDF-отчетов

#[2025.21.0] - 2025-05-21

- Добавлено новое условие политики "Зависимость является протестным ПО". Угрозы, связанные с протестным ПО, помечаются идентификатором CSPW
- Добавлена адаптивность интерфейса под разные размеры экранов, системой теперь удобнее пользоваться на планшетах и мобильных устройствах

- Добавлена возможность задавать имя VCS проекта и создавать несколько VCS проектов для одного репозитория
- Добавлена возможность запуска SCA анализа для VCS проекта с выбором конкретной ветки или тега, не меняя ветку по умолчанию
- Изменено создание и редактирование политик: теперь указывать проект можно независимо от выбранных групп и владельцев, политика будет работать на все выбранные группы и проекты
- Добавлена ссылка на менеджер репозиториев на странице пакета в OSA
- Добавлен фильтр по технологиям и соответствующая колонка в раздел "Алерты", колонка скрыта по умолчанию
- Добавлен фильтр по группам проектов в разделах "Алерты" и "Зависимости"
- Добавлен множественный выбор для фильтров "Поверхность атаки", "Функция безопасности", "Найдено", "Окружение зависимости" в настройках зависимостей проекта
- Добавлена колонка "Заметка" в раздел "Игноры политики", колонка скрыта по умолчанию
- Добавлены фильтры "Тип", "Тип авторизации", "Активен" и поиск по названию и адресу в раздел "Реестры"
- Добавлен часовой пояс в дате генерации PDF-отчета
- Добавлено ограничение количества запросов на логин от одного и того же пользователя в единицу времени, по умолчанию 10 попыток в минуту
- Добавлена поддержка TLS-шифрования для PostgreSQL и PgBouncer при установке через docker compose
- Добавлены фильтры на страницу истории сканирований SCA
- Обновлены карты карты активности проектов и авторов, а также карта сложности и дубликатов: изменено название файла изображения при скачивании, убраны надписи в ячейках, улушена работа масштабирования, исправлены ошибки рендеринга
- Изменена работа с чувствительными данными, такими как токены, ключи и пароли, в API и UI системы
- Изменена логика работы фильтров по всей системе. Фильтры теперь загружаются по запросу (lazy load), оптимизирована часть запросов. При возврате на страницу повторная загрузка данных фильтров не происходит
- Изменено добавление пользователя или проект в группу: существующие не будут предлагаться для выбора
- Обновлена OpenAPI спецификация, для поля References в типе `VulnerabilitySummaryDetail`
- Добавлена информация в секцию metadata tools при выгрузке SBOM в формате CycloneDX
- Изменены настройки автовакуума на более низкие пороги для таблиц с частыми обновлениями
- Добавлена настройка `max_client_conn` для pgbouncer, параметр регулирует общее количество соединений, увеличено значение по умолчанию
- Изменена валидация поля номера телефона для поддержки международных номеров
- Изменён вывод родительских зависимостей в таблице зависимостей у проекта, показываются только первые 5 значений

- Изменён вывод событий в таблице вебхуков, показываются только первые 5 значений
- Исправлена ошибка сортировки уязвимостей образов по Fixed Version
- Исправлен экспорт данных о проектах в CSV, снижено потребление памяти
- Добавлена маскировка чувствительных данных в логах платформе
- Улучшена русскоязычная локализация
- Исправлен вывод ошибки в UI при попытке создать уже существующий проект
- Исправлены ошибки ограничения прав доступа
- Исправлена анимация при переходе между вкладками формы редактирования проекта

#[2025.13.3] - 2025-05-07

- Исправлено поведение игноров политик, когда игнорировались алерты вне области действия игнора по проектам и образам
- Исправлена ошибка, при которой проект не мог быть сохранен из-за нового формата расписаний сканирования проектов

#[2025.13.2] - 2025-04-23

- Добавлен просмотр детальной информации по уязвимостям для инсталляций, имеющих только модуль OSA
- Добавлена проверка подключения к Gerrit по SSH
- Оптимизирована утилизация памяти при расчёте политик

#[2025.13.1] - 2025-04-08

- Добавлена возможность указывать дни недели и время в расписании запуска анализов для модулей SCA и Secrets
- Изменен подход к указанию даты последнего запроса артефакта в OSA для избежания повышенной нагрузки на диск

#[2025.13.0] - 2025-03-28

- Прекращена поддержка указания схемы БД, отличной от `public`, через переменную окружения `DATABASE_SCHEMA`. В случае, если используется такая конфигурация, необходимо ознакомиться с инструкцией.
- Произведена глубокая переработка и модернизация конфигурации CodeScoring в Docker Compose. Перед обновлением, пожалуйста, ознакомьтесь с инструкцией.
- Добавлена поддержка манифестов экосистемы Swift Package Manager
- Добавлена возможность гранулярно настраивать проекты и группы в действиях с политиками для отправки уведомлений на разные email-адреса или разные проекты в Jira в рамках одной политики

- Добавлены разные режимы отправки уведомлений на email и создания задач в Jira в рамках действий с политиками: по одному на алерт или дайджестом на сканирование
- Добавлена обработка результатов анализа секретов при работе с модулем через CLI с помощью консольного агента johnny
- Добавлена возможность пересчета информации о секретах в разделе управления моделью машинного обучения
- Добавлена базовая работа с историей сканов секретов
- Добавлен оператор "не соответствует" в словарных политиках
- Добавлены иконки модулей в меню системы
- Добавлено скрытие токена API на странице настроек пользователя
- Добавлено возвращение uuid заблокированного компонента в OSA API в отдельном поле
- Добавлено детальное отображение ошибки валидации пароля при создании нового пользователя
- Добавлено детальное отображение ошибки валидации пароля в форме смены пароля
- Исправлена работа фильтра по названию проекта в разделе `Настройки -> Игноры политики`
- Исправлено отображение ссылок на пакеты в условия политики в разделе Алерты
- Исправлено поведение системы при получении результатов от johnny без ключа `--save-results` с указанием проекта, теперь результаты сохраняться не будут
- Исправлена некорректная сортировка по названию проекта в списках проектов
- Добавлено скрытие взаимоисключающих полей Access Token и SSH Key при разных режимах настройки подключения к VCS для избежания неправильной валидации
- Исправлено некорректное отображение окружения на графе зависимостей
- Исправлена активность кнопки запуска анализа для CLI-проектов без загруженных зависимостей
- Исправлена ошибка в логике применения политик при использовании групп
- Оптимизирована скорость работы страницы политик
- Исправлена ошибка возможного дублирования уязвимостей
- Оптимизирован механизм обновления информации об уязвимостях для снижения количества записей в базу

#[2025.7.2] - 2025-03-14

- Исправлено падение сервиса OSA API при обращении к несуществующему ключу в Redis в версии 2025.7.0
- Исправлено отображение страниц с результатами сканирования модуля OSA, если пакет запрашивался периодически с интервалом меньше недели
- Увеличен период запроса обновлений кода в VCS-проектах чтобы уменьшить нагрузку на VCS

#[2025.7.1] - 2025-03-07

- Исправлена ошибка в работе механизма игнорирования алертов
- Оптимизирован фоновый пересчет политик для OSA при обновлении информации об уязвимостях
- Исправлена ошибка в работе интеграции с OSSIndex
- Оптимизирована работа обновления даты последнего запроса пакета или образа для уменьшения нагрузки на диск

#[2025.7.0] - 2025-02-14

- Добавлена возможность выбора лицензии в интерфейсе управления полями зависимостей
- Добавлена возможность выбора группы при создании политики
- Добавлена возможность выбора проектов в политике без предварительного выбора собственника
- Добавлена настройка включения и отключения облачного резолва для проектов в модуле SCA
- Добавлены расширенные настройки для VCS проектов: игноры, включение/выключение рекурсивного поиска
- Изменена группировка и отображение настроек проекта
- Добавлен вывод метрики доступности Index API в стандартный механизм отслеживания метрик платформе
- Добавлена настройка через переменную окружения `INDEX_API_FAILURE_RATE_THRESHOLD`, которая определяет, сколько неудачных запросов к Index API в модуле OSA должно произойти, прежде чем система начнёт считать индекс недостижимым
- Добавлена настройка `Skip TLS Verification` при создании подключения к реестру образов
- Добавлены вебхуки для событий модуля Secrets
- Добавлено передподключение к Postgres при потере соединения в сервисе osa-registration
- Обновлены карты в модуле TQI. Рендеринг переведён на фронтенд, реализована более удобная навигация и добавлены дополнительные фильтры по периоду и количеству проектов
- Оптимизирована работа списка зависимостей в модулей SCA
- Оптимизирована работа списка запросов в модуле OSA
- Исправлено сохранение состояния фильтров и настроек пагинации в таблице редактирования зависимостей
- Исправлена ошибка валидации при автозаполнении поля Instance URL в создании подключений к VCS
- Исправлена ошибка настройки колонок в списке проектов в модуле SCA
- Исправлены ошибки переводов при использовании числительных
- Исправлены ссылки на зависимости и уязвимости в Email-дайджесте и Jira Issue
- Исправлена некорректная работа проверки подключения при настройке Email сервера
- Удалён метод API `get_package_info`

#[2024.52.2] - 2025-02-26

- Исправлена работа механизма игнорирования алертов

#[2024.52.1] - 2025-01-24

- Исправлена ошибка, которая могла приводить к резкому увеличению времени сканирования образов
- Изменен подход к фоновому перерасчету политик для контейнерных образов в модуле OSA для оптимизации скорости работы и потребления ресурсов
- Добавлена отдельная очередь повторного фонового сканирования пакетов в модуле OSA для разгрузки основной очереди

#[2024.52.0] - 2024-12-28

- Добавлено более явное разделение модулей в меню
- Добавлены отдельные списки проектов в модулях SCA, TQI и Secrets
- Добавлена поддержка экосистемы Conda
- Добавлено редактирование зависимостей контейнерных образов для выгрузки SBOM
- Добавлен множественный выбор проектов и образов в создание игнора политики
- Добавлена возможность указывать policy stage при создании CLI-проекта
- Добавлена возможность фильтровать списки в разделах Vulnerabilities, Policy Alerts и Projects по нескольким значениям Severity, Policy и Technology
- Добавлено сохранение и отображение редактирования SBOM в аудит-логе
- Добавлено отображение названия CLI-проектов в аудит-логе
- Добавлен фильтр по тэгу образа в раздел Container Images
- Добавлены даты первого и последнего SCA сканирования в списке проектов
- Реализована возможность добавлять проекты в существующие группы через API, интерфейс и опции консольного агента для пользователей с активным флагом **Can create CLI projects via API**
- Вынесено в отдельное окно полное отображение секрета в разделе Secrets
- Дополнен перевод платформе на русский язык
- Добавлена валидация обновления API-токена
- Изменён формат поля recommendation в выгрузке SBOM формата CycloneDX для корректной обработки случаев, когда уязвимость затрагивает несколько версий одной библиотеки
- Исправлена ошибка создания задачи в Jira при срабатывании политики
- Исправлена ошибка фильтрации по статусу в разделе Policy Alerts при сбросе фильтров
- Ошибки введенного URL теперь показываются после завершения ввода

