#Full Documentation

#Functional characteristics

#General functional characteristics

The modular platform for secure development **CodeScoring** provides solutions for the problems of secure use of **open source**, identification of secrets in code and assessment of development quality.

The platform includes the following modules:

- **CodeScoring.Save** – artifact management with proxy and hosted repositories, security check integration, and access control;
- **CodeScoring.OSA (Open Source Analysis)** – supply chain protection: verification of third-party components for compliance with security policies and blocking in case of non-compliance;
- **CodeScoring.SCA (Software Composition Analysis)** – compositional analysis: auto-detection of **open source** dependencies, identification of vulnerabilities and malicious components, verification of license compatibility, formation of a list of software components and construction of a dependency graph;
- **CodeScoring.Secrets** – search for secrets in code: orchestration of source code checks for confidential information (**secrets**) and assessment of truth using our own machine learning model;
- **CodeScoring.TQI (Teams & Quality Intelligence)** – assessment of code quality through calculation of key quality metrics, tracking development dynamics and building profiles of development participants with confirmed competence in projects.

The platform comprehensively provides:

- analysis of 20 programming languages (the list of languages may vary for individual modules): **Java**, **Kotlin**, **JavaScript**, **TypeScript**, **Python**, **C**, **C++**, **Go**, **PHP**, **Ruby**, **C#**, **Objective-C**, **Swift**, **Rust**, **Scala**, **Bash**, **Perl**, **SQL**, **PLSQL**, **PGSQL**;
- analysis of manifests of packaging ecosystems: **Maven**, **Gradle**, **Apache Ivy**, **NPM**, **Yarn**, **pnpm**, **NuGet**, **Paket**, **pip**, **Poetry**, **Pipenv**, **Conda**, **Ruby Gems**, **Cocoapods**, **Swift**, **Composer**, **Conan**, **Go Modules**, **Cargo** and **sbt**;
- integration with repository managers and container image registries: **Nexus Repository Manager**, **JFrog Artifactory Pro**, **Harbor**, **GitLab**, **GitFlic**, **Sfera.Distributions and Licenses**;
- repository management through **CodeScoring.Save** with proxy and hosted modes for **Maven**, **npm**, **NuGet**, **PyPI**, **Go**, **Docker/OCI**, and **raw**;
- integration with **hosted** and **cloud** code hosting systems: **GitFlic**, **GitLab**, **Github**, **Bitbucket** and **Azure DevOps**;

- universal integration into build systems using an agent: **Jenkins**, **GitLab**, **GitFlic**, **TeamCity** and others;
- management of organizational security policies;
- automation of scheduled scan launches;
- integration for email notifications;
- integration with the **Jira** task management system;
- user management system;
- **LDAP** integration;
- creation of webhooks;
- differentiation of access rights for different users;
- logging of all performed operations (**audit log**);
- collection of metrics on performed operations;
- open and documented programming interface (**API**).

#Functional characteristics of the CodeScoring.Save module

- artifact storage with two repository types:
  - **proxy repositories** for forwarding requests to upstream repositories, caching artifacts, and running security checks;
  - **hosted repositories** for storing internal organization artifacts;
- support for package and artifact ecosystems:
  - **Maven** (Java, Kotlin, Scala, and other JVM tooling using Maven layout);
  - **npm** (JavaScript / TypeScript);
  - **NuGet** (.NET);
  - **PyPI** (Python);
  - **Go** (Go Modules);
  - **Docker / OCI** (container images, Helm OCI charts, and oras artifacts);
  - **RAW** (arbitrary files);
- service separation:
  - **Save API service** for artifact and repository operations;
  - **Auth / RBAC service** for authentication and authorization;
- security check integration:
  - artifact scanning through **CodeScoring.OSA**;
  - policy checks when components are downloaded;
  - blocking artifacts that do not comply with configured policies;
- access control:
  - role-based access control (**RBAC**) with **global / project / repository** scopes;
  - user and robot account management;
  - project membership management with granular permissions;
  - **JWT** issuing and validation for API requests with local JWKS signature verification in Save;
  - support for Basic Auth, Basic Auth with `sa$` robot account prefix, Bearer JWT, Docker v2 JWT, `X-NuGet-ApiKey`, and opaque Bearer token for npm clients;
  - **OCI Distribution Spec / Docker Registry v2** support with short-lived access tokens;
  - audit log for access and artifact management operations;

- organization model:
  - **project → repository → artifact** hierarchy;
  - project management and member assignment;
  - access permissions at project and repository levels;
- artifact lifecycle management:
  - **cleanup policies** for automatic deletion of outdated or unused artifacts;
  - cleanup rules by age, number of versions, last download date, pattern, and size;
  - dry run / preview before executing a cleanup rule;
- web interface for:
  - creating and configuring projects and repositories;
  - viewing and downloading artifacts and metadata;
  - managing users, roles, and access permissions;
  - configuring cleanup policies;
  - viewing configured parameters;
  - monitoring operations and exporting audit events;
- deployment options:
  - **hosted installation** on **k3s** for small teams;
  - **basic installation** on **Kubernetes** with **PostgreSQL** and **S3-compatible storage** for production use;
  - storage configuration with **SQLite/PostgreSQL** for metadata and **S3/file system** for artifacts;
- operation and observability:
  - stateless architecture for horizontal scaling;
  - in-pod cache and local Bearer JWT validation through JWKS;
  - streaming processing of large files without full in-memory buffering;
  - metrics export to **Prometheus** in OpenTelemetry format;
  - structured JSON logging;
  - health checks for service status;
- API:
  - REST API for managing projects, repositories, and artifacts;
  - compatibility with package manager protocols: Maven 2 layout, npm Registry API, NuGet v3, PyPI Simple PEP 503/691, Go Module Proxy, and OCI Distribution Spec;
  - API for users, roles, robot accounts, and access permissions;
  - internal APIs for Save and cs-auth service-to-service interaction protected by shared secret.

#Functional characteristics of the CodeScoring.OSA module

- blocking the download of unwanted components when trying to download them through the command interface of the package manager or the web interface;
- forming a list of software components (**SBOM**, Software Bill of Materials);
- checking artifacts in the registries of container images;
- proxying and control of package downloads via the OSA Proxy service for popular package managers (Maven, NPM, PyPI, NuGet);

- archive analysis, including:
  - parsing the contents of popular archive formats (**zip**, **jar**, **tar**, **war**, **tgz** and others);
  - analysis of system packages, including support for popular package managers:
    - **DEB** (Debian, Ubuntu);
    - **RPM** (RHEL, CentOS, Fedora);
    - **APK** (Alpine Linux);
- setting up security criteria (**policies**) by 40 criteria, including:
  - package metadata: name, version, package author, age, date;
  - vulnerability criteria: identifier, **CVSS** score, threat level, publication date, age, presence of exploit, impact;
  - license categories and types, license compatibility (**license purity**);
  - package usage indicators: directive/transitive and environment (**scope**);
  - presence of malicious code;
  - package blacklists, including a proprietary **protestware** feed;
- detection and deduplication of vulnerabilities from 20 knowledge bases, including aggregation (**NVD**, **OSV**, **GHSA** and others), ecosystem (**Debian**, **RPM**, **Alpine** and others), commercial (**Kaspersky OSS Threats Data Feed**) and proprietary, including data on **protestware**;
- component management in the interface, including:
  - viewing and filtering the list of packages;
  - viewing and filtering the list of container images;
  - viewing and filtering component requests;
- export of reports.

#Functional characteristics of the CodeScoring.SCA module

- analysis of package manager manifests;
- formation of a list of software components (**SBOM, Software Bill of Materials**);
- checking **open source** at all stages of the development cycle with the ability to set security policies for individual stages of the development cycle:
  - code checking in development environments (**IDE**): **IntelliJ-based**, **VSCode**, **OpenIDE**;
  - checking on the developer's local machine (**CLI agent**);
  - continuous code monitoring (scanning repository branches);
  - checking in the **CI pipeline** with a universal agent with the ability to block the build;
  - post-release monitoring **SBOM**;
  - post-release code monitoring (scanning repository tags);
- detection of **open source** dependencies:
  - by ecosystem, package name and version;
  - by configuration files (**manifests**) of package managers;
  - via the mechanism for resolving transitive dependencies;
  - via the mechanism for identifying **open source** inclusions in the vendor metadata base, using flexible search algorithms;

    - via assembly analysis for the **C** and **C++** languages;
    - separation of directive and transitive dependencies and determination of the relationship to the development environments (**scope**): **runtime**, **compile**, **test**, **provided** and others;
- construction of a dependency graph with the ability to trace the use of transitive components;
- updating information about detected dependencies:
    - general information about the dependency: detection method; package release date; package author; official web page; link to placement in the package index; scope of use (**scope**) and link to the relationship graph; link to the manifest;
    - vulnerability information;
    - license information;
    - usability in the organization's projects;
- provision of vulnerability information:
    - identifiers in knowledge bases;
    - vulnerability description;
    - severity assessment according to **CVSS 2**, **CVSS 3.1**, and **CVSS 4.0**;
    - links to additional vulnerability indices;
    - recommended package version for update;
    - list of all projects affected by the vulnerability;
    - additional materials containing links to patches, exploits, and vulnerable commits;
- provision of license information:
    - project license landscape;
    - **SPDX** license identifier;
    - license text;
    - brief overview of the license terms;
    - information on license compatibility (**license compliance**);
- detection and deduplication of vulnerabilities from over 20 knowledge bases, including aggregation (**NVD**, **OSV**, **GHSA** and others), ecosystem (**Debian**, **RPM**, **Alpine** and others), commercial (**Kaspersky OSS Threats Data Feed**) and proprietary, including data on **protestware**;
- configuration of security policies by 40 criteria, including:
    - package metadata: name, version, package author, age, date;
    - vulnerability criteria: identifier, **CVSS** score, threat level, publication date, age, presence of exploit, impact;
    - license categories and types, license compatibility (**license purity**);
    - package use indicators: directive/transitive and environment (**scope**);
    - presence of malicious code;
    - package blacklists, including a prepared **protestware** feed;
- configuring security policies separately for scanning code in repositories, **CI pipeline** and the local developer machine, indicating the stage of the development cycle;
- a system for displaying scan results for each individual stage and the ability to manage ignored events;
- the ability to configure protection against popular attacks on the supply chain;

- the ability to configure temporary ignoring of triggering policies based on various criteria: project, technology, package, license type, vulnerability identifier, for individual policies;
- exporting reports in popular formats: **CycloneDX**, **SPDX**, **JUnit**, **SARIF**, **CSV**, **GitLab Dependency Scanning Report**, **GitLab Code Quality Report**;
- vulnerability reachability analysis.

#Functional characteristics of the CodeScoring.Secrets module

- detection of sensitive data in code, including passwords, **API keys**, tokens, credentials;
- use of a proprietary machine learning model to filter false positives;
- integration with version control systems (**VCS**) for automatic repository monitoring;
- customization of secret search criteria with the ability to configure detection rules;
- launching secrets analysis:
  - selection of scanning of a separate branch or the entire repository;
  - manual launch of analysis in individual projects;
  - general scanning of all projects of the organization;
  - uploading reports on secrets found by third-party tools to the system;
- providing detailed information on found secrets:
  - detection context (project, file, change author);
  - assessment of the probability of the truth of the secret;
  - date of secret detection and correction;
- management of found secrets:
  - viewing the list of found secrets;
  - confirmation of false positives and their exclusion from further scans;
  - marking secrets as resolved;
- managing the **ML model** training process to improve detection accuracy:
  - marking detected secrets by the user;
  - retraining the model on the marked data;
  - comparing the results of the retrained model with the original version;
  - ability to roll back to the base model;
- integration with **DevSecOps** processes:
  - work in the **CI/CD pipeline** with the ability to interrupt the build when secrets are detected;
  - analysis of secrets on the developer's local machine via the **CLI agent**;
- reporting and notification system:
  - displaying scan results at the level of the organization, project, and individual repository;
  - filtering results;
  - exporting reports with a list of secrets or their properties.

#Functional characteristics of the CodeScoring.TQI module

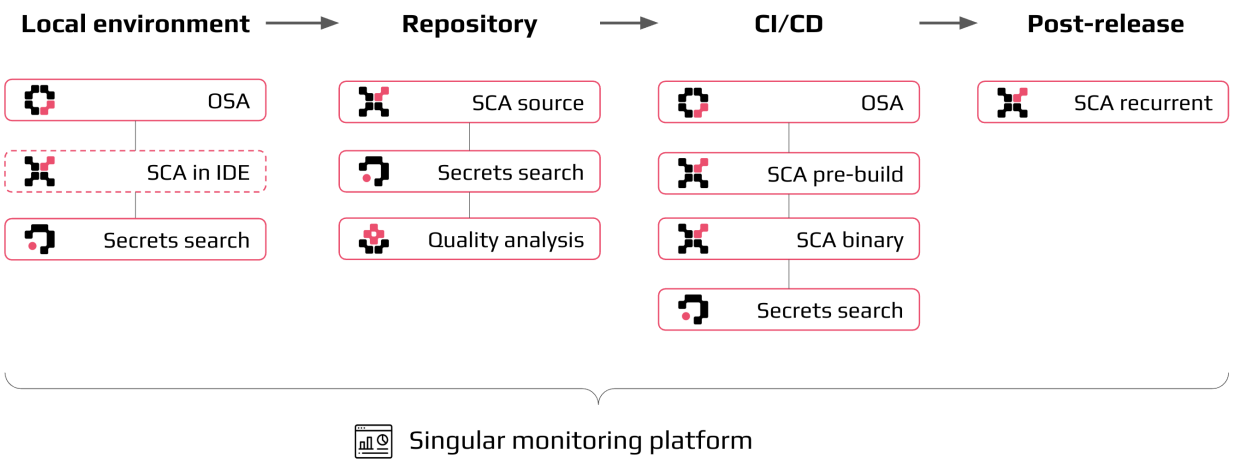
- building a project profile, detailing the technical and authoring composition of the development;
- building a developer profile, detailing their work in the analyzed projects taking into account the volume of work and the quality of the changes made;
- automatic assessment of the similarity of developers in the analyzed projects by experience of participation and technologies;
- searching for developer activity in **open source** projects;
- analysis of borrowings using fuzzy search algorithms (tolerance to renaming of objects in the code):
    - searching for intra-project code duplicates;
    - searching for cross-project code duplicates;
    - determining the volume of found borrowings;
    - building an interactive map of borrowings;
    - determining the time of borrowing;
    - linking authorship to found code duplicates;
- assessing the complexity of project maintenance:
- calculating the cyclomatic complexity of the code in the context of the project;
- calculation of cyclomatic complexity of code in terms of the author;
- construction of analytical retrospective maps:
    - development of projects;
    - work of authors;
    - evolution of project complexity;
- integration into the software development cycle (**SDLC**) through integration with source code versioning systems.

#Integration stages

The **CodeScoring** secure development platform integrates into the software development lifecycle, providing security control, policy compliance, and quality analysis at every key stage:

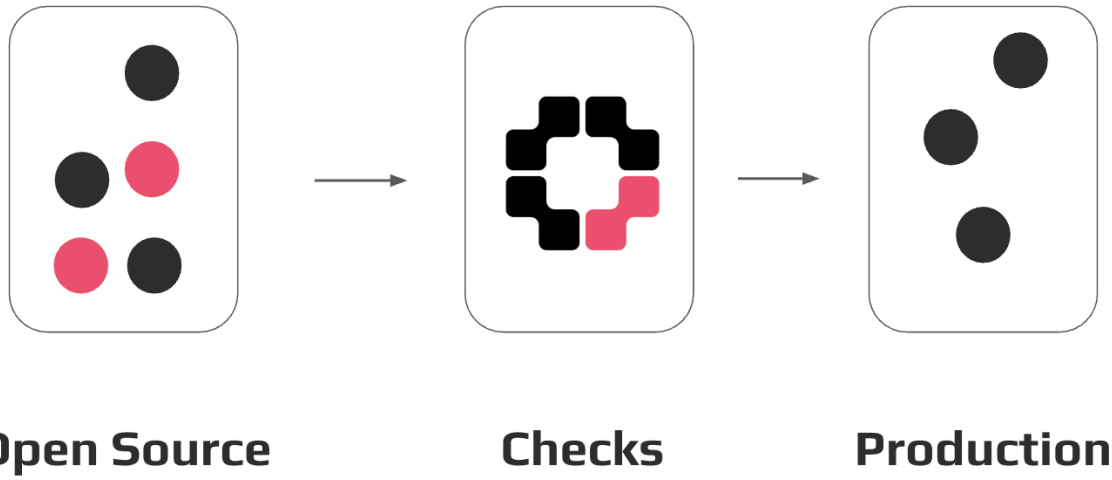
- local development;
- source code storage;
- software product assembly;
- post-release support.

The general integration scheme is presented below:



Important: the main platform functionality is listed by stages. The full list of capabilities is available on the functional characteristics page.

#Local development



During local development, CodeScoring helps prevent vulnerable or malicious components from getting into the codebase, as well as perform composition analysis and search for secrets using the CLI agent Johnny.

Functionality:

- analysis and blocking of open source components during installation from proxy repositories;
- inventory and registration of component requests;
- composition analysis;
- search for confidential information in the source code.

To implement blocking of components, the CodeScoring.OSA plugins for **Nexus Repository Manager** and **JFrog Artifactory PRO** are provided.

#Source code storage

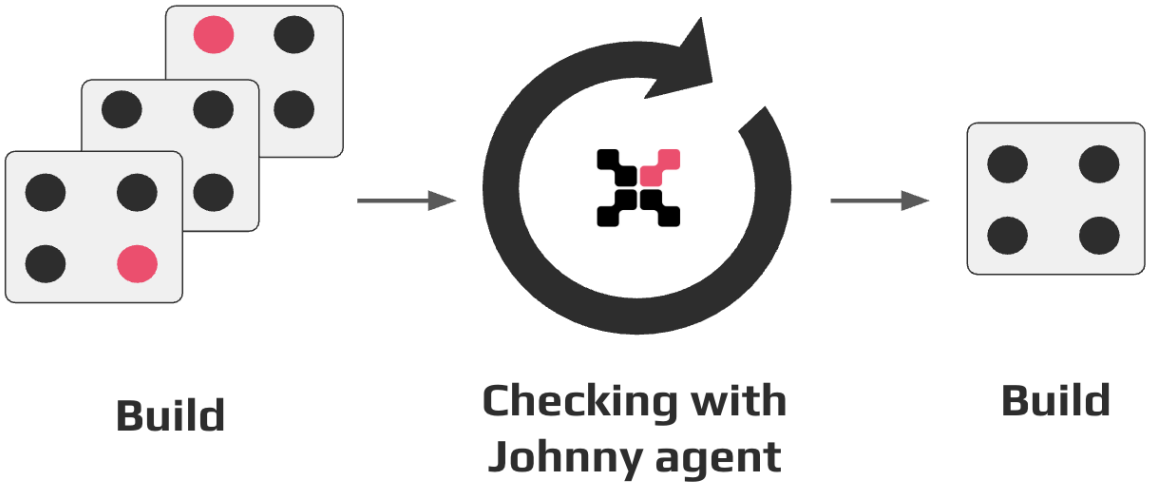
During the stage of storing and managing the source code, CodeScoring allows for continuous quality control and security of repositories.

Integration with major development platforms using git is supported: **GitFlic**, **GitHub**, **GitLab**, **Bitbucket**, **Azure DevOps**, etc.

Functionality:

- inventory of third-party components in repositories;
- detection of vulnerabilities and potentially dangerous components;
- search for secrets;
- analysis of development quality.

#Assembly in CI/CD pipeline



At the assembly stage, CodeScoring provides software analysis in the CI/CD pipeline and checks the artifacts used.

Supported automation tools: **GitLab CI/CD**, **Jenkins**, **Bitbucket Pipelines**, **TeamCity**, **Bamboo**, **GitFlic** and others.

Functionality:

- automatic generation of a software bill of materials (SBOM);
- detection of vulnerabilities and potentially dangerous components;
- analysis of license compatibility;
- control of the build's compliance with security policies.

The analysis is performed using the Johnny agent, available as a binary file or container image. If security policies are violated, the agent terminates

execution with the corresponding error code, which allows to stop the build before the unsafe artifact gets into the release.

#Post-release monitoring

After the product is published, CodeScoring provides continuous monitoring of the source code security and the composition of components. This allows to promptly respond to new vulnerabilities and threats in already released versions.

Functionality:

- periodic scanning of source code and SBOM;
- automatic updates of threat data;
- notifications via Email and task managers;
- integration with **ASPM/SIEM**-systems.

#Supported ecosystems and analysis methods

#Manifests

To find dependencies, CodeScoring primarily relies on parsing package manager manifest files. The platform supports parsing the following technologies:

Ecosystem	Package manager or build tool	File format

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

Ecosystem	Package manager or build tool	File format
	Maven	`pom.xml` `maven-dependency-tree.txt`
	Apache Ivy	`ivy.xml`

JavaScript and 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 and C++	Conan	`conanfile.txt`
		`conan.lock`
		`conanfile.py`

	GCC, Clang, CMake, Make etc.	Build scanning
Go	Go Modules	`go.mod` `go.sum`
PHP	Composer	`composer.json` `composer.lock`

Ecosystem	Package manager or build tool	File format

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

.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`

The best result will be achieved by combining the main manifest file and the corresponding lock file, if provided by the package manager mechanism.

#PURL Types and Components

For unified dependency description, CodeScoring uses the **Package URL (PURL)** standard.

PURL example

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

When analyzing SBOM via the agent command or importing into the platform, CodeScoring recognizes and supports the following PURL types according to the specification:

PURL type	Description	Specification
`cocoapods`	Libraries for **Objective-C** / **Swift** via CocoaPods	CocoaPods Definition
`conan`	Packages in the **C** / **C++ (Conan)** ecosystem	Conan Definition
`conda`	Packages in the **Python** / **Conda** ecosystem	Conda Definition
`nuget`	Components for **.NET** / **NuGet**	NuGet Definition
`golang`	**Go Modules** packages	Go Definition
`maven`	**Java** / **Kotlin** artifacts (Maven / Gradle)	Maven Definition
`npm`	**JavaScript** / **TypeScript** packages	NPM Definition
`composer`	**PHP (Composer)** packages	Composer Definition
`pypi`	**Python (PyPI)** packages	PyPI Definition
`gem`	**Ruby (RubyGems)** packages	RubyGems Definition
`cargo`	**Rust (Cargo)** packages	Cargo Definition
`generic`	General type for arbitrary binary or custom artifacts	Generic Definition
`apk`	**Alpine Linux** system packages	APK Definition
`deb`	**Debian** / **Ubuntu** system packages	DEB Definition
`rpm`	**RHEL** / **CentOS** / **Fedora** system packages	RPM Definition
`swift`	**Swift Package Manager** packages	Swift Definition
`oci`	**OCI** / **Docker** container images	OCI Definition
`docker`	**Docker Hub** / **Docker** images	Docker Definition
`github`	**GitHub** repositories	GitHub Definition

PURL type	Description	Specification
`huggingface`	**Hugging Face Hub** models	HuggingFace Definition
`mlflow`	**MLflow Model Registry** models	MLflow Definition
`pub`	**Dart / Flutter (pub.dev)** packages	Pub Definition
`swid`	**SWID Tags (Software Identification Tags)**	SWID Definition

Each component with a PURL is classified by type, which CodeScoring recognizes when importing SBOM files. The type is indicated in the `type` field within the component description.

Difference between PURL and component types

A component type describes its functional role within a product—for example, a library, framework, or firmware. A PURL type, in turn, defines the ecosystem and source from which the component was obtained.

SBOM component example

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

Supported component types:

Component type	Description
library	A library, package, or third-party code module used in the project.
framework	An infrastructure or application framework containing a set of libraries.

Component type	Description
firmware	Executable binary or embedded software analyzed for the presence of third-party components.

#System packages

As part of the OSA module, the platform supports analysing system packages of the following formats:

- Debian;
- Alpine;
- RPM;
- Astra Linux;
- ALT Linux;
- RED OS.

#Archive scanning

CodeScoring.OSA plugins support archive scanning in the following formats:

Ecosystem	Archive format
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`
Debian	`.deb`, `.xz`, `.gz`
Yum	`.rpm`
Alpine	`.apk`
Docker	`.json`

Archive scanning with Johnny

The Johnny CLI agent also supports archive scanning. The list of supported formats is available on the Scanning archives page.

#Resolution mechanism in the absence of a lock file

If there is no lock file the system will try to resolve transitive OSS dependencies itself:

- Maven
  - for pom.xml and build.gradle format generation of `maven-dependency-tree` via the corresponding maven plugin
  - Maven version 3.8.8 and OpenJDK version 11 are used
- PyPi
  - generation of `poetry.lock` using the Poetry package manager
  - Python version 3.11.7 is used
- N.P.M.
  - generation of `yarn.lock` using the Yarn package manager
  - uses Node.js version 20.9.0
- Nuget
  - for csproj and sln format generation of `project.assets.json` using built-in nuget tools
  - uses .NET SDK version 8.0.404
- Packagist
  - generation of `composer.lock` using the Composer package manager
  - PHP version 8.2.26 is used
- Rubygems
  - generation of `Gemfile.lock` using the Bundler package manager
  - uses Ruby version 3.1.2p20

Generation of lock files by the system does not produce the best results in every case, since it often depends on the environment.

#Resolving dependencies in the build environment

Some package managers do not include transitive dependencies in their manifests by default. For high-quality compositional analysis when working with them, it is recommended to use a dependency resolution mechanism in the build environment.

When resolving dependencies in the environment, the agent checks for the absence of a lock file, independently launches the package manager or build tool, and generates a complete list of components taking into account the correct version of the build. The functionality is currently available for the following ecosystems:

- .NET
- Go
- Gradle
- Maven

- npm
- Poetry
- sbt
- yarn
- Conda

#Mechanism for searching dependencies using hashes

Search using hashes implies detection of direct inclusion of libraries in project's code. As part of this mechanism, all project files are hashed and these signatures are compared with known open source libraries.

Currently, hash searches occur for the following package manager indexes and the following file types:

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

Hashes of files whose size does not exceed 512 bytes **are not sent** to the cloud.

#Build scanning for C and C++

If the Conan package manager and corresponding manifests are not used to build a C/C++ project, a special mode for analyzing the build process output can be used to get a list of used libraries.

In this mode, the Johnny console agent analyzes the build process using compiler flags and identifies the libraries used. Then, using the system cache, the location of the libraries and their source are determined.

#Standards compliance

The **CodeScoring** platform helps organizations comply with secure development standards and regulatory requirements.

The system covers key secure development practices, including source code quality control, third-party software security control, vulnerability scanning, confidential information (secrets) scanning, and supply chain security incident response.

#GOST R 56939–2024

GOST R 56939–2024 "*Secure Software Development. General Requirements*" regulates the stages, requirements, and methods of secure software development, including architectural practices, code analysis, and dependency handling.

CodeScoring ensures compliance with 8 of the 25 standard points:

Requirement	Implementation Module	Comment
5.9 Source Code Expertise	CodeScoring.TQI	Provides automated code quality expertise using metrics (cyclomatic complexity, duplication, change dynamics). Systematic assessment of code quality and maintenance during development is supported.
5.12 Using a Secure Software Build System **5.13 Ensuring the Security of the Software Build Environment**	CodeScoring.OSA, CodeScoring.SCA	Dependencies used during the build are analyzed, and unsafe components are identified. Software composition is checked before the build is started, which helps reduce the risk of introducing vulnerabilities and errors from the system and build environment in accordance with goals 5.12.1.1 and 5.13.1.1.
5.15 Security of Used Secrets	CodeScoring.Secrets	Searches for keys, passwords, tokens, and other secrets in source code, commit history, and configuration files. Machine learning methods are used to reduce false positives.
5.16 Compositional Analysis	CodeScoring.SCA	All dependencies, including transitive ones, are analyzed, identifying their sources, licenses, and known vulnerabilities. A dependency list (PPK, SBOM) is generated and updated, ensuring composition control, vulnerability identification, and the application of corrective measures in the supply chain.
5.17 Supply Chain Malware Injection Scanning	CodeScoring.OSA, CodeScoring.SCA	Monitors third-party libraries and components used in the project. Implemented: • *5.17.2.4* – Detection and

Requirement	Implementation Module	Comment
		control of pre-built vendor software; • *5.17.2.5* – Dependency analysis for malicious code (excluding anti-virus scanning, which is considered additional protection).
5.23 Vulnerability Information Response	CodeScoring.OSA, CodeScoring.SCA, CodeScoring.Secrets	Provides regular vulnerability database updates and automatic notification of new risks. Integration with corporate systems (mail, task manager) is supported, enabling timely response within the organization's security policy.
5.24 Vulnerability Scanning During Operation	CodeScoring.SCA	Provides ability to regularly re-check released builds for recurrent vulnerability scanning. Notifications about new vulnerabilities are generated through analysis of source code, container images, and SBOM files during the production phase.

For more information on the platform's functionality, see the Functional specifications section.

#Installation requirements

#Operating system

Installation of the on-premise version is possible on **GNU/Linux** distributions.

#Server resources

#Baseline requirements for production installation

- Minimum supported resources for application servers: **32Gb RAM, 16 CPU cores**.

Resource recommendation

For production installations, it is not recommended to go below these values, as this may lead to performance degradation and unstable platform operation.

Resource recommendation

Use a CPU compatible with the Intel Scalable Gen 2 family. CodeScoring requires CPU instructions that may not be present in older CPUs.

#Data services requirements

- **PostgreSQL**: from **32Gb RAM**;
- **Redis**: from **2Gb RAM**;
- `shm` size for **PostgreSQL**: at least **4Gb**;
- If an external database is used, **PostgreSQL** is recommended to have at least **64Gb RAM**.

#Storage requirements

- For **CodeScoring.SCA** with VCS projects, calculate `analysis-root` volume based on the size of analyzed repositories multiplied by three;
- For **CodeScoring.SCA** with CLI projects and for **CodeScoring.OSA**, there is no mandatory ×3 storage formula;
- For the offline installation allocate at least **300Gb** for the CodeScoring Index database, with additional capacity reserved for further updates.

#Example of a high-load installation

- Database server: **96Gb RAM, 24 CPU cores**;
- Application server: **192Gb RAM, 48 CPU cores**.

#Supported versions of external services

When using your own database instances, make sure that their versions meet the requirements below:

#Redis

- Minimum version: **7.0.0**;
- Tested version: **7.4.9**.

#PostgreSQL

- Minimum version: **15.x** (any minor);
- Tested version: **15.15**.

Using other major branches does not guarantee correct results and may lead to errors or performance degradation.

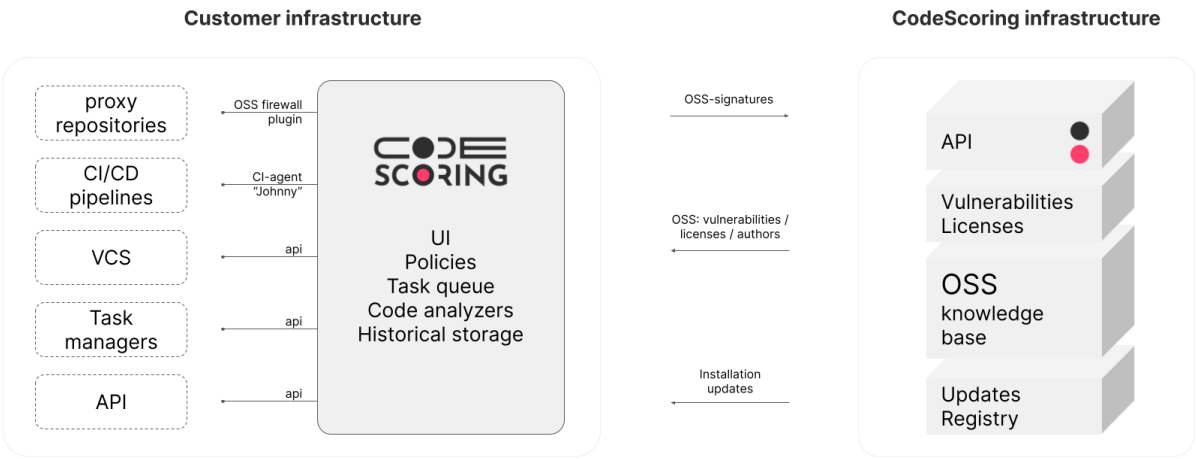
#External requests

To install the system, the Docker Registry with CodeScoring container images must be available, the address of which is provided along with the activation key.

For correct operation, the address `index.codescoring.ru` with a constantly updated database of known packages must also be available.

From the Index API the platform receives additional information on the dependencies found, their licenses and vulnerabilities.

The general architecture of the work is shown in the image below.



The source code is not transferred from the platform to the CodeScoring cloud. However, to obtain information on dependencies and control licensing terms, the following is sent:

1. anonymized information on found package manager manifests and their contents;
2. hashes of source code files to search for direct inclusions of Open Source libraries in project code;
3. number of active authors over the past year;
4. number of projects in the system;
5. platform version.

The paths of the manifests and the names of the hashed files are specifically anonymized. The hashes of files whose size does not exceed **512 bytes** are not being sent from the platform to the cloud.

An example of a request from an platform to the Index API with data from package manager manifests:

```
[
 {
 "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": {}
 }
 }
]
```

An example of a request from the platform to the Index API with data on hashes of source code files:

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

#Installation guide

1. Install Docker Engine for the desired operating system in accordance with the documentation: https://docs.docker.com/engine/install/.
2. Log in to the private registry of Docker images of the CodeScoring system using the command `docker login REGISTRY_URL`, with the URL, login and password received from the vendor.
3. Download the archive with platform files received from the vendor and unpack it.
4. Go to the created directory in the console.
5. Copy the template file with settings:

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

As a rule, no changes to the file are required for correct operation. If you need to configure **CodeScoring** to work through a proxy, please refer to the instructions.

6. Copy the template file with secrets:

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

```
Set up configuration settings in a new file.

    - A list of domains for proper CSRF protection. It is
recommended to list localhost on the internal and external
ports, as well as the external domain (or IP:port combination).
If you do not change the parameters in this file, the system
will be accessible by default at http://localhost:8081 .
Specifying the protocol is mandatory, for example:
        - `DJANGO_CSRF_TRUSTED_ORIGINS=http://
localhost:18000,https://localhost:8081,https://
example.com:8081`
    - Connection parameters to the PostgreSQL database. The
database is supplied with the platform. Specifying accesses
separately is a precaution and control measure. When using your
own database, you must ensure that it corresponds to
[requirements](/en/admin-guide/server-requirements#supported-
versions-of-external-services).
        - `POSTGRES_DB` — database name
        - `POSTGRES_USER` — username. When using your own
database, you must ensure that the user has the following
rights: **Superuser**, **Create role**, **Create DB**,
**Replication**, **Bypass RLS**.
        - `POSTGRES_PASSWORD` - password
        - `POSTGRES_HOST` - the host on which the database is
available
        - `POSTGRES_PORT` - port on which the database is
available
    - platform secret
        - `SECRET_KEY` - random string of characters
    - System domain settings
        - `NGINX_HOST` - the host on which the system will be
available
        - `NGINX_PORT` - port on which the system will be
accessible
        - `SITE_SCHEME` - data transfer protocol, default https
    - Exception paths
        - `ANALYSIS_IGNORED_PATHS` - list of paths that will be
ignored by the system during analysis. More details on adding
exclusion paths can be found [here](/en/admin-guide/analysis-
ignore-paths)
    - System version
        - `CODESCORING_VERSION` is a required variable. The
current version can be found in the [Changelog](/en/changelog/
```

```
on-premise-changelog) section
    - Compose-related settings
        - `COMPOSE_PROJECT_NAME` - docker compose project name,
used to prefix the names of resources created by docker compose

:::warning # symbol
Do not use the `#` symbol in parameters; it may be interpreted
incorrectly by the system during installation.
:::

:::note SSL Certificates
If you need to use self-signed certificates, please read the
[instructions for adding a certificate](/en/admin-guide/self-
signed-ssl) before starting the system.
:::
```

7. Run the CodeScoring installation command (the command must be executed with system superuser rights):

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

8. To view the logs you can use the command:

```
  ```bash
  docker compose logs -f
  ```
```

9. After starting, the service will be available at the configured domain or address `http://localhost:8081`. During the first launch, additional database migrations are performed; the operation may take longer than on subsequent runs.

```
  **Note**: for the platform to work via the HTTPS protocol,
you need an external balancer that implements SSL termination.
```

10. To log in, you must first create a user with administrator rights using the following command:

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

1. To change the administrator password, you can use the following command:

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

#Running CodeScoring in Docker Compose with an external DBMS

1. If a database schema other than `public` is used, the user's `search_path` must be explicitly set to include the target schema to ensure correct object resolution:

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

1. You must configure necessary variables in the `.env` file:
    - `POSTGRES_DB`
    - `POSTGRES_USER`
    - `POSTGRES_PASSWORD`
    - `POSTGRES_HOST`
    - `POSTGRES_PORT`
2. For managing the installation, you must apply the `external-db.override.yml` file, which is shipped with `docker-compose.yml` file:
    - Launch CodeScoring:

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

    - Read logs:

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

    - Stop CodeScoring:

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

#Running CodeScoring in Kubernetes

#Installation with a Helm chart

Installation procedure:

1. Create a namespace.

   ```
   kubectl create namespace codescoring
   ```

2. Create a secret for accessing the private Docker image registry for CodeScoring. Use the registry address (`REGISTRY_URL`), login (`USERNAME`), and password (`PASSWORD`) received from the vendor.

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

   Alternatively, you can create a registry access secret through `values.yaml` by adding the following entry to `.Values.imagePullSecrets`:

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

3. Install Helm using your preferred method.

4. Run the following commands to add the current Helm repository to the local machine:

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

#Helm chart settings

Important

It is strongly recommended to make the necessary changes **before installing CodeScoring**. Otherwise, a complete system reinstall may be required. These instructions assume that the **specialist has experience working with Kubernetes clusters and Helm**.

Tip

All resource limits shown in this document are approximate. Actual resource consumption depends on installation load and specific usage scenarios.

To edit CodeScoring parameters conveniently, download and unpack the Helm chart source code:

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

In `values.yaml`, edit the required variables and then run the installation command from the Helm chart source directory:

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

The chart stores all configuration, including Kubernetes resource definitions, in `values.yaml`, which makes the file fairly large and not always convenient to edit. If you need to override any parameters, it is acceptable and recommended to create an additional file with an arbitrary name, for example `values-override.yaml`. This document uses that name below. For convenience, the chart includes a `values-override.yaml` template.