#[2024.48.1] - 2024-12-04

- Исправлена ошибка при запуске сканирования образа из интерфейса

#[2024.48.0] - 2024-11-30

- Добавлена возможность отправки вебхуков на ключевые события в системе
- Добавлена возможность администратору указать значения для полей SBOM `GOST:attack_surface`, `GOST:security_function` и ссылки на VCS, значения будут учтены в рамках выгрузки SBOM в формате `CycloneDX 1.6 Ext`
- Обновлено отображение matched criteria в алертах
- Добавлена возможность вывести колонку Source files в таблицу раздела уязвимостей (Vulnerabilities) и в таблицу Affected dependencies на странице уязвимости
- Добавлены подсказки для пользователя в форме создания и редактирования политики
- Добавлены ссылки со страницы результатов сканирования по проекту на страницу настроек проекта и обратно
- Улучшена типизация ссылок в секции `externalReferences` при выгрузке SBOM в CycloneDX
- Ускорена загрузка графика распределения по лицензиям
- Изменён график распределения по технологиям на главной странице системы и на вкладке SCA для VCS проектов, расчёт производится на основе технологий зависимостей проекта по итогам композиционного анализа
- Исправлена логика работы политик при сочетании нескольких условий для окружения (env) зависимости
- Исправлен импорт SBOM файлов в формате CycloneDX, содержащих информацию в полях `components[i].evidence.identity`
- Исправлены переводы на русский язык для числительных и некоторых словарей системы
- В письмах с оповещениями об алертах идентификатор уязвимости сделан гиперссылкой

#[2024.44.3] - 2024-11-13

- Исправлено несоответствие CVSS Score и CVSS Severity при наличии уязвимости в нескольких базах знаний

#[2024.44.2] - 2024-11-07

- Ускорена загрузка разделов `Components -> Packages` и `Components -> Container Images`

#[2024.44.1] - 2024-11-05

- Добавлена бета-версия локализации интерфейса на русский язык, переключение языков доступно на странице профиля пользователя
- Добавлена поддержка спецификации CycloneDX 1.6 для импорта и экспорта SBOM
- Добавлена выгрузка в формат CycloneDX 1.6 Ext с добавлением полей `GOST:source_lang`, `GOST:attack_surface` и `GOST:security_function` для соответствия требованиям ФСТЭК России. Поля заполняются значением по умолчанию
- Для новых результатов SCA-анализов добавлена возможность выбора версии CycloneDX при скачивании SBOM
- Улучшена выгрузка SBOM во все версии CycloneDX: добавлена информация об отсканированном приложении в `metadata->component`, добавлена информация про версию платформы в `metadata->tools`, обновлён устаревший формат указания авторства компонентов для версий CycloneDX 1.5 и 1.6, исправлен формат лицензии компонентов. Изменения доступны для новых результатов SCA-анализа
- Добавлено отображение дерева зависимостей в PDF-отчёты
- Добавлен сбор данных о malware из GitHub Security Advisory
- Добавлена классификация “Опасный пакет” и соответствующая политика для модуля OSA. Опасными помечаются пакеты с известными Malware и определёнными типами CWE в уязвимостях
- Добавлены дополнительные даты на странице просмотра пакета в модуле OSA: даты первого и последнего запроса к пакету, дата последнего расчёта политик, а также дата обновления информации по пакету
- Добавлено значение `Source files` в выгрузку уязвимостей в разделе Vulnerabilities
- Добавлены условия политик на регистрозависимый поиск строки в названии пакета `contains (case sensitive)`, а также изменены названия регистронезависимых условий условий с `icontains` на `contains (case insensitive)`
- Добавлен фильтр `Has vulnerabilities` и колонка с количеством уязвимостей при просмотре списка в разделах Components и Container images модуля OSA
- Добавлена возможность запуска массового анализа секретов в Workmode
- Добавлена обработка нового типа манифеста `application/vnd.docker.distribution.manifest.list.v2+json` при анализе контейнерных образов
- Добавлена таблица с проектами, в которых используется компонент, на странице просмотра компонента в модуле OSA
- Добавлен новый шаблон `%USER_DN%` для фильтра по группам при настройке LDAP
- Добавлена возможность запустить анализ пакета с его страницы в разделе Components
- Добавлено оповещение об окончании срока действия ключа активации
- Зафиксированы ключевые колонки в таблицах при горизонтальном скроллинге

- Реализован периодический перезапуск фоновых задач для оптимизации потребления памяти
- Стабилизировано время запуска анализов по расписанию
- Оптимизировано обновление информации на странице списка секретов при разметке результатов
- Исправлены ошибки поведения некоторых списков с множественным выбором
- Исправлено отображение записей о группах пользователей в разделе диагностики интеграции с LDAP
- Исправлена загрузка списка контейнерных образов из реестров в случае, если не удалось получить метаданные о некоторых образах
- Исправлены ошибки работы фильтров в таблице раздела Secrets
- Устранена ошибка при попытке отфильтровать зависимости по `License Category = N/A`
- Исправлено отображение пагинаторов на вкладках SCA и TQI на странице проекта
- Изменена конфигурация пулов соединений к PostgreSQL. Для оптимизации потребления памяти платформой внедрено разделение подключений к Postgres на подключения через пулы соединений, работающие в сессионном и транзакционном режиме. В случае, если система установлена через docker compose, необходимо обновить файл `docker-compose.yml`. При использовании кастомных конфигураций пулов соединений, пожалуйста, проконсультируйтесь по процессу обновления со службой заботы.

#[2024.40.1] - 2024-10-09

- Исправлено отображение ветки/тэга репозитория при выборе в настройках проекта

#[2024.40.0] - 2024-10-04

- Добавлена бета-версия нового модуля Secrets, доступ для тестирования можно запросить у вендора
- Реализована зависимость фильтров друг от друга в разделах Policy Alerts, Dependencies и Vulnerabilities
- Добавлено отображение признака Has Exploit в таблицу уязвимостей и на страницу уязвимости
- Реализована возможность свернуть блок фильтров в разделах с фильтрацией
- Добавлен вывод Policy Alerts в PDF-отчет по проекту
- Добавлена возможность сортировки по столбцу Fixed version в таблице уязвимостей
- Добавлена сортировка уязвимостей в PDF-отчете
- Исправлен пересчет даты для критерия политики Dependency Age (Days)
- Исправлено смещение запуска ежедневного анализа
- Исправлена работа маппинга LDAP групп в случае если несколько LDAP групп сопоставляется с одной группой в CodeScoring

- Исправлено неправильное отображение активной вкладки на странице зависимости
- Исправлен вывод названий лицензий в настройке политик
- Исправлено значение поля Last Updated для CLI проектов
- Оптимизирован поиск названий групп в LDAP

#[2024.38.0] - 2024-09-18

- Исправлен экспорт CSV в разделе Dependencies

#[2024.37.0] - 2024-09-12

- Добавлена таблица с командой на страницу проекта
- Добавлена возможность экспорта в CSV в разделе Components
- Ускорен SCA анализ в случаях, когда в рамках манифеста один и тот же пакет встречается множество раз
- Устранена ошибка при формировании отчета по проекту в случае наличия в SBOM некорректных PURL'ов

#[2024.35.0] - 2024-08-29

- Добавлено новое условие для политик – количество уязвимостей в зависимости
- Добавлены метрики OSA на дашборде
- Добавлено количество блокирующих Policy Alerts на дашборде
- Добавлена возможность запускать сканирование проекта через интерфейс параллельно сканированию в Johnny
- Добавлена колонка с количеством Policy Alerts в подразделе `Settings -> Policies`
- Добавлен фильтр по дате последнего запроса в подраздел `Components -> Container Images`
- Реализовано фоновое формирование скачиваемых файлов и ускорена их раздача
- Реализована пагинация при взаимодействии с LDAP
- Ускорена работа раздела Dependencies
- Дополнен список типов репозиториев, чтобы в разделе Repository Managers отображались все репозитории, включая неподдерживаемые

#[2024.32.0] - 2024-08-09

- Добавлена поддержка Bearer авторизации для Container Registries
- Добавлена реализация поиска LDAP групп через записи об атрибутах пользователей
- Добавлен вывод типа репозитория в разделе Repository Managers в случаях если он не поддерживается
- Улучшены инструменты для диагностики интеграции с LDAP
- Исправлено сравнение версий системных пакетов при расчете политик
- Ускорена работа разделов `Components -> Packages`, `Components -> Requests` и Repository Managers

- Ускорена очистка запросов OSA

#[2024.29.1] - 2024-07-17

- Добавлена возможность указывать лицензию для CLI проектов при создании через API
- Исправлена ошибка отображения Matched Criteria в разделе Policy Alerts
- Исправлен цвет уровня предупреждения на странице о заблокированном компоненте

#[2024.28.0] - 2024-07-09

- Реализована новая настройка условий политики безопасности с формированием логических выражений
- Оптимизировано потребление памяти при генерации PDF-отчета
- Поиск в атрибутах пользователей LDAP теперь не зависит от регистра
- Исправлена сортировка в таблице уязвимостей на странице просмотра Container Image
- Исправлен расчет статуса блокировки в OSA с учетом выбранных компонентов в политике

#[2024.25.0] - 2024-06-18

- Добавлена настройка соответствия LDAP группы и уровня доступа пользователя
- Добавлена возможность игнорирования Policy Alerts по PURL
- Добавлена проверка доступности Container Registry перед запуском сканирования образа на платформе
- Добавлено описание методов OSA API в Swagger
- Улучшена стабильность работы платформе в старых браузерах
- Ускорена загрузка раздела `Components -> OSA Packages`
- Исправлено отображение графиков и пагинация таблицы коммитов на вкладке TQI страницы проекта
- Исправлена работа фильтра по дате запроса в разделе `Components -> Packages`
- Исправлена смена устаревшего токена для VCS
- Удален устаревший API метод `/api/dashboard/security/top5_vulnerable_projects/`

#[2024.24.0] - 2024-06-11

- Оптимизировано использование памяти при загрузке описаний уязвимостей

#[2024.22.2] - 2024-05-28

- Добавлена возможность запуска платформы с опцией readOnlyRootFilesystem в Kubernetes

- Ускорен поиск в разделах Dependencies, Policy Alerts и Components
- Улучшена сортировка по полю Dependency/Package в разделе Policy Alerts
- Добавлена очистка временных файлов после принудительного завершения сканирования образа
- Исправлено отсутствие графика Distribution by technology в CLI проектах после первого сканирования
- Исправлена ошибка формирования PDF отчета в случае наличия ссылок в PURL

#[2024.22.1] - 2024-05-27

- Изменен статус в плагинах OSA при получении неизвестного Repository Manager

#[2024.22.0] - 2024-05-27

- Исправлена ошибка при отображении раздела `Settings -> Workmode`

#[2024.21.0] - 2024-05-24

- Добавлена возможность подключения менеджеров репозиториев через интерфейс
- Добавлен фильтр OSA пакетов по менеджерам репозиториев и отдельным репозиториям
- Добавлена настройка политик для определенных репозиториев
- Улучшен маппинг LDAP групп

#[2024.17.2] - 2024-04-27

- Добавлена дата релиза пакета на страницу в разделе `Components -> Packages`
- Выведена подсказка по колонке Plugin mode в разделе `Components -> Requests`
- Исправлена ошибка открытия страницы редактирования Policy Ignore с заданным Container Image
- Исправлено формирование ссылок на коммиты в Gitlab

#[2024.17.1] – 2024-04-23

- Добавлена возможность указывать тип OSA компонента, для которого будет применяться политика
- Добавлен раздел Group mapping для настройки LDAP групп
- Добавлен маппинг LDAP групп на внутренние группы CodeScoring
- Добавлены ссылки на уязвимости и лицензии в поле Matched criteria раздела Policy Alerts
- Добавлен поиск в условие политики по полю Technology
- Добавлено поле Matched criteria в таблицу Policy Alerts на странице проекта

- Возвращено отображение иконки загрузки SBOM
- Колонки License и Vulnerability в таблицах Policy Alerts теперь скрыты по умолчанию
- Ускорена загрузка раздела `Components -> Requests`
- Улучшена периодическая очистка запросов OSA из базы данных
- Исправлен редирект при удалении пользователя
- Исправлено формирование Dependency Name из PURL

#[2024.15.0] - 2024-04-09

- Добавлена возможность настройки нескольких LDAP интеграций
- Добавлен вывод Matched criteria в разделе Container images
- Добавлены колонки и фильтры Scan schedule, Scan with hashes в раздел Projects
- Добавлены колонки и фильтры Scan schedule, Excluded from analysis в раздел `Settings -> Projects`
- Добавлена в Audit Log информация о предыдущем уровне доступа пользователя
- Добавлена поддержка нестандартных портов для SSH при добавлении VCS
- Добавлен процент схожести между авторами в разделе Authors
- Реализована периодическая очистка запросов OSA в базе данных
- Возвращена стадия политики dev в настройки проекта

#[2024.13.0] – 2024-03-27

- Исправлен повторный запуск периодических задач в очереди
- Исправлено длительное ожидание ответа от Index API в случае прерывания соединения

#[2024.12.0] – 2024-03-22

- Добавлен раздел Components, включающий в себя списки компонентов и запросов OSA
- Добавлено поле Vulnerabilities в CSV-отчет по проектам
- Добавлена метрика pool_used для просмотра занятых соединений из пула
- Оптимизирован запуск анализа и выгрузка SBOM CLI проекта
- Исправлено игнорирование флага load full images list при создании или обновлении Registry
- Исправлено отображение количества активных политик в разделе Dashboard
- Исправлено отсутствие вариантов для политики по CVSS2 Authentication
- Исправлена ошибка при вычислении Policy Alerts политики с условием CVSS Score

#[2024.11.0] – 2024-03-18

- Добавлена информация о пакете в секции Policy alerts и Vulnerabilities на странице заблокированного в OSA образа

- Добавлен фильтр Group в разделе Vulnerabilities
- Переименован признак политики Blocks build в Blocker
- Оптимизирован анализ CLI проектов в ситуации, когда зависимости анализируемого проекта часто встречаются в других проектах
- Ускорена загрузка информации о похожих авторах
- Исправлено отображение связей на одном уровне в графе зависимостей проекта

#[2024.10.0] – 2024-03-06

- Исправлено исчезновение кнопки выгрузки HTML для графиков
- Изменено форматирование сообщений коммитов в таблице коммитов проекта
- Исправлено отображение метрик на страницах проектов, для которых анализ не проводился
- Исправлена доступность скачивания PDF-отчета и SBOM в зависимости от статуса анализа проекта

#[2024.9.1] – 2024-02-29

- Исправлена нечувствительность к регистру при создании пользователей из LDAP

#[2024.9.0] – 2024-02-28

- Добавлено отображение статуса анализа в истории для проектов и образов
- Добавлено отображение сообщений коммитов в проектах за прошлые периоды на странице TQI
- Расширены метрики SCA на странице проекта
- Исправлено поведение при загрузке и сканировании образов без тегов
- Исправлено обновление информации о Container registry после сохранения изменений
- Исправлена невозможность просмотра коммитов проекта за последнюю неделю
- Исправлено отображение метрик на страницах проектов, для которых анализ не проводился
- Исправлена интеграция с LDAP
- Уменьшен таймаут при проверке доступности Registry

#[2024.8.0] – 2024-02-22

- Исправлен запуск общего анализа клонов
- Исправлено отображение путей до файлов внутри клонов
- Исправлены переходы на списки клонов с карты клонов
- Исправлено отображение автора при отсутствии поля email
- Исправлено отображение скрытых колонок в таблицах
- Исправлено отображение переменной `block_status` в списке раздела Container Images

- Исправлено обновление списка образов при наличии в registry образов без указания платформы
- Убрано пустое значение при выборе в фильтрах
- Убран автоматический перезапуск сканирования по расписанию в случае падения

#[2024.7.0] – 2024-02-16

- Добавлены уязвимости из БДУ ФСТЭК
- Реализована страница для вывода информации о заблокированной в OSA компоненте
- Реализован запуск анализа авторов по одному проекту
- Реализован запуск анализа клонов по одному проекту
- Добавлена таблица коммитов на вкладке TQI страницы проекта
- Исправлен фильтр по уязвимостям в разделе Policy Alerts
- Исправлен переход на страницу политики из раздела Dashboard

#[2024.5.0] – 2024-02-02

- Обновлен UI системы
- Добавлены новые метрики SCA на странице проекта
- Добавлено условие политики на соответствие PURL регулярному выражению
- Добавлено условие политики на поиск текстовой строки в PURL
- Добавлена проверка на скрытые символы при вводе активационного ключа
- Исправлен фильтр по технологиям в разделе Projects
- Исправлен вывод технологий для CLI проектов в разделе Projects
- Оптимизирован механизм перерасчета политик
- Уточнены сообщения об ошибках соединения с GitLab
- Изменена ориентация раздела Vulnerabilities в PDF отчете по проекту

#[2024.2.0] – 2024-01-12

- Добавлена информация про архитектуру сканируемых образов
- Добавлен фильтр Group на странице проектов
- Добавлена поддержка спецификации CycloneDX 1.5 при импорте SBOM
- Добавлены новые статусы блокировки компонентов OSA
- Улучшена производительность OSA
- Ускорена загрузка Complexity map в разделе Projects
- Убраны сканирования, зависшие в статусе in progress
- Исправлен экспорт отчета в PDF при активном сканировании проекта и для CLI проектов

#[2023.49.0] – 2023-12-08

- Добавлена возможность экспорта PDF-отчета с результатами сканирования проекта

- Добавлена поддержка анализа зависимостей Rust через манифесты Cargo
- Добавлена возможность клонирования репозитория через SSH
- Добавлен новый тип VCS "Other Git"
- Добавлено отображение количества найденных уязвимостей в списке раздела Projects
- Добавлен поиск по вложенным (имеющим записи в разных фидах) уязвимостям в разделе Vulnerabilities
- Добавлена настройка отображения ID проекта в разделе `Settings -> Projects`
- Добавлены нулевые значения в Prometheus метрики OSA API
- Автоматическое проведение анализа после клонирования VCS проекта стало опциональным
- Исправлена ошибка при сортировке таблицы по полю CVSS3 Attack Complexity в разделе Vulnerabilities
- Исправлена некорректная работа OSA с пакетами, имеющими верхний регистр в версии

#[2023.48.0] – 2023-11-22

- Уменьшен размер поставляемых Docker-образов CodeScoring
- Добавлен параметр Matched criteria с причиной сработавшей политики в раздел Policy alerts
- Добавлен новый тип Container registries "Other"
- Добавлено право пользователям с уровнем доступа User создавать CLI проекты через API
- Добавлены Prometheus метрики по статусу сканирования и статусу блокировки компонента в OSA
- Добавлено скрытие пароля и токена в настройках подключения к Jira
- Убраны дубликаты уязвимостей в SBOM, имеющие разные затронутые версии
- Удален устаревший endpoint API `/integration_api/v1/`
- Ускорена работа анализа при большом количестве Policy ignores
- Исправлена ошибка UnsafeOption при работе с Azure и Bitbucket
- Исправлен вывод доступных значений для фильтра Container images в разделе Policy alerts
- Исправлен поиск по зависимости в разделе Policy alerts
- Исправлено отображение фильтра CWE в разделе Vulnerabilities

#[2023.44.0] – 2023-10-31

- Добавлено сохранение фильтров между вкладками в разделе Policy alerts
- Исправлено отображение решенных Policy alerts на странице проекта

#[2023.43.0] - 2023-10-27

- Добавлена поддержка сканирования proxy Docker репозиториев на базе Sonatype Nexus и JFrog Artifactory

- Добавлен Digest и тэги на странице контейнерного образа
- Добавлено фоновое обновление уязвимостей и фоновая работа политик для компонентов из контейнерных образов
- Добавлены метрики количества и времени запросов для CodeScoring OSA
- Дополнены метрики очередей на платформе – теперь можно отдельно посмотреть типы анализов в очереди
- Добавлено новое условие политики – возраст уязвимости
- Добавлена возможность не загружать автоматически список образов при добавлении Container Registry
- Исправлено открытие страницы просмотра Policy Ignore
- Исправлено отображение ошибки git 128 при работе с VCS

#[2023.41.0] - 2023-10-09

- Для запуска CodeScoring теперь не требуется наличие прав суперпользователя внутри контейнера. Инструкция по миграции с root-контейнеров на rootless доступна у вендора

#[2023.40.0] - 2023-10-04

- Добавлена возможность сканирования образов из hosted Docker репозиториев на базе Sonatype Nexus и JFrog Artifactory
- Добавлена возможность блокирования загрузки образов из Sonatype Nexus Repository при несоответствии политикам безопасности

#[2023.38.0] - 2023-09-20

- Добавлена возможность указания спецсимволов в пароле для подключения к базе данных
- Исправлена ошибка отображения игнорируемых алертов в разделе Policy Ignores
- Исправлено отображение полей Source files и Parents на странице зависимости
- Возвращено моментальное удаление алерта из списка Active после создания правила игнорирования
- Дополнена схема обновления на актуальную версию продукта.

#[2023.35.0] - 2023-08-31

- Добавлена история SCA сканирований в проекте
- На странице проекта информация модулей SCA и TQI теперь отображается в отдельных вкладках
- Добавлено отображение частей CVSS векторов в списке уязвимостей
- Добавлено новое условие политики "Vulnerability has fixed version"
- Добавлено поле CWE в экспорт CSV-таблицы уязвимостей
- Добавлен баннер CodeScoring и логирование версии в консоль при запуске Docker-контейнера с платформой

- Добавлены метрики количества запущенных анализов для Prometheus
- Удален deprecated endpoint `/api/policy_alerts/ignored/`
- Удален deprecated endpoint `/api/policy_alerts/resolved/`
- Удален deprecated endpoint `/api/policy_alerts/`
- Исправлен вывод технологий на списке Policy ignores
- Исправлена работа политики по purl и CVSS для rpm
- Убран текст лицензий из генерируемых SBOM по проекту

#[2023.31.0] - 2023-08-04

- Ускорена работа CodeScoring OSA

#[2023.30.0] - 2023-07-27

- Добавлена fixed version в таблице Affected dependencies на странице уязвимости
- Добавлены метрики состояния очередей для Prometheus
- Перенесена версия платформы из футера в боковое меню
- Исправлено несоответствие Swagger-схемы с API
- Убрана возможность нажать кнопку Upload SBOM без наличия прикрепленного файла

#[2023.26.0] - 2023-06-30

- Добавлена поддержка системных пакетов для OSA-плагина в NXRM
- Добавлены новые условия политики - по наличию эксплойта уязвимости и частям CVSS-вектора
- Добавлен вывод названия политики, stage и level в список Policy alerts на странице проекта
- Добавлен фильтр по типу проекта в списке проектов
- Пометили метод `/api/dependencies/csv/` как устаревший. Теперь надо использовать `/api/dependencies/by_project/csv/`
- Исправлена ошибка при добавлении проекта без предварительного создания VCS
- Исправлено отображение информации о stages на карточке описания политики для уровня доступа User

#[2023.24.0] - 2023-06-13

- Добавлен фильтр для поиска по названию политик в разделе Policies
- Исправлен экспорт зависимостей с пустым фильтром