In this case, the installation command looks as follows. The order of files matters:

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

#Installation with built-in PostgreSQL and Redis

The chart can deploy PostgreSQL and Redis using StatefulSet resources. To enable PostgreSQL and Redis installation, define the following parameters in `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"
```

When using built-in databases, no additional connection configuration is required.

#Connecting external databases

Important

When using your own database, make sure it meets the requirements.

#Connecting external Redis

To connect external Redis, specify the corresponding connection strings in the following fields and keep the database numbers consistent:

```
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"
```

#Connecting to PostgreSQL through PgBouncer

Important

External PostgreSQL must be connected through a connection pooler.

This option is suitable when PostgreSQL is already deployed in the existing infrastructure, but a connection pooler is not used. The Helm chart deploys

PgBouncer and connects it to the existing PostgreSQL. Complete the following steps:

1. Configure the connection pooler by specifying the corresponding values:

```
secrets:
  pgbouncer-secrets:
    data:
      # secrets for connecting pgbouncer to PostgreSQL
      POSTGRES_DB: "codescoring_db"
      POSTGRES_USER: "codescoring_user"
      POSTGRES_PASSWORD: "changeme"
      # secrets for internal pgbouncer pools
      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:
      # parameters for connecting pgbouncer to PostgreSQL
      POSTGRES_HOST: "codescoring-postgresql"
      POSTGRES_PORT: "5432"
      # parameters for internal pools
      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"
      # virtual pgbouncer databases that exist only inside
pgbouncer
      TRANSACTION_POOL_DATABASE_NAME:
"codescoring_generic"
      SESSION_POOL_DATABASE_NAME: "codescoring_generic-
session"
      MAX_CLIENT_CONN: "500"
```

2. Pass the configuration parameters created in step 1 to CodeScoring components:

```
secrets:
  ipcs-secrets:
    data:
      # secrets for connecting to pools from pgbouncer-
secrets
      TRANSACTION_POOL_PASSWORD: "changeme"
      SESSION_POOL_PASSWORD: "changeme"
```

```
      DATABASE_PASSWORD: "changeme" # matches
TRANSACTION_POOL_PASSWORD

configMaps:
  ipcs-backend-env:
    data:
      # connection parameters for osa-api and judge
      DATABASE_HOST: "pgbouncer"
      DATABASE_PORT: "6432"
      DATABASE_USERNAME: "codescoring_user" # matches
TRANSACTION_POOL_USER
      DATABASE_NAME: "codescoring_db" # matches
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"
```

#Configuring volumes

#Dynamic Volume Provisioning with the required StorageClass

The chart creates the required volumes through Dynamic Volume Provisioning using an explicitly specified `StorageClass`.

To create PersistentVolumeClaim resources, fill in the following values section:

```
pvcs:
  # replace storageClassName with the StorageClass used in the
cluster
  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"
```

If the cluster supports volumes with the ReadWriteMany (RWX) access mode, it is recommended to use that mode because it allows installation components to be placed on different cluster nodes. If the cluster does not support RWX, or if RWO is required for another reason, configure podAffinity so that pods using the same volumes are scheduled on the same node. To do this, add the following block to `values-override.yaml`:

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

#PersistentVolumeClaim for pre-created PersistentVolume resources

You can specify names of pre-created PersistentVolume resources in the `volumeName` field, for example:

```
pvcs:
  codescoring-ipcs-analysis-root:
    accessModes:
      - ReadWriteOnce
    size: 10Gi
    storageClassName: "default"
    volumeName: "codescoring-precreated-pv"
```

#Configuring resource limits

By default, `requests` and `limits` are set to demonstration values that are not intended for production use. This makes it possible to run CodeScoring in clusters with limited resources, for example in minikube, for testing purposes. When running in a **production environment**, you may need to configure resource limits. Edit the following fields:

#Resource limits for init containers

```
## limits applied to all init containers
## this field can only be edited in values.yaml; override is
not possible because of YAML structure limitations
init-container-resources: &init-container-resources
  resources:
    limits:
      cpu: 1000m
      memory: 1Gi
    requests:
      cpu: 1m
      memory: 100Mi

## limits for backend init containers are configured separately
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
```

#Resource limits for main component containers

In general, resources are configured using the following template:

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

```
            cpu: 1m
            memory: 500Mi
```

You can specify `requests` and `limits` together or separately. However, if only `limits` are specified, Kubernetes automatically sets `requests` to the same values, which may negatively affect pod scheduling.

#Adding a Certificate Authority (CA) certificate

To allow CodeScoring to access resources with TLS certificates signed by a corporate Certificate Authority (CA), add the root CA certificate (RootCA) to `values.yaml` in `key: value` format. The key is the certificate file name, including the `.crt` extension. The value is the certificate in PEM format. Add it to the following field:

```
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-----
```

You can specify any number of files, but **each certificate must be stored in a separate file**.

#Environment variables

#Managing environment variables through values

Environment variables and configuration files used by installation components are described in the `configMaps` and `secrets` blocks in `values.yaml`. The most commonly overridden configuration parameters are included in `values-override.yaml`.

#Managing environment variables through External Secret

External secret storage can also be connected. To use it, the cluster must have **External Secrets Operator (ESO)** installed. ESO adds the required CRDs (Custom Resource Definitions) to the cluster and connects it to secret storage.

To create an `ExternalSecret` resource and connect it to an existing `SecretStore`, add the corresponding entry to the `vaults` block:

```
vaults:
  ipcs-backend-secret-
external: # name of the secret created as a result
    apiVersion: external-secrets.io/v1 # external-secrets.io/
v1beta1 by default
    enabled: true
    store:
      name: vault-backend # name of the pre-created SecretStore
```

```
      kind: ClusterSecretStore # type of the pre-created
SecretStore
    path: secret/data/my-vault-secret # path to the secret in
Vault
```

To pass the created secret to all `Deployment` resources, add the resource name from the `vaults` block to `deploymentsGeneral.envSecrets`, for example:

```
deploymentsGeneral:
  envConfigmaps:
    - ipcs-backend-env
  envSecrets:
    - ipcs-secrets
    - ipcs-backend-secret-
external # the resource name from the vaults block
```

To pass the secret only to one `Deployment` resource, specify it in the `deployments` block along with all secrets already used by that resource, for example:

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

#Migration from Helm Chart 2.x.x

Important

If built-in PostgreSQL was used, create a backup before any manipulations.

The installation will be unavailable during migration, so it is recommended to allocate a maintenance window. The migration takes 5-10 minutes in total.

Migration steps:

1. Delete all Deployment resources:
   `kubectl delete deployment --all -n codescoring`.
2. Delete all Service resources:
   `kubectl delete service --all -n codescoring`.
3. Run helm upgrade after filling in `values.yaml` and `values-override.yaml` according to Helm chart settings:

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

#Installation in Kubernetes (legacy)

Legacy version

This page describes the legacy Helm chart version. For new installations, use the current Installation in Kubernetes guide.

#Installation using Helm chart with default parameters

Important!: This installation option does not provide an ability to scale CodeScoring horizontally. To install CodeScoring with horizontal scaling support, please refer to the relevant documentation section below.

Important!: You must have a configured default `StorageClass` in the cluster. By default, volumes are created **with a capacity of 20 GiB**

Installation order:

1. Create a namespace.

   ```
   kubectl create namespace codescoring
   ```

2. Create a secret to access the private registry of Docker images of the "CodeScoring" system, using the address (`REGISTRY_URL`), login (`USERNAME`) and password (`PASSWORD`) received from the vendor.

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

3. Install Helm using your preferred method.

4. Run the following commands to add the current Helm repository to the local machine:

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

5. Create a `values.yaml` file with the following content:

   Important!: Please replace the values in the sensitive data fields with your own. These fields include `secretKey`, `defaultSuperuserUsername`,

defaultSuperuserPassword, defaultSuperuserEmail, and all fields containing username or password. It is also important to note that all such variables are mandatory.

```
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 # site scheme http or https
   siteHost: "codescoring.k8s.local" # domain where
CodeScoring will be available
   djangoCSRFTrustedOptions: "https://
codescoring.k8s.local" # Domain where CodeScoring will be
available, including schema
   secretKey: "" # secret key for the application backend,
random string of characters
   defaultSuperuserUsername: "admin" # administrator name
on the system
   defaultSuperuserPassword: "changeme" # system
administrator password
   defaultSuperuserEmail: "mail@example.com" # e-mail of
the administrator in the system
   databaseHost: pgbouncer
   databasePort: 6432
   postgresqlDatabase: "codescoring"
   postgresqlUsername: "codescoring"
   postgresqlPassword: "changeme"

 frontend:
   ingress:
     enabled: true
     className: "nginx"
     hosts:
     - host: codescoring.k8s.local # domain where
CodeScoring will be available
     paths:
```

```
      - path: /
      pathType: ImplementationSpecific
```

6. Run the command to install the chart

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

#Changing the admin password

To change the admin password without manually editing the `values.yaml` file, you can use the following command:

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

#Advanced settings for Helm chart parameters

Important: It is highly recommended that you make the necessary changes **before installing CodeScoring**, otherwise a complete system reinstall may be required. These instructions assume that the **specialist has experience working with a Kubernetes cluster and the Helm utility**.

To easily edit CodeScoring parameters, you can download and unpack the Helm chart source code with the command:

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

In the `values.yaml` file you can edit the necessary variables, and after that run the platform command in the directory with the Helm chart source code:

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

#Connecting to external PostgreSQL and Redis

By default, PostgreSQL and Redis run in separate `StatefulSets`. This option may not be suitable for usage in a **production environment**, because it is not fault tolerant.

Important: when using your own database, you must ensure that it corresponds to requirements.

#Connecting to external Redis

To connect to external Redis, you must do the following:

1. Disable Redis deployment by specifying the variable - `redis.enabled: false`
2. In the `codescoring.config.djangoCachesRedisUrls` and `codescoring.config.hueyRedisUrl` variables, specify the connection strings for external Redis.

#Connecting to external Redis via TLS

To connect to external Redis, in addition to abovementioned you must do the following:

1. Set the `codescoring.trustedCA.enabled` variable to `true`
2. Add the Redis-server root certificate to `codescoring.trustedCA.certificates`
3. In the `codescoring.config.djangoCachesRedisUrls` and `codescoring.config.hueyRedisUrl` variables, specify the connection strings for external Redis using the following format: `rediss://redis.example.com:6379/0`, where 0 is the Redis database number.

#Connecting to PostgreSQL via Pgbouncer pooler

Important!: Connecting to external PostgreSQL must be done using a connection pooler.

This option is suitable if PostgreSQL is already deployed in the existing infrastructure, but the connection pool is not used. Helm-chart will deploy the Pgbouncer puller and connect it to the existing PostgreSQL. You need to do the following:

1. Disable PostgreSQL deployment by specifying the variable - `postgresql.enabled: false`

2. Connect the Pgbouncer pooler to external PostgreSQL, replacing the appropriate parameters with the required ones:

```
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"
```

#Connecting to an external PostgreSQL pooler

This option is suitable if PostgreSQL and a connection pooler (for example, PgBouncer) are already deployed in the existing infrastructure. In this case, deployment of the Pgbouncer pooler is not required. You need to do the following:

1. Disable PostgreSQL deployment by specifying the variable - `postgresql.enabled: false`
2. Disable Pgbouncer deployment by specifying the variable - `pgbouncer.enabled: false`
3. Connect codescoring directly to an external pooler, in the `codescoring.config` section the parameters are:

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

1. Enable an external pooler and adjust the connection settings in the `codescoring.config.externalPooler` section:

Important!: An external pooler must support both transaction and session work modes. The connection settings for the pools must be adjusted in corresponding sections: `externalPooler.transactionPool` and `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"
```

#Setting up volumes (PV)

By default, the chart creates the required volumes via Dynamic Volume Provisioning using the default `StorageClass`. If this volume deployment option is not suitable, you can configure volume creation in several ways.

Important!: The options described below are **mutually exclusive**. You must select **ONLY ONE** deployment option for each volume. You can select different deployment options for different volumes.

Tip

To change the size of created volumes (except for local ones), you need to change the `size` parameter in the corresponding sections.

#Dynamic Volume Provisioning using the required StorageClass

You can set the required `StorageClass` in the following variables:

- `codescoring.persistentVolumes.analysisRoot.storageClass`
- `codescoring.persistentVolumes.mediaRoot.storageClass`
- `codescoring.persistentVolumes.djangoStatic.storageClass`
- `codescoring.backup.persistentVolume.storageClass`
- `redis.persistentVolume.storageClass` (if using built-in Redis)
- `postgresql.persistentVolume.storageClass` (if using built-in PostgreSQL)

In this case, volumes will be created using the specified `StorageClass`

#PersistentVolumeClaim for pre-created PersistentVolumes

The name of pre-created volumes can be set in the following variables:

- `codescoring.persistentVolumes.analysisRoot.volumeName`
- `codescoring.persistentVolumes.mediaRoot.volumeName`
- `codescoring.persistentVolumes.djangoStatic.volumeName`
- `codescoring.backup.persistentVolume.volumeName`
- `redis.persistentVolume.volumeName` (if using built-in Redis)
- `postgresql.persistentVolume.volumeName` (if using built-in PostgreSQL)

In this case, only `PersistentVolumeClaim` will be created for the volumes specified in these variables

#Using pre-created PersistentVolumeClaim

The name of pre-created PVCs can be set in the following variables:

- `codescoring.persistentVolumes.analysisRoot.existingClaim`
- `codescoring.persistentVolumes.mediaRoot.existingClaim`
- `codescoring.persistentVolumes.djangoStatic.existingClaim`
- `codescoring.backup.persistentVolume.existingClaim`
- `redis.persistentVolume.existingClaim` (if using built-in Redis)
- `postgresql.persistentVolume.exsistingClaim` (if using built-in PostgreSQL)

In this case, the specified PVC name will be inserted directly into the `volumes` section for Pod.

#Using local volumes

If there is no external data storage in the Kubernetes cluster, it is possible to run CodeScoring using local volumes. In this case, the data will be stored on one of the cluster nodes.

To create local volumes, you must perform the following steps:

1. Assign the value `true` to the following variables:

   - `codescoring.persistentVolumes.analysisRoot.localVolume.enabled`
   - `codescoring.persistentVolumes.mediaRoot.localVolume.enabled`
   - `codescoring.persistentVolumes.djangoStatic.localVolume.enabled`
   - `codescoring.backup.persistentVolume.localVolume.enabled`
   - `redis.persistentVolume.localVolume.enabled` (if using built-in Redis)
   - `postgresql.persistentVolume.localVolume.enabled` (if using built-in PostgreSQL)

1. Set the path to the **directory on the cluster node**, in which the data in the following variables will be placed:

   - `codescoring.persistentVolumes.analysisRoot.localVolume.path`
   - `codescoring.persistentVolumes.mediaRoot.localVolume.path`
   - `codescoring.persistentVolumes.djangoStatic.localVolume.path`
   - `codescoring.backup.persistentVolume.localVolume.path`
   - `redis.persistentVolume.localVolume.path` (if using built-in Redis)
   - `postgresql.persistentVolume.localVolume.path` (if using built-in PostgreSQL)

1. Specify the name of the node on which the local volume will be created in the following variables:

   - `codescoring.persistentVolumes.analysisRoot.localVolume.nodeHostname`

- `codescoring.persistentVolumes.djangoStatic.localVolume.nodeHostname`
- `codescoring.backup.persistentVolume.localVolume.nodeHostname`
- `redis.persistentVolume.localVolume.nodeHostname` (if using built-in Redis)
- `postgresql.persistentVolume.localVolume.nodeHostname` (if using built-in PostgreSQL)

It is allowed to use different nodes for different volumes.

#Setting up storage for temporary scan files

By default, temporary files during the scanning process are stored in the `/tmp` directory inside containers, to which Ephemeral Volumes of type `emptyDir` are mounted:

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

However, in some cases you may need to use Persistent Volume instead of Ephemeral Volume. In this case, you should comment out the appropriate sections in `ephemeralVolumes` for one or more services, depending on which services require volumes to be mounted:

```
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: {}
```

Then you need to set the value `enabled: true` in one or more of the following sections:

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

As a result, PersistentVolumeClaim will be created for the corresponding services. It is worth noting that the possibilities for configuring these volumes fully correspond to those described in the section Configuring volumes (PV).

When horizontally scaling services, you need to configure volumes in accordance with the instructions in the Horizontal scaling CodeScoring section.

#Horizontal scaling CodeScoring

Important!: To horizontally scale the CodeScoring system, the Kubernetes cluster must have the ability to create volumes with the access type **ReadWriteMany (RWX)**

To scale CodeScoring horizontally, you need to create `analysis-root`, `media-root` and `django-static` volumes with an access type of `ReadWriteMany`.

To do this, you need to replace the value of `ReadWriteOnce` with `ReadWriteMany` in the variables:

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

Then, you need to comment out the variables:

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

If this is not done, then all pods will be launched on only one node in the cluster.

#Setting resource limits

By default, `requests` and `limits` are not specified. This is done to ensure the ability to run the CodeScoring system in clusters with a small number of resources (for example, minikube) for testing purposes. When running in a **production environment**, you may need to configure resource limits. This can be done by setting the following variables:

- `postgresql.resources` (when using built-in PostgreSQL)
- `redis.resources` (when using built-in 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`

It is possible to specify both `resources` and `limits` together or separately, for example:

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

Below are approximate `limits` values for a CodeScoring platform with 8-10 projects:

```
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
```

#Adding a Certificate Authority (CA) certificate

To access CodeScoring resources with TLS certificates signed by a corporate certification authority (CA), you must:

1. Set the `codescoring.trustedCA.enabled` variable to `true`
2. Add the root certificate of the certification authority (RootCA) in PEM format to the `codescoring.trustedCA.certificates` variable in the `key: value` format, where key is the name of the certificate file, including the `.crt` extension, value is the certificate in PEM format.

For example:

```
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-----
```

If you have multiple root CAs, you need to add them to separate keys, for example:

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

#Secret Management

By default, the **Secret** objects are created for `ipcs-backend`, `pgbouncer` and `postgresql` templates. The values for their variables are provided in the values file.

It is also possible to use external secret storages. To achieve that, it is necessary to have an **External Secrets Operator (ESO)** installed in the

cluster. It adds all required CRDs (Custom Resource Definitions) to the cluster and connects to an external secret provider.

In order to connect ESO to an external secret storage, you must configure a provider for the **SecretStore** in the `codescoring.secretStore` block.

Next, you need to configure the **ExternalSecret** objects to fetch secrets from the external provider in the `codescoring.config.externalSecret`, `pgbouncer.externalSecret`, `postgresql.externalSecret` blocks.

All the configurations must be made in accordance with the official ESO documentation.

Important!: Some providers may bill requests to secret storages. The request interval can be configured using the `externalSecret.refreshInterval` parameter for each particular service.

#Monitoring

For metrics collection purpose there are the **ServiceMonitor** resources provided in the chart. The metrics are collected from the backend and the osa-api services.

The ServiceMonitor can be configured in the following values sections: `codescoring.backend.prometheus.serviceMonitor`, `codescoring.osa_api.prometheus.serviceMonitor`.

In addition there are **PrometheusRule** resources provided for the abovementioned services. They are required for the alerting rule setup and can be configured in the following values sections: `codescoring.backend.prometheus.alerts`, `codescoring.osa_api.prometheus.alerts`.

All the settings must be adjusted in accordance with the official Prometheus Operator documentation.

#Updating CodeScoring

In order to update CodeScoring you need to actualize the helm repository by running

```
helm repo update
```

and then update the platform with the following command, where the `CHART_NAME` variable must contain the version you are going to update to:

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

#Offline installation

CodeScoring supports operation in air-gapped environments. In this mode, access to the package and vulnerability database is provided by the Index API Offline service.

Product prerequisites

Before using Index API Offline, the CodeScoring platform must already be deployed on the server using Docker or Kubernetes.

#Installation and update resources

The resource address for the installation files, Docker Registry, and offline database files can be obtained from the vendor.

The examples below use the following placeholders:

- <REGISTRY_URL> — HTTPS address of the distribution resource;
- <REGISTRY_HOST> — Docker Registry address without the protocol.

The following resources are available:

- Docker Registry with CodeScoring and Index API Offline images
- Full offline Index API database: <REGISTRY_URL>/db/index-api.db
- Incremental offline database updates: <REGISTRY_URL>/#browse/browse:codescoring-offline-files:db%2Fv1%2Fupdates

The offline database is provided as an encrypted SQLite file. Incremental updates are distributed as WAL files.

#Downloading the database

1. Select a server with at least 300 GB of available disk space.
2. Create a curl.config file with the following credentials:

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

3. Start downloading the database:

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

It is recommended to perform the download inside screen or tmux, as the process may take a significant amount of time.

#Installing Index API Offline

1. Log in to the Docker Registry:

   ```
   docker login <REGISTRY_HOST>
   ```

2. Download the archive with installation files and extract it:

   ```
   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. Move the downloaded database file to the service directory, for example:

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

4. Navigate to the configuration directory:

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

5. Copy the `.env.template` configuration template to `.env` and configure it:

   Main parameters:

   - `INDEX_API_OFFLINE_VERSION` — Index API Offline version;
   - `NGINX_SSL_ENABLED` — enable SSL (true | false);
   - `NGINX_HOST` — hostname for nginx;
   - `OFFLINE_DB_UPDATE_ENABLED` — enable automatic database updates;
   - `WAL_CHECK_INTERVAL` — update check interval;
   - `NEXUS_HOST` — update repository address;
   - `NEXUS_USERNAME` — Nexus username;
   - `NEXUS_PASSWORD` — Nexus password;
   - `CODESCORING_ACTIVATION_KEY` — license key.

   Authentication parameters for database updates and Prometheus metrics access (optional):

   - `MAINTENANCE_USERNAME` — Basic Auth username for `/system/update_database`;
   - `MAINTENANCE_PASSWORD` — Basic Auth password for `/system/update_database`;
   - `METRICS_USERNAME` — Basic Auth username for `/metrics`;
   - `METRICS_PASSWORD` — Basic Auth password for `/metrics`.

6. When SSL is enabled (`NGINX_SSL_ENABLED=true`), place the certificate and key files in the `ssl` directory:

   Important

   The certificate must be in PEM format. The certificate file must have the `.crt` extension, and the private key file must have the `.key` extension.

7. Start the service:

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

After installation and startup, the CodeScoring interface displays the version of the latest WAL file uploaded to Index API Offline.

#Database Updates

#Automatic offline database updates

The database update process can be performed automatically.

The update mechanism works as follows:

1. The running Index API Offline service periodically checks the update repository.
2. The service stores the current database version number (for example, 72).
3. When the next available update is detected (for example, 73), the file is downloaded automatically.
4. The downloaded WAL file is applied to the local database.

The check interval is defined by the `WAL_CHECK_INTERVAL` parameter.

If `OFFLINE_DB_UPDATE_ENABLED` is disabled, automatic updates will not be performed.

#Manual database update

Manual updating is used when automatic WAL file downloading is disabled (`OFFLINE_DB_UPDATE_ENABLED=false`) or is not possible due to network restrictions.

#Updating using individual update files

If you have deployed the first version of an offline database, you must sequentially download and upload all available WAL update files, starting with the version following the current database version.

The update files must be placed in the directory specified by the WAL_DIR parameter in the Index API Offline configuration. The service monitors files in this directory and applies them to the database.

The update package includes:

- `*.wal` — a file containing the changes (the main update data);
- `*.shm` — an auxiliary service file required by SQLite for correct WAL application.

#Updating using the full database version

If you are downloading the latest **full offline database** file, you must do the following:

1. Stop the Index API Offline service.
2. Move the current database file to a backup directory (it is recommended to save a copy until the upgrade is complete).
3. Place the new database file in the directory specified in the `DB_FILE` parameter.
4. Start the Index API Offline service.
5. Verify that the service has started successfully.

After this, if necessary, you can continue the upgrade using WAL files, placing them in the `WAL_DIR` directory in the standard order.

#Starting the upgrade process

When using manual upgrade mode, the upgrade process is also started manually.

1. Specify authentication parameters in the `.env` file:

```
MAINTENANCE_USERNAME=
MAINTENANCE_PASSWORD=
```

2. Place the update files (`*.wal`, `*.shm`) in the directory specified in the `WAL_DIR` parameter.

To start the update process, run the following query:

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

Where:

- `{{INDEX_API_URL}}` is the value from the `INDEX_API_URL` variable in `.env`;
- `{{metrics_auth}}` — a Base64-encoded string similar to `user:password`.

The update status is tracked in the Index API Offline service logs.

Possible messages:

- `WAL files download scheduler started with interval:` — the update service has started;
- `missing mandatory Nexus configuration parameters` — a valid message for manual mode;
- `The WAL file [%s] was successfully loaded into the main database file` — the update was successfully applied;
- `Failed to checkpoint WAL file:` — an error applying the WAL file;
- `No WAL files to checkpoint` — there are no files to apply;
- `Error getting latest database update:` — an error retrieving update information.

#Updating the system

#Standard update guide

Backup

Before performing an update, make sure to create a full backup of the platform.

Update the compose files before starting a new version

Changing only the `CODESCORING_VERSION` value in `.env` is not enough. Before the update, make sure to replace `docker-compose.yml`, `external-db.override.yml`, `app.env`, and `.env` with the latest versions provided by the vendor. If old compose files are left in place, the platform may start with an incomplete or incorrect configuration.

In the `.env` file, the variable `CODESCORING_VERSION` specifies the target platform version. The latest available version can be found in the Changelog.

Then follow these steps:

1. Navigate to the directory containing the deployment files:

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

2. Pull the latest images:

   ```
   docker compose pull
   ```

3. Restart the platform:

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

#Restoring a previous version

If errors occur after the update or the system becomes unstable, you can restore the previous platform version from a backup:

1. Stop the current installation:

   ```
   docker compose down
   ```

2. Clear the database using any preferred method:

   - via Docker:

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

   - by manually dropping the database (`DROP DATABASE`);
   - or, if using Kubernetes:

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

3. Restore the database from the previously created backup.

4. In the `.env` file, set the `CODESCORING_VERSION` variable to the previous version value.

5. Restart the platform:

   ```
   docker compose up -d
   ```

Detailed instructions for creating and restoring backups are available in the Backup guide.

#Update guides for versions with changes in configuration

#[2025.21.0] – 2025-05-21

Starting with this version, the value of the environment variable $SECRET_KEY will be used to encrypt sensitive data in the database and changing the value of this variable will require additional operations.

Before upgrading, you must ensure that the `.env` file specifies the correct (**unique, unpredictable**) value of $SECRET_KEY, and not the default value.

#[2025.13.0] - 2025-03-28

- You must ensure that the version of `Docker Engine` is greater than or equal to 25. To do this, run the `docker version` command on the

machine with the platform. If the Docker Engine version is lower than 25, you need to update Docker.
    - **IMPORTANT!** Before updating Docker, you must stop the platform normally.
- You must add the name of the docker compose project to the configuration:
    - Before shutting down the system for updating, you must note the name of the docker compose project in which the platform is currently running.
    - This is either the value passed with the `-p` parameter to `docker compose`, or the name of the directory where the `docker-compose.yml` file was located, by default -- `on-premise` or `on-premise-split-db`
    - This value is used as a prefix in the name of resources created by compose: volumes, containers, networks
    - You must enter this value in the `.env` file with the key `COMPOSE_PROJECT_NAME=`
    - **IMPORTANT!** If you do not do this, the platform will not start. If you enter an incorrect value, volumes with a new prefix will be created, and the platform on the new version will start "from scratch"
    - After the value is added to the `.env` file, requests to `docker compose` can be made without the `-p PROJECT_NAME` option
- You need to download the updated `docker-compose.yml` and `external-db.override.yml` files from the CodeScoring registry and place them in the directory with the compose file.

#PostgreSQL Upgrade in Docker

Upgrading PostgreSQL to a new major version requires initializing the database using the new PostgreSQL version, creating a dump, and restoring it. Below is the procedure for CodeScoring installations deployed with Docker Compose.

Correct update order

It is not recommended to update the installation and database simultaneously. The update should be performed sequentially: first update the installation and ensure the service is working correctly, then update the database.

#Prerequisites

#Disk space

You must have free disk space of at least **the size of the `db-data` volume**.

Example of disk space usage during an upgrade:

- PostgreSQL versions:
    - source: 13.21

    - target: 15.15
- CodeScoring installation data:
    - ~500,000 packages
    - ~1,200 projects
    - ~22,000 images
- `db-data` volume size: ~50 GiB
- disk space consumption during the upgrade:
    - compressed backup: ~9 GiB
    - compressed dump: ~3 GiB
    - `db-data` size after restore: ~32 GiB

#Planned maintenance window

The procedure requires a complete shutdown of CodeScoring. The actual execution time of upgrade depends on the data volume.

For the example above:

- compressed backup creation: ~11 minutes 45 seconds
- backup restoration : ~2 minutes 30 seconds
- `pg_dump` + `pg_restore`: ~3 minutes 30 seconds
- data transfer between volumes: ~1 minute
- query planner statistics collection: ~1 second

#Docker compose override configuration

The upgrade requires a Docker Compose override file. This file (`postgres-upgrade.override.yml`) is shipped starting from CodeScoring version 2026.3.1 and is located in the same directory as `docker-compose.yml`.

#Components of the override file

#Volumes

- `upgrade-new-db-data` - temporary PostgreSQL `pgdata` for the new version
- `upgrade-dump` - compressed database dump
- `upgrade-backup` - compressed backup

#Services

- `psql-new` - PostgreSQL of the new version with configuration identical to the `psql` service
- `upgrade-dump` - creates a compressed database dump
- `upgrade-dump-restore` - restores the dump into the new PostgreSQL version
- `upgrade-cleanup-transfer` - performs cleanup and transfers the data into the main `db-data` volume
- `upgrade-analyze-in-stages` - iteratively collects statistics for PostgreSQL Query Planner

- `upgrade-backup` - backup creation
- `upgrade-backup-restore` - backup restoration

#Backup

#Backing up

Stop CodeScoring:

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

Create a backup:

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

#Restoring the backup

Stop CodeScoring:

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

Restore the backup:

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

Ensure that the `POSTGRES_IMAGE` value in the `.env` file matches the PostgreSQL version **prior** to upgrade:

```
POSTGRES_IMAGE=postgres:13.23-rev1 # previous PostgreSQL
version
```

#Upgrade procedure

Set in the `.env` file:

```
POSTGRES_IMAGE=postgres:13.23-rev1 # previous PostgreSQL
version
POSTGRES_UPGRADE_IMAGE=postgres:15.15-rev1 # new PostgreSQL
version
```

Pull the images required for the upgrade:

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

Stop all services:

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

Initialize the new PostgreSQL instance, perform the dump and restore:

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

Stop the services again:

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

Transfer the data into the main `db-data` volume:

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

Set the PostgreSQL version for the image used by the `psql` service in the `.env` file:

```
POSTGRES_IMAGE=postgres:15.15-rev1 # new PostgreSQL version
```

Start statistics collection (no need to wait for completion):

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

Start CodeScoring:

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

Check the logs and verify that CodeScoring is operating correctly:

```
docker compose logs -f
```

Remove the temporary volumes:

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

#Backup

#Creating a backup copy of the platform

1. Go to the directory with the startup files:

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

1. To create a backup, run the command:

```
docker compose run backup create
```

The backup file will be saved to the backup directory.

#Restoring from a backup copy

1. To restore from a backup copy, run the command:

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

`BACKUP_FILENAME` is the name of the backup file. A list of available copies can be obtained by running the command:

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

1. Restart the platform:

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

#Proxy usage

If you need to use the system through a proxy, you need to uncomment and set the values of the corresponding variables in the `app.env` file:

- `HTTP_PROXY` and `HTTPS_PROXY` - proxy server address

Important: For both variables, the value must use the `http` scheme.

- `NO_PROXY` — list of URLs of external systems, requests to which should not go through a proxy. Among the values, it is possible to specify IP addresses, for example:

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

If adding a list of VCS URLs to the `NO_PROXY` variable is not possible, then you need to add the proxy certificate to the SSL directory.

Important: with the HTTP Proxy settings specified, all requests to the URLs of external systems specified in the `NO_PROXY` variable will go past the proxy server. This applies to both version control systems and, for example, connected task managers.

#Using self-signed SSL certificates

When working with external systems, CodeScoring checks the validity of SSL certificates of remote hosts and by default will not connect projects from systems whose certificates have not been validated or are signed by a certification authority unknown to the system.

To connect an external system whose domain SSL certificate is self-signed, you need to add the root certificate to trusted at the installation level. To do this, before starting the system, it must be placed in the `ssl` directory in the system installation files. It is advisable to give the file a meaningful name, for example, `codescoring-root-CA.crt`.

Important: the file extension must be `crt`.

To view the entire chain of certificates used for a domain and highlight the root one, you can use the command:

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

#Connecting to PostgreSQL/PgBouncer using TLS

Connecting to PostgreSQL/PgBouncer using TLS provides secure and encrypted data transfer between client and server. This option is available starting from CodeScoring version 2025.21.0.

#Guide for connection

1. Before starting the system, place certificate and key files in the `ssl` directory in the system platform files;
2. Give the files self-explanatory names, e.g. `pgbouncer_server.crt`, `pgbouncer_server.key`;

   Important: the extension of the certificate files must be `crt`;
3. Change the owner of the key files as follows:

   ```
   chown 999:0 ./ssl/postgresql_server.key
   chown 1050:0 ./ssl/pgbouncer_server.key
   ```
4. Uncomment and edit your SSL/TLS configuration files.

#Example of how to enable PostgreSQL/PgBouncer in TLS mode with self-signed certificates

1. Generate certificates in the `ssl` directory in the system platform files using the `mkcert` utility

   ```
   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. Change key file owners

   ```
   chown 999:0 ./ssl/postgresql_server.key
   chown 1050:0 ./ssl/pgbouncer_server.key
   ```
3. Copy the SSL/TLS configuration templates

```
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. Uncomment and edit the following lines in the configuration file `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. Uncomment and edit the following lines in the configuration file `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'
```

#Analysis exception paths

#Default values

There are exception paths for which manifests and files are not **searched** and quality analysis is not performed. By default, the following values are added to the exceptions (expression format is `glob`):

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

#Adding exceptions

To add your own values to the list, in the `app.env` file you need to **add** values in the `ANALYSIS_IGNORED_PATHS` variable in the following format:

- `**/ignoring_prj_1/**` - to exclude the directory `ignoring_prj_1` from analysis;
- `**/ignoring_projects_*` - to exclude from the analysis directories that have `ignoring_projects_` in their name
- `**/ignoring_file.pom` - to exclude the file `ignoring_file.pom` from analysis

Important: It is not recommended to delete exception paths specified in the default variable.

Paths are added using `,`. Here is an example with `**/migrations/**` exception added:

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

#Scripts for platform management

#Script execution

Scripts are executed in the platform backend service:

- for Docker Compose:

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

- for Helm:

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

All usage examples are provided for docker-compose but can also be run in k8s. To do so, replace `docker` with `kubectl`.

Arguments passed to the script are strictly positional and are specified with the `--script-arg` key. Optional arguments are enclosed in square brackets.

#Available commands

#update_cwes

Updates CWE (Common Weakness Enumeration) and vulnerability data in the installation database. An optional argument can be used to control the date from which the update will be performed; by default, the update date and offset will be taken from the cache of the last successful update.

Syntax

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

#load_licenses

Updates license data in the installation database. An optional argument can be used to control the date from which the update will be performed; by default, the update date and offset will be taken from the cache of the last successful update.

Syntax

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

#set_new_secret_key

This command sets a new value for the SECRET_KEY environment variable. To avoid issues with encoding, aliases, or other unexpected console behavior, it is recommended to copy the NEW_SECRET_KEY value from the terminal output after the script has finished.

Syntax

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

Usage examples

1. Changing from explicitly specified OLD_SECRET_KEY to explicitly specified NEW_SECRET_KEY. This is required when the SECRET_KEY environment variable value during platform launch differs from OLD_SECRET_KEY.

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

   This command will re-encrypt all sensitive fields from the given OLD_SECRET_KEY to NEW_SECRET_KEY.

Important

If the SECRET_KEY variable at the moment of script execution does not match the NEW_SECRET_KEY, after successful script completion you must update the SECRET_KEY variable to NEW_SECRET_KEY and restart the platform.

2. Changing to explicitly specified NEW_SECRET_KEY without specifying OLD_SECRET_KEY. This is required when OLD_SECRET_KEY matches the one defined in settings.SECRET_KEY.

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

This command will re-encrypt all sensitive fields from the key set in settings.SECRET_KEY to NEW_SECRET_KEY.

After successful script completion you must update the SECRET_KEY variable to NEW_SECRET_KEY and restart the platform.

#Variable Definitions

This section provides a description of the environment variables necessary for platform installation and configuration, including those detailed in sections Installing the System in Docker and System Operation in Kubernetes.

Modifying values

Consult with technical support before modifying any of the optional, non-required variables during installation. Altering default values may impact system performance and operational efficiency.

Avoid using the

symbol in parameters. It may be misinterpreted by the system during installation.

#env.template

The env.template file contains environment variables required to launch a CodeScoring installation using Docker Compose. It defines parameters necessary for the correct execution of the application within a containerized environment.

- **COMPOSE_PROJECT_NAME** - Docker Compose project name, used to prefix the names of resources created by Docker Compose;
- **CODESCORING_VERSION** - specifies the CodeScoring version for installation. The latest version can be found in the Changelog section;

- **SECRET_KEY** - CodeScoring utilizes a randomly generated string as its installation secret;

- **DJANGO_CSRF_TRUSTED_ORIGINS** - list of domains required for proper CSRF protection. Specifying the protocol is required; for example:

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

#Nginx

CodeScoring uses Nginx for system domain configuration.

- **NGINX_HOST** - specifies the hostname for system accessibility;
- **NGINX_PORT** - specifies the port for system accessibility;
- **SITE_SCHEME** - specifies the data transmission protocol. Default value: `https`.

#PostgreSQL

CodeScoring leverages `PostgreSQL` for data management, processing and storage.

- **POSTGRES_DB** - specifies the database name;
- **POSTGRES_USER** - specifies the database username. When using a custom database, ensure the user has the following privileges: `Superuser, Create role, Create DB, Replication, Bypass RLS`;
- **POSTGRES_PASSWORD** - specifies the database password;
- **POSTGRES_HOST** - specifies the hostname for database accessibility;
- **POSTGRES_PORT** - specifies the port for database accessibility.

#Sentry

Integration with Sentry is disabled by default, `SENTRY_ENABLE = False`. To enable Sentry for error reporting and system event tracking, set `SENTRY_ENABLED = True` and provide the following variables:

- **SENTRY_DSN** - project identifier within Sentry, to which epy data will be sent;
- **SENTRY_ENVIRONMENT** - the environment to which sent data will belong;
- **SENTRY_RELEASE** - release associated with the sent data.

#app.env

The `app.env.template` file contains environment variables required to configure the CodeScoring installation. It defines settings for the database, web server, and queues, as well as other critical system parameters.