#[2023.22.0] - 2023-06-01

- Исправлена загрузка SBOM в формате CycloneDX, полученного от `Microsoft.SBOMTool`
- Исправлена обработка дробного рейтинга уязвимости при формировании SBOM

#[2023.21.3] - 2023-05-30

- Исправлена загрузка SBOM в формате CycloneDX, полученного с помощью сторонних инструментов
- Ускорена загрузка страниц Code Clones

#[2023.21.2] - 2023-05-23

- Добавлена возможность загрузки и скачивания SBOM по проекту в формате CycloneDX
- Добавлена возможность проведения анализа на CLI-проекте через интерфейс
- Добавлено ручное обновление кода проекта в настройках
- Добавлена возможность подключения нескольких VCS с одним адресом, но разными токенами
- Добавлена поддержка basic auth для интеграции с Jira
- Исправлен вывод списка лицензий на графике
- Исправлено появление дубликатов алертов при большой нагрузке

#[2023.15.0] - 2023-04-14

- Добавлен новый тип проектов без привязки к репозиторию — CLI
- Добавлена группировка Policy alerts в дайджесты для уведомлений через email
- Добавлен поиск по названию политики и имени зависимости в Policy alerts
- Добавлены рекомендации по устранению уязвимостей (fixed version)
- Добавлена поддержка разбора пакетов в форматах deb/apk/rpm
- Добавлен вывод предложения обновить страницу, если версия клиента отличается от версии сервера
- Добавлены незначительные улучшения в UI

#[2023.11.0] - 2023-03-16

- Исправлена долгая загрузка графика Distribution by license на Dashboard
- Добавлено поле Note к Policy ignore
- Добавлено поле Description к Policy
- Добавлен новый уровень доступа Auditor
- Добавлена настройка времени жизни сессии через переменную окружения
- Добавлены ссылки на граф зависимостей

#[2023.6.0] - 2023-02-10

- Добавлены графы зависимостей проекта (ссылка есть на странице проекта)
- Добавлена опция отключения сбора хешей во время SCA на платформе

- Добавлен кеш ответов Index API для OSA (по-умолчанию от 1 часа до 1.5 часов, настраивается через переменные окружения)
- Дополнена информация об ограничениях в использовании OSSIndex
- Запуск массового SCA теперь логируется в Audit log
- Для Swagger больше не нужен интернет
- Изменен путь до статики из бекенда (требуется поправить `docker-compose.yaml`)
- Исправлена ошибка, из-за которой в одноименных пакетах (с разными версиями), находящихся в разных манифестах, неправильно отображалась информация о файле, в котором пакет найден

#[2022.49.1] - 2022-12-07

- Исправлена работа страницы проекта

#[2022.49.0] - 2022-12-07

- Добавлены события логина в Audit log
- Добавлена обработка ноды unresolved
- Добавлено новое условие в политиках на отсутствие версии компонента
- Изменено API платформы
- Исправлена работа анализа при отсутствии env в графе зависимостей

#[2022.48.2] - 2022-12-04

- Добавлена поддержка modern yarn
- Изменен механизм отнесения зависимостей к прямым
- Добавлена секция unresolved для зависимостей, которые есть в lock файле, но не имеют родительского компонента

#Johnny Changelog

#[2026.20.1] - 2026-05-28

#Исправлено

- Исправлено формирование SARIF отчета для результатов без уязвимостей
- Исправлено формирование отчета `gl_dependency_scanning`: добавлено заполнение блока `severity`

#[2026.20.0] - 2026-05-12

#Добавлено

- Добавлена поддержка trufflehog для сканирования секретов

- Добавлен флаг `--commit` для секретов при сканировании директорий
- Добавлен режим git для сканирования секретов
- Добавлены флаги для передачи конфигураций gitleaks и trufflehog
- Добавлена поддержка линковщика ld.lld в командах scan build
- Добавлена обработка свойства `GOST:provided_by` компонента при импорте SBOM
- Добавлена обработка свойства `inner_source` компонента при импорте SBOM
- Добавлена поддержка свойства `source-distribution` компонента при импорте SBOM
- Добавлена поддержка `Directory.Packages.props` файлов для nuget проектов
- Добавлены флаги `--include-envs` и `--exclude-envs`, позволяющие фильтровать прямые зависимости участвующие в анализе по их средам
- Добавлена поддержка `pdm` манифестов `pdm.lock` и `pylock.toml`
- Добавлена поддержка резолва для `pdm`
- Добавлена поддержка алгоритма Streebog при импорте SBOM в формате CycloneDX 1.6
- Добавлена поддержка графа вызовов Joern для языка JavaScript

#Изменено

- Изменена фильтрация пакетов при импорте SBOM, теперь неизвестные/неподдерживаемые экосистемы получают тип generic
- Изменено формирование PURL для компонентов из файла `--lib-versions` при анализе сборок C/C++: теперь используется тип `generic`
- Изменено определение pid корневого процесса с учетом userspace в команде scan build ebpf

#Исправлено

- Исправлен формат создания отчётов для секретов
- Исправлен вывод ошибок при валидации SBOM
- Исправлен текст ошибки при попытке добавить проект в группу в которой он уже состоит
- Исправлен расчет статуса отложенных блокирующих политик
- Исправлено определение корневых зависимостей при сканировании pipdeptree вывода
- Исправлена ошибка при импорте SBOM с лицензией не входящей в список поддерживаемых: обновлен список лицензий

#[2026.11.1] - 2026-04-16

#Исправлено

- Исправлено создание группы при совпадении имени группы с префиксом большого количества уже созданных групп

- Исправлено определение родительского пакета при работе резолва в окружении go
- Исправлена ошибка шаблона формирования csv отчета политик

#[2026.11.0] - 2026-03-10

#Добавлено

- Добавлена поддержка CVSSv4 в отображении результатов анализа
- Добавлена обработка отложенной блокировки политики
- Добавлен вывод проигнорированных компонентов при сканировании SBOM
- Добавлена поддержка C# в анализе достижимости
- Добавлена сборка под Windows в модуле Svace (начиная с версии 5.0.260311)
- Добавлена поддержка удалённого анализа в модуле Svace (начиная с версии 5.0.260311)

#Изменено

- Изменено поведение флага `--project-group`, при использовании с существующим проектом этот проект будет добавлен в указанную группу
- Улучшен механизм поиска библиотек на rpm-based ОС в командах `scan build` и `scan build ebpf`

#Исправлено

- Исправлена ошибка при создании проекта с уже существующим в системе именем и без прав на работу с ним
- Исправлена ситуация при формировании sarif, когда при отсутствии location для компонента выставлялся путь до проекта
- Исправлена ошибка с определением среды зависимостей в `package-lock.json`
- Исправлено некорректное отображение security-severity при формировании результата в sarif формате
- Исправлен вывод уязвимых функций в случае, когда уязвимая функция вызывала сама себя
- Исправлен вывод line_start, line_end на start_line, end_line согласно схеме Gitlab для уязвимостей в CI/CD для Секретов
- Исправлена ошибка при работе с кэшем в модуле Svace (начиная с версии 5.0.260311)
- Исправлена ошибка анализа при задании адреса сервера с завершающим слэшем в модуле Svace (начиная с версии 5.0.260311)

#[2026.3.2] - 2026-01-28

#Исправлено

- Исправлено отображение условий политики для алертов на лицензионную несовместимость
- Исправлена обработка поля workspaces при сканировании `package-lock.json`
- Убрана лишняя информация ("сохранено в...") при сохранении отчётов в файл
- Исправлена критическая ошибка при сканировании секретов
- Исправлена обработка корневого манифеста в `tool.uv.workspace` для `pyproject.toml`

#[2026.3.1] - 2026-01-14

#Изменено

- Отключена проверка значения `externalReference.url` на соответствие формату iri-reference при валидации SBOM перед сканированием

#[2026.3.0] - 2026-01-13

#Добавлено

- Добавлена поддержка текстового формата lockfile для пакетного менеджера bun (1.2 и старше)
- Добавлена поддержка графа вызовов модуля Svace для языка Kotlin
- Добавлен флаг `--create-project-group` для создания группы и добавления в нее проекта
- Добавлен флаг `--project-categories` для указания категорий создаваемого проекта
- Добавлен флаг `--localization` для локализации результатов работы агента (форматы coloredtable, table, text, csv)
- Добавлен флаг `--gitleaks-config` команды `secrets gitleaks` для передачи пути к файлу конфига gitleaks
- Добавлена пара флагов `--policy-ignores` и `--ignores-format` для вывода игноров политик в указанном формате
- Добавлена сборка консольного агента Johnny для Linux с процессорами ARM
- Добавлена валидация при сканировании SBOM (поддерживаемые форматы: 1.4, 1.5, 1.6, 1.6_ext, 1.7)
- Добавлена поддержка выгрузки SBOM в формате 1.7
- Добавлена корректная обработка транзитивных зависимостей без purl при импорте SBOM
- Добавлены координаты обнаруженного и отслеживаемого секрета
- Добавлен вывод импакта уязвимости при выгрузке результата в формат CSV

#Изменено

- Изменен уровень угрозы в отчете по секретам на `Critical` для каждого секрета для совместимости с отчетом gitlab
- Изменены пути в результатах сканирования секретов с абсолютных на относительные
- Изменен вывод списка обработанных агентом манифестов. В него попадают все манифесты, даже если в них не обнаружено зависимостей
- Изменен вывод общей информация по результату анализа. Информация всегда выводится в консоль, даже при перенаправлении вывода результатов в файл
- Улучшена логика определения ОС Linux для более корректной идентификации ОС семейства RHEL в команде `scan build`
- Изменена обработка библиотек, не прошедших проверку по ldconfig в команде `scan build`. Они добавляются в список unresolved

#Удалено

- Прекращена публикация образов консольного агента с тегом `{VERSION}-busybox`

#Исправлено

- Исправлена паника при импорте некоторых SBOM

#[2025.45.4] - 2025-12-12

#Исправлено

- Исправлено зацикливание на некоторых проектах JavaScript при обработке workspace

#[2025.45.3] - 2025-12-01

#Изменено

- Оптимизирована обработка gradle-dependency-tree

#[2025.45.2] - 2025-11-19

#Добавлено

- Добавлен вывод поля Summary при экспорте результатов сканирования в формате CSV

#[2025.45.1] - 2025-11-14

#Исправлено

- Исправлены относительные пути до манифестов в результатах сканирования .NET, uv и npm/pnpm проектов
- Исправлен механизм ассоциации `packages.lock.json` в проектах с использованием sln
- Исправлен вывод результатов сканирования образов

#[2025.45.0] - 2025-11-05

#Добавлено

- Добавлена поддержка Go в анализе достижимости
- Добавлена поддержка Python в анализе достижимости
- Добавлена поддержка пакетного менеджера UV для экосистемы Python
- Добавлен вывод Matched criteria в отображении алертов политик (начиная с версии **2025.45.0** платформы)
- Добавлена поддержка флага `--branch-or-tag` и `--commit` для команды `scan bom` для указания ветки или тега репозитория
- Добавлена поддержка workspaces для проектов npm и pnpm
- Добавлен вывод о наличии exploit в уязвимостях для отчетов junit и gitlab
- Добавлен поиск информации для статических библиотек через dpkg в командах `scan build` и `scan build ebpf`
- Добавлена поддержка `pipdeptree` для экосистемы Python через опцию `--pipdeptree-resolve`
- Добавлено определение и вывод путей до файлов манифестов внутри сканируемого образа

#Изменено

- Добавлено ограничение на вывод путей достижимости в таблице. Выводится 5 путей и количество не отображенных
- Оптимизирован вывод путей достижимости в текстовом виде

#Удалено

- Команды `scan build` и `scan build ebpf` исключены из исполняемых файлов консольного агента для ОС, отличных от Linux
- Прекращена сборка scratch образов консольного агента. Все теги образов консольного агента указывают на busybox версию

#Устарело

- Сборка образов консольного агента с тегом `*-busybox` будет прекращена в релизе 2026.3.0

#Исправлено

- Исправлена разница в поведении при анализе npm зависимостей с `--npm-resolve` и при наличии уже имеющегося `package-lock.json`
- Исправлена ошибка дублирования уязвимости в графе достижимостей, текстовом и табличном выводе
- Исправлен указываемый путь до манифестов в результате сканирования при использовании resolve
- Исправлен вывод алертов политик: добавлена валидация для исключения пустых объектов
- Исправлена ошибка разбора файла `gradle-dependency-tree.txt`
- Исправлена обработка комментариев при работе с рядом типов манифестов
- Убрано дублирование найденных зависимостей при сканировании образов с опцией `--scan-files`

#[2025.37.2] - 2025-10-06

#Исправлено

- Исправлена ошибка определения путей до манифестов `.csproj` и `project.assets.json` при сканировании .NET проектов

#[2025.37.1] - 2025-09-22

#Исправлено

- Исправлена ошибка при распознании версии локального gitleaks

#[2025.37.0] - 2025-09-08

#Добавлено

- Добавлен анализ достижимости уязвимостей для Java и графа вызовов Svace, а также вывод примеров достижимости в SARIF (начиная с версии 2025.37.0 платформы)
- Добавлена возможность запуска локального сканирования без указания параметров активации (`--api_url` и `--api_token`) с формированием в результате SBOM на основе найденных манифестов без обогащения и применения политик
- Добавлена поддержка манифестов `deps.json` и `sln` для стека .NET
- Добавлена поддержка компонентов всех типов из спецификации PURL, включая тип generic, а также компонентов с пустым или невалидным PURL при анализе SBOM командой `scan bom`
- В команде `scan bom` Добавлено предварительное конвертирование файла в кодировку UTF-8 для дальнейшей корректной обработки
- Добавлена возможность запуска команд `sign bom` и `verify bom` без указания параметров активации (`--api_url` и `--api_token`)
- Добавлена передача размеченных данных (поля `GOST:attack_surface`, `GOST:security_function`, `GOST:source_langs`,

VCS, `licenses`) в платформу при импорте SBOM с разметкой (начиная с версии 2025.37.0 платформе)
- Добавлена возможность указания ветки/тега и коммита при сканировании образа
- Добавлена возможность выбора типа прогресс-бара `spinner` или `text` (по умолчанию `spinner`)

#Изменено

- Изменен уровень логирования при срабатывании политики с error на warning
- Изменен вывод количества алертов по сработавшим политикам
- Изменена логика работы резолва в окружении для JavaScript. Локальный резолв не выполняется при наличии любого из известных lock-файлов (`package-lock.json`, `yarn.lock`, `pnpm-lock.yaml`)
- Доработан механизм связывания для gradle при использовании произвольных имен манифестов. Теперь `build.gradle` и `gradle.lockfile` автоматически связываются, даже если их имена не совпадают. При наличии `gradle-dependency-tree.txt` приоритет отдается ему, а остальные файлы обрабатываются отдельно
- Изменен порядок проверки возможности запуска команды и получения результата анализа в соответствии с лицензией платформы. Теперь она происходит перед выполнением сканирования

#Исправлено

- Исправлена обработка `requirements.txt`, содержащего комментарий о другом источнике зависимостей

#Устарело

- Сборка scratch образов консольного агента будет прекращена в релизе 2025.45.0

#[2025.29.3] - 2025-08-22

#Исправлено

- Исправлена передача аргументов пакетным менеджерам при использовании механизма локального резолва при отсутствии lock-файла

#[2025.29.2] - 2025-07-25

#Исправлено

- Исправлено аварийное завершение при обработке зависимостей, для которых не удалось определить окружение

#[2025.29.1] - 2025-07-18

#Исправлено

- Исправлена ошибка при выгрузке результатов в формате `junit`

#[2025.29.0] - 2025-07-16

#Добавлено

- Добавлен разбор зависимостей, объявленных в объединенном формате в файле `build.gradle`
- Добавлена команда `sign bom` для подписи SBOM файлов
- Добавлена команда `verify bom` для проверки подлинности подписи SBOM файла
- Добавлена работа с предопределенной задачей `CodeScoring_All_Dependencies` для корректного разрешения зависимостей в мультимодульных проектах окружения gradle
- Добавлен параметр `project-proprietor` для привязки сканируемого проекта к подразделению (начиная с версии **2025.29.0** платформы)
- Добавлена поддержка алиасов для `yarn.lock` и `pnpm-lock.json`
- Добавлена поддержка отчетов для алертов в форматах coloredtable, table, text, json, csv. Формат управляется параметром `--alerts-format`
- Добавлены флаги `--branch-or-tag` и `--commit` в команды `scan build` и `scan build ebpf`
- Добавлена выгрузка признака HasExploit в формат sarif
- Добавлен вывод информации о лицензиях в форматы text, table, coloredtable
- Добавлена выгрузка данных Relation, Parents, Match type, Env в формат CSV
- Добавлена возможность передавать флаги пакетным менеджерам при разрешении зависимостей
- Добавлен вывод предупреждения об ошибках парсинга в процессе сканирования
- Добавлена поддержка групп зависимостей с произвольным названием в `pyproject.toml`
- Добавлена возможность передачи SHA-хэша образа в параметре `--hash` команды `scan image` (начиная с версии **2025.29.0** платформы)
- Добавлена проверка на доступность команды `dir` локальной версии `gitleaks`

#Изменено

- Отключен запуск парсера в окружении pip по умолчанию в команде `scan python`. Включается явно флагом `--pip-resolve`
- Улучшена работа команды `scan build ebpf`

#Исправлено

- Исправлено поведение игнорирования для пустых значений параметра `--ignore`
- Исправлена ошибка определения relation зависимости при разборе пары манифестов `package.json` и `yarn.lock`
- Исправлено определение окружения в случаях, когда зависимость одной и той же версии представлена в нескольких окружениях
- Исправлено попадание `poetry-core` из секции `build-system` в список зависимостей при парсинге манифеста `pyproject.toml`

#[2025.21.0] - 2025-05-21

- Добавлена команда `scan build ebpf` для сканирования сборки C/C++ проектов с использованием eBPF
- Добавлена выгрузка в sarif данных о связи зависимости в рамках проекта, прямая или транзитивная, в формате: results.properties.relation: direct|indirect
- Добавлено игнорирование закомментированных строк при разборе файлов `conanfile.py`
- Исправлено определение версии из требований вида `==3.0.0.post1` манифестов Python
- Исправлена выгрузка в sarif уязвимостей, у которых указана критичность без численной оценки
- Исправлен парсинг в окружении Go: транзитивные зависимости, для которых не удалось определить родительский пакет, исключаются из результатов сканирования

#[2025.13.0] - 2025-03-28

- Добавлена обработка поврежденных файлов `scala-dependency-tree.txt`
- Добавлена поддержка парсинга манифестов экосистемы Swift: `Package.swift` и `Package.resolved` (начиная с версии **2025.13.0** платформы)
- Добавлена бета-версия работы консольного агента с модулем Secrets (начиная с версии **2025.13.0** платформы)
- Добавлен разбор зависимостей, объявленных в необъединенном формате в файле `build.gradle.kts`
- Добавлено игнорирование `.nuspec` файлов в команде `scan csharp`
- Добавлена поддержка операционных систем семейства AltLinux в команде `scan build`
- Исключены из сканирования архивы Java при неактивном флаге `--scan-archives`
- Исправлен импорт SBOM в котором у библиотеки указано несколько значений свойства env
- Исправлена обработка SBOM-файлов в формате CycloneDX, содержащих информацию о компонентах внутри компонентов

#[2025.7.0] - 2025-02-13

- Добавлены команды сканирования директории с предзаданными настройками в зависимости от выбранной технологии (например, `./johnny scan java`)
- Добавлен вывод информации о наличии Exploit для уязвимостей в результаты работы агента
- Добавлен параметр `--cloud-resolve` для активации облачного резолва (совместимо с платформой версии **2025.7.0** и выше)
- Добавлена поддержка механизма Selective dependency resolutions для Yarn
- Добавлена поддержка механизма NPM Package Aliases для `package-lock.json`
- Оптимизирована обработка больших файлов `gradle-dependency-tree.txt`
- Исправлена ошибка в определении версий пакетов в файле `gradle.lockfile` при наличии в версии суффикса

#[2024.52.2] - 2025-01-23

- Исправлено поведение агента, приводящее к росту очереди tasks-policy в платформе

#[2024.52.1] - 2025-01-16

- Исправлена обработка нескольких файлов `gradle.lockfile` на один модуль

#[2024.52.0] - 2024-12-24

- Добавлена команда scan build для анализа сборки для языков C и C++
- Добавлены новые форматы выгрузки результатов работы: GitLab Dependency Scanning Report и GitLab Code Quality Report
- Добавлена обработка параметра `--ignore` при сканировании архивов и файлов внутри образов
- Добавлена возможность указания ссылки на ветку/тэг и коммита параметрами `branch-or-tag` и `commit` при сканировании файла и каталога (при взаимодействии с платформой версии **2024.52.0** и выше)
- Добавлена возможность указания хэша параметром hash при сканировании образов (при взаимодействии с платформой версии **2024.52.0** и выше)
- Добавлена возможность указания policy stage при создании CLI проекта (при взаимодействии с платформой версии **2024.52.0** и выше)
- Добавлено указание путей к манифестам внутри сканируемых образов, в которых найдена информация об уязвимом пакете
- Добавлены пути к манифестам, в которых найден уязвимый пакет, в формат `sarif`

- Исправлено аварийное завершение при обработке некорректного файла в формате `yaml`
- Добавлена обработка ошибки, возникающей в случае, когда файл был удален в процессе сканирования
- Исправлено наличие лишних символов при выгрузке в формат `sarif`
- Исправлено определение окружения при анализе манифестов Poetry
- Исправлено сканирование образов на основе RedHat

#[2024.48.2] - 2024-12-13

- Исправлено аварийное завершение при обработке некоторых файлов `gradle-dependency-tree.txt`
- Исправлен парсинг lock-файлов npm и yarn в паре с манифестом

#[2024.48.1] - 2024-12-10

- Добавлена поддержка множественности версий одного пакета в файле `poetry.lock`
- Добавлена возможность запуска без ожидания результатов анализа (параметр `--no-wait`)

#[2024.48.0] – 2024-11-29

- Добавлена поддержка парсинга манифестов экосистемы Conda: `environment.yml`, `meta.yml`, `conda-lock.yml`
- Добавлена поддержка парсинга компонентов Conda в окружении сборки
- Добавлен вывод предупреждения для пакетов с невалидным именем
- Улучшено построение графа зависимостей для форматов, допускающих несколько версий одного пакета
- Улучшено построение графа зависимостей при наличии обоих файлов пары манифест-локфайл
- Исправлены ошибки формирования PURL и версий go пакетов при сканировании Docker образов
- Исправлена обработка SBOM файлов в формате CycloneDX, содержащих информацию в полях `components[i].evidence.identity`
- Изменена логика формирования свойства distro для PURL пакетов ALT Linux при сканировании Docker образов
- В выгрузку в формате `sarif` добавлена информация о Location и Fixed Version уязвимости