- **PATH** - specifies the path to executable files, virtual environment (venv) files, and standard system paths;

```
   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** - specifies exception paths for which manifests and files are not searched and quality analysis is not performed. The following paths are excluded by default:

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

- Variables specifying the location of core installation files:
  - **HOME** - value by default: `/ipcs-backend`;
  - **BASE_DIR** - value by default: `/ipcs-backend`;
  - **AZURE_DEVOPS_CACHE_DIR** - value by default: `/ipcs-backend`.
- **INDEX_API_URL** - the URL for the index API, used for knowledge base updates, is `https://index.codescoring.ru`. This parameter should not be modified;
- Deprecated **INDEX_API_TIMEOUT** - deprecated environment variable for a single Index API request timeout. Use the separate timeouts below;
- **INDEX_API_POOL_TIMEOUT** - time to obtain a connection from the pool (seconds). Default: `60`;
- **INDEX_API_CONNECT_TIMEOUT** - time to establish a connection (seconds). Default: `60`;
- **INDEX_API_WRITE_TIMEOUT** - time to write the request body (seconds). Default: `60`;
- **INDEX_API_READ_TIMEOUT** - time to read the response body (seconds). Default: `60`;
- **INDEX_PROXY_URL** - specifies the URL address of the Index Proxy container within the Docker network. No modification required. Default value: `http://index-proxy:8000`;
- **ALLOWED_HOSTS** - specifies the allowed hosts for Django. Defines the permissible addresses from which Django can accept requests. If a host

is not included in the list, Django rejects incoming requests. By default, all hosts are permitted *;

- **DJANGO_CACHES_REDIS_URL** - the URL of the Redis server used as the Django caching system. Value by default: `redis://redis:6379/1`;
- **JOHNNY_BIN** - specifies the directory path to the `JOHNNY` executable scanning agent. Value by default: `/agents/johnny-linux-amd64`;
- **HASHER_BIN** - specifies the directory path to the `HASHER` executable agent, responsible for file hashing. Value by default: `/agents/hasher-linux-amd64`;
- **ANALYSIS_ROOT** - specifies the root directory for cloning repositories intended for analysis. Files are copied as temporary files and removed after analysis. Value by default: `/analysis-root`;
- **MEDIA_ROOT** - specifies the directory used for temporary storage of downloaded and uploaded files (including reports, SBOMs, etc.). Value by default: `/analysis-root`;
- **NODE_PATH** - specifies the directory used for containing `node_modules`. Value by default: `/jscpd/node_modules` ;
- **REQUESTS_CA_BUNDLE** and **SSL_CERT_FILE** - specifies the location of SSL-certificates used by the installation for secure communication:

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

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

- **USE_SMART_FILTERS** - enables the use of "smart filters" in Celery. These filters allow for more flexible control over task execution order. Available values: `True` or `False`. Default value: `True`;
- To enable proxy access for the installation to reach the internet or a version control system, uncomment and configure the necessary variables. Provide the proxy URL including the protocol, along with the username and password if necessary. For example:
  - **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 - provides queue and task management functionality. Modifications to the worker count should be performed only in consultation with technical support.

- **HUEY_REDIS_URL** - the URL for connecting to the Redis server used by the Huey. Default value: `redis://redis:6379/0`;
- **HUEY_WORKERS** - the number of Huey workers processing the tasks-main queue. Default value: 8;
- **HUEY_HIGH_PRIORITY_WORKERS** - the number of Huey workers for high-priority tasks. Default value: 8;
- **HUEY_OSA_PACKAGE_SCAN_WORKERS** - the number of Huey workers for the OSA package scanning module. Default value: 4;
- **HUEY_OSA_CONTAINER_IMAGE_SCAN_WORKERS** - the number of Huey workers for the OSA package scanning module. Default value: 4;
- **HUEY_POLICY_WORKERS** - the number of Huey workers for security policy execution. Default value: 4;
- **HUEY_TQI_WORKERS** - the number of Huey workers for the TQI module. Default value: 4;
- **HUEY_SECRETS_WORKERS** - the number of Huey workers for the Secret scanning module. Default value: 4;
- **HUEY_MEDIA_WORKERS** - the number of Huey workers for media upload processing. Default value: 4;
- **HUEY_SCA_EXTERNAL_SCAN_WORKERS** - the number of Huey workers responsible for the analysis queue, launched via the Johnny agent (`tasks-sca-external-scan`). Default value: 4;
- **HUEY_WEBHOOKS_WORKERS** - the number of Huey workers responsible for handling webhooks. Default value: 2;
- **HUEY_WORKER_MAX_TASKS** - the maximum number of tasks that a single Huey worker can process. Default value: `500`.

#Redis

Redis is an in-memory data store designed for caching and real-time data management. The following configuration variables define the parameters for interacting with the Redis server:

- **REDIS_BACKOFF_CAP** - specifies the maximum number of connection retry attempts in case of failure. If a connection cannot be established after this limit, an error will be returned. Default value: 5;
- **REDIS_BACKOFF_BASE** - base delay (in seconds) between retry attempts upon failure. Default value: `0.08`;
- **REDIS_RETRIES** - specifies the total number of connection attempts to the Redis server. Default value: 5;

- **REDIS_SOCKET_CONNECT_TIMEOUT** - specifies the timeout (in seconds) for Redis server operations. If an operation does not complete within this duration, an error will be raised. Default value: 5.0;
- **REDIS_SOCKET_TIMEOUT** - specifies the timeout (in seconds) for Redis server operations. If a connection is not established within this duration, a retry attempt will be made or an error will be raised. Default value: 5.0;
- **REDIS_SOCKET_KEEPALIVE** - enables a keep-alive functionality to maintain an active connection with the Redis server, preventing premature closure due to inactivity. Available values: True or False. Default value: True;
- **TASK_RESULT_EXPIRATION_PERIOD** - specifies the duration (in seconds) for which task results will be stored in Redis. After this time, they will be deleted. Default value: 14400.

#Celery

Celery is a task queue management system designed to offload core application processes and enhance responsiveness. Variables for configuring Celery worker settings:

- **CELERY_WORKER_CONCURRENCY** - specifies the number of tasks a single Celery worker can process concurrently. Default value: 6;
- **CELERY_WORKER_MAX_CONCURRENCY** - specifies the maximum number of tasks a single Celery worker can process. This parameter is used to prevent system overload when a large number of tasks are present. Default value: 12;
- **CELERY_MEDIA_WORKER_CONCURRENCY** - specifies the number of tasks a Celery worker responsible for processing media tasks (e.g., report generation) can process concurrently. Default value: 2;
- **CELERY_MEDIA_WORKER_MAX_CONCURRENCY** - specifies the maximum number of tasks a Celery worker responsible for processing media tasks can process. This parameter is used to prevent system overload when a large number of media tasks are present. Default value: 4;

#Archiving of CodeScoring OSA Data

The following variables configure data archiving for the CodeScoring system service - OSA. By default, the variables OSA_ARCHIVE_THRESHOLD_DAYS, OSA_ARCHIVE_AUTO_CLEANUP_ENABLED, OSA_ARCHIVE_RETENTION_PERIOD_DAYS, and OSA_ARCHIVE_CHUNK_SIZE are commented out. Uncomment them as needed to enable their use.

- **OSA_ARCHIVE_THRESHOLD_DAYS** - specifies the number of days after which a package or container image that has not received requests will be archived. Default value: 14;
- **OSA_ARCHIVE_RETENTION_PERIOD_DAYS** - specifies the number of days that archive packages or container images will be stored before

being deleted (when `OSA_ARCHIVE_AUTO_CLEANUP_ENABLED` is enabled). Default value: `30`;
- **OSA_ARCHIVE_AUTO_CLEANUP_ENABLED** - Enables or disables automatic cleanup of OSA archive components. If set to `True`, the system automatically removes archived packages after the retention period specified in `OSA_ARCHIVE_RETENTION_PERIOD_DAYS`. If `False`, cleanup is performed manually. Default value: `False`.
- **OSA_ARCHIVE_CHUNK_SIZE** - defines the size of "chunks" for processing archive components. This parameter can affect the performance and efficiency of the archiving/cleanup process. Default value: `1000`.

#Judge

The following variables are used to configure the Judge component, which is responsible for policy execution within the OSA Proxy service. These variables should only be uncommented and configured if the OSA Proxy is in use:

- **WEB_CONCURRENCY** - defines the number of workers for parallel validation of components or containers against security policies. Default value: `5`;
- **MIN_DATABASE_CONNECTION_POOL_SIZE** - defines the minimum pool of active database connections. The variable is commented out by default and has a value of `0`;
- **MAX_DATABASE_CONNECTION_POOL_SIZE** - defines the maximum pool of active database connections. The variable is commented out by default and has a value of `10`.

#Installation Service Overview

This section provides an overview of the core services used by the system.

#Core Components

- **Frontend** – the user interface, responsible for system interaction. Displays analysis results, configuration settings, and policy parameters.
- **Backend** – handles requests from the Frontend and orchestrates the operation of other system components;
- **Celery-beat** – scheduler responsible for initiating tasks within the Celery-worker according to a defined schedule;
- **Celery-worker** – executes tasks received from various services (e.g., Celery-beat and Backend);
- **Collectstatic** – responsible for building static files required for system operation;
- **Fluentd** – logging agent that aggregates logs from all system components;
- **Index-proxy** – enhances queries to the index (https://index.codescoring.ru) by adding supplementary data and headers;
- **Migrate** – performs system configuration upon startup and updates the database schema;

- **Osa-api** – facilitates interaction between the core system and the OSA plugin (installed in Sonartype Nexus Repository or JFrog Artifactory) via API;
- **Osa-registration** – responsible for scanning and validating packages and container images for policy compliance;
- **Pgbouncer** – manages and optimizes database connection requests;
- **Psql** – a PostgreSQL client used for managing the database;
- **Redis** – a caching and message broker, providing data storage and component coordination;
- **Tasks-high-priority** – a queue for executing high-priority tasks;
- **Tasks-main** – a queue for executing standard tasks, such as project scanning, VCS upload, policy deployment, etc.;
- **Tasks-main-scheduler** – a scheduler for periodic tasks (e.g., scheduled scanning);
- **Tasks-media** – a queue for uploading and downloading media files;
- **Tasks-osa-container-image-scan** – responsible for scanning container images conforming to Docker, OCI and Singularity standards for vulnerability detection;
- **Tasks-osa-package-scan** – responsible for scanning packages for vulnerabilities;
- **Tasks-policy** – executes tasks to verify policy compliance;
- **Tasks-sca-external-scan** – a queue for scans initiated via the Johnny console agent;
- **Tasks-tqi** – the CodeScoring TQI (Teams and Quality Intelligence) module agent, analyzing source code quality;
- **Tasks-secrets:** – the CodeScoring Secrets module agent, responsible for identifying sensitive information within code;
- **Judge** - responsible for policy enforcement within the OSA Proxy service.

#Troubleshooting

#Working with logs

1. Go to the directory with the startup files:

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

1. Execute the command to copy the log file from the container to the file `codescoring_onprem.log`

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

1. Send the `codescoring_onprem.log` file to the vendor.

#System re-installation

If you want to start the installation process from scratch, you need to clean up the volumes.

If there are no other containers on the server with docker other than the CodeScoring project, run the command:

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

If there are other docker projects on the server:

1. stop docker compose:

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

1. run the command:

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

1. If an error occurs that this volume is being used by the container, you should run the command and repeat the previous steps (`CT_HASH` will be in the error message):

```
docker rm CT_HASH
```

If the problem persists, contact your vendor support contact for further instructions.

#System activation

#Entering the activation key

To operate the system, it must be activated using a key that is transferred to the client in a separate `txt` file.

To enter the key, go to the `Settings -> Activation` key section, copy the text from the file into the field without any changes and click the **Save** button. If successful, information on the key will appear on the page and the Status field will change to **Active**.

#Key parameters

If the system is successfully activated, the `Settings -> Activation` key section displays the parameters of the key used:

- **Status** – system activation status;

- **Owner** – name of the organization in whose name the key was issued;
- **Issued** – day the key was issued;
- **Expiration** – day the key expires;
- **Authors limit** – the maximum number of authors (developers) for which the system is licensed;
- **Private feeds** – a list of connected private vulnerability databases, such as **Kaspersky OSS Threats Data Feed**;
- **Available modules** – a list of connected modules with the key expiration date for each.

#Activation key expiration

When the activation key expires, the platform continues to operate with limitations:

- Previously performed scans and their results remain available for viewing;
- An attempt to perform a new scan results in an error related to the activation key expiration.

#Calculating the number of authors

The author audit feature is available in the platform starting with version **2025.37.0**. It allows you to get statistics on unique authors of commits in selected repositories.

You can calculate the number of unique authors in the `Settings → Activation` Key section by clicking the **Calculate** button.

#Supported platforms

Auditing is currently available for the GitLab platform.

#Configuring parameters

After clicking the button, a form with analysis parameters opens:

- **Provider** — repository data source. **GitLab** is currently supported;
- **VCS URL** — host address where the repositories are hosted (without specifying a specific repository). Example:

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

- **Access Key** — a personal token with `read_api` permissions. The token is not saved and is used only for the current calculation. In GitLab, the token can be obtained at:

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

- **Author Email Filter** — a list of domains or regular expressions for filtering authors. Multiple values can be specified, separated by `|`.

Important

The VCS URL must begin with the data transfer protocol. The trailing slash is optional.

#Performing analysis

After filling out the form, click **Calculate**. While the scan is in progress, you can cancel the process by clicking **Cancel**.

#Calculation Results

After the analysis is complete, the following data is displayed in the Activation Keys section:

- **Status** — shows the result of the analysis. If an error occurs, the cause can be found in the **Audit log** section;
- **Date** — the date and time the analysis was performed;
- **Unique authors per year** – the number of commit authors over the last 365 days. This parameter can be changed via the API;
- **Repos processed with success** – the number of repositories processed without errors;
- **Repos processed with error** – the number of repositories with read errors. Details can be found in the backend logs;
- **JSON format report URL** – a link to the full report in JSON format with analysis details and a list of processed repositories.

#Troubleshooting

If the analysis failed:

1. Check the **VCS URL** and the correctness of the **token**. A **401** error typically indicates an invalid URL or an invalid token.
2. Additional information can be found in the **Audit log** section.

#Managing user accounts

#Creating accounts

The CodeScoring platform supports multiple users with separate accounts. User accounts are created and managed in the `Settings -> Users` section.

To create a new user, go to the form by clicking the **Create New** button and fill in the following fields:

- **Username** — username in the system;
- **First name** — first name;
- **Last name** — last name;
- **Contact email** — email;
- **Proprietor** — affiliation with the organization's department within the system;
- **Access level** — access level within the system;
- **Password** — password for logging into the system;
- **Can create CLI projects via API** – ability to create CLI type projects using API.

The list of created users can be filtered by the following parameters:

- **Proprietor**;
- **Access level**;
- **Active** — an active account;
- **LDAP server** — LDAP server connected to the system.

#Editing account settings

Created accounts can be edited or deleted in the `Settings -> Users` section. You can add a user to a project with the specified role by clicking the **Add users** button on the "Projects" tab of the user editing page.

The session time for an inactive user is limited. By default, a user's session ends 2 weeks after the last activity, after which you must log in to the system again.

The environment variable (in seconds) `SESSION_COOKIE_AGE` is available for configuring the session lifetime.

Important