#[2024.44.1] - 2024-11-15

- Исправлена ошибка с пропуском gem пакетов в команде `scan bom`
- Исправлена работа флага `ignore` на ОС Windows
- Исправлена ошибка в работе парсера в окружении Go на проектах без зависимостей

#[2024.44.0] - 2024-11-02

- Добавлен парсинг манифестов `pnpm-lock.yaml`. Поддерживаемые версии: 5.0-5.4, 6.0, 9.0
- Добавлен парсинг в окружении pnpm
- Учтено использование файла конфигурации `pnpm-workspaces.yaml` при парсинге `package.json`
- Добавлена возможность указать группу при создании CLI проекта, доступно только для пользователей с ролью администратора
- Добавлена возможность указания формата генерируемого SBOM с помощью параметра `--bom-format` (начиная с версии on-premise 2024.44.1)
- Реализован парсинг в окружении pip
- Реализован парсинг в окружении composer
- При разрешении зависимостей в окружении go улучшен механизм определение родительской библиотеки для транзитивных зависимостей, полученных из тестового окружения
- Исправлена ошибка `unsupported type` для composer компонентов в команде `scan bom`

#[2024.40.2] - 2024-10-18

- Исправлено построение графа зависимостей в случаях, когда компонент встречается несколько раз с разным `bom-ref`

#[2024.40.1] - 2024-10-10

- Добавлено слияние результатов парсинга `pom.xml` и `mvn-dependency-tree.txt` для исключения лишнего резолва зависимостей
- Исправлена ошибка в проверке наличия лок-файла при использовании разрешения зависимостей в окружении

#[2024.40.0] - 2024-10-02

- Добавлен разбор workspaces в парсинг манифестов npm
- Исправлена работа парсера `Gemfile.lock` для случаев с несколькими секциями Gem

#[2024.39.0] - 2024-09-23

- Разделены тэги при выгрузке в формате sarif для отображения в DefectDojo всех версий найденного пакета
- Изменена выгрузка severity в формате sarif для корректного отображения CVSS3 в DefectDojo
- Исправлена ошибка сканирования SBOM с пакетами Go
- Исправлена паника при парсинге пустого `cargo.lock`
- Убрано дублирование уязвимостей в формате sarif для случаев нескольких версий одного пакета

- Убрана возможность одновременного использования флагов `format` и `no-summary`

#[2024.36.0] - 2024-09-05

- Добавлена возможность настройки используемых парсеров через файл конфигурации
- Добавлена возможность указания парсера, используемого в команде scan file
- Исправлен парсинг мультипроектного/модульного gradle-dependency-tree

#[2024.35.0] - 2024-08-20

- Исправлен парсинг gradle-dependency-tree kotlin

#[2024.32.0] - 2024-08-09

- Добавлен анализ стандартных библиотек go в парсер в окружении go (`--go-resolve`)
- Добавлена возможность указания лицензии при создании проекта
- Исправлена ошибка при парсинге `pom.xml`, который содержит переменные вида `xxx.xxx.xxx.xxx`
- Исправлен парсер `scala-dependency-tree.txt`
- Исправлена ошибка при сканировании SBOM без секции компонентов

#[2024.29.0] – 2024-07-19

- Добавлена выгрузка ссылок и CWE в формат sarif

#[2024.26.0] - 2024-06-24

- Добавлен парсинг в окружении npm
- Добавлен парсинг в окружении dotnet
- Добавили парсинг в окружении poetry
- Добавлен параметр запуска `--block-on-empty-result` (возвращает код 3 при пустом результате сканирования)
- Добавлен флаг `--python-version` для указания версии python в семействе манифестов pypi
- Исправлено построение графа зависимостей на паре `package.json` и `package-lock.json`
- Улучшен парсинг `project.assets.json`

#[2024.21.0] - 2024-05-24

- Улучшен парсинг yarn.lock
- Исправлен парсинг в окружении yarn

#[2024.17.0] – 2024-04-27

- Добавлена сборка Johnny для Mac с процессорами Intel
- Исправлен парсер scala-dependency-tree

#[2024.15.0] – 2024-04-11

- Добавлена поддержка выгрузки результата сканирования в формате CSV
- В выгрузку результата сканирования добавлен путь до исходного файла, в котором нашлась зависимость
- Улучшен поиск .net пакетов при сканировании образов

#[2024.13.0] – 2024-03-28

- Добавлена поддержка выгрузки результатов сканирования в форматах SARIF и XML

#[2024.10.2] – 2024-03-07

- Исправлено объединение lock-файлов с манифестами на Windows

#[2024.9.0] – 2024-02-29

- Исправлено падение при парсинге go.sum

#[2024.7.0] – 2024-02-12

- Уменьшен размер Docker-образа с агентом
- Исправлена ошибка при хэшировании пустых файлов

#[2024.5.0] – 2024-01-31

- Добавлена поддержка Scala
- Добавлено разрешение зависимостей в go окружении (`--go-resolve`)
- Добавлено разрешение зависимостей в maven окружении (`--maven-resolve`)
- Добавлен разрешение зависимостей в yarn окружении (`--yarn-resolve`)
- Улучшены сообщения об ошибках в параметрах запроса
- Добавлены переменные URL (`cli.api_url`) и TOKEN (`cli.api_token`) платформы в конфиг
- В summary теперь считается количество уязвимостей, а не пакетов
- Увеличена ширина таблиц, при невозможности определения ширины терминала

#[2023.49.0] – 2023-12-08

- Добавлена поддержка разбора манифестов Rust `cargo.lock` и `cargo.toml`
- Добавлен параметр `--no-recursion` для выключения рекурсивного скана команды scan dir

#[2023.48.0] – 2023-11-22

- Добавлена настройка формата вывода таблицы с результатами `-f --format` (с возможностью отключения цветов)
- Добавлена настройка группировки уязвимостей в выводе `-g --group-vulnerabilities-by`
- Добавлена настройка сортировки уязвимостей в выводе `-s --sort-vulnerabilities-by`
- Добавлена настройка ограничения по времени ожидания анализа `-t --timeout`

#[2023.43.0] – 2023-10-27

- В консольный вывод добавлена сводная информация о степени критичности уязвимостей
- Исправлен разбор манифестов `.gradle.kts`

#[2023.38.0] - 2023-09-20

- Улучшен разбор манифестов package.json и composer.json

#[2023.35.0] - 2023-08-31

- Улучшен парсинг поля environment для манифестов `Gemfile` и `Gemfile.lock`
- Убрано автоматическое объединение ячеек с одинаковым значением CVSS в таблице с уязвимостями

#[2023.33.0] - 2023-08-17

- Оптимизирован вывод таблиц в консоль на малых экранах

#[2023.30.0] - 2023-07-27

- Добавлен парсинг `conanfile.py` для Conan
- Добавлена индикация активного процесса анализа в виде progress bar
- Добавлено табличное отображение для вывода алертов и уязвимостей в консоль
- Унифицирована обработка слэша в конце строки для команды `scan dir`

#[2023.27.0] - 2023-07-06

- Исправлен panic при анализе некоторых Go проектов
- Исправлено сканирование образов в части некорректного определения компонентов, которые не являются зависимостями

#[2023.26.0] - 2023-06-30

- Улучшен парсинг `gradle-dependency-tree` в части работы со строками classPath
- Исправлен вывод Policy Alerts в консоль

#[2023.23.0] - 2023-06-08

- Добавлен парсинг разных версий формата `conan.lock`
- Исправлено сбрасывание признака парсера по достижении пустой строки в `conanfile.txt`
- Исправлен парсинг `yarn.lock`

#[2023.21.0] - 2023-05-23

- Добавлен параметр запуска `--scan-depth` для настройки глубины сканирования архивов
- Добавлен флаг `--scan-files` в команде scan image для сканирования файлов внутри docker-образа
- Улучшено определение вложенных зависимостей jar-пакетов
- Исправлен парсинг `Gemfile`

#[2023.15.0] - 2023-04-14

- Добавлен вывод Fixed version
- Добавлена возможность сохранения результатов сканирования
- Добавлена возможность создания проекта
- Добавлен поиск системных зависимостей в docker-образе
- Оптимизирован парсинг `package-lock` v3 манифеста для NPM
- Исправлены некоторые ошибки

#[2023.11.0] - 2023-03-16

- Добавлена поддержка консольных команд
- Улучшен разбор `pyproject.toml`
- Добавлена очистка /tmp директории после сканирования docker образа

#[2023.5.0] - 2023-02-01

- Добавлено сканирование docker-образов
- Изменено поведение при запуске без проекта
- Обновлен Golang до 1.19
- Исправлено хеширование в архивах с опцией `--only-hashes`

- Исправлено определение битых и запароленных архивов

#[2023.3.0] - 2023-01-20

- Исправлена ошибка при парсинге `gradle-dependency-tree`

#[2023.2.0] - 2023-01-13

- Добавлено сканирование архивов, флаг для запуска `--scan-archives`

#[2022.52.0] - 2022-12-30

- Исправлен парсинг `pom.xml` в части работы с секцией dependencyManagement

#[2022.50.0] - 2022-12-12

- Добавлена поддержка парсинга `conan.lock` файлов
- Исправлена передача дополнительных данных для резолвера из Nuget манифестов

#Nexus OSA Changelog

#[2026.19.0] - 2026-05-06

#Добавлено

- Добавлена Capability `CodeScoring Repository Mask Scan` для настройки сканирования репозиториев по regex-маске
- В Capability `CodeScoring All Repositories Scan` добавлено поле `Repository exclude masks (regex, comma-separated)` для исключения репозиториев из сканирования по regex-маскам

#[2026.8.0] - 2026-02-20

#Добавлено

- Добавлена обработка неблокирующего статуса `delayed_block`
- Добавлена настройка `Append block URL to custom message` для управления добавлением ссылки на блокировку в кастомное сообщение
- Добавлена поддержка conda-репозиториев

#Исправлено

- Исправлена проблема сохранения Capability `Configuration`: если поля `Pool Size` и `Timeout` пустые, после снятия флага

`Block downloads in case of plugin...` изменения теперь корректно сохраняются
- Исправлена проблема со сканированием maven пакетов, содержащих версию в artifactId

#[2025.32.0] - 2025-08-06

#Изменено

- Плагин теперь разрешает загрузку образов при несконфигурированном реестре (registry) в платформе CodeScoring, если флаг `Block downloads in case of plugin or CodeScoring errors` отключен.

#[2025.23.0] - 2025-06-02

- Добавлена опция выставить timeout на ожидание ответа от платформе в общей конфигурации плагина

#[2025.12.0] - 2025-03-17

- Повышена точность анализа в RPM и Debian репозиториях: улучшено определение namespace и квалификаторов дистрибутива и архитектуры в PURL
- В плагин добавлен механизм Circuit Breaker, предотвращающий деградацию производительности Nexus при недоступности или таймаутов со стороны платформы

#[2025.9.0] - 2025-02-25

- Исправлена ссылка на Docker образ, отправляемая плагином для анализа в платформу CodeScoring. Формат ссылки поменялся в релизе Nexus Repository 3.75

#[2025.5.2] - 2025-01-31

- Исправлена 500 ошибка при попытке скана неизвестного плагину формата репозитория

#[2025.5.1] - 2025-01-27

- Исправлено отсутствие namespace в PURL для NPM пакетов в релизе Nexus Repository 3.75

#[2025.5.0] - 2025-01-27

- Исправлена проблема с пробелами в списке игнорируемых имен репозиториев и в списке форматов репозиториев для скана

#[2024.49.0] - 2024-12-05

- Добавлена отправка в CodeScoring ссылки на пакет и пользователя, скачивающего его. Совместимо с версией платформы 2024.48.0 и выше
- Добавлена поддержка Nexus 3.75

#[2024.46.0] - 2024-11-15

- Добавлена поддержка PHP Composer репозиториев. Доступно при использовании community-плагина
- Обновлена поддержка совместимости плагина с разными версиями Nexus Repository:
  - для Nexus Repository OSS **3.71+** и Nexus Repository Pro версий с **3.33.1-01** по **3.71+** выпущена версия плагина с поддержкой H2 и PostgreSQL
  - для Nexus Repository OSS версий с **3.33.1-01** по **3.70.X** выделена legacy версия плагина с поддержкой OrientDB

#[2024.42.0] - 2024-10-16

- Добавлена возможность указывать формат репозиториев для анализа с помощью capability `All Repositories Scan`

#[2024.34.0] - 2024-08-19

- Добавлена поддержка Nexus 3.71

#[2024.28.0] - 2024-07-10

- Добавлено сканирование архивов в формате `.gem` для Ruby репозиториев

#[2024.26.0] - 2024-06-27

- Добавлена настройка `Append repository name to image name for Docker repositories` для возможности работы с подходом RepoPath для Docker registries
- Удалена возможность ручного запуска сканирования всего репозитория со стороны плагина (флаг `Run manual scan on save`)
- Настройка `Block downloads in case of plugin or CodeScoring errors` теперь учитывает падение сканирования (статус `blocked_scan_failed`)

#[2024.21.0] - 2024-05-24

- Добавлена Capability, которая активирует сканирование для всех репозиториев (`CodeScoring All Repositories Scan`)

- Добавлена возможность указания собственного сообщения о блокировке
- Добавлена ссылка на причину блокировки в свойства компонента
- Понижен уровень логирования для сообщения о пропуске сканирования репозитория
- Добавлена настройка URL менеджера репозиториев

#[2024.7.0] – 2024-02-17

- Добавлена ссылка на страницу компонента в CodeScoring в сообщении о блокировке
- Добавлено сообщение о блокировке для ситуаций, когда registry не добавлен в CodeScoring

#[2024.2.0] – 2024-01-12

- Расширен перечень архитектур образов
- Добавлена поддержка новых статусов блокировки компонентов

#[2023.48.0] – 2023-11-22

- Добавлен новый режим работы `spectator`
- Добавлена поддержка сканирования мультиплатформенных Docker-образов

#[2023.44.0] – 2023-10-31

- Добавлена поддержка сканирования proxy Docker репозиториев на базе Sonatype Nexus и JFrog Artifactory

#[2023.43.0] – 2023-10-27

- Добавлены различные режимы работы плагина

#[2023.40.0] - 2023-10-04

- Добавлена Capability для проверки образов из Hosted Docker Repository

#[2023.37.0] - 2023-09-11

- Исправлена ошибка сохранения результатов сканирования при отсутствии обязательных полей

#[2023.36.0] - 2023-09-05

- Добавлена опция блокировки сборки при получении ошибки взаимодействия с платформой CodeScoring

#[2023.33.0] - 2023-08-15

- Добавлена опция сохранения результатов сканирования артефакта в базу Nexus
- Добавлен метод в API для извлечения результатов сканирования артефакта

#[2023.26.0] - 2023-06-30

- Добавлена поддержка сканирования open-source компонентов в hosted репозиториях

#[2023.6.2] - 2023-02-09

- Добавлена настройка прокси-сервера в конфигурации плагина

#[2023.6.0] - 2023-02-07

- Добавлен параметр HTTP Client Connection Pool Size в конфигурации плагина для управления количеством запросов к платформе

#[2023.4.0] - 2023-01-24

- Улучшен парсинг компонентов RubyGems

#[2022.51.0] - 2022-12-20

- Улучшено логирование ответов от платформы

#JFrog plugin changelog

#[2026.16.0] - 2026-04-14

#Добавлено

- Добавлена возможность настройки параметров сканирования репозиториев по маскам через регулярные выражения (`repositoryMasks` и `excludeRepositoryMasks`). Подробнее в документации

#Исправлено

- Устранена проблема совместимости с рядом версий JFrog Artifactory, в том числе с 7.77.x, связанная с некорректной работой ClassLoader в этих версиях

#[2026.8.0] - 2026-02-20

#Добавлено

- Добавлена поддержка conda-репозиториев
- Добавлена поддержка conan-репозиториев
- Добавлена поддержка swift-репозиториев

#Изменено

- Изменено применение политик в зависимости от типа репозитория, в который пришел запрос (virtual/remote): в запросе к платформе теперь отправляется фактическое имя репозитория

#[2026.4.0] - 2026-01-20

#Исправлено

- Исправлена ошибка совместимости с JFrog Artifactory версии 7.125.x и выше

#[2025.43.0] - 2025-10-23

#Добавлено

- Реализован неблокирующий статус `delayed`

#Исправлено

- Исправлена ошибка с двойными запросами для незаблокированных пакетов
- Исправлен некорректный переход Circuit Breaker в состояние OPEN при тихом закрытии соединений сервером из пула подключений плагина

#[2025.32.0] - 2025-08-06

#Изменено

- Плагин теперь разрешает загрузку docker образов при несконфигурированном реестре (registry) в платформе CodeScoring, если в конфигурации флаг `blockOnErrors` установлен в `false`

#[2025.12.0] - 2025-03-17

- Повышена точность анализа в RPM и Debian репозиториях: улучшено определение namespace и квалификаторов дистрибутива и архитектуры в PURL

#[2025.11.0] - 2025-03-13

- В плагин добавлен механизм Circuit Breaker, предотвращающий деградацию производительности Artifactory при недоступности или таймаутах со стороны платформе

#[2024.49.0] - 2024-12-05

- Добавлен новый метод `codeScoringVersion`, позволяющий узнать версию плагина. Для этого выполните команду:
  `curl http://localhost:8082/artifactory/api/plugins/execute/codeScoringVersion` .
  Если версия не отображается, перезагрузите JFrog

#[2024.48.0] - 2024-11-29

- Добавлена отправка в CodeScoring ссылки на артефакт и пользователя, скачивающего его. Совместимо с версией платформе 2024.48.0 и выше
- Добавлена поддержка JFrog Artifactory v7.90+
- Для deb пакетов улучшено получение информации в случае использования нестандартных разделителей в названии файлов
- В логах визуально выделена инициализация плагина и перезагрузка конфига

#[2024.42.0] - 2024-10-16

- Пакеты, не содержащие версию, теперь не отправляются на анализ в CodeScoring и будут пропущены плагином
- Исправлен парсинг версий deb для альтернативных разделителей в формате `package-version-architecture.type`

#[2024.39.0] - 2024-09-26

- Добавлена настройка сканирования выбранного типа репозитория `repositoryTypes` (maven, npm, etc)
- Исправлено падение в версии Artifactory 7.49

#[2024.33.0] - 2024-08-12

- Добавлена поддержка `cargo` и `composer` репозиториев
- Добавлен флаг для сохранения результатов сканирования в свойства артефакта

#[2024.28.0] - 2024-07-10

- Добавлено сканирование архивов в формате `.gem` для Ruby репозиториев

#[2024.26.0] - 2024-06-28

- Добавлена возможность подключения плагина ко всем поддерживаемым репозиториям без необходимости их перечисления (`scanAllRepositories`)
- Добавлена возможность исключать репозитории из списка подключенных через опцию `scanAllRepositories`
- Добавлен флаг `deleteBlocked` для удаления компонента, если он заблокирован политиками
- Добавлена установка свойств компонента: дата сканирования, причина блокировки, ссылка на страницу пакета
- Детализировано описание конфигурационного файла
- Улучшено логирование при пропуске скана
- Флаг `blockOnErrors` теперь учитывает падение сканирования (статус `blocked_scan_failed`)

#[2024.11.0] – 2024-03-15

- Добавлена ссылка на страницу компонента в CodeScoring в сообщении о блокировке
- Добавлено новое сообщение о блокировке компонента в ситуациях, когда registry не добавлен в CodeScoring
- Добавлена настройка для работы с Docker registry Repository path (`stripRepoNameInDockerImageName`)
- Улучшено определение имени и версии артефакта для PyPI, NPM, Debian и Alpine репозиториев
- Исправлена ошибка сканирования для незакэшированных образов в remote docker репозиториях

#[2024.5.0] – 2024-02-02

- Добавлена обработка zip-архивов в golang репозиториях
- Улучшена работа с debian пакетами
- Исправлена ошибка проверки alpine пакетов
- Исправлено сканирование артефактов в virtual репозиториях

#[2024.2.0] – 2024-01-12

- Расширен перечень архитектур образов
- Добавлена поддержка новых статусов блокировки компонентов
- Добавлено логирование тела запроса на загрузку компонента и ответа
- Исправлена ошибка блокирования пакетов в debug режиме

#[2023.50.0] – 2023-12-15

- Добавлен новый режим работы `spectator`
- Добавлена поддержка сканирования мультиплатформенных Docker-образов
- Файл для настройки плагина теперь использует формат `.yaml`

- Добавлена возможность указания режима работы на каждый репозиторий отдельно
- Добавлена возможность изменения значений по умолчанию в файле для настройки
- Добавлен параметр HTTP Client Connection Pool Size для управления количеством запросов к платформе
- Добавлен параметр выключения плагина

#[2023.48.0] – 2023-11-23

- Добавлен вывод конфига в лог при старте плагина

#[2023.43.0] – 2023-10-27

- Добавлены различные режимы работы плагина
- Добавлена поддержка сканирования local/remote Docker репозиториев

#[2023.28.2]

- Добавлен префикс `CodeScoring:` ко всем логам

#[2023.28.1]

- Понижен уровень всех ошибок взаимодействия с платформой CodeScoring до `[INFO]`

#[2023.28.0]

- Исправлена ошибка с неработающим флагом `blockDownloads` в properties

#[2023.27.0]

- Добавлен флаг `blockDownloads` в properties, который позволяет контролировать загрузку пакета при наличия ошибок CodeScoring API или плагина

#[2023.26.0]

- Исправлено сохранение properties пакета при его скачивании из virtual-репозитория

#[2023.22.0]

- Исправлена ошибка с обязательным полем `project_name` в OSA API

#[2023.21.0]

- Изменен формат поставки плагина: теперь он поставляется в виде jar-файла
- Изменена инициализация плагина: вместо перезагрузки на каждый запрос происходит единоразовая инициализация при запуске

#Changelog плагина для Сфера.Дистрибутивы и лицензии

#[2026.7.0] - 2026-02-13

#Исправлено

- Исправлена передача тега вместо sha256 в PURL во время использования команды `docker pull`

#[2025.52.0] - 2025-12-30

#Добавлено

- Первый релиз плагина для Sfera
- Поддержка следующих форматов репозиториев:
  - Maven
  - NPM
  - NuGet
  - PyPI
  - RubyGems
  - Go
  - APT (Debian)
  - YUM (RPM)
  - Docker

#Visual Studio Code Extension Changelog

#[2026.8.0] - 2026-02-18

#Добавлено

- Добавлено окно просмотра алертов по политикам

#Исправлено

- Исправлены ошибки автоматического применения исправлений

#[2025.33.0] - 2025-08-14

- Расширение умеет осуществлять сканирование всех поддерживаемых платформой манифестов, показывать информацию по уязвимостям, предлагать автоматическое обновление на fixed versions, а также предоставляет инструменты для работы со SBOM в формате CycloneDX. Первая версия расширения поставляется в виде .vsix файла

#IntelliJ Plugin Changelog

#[2026.8.0] - 2026-02-18

#Добавлено

- Добавлено окно просмотра алертов по политикам

#Исправлено

- Исправлены ошибки автоматического применения исправлений
- Исправлены ошибки совместимости с IntelliJ версий 2024 и ранее

#[2025.33.0] - 2025-08-14

- Плагин умеет осуществлять сканирование всех поддерживаемых платформой манифестов, показывать информацию по уязвимостям, предлагать автоматическое обновление на fixed versions, а также предоставляет инструменты для работы со SBOM в формате CycloneDX. Первая версия плагина поставляется в виде .zip файла.