```
Modifying `username` or `password` of users from external
identity providers is prohibited.
```

#Separation of access levels

When creating an account, it must be assigned one of the following access levels - **User**, **Administrator**, **Auditor** or **Security Manager**.

#Administrator access level

The Administrator access level provides access to all projects. This access level also allows to view and change all settings in the system without restrictions.

#Security Manager access level

The Security Manager access level provides access to all projects. This access level also allows you to view all security-related sections and run all scan types, but only change policies.

#Auditor access level

The Auditor access level provides access to all projects. This access level also allows to view all settings and projects in the system without the ability to make and save changes.

#User access level

For the User access level, access is organized individually. For each project, access can be provided with the following roles:

- **Viewer** — access only to view analysis results within the project;
- **Developer** — all rights of the Viewer role, as well as access to launching analysis in the web interface, via the agent and via the proxy repository plugin;
- **Owner** — all rights of the Developer role, access to changing project settings and managing access of other project users.

For each role within the **User** access level, CLI project creation via API is available when the **Can create CLI projects via API** parameter is activated in the user profile.

A project can have multiple users with the same roles, including multiple **Owner**. If there are no users in the **Owner** role, only a user with the **Administrator** access level can manage the project.

#Available actions

A more detailed list of available actions for each access level is presented in the table below:

Action	User (Viewer)	User (Developer)	User (Owne
Analysis: viewing the analysis results	✔	✔	✔
Analysis: set analysis results	✖	✔	✔
Analysis: launching SCA analysis	✖	✔	✔
Analysis: launching Secrets analysis	✖	✔	✔

Action	User (Viewer)	User (Developer)	User (Owne
Analysis: set Secrets status	✗	✗	✓
Analysis: launching Authors analysis	✗	✗	✗
Analysis: launching Quality analysis	✗	✗	✗
Activation key: viewing information about the activation key	✗	✗	✗
Activation key: saving the activation key	✗	✗	✗
Audit log: view audit log	✗	✗	✗
Audit log: export audit log	✗	✗	✗
Authors merge: view rules	✗	✗	✗
Authors merge: creating rules	✗	✗	✗
Dashboard: viewing the page	✓	✓	✓
Dependencies: viewing list of dependencies	✓	✓	✓
Dependencies: export list of dependencies	✓	✓	✓
Email: viewing email settings	✗	✗	✗
Email: editing email settings	✗	✗	✗
Groups: viewing user groups	✗	✗	✗
Groups: creating user groups	✗	✗	✗
Groups: editing user groups	✗	✗	✗
Groups: deleting user groups	✗	✗	✗
LDAP: viewing LDAP settings	✗	✗	✗

Action	User (Viewer)	User (Developer)	User (Owne
LDAP: editing LDAP settings	✗	✗	✗
OSS Index: viewing OSS Index settings	✗	✗	✗
OSS Index: editing OSS Index settings	✗	✗	✗
Policies: viewing policies	✓	✓	✓
Policies: creating policies	✗	✗	✗
Policies: editing policy settings	✗	✗	✗
Policies: delete policies	✗	✗	✗
Policy alerts: view list of alerts	✓	✓	✓
Policy alerts: export list of alerts	✓	✓	✓
Policy alerts: send notifications	✓	✓	✓
Policy ignores: view rules	✓	✓	✓
Policy ignores: creating rules	✗	✗	✗
Policy ignores: editing rules	✗	✗	✗
Policy ignores: removing rules	✗	✗	✗
Projects: viewing projects	✓	✓	✓
Projects: viewing Contribution map	✓	✓	✓
Projects: viewing Complexity map	✗	✗	✗
Projects: creating projects	✗	✗	✗
Projects: editing project settings	✗	✗	✗
Projects: deleting projects	✗	✗	✗

Action	User (Viewer)	User (Developer)	User (Owne
Projects: managing group permissions for projects	✗	✗	✓
Projects: managing user permissions for projects	✗	✗	✓
Projects: SBOM upload	✗	✓	✓
Projects: editing dependencies for SBOM export	✗	✓	✓
Project categories: view categories	✗	✗	✓
Project categories: creating categories	✗	✗	✗
Project categories: editing categories	✗	✗	✗
Project categories: removing categories	✗	✗	✗
Proprietors: viewing code owners	✗	✗	✓
Proprietors: creating code owners	✗	✗	✗
Proprietors: editing code owners	✗	✗	✗
Proprietors: removing code owners	✗	✗	✗
Task managers: view integrations	✗	✗	✗
Task managers: adding integrations	✗	✗	✗
Task managers: editing integration settings	✗	✗	✗
Task managers: removing integrations	✗	✗	✗
Task managers: performing configuration checks	✗	✗	✗

Action	User (Viewer)	User (Developer)	User (Owne
Users: viewing users	✖	✖	✖
Users: creating users	✖	✖	✖
Users: editing user settings	✖	✖	✖
Users: deleting users	✖	✖	✖
VCS: browsing list of repositories	✖	✖	✖
VCS: adding repositories	✖	✖	✖
VCS: editing repository settings	✖	✖	✖
VCS: deleting repositories	✖	✖	✖
VCS: performing settings check	✖	✖	✖
Vulnerabilities: viewing a list of vulnerabilities	✔	✔	✔
Vulnerabilities: export list of vulnerabilities	✔	✔	✔

Important

To be able to run a scan, also make sure that the appropriate analysis module is included in the license.

#User groups

Users within the system can be divided into groups. Groups are created and managed in the `Settings -> Groups` section.

To create a new user group, you must go to the form using the **Create New** button and fill in the following fields:

- **Name** — group name;
- **Description** — group description.

Groups can be added to created projects to more easily track users associated with a project.

#Creating groups

Groups are used to join users and associate them with specific projects. This allows to conveniently manage mass access to projects in the system.

Groups are managed in the `Settings -> Groups` section. You can go to the group creation form by clicking the **Create new** button. In the form, you must fill in the field **Name** and optionally specify a group description and icon color.

#Configuring groups

Each group has its own list of users and a set of associated projects. You can change these parameters using the **Edit** button.

You can link a project to a group using the **Add project** button. After linking a group, each user will have access to the project according to the assigned role.

When adding a new user to a group using the **Add user** button, you must select an existing account in the CodeScoring system and assign one of the roles within the group:

- **Viewer** - access only to viewing the results of project analyzes;
- **Developer** - access to launch analysis in the web interface, through an agent and through a proxy repository plugin;
- **Owner** - access to view project policies, change project settings and manage access of other project users.

Within the system, it is also possible to configure mapping of LDAP groups into CodeScoring groups and roles.

#LDAP Settings

#LDAP integration capabilities

CodeScoring supports authentication and authorization of users via **LDAP** and mapping of user attributes in **LDAP** to user attributes in the system.

#CodeScoring Authentication Page

On the authentication page, a menu is available with a choice of authentication provider. In addition to the default provider (local accounts, `internal directory`), active integrations with **LDAP** servers are available for selection.

Login provider

Internal directory

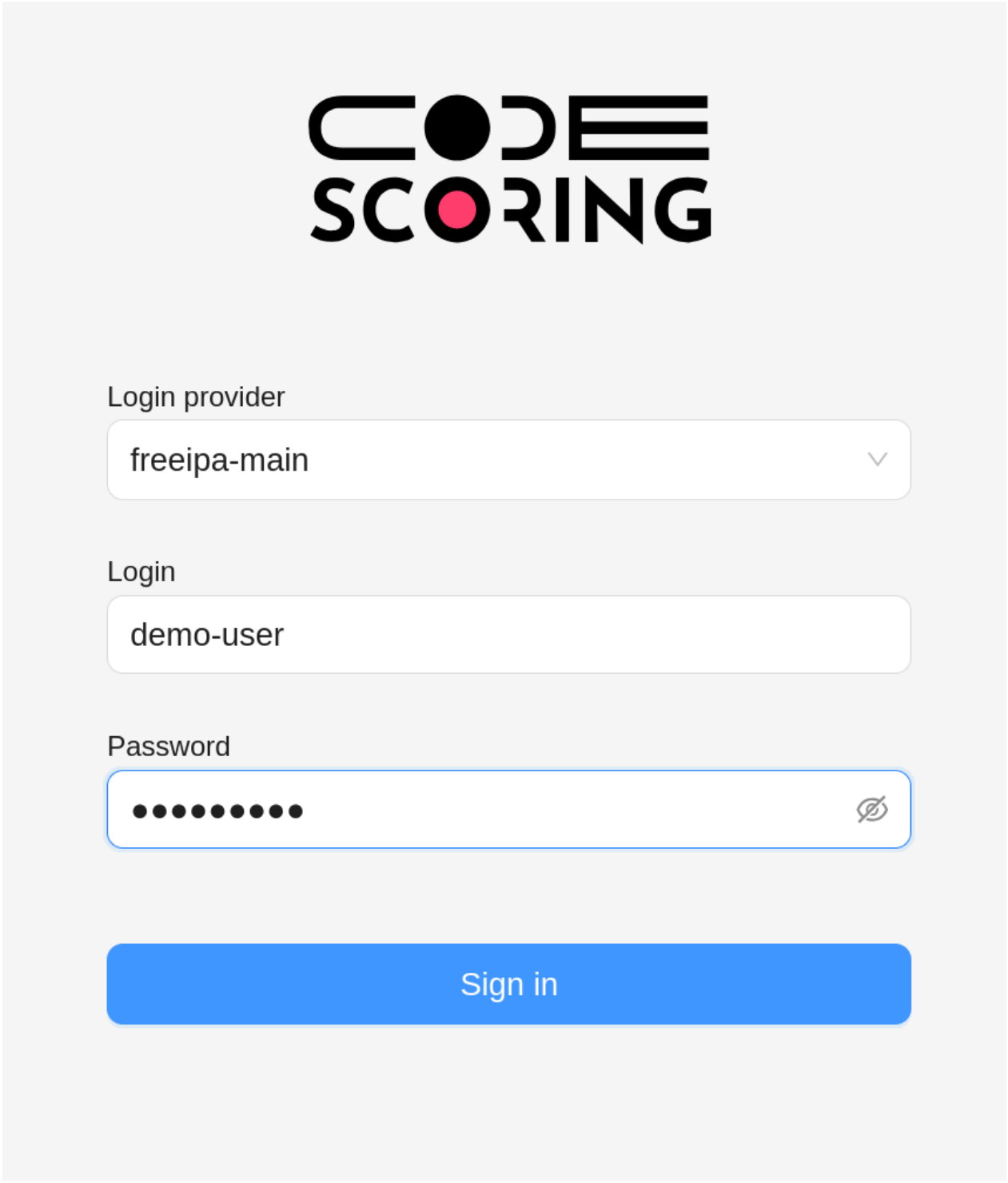
Internal directory internal

freeipa-main ldap

keycloak-main oidc

Password

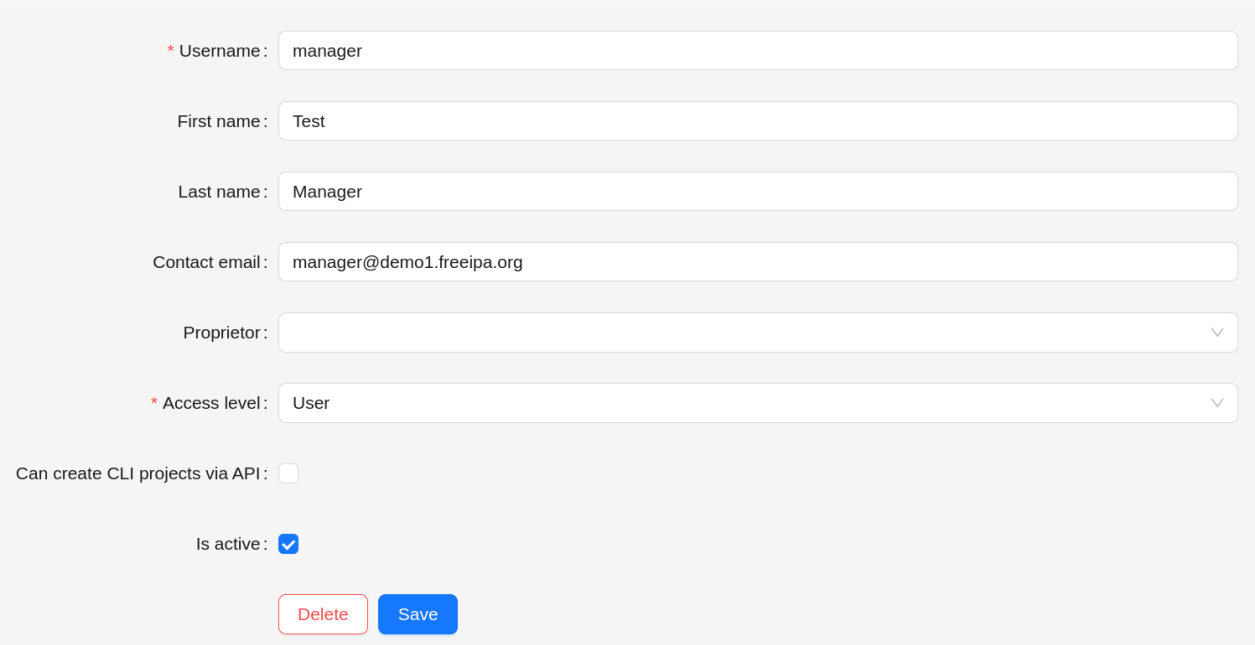
Sign in



#Mapping LDAP user record attributes to CodeScoring user attributes

When authenticating via **LDAP**, the following data is mapped from a directory entry to an account in CodeScoring:

- username;
- first name;
- last name;
- e-mail.



#Mapping LDAP groups to CodeScoring groups and roles

When authenticating via LDAP, CodeScoring can query a user's LDAP group data and map it to its own access levels, groups and roles. To activate this feature, the following fields must be filled in the LDAP settings:

- `Group search base`.
- `Group search filter for all groups`
- `Group name field`
- `Group member field` (if `User group search method` field value equals "User record")
- `Group search filter for user` (if `User group search method` field value equals "Group search")

CodeScoring performs mapping according to the rules, the management of which is available in the `Settings -> Group mapping` section.

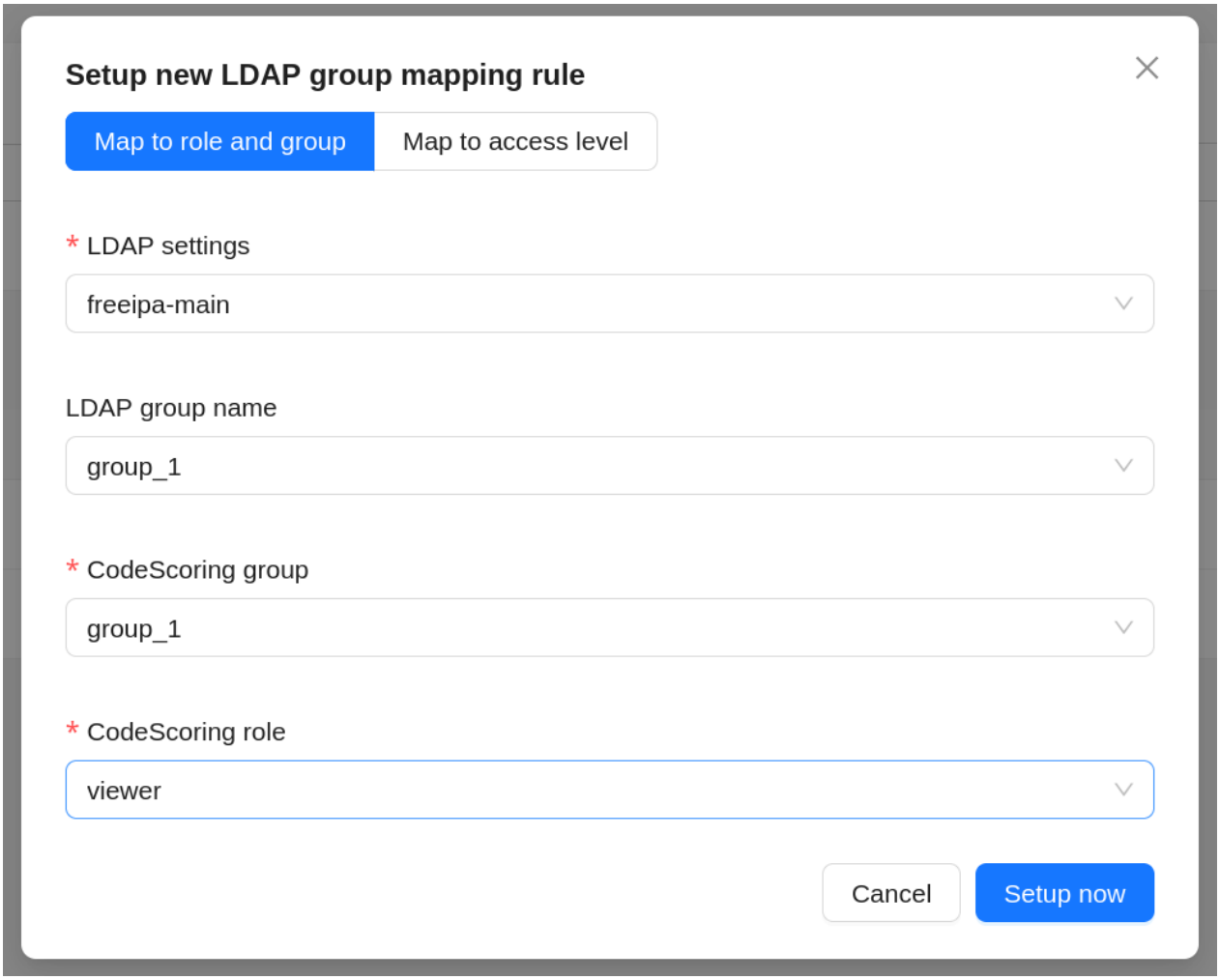
Mapping is triggered only in two cases:

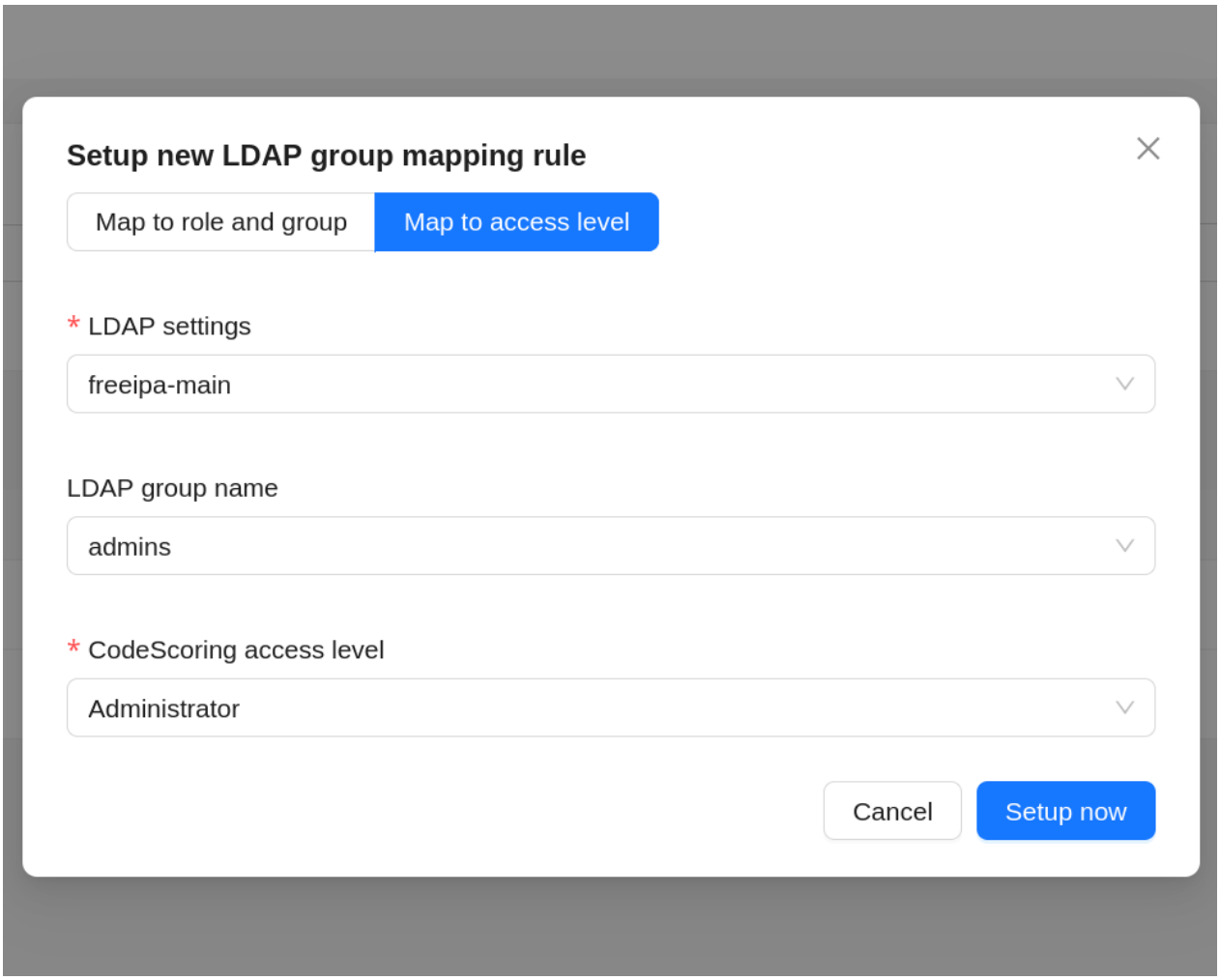
- when the user logs in;
- when the `Apply all rules` button is pressed in `Settings -> Group mapping` section.

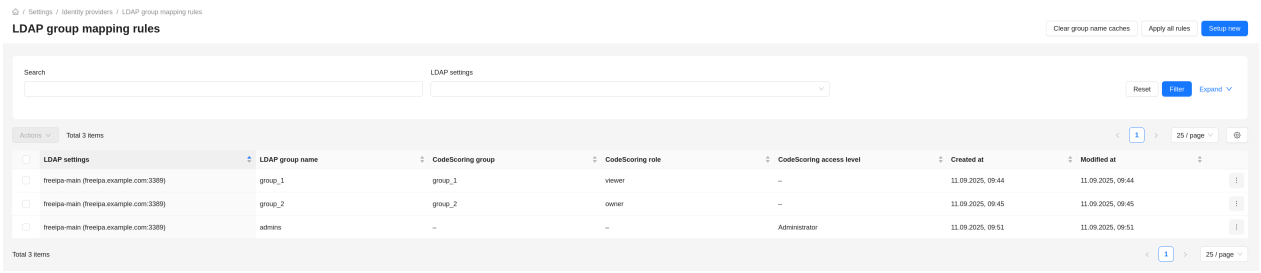
Mapping rules are applied as follows:

- during the mapping process the composition of the user's groups is updated;
- adding, modifying and deleting mapping rules will not update the user groups. In order to apply the changed rules, the mapping process must be initiated in one of the two ways specified above;
- user groups added or edited manually by the administrator are not changed during the mapping process, i.e. manual changes take precedence;

- if an error occurs when searching for LDAP groups after successful authentication in LDAP, the user groups applied according to the mapping rules will be deleted.

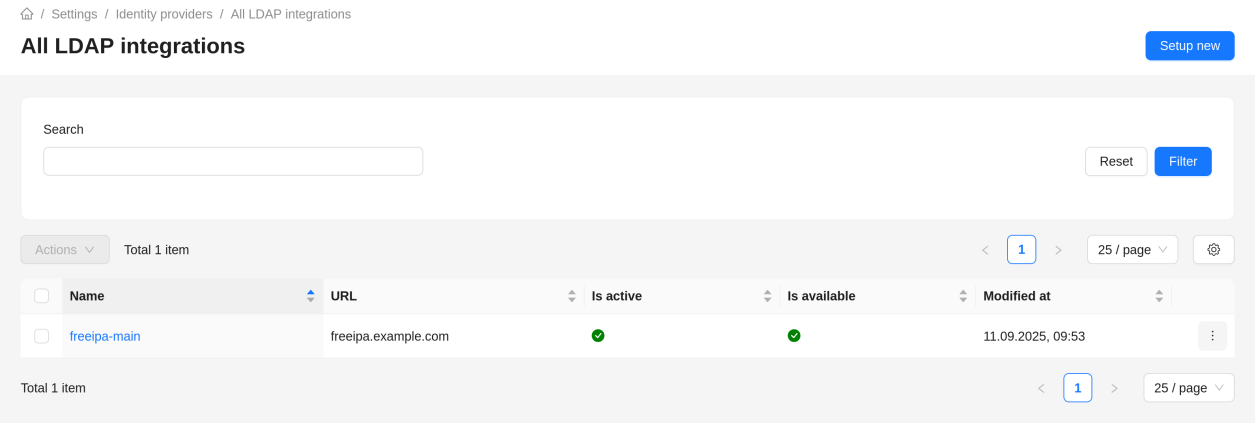






#View existing LDAP integrations

Viewing existing integrations is available in the `Settings -> LDAP` section. The section displays a table with a list of configured LDAP integrations, a button to create a new integration (`Setup new`), and a search box (`Search`).



#View details about an existing LDAP integration

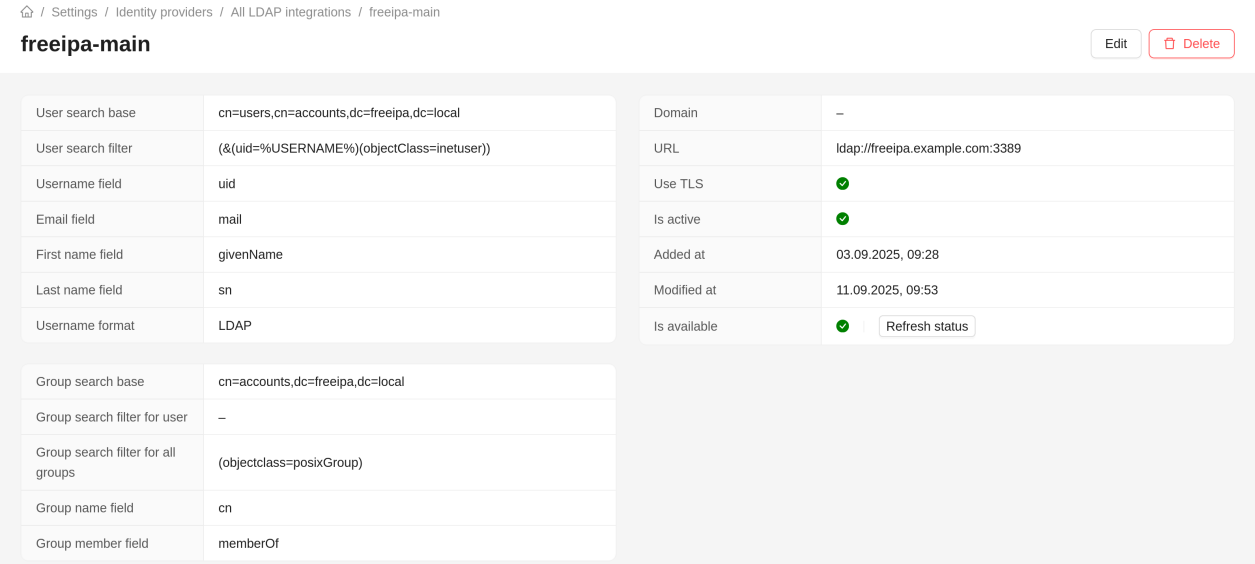
Detailed view opens by clicking on the link with the name of the integration or by clicking the **View** button in the Actions section.

When viewing, the following actions are available:

- deleting an integration (`Delete`);
- editing the integration (`Edit`);
- checking availability (`Refresh status`).

In addition to the basic configuration fields (described below), when viewing details about an integration with **LDAP**, data about:

- creation date;
- date of the last update;
- availability status;
- (optional) reason for unavailability.



#Create or edit an LDAP integration

In order to create a new integration with **LDAP**, you must click on the `Setup new` button in the list of integrations view. In order to edit an existing

integration, you must click the Edit button on the page or in the list of integrations. The create and edit forms are identical.

#Description of form fields

- Name - name of the integration, displayed on the authentication page;
- LDAP protocol - choice between **LDAP** and **LDAPS** protocols (**LDAP** over **SSL**);
- LDAP hostname - domain name or IP address of the **LDAP** server;
- LDAP port - port of **LDAP** server;
- User search base - specifies where to search for user records;
- User search filter - filter for searching user records. Must contain the %USERNAME% pattern, when searching it will be replaced by the name of the user;
- Username field - attribute storing the name of the user's KM in the record about it;
- Email field - attribute storing the user's email in the record about it;
- First name field - attribute storing user's name in its record;
- Last name field - attribute storing the user's last name in the user's record;
- Username format - selection of authorization string format;
- Domain - specifying the **Active Directory** domain (appears when the corresponding **Username format** is selected);
- Service user - login of the user with permissions to read directories on the **LDAP** server;
- Service user password - password of the user with directory reading rights on the **LDAP server**;
- Group search base - specifies where to search for group records;
- Group search filter for user - filter to search for records of groups to which the user belongs. Must contain the %USERNAME% template, during the search it will be replaced by the name of the KM for which authentication via **LDAP** is performed;
- Group search filter for all groups - filter to search for records of all groups in **LDAP**;
- Group name field - attribute that stores the name of the group in the group record;
- Group member field - attribute that stores group member identifiers in the group record;
- Use tls - use **TLS**;
- Is active - whether integration with **LDAP** is used for user authentication.

* 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

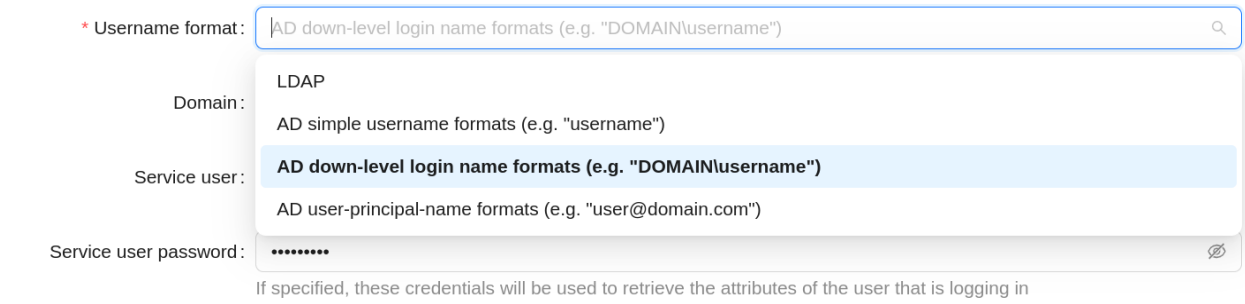
memberOf

Is active

Use TLS

Save Cancel

#Available options for username format



#Testing the LDAP integration configuration

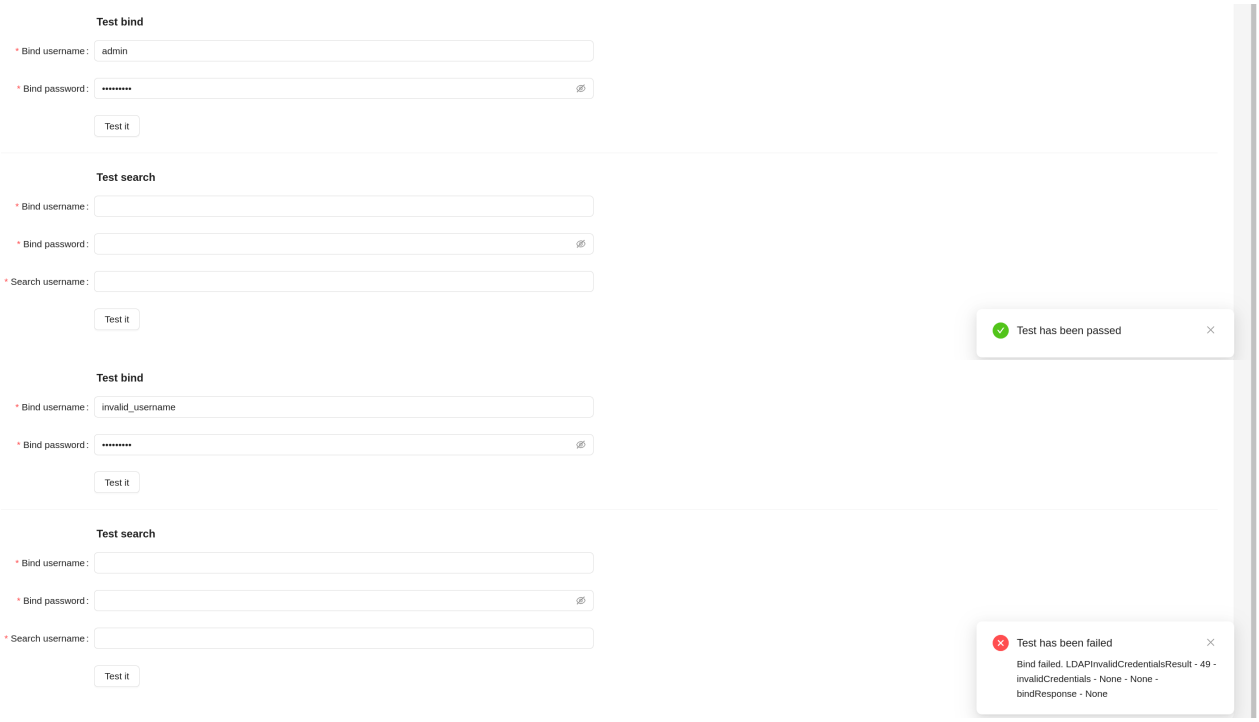
For ease of configuration, 2 forms for connection testing are available to users:

- connection and authentication testing (`test bind`);
- testing of search (`test search`).

Both tests combine data from the main form with data from the test form. Data from the fields `Service user` and `Service user password` fields are ignored.

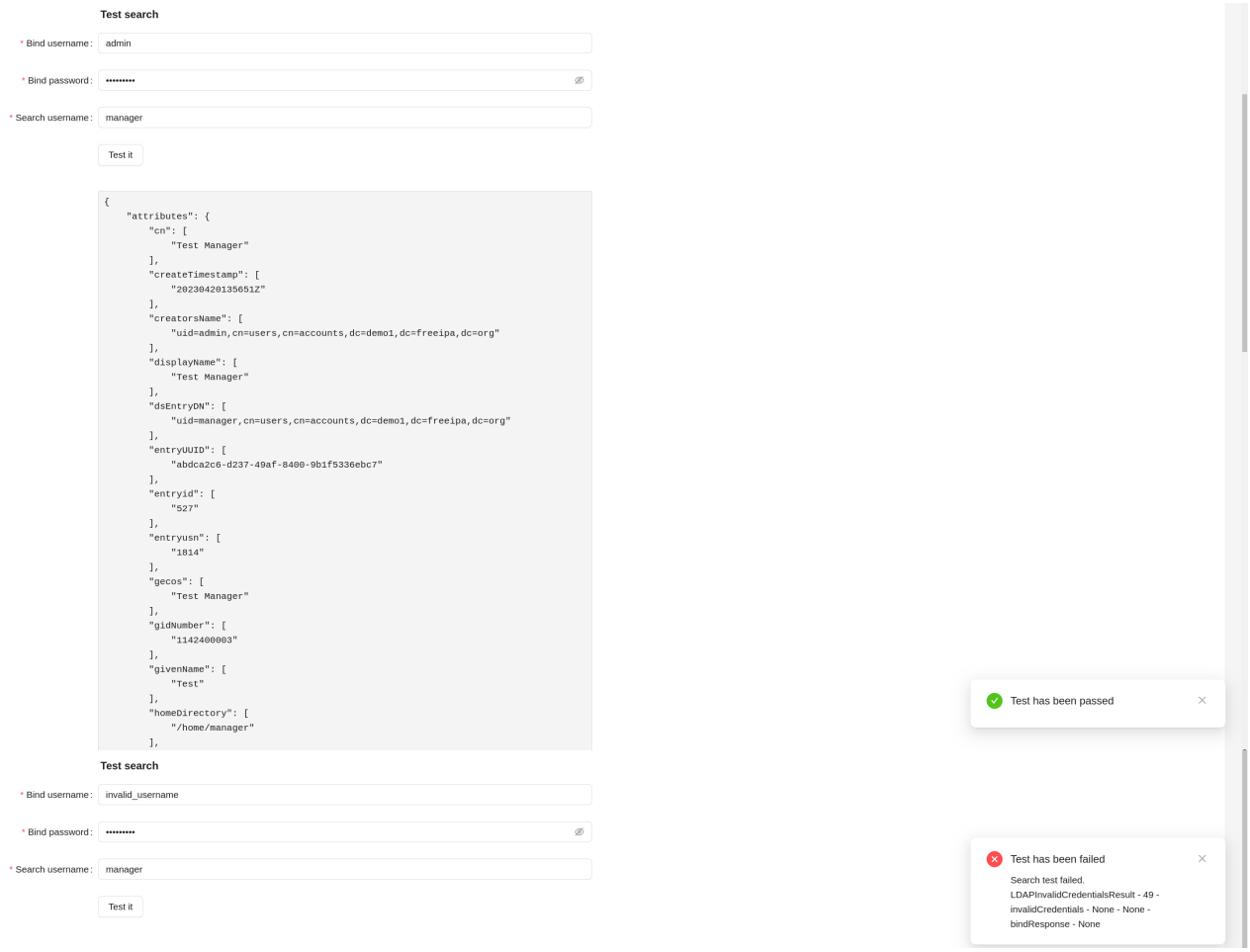
#Connection and authentication testing

When the button (`Test it`) in the **Test bind** section is clicked, connection to the LDAP server is made (`bind` operation). In case of successful test a notification about the success of the operation is displayed. In case of test failure - an error message.



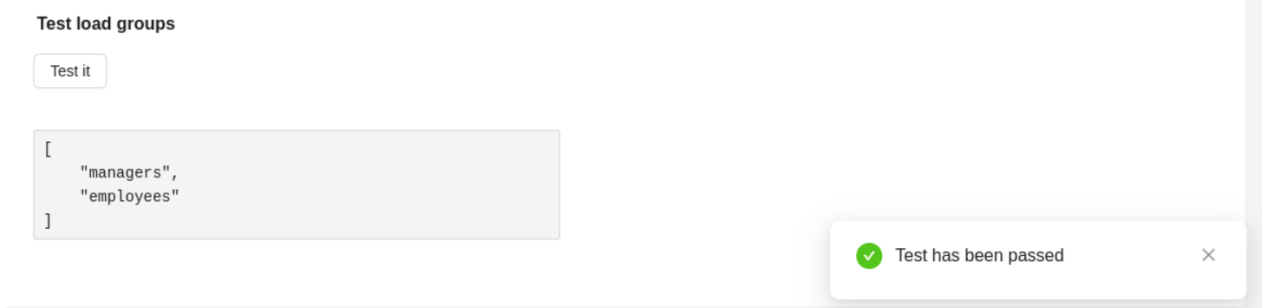
#User Data Load Test

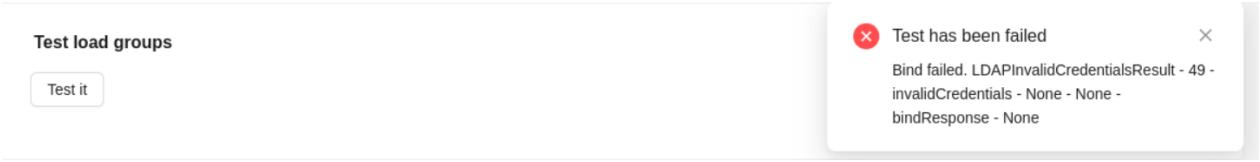
When the button (`Test it`) in the **Test user search** is clicked, the connection to the LDAP server is made (`bind` operation) and the data about the user data (`search` operation) according to the data in the form. In case of a successful test, a notification about the success of the operation and the search result is displayed. In case of test failure - an error message.



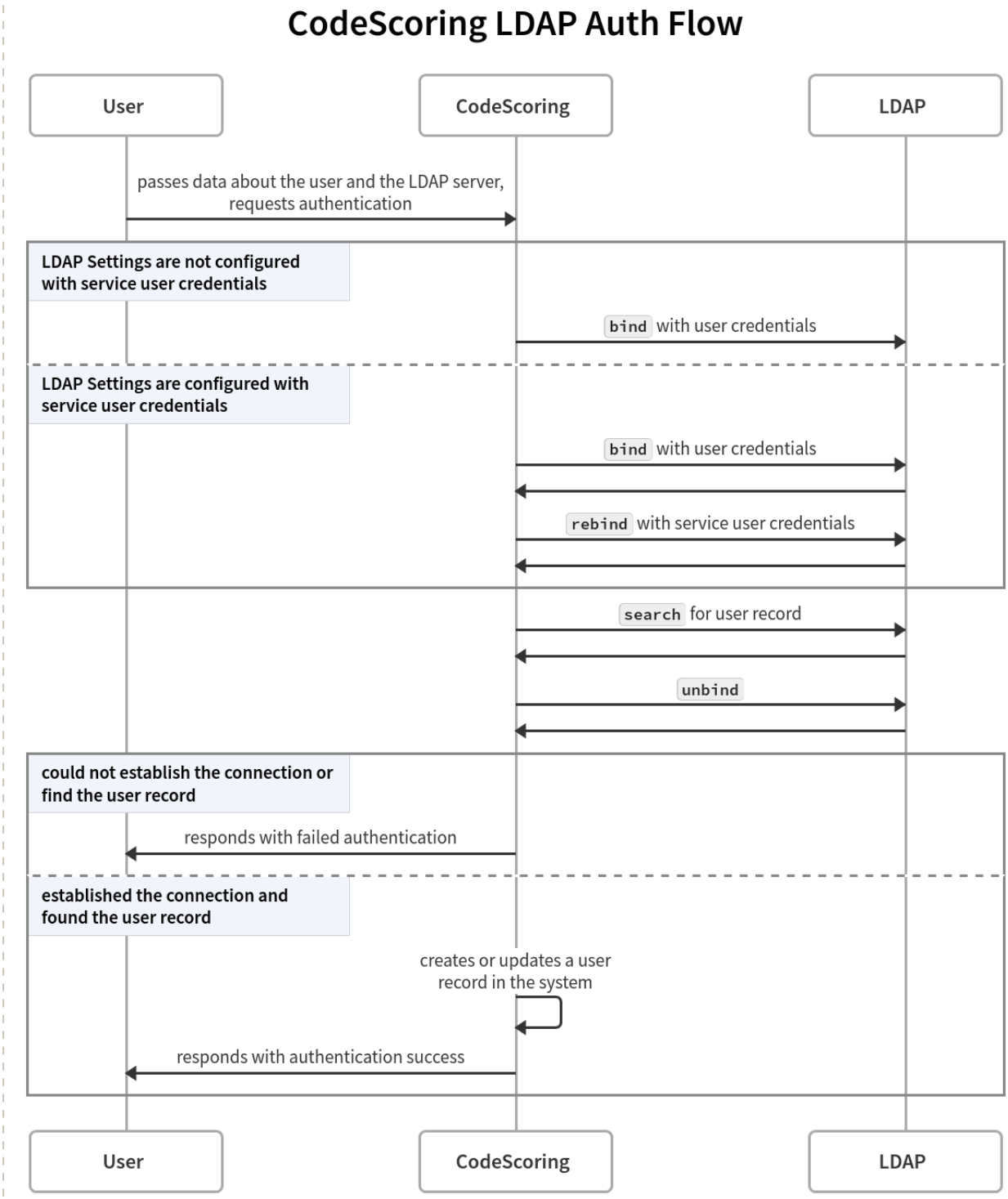
#Group Data Loading Test

When the test button (`Test it`) in the **Test load groups** is clicked, the connection to the LDAP server (`bind` operation) is made and search for data on the groups (`search` operation) according to the data in the form. In case of a successful test, a notification about the success of the operation and the search result is displayed. In case of test failure - an error message.





#LDAP authentication mechanism



#Notes

- Using authentication via **LDAP** does not imply full directory synchronization with user information from **Directory Service**.
- It is not possible to edit the username and assign a password to a user from **LDAP**.
- It is possible to have users from different authentication providers with the same username.
- It is possible to assign any access level (`User`, `Auditor`, `Administrator`, `Security Manager`) to a user from **LDAP**.

#Setting up OpenID Connect

#CodeScoring Authentication Page

CodeScoring supports authentication and authorization of users via **OpenID Connect**. On the authentication page, a menu is available with a choice of authentication provider. In addition to the default provider (local accounts, `internal directory`) and LDAP, active integrations with **OpenID Connect** servers are available for selection.

Login provider

Internal directory

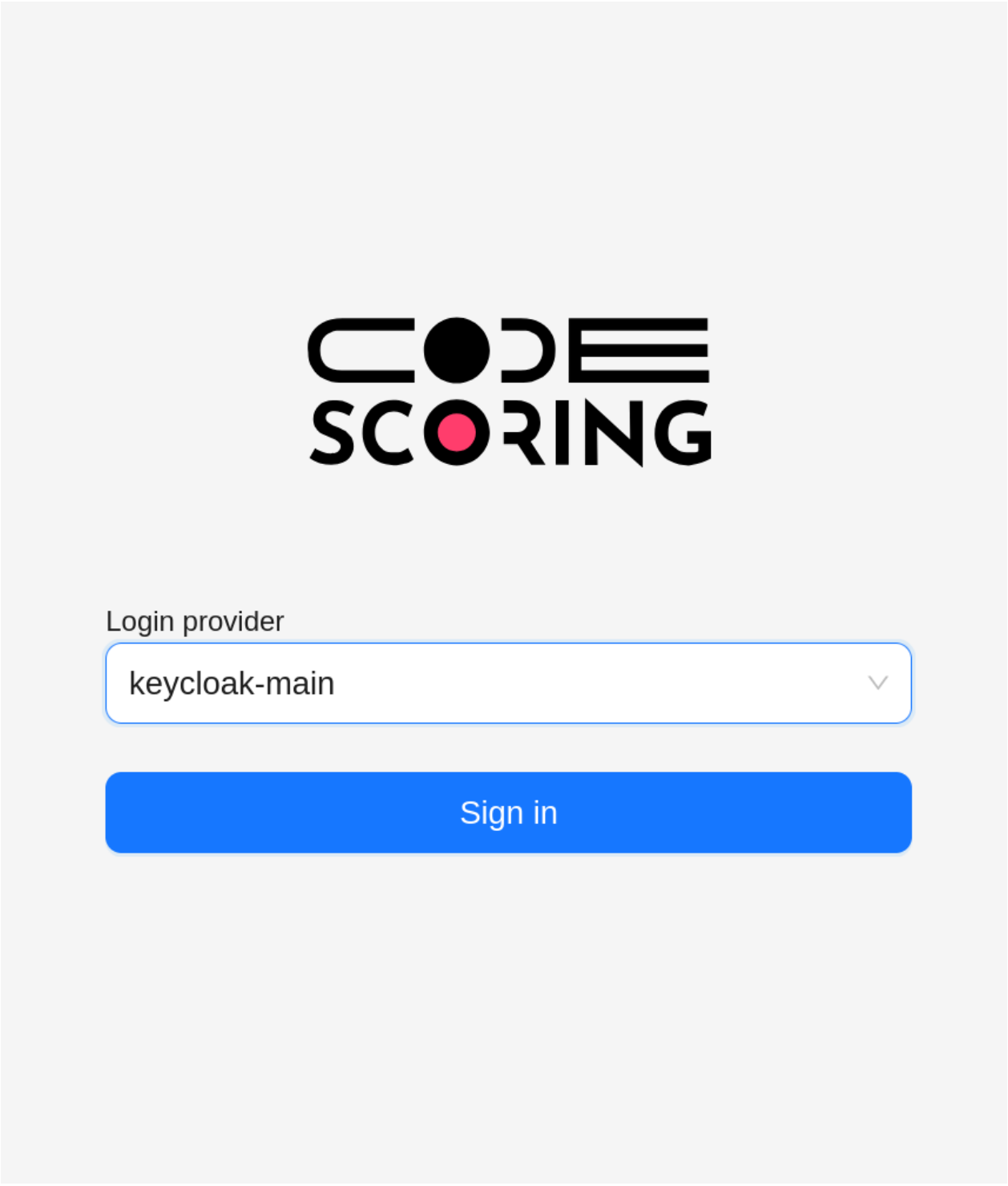
Internal directory internal

freeipa-main ldap

keycloak-main oidc

Password

Sign in



#Configuring the OpenID Connect Client on the Side of Provider

Below is an example of valid OpenID Connect client configuration on the side of provider given that the URL of the CodeScoring platform is `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.

#Configuring OpenID Connect Client on the side of CodeScoring

In order to configure OpenID Connect, go to `Settings -> Identity Providers -> OpenID Connect`.

You can go to the OpenID Connect integration creation for by clicking the **Create new** button. Fill in the fields below:

#Main OIDC Settings

- **Name** - the name of the integration, it is displayed at the login page;
- **Is active** - the activity flag, depending on the value, users will be able to select this integration as an identity provider at the login page;
- **Client ID** - identifier of the OpenID Connect Client, must match the identifier configured on the Provider side;
- **Client secret** - secret of the OpenID Connect Client, must match the secret configured on the Provider side.

#Connection settings

CodeScoring can configure itself based on the `/.well-known/openid-configuration` endpoint of the issuer via the `Get settings` button.

- **Provider issuer URL** - Issuer URL of the OpenID Connect provider;
- **Provider authorization URL** - Authorization URL of the OpenID Connect provider, can be obtained at `$ISSUER_URL/.well-known/openid-configuration`;
- **Provider token URL** - Token URL of the OpenId Connect provider, can be obtained at `$ISSUER_URL/.well-known/openid-configuration`;
- **Provider UserInfo URL** - UserInfo URL of the OpenID Connect provider, can be obtained at `$ISSUER_URL/.well-known/openid-configuration`;
- **Provider JWKS URL** - JWKS URL of the OpenID Connect provider, can be obtained at `$ISSUER_URL/.well-known/openid-configuration`;
- **Client scopes** - these values are passed in the `scope` parameter of the authorization request.

#Field mapping settings

- **Subject identifier field** - name of the field that contains the **subject** in the response, returned by UserInfo endpoint;
- **Username field** - name of the field that contains the **display username** in the response, returned by UserInfo endpoint;
- **First name field** - name of the field that contains the **first name** in the response, returned by UserInfo endpoint;

- **Last name field** - name of the field that contains the **last name** in the response, returned by UserInfo endpoint;
- **Email field** - name of the field that contains the **email** in the response, returned by UserInfo endpoint.

#OpenID Connect-managed accounts in CodeScoring

During the authentication via OpenID Connect, Codescoring will create or update a CodeScoring user account.

The following fields are mapped from the UserInfo response to the CodeScoring users:

- Subject identifier;
- Username;
- First name;
- Last name;
- Email.

Accounts, created via OpenID connect, can be added to groups, and their access level can be elevated the same way, as it is done for regular CodeScoring users.

For CodeScoring user accounts, managed by OpenID Connect, it is impossible to modify the username or set the password.

#Architecture

CodeScoring.Save uses a microservice architecture. Backend API and authentication / authorization are split into separate services: **Save API service** and **cs-auth (Auth/RBAC service)**.

#Technology Stack

- **Development language:** Go 1.25+.
- **Metadata storage:** PostgreSQL 14+ (recommended for production) or SQLite 3.x.
- **Object storage:** S3-compatible storage (MinIO, AWS S3, Yandex Object Storage) or local file system.
- **Authentication:** RSA-2048 JWT with JWKS, issued by cs-auth and validated locally by Save through the public key.

#Architecture Diagram

Database schema

In a **PostgreSQL** deployment, both services can share one database instance and one schema; data is separated by table name prefixes. In a **SQLite** deployment, each service uses a separate database file.

#System Components

#Save API service

The main backend service that processes incoming requests to repositories and the administrative API.

Functions:

- Processing HTTP/HTTPS requests from package managers and the web interface.
- Token validation and integration with the dedicated Auth/RBAC service.
- Routing requests to the storage backend.
- Integration with CodeScoring.OSA through OSA Proxy for artifact scanning when downloads are routed through the proxy.
- Applying security policies and cleanup policies.
- Structured logging and Prometheus metrics export.

Technical characteristics:

- Stateless architecture for horizontal scaling.
- Graceful shutdown during updates.
- Health checks for Kubernetes.

#Auth/RBAC service

A dedicated authentication and authorization service deployed separately from Save API service.

Functions:

- Authentication of users and service accounts.
- Issuing and refreshing JWT tokens (RSA-2048, alg=RS256).
- Managing roles and permissions at global / project / repository levels.
- Managing API keys, robot accounts, and project membership.
- Audit of access-related and system-wide operations.
- Internal API for interaction with Save API service.
- OCI Distribution Spec token endpoint for Docker / Helm / oras clients.

Technical characteristics:

- Separate codebase and separate entry point.
- Dedicated PostgreSQL schema in a shared database instance, or a separate SQLite file.
- Horizontal scaling independently from Save API service.
- **Required scaling condition:** all replicas must share the same RSA key for signing JWT tokens.

#Storage Backend

Artifact storage subsystem:

- **Metadata Store** — PostgreSQL or SQLite for metadata.
- **Blob Store** — object storage for artifacts: S3-compatible storage or file system.

Operations: storing and retrieving artifacts, version management, checksum generation.

#Background Workers

Background processes for deferred tasks:

- Cleanup policies — scheduled removal of outdated artifacts.
- Background metadata operations.
- Asynchronous delivery of audit events to cs-auth.

#Web UI

Web interface for managing projects, repositories, artifacts, and access settings. Module capabilities are described on the Functional characteristics page.

#Service Interaction

- External clients (Web UI and package managers) access only Save API service through ingress; cs-auth is not exposed outside the cluster.
- Save API service delegates authentication, role checks, and token issuing to Auth/RBAC service.
- Both services can use separate or shared PostgreSQL instances, but they work in separate schemas/databases.
- Save API service works directly with metadata store and object storage; cs-auth works only with metadata store and has no access to object storage.
- Save calls cs-auth in several cases:
  - **Proxying part of the administrative API** — Save forwards requests without processing them itself.
  - **Checking credentials** that are not present in the local pod cache — Basic Auth, API key, NuGet API key, NPM token, Docker token.
  - **Sending audit events** — one event log for all services.
- All internal endpoints are additionally protected by a shared secret.
- On the hot path, Save validates Bearer JWT **locally** using a cached JWKS public RSA key without a network call to cs-auth.

#Data Model

#Main Entities

#Project

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

#Repository

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

#Artifact

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

#User

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

```
- LastLoginAt
- Created/Updated timestamps
```

#Role / Robot account

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

#APIKey

```
- ID
- UserID            (FK to user with IsService=true)
- KeyPrefix         (first 12 hex characters of the plaintext
key, used for O(1) lookup)
- KeyHash           (bcrypt(plaintext))
- Description
- ExpiresAt         (NULL = no expiration)
- LastUsedAt
- CreatedAt
```

The full plaintext key is returned to the client only once, when it is created.

#Cleanup Policy

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

#Installation Options

CodeScoring.Save supports several installation profiles. Requirements, profile differences, and deployment steps are described in Installation Options.

#Security

#Authentication and Authorization

- JWT tokens for API, issued by Auth/RBAC service
- Basic Auth / API keys for package managers and service accounts
- Role-based access control (RBAC) with three scopes (global / project / repository), managed by the dedicated Auth/RBAC service
- Inter-service token validation through the JWKS endpoint

Client-side authentication methods are described in detail in Authentication.

#Data Encryption

Save and cs-auth services do not perform application-level encryption; encryption is provided by the underlying components:

- **Encryption at rest** is provided by the selected object storage (S3 server-side encryption, MinIO encryption-at-rest) and PostgreSQL (through TDE extensions or volume-level encryption).
- **Encryption in transit** is provided by TLS on ingress / load balancer and in inter-service connections.
- **User passwords and robot account API keys** are stored as bcrypt hashes in cs-auth.

#Audit

- Complete logging of all operations into a unified event log used by both services.
- Export to machine-readable format (JSON) and HTML. A single export can contain up to 10,000 records.
- Supported filters for list/export: `from`, `to`, `username`, `action`, `resource_type`, `q`, `user_id`. In the web UI, search is available only by username, resource type, and action.

#Performance

#Optimizations

- **Caching:**
  - In-process TTL cache `credential -> PermissionSet` in each Save API service pod, removing the round trip to cs-auth for repeated requests with the same credentials.
  - Local Bearer JWT validation through the JWKS public key, without a network call to cs-auth on the hot path. JWKS is refreshed periodically in the background.
- **Connection pooling:**
  - Database connection pool.

    - Reused HTTP clients for upstream repositories.

- **Parallel processing:**
    - Concurrent uploads / downloads through goroutines.
    - Batch (bulk) operations API.

- **Streaming large files:**
    - Artifacts are transferred through `MultiWriter` without buffering the full file in memory.
    - Atomic disk writes through temp-file rename to avoid corruption during concurrent writes.

#Target Metrics

For a basic installation:

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

#Scaling

#Horizontal Scaling

- Save API service: stateless, scaled by adding replicas.
- Auth/RBAC service: stateless, scaled by adding replicas. **Required condition:** all replicas must share the same RSA key for signing JWT tokens (`AUTH_JWT_RSA_KEY_PATH`).
- PostgreSQL: vertical scaling + read replicas if needed.
- Object storage: scaled horizontally by the provider (S3, MinIO in a cluster).

#Vertical Scaling

- Increasing pod resources.
- Increasing PostgreSQL capacity.
- Expanding object storage.

#Monitoring and Observability

#Metrics (Prometheus)

Save exports metrics in OpenTelemetry format with Prometheus exporter.

HTTP metrics:

- `http_requests_total{method,path,status}` — HTTP request counter.

- `http_request_duration_seconds{method,path,status}` — latency histogram.
- `http_requests_active` — active request gauge.

Operation metrics:

- `storage_operations_total{operation,backend,success}` — storage operations.
- `artifact_uploads_total{artifact_type,repository}` — artifact uploads.
- `artifact_downloads_total{artifact_type,repository}` — artifact downloads.
- `proxy_requests_total{artifact_type,repository,cache_hit,success}` — proxy requests. Cache hit rate is calculated as `rate(proxy_requests_total{cache_hit="true"}[5m]) / rate(proxy_requests_total[5m])` .
- `cleanup_operations_total{policy_type,repository,artifacts_deleted,success}` — cleanup operations.

#Logging

- Structured logs in JSON format (or text, depending on configuration).
- Levels: `debug`, `info`, `warn`, `error`.
- Output: stdout, stderr, or file.
- Compatible with any log aggregator that can process JSON (ELK, Loki, Vector).

#Availability and Resilience

CodeScoring.Save does not implement automatic failover, point-in-time recovery, or built-in backup on its own. These capabilities are provided by the standard tools of the infrastructure where the service is deployed:

- **Multiple Save and cs-auth replicas** in Kubernetes behind a load balancer provide availability if one pod fails.
- **PostgreSQL replication** and **database backup** are configured through the PostgreSQL operator or managed service.
- **Object storage snapshots and replication** are handled by the S3-compatible storage provider.
- **Multi-node Kubernetes cluster** provides resilience to node failure.

Save does not lose data on restart because all state is external: metadata is stored in the database and artifacts are stored in object storage. Stateless Save architecture allows scaling and pod recreation without data loss.

#CodeScoring.Save Installation Requirements

CodeScoring.Save supports two installation options: a SQLite-based option in k3s and a PostgreSQL-based option in Kubernetes. Both options are installed through the save Helm chart.

#General Requirements

- Linux-based OS (Ubuntu 20.04+, CentOS 8+, RHEL 8+)
- Kubernetes-compatible cluster: k3s for the SQLite option, full Kubernetes cluster for the PostgreSQL option
- Helm 3.x
- Access to the container registry with CodeScoring.Save images

#Hosted Installation

Minimum requirements:

- 2 CPU cores
- 4 GB RAM
- 50 GB disk space (SSD recommended)

Recommended requirements:

- 4 CPU cores
- 8 GB RAM
- 100 GB disk space (SSD)

Software:

- k3s v1.25+
- SQLite 3.x (included in the distribution)

#PostgreSQL Installation

Kubernetes cluster:

- 3+ worker nodes
- 4 CPU cores per node
- 8 GB RAM per node
- 50 GB disk per node

PostgreSQL:

- 4 CPU cores
- 8 GB RAM
- 100 GB storage (SSD recommended)
- PostgreSQL 14+

Object Storage:

- Built-in MinIO from the `minio` chart dependency or external S3-compatible storage: MinIO, AWS S3, Yandex Object Storage
- 500 GB+ storage, depending on artifact volume

#CodeScoring.Save Installation Options

CodeScoring.Save supports two installation options: a SQLite-based option in k3s and a PostgreSQL-based option in Kubernetes. Both options are installed through the save Helm chart, but they target different infrastructure and operation scenarios.

The Helm repository address and Docker Registry with CodeScoring.Save images are provided by the vendor. Installation examples use the placeholders <HELM_REPOSITORY_URL> and <REGISTRY_HOST>.

#Available Options

Option	When to use
Hosted installation	Testing, demonstrations, small teams, and quick start without separate PostgreSQL and S3 infrastructure
PostgreSQL installation	Production use, scaling, and operation with external or built-in PostgreSQL and S3-compatible storage

Before choosing an option, check installation requirements and prepare a shared `values.yaml` file. The main parameters are set in these sections:

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

The `image` section defines the shared registry, tag, and pull secret for backend, frontend, and auth images. Service environment variables are defined in `envs`.

To pass values from `envs` to Pods, enable the `envs` ConfigMap and attach it to deployments through `deploymentsGeneral.envConfigmaps`.

#CodeScoring.Save Hosted Installation

The hosted option is deployed in k3s and uses the SQLite profile. This option is suitable for testing, demonstrations, small teams, and quick start without separate PostgreSQL and S3 infrastructure.

#Installing k3s

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

## Check status
sudo systemctl status k3s

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

## Check cluster
kubectl get nodes
```

#Preparing Helm and Namespace

The Helm repository address and Docker Registry with CodeScoring.Save images are provided by the vendor. The examples below use the placeholders <HELM_REPOSITORY_URL> and <REGISTRY_HOST>.

```
## Add CodeScoring Helm repository
helm repo add codescoring <HELM_REPOSITORY_URL>
helm repo update

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

Create an imagePullSecret for access to the private registry. The secret name must match the `image.pullSecrets` value in `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 -
```

#SQLite Configuration

For the SQLite option, disable built-in PostgreSQL and S3:

```
postgres:
  enabled: false

minio:
  enabled: false
```

Set the registry, tag, and pull secret for Save images in `image`:

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

Environment variables are defined in `envs`. To pass them to Pods, enable the `envs` ConfigMap and attach it to 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"
```

#Installation

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

#Installation Check

```
## Check pod status
kubectl get pods -n codescoring-save

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

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

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

## Check services
kubectl get svc -n codescoring-save

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

#Initial Setup

The administrator password is set in `AUTH_ADMIN_PASSWORD`:

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

After installation, open the web interface at the address configured in ingress.

#Next Steps

After installation:

1. Create the first repository
2. Configure cleanup policies
3. Create users and assign roles

#CodeScoring.Save PostgreSQL Installation

PostgreSQL installation is intended for production use, larger teams, availability requirements, scaling, and performance requirements.

#Preparing Infrastructure

#PostgreSQL

By default, PostgreSQL is deployed as a Helm chart dependency. To use it, enable `postgres.enabled=true` in `values.yaml` and set connection parameters:

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

When built-in PostgreSQL is used, backend and auth must connect to host `postgres`.

For external PostgreSQL, disable the built-in dependency:

```
postgres:
  enabled: false
```

Then set the external PostgreSQL host in `envs.DATABASE_HOST`.

Prepare the database and user in external PostgreSQL:

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

#Object Storage

By default, MinIO is deployed as a Helm chart dependency. To use it, enable `minio.enabled=true` in `values.yaml`. The service will be available as `save-minio`.

```
minio:
  enabled: true
  fullnameOverride: save-minio
  auth:
    rootUser: <ACCESS_KEY>
```

```
    rootPassword: <SECRET_KEY>
  provisioning:
    enabled: true
    buckets:
      - save
```

For a stateful installation, enable persistence for PostgreSQL and MinIO:

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

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

The common `pvcs` block must not be used for PostgreSQL and MinIO in the provided chart. If `postgres.persistence.enabled` or `minio.persistence.enabled` is disabled, the corresponding service data is not preserved after Pod recreation.

For built-in MinIO, use endpoint `http://save-minio:9000` in `envs`. The `minio.provisioning` section creates the bucket during chart installation, so `S3_BUCKET` must match one of the values from `minio.provisioning.buckets`.

For external S3-compatible storage, disable built-in MinIO:

```
minio:
  enabled: false
```

Then set `S3_ENDPOINT`, `S3_BUCKET`, `S3_ACCESS_KEY`, and `S3_SECRET_KEY` in `envs`.

#CodeScoring.Save Configuration

Use the base `values.yaml` from the chart and change the required parameters.

For built-in PostgreSQL and MinIO, use these values:

```
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
```

The AUTH_INTERNAL_SECRET value is used for internal interaction between Save services. The minio.provisioning section creates the bucket for built-in MinIO, and S3_BUCKET must match one of the values from minio.provisioning.buckets.

#Installation

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

#Installation Check

```
## Check pod status
kubectl get pods -n codescoring-save -w

## Check all resources
kubectl get all -n codescoring-save

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

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

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

## Check readiness
kubectl get pods -n codescoring-save -o wide
```

#Next Steps

After installation:

1. Create the first repository
2. Configure cleanup policies
3. Create users and assign roles

#System Update

Updates are performed through Helm. Before updating, save the `values.yaml` file in use, check the target chart version, and prepare a data backup for PostgreSQL installations.

#Hosted Installation

```
## Update Helm chart
helm repo update

## Upgrade to a new version
helm upgrade codescoring-save codescoring/save \
  --namespace codescoring-save \
  --values values.yaml \
  --version <NEW_VERSION>
```

#PostgreSQL Installation

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

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

## Check deployments
kubectl get deploy -n codescoring-save

## Check backend rollout
kubectl rollout status deployment/codescoring-save -n
codescoring-save

## Check auth rollout
kubectl rollout status deployment/codescoring-save-auth -n
codescoring-save

## Check frontend rollout
kubectl rollout status deployment/codescoring-save-save-front
-n codescoring-save
```

#Uninstalling the Installation

Namespace deletion

Deleting the Helm release stops CodeScoring.Save components. Deleting the namespace additionally removes all resources in that namespace, so save the required data and configuration before this operation.

```
## Delete Helm release
helm uninstall codescoring-save --namespace codescoring-save

## Delete namespace
kubectl delete namespace codescoring-save
```

#SSL/TLS Setup for CodeScoring.Save

SSL/TLS is configured at the ingress level. This page describes the base `values.yaml` structure and two common ways to attach a certificate: through cert-manager or through a pre-created Kubernetes secret.

Ingress and TLS parameters are configured in `values.yaml` in the `app.ingresses` section. The section format must match the `codescoring-generic` schema.

Note

The exact set of annotations depends on the ingress controller and certificate issuance method.

Example structure:

```
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
```

#Using cert-manager

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

## Create 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
```

#Using Self-Signed Certificates

```
## Generate certificate
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout tls.key -out tls.crt \
  -subj "/CN=save.example.com"

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

#CodeScoring.Save Troubleshooting

This section helps collect initial information about the installation state: pod status, service logs, database availability, and storage availability. The commands below do not fix issues automatically, but provide data for further analysis.

#Pod Issues

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

## Pod logs
kubectl logs <POD_NAME> -n codescoring-save --previous

## Enter pod for diagnostics
kubectl exec -it <POD_NAME> -n codescoring-save -- /bin/sh
```

#Database Issues

```
## Check PostgreSQL connection from the postgres pod
kubectl exec -it -n codescoring-save postgres-0 -- \
  psql -U save_user -d save_db

## Check tables inside psql
\dt
\q
```

#Storage Issues

```
## Check MinIO access
kubectl port-forward -n codescoring-save svc/save-minio
9001:9001

## Open in browser:
## http://localhost:9001
```

#General

This section covers core platform workflows:

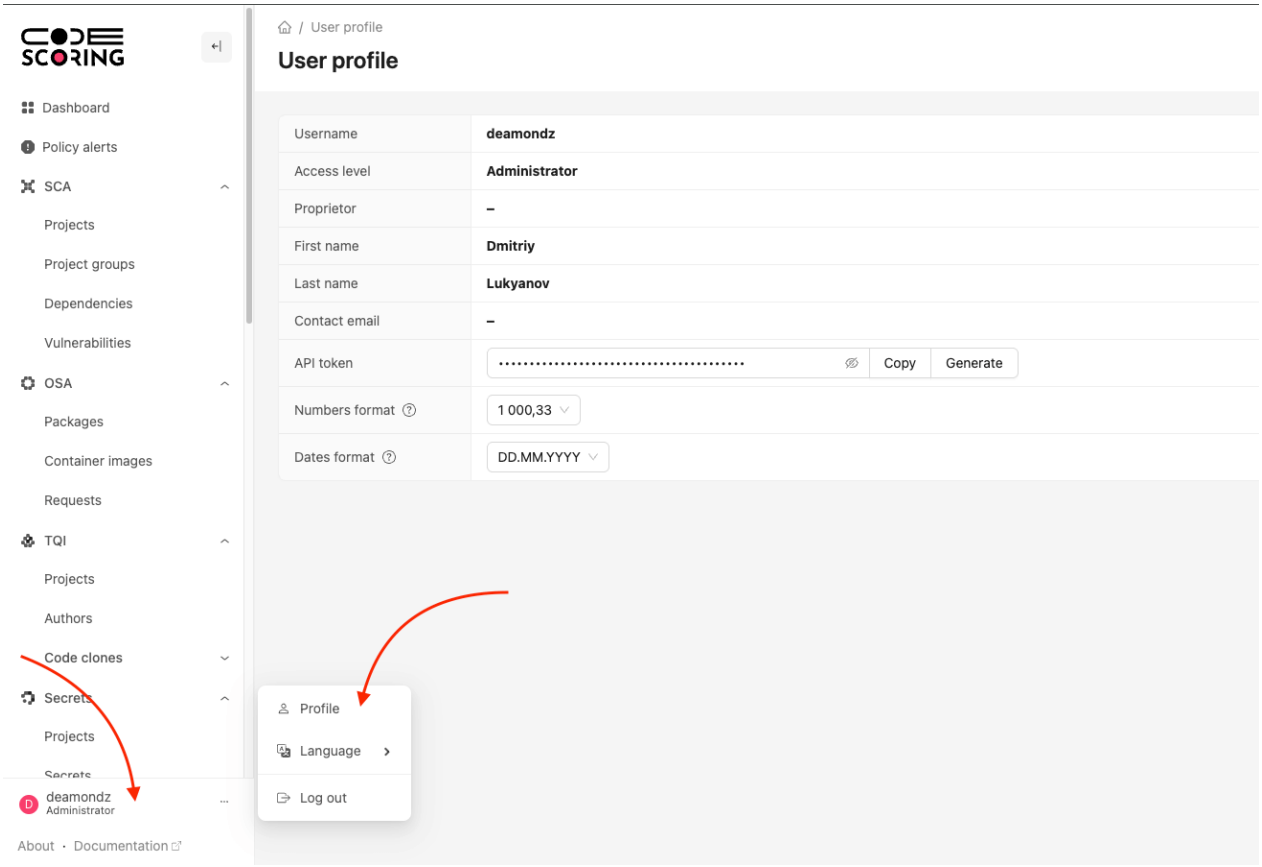
- user profile configuration;
- creation and setup of security policies;
- VCS integration and project management;
- advanced setup (proprietors, notifications, audit log, metrics, webhooks, API).

Use the left navigation to open the required subsection.

#Configuring user profile

The **Profile** section allows users to view and update their personal information and customize the interface appearance.

The profile can be accessed by clicking on the username in the bottom-left corner of the interface.



The page displays the following parameters:

- **Username** — a unique login used to access the system. Cannot be edited;
- **Access level** — defines the user's permissions. Available values: `User`, `Auditor`, or `Administrator`;
- **Proprietor** — the organizational unit the user belongs to (if used);
- **First name** — shown in the interface and reports;
- **Last name** — shown in the interface and reports;
- **Email** — contact address displayed in user lists;
- **API token** — used for integration with external tools. Can be copied or regenerated (the previous one becomes invalid);
- **Number format** — numeric display format in the system;
- **Date format** — date display format.

Important: number and date format affect display only. They do not influence data input or export.

#Editing the profile

Click the **Edit** button to open the profile editing form. The following fields can be updated:

- First name
- Last name
- Email

Changes are saved immediately after clicking **Save**.

#Managing policies

#General description

Policies on the CodeScoring platform are a mechanism for tracking and blocking open source components during software development. They can be related to security checks, license compatibility, or other criteria for including third-party components in development.

Policies can be created for:

- the entire organization;
- proprietor;
- groups;
- project;
- development environment;
- repository;
- component type.

The policy mechanism takes into account the specified stage of software development: from the download of third-party components into the organization's perimeter to tracking builds and writing new code.

Policies are configured based on conditions combined by logical expressions **AND/OR**. In addition to the standard security policy settings based on vulnerability severity level, conditions can be configured based on component metadata: release date, license, author, and others. A total of **40+ categorized conditions** are supported. The checks also include a built-in policy for checking for license compatibility.

When a policy is triggered, corresponding **alerts** are created in CodeScoring. Alerts can be temporarily or permanently ignored, or they can be downloaded as a report.

Policies can be **blocking**: when such a policy is triggered, the components used are blocked in the artifact storage (proxy repositories), or the CI build is stopped until the detected defect is fixed.

Additionally, when a policy is triggered, a notification can be sent to the responsible specialists in the task management system or an email describing the problem.

#Policy stages

Policy stages are configured by the user when editing project parameters in the **Settings → Projects** section in the **Policy stage** field or specified via the `--stage` parameter when running the console agent Johnny. The stages have the following meanings:

- `dev` – development stage;
- `stage` – intermediate (pre-production) stage;
- `test` – testing stage;
- `prod` – production circuit.

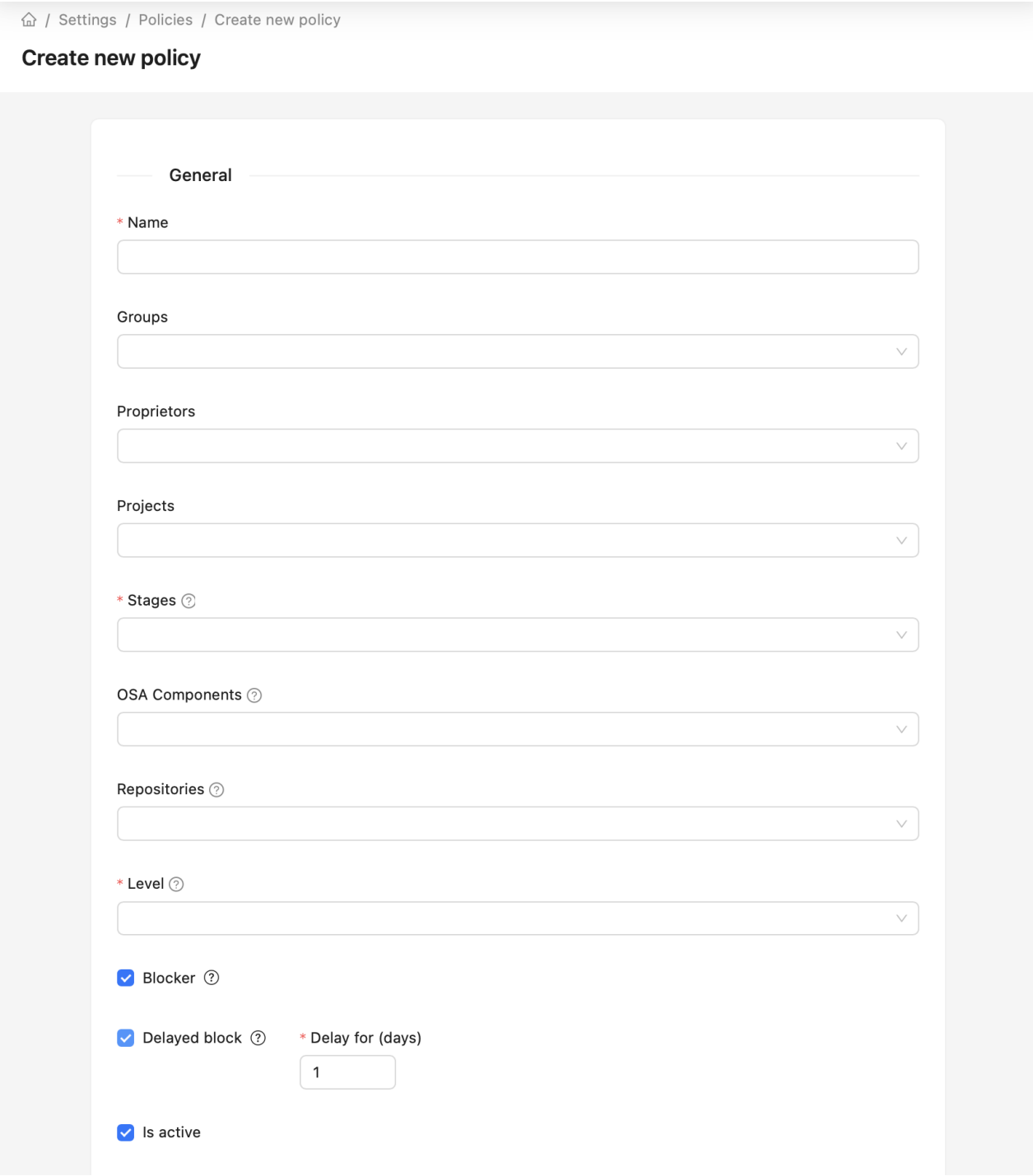
When creating or editing a policy in the **Settings → Policies** section, you must specify the stages to which it will be applied.

In addition, there are special stage values used by default for certain tasks:

- `proxy` – for the plugin in the OSA module;
- `source` – for VCS project analysis;
- `build` – for the Johnny console agent.

#Creating a policy

Policies are created in the `Settings -> Policies` section. You can go to the policy creation form by clicking the **Create** button.



The policy creation form specifies the policy context using the following parameters:

- **Name**;
- **Groups** — project groups to which the policy applies. If the parameter is empty, the policy applies to the entire organization;
- **Proprietors** — organization departments to which the policy applies. If the parameter is empty, the policy applies to the entire organization;
- **Projects** — projects to which the policy applies;
- **Stages** — development cycle stages to which the policy applies;
- **OSA Components** — component type in the repository manager with the OSA plugin to which the policy applies (**Packages** or **Container Images**);

- **Repositories** — list of repositories with the OSA plugin to which the policy applies;
- **Level** — policy criticality level (does not affect actions in the system);
- **Blocker** — blocking CI build or download of a component from a proxy repository;
- **Delayed block** — delay (days) from the first policy trigger to blocking;
- **Is active** — policy status;
- **Description** — policy description;
- **Conditions** — list of conditions in the policy.

Next, the conditions for triggering the policy are configured; the following parameters are supported:

- **PURL** — package URL, component identifier;
- **Dependency name** — name of the used dependency;
- **Dependency version** — specific version of the component detected in the project;
- **Dependency author** — name or organization listed as the author of the dependency;
- **Dependency release date** — date when the dependency version was first published;
- **Dependency age (days)** — number of days since the dependency was published;
- **Dependency vulnerability count** — total number of known vulnerabilities in the component;
- **Dependency is dangerous** — the dependency is considered dangerous if it includes any vulnerabilities with:
  - prefix `MAL-` (malicious package feed from OSV);
  - one of the following CWEs:
    - CWE-506: Embedded Malicious Code;
    - CWE-507: Trojan;
    - CWE-509: Non-Propagating Malicious Code;
    - CWE-509: Propagating Malicious Code (virus or worm);
    - CWE-510: Hidden Functionality;
    - CWE-511: Logic or Time Bomb;
    - CWE-512: Spyware;
    - CWE-912: Backdoor Function;
  - or one of the following Impacts values from Kaspersky OSS Threats Data Feed:
    - Malicious software;
    - Other impact.
- **Dependency is protestware** — the dependency is considered protestware if it includes any vulnerabilities from the proprietary protestware feed;
- **Dependency is ransomware** — the dependency is considered ransomware if it includes any vulnerabilities used in ransomware according to CISA KEV;
- **Dependency is revoked** - dependency is revoked from source;
- **Dependency is a descendant of** — searches across all levels for lower-level transitive dependencies associated with the specified parent component (e.g. in `a<-b<-c<-d`, c and d are descendants of b);

- **Dependency Transitivity depth** - controls the depth of the dependency search, where 1 is a direct dependency, 2 or more are transitive. Only natural numbers are allowed;
- **Technology** — programming language or ecosystem;
- **License** — SPDX license identifier;
- **License category** — classification of the license (e.g. permissive, copyleft);
- **Vulnerability ID** — vulnerability identifier;
- **CVSS2 Score** — numerical threat score according to the CVSS 2 standard;
- **CVSS2 Severity** — threat level according to the CVSS 2 standard;
- **CVSS2 Access Vector (AV)** — vulnerability exploitation path (physical or network);
- **CVSS2 Access Complexity (AC)** — vulnerability exploitation complexity;
- **CVSS2 Authentication (Au)** — authentication requirements for vulnerability exploitation;
- **CVSS2 Availability Impact (A)** — degree of data availability loss;
- **CVSS2 Confidentiality Impact (C)** — degree of data confidentiality loss;
- **CVSS2 Integrity Impact (I)** — degree of data integrity loss;
- **CVSS3 Score** — numerical CVSS 3 threat assessment;
- **CVSS3 Severity** — threat level according to the CVSS 3 standard;
- **CVSS3 Attack Vector (AV)** — attack vector;
- **CVSS3 Attack Complexity (AC)** — attack complexity;
- **CVSS3 Priviliges Required (PR)** — required access level to exploit the vulnerability;
- **CVSS3 User Interaction (UI)** — presence of user interaction;
- **CVSS3 Scope (S)** — component security scope;
- **CVSS3 Confidentiality (C)** — degree of loss of data confidentiality;
- **CVSS3 Integrity (I)** — degree of loss of data integrity;
- **CVSS3 Availability (A)** — degree of loss of data availability;
- **CVSS4 Score** — numerical CVSS 4 threat assessment;
- **CVSS4 Severity** — threat level according to the CVSS 4 standard;
- **CVSS4 Attack Vector (AV)** — attack vector;
- **CVSS4 Attack Complexity (AC)** — attack complexity;
- **CVSS4 Attack Requirements (AT)** — whether any attack prerequisites required;
- **CVSS4 Privileges Required (PR)** — required access level to exploit the vulnerabilit;
- **CVSS4 User Interaction (UI)** — presence of user interaction;
- **CVSS4 Confidentiality Impact to the Vulnerable System (VC)** — confidentiality impact to the vulnerable system;
- **CVSS4 Confidentiality Impact to the Subsequent System (SC)** — confidentiality impact to the subsequent system;
- **CVSS4 Integrity Impact to the Vulnerable System (VI)** — integrity impact to the vulnerable system;
- **CVSS4 Integrity Impact to the Subsequent System (SI)** — integrity impact to the subsequent system;
- **CVSS4 Availability Impact to the Vulnerable System (VA)** — availability impact to the vulnerable system;

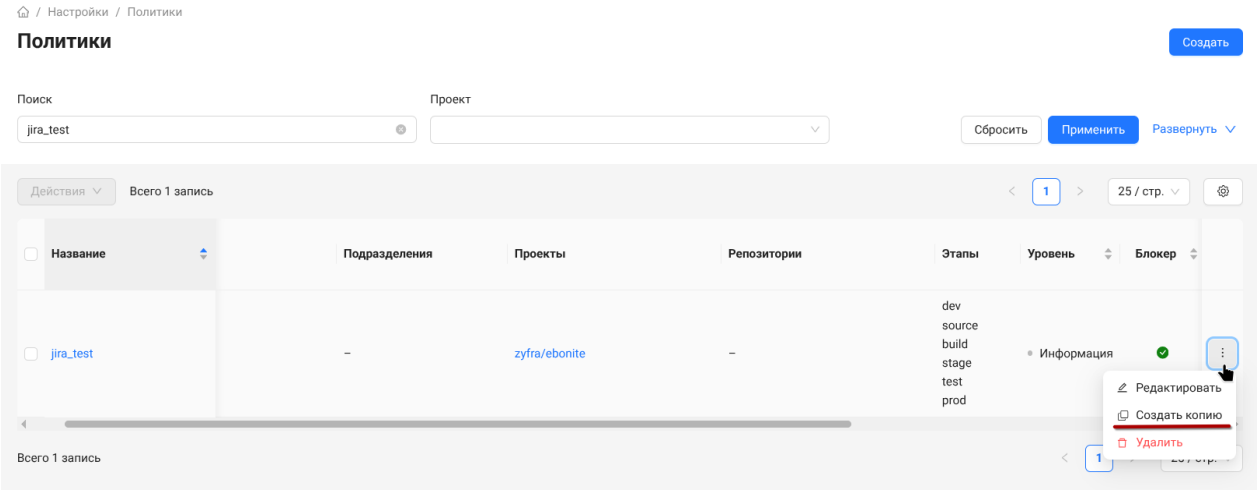
- **CVSS4 Availability Impact to the Subsequent System (SA)** — availability impact to the subsequent system;
- **Vulnerability publication date** — initial disclosure date of the vulnerability;
- **Vulnerability update date** — most recent update of the vulnerability record;
- **Impacts (Kaspersky)** — possible Impacts values from Kaspersky OSS Threats Data Feed;
- **Vulnerability has exploit** — indicates the presence of a public exploit in databases like NVD, GHSA, FSTEC, etc.;
- **Vulnerability has fixed version** — indicates the availability of a safe version that mitigates the vulnerability;
- **Vulnerabilities are reachable** – the vulnerable method of the component is used in the source code. More details about reachability analysis can be found in the Johnny agent documentation.
- **Vulnerability age (days)** — number of days since the vulnerability was disclosed;
- **Env** — development environment (e.g., prod, dev, source);
- **Match type** — dependency detection method (by manifest, project content, or as a result of dependency resolution);
- **Relation** — dependency relationship in the project (direct or transitive);
- **CWE** — vulnerability type identifier according to the Common Weakness Enumeration standard
- **Protest package category** — protestware category;
- **SSVC2 Exploitation** — the present state of exploitation of the vulnerability;
- **SSVC2 Automatable** — can the vulnerability exploitation be automated;
- **SSVC2 Technical impact** — technical impact to the vulnerable system;
- **EPSS percentage** — estimated probability that the vulnerability will be exploited within the next 30 days;
- **EPSS percentile** — the share of vulnerabilities with the same or lower percentage.

There is an option to specify list-based values for the following parameters:

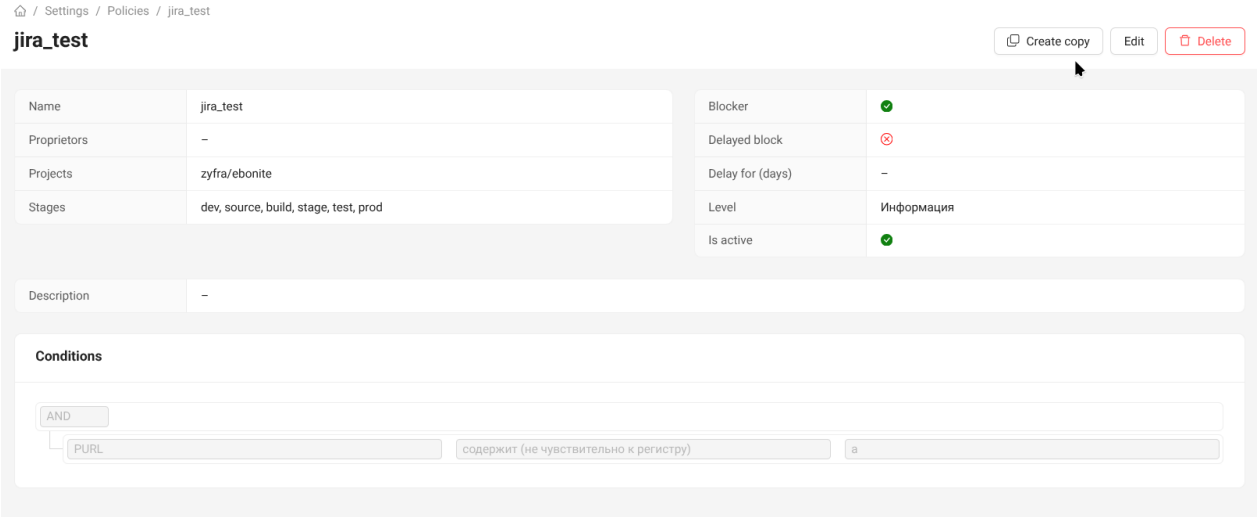
- **PURL list**
- **Dependency name list**
- **Dependency author email list**
- **Vulnerability ID (CVE) list**
- **Environment list**
- **Technology list**
- **License list**
- **License category list**
- **CWE list**
- **CVSS2 severity list**
- **CVSS3 severity list**
- **CVSS4 severity list**
- **Protest package category list**

#Creating a copy of the policy

If you need to duplicate an existing policy, you can use the context menu item **Create a copy** or click the button.



Or click on the policy form **Create a copy**.



If a copy of the policy is created, a new policy is created with the same description, conditions, and related actions.

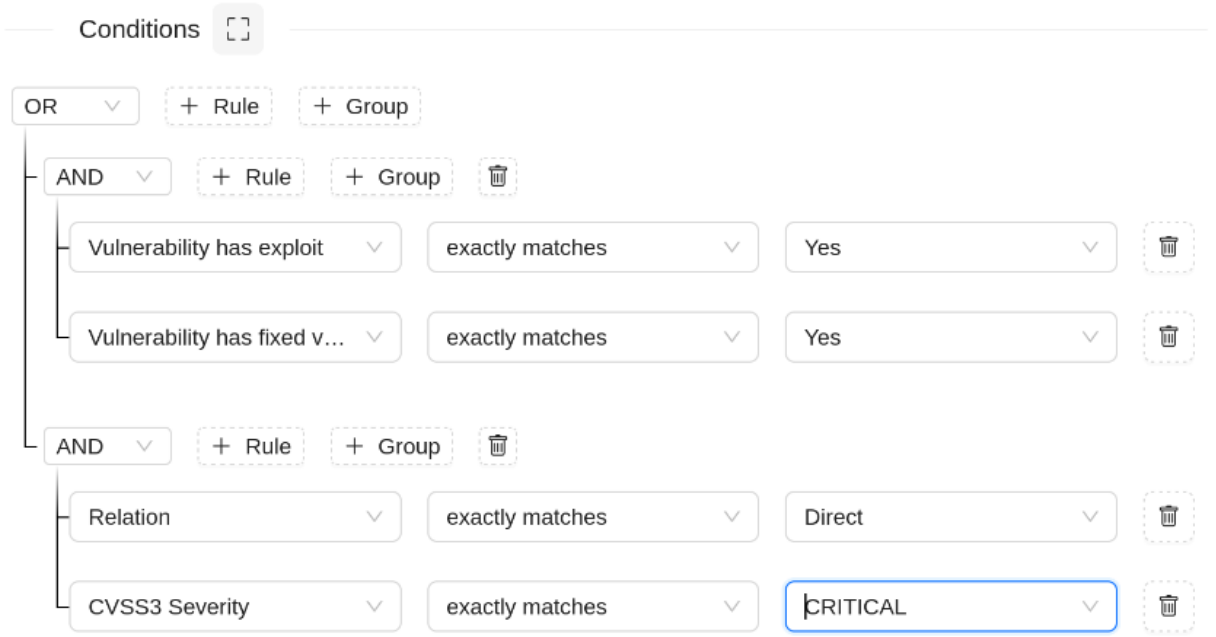
#Policy example

Policy conditions can be combined into groups using logical expressions **AND/OR**. Groups have no restrictions on the nesting level and the number of conditions.

For example, you can set policy conditions for the following scenario - either the dependency contains a vulnerability with an exploit and a fixed version, or the dependency is direct and contains a critical vulnerability according to the CVSS 3 standard.

To create such a policy, you need to add two groups united by the **OR** expression. This means that the policy will be triggered if any of the listed

groups of conditions are met. Within the group, conditions are set, united by the **AND** expression.



The policy becomes active immediately after creation by clicking the **Create** button. For the created policy, you can configure actions when it is triggered: email notification or creation of a task in Jira or Kaiten.

Important: policies are triggered during analysis, so it is important to create them before running the analysis.

Recommendation: if you leave the `Proprietors`, `Groups` and `Projects` fields empty, the policy will apply to all active projects in the system.

#Ignoring policies

Triggered policies can be temporarily or permanently ignored during analysis. The ignore condition will allow you to leave policies in the system without receiving alerts about their triggering, for example, if a vulnerability in a component is not applicable to a specific project.

#Creating ignores

Ignore conditions are created and configured in the `Settings -> Policy ignores` section.

To create an ignore condition for one or more policies, click the **Create** button and fill in the following fields:

- **Groups** - project groups to which the ignore is applied;
- **Projects** - projects to which the ignore is applied;
- **Container images** - images in registries to which the ignore is applied;

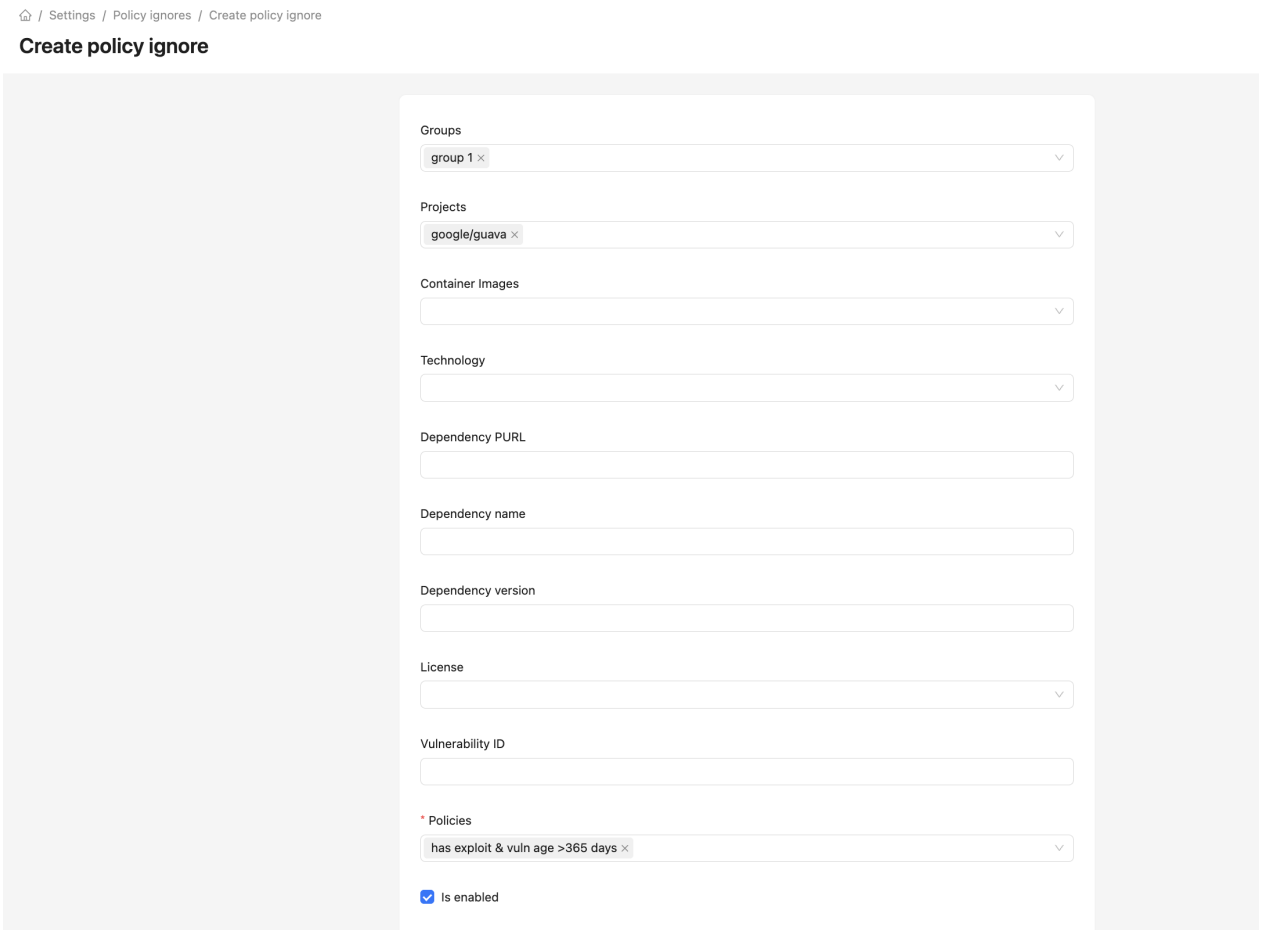
- **Technology** - programming language or ecosystem;
- **Dependency PURL** — package URL, component identifier;
- **Dependency name**;
- **Dependency version**;
- **License**;
- **Vulnerability ID** - vulnerability identifier;
- **Policies**;
- **Active** - ignore state;
- **Activation date** - date from which the ignore will start working;
- **End date** - date after which the ignore will stop working;
- **Note**.

Ignore by partial match

Alert ignore based on PURL uses partial matching semantics: if a specified PURL omits any components, the match is treated as positive.

For example: alert with dependency PURL `pkg:conda/setuptools@65.1.0?build=py310h2ec42d9_0&channel=conda-forge&subdir=osx-64&type=tar.bz2`
would be ignored by specified ignore PURL `pkg:conda/setuptools@65.1.0`.

However, the dependency PURL will not be treated as a match for `pkg:conda/setuptools@65.1.0?subdir=foo`, as the `subdir` qualifier differs.



#Ignore results

The policies that were subject to the ignore condition are displayed on the "Ignored" tab of the `Alerts` section.

Triggered active policies can also be quickly ignored from the "Active" tab using the **Ignore** button.

Access

The user can only view the ignores that are related to the projects available to him.

#Policy alerts

#Policy alerts section

Policy results are displayed in the `Policy alerts` section. The section has three tabs:

- **Active** – a list of alerts based on the results of the last analysis (project, build, or component in the proxy repository);
- **Ignored** – a list of ignored alerts;
- **Resolved** – a list of alerts that were resolved after the last analysis (the policy condition is no longer relevant).

The policy trigger is displayed in the **Matched criteria** field, including the specified conditions and the component data found. For example, the value `django@4.2.2 has CVE-2024-38875, CVSS3 Score 7.5 >= 7.00` implies that the component blocking policy with CVSS3 equal to or higher than 7.00 was triggered on the django component version 4.2.2 with a vulnerability score of 7.5.

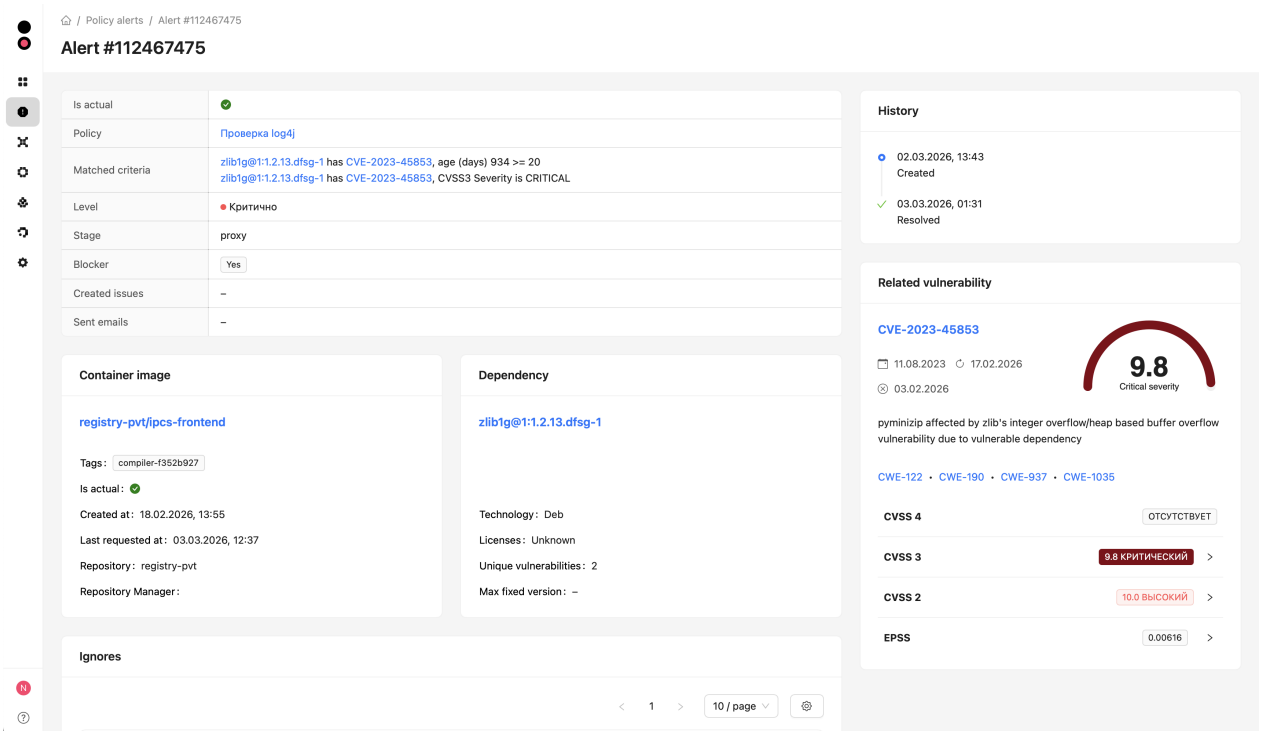
#Alert page

The alert page opens when you click an alert in the list. The header shows the key fields:

- **Is actual** – whether the alert is current;
- **Policy** – the policy that triggered the alert (link to the policy page);
- **Matched criteria** – matched conditions that caused the trigger;
- **Level** – policy severity level;
- **Stage** – policy stage;
- **Blocker** – blocking policy flag;
- **Created issues** – created tasks (for example, Jira);
- **Sent emails** – email notification recipents.

Related entities and details are displayed below, depending on the conditions that triggered the alert:

- **Project/Container image/Package/Dependency** – component details and source (repository, tags, licenses);
- **Related vulnerability** – short description, CVE, CVSS, and EPSS values;
- **History** – alert timeline (creation time, ignore time, resolve time).