#OSA Proxy Changelog

#[2026.22.0] - 2026-05-27

Note

Перечисленные ниже новые возможности доступны только в Go-реализации OSA Proxy.

#Добавлено

- Добавлена новая реализация OSA Proxy на Go. Подробнее в документации
- Добавлена поддержка Composer/Packagist-репозиториев: проверка metadata, проверка dist-архивов и переписывание dist URL через OSA Proxy Go. Подробнее в документации
- Добавлена поддержка RubyGems-репозиториев: проверка metadata RubyGems и скачиваемых `.gem`-пакетов. Подробнее в документации

- Для Docker Registry API v2 в Go-реализации добавлена маршрутизация нескольких Docker-репозиториев по host/subdomain: имя поддомена соответствует `repository[*].name`, а запросы Docker-клиента остаются на стандартных `/v2/...` и `/token`. Подробнее в документации
- Добавлена фильтрация типов файлов на уровне репозитория через `repository[*].file-type-filter` для ограничения артефактов, которые отправляются на пакетное сканирование.

#Изменено

- В Go-реализации метрики Prometheus доступны напрямую по `/metrics`; Spring Boot Actuator endpoints `/actuator/metrics` и `/actuator/prometheus` не используются.

#[2026.13.2] - 2026-04-30

#Добавлено

- Добавлена поддержка нескольких реестров пакетов для PyPI через параметр `additional-packages-registries`: позволяет указывать дополнительные источники пакетов с автоматической маршрутизацией по хосту. Актуально для репозиториев, отдающих пакеты с разных хостов (например, PyTorch). Подробнее в документации

#Исправлено

- Исправлена обработка WHL-индексов PyPI без версий пакетов (например, поддиректории `/whl/cu124/`): такие страницы больше не передаются на сканирование в CodeScoring
- Исправлена ошибка, при которой прокси не следовала по HTTP-редиректам при загрузке пакетов вне режима отладки

#[2026.13.1] - 2026-04-16

#Добавлено

- Добавлена возможность настройки режима обработки заблокированных версий в манифестах npm, PyPI и NuGet через параметр `codescoring.remove-blocked-versions` (по умолчанию: `true`): при `true` версии полностью удаляются из манифеста, при `false` — помечаются как устаревшие (`deprecated`) с указанием сработавшей политики
- Добавлена поддержка параметров трассировки через Micrometer Tracing — идентификатор trace теперь отображается в логах
- Пропуск сканирования attestation-манифестов Docker: манифесты типа `attestation-manifest` больше не передаются на сканирование в CodeScoring

#Исправлено

- Исправлена ошибка `Registry path cannot be blank` при проксировании Go-пакетов с пустым путём реестра

#[2026.13.0] - 2026-04-02

#Исправлено

- Исправлена установка пакетов из поддиректорий PyPI-репозитория PyTorch (`/whl/cu124/`, `/whl/cu130/` и других)
- Из HEAD-запросов теперь удаляются кэширующие заголовки (`Last-Modified` и др.). Это устраняет проблему, при которой JFrog Artifactory считал манифест актуальным и не запрашивал его повторно — теперь обновленный манифест с применёнными политиками загружается своевременно

#Добавлено

- Добавлена поддержка `sum.golang.org` для проксирования Go-модулей. Подробнее в документации

#Изменено

- Вместо удаления уязвимых версий из манифестов npm, PyPI и NuGet они теперь помечаются как устаревшие (`deprecated`). Поведение при установке пакетов с незафиксированными версиями сохраняется прежним, при этом добавляется ясность по заблокированным версиям — в манифесте отображается сработавшая политика

#[2026.5.3] - 2026-03-02

#Исправлено

- Исправлена проблема, при которой загрузка пакета блокировалась даже при включенном флаге отключения блокировки в случае ошибок; некорректное поведение наблюдалось при ответе `blocked_scan_failed`
- Исправлена ошибка при обращении к `jitpack.io`

#Изменено

- Унифицировано логирование данных о заблокированных версиях в манифестах и заблокированных загрузках пакетов: теперь используется единая точка логирования `ru.codescoring.proxy.logging.PolicyLogger`

#[2026.5.2] - 2026-02-20

#Добавлено

- Добавлена поддержка репозитория `https://download.pytorch.org`

#[2026.5.1] - 2026-02-11

#Добавлено

- Реализовано проксирование `docker.io` через команду `docker pull` без префикса `library`

#Изменено

- Заменена команда KEYS на SCAN для поиска ключей в Redis, чтобы Redis работал в режиме ограниченных привилегий (`+@all -@dangerous`)
- Расширено логирование анализа манифестов: теперь фиксируются заблокированные версии пакетов и применённые политики блокировки

#Исправлено

- Отключена передача заголовков кеширования для HEAD-запросов, таким образом ускоряя обновление манифестов в JFrog Artifactory и применение политик

#[2026.5.0] - 2026-01-30

#Добавлено

- Добавлены метрики обращений в CodeScoring
- Добавлена возможность подменять URL в ссылке на причину блокировки на указанный в конфиге `OSA-Proxy` `codescoring.host`. Подробнее в документации
- Добавлена поддержка APK и RPM репозиториев
- Добавлена поддержка настройки `JAVA_OPTS` в Helm-чарте. Подробнее в документации
- Добавлена поддержка Docker-репозиториев

Особенности загрузки официальных образов

Загрузка официальных образов без префикса `library` (например, `docker pull <osa-proxy-host>/postgres`) будет добавлена в следующем патч-релизе. На данный момент необходимо использовать полный путь: `docker pull <osa-proxy-host>/library/postgres`

#Изменено

- Изменены метрики `gateway_route_<package_type>_requests_seconds` с Histogram на SLO (10ms, 25ms, 50ms, 100ms, 250ms, 500ms, 1s, 2s, 5s)

#Исправлено

- Исправлена работа с удаленными репозиториями за **Cloudflare**, приводившая к коду ошибки 1101 от Cloudflare
- Исправлена ошибка с добавлением репозитория к имени пакета в PURL для Go

#Удалено

- Удалена дублирующая `gateway_route_` метрика `http_server_requests`

#[2025.52.2] - 2026-01-19

#Исправлено

- Исправлено отображение пустых путей в дереве зависимостей, приводившее к некорректной работе в NuGet-cli. Если версии компонентов заблокированы, соответствующие им родительские узлы теперь также удаляются

#[2025.52.1] - 2026-01-15

#Исправлено

- Исправлена ошибка проверки состояния (healthcheck) Redis в случае, если кэш отключён

#[2025.52.0] - 2025-12-23

#Добавлено

- Добавлено кэширование результатов проверки политик в Redis с поддержкой TTL и фонового обновления. Подробнее в документации
- Добавлен механизм проактивного обновления кэша по таймеру
- Добавлен Swagger UI для документации API (`/api/swagger`)
- Добавлен REST API для управления кэшем (`/api/cache`)
- Добавлено обновление заголовков (`Last-Modified`, ETag) для манифестов. Это решает проблему "устаревших" манифестов, с точки зрения политик.

#Изменено

- Увеличено значение по умолчанию для переменной конфигурации `max-in-memory-size` с 50MB до 150MB (для обработки больших манифестов)
- Оптимизирована обработка манифестов NPM с использованием потокового процессора для повышения скорости обработки больших файлов и снижения нагрузки для сборщика мусора

#[2025.48.3] - 2025-12-08

#Исправлено

- Устранена утечка ресурсов, которая проявлялась непосредственно перед циклом сборки мусора (GC)
- Исправлено поведение, приводившее к некорректному срабатыванию ошибки `Connection prematurely closed BEFORE response` в случаях, когда соединение было закрыто на стороне сервера или обратного прокси

#[2025.48.2] - 2025-12-02

#Исправлено

- Исправлена ошибочная блокировка пакетов, имеющих дату публикации при активной политике `Дата публикации зависимости отсутствует`

#[2025.48.0] - 2025-11-28

#Исправлено

- Исправлено применение политик безопасности для версий пакетов, данные о которых отсутствуют в CodeScoring Index на момент сканирования

#[2025.47.0] - 2025-11-19

#Добавлено

- Реализована поддержка пакетов Go и Debian
- Введена обработка статуса DELAYED
- Расширена функциональность прокси-сервера для передачи контекста через URL, что обеспечивает применение политик, привязанных к репозиториям, в конфигурации `jfrog/nexus -> OSA proxy -> internet`. Подробнее в документации.

- При запросе манифеста, в случае блокировки всех версий пакета, теперь отображается список соответствующих блокирующих политик

#Исправлено

- Устранена ошибка, приводившая к неприменению политик для репозиториев в конфигурации
  `OSA proxy -> jfrog/nexus -> internet`

#[2025.39.2] - 2025-10-22

#Добавлено

- Внедрены метрики для каждого типа пакетного менеджера, доступные по `gateway.route.[maven|npm|nuget|pipy].requests`
- При полной блокировке всех версий пакета в манифесте соответствующими политиками сервис теперь возвращает корректный статус ошибки и причину блокировки вместо пустого манифеста. На текущий момент только `nuget` корректно обрабатывает и отображает данную ошибку в консоли, остальные пакетные менеджеры сообщают, что подходящей версии не найдено
- Реализовано логирование предупреждений (`Warn`) в случае некорректной конфигурации, если манифест не содержит ссылок на пакеты, использующие URL из `repository.registry`. В редких случаях возможны ложно-положительные сообщения в логах, которые будут устранены в случае обнаружения

#Исправлено

- Устранена ошибка, возникавшая при запросах JFrog Artifactory к актуальному индексу `pypi` пакетов по маршруту `/simple/`
- В режимах `warmup` & `spectator` блокировка версий в манифестах больше не производится

#[2025.39.1] - 2025-09-23

#Сканирование пакетов

Два уровня сканирования:

- **Манифесты**: анализ и исключение заблокированных версий.
- **Пакеты**: анализ загружаемых файлов.

#Блокировка небезопасных компонентов

- Заблокированные политиками версии исключаются из манифеста.
- Загрузка заблокированных политиками архивов блокируется.
- Возвращается настраиваемый код состояния с сообщением о блокировке в status-line.

#CodeScoring.Save Changelog

#[2026.21.0] - 2026-05-26

#Добавлено

- Первый релиз CodeScoring.Save — менеджера репозиториев артефактов с proxy- и hosted-режимами, web UI, RBAC, cleanup-политиками, аудитом и метриками, а также с интеграцией с OSA.Proxy
- Поддержка форматов: Maven, npm, NuGet, PyPI, Go, Docker / OCI, raw
- Развёртывание вместе с OSA.Proxy через Helm-чарт

Подробнее — в разделах Функциональные характеристики, Архитектура и Управление репозиториями.