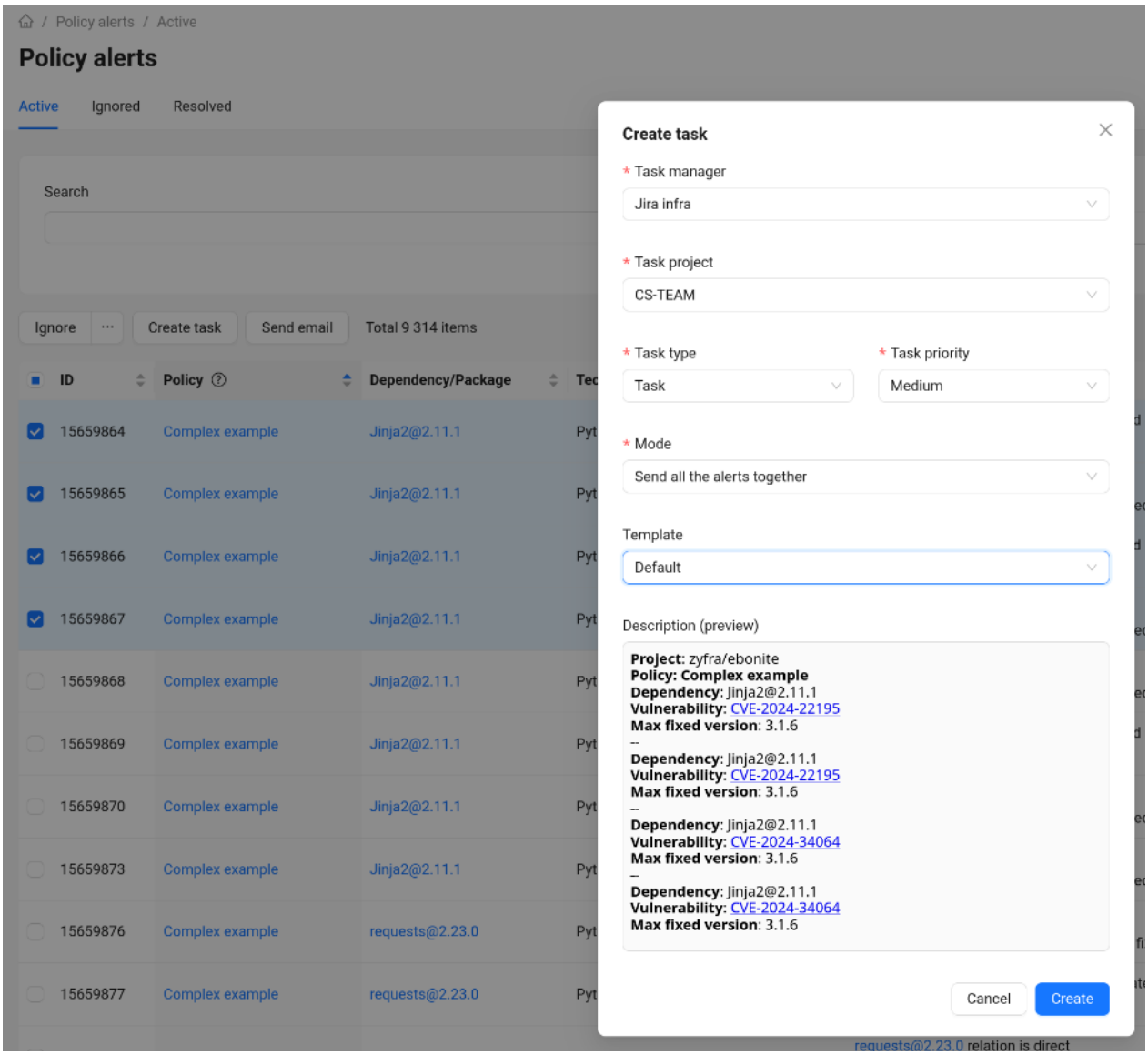


#Alert actions

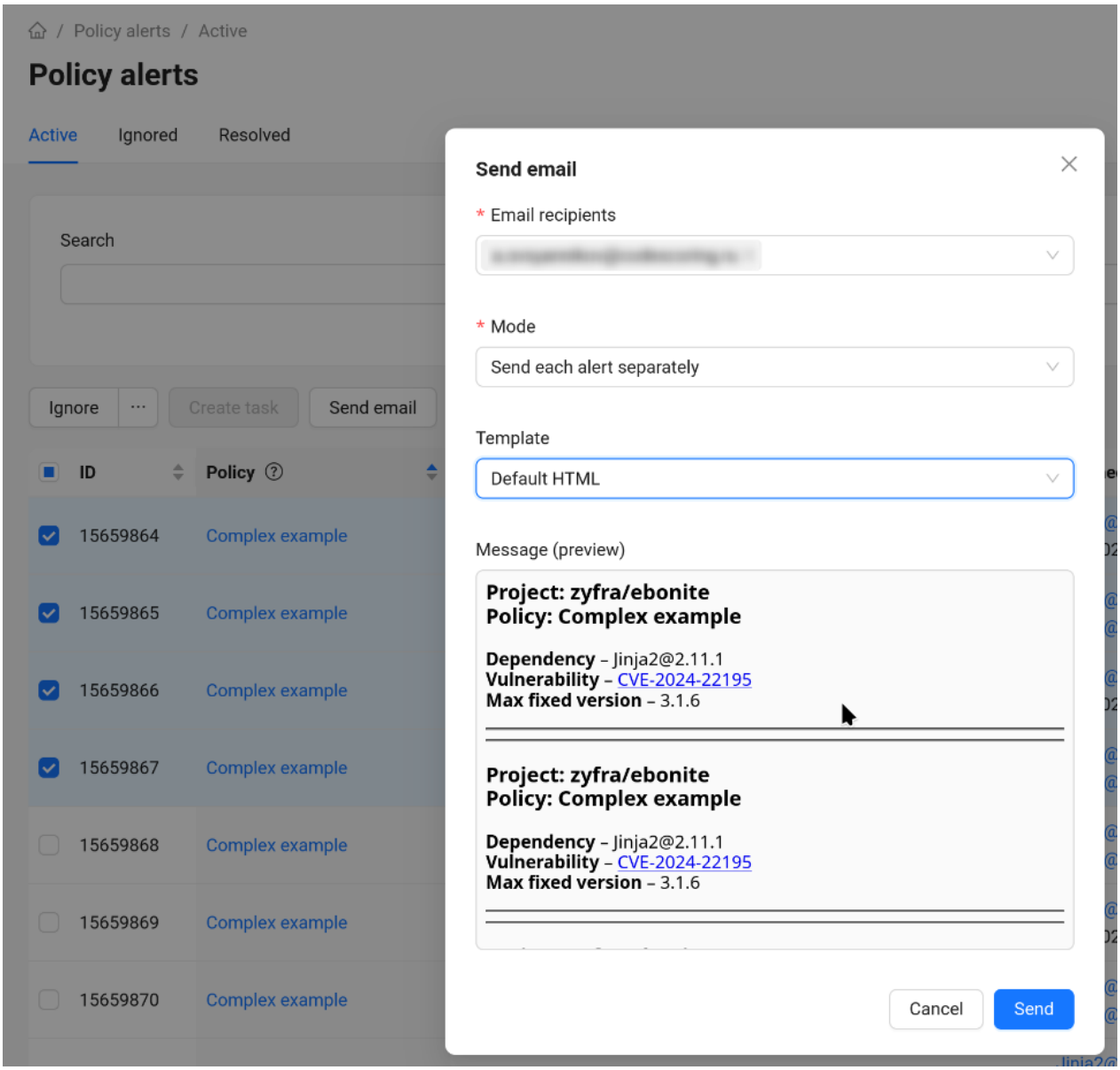
To create a task or send an email from the alerts list, select one or more alerts and click the corresponding button.

For example:

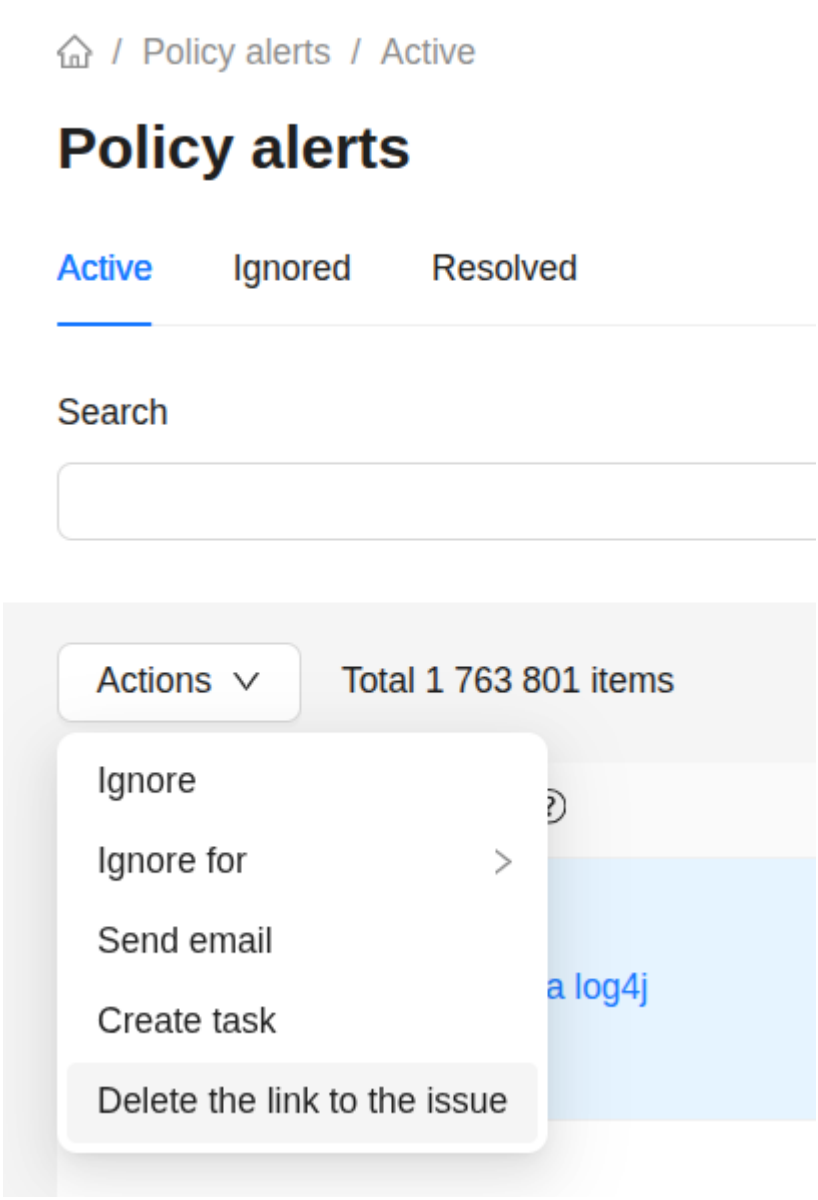
- creating tasks



- sending emails



Linked Jira issues can be unlinked using the bulk action `Delete the link to the issue`.



#Connecting to a VCS

To add projects (Git repositories) to the system for analysis, you must first create a connection to the version control system (VCS). CodeScoring supports the following platforms:

- GitLab
- GitHub
- BitBucket (Data Center and Server only)
- Azure DevOps Git
- Gitflic
- Other platforms using Git

Integration occurs through two possible mechanisms: Personal Access Token or SSH key. For the Personal Access Token mechanism, it is possible to connect several version control systems with the same address, but different tokens.

After connecting the version control system, it is impossible to change its type (`Instance type`) and address (`Instance URL`).

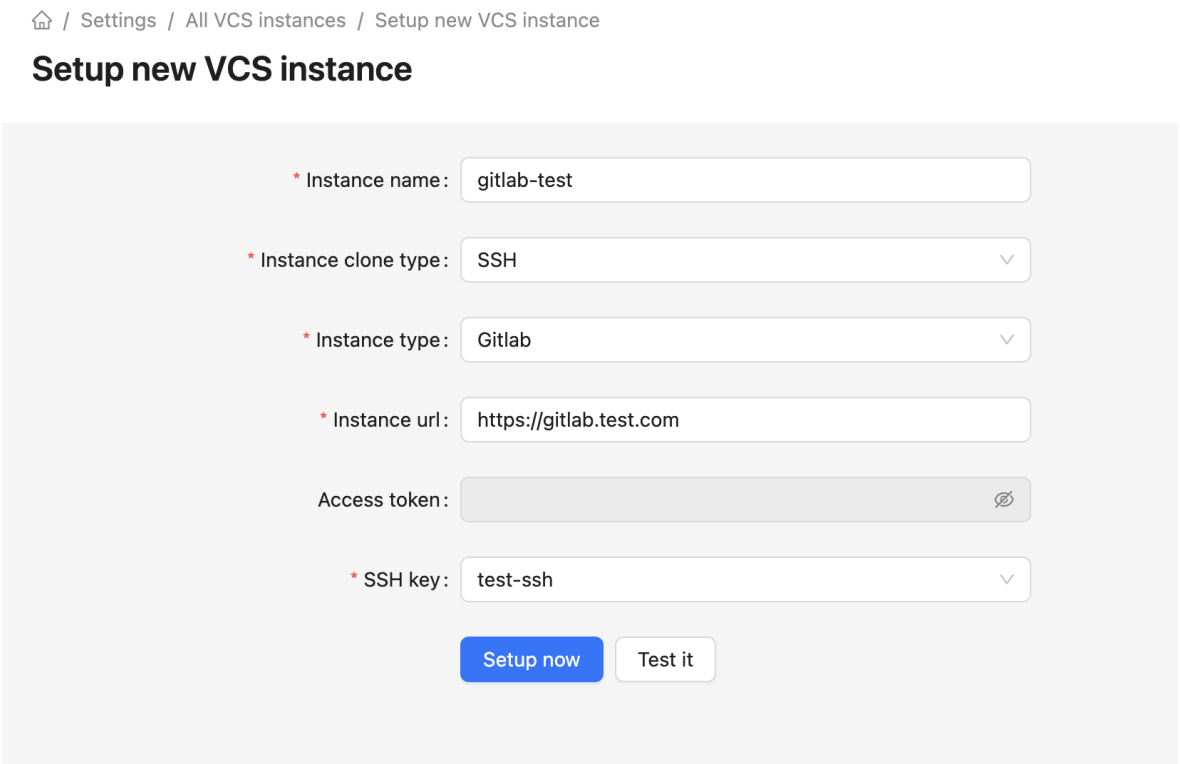
Note: if private NS servers are used to access the version control system, they must be specified in the platform settings. To obtain the appropriate templates, you must contact the vendor.

#Adding an SSH key

1. Copy an existing or generate a new private SSH key in the version control system according to the instructions for:

- Gitlab
- Github
- BitBucket
- Azure DevOps Git

1. In the CodeScoring interface, go to the `Settings -> SSH Keys` section
2. Click **Setup new** in the upper right corner.
3. Fill out the form. The private SSH key is inserted into the **Instance private key** field.
4. Go to the `Settings -> VCS` section.
5. Click **Setup new** in the upper right corner.
6. Fill out the form as shown in the screenshot. The SSH key is selected from the list in the **SSH key** field.

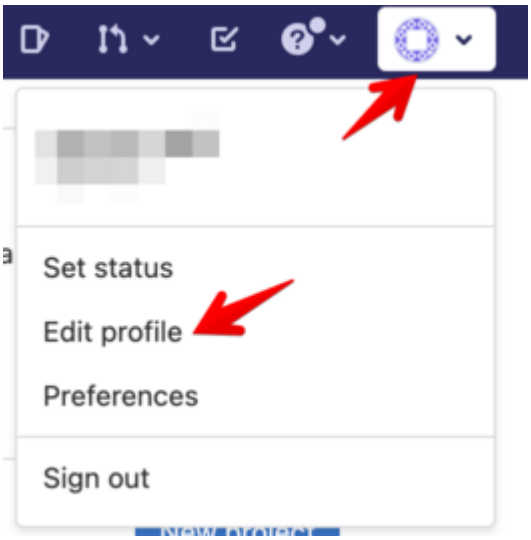


7. You can check the connection by clicking the **Test it** button. To create a connection, click on the **Setup now** button.

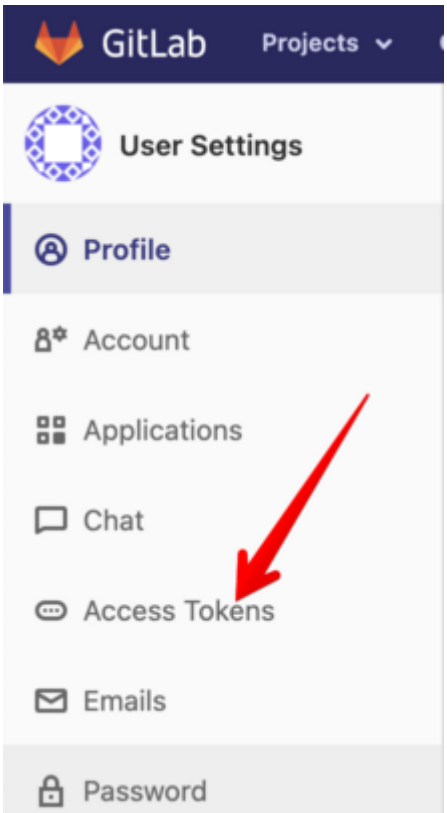
#Adding a token for GitLab

Original instructions for generating a token: https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html#create-a-personal-access-token

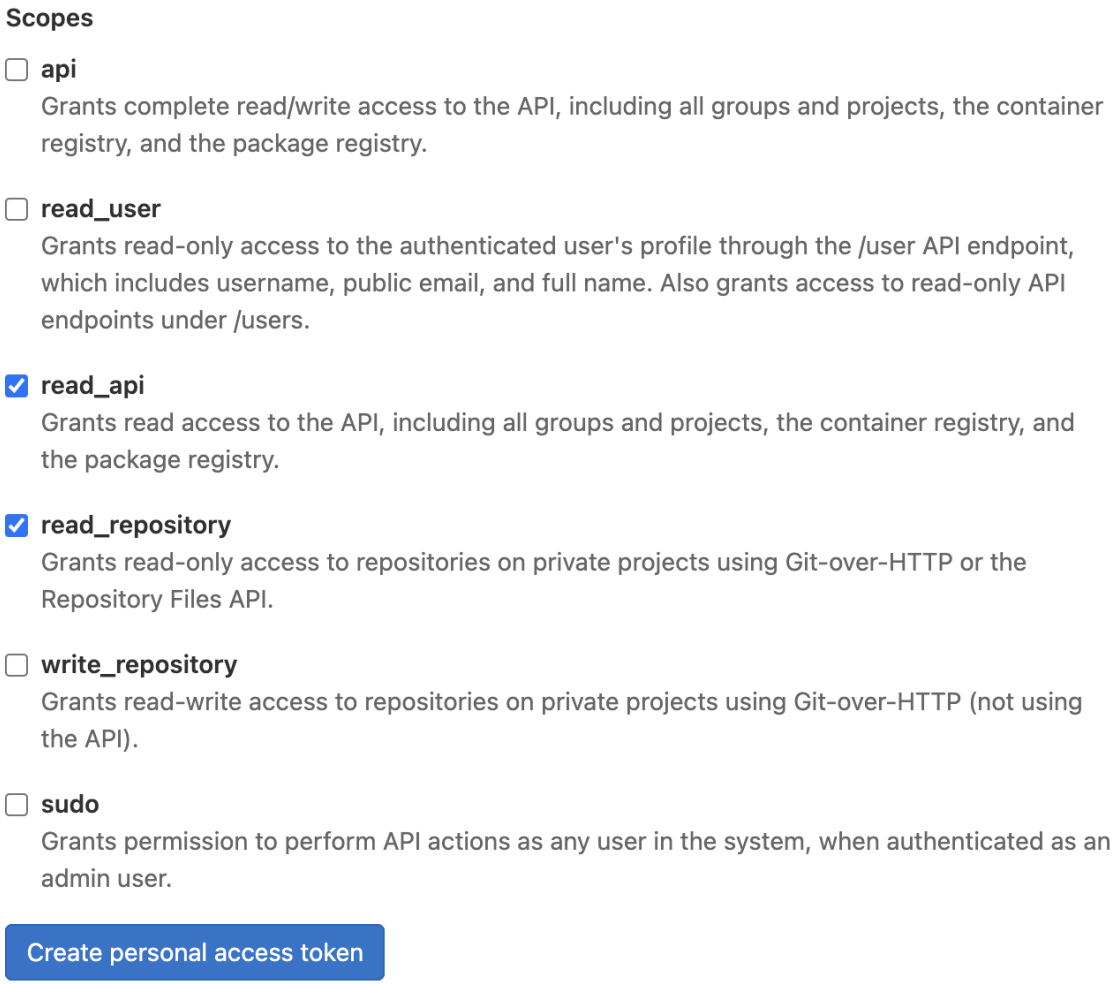
1. Log in to your GitLab account.
2. Through the user menu in the upper right corner, go to the **Edit profile** section.



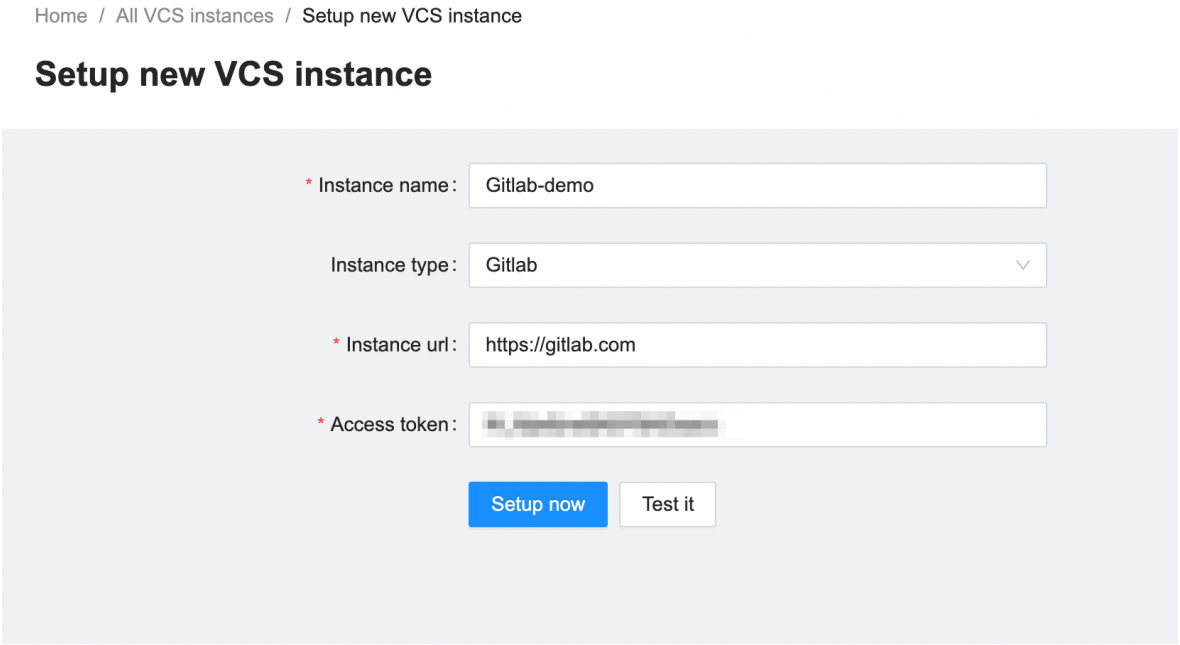
1. Next, in the left menu, select the **Access Tokens** section.



1. Give the token a name, for example, “*codescoring-demo*”, the date can be left blank
2. In the *scopes* section, select **read_api** and **read_repository**.



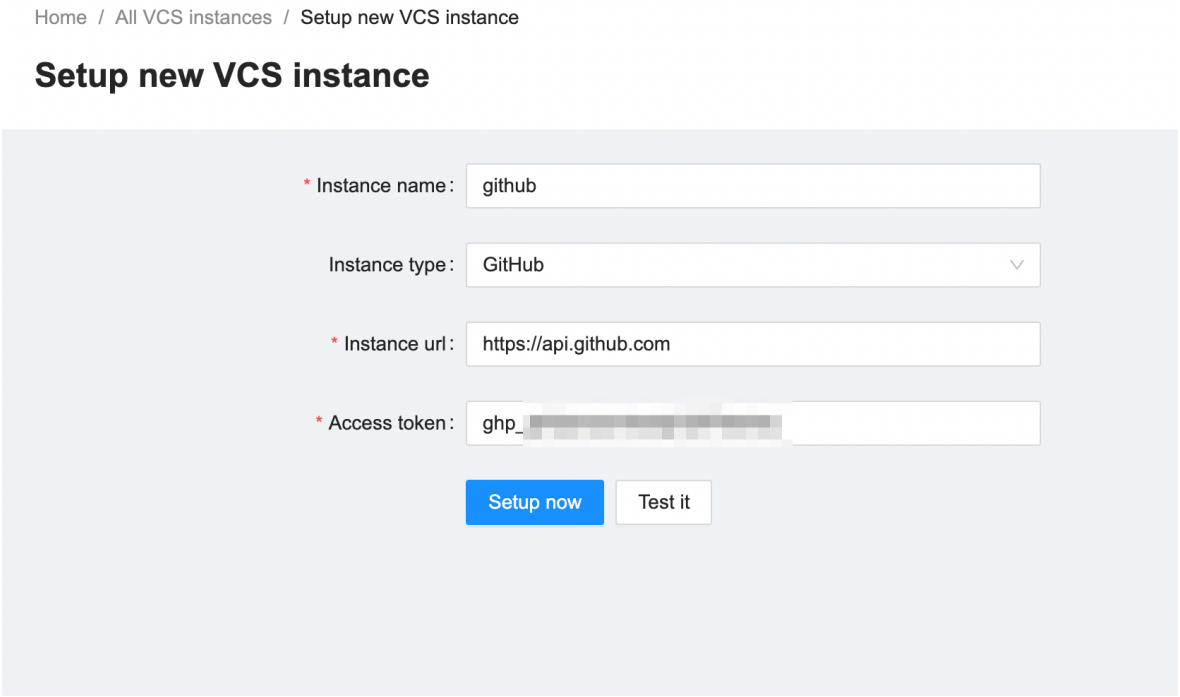
3. Click the **Create personal access token** button.
4. Copy the generated token.
5. In the CodeScoring interface, go to the `Settings -> VCS` section.
6. Click **Setup new** in the upper right corner.
7. Fill out the form as shown in the screenshot. The token is inserted into the *Access token* field.



#Adding a token for GitHub

Original instructions for generating a token: https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token

1. Log in to your GitHub account; for the cloud version, the login is available at https://github.com/login.
2. If your account email is not verified, be sure to do this according to instructions.
3. Through the user menu in the upper right corner, go to the **Settings** section.
4. Next, in the left menu, select the **Developer settings** section.
5. In the left menu, select the **Personal access tokens** section.
6. Give the token a name, for example, “codescoring-demo”.
7. In the Select scopes section, select all options from the *repos* list.
8. Click the **Generate token** button.
9. Copy the generated token.
10. In the CodeScoring interface, go to the `Settings -> VCS` section.
11. Click **Setup new** in the upper right corner.
12. Fill out the form as shown in the screenshot. The token is inserted into the *Access token* field.

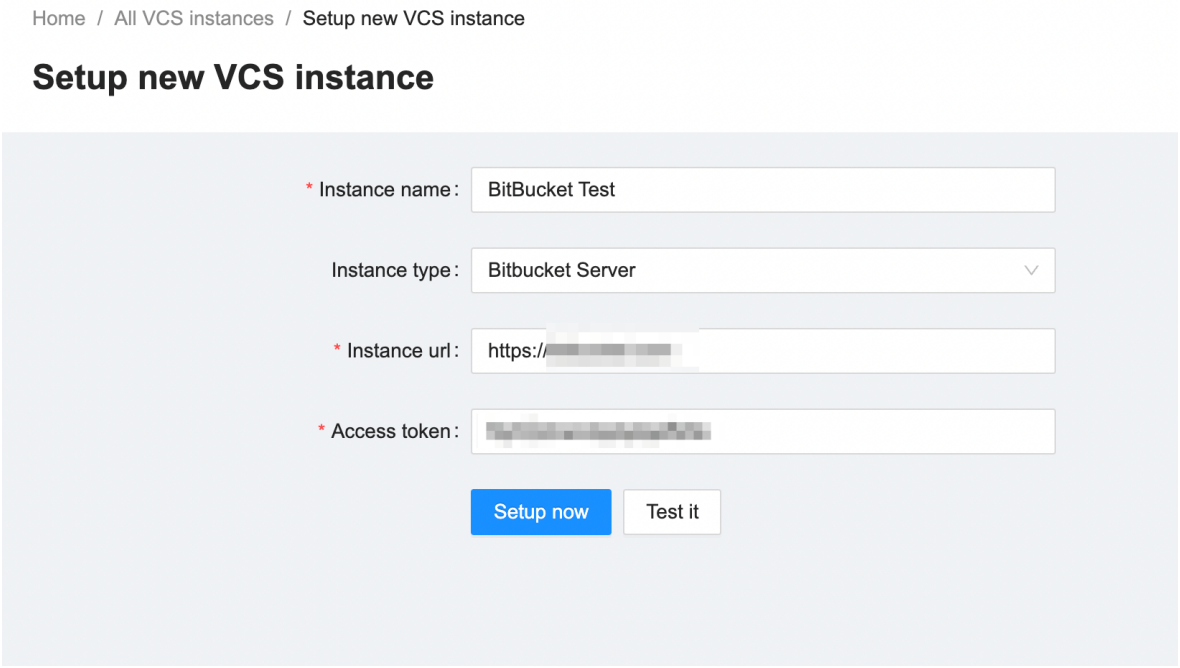


#Adding a token for BitBucket Data Center and Server

Original instructions for generating a token in English: https://confluence.atlassian.com/bitbucketserver072/personal-access-tokens-1005335924.html

1. Log in to your BitBucket account.
2. Through the user menu in the upper right corner, go to the **Manage account** section.
3. Next, in the left menu, select the **Personal access tokens** section.

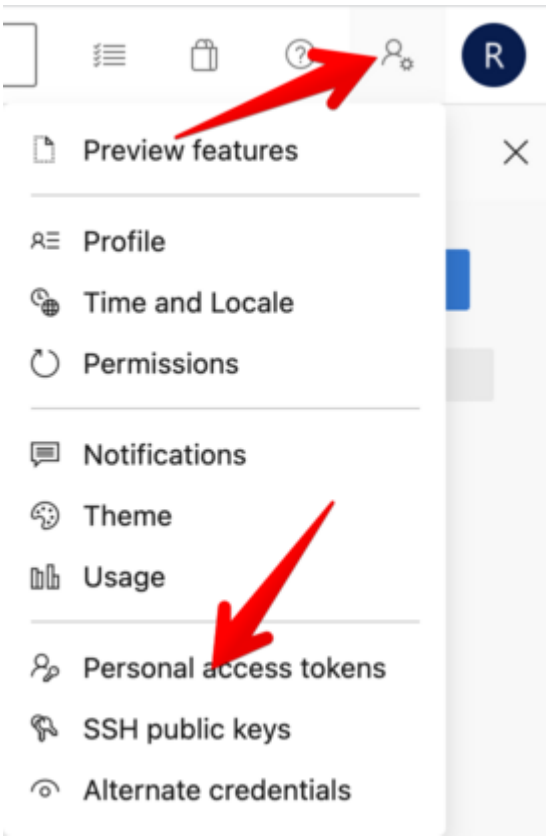
4. Click on **Create token**.
5. Give the token a name, for example, “codescoring-demo”.
6. In the **Permissions** section, give read rights to projects and repositories.
7. In the **Expiry** section, if desired, set the token lifetime.
8. Click the **Create** button.
9. Copy the generated token.
10. In the CodeScoring interface, go to the `Settings -> VCS` section.
11. Click **Setup new** in the upper right corner.
12. Fill out the form as shown in the screenshot. The token is inserted into the *Access token* field.



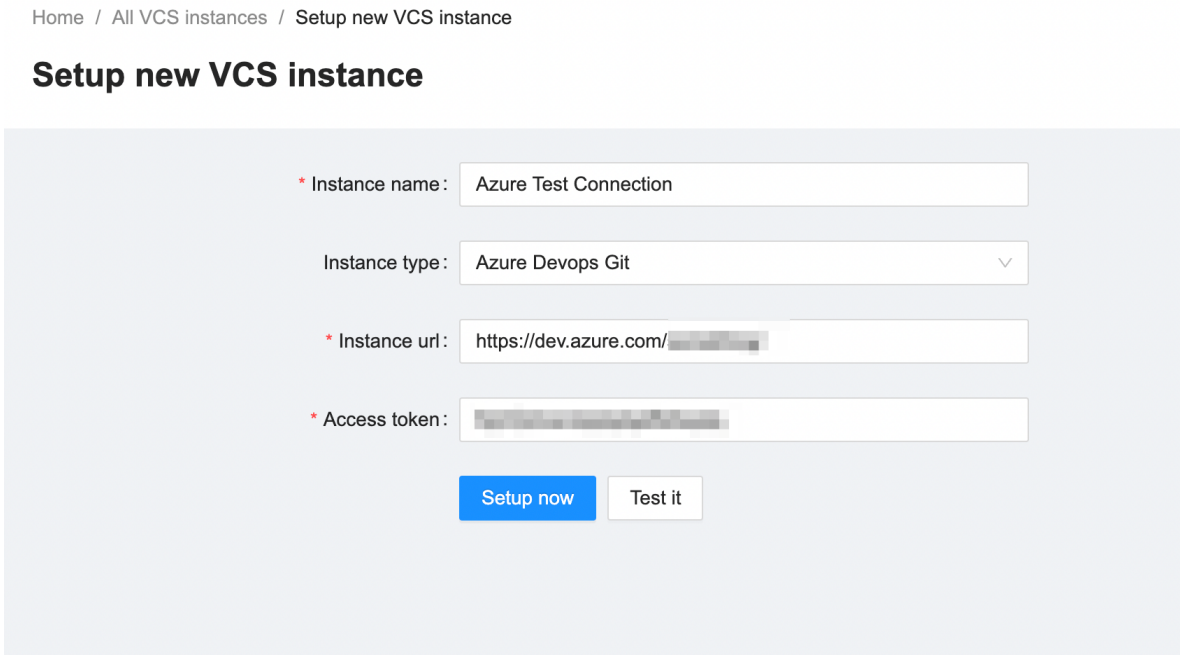
#Adding a token for Azure DevOps Git

Original instructions for generating a token in English: https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate?view=azure-devops

1. Sign in to your Azure DevOps account.
2. Through the user menu in the upper right corner, go to the **Personal access tokens** section.



1. Next, click the **New token** button.
2. Set a name for the token, for example, “codescoring-demo”, and the validity period of the token.
3. In the *Scopes* section, be sure to check **Read** access for the **Code** and **Identity** entities.
4. Click the **Create** button.
5. Copy the generated token.
6. In the CodeScoring interface, go to the `Settings -> VCS` section.
7. Click **Setup new** in the upper right corner.
8. Fill out the form as shown in the screenshot. The token is inserted into the *Access token* field.



#Connecting GitFlic via ssh

1. Create a new ssh key pair without passphase with type **RSA_SHA512**,**SSH_ED25519** or **ECDSA-SHA2-nistp256**.

   ```
   ssh-keygen -t ed25519 -N "" -q -f /tmp/id25519
   ```

2. Go to GitFlic web interface, `Settings -> Keys` and add **public** key to GitFlic key pool.

   You can view the contents of the public key using the following command:

   ```
   cat /tmp/id25519.pub
   ```

3. Go to the CodeScoring interface in the `Settings -> SSH Keys` section, open the form for adding a key by clicking the **Setup New** button and add the **private** key to the CodeScoring key pool.

   You can view the contents of the private key using the following command:

   ```
   cat /tmp/id25519
   ```

4. Go to the `Settings -> VCS` section, open the form for adding a VCS by clicking the **Setup New** button and fill in the fields for creating a VCS instance:

   - **Instance name** – VCS connection name;
   - **Instance clone type** – cloning type (select SSH type);
   - **Instance type** – VCS instance type (select GitFlic type);
   - **Instance url** – the address where VCS is available;
   - **SSH key** – the SSH key created in step 3.

5. Go to the `Settings -> Project` section, open the form for creating a new project by clicking the **Create New** button and fill in the project creation fields:

   - **Repository** – a link to the repository in GitFlic;
   - **VCS** – the name of the VCS instance created in step 6;
   - **Name** – the name of the project in Codescoring.

6. Save the project.

#Synchronization of data from VCS

CodeScoring automatically synchronizes data from version control systems when adding a project, running analysis, and as part of background tasks. All mechanisms for obtaining and updating data from `git` are described below:

1. When adding a VCS-project, CodeScoring creates a local copy of the repository in `bare` mode and saves it in the project directory.
2. When running SCA, TQI or Secrets analysis, the system downloads the current state of the branch specified in the project settings.
3. Once a month, CodeScoring initiates a background task that synchronizes all connected VCS projects with remote repositories. This is a preventive update that does not replace synchronization when running analysis, but only speeds it up for projects where the analysis has not been performed for a long time.
    - `git fetch` is performed for all VCS projects
    - Tasks are distributed using the Huey tool to evenly distribute the load over the remaining days of the month

Note: if you need to synchronize the repository early, you can use a manual code update on the project page in the `Settings -> Projects` section.

#Managing projects

A project in CodeScoring is a part of the analyzed code base. It is possible to create two types of projects in the system:

- **VCS project** – linked to a repository in the version control system;
- **CLI project** – not linked to a repository and allows you to save scanning results from the johnny console agent or download a ready-made SBOM file.

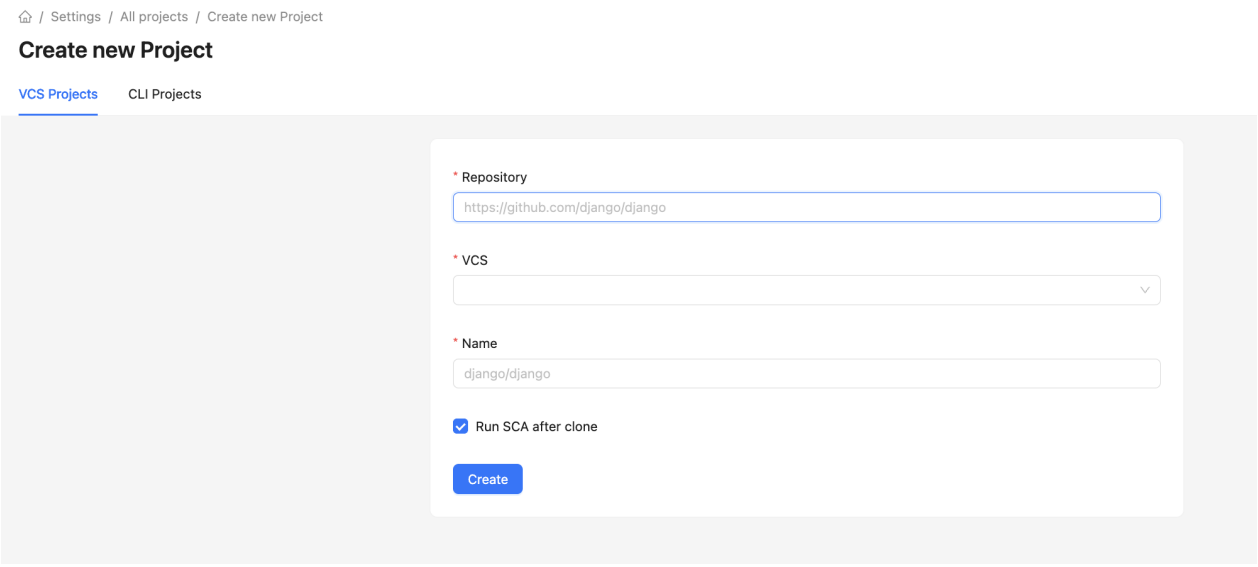
Projects are managed in the `Settings->Projects` section.

#Creating a VCS project

Important

You can create a VCS project only after creating a connection to the version control system.

To go to the VCS project creation form, click the **Create** button and select the **VCS projects** tab. The project type selected during creation cannot be converted to another.



1. To add a project, you need to add a link to the repository in the form, select the appropriate version control system from the list, set the project name and the option to run SCA analysis immediately after cloning.
2. After adding, the initial cloning of the project occurs, the time of which will depend on the size of the repository.
3. After cloning, the project will be available for editing and new analyses.

The system accepts links to repositories in the following formats:

- 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>`

After adding a repository, manual update of the project code is available on the VCS project settings page by clicking the button **Update project code**.

Attention! If the analyzed project is commercial, it is recommended to specify the **Commercial License** license category for it to ensure that license compatibility policy works correctly.

#Creating a CLI project

To go to the CLI project creation form, click the **Create new** button and select the CLI Projects tab.

To add a project, simply fill in its name in the **Name** field.

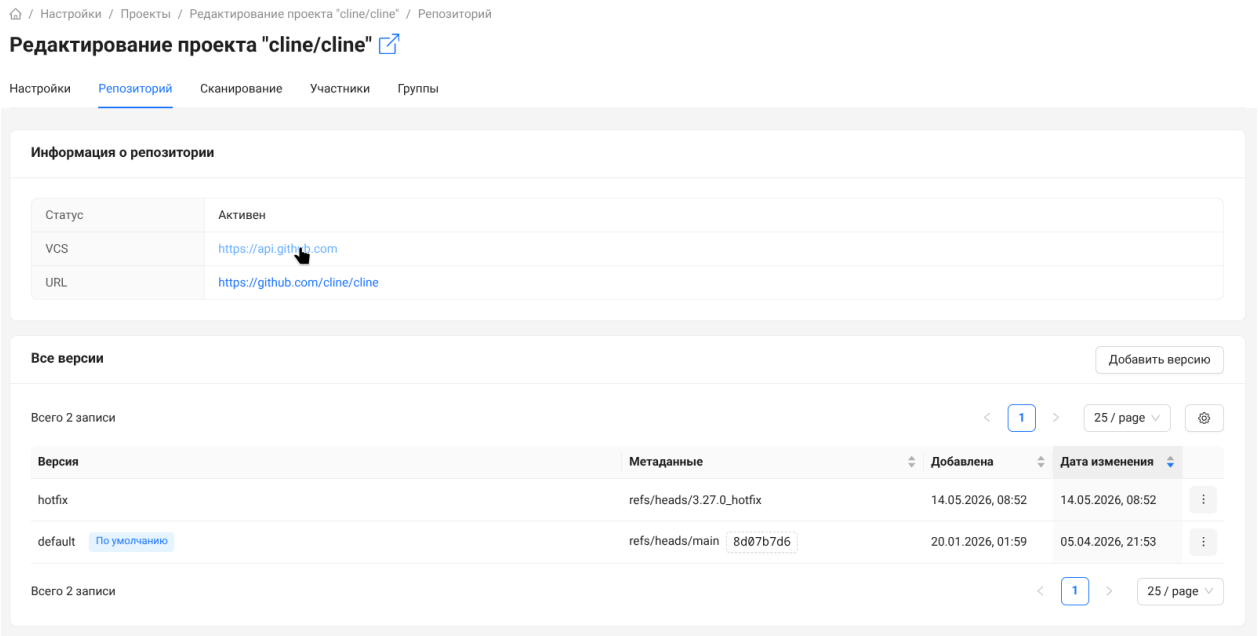
#Creating project categories

Categories are used to group system projects by semantic groups.

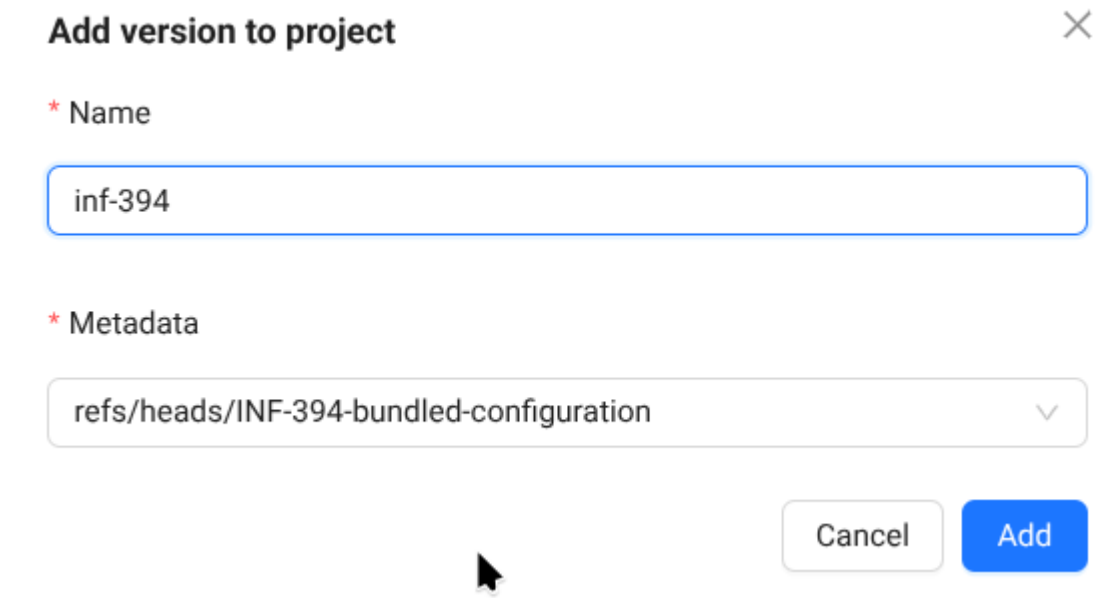
Categories are managed in the `Settings -> Categories` section. You can go to the category creation form by clicking the **Create** button. To create a category, simply give it a name.

#Version control

Version control takes place in the project settings, in the `Repository` section.

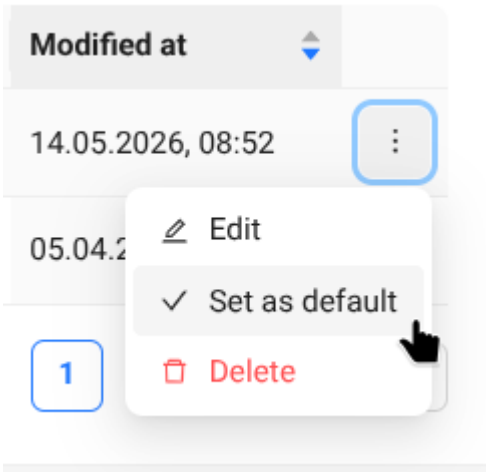


You can add a new version by clicking **Add version**. To create a version in a CLI project, it is enough to set a name for it. For a VCS project, you must specify the metadata of the tag or repository header.



You can also add a new version when scanning the project using the corresponding menu.

You can set the default version by clicking on the button in the context menu to record the version. **Set as default**.



You can delete a version in the context menu of the version entry by clicking the **Delete** button. The default version cannot be deleted.

When editing a version, you can change the name and metadata.

#Creating proprietors

Proprietors are abstract entities within an organization. They make it easy to group projects into different groups of responsible users.

Entities are managed under `Settings -> Proprietors`. You can go to the proprietor creation form by clicking the **Create new** button. In the form, it is

mandatory to fill in the **Name** field of the proprietor name and optional fields as desired.

#Binding of authors

Within the system there is a possibility to create a link between a code author and a particular proprietor. To do this, you should go to the list of authors using the **Manage authors mapping** button and select the required entity in the Proprietor field.

#Notification settings

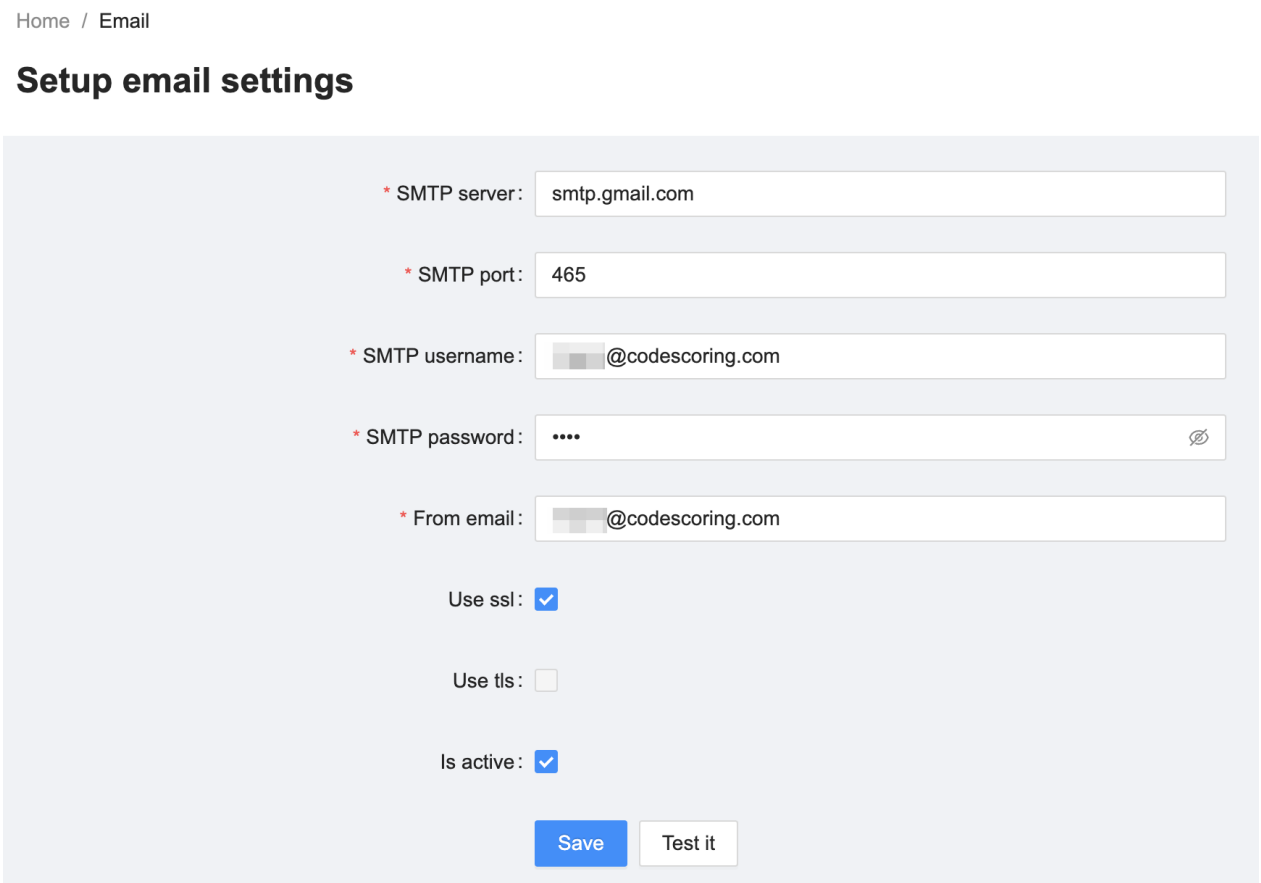
For each policy, you can configure additional notifications, in addition to viewing the results in the `Policy Alerts` section. There are currently three notification methods available: via **email** and via the **Jira** and **Kaiten** task managers.

#Email notifications

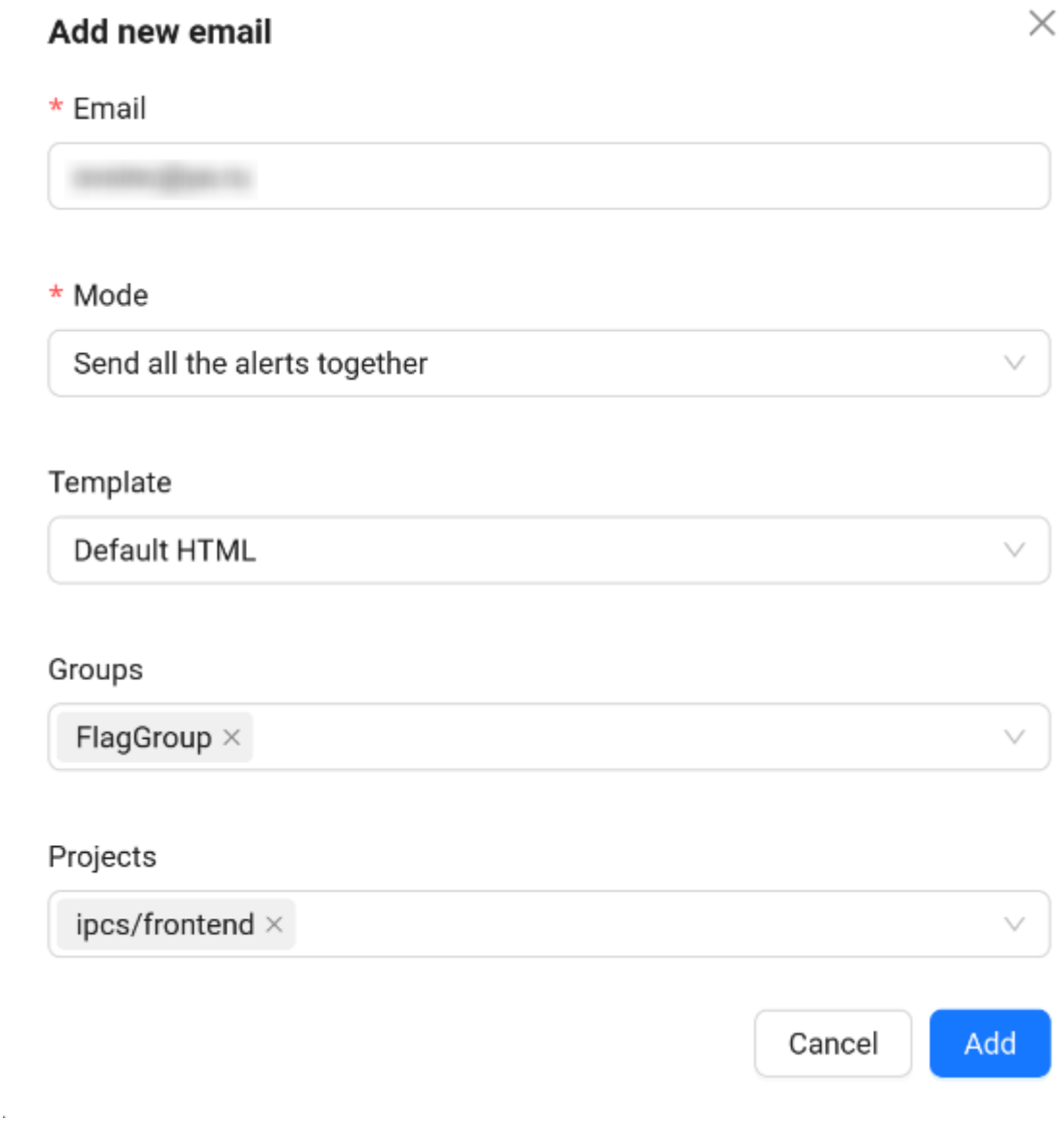
Email notifications are sent via SMTP integration.

To send notifications via email, the mail server must be configured beforehand in the `Settings -> Notifications -> Email` section. To do this, you need to fill in all mandatory fields and check the **Is active** flag.

You can check if the configuration is correct by clicking the **Test it** button.



After configuring the mail server, in the `Actions` section of the policy page you can add an email address to which emails with policy alerts will be sent:



- **Email** — email address;
- **Mode** — email sending mode:
  - Send all the alerts together;
  - Send each alert separately;
- **Template** - template name. If not specified the default template would be used;
- **Groups** — groups of projects to which notification is made. If not specified, all groups are implied;
- **Projects** — the specific projects for which the notification is made. If not specified, all projects are implied.

If both groups and projects are specified, alerts will include information for all projects in the specified groups and for all specified projects.

The policy results email is sent **after the project** has been scanned. The content of the email depends on the selected template.

#Task managers integration

CodeScoring supports integration with the Jira and Kaiten task managers to generate tasks for triggered policies. The integration is configured in the `Settings -> Notifications -> Task managers` section.

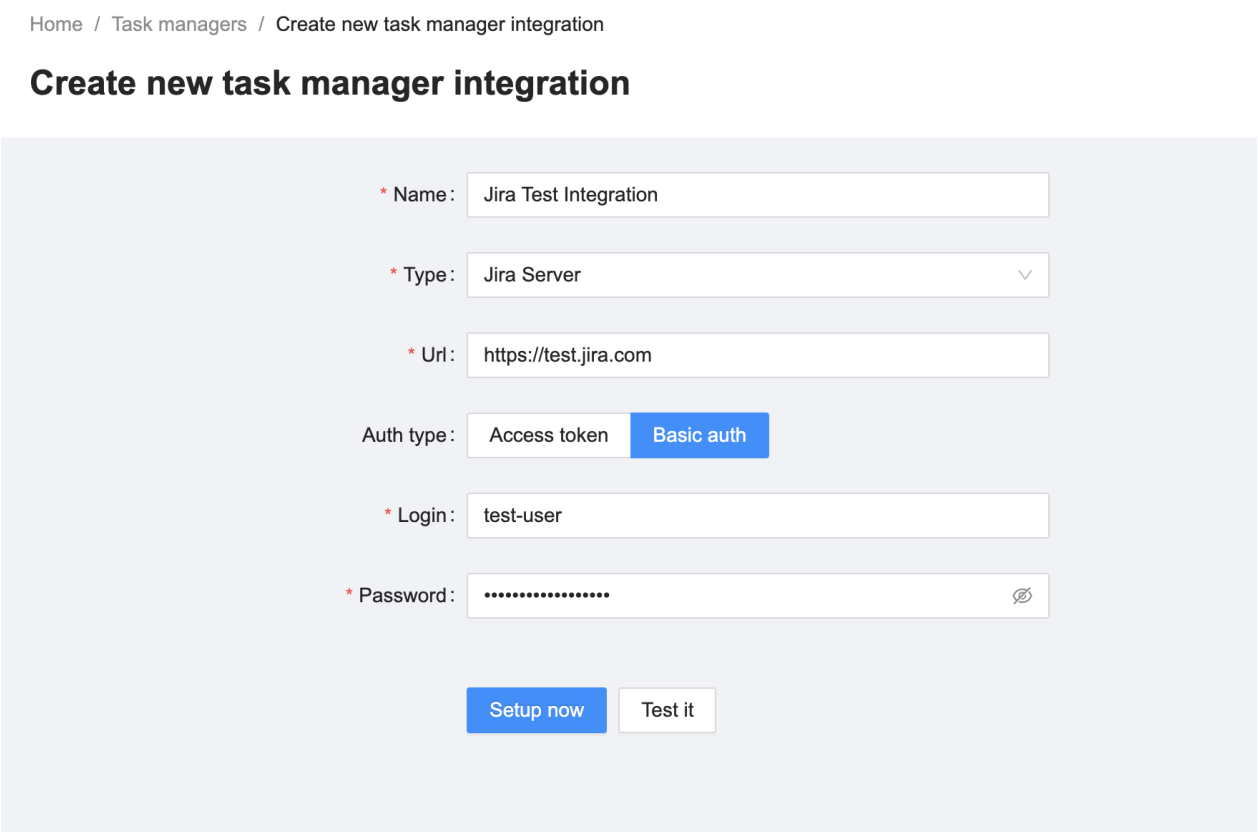
The **Setup new** button form is used to create a new integration.

- **Name** - name of the integration;
- **Type** - type of task manager;
- **URL** - address where the task manager can be accessed;
- **Auth type** - authentication type (via access token or login and password).

Kaiten authentication type

Kaiten only supports authentication via access token.

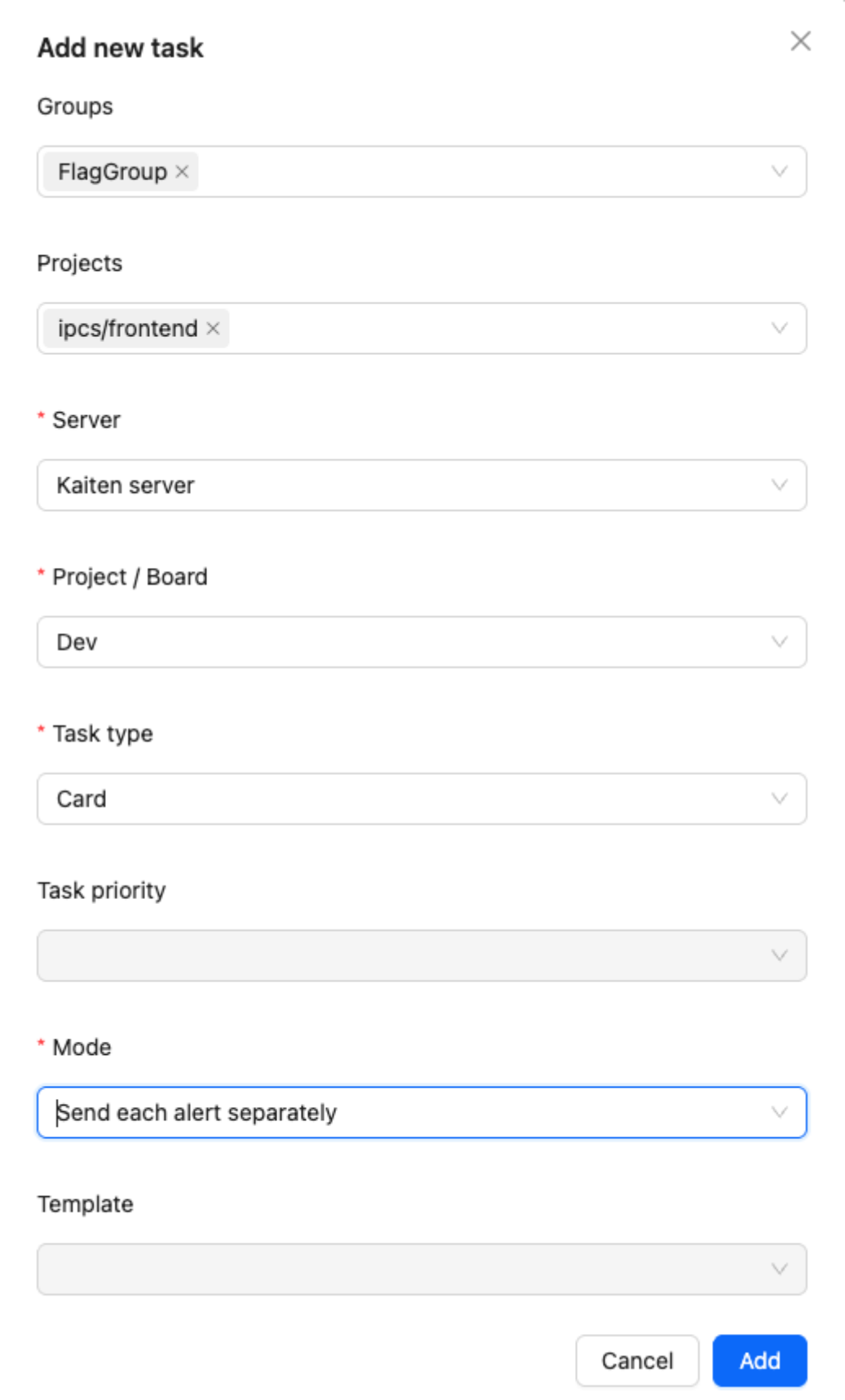
After filling in the fields, you can test the connection to the server by clicking **Test it**, or complete the creation by clicking **Setup now**.



#Creating tasks in task managers

Once the integration is configured, under `Actions` on the policy page, you can add a Jira or Kaiten server on which to create a task with the policy results:

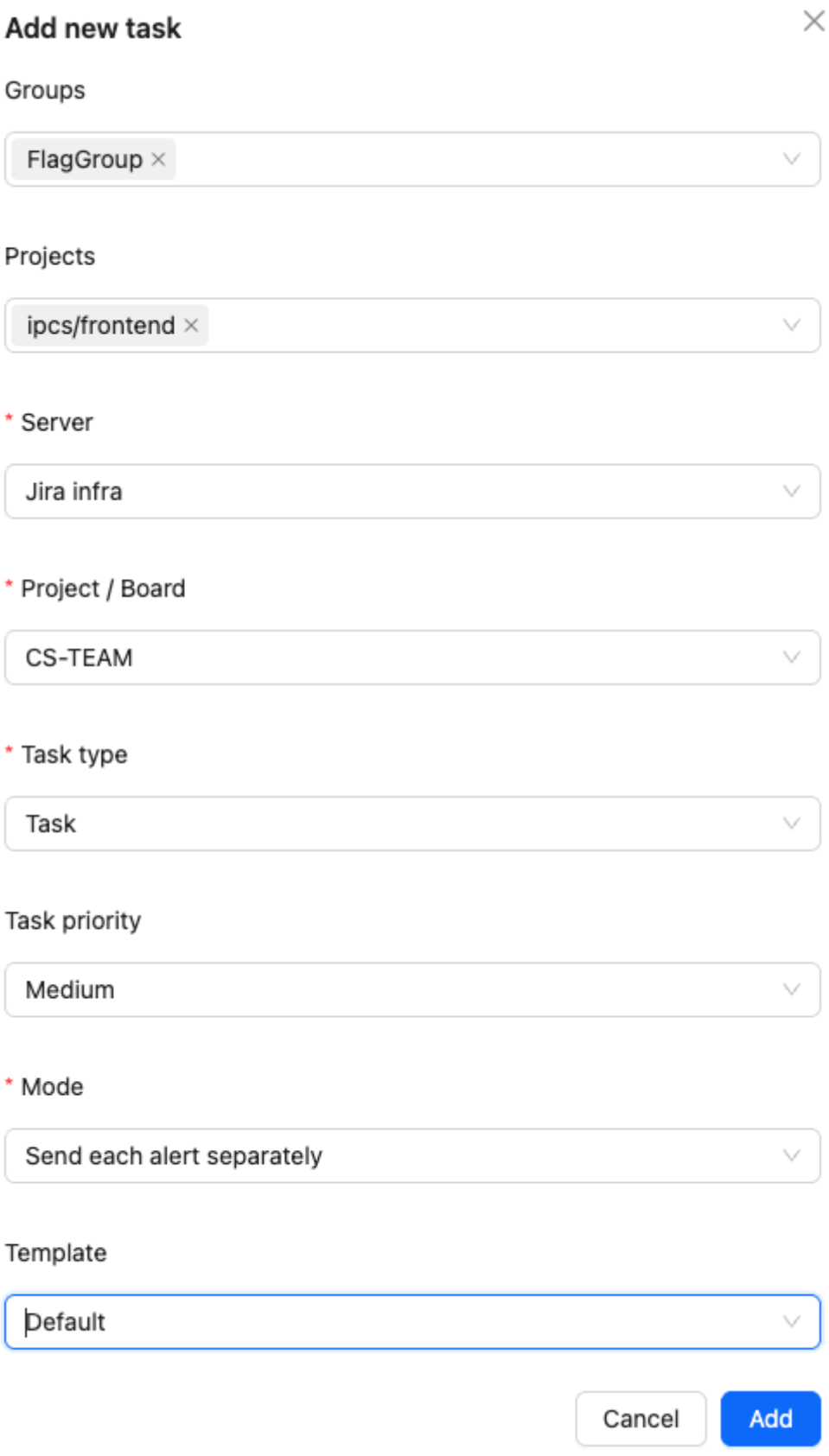
#Creating tasks in Kaiten



- **Mode** — task sending mode:
  - Send all the alerts together;

    - Send each alert separately.
- **Groups** — groups of projects to which notification is made. If not specified, all groups are implied;
- **Projects** — the specific projects for which the notification is made. If not specified, all projects are implied.
- **Server** — task manager (in this case Kaiten);
- **Project/Board** — Kaiten-boards;
- **Task type** — card type;

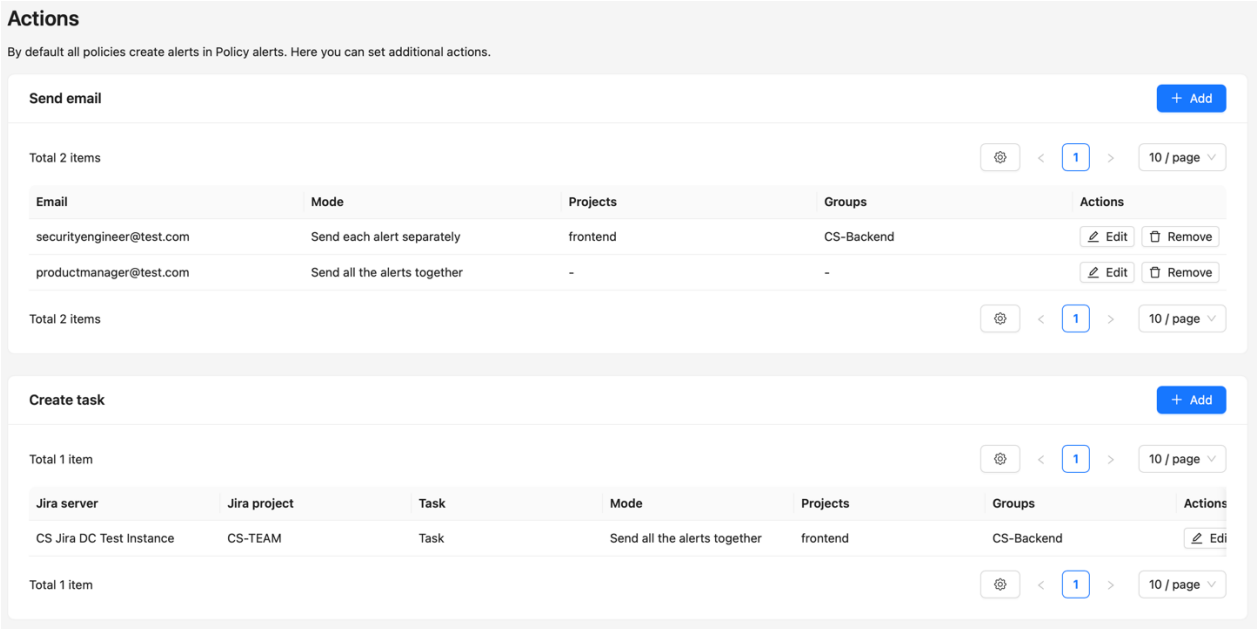
#Creating tasks in Jira



- **Mode** — task sending mode:
  - Send all the alerts together;

    - Send each alert separately.
- **Groups** — groups of projects to which notification is made. If not specified, all groups are implied;
- **Projects** — the specific projects for which the notification is made. If not specified, all projects are implied.
- **Server** — task manager (in this case Jira);
- **Project/Board** — Jira-project;
- **Task type** — card type: *Task*, *Story* or *Bug*;
- **Task priority** - card priority. If not specified the default Jira task priority would be used;
- **Template** - template name. If not specified the default template would be used.

If both groups and projects are specified, alerts will include information for all projects in the specified groups and for all specified projects.



#Template management

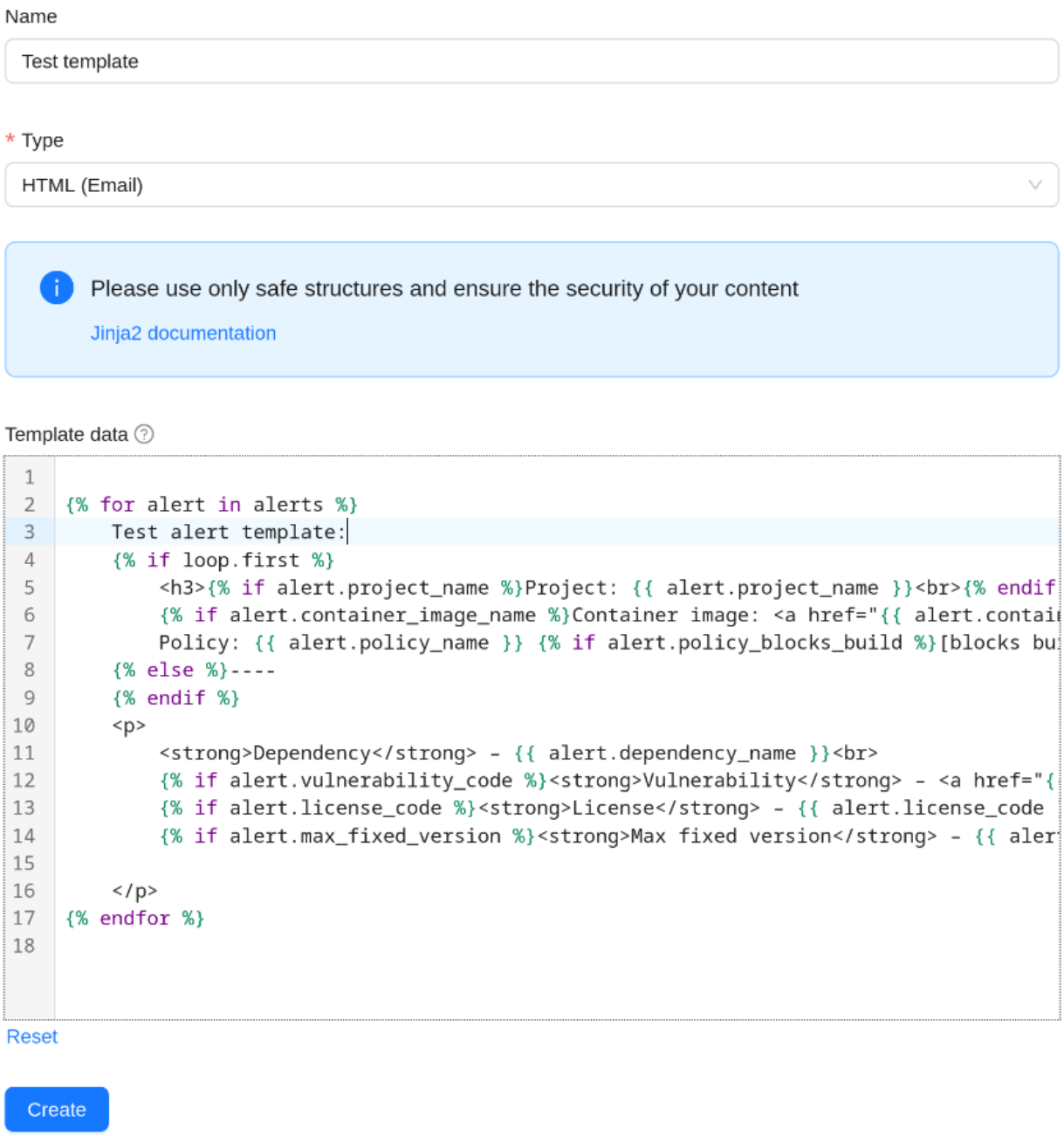
CodeScoring supports the ability to use custom templates for email notifications or creating Jira issues. Template management is available in the section `Settings -> Notifications -> Templates`.

To create a new template, use a form with the following fields:

- Name;
- Type - templates are divided into types depending on their use: Markdown for Jira issues and HTML for email notifications;
- Template data - template content in jinja2 format.

Important

Use only secure structures. ::: Before finalizing the template, make sure your data is secure.



When filling in the **Template data** field of the form, please note that the template can be used both for the "Send each notification separately" and for "Send all notifications together" sending modes.

The content of email message or jira task is generated based on the template and `alert context`. The context provides a `collection of alerts` (when using "separate" send mode, the collection contains only one alert).

For each `alert` you can use the following variables:

- policy_alert_level: str - alert criticality level;
- policy_alert_stage: str - development cycle stages;
- policy_alert_matched_criteria_list: list[str] - list of The policy triggers;
- policy_name: str - policy name;

- policy_blocks_build: bool - blocking CI build or download of a component from a proxy repository;
- policy_block_delay: int - delay (days) from the first policy trigger to blocking;
- policy_is_block_delayed: bool - block delay is used;
- dependency_name: str - name of the dependency;
- dependency_link: str - link to dependency;
- dependency_technology: str - programming language or ecosystem;
- vulnerability_code: Optional[str] - vulnerability identifier from external database;
- vulnerability_link: Optional[str] - link to the vulnerability;
- max_fixed_version: Optional[str] - max fixed version;
- license_code: Optional[str] - license;
- project_name: Optional[str] - project name;
- container_image_name: Optional[str] - name of container image;
- container_image_link: Optional[str] - link to container image.

:::note Note All links are leading to the platform on which the data for the email or task in Jira was generated.

#Audit log

Audit log is a journal of events in the **Codescoring** system. It is located at `Settings -> Audit log`.

In the audit log all the system events, user actions and errors are listed. Each event contains the following data:

- **Event time** - date and time of event;
- **Actor** - username (for system actions the value is `system`);
- **Message** - a message with details;
- **Duration** - the duration of analysis and integration events.

The records in the journal can be filtered by date range or username, and specific events can be found using the **Search** field. Exporting of the journal in CSV format is also available.

Events in the audit log are divided into several categories. Below is a complete list of possible events for each category, along with their descriptions.

#License Activation

Event Text	Description
Object <ActivationKey: Owner: {owner.name}. Expired: YYYY-MM-DD HH:MM:SS+HH:MM (timezone). Authors limit: {limit.number}> created	Activation of software license with owner and author limit specified

Event Text	Description
No activation key	No activation key present
Problem with activation key: {status.lower()}	Issue with activation key

#User Authentication

Event Text	Description
User logged in	User successfully authenticated in the system
User logged out	User logged out of the system
Failed login attempt {username}	Incorrect password entered during authentication attempt

#Object Management

Event Text	Description
Object {instance!r} created	Creation of any object in the system via user interface
Object {instance!r} updated	Update of any object in the system via user interface
Object {instance!r} deleted	Deletion of any object in the system via user interface

#SCA Analysis Execution

Event Text	Description
[SCA][{analysis_run.project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Analysis started	SCA analysis of the project started
[SCA][{analysis_run.project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Analysis finished	SCA analysis of the project completed

Event Text	Description
[SCA][{analysis_run.project.name}] [{analysis_run.sequence}/{analysis_run.pk}] Analysis failed. Check server logs.	SCA analysis of the project failed. Check server logs.
Failed to clone for repository {repository.name}.	Repository cloning failed
Failed to detect branch for repository "{repository.name}	Failed to detect branch for repository
[SCA][{project.name}][{analysis_run.sequence}/{analysis_run.pk}] Clone source code	Project source code cloning started
[SCA][{project.name}][{analysis_run.sequence}/{analysis_run.pk}] Collect files data	File data collection for the project started
[SCA][{project.name}][{analysis_run.sequence}/{analysis_run.pk}] Collect manifests	Manifest collection for the project started
[SCA][{project.name}][{analysis_run.sequence}/{analysis_run.pk}] Create pipeline	Pipeline creation for SCA analysis started
[SCA][{project.name}][{analysis_run.sequence}/{analysis_run.pk}] Check policies	Policy checking for SCA analysis started
[SCA][{project.name}][{analysis_run.sequence}/{analysis_run.pk}] Update project metrics	Project metrics update for SCA analysis started
[SCA][{project.name}][{analysis_run.sequence}/{analysis_run.pk}] Analyze dependencies	Dependency analysis for SCA started
[[SCA][{project.name}] Analysis didn't start (Reason: {err.message})]	SCA analysis did not start due to an error
Overall SCA run started for {len(projects)} project(s)	Overall SCA analysis for projects started

Each SCA analysis event contains a sequential analysis number in the project and a **UUID** of the execution.

#TQI Analysis Execution

Event Text	Description
Rebuild author {primary_email}	Updating author information based on merge rules by primary email

Event Text	Description
Generate authors merge rules	Creating author merge rules
(Run #{run_id}) Collect commits data for project {project.name}	Commit data collection for the project started
(Run #{run_id}) Create authors	Author creation started
(Run #{run_id}) Load authors OSS contributions	Loading authors' contributions to OSS
(Run #{run_id}) Authors analysis started	Author analysis started
(Run #{run_id}) Authors analysis completed	Author analysis successfully completed
(Run #{run_id}) Authors analysis failed. Check server logs.	Author analysis failed. Check server logs.
(Run #{run_id}) Authors analysis cancelled	Author analysis canceled
(Run #{run_id}) Update project {project.name}	Project update started
(Run #{run_id}) Clones analysis started	Cloned code analysis started
(Run #{run_id}) Clones analysis completed	Cloned code analysis completed
(Run #{run_id}) Clones analysis failed. Check server logs.	Cloned code analysis failed. Check server logs.
(Run #{run_id}) Clones analysis cancelled	Cloned code analysis canceled
(Run #{run_id}) Clone source code for project {project.name}	Project source code repository cloning started

Each TQI analysis event contains a **UUID** of the execution.

#Policy Management

Event Text	Description
Policy ignore {policy_ignore} created	Creation of a policy ignore rule
Policy ignore {policy_ignore} activated	Activation of a policy ignore rule
To Policy ignore {policy_ignore} added Policy {policy_alert.policy}	Policy added to an existing ignore rule
Policy "{policy.name}" (id: {policy.pk}) skipped. Reason: {err!r}	Policy skipped due to an error

#Secrets Analysis

Event Text	Description
[Secrets][{analysis_run.analysis_object}] Analysis started	Secrets analysis started
[Secrets] Training run started	Training of the user model based on labeling results started
[Secrets][{analysis_run.analysis_object}] Analysis finished	Secrets analysis completed
[Secrets] Training run finished	Training of the user model based on labeling results completed
[Secrets][{analysis_run.analysis_object}] Analysis failed. Check server logs.	Secrets analysis failed
[Secrets] Training run failed. Check server logs.	User model training failed

#Container Image Analysis

Event Text	Description
In container image {container_image} dependency {dep_name_and_version} was changed	A dependency was changed in the container image

Event Text	Description
Updating images list for registry {container_registry} triggered via update button.	Forced update of the image list from the registry started

#LDAP Operations

Event Text	Description
Applying all LDAP group mapping rules triggered	Execution of all LDAP group mapping rules started
{message} While processing, failed to apply some of rules related to following LDAP servers: {', '.join(ldap_servers_mapping_failed_for)}. Check server logs.	Error applying group mapping rules for specified LDAP servers

#Miscellaneous

Event Text	Description
(Run #{task.id}) Analysis started via API	Analysis started via API
Some tasks in analysis failed	Some tasks in the analysis failed
Could not connect to OSS Index, reason: {err}	Could not connect to OSS Index due to an error
There is already running analysis	Analysis is already running
Another analysis in progress. Parallel execution forbidden.	Parallel execution of analysis is forbidden
Repo path for {project} does not exist, setting status to Not cloned	Repository path not found, status set to "Not cloned"
Failed to clone for repository {project.repo_name} because project was deleted	Failed to clone the repository because the project was deleted
Updating {update_type} since {updated_after} / Updating all {update_type}	Licenses and vulnerabilities update info

#Configuring metrics

CodeScoring stores metrics in a format supported by the **Prometheus** monitoring tool. Currently there are metrics available from CodeScoring platform and plugins in proxy repositories.

#Collect metrics from platform

Metrics from platform are available via **CodeScoring API** at `{platform-url}/api/metrics`. In order to configure metrics tracking in Prometheus, the following steps are required:

1. open the `prometheus.yml` configuration file and add parameters to monitor metrics. The following is an example:

```
global:
  scrape_interval: 15s

  scrape_configs:
  - job_name: 'demo-codescoring'
    metrics_path: '/api/metrics'
    static_configs:
      - targets: ['{platform-url}'] # platform host
address
  - job_name: 'osa'
    metrics_path: '/api/osa/metrics'
    static_configs:
      - targets: ['{platform-url}'] # platform host
address
```

2. Restart Prometheus for the changes to take effect.

3. Open the Prometheus interface and go to the **Graph** page. In the query field, enter one of the metrics:

#Queue metrics

- **codescoring_tasks_queue_size_total** – total number of tasks waiting to be executed;
- **codescoring_tasks_schedule_queue_size_total** – total number of scheduled tasks waiting to be executed;
- **codescoring_tasks_running_tasks_total** – total number of tasks currently running;
- **codescoring_celery_queue_size_total** – total number of tasks in Celery queues;
- **codescoring_celery_running_tasks_total** – total number of running tasks in Celery queues.

These metrics can be filtered by the queue label with the following possible values:

#Main Queues:

- **ipcs** - main queue for asynchronous tasks;
- **osa-container-image-scan** - used for container image scans;
- **osa-package-scan** - used for scans of packages, requested through the OSA module;
- **policy** - used for asynchronous policy calculations;
- **tqi** - used during TQI analyses;
- **sca-external-scan** - used for SCA runs launched via Johnny CLI agent;
- **secrets** - used during Secrets analyses.

#Celery Queues:

- **default** - main queue for asynchronous tasks;
- **webhooks** - used for webhook tasks execution;
- **secrets-external-scan** - used for TQI analyses launched via Johnny CLI agent;
- **osa-maintenance** - used for maintenance tasks related to OSA module;
- **osa-background-update** - used for background updates of OSA-related entities;
- **oss-index** - used for loading data from OSS Index in cases, when OSS Index integration is enabled;
- **migration-background-tasks** - used for tasks related to background recalculation of data caused by the database schema change;
- **media** - used for asynchronous generation of reports;
- **media-cleaner** - used for periodic cleanup of reports;
- **sca-tasks** - used for SCA-related tasks.

#Analysis metrics

codescoring_running_analyses_total – total number of running analyses.

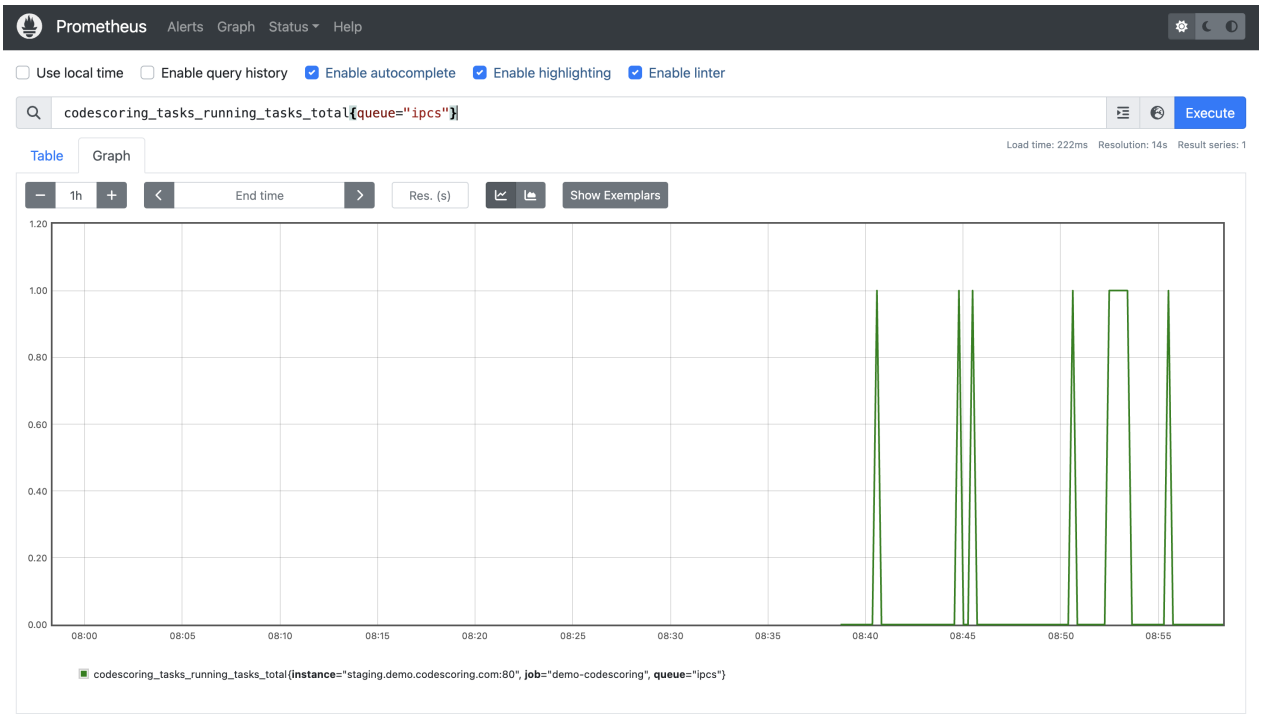
This metric can be filtered by the `analysis_type` label with the following possible values:

- `sca` – dependency analysis;
- `authors` – code authors analysis;
- `clones` – code clone search.

#OSA plugin request metrics

- **codescoring_registration_packages_queue_size** – package registration queue from OSA;
- **codescoring_registration_container_images_queue_size** – container image registration queue from OSA.

Example of metrics visualization:



#OSA metrics collection

OSA metrics contain information about connection status, number and time of requests, as well as scan and block status of components in repository managers. Metrics are available in the **CodeScoring API** at `{platform-url}/api/osa/metrics`.

#Connection metrics

codescoring_osa_api_db_connection_pool – database connection pool status.

The metric can be filtered by the `measure` label with the following possible values:

- `pool_min` – minimum pool size;
- `pool_max` – maximum pool size;
- `pool_size` – current pool size;
- `pool_available` – available connections;
- `requests_waiting` – number of pending requests;
- `pool_used` – used connections.

codescoring_osa_api_redis_connection_pool – Redis connection pool status.

The metric can be filtered by the `measure` label with the following possible values:

- `available_connections` – available connections;
- `in_use_connections` – used connections;
- `total_connections` – total number of connections.

#Component request metrics

codescoring_osa_api_http_request_duration_seconds – HTTP request duration in seconds.

The metric is presented as a histogram and includes:

- `codescoring_osa_api_http_request_duration_seconds_sum` – total request time;
- `codescoring_osa_api_http_request_duration_seconds_bucket` – request time distribution by time intervals;
- `codescoring_osa_api_http_request_duration_seconds_count` – total number of requests.

The metric can be filtered by labels:

- `handler` – request handler (e.g. `/api/osa/packages/`);
- `method` – request method (e.g. POST);
- `status` – HTTP status of the response (e.g. 2xx, 4xx, 5xx).

Example:

```
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 – total number of HTTP requests.

The metric can be filtered by the same labels as `codescoring_osa_api_http_request_duration_seconds`.

#Component status metrics

codescoring_osa_api_requested_component_block_status_total – number of components in different blocking statuses.

The metric can be filtered by labels:

- `block_status` – blocking status (e.g. `not_blocked`, `blocked_by_policies`, `blocked_scan_failed`);
- `object_type` – object type (e.g. `package`, `container_image`).

Example:

```
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 – number of components in different scan statuses.

The metric can be filtered by labels:

- `scan_status` – scan status (e.g. `not_scanned`, `scanned`);
- `object_type` – object type (for example, `package`, `container_image`).

Example:

```
codescoring_osa_api_requested_component_scan_status_total{obje
ct_type="container_image",scan_status="not_scanned"} 0.0
```

#Index API-related metrics

index_api_failure_rate - number of sequentially failed requests to Index API.

#Connecting webhooks

CodeScoring supports a webhook-based notification system. When events occur, a POST HTTP request is sent to the specified URL.

#Adding a new webhook

To add a new webhook to the platform, follow these steps:

1. Go to the `Settings -> Webhooks` section.
2. Click the **Setup new** button.
3. Fill in the fields in the form:
    - **Name** – name in the CodeScoring system;
    - **URL** – address with the protocol. For example: `http://webhook.com/`;
    - **Projects** – list of projects for which webhooks will be applied. If the field is empty, all projects will be taken into account;
    - **Events** – list of events for which the notification will be triggered;
    - **Token** – token that is transmitted in the HTTP request (`X-CodeScoring-Authentication` header);
4. Test the connection after filling in the data by clicking the **Test it** button. When testing, the `test` trigger is used with an empty payload.

After creating a new connection by clicking the **Setup now** button, the webhook will be displayed in the section list, with the ability to view information about it (**View**), change the webhook parameters (**Edit**), or delete it (**Delete**).

#Request body structure

```
{
  "events": [
    {
      "created_at": "datetime_in_iso_format",
      "trigger": "trigger_name",
      "payload": {
        ...
      }
    },
    ...
  ]
}
```

#Request sending mechanism

Requests are processed every 5 seconds, according to a periodic task performed by the system.

A webhook usually sends one event per request. However, in the following cases, an array of events can be sent:

- **Sending accumulated events:** if several events occurred within a unit of time (default is 5 seconds), they will be sent in a single request;
- **Retries:** if the server did not respond or returned an error on the previous attempt, all unsuccessful events are resent in a single request.

HTTP codes in the range [200; 299] are considered successful. If the server's response is outside this range, the event is considered unsuccessful, accumulated, and resent according to the following schedule: 1 minute → 5 minutes → 30 minutes → 3 hours → 12 hours → 24 hours → 48 hours.

If at least one request remains unsuccessful after 48 hours, the webhook is disabled and no more events are sent.

#Webhooks management via API

To manage webhooks, an API is provided at the endpoint `/api/settings/webhooks` with the following commands:

- `GET /api/settings/webhooks/` — get a list of all webhooks;
- `POST /api/settings/webhooks/` — create a new webhook;
- `GET /api/settings/webhooks/{id}/` — get information by ID;
- `PUT /api/settings/webhooks/{id}/` — completely update data;
- `PATCH /api/settings/webhooks/{id}/` — partially update data;
- `DELETE /api/settings/webhooks/{id}/` — delete a webhook by ID;
- `POST /api/settings/webhooks/{id}/refresh_availability_status/` — refresh the availability status;
- `POST /api/settings/webhooks/test/` — test the connection;

- GET `/api/settings/webhooks/triggers/` — get a list of available triggers.

#Available events

Event name	Trigger value	Request payload struc
SCA Policy has been triggered	sca_policy_has_been_triggered	``` "alert_id": 0, "stage": "dev|sourc "level": "info|warn "policy_id": 0, "policy_name": 0, "dependency_id": 0, "dependency_purl": "matched_criteria": "project_id": 0 ```
OSA Policy has been triggered	osa_policy_has_been_triggered	``` "alert_id": 0, "stage": "dev|sourc "level": "info|warn "policy_id": 0, "policy_name": 0, "dependency_id": 0, "dependency_purl": "matched_criteria": "container_image_id "artifact_repositor ```
Project SCA analysis started	project_sca_analysis_started	``` "project_id": 0, "project_name": "pr ```
Project SCA analysis finished	project_sca_analysis_finished	``` "project_id": 0, "project_name": "pr "vulnerabilities_co "dependencies_count ```

Event name	Trigger value	Request payload struc
Project SCA analysis cancelled	project_sca_analysis_cancelled	``` "project_id": 0, "project_name": "pr ```
Project SCA analysis failed	project_sca_analysis_failed	``` "project_id": 0, "project_name": "pr "error": "error_des ```
Container Image SCA analysis started	container_image_sca_analysis_started	``` "container_image_id "container_image_na ```
Container Image SCA analysis finished	container_image_sca_analysis_finished	``` "container_image_id "container_image_na "vulnerabilities_co "dependencies_count ```
Container Image SCA analysis cancelled	container_image_sca_analysis_cancelled	``` "container_image_id "container_image_na ```
Container Image SCA analysis failed	container_image_sca_analysis_failed	``` "container_image_id "container_image_na "error": "error_des ```
Clone analysis started	clones_analysis_started	

Event name	Trigger value	Request payload struc
		``` "project_id": 0, "project_name": "pr ```
Clone analysis finished	clones_analysis_finished	``` "project_id": 0, "project_name": "pr ```
Clone analysis cancelled	clones_analysis_cancelled	``` "project_id": 0, "project_name": "pr
Clone analysis failed	clones_analysis_failed	``` "project_id": 0, "project_name": "pr "error": "error_des ```
Authors analysis started	authors_analysis_started	``` "project_id": 0, "project_name": "pr ```
Authors analysis finished	authors_analysis_finished	``` "project_id": 0, "project_name": "pr ```

Event name	Trigger value	Request payload struc
Authors analysis failed	authors_analysis_failed	``` "project_id": 0, "project_name": "pr "error": "error_des ```
Authors analysis cancelled	authors_analysis_cancelled	``` "project_id": 0, "project_name": "pr ```
Cloning of repository started	cloning_of_repository_started	``` "project_id": 0, "project_name": "pr "repo_url": "repo_u "repo_ref_name": "r ```
Cloning of repository finished	cloning_of_repository_finished	``` "project_id": 0, "project_name": "pr "repo_url": "repo_u "repo_ref_name": "r ```
Cloning of repository failed	cloning_of_repository_failed	``` "project_id": 0, "project_name": "pr "repo_url": "repo_u "repo_ref_name": "r "error": "error_des ```
Project Secrets analysis started	project_secrets_analysis_started	``` "project_id": 0, "project_name": "pr ```

Event name	Trigger value	Request payload struc
Project Secrets analysis finished	project_secrets_analysis_finished	``` "project_id": 0, "project_name": "pr ```
Project Secrets analysis cancelled	project_secrets_analysis_cancelled	``` "project_id": 0, "project_name": "pr ```
Project Secrets analysis failed	project_secrets_analysis_failed	``` "project_id": 0, "project_name": "pr ```
Secrets ML user model training started	ml_model_user_secrets_training_run_started	``` Empty payload ```
Secrets ML user model training finished	ml_model_user_secrets_training_run_finished	``` Empty payload ```
Secrets ML user model training cancelled	ml_model_user_secrets_training_run_cancelled	``` Empty payload ```
Secrets ML model user training run failed	ml_model_user_secrets_training_run_failed	``` Empty payload ```
	ml_model_user_secrets_accepted	

Event name	Trigger value	Request payload struc
Secrets ML user model accepted		``` Empty payload ```
Secrets ML user model purged	ml_model_user_secrets_purged	``` Empty payload ```

Example of a simple Flask application

```
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'
```

#Working with API

Codescoring has open API, which allows you to programatically interact with the system. All the API commands are described in Swagger, which is available via the link **[platform-url]/api/swagger**.

#Beginning

To start working with Codescoring API, you need a token to authenticate requests. You can get a token in the `User profile` section by pressing the **Generate** button in the `API token` field.

To authenticate requests outside Swagger, it is necessary to prescribe a token in Header as follows:

`Authorization: Token <YOUR_TOKEN>`

#API structure

API provides a number of endpoints that allow you to perform basic operations in the system. Endpoints are grouped into sections corresponding to objects in the Codescoring system - dependencies, licenses, vulnerabilities, authors, etc.

#Pagination

For paginated response you can set page (for page number) and `per_page` (for per-page items amount) query parameters to the request.

The value of `per_page` parameter can be set with `CODESCORING_API_MAX_PAGE_SIZE` environment variable and defaults to 100.

Recommendation for the CODESCORING_API_MAX_PAGE_SIZE variable

Setting this value above 100 may negatively impact system performance.

Some endpoints work **only** in pagination mode and these parameters are mandatory, for the details see **[platform-url]/api/swagger**.

#Usage examples

- Run the analysis of all projects:

```
Curl -x 'Post' \
 '[platform_url]/API/Analysses/Overall_Sca/Start/' \
 -H 'Accept: Application/Json' \
 -H 'authorization: token <your_token>'
```

- Add policy:

```
Curl -x 'Post' \
 '[platform_URL]/API/POLICES/' \
 -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"
} '
```

- Get information about a separate project:

```
Curl -x 'get' \
 '[platform_url]/API/Projects/340/' \
 -H 'Accept: Application/Json' \
 -H 'authorization: token <your_token>'
```

- Get a list of available licenses:

```
Curl -x 'get' \
 '[platform_url]/API/License/' \
 -H 'Accept: Application/Json' \
 -H 'authorization: token <your_token>'
```

Note

The requests for creating and changing the main objects in the system, such as projects, are located in sections with the prefix **settings>**.

#Data Feeds

To effectively search for threats in open source components, CodeScoring integrates data from more than **20** knowledge sources (feeds). Records from all sources are deduplicated and combined under universal identifiers in the system.

Threat knowledge sources complement the unified **CodeScoring Index** database with information about published components, including its own protestware feed.

This section contains a detailed description of individual feeds and the process of working with them.

Below is a table with the data sources and their update frequency.

Source	Link	Update frequency
CodeScoring Protestware Feed	-	once a day
FSTEC of Russia (Federal Service for Technical and Export Control) Security Advisories	https://bdu.fstec.ru/	every 30 minutes
Kaspersky Open Source Software Data Feed	https://www.kaspersky.ru/open-source-feed	every 30 minutes
Astra Linux Security Advisories	https://astra.ru	every 4 hours
ALT Linux Security Tracker CVE	https://packages.altlinux.org	every 4 hours
Red OS Security Advisories	https://redos.red-soft.ru	once a day
CVE.org (MITRE CVE Program)	https://www.cve.org	once a day
National Vulnerability Database (NVD, NIST)	https://nvd.nist.gov/	every hour
Open Source Vulnerabilities (OSV)	https://osv.dev	every 20 minutes
GitHub Advisory Database	https://github.com/advisories	every 10 minutes
GitHub Security Repositories (More than 2000 repositories)	https://github.com	every hour
OpenSSF Malware Database (OSSF Malicious Packages)	https://openssf.org	every hour
Go Vulnerability Database (Golang)	https://vuln.go.dev	every 30 minutes

Source	Link	Update frequency
GitLab Security Advisory Database	https://advisories.gitlab.com	once a day
Packagist PHP Package Security Advisories	https://packagist.org	every 30 minutes
PyPI Security Advisories (PySec)	https://pypi.org	every 30 minutes
Ubuntu Security Notices	https://ubuntu.com/security/notices	every 30 minutes
Alpine Linux Security Advisories	https://alpinelinux.org	every 30 minutes
Debian Security Tracker	https://security-tracker.debian.org	every 30 minutes
Red Hat Security Advisories	https://access.redhat.com	every 30 minutes

#Working with protestware feed

Since April 2022, the CodeScoring team has been tracking cases of protestware inclusion in open source software components. In addition to open data, additional mechanisms for identifying such packages were first introduced in CodeScoring version **2022.49.0**.

Starting with version 2025.21.0, the platform supports its own centralized feed at the **CodeScoring Index** knowledge base level for detecting protestware. This feed is regularly updated and allows for automatic detection of such components within the CodeScoring.OSA and CodeScoring.SCA modules.

#What is protestware?

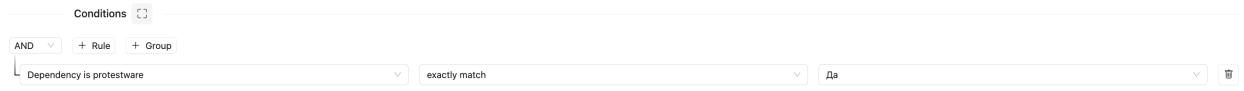
Protestware are components that contain compromising illegal constructs presented in the source code or accompanying data. Such software can change its behavior or be a prerequisite for the emergence of undeclared capabilities.

The Open Source Initiative considers protestware as a threat to the neutrality and reproducibility of open source software.

#Policy settings

CodeScoring has a built-in security policy to check for protestware in third-party components.

To activate it, go to the form in the `Settings -> Policies` section and specify the condition **Dependency is protestware**



If necessary, you can set the **Blocker** flag to make the policy blocking – when this condition is triggered, the software build and component download from the proxy repository will be interrupted.

Important

It is recommended to use the blocking flag only after a preliminary impact assessment (inventory analysis), as this may affect the development process.

#Analysis results

Protestware-related threats are marked with the **CSPW** identifier in the `Vulnerabilities` section. By going to the page of a separate entry, you can see detailed information about the component, the context of the threat, and the affected part of the code base.

If the corresponding security policy is configured, triggers on it are recorded in the `Alerts` section. The **Matched criteria** field displays the specific reason for the trigger, for example:

```
es5-ext@0.10.64 is protestware
```

#Using Kaspersky Open Source Software Threats Data Feed

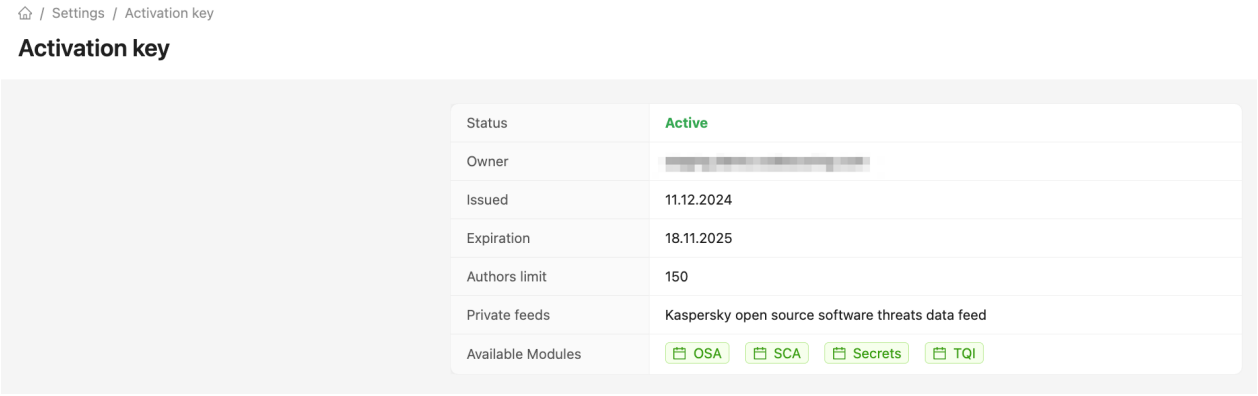
CodeScoring integrates the Kaspersky Open Source Software Threats Data Feed (Kaspersky OSSTDF) feed into the CodeScoring.SCA and CodeScoring.OSA modules, providing access to information about threats in detected open source components. The feed is useful not only as a source of data on new vulnerabilities, but also as a tool for enriching already known records from other sources.

The feed is integrated at the CodeScoring Index knowledge base level, which ensures deduplication of detected vulnerabilities and a single data presentation for the user.

#Connection check

The CodeScoring activation key contains data on the availability of private feeds.

If Kaspersky OSSTDF is successfully connected, the "Private feeds" field in the `Settings -> Activation Key` section has the value “Kaspersky open source software threats data feed”.



#Analysis results

CodeScoring integrates the feed not only from a data perspective, but also contextual extensions of functionality. In particular, with the connection of the feed, the **Impacts** field appears in the list of vulnerabilities and their individual pages, for which a separate security policy can be configured. The field value indicates the impact on the system that the threat has.

For example, the abbreviation **RLF** means Read Local Files - such a vulnerability allows access to read files on the user's device. This information helps to understand the context of the vulnerability and determine the direction of a potential attack.

The list of possible values of the variable:

- Arbitrary code execution (ACE);
- Code injection (CI);
- Denial of service (DoS);
- Loss of integrity (LoI);
- Overwriting arbitrary files (OAF);
- Obtaining sensitive information (OSI);
- Privilege escalation (PE);
- Reading local files (RLF);
- Security bypass (SB);
- Substitution of the user interface (SUI);
- Writing local files (WLF);
- Cross-site scripting (XSS/CSS).

In case of compromised packets, this field has the value `Other`.

For packets with malicious code, the following values are possible, among others:

- Malware;
- Hacktool.

More details about the available fields can be found in the OSSTDF documentation.

#Vulnerability details

Additional information appears on the vulnerability page obtained from the Kaspersky OSSTDF feed. The table with the main data contains the Kaspersky field, which can be used to determine the vulnerability ID, and the **Impacts (Kaspersky)** field.

#Setting up policies

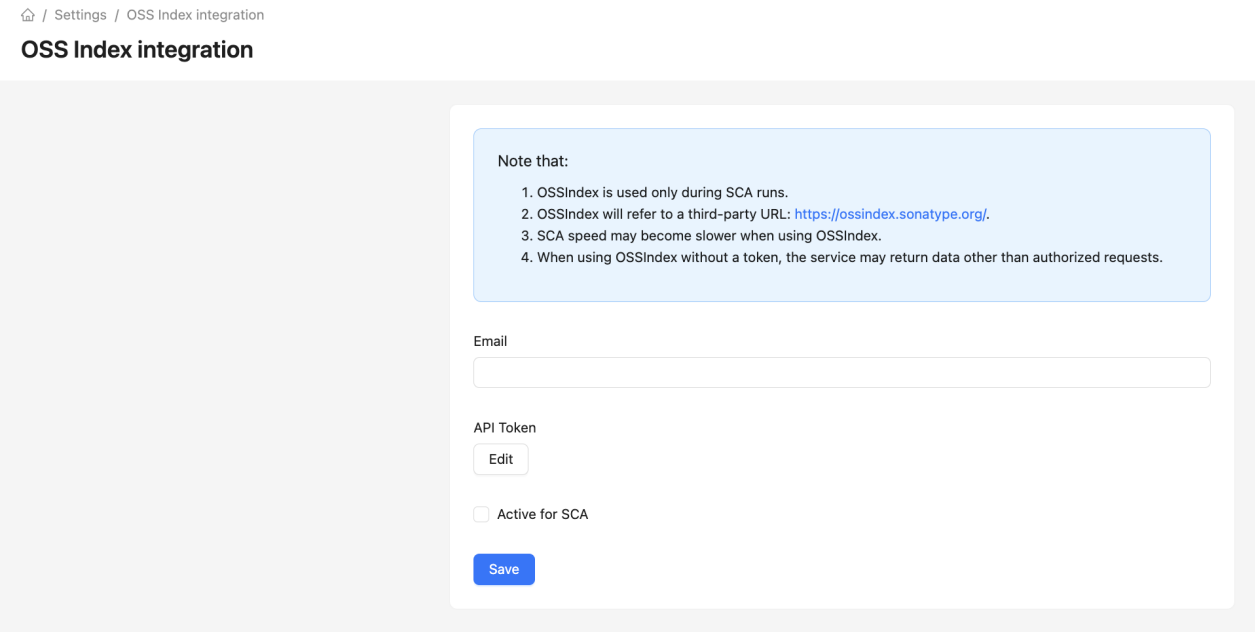
The data from the feed can be used to create security policies for both analyzing the current code base and checking downloaded components.

The "Impacts" field allows to precisely specify the types of vulnerabilities that require attention. For example, to set up a policy that blocks the use of components with malicious code, one needs to create the condition `Vulnerability mpacts (Kaspersky) -> exactly match -> Malware`.

#Integrating OSS Index

In addition to the internal **CodeScoring Index** database, you can also connect a third-party feed Sonatype OSS Index for advanced analysis.

Integration is carried out in the `Settings -> OSS Index` section. To connect, you must fill in the Email and API token received during user registration in Sonatype OSS Index.



Data quality disclaimer

The OSS Index integration uses a third-party Sonatype OSS Index feed. Data from this source may include false positives and inaccuracies, so it is recommended to use it as an additional source and separately validate results before making blocking decisions.

Important:

1. OSS Index is used only during SCA startup.
2. OSS Index will refer to a third-party URL: https://ossindex.sonatype.org/.
3. SCA speed may decrease when using OSS Index.
4. When using OSS Index without a token, the system may return data different from authorized requests.

#CodeScoring.Save

#General Description

CodeScoring.Save is an artifact repository manager for Maven, npm, NuGet, PyPI, Go, Docker / OCI, and raw files. The service is deployed in Kubernetes and integrates with other CodeScoring platform modules.

The service covers three main tasks:

- **Proxy and cache** for external artifacts with security policy checks when downloads are routed through OSA Proxy.
- **Hosted storage** for internal libraries, images, and arbitrary organization files.
- **Unified access control** for artifacts through the dedicated authentication and authorization service `cs-auth`.

#Supported Formats

Maven, npm, NuGet, PyPI, Go, Docker / OCI, raw. Module capabilities are described on the Functional characteristics page.

#Security

Access control is handled by the dedicated authentication and authorization service `cs-auth`.

#Additional Sections

- Functional characteristics — platform and module capabilities.
- CodeScoring.Save installation and maintenance — architecture, requirements, deployment, update, and SSL/TLS setup.
- Repository management — repository usage, authentication, and cleanup policies.
- Integration with CodeScoring.OSA — configuring security checks through OSA Proxy.

#Repository and Artifact Management

#Context

CodeScoring.Save stores and serves artifacts for a development team: a builder publishes a package, and IDEs, CI agents, and engineers download it from the same URL — without leaving the company perimeter.

All artifacts live inside **repositories**, and repositories are grouped into **projects**. The same hierarchy is reflected in the web interface:

```
›
Project(e.g. backend-team)
  ›
  Repository(e.g. maven-central)
    ›
    Artifact(e.g. commons-lang3:3.12.0)
```

A project is a container for the team's repositories. A repository is a concrete storage of one format (Maven, npm, Docker, etc.) and one of two types. An artifact is what lives inside a repository and what clients pull.

#Two Repository Types

CodeScoring.Save supports only two repository types, and each addresses a different need.

Proxy repository — a caching proxy in front of an external source (for example, Maven Central or npmjs.com). When a client requests an artifact for the first time, Save fetches it from upstream, caches it, and serves it. All

subsequent requests are served from the cache — this speeds up builds and reduces dependency on external services. Proxy repositories also benefit from OSA Proxy security policies (when configured), which can block downloads of unsafe components.

Hosted repository — internal storage where the team publishes its own artifacts. This is where internal libraries, vetted third-party components, and build artifacts go.

A single project may contain any number of repositories of either type.

#URL Access Scheme

An external client (Maven, npm, Docker, etc.) reaches a repository through URLs of the form:

```
https://save.example.com/<format>/<project>/<repository>/
<artifact-path>
```

Format	Template
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>` (used as GOPROXY)
Raw	`https://save.example.com/raw/<project>/<repository>/<filepath>`
OCI / Docker	`https://save.example.com/v2/<project>/<repository>/<image>/...`

For OCI / Docker, the standard `/v2/` prefix is used as required by the OCI Distribution Spec. Nexus-compatible flat URL routing is also supported — see Working with OCI / Docker.

#Web Interface Layout

The left-hand vertical navigation has the following sections:

- **Projects** — the main section: a list of projects, with a list of repositories inside each project, and a tree of artifacts inside each repository;
- **Cleanup** — automatic cleanup policies for unwanted artifacts;
- **Settings** — accounts, roles, service accounts, configuration, and the audit log.

At the top of the interface is a **global search** — opened with the `/` key, it searches across projects, repositories, and artifacts simultaneously, with real-time suggestions.

#Basic Scenario: Connect Your First Repository

#Step 1. Create a Project

A project is a container for the repositories that will live inside it. It is convenient to split projects by team, product, or environment.

1. In the sidebar, choose **Projects**.
2. Click **Create project** in the top-right corner.
3. Fill in the form:
    - **Name** — a human-readable project name (for example, `Backend Team`);
    - **Color** — a color that helps tell projects apart in lists;
    - **Key** — a URL-safe project identifier (for example, `backend-team`). If left empty, it is auto-generated from Name, even when Name does not use Latin characters. **Key cannot be changed after creation**;
    - **Description** — an optional project description;
    - **Cleanup policy** — a two-pane selector listing the existing cleanup policies. It can be left empty and assigned later.
4. Click **Create project**.

After creation, the project page opens with an (empty) list of repositories and the project's metadata in the header.

Key vs. Name

Name is shown in the UI and can be changed at any time. Key participates in every URL and API path, so changing it would require reconfiguring every client.

#Step 2. Create a Repository in the Project

A repository — concrete storage for artifacts of a single format — is created inside a project.

1. Open the project you just created.
2. Click **Create repository** in the top-right corner of the project page.

3. Fill in the common fields:
    - **Name** — a human-readable name (for example, `Maven Central (proxy)`);
    - **Color** — a color for visual distinction;
    - **Key** — a URL-safe identifier (for example, `maven-central`). It cannot be changed after creation;
    - **Description** — an optional description.
4. Choose **Format** — one of the supported package formats: `maven`, `npm`, `docker` (OCI), `nuget`, `pypi`, `go`, `raw`. The format cannot be changed after creation.
5. Choose **Type**:
    - **Proxy** — to proxy an external registry;
    - **Hosted** — to host your own artifacts.
6. If **Proxy** is selected, additional fields appear:
    - **Proxy URL** — the upstream repository URL (for example, `https://repo1.maven.org/maven2/`);
    - **Cache TTL, seconds** — the metadata cache lifetime in seconds. For typical external registries, `86400` (one day) is enough.
7. Optionally attach **Cleanup policy** — a list of cleanup policies that should run against this repository.
8. Click **Create repository**.

After creation, the repository page opens. The header shows the status (Enabled/Disabled), the format, the type, the creation and update timestamps, and — for a proxy repository — a clickable **Remote URL** link to the upstream.

Changing the repository status

The status (Enabled/Disabled) is set on the edit form: open the repository, click **Edit repository** in the header, flip the top **Status** radio toggle, and click **Save**. A disabled repository does not accept or serve requests, but it is not deleted and keeps its artifacts.

#Step 3. Get Ready-Made Snippets for Connecting Clients

In the right part of the repository header there is a chain-link icon button. Clicking it opens the **Useful snippets** popover — a set of ready-made configuration fragments and commands tailored to the repository format.

1. Click the chain-link button in the repository header.
2. In the card that appears, scroll through the snippets:
    - each snippet has a **title** (for example, `settings.xml`, `.npmrc`, `pip.conf`, `docker login`), a syntax-highlighted code block, and a short description;
    - the set of snippets is composed server-side based on the repository's format and type. For Maven, you typically see a `<mirror>` fragment for `settings.xml` and a `<repository>` fragment for `pom.xml`; for Docker — `docker login` and `docker pull`; for npm — a line for `.npmrc`; and so on.
3. Click **Copy** to the right of the snippet you need — it goes straight to the clipboard.

4. Paste the snippet into the client's configuration file, or run the command in a terminal.

#Step 4. Publish and Download the First Artifact

From this point on, everything happens on the client side — Save accepts the standard requests from each package manager. For a hosted repository, the usual path is publishing via `mvn deploy`, `npm publish`, `twine upload`, or `docker push`. For a proxy repository, an ordinary pull is enough — Save will cache the artifact on the way through.

The first request may require authentication — see Connecting Clients and Authentication.

Once a client uploads or downloads something for the first time, the artifact appears in the tree on the repository page.

#Working with Artifacts in a Repository

The repository page has two columns: the artifact tree with search and sort on the left, the details of the selected artifact on the right.

#Browsing the Tree

The tree shows the structure of the repository — folders and files with icons. Files with a "lock" icon are protected (locked/release) artifacts that cannot be deleted or overwritten without extra steps. Folders are expanded lazily: children are fetched on click.

Clicking a file opens its details in the right pane: name, size, path, checksum, and format-specific metadata (Maven `groupId`/`artifactId`/`version`, Docker tags, and so on).

#Search, Sort, and Filters

Above the tree is the **Search artifacts** field: type a substring and press Enter — a flat list of matching artifacts appears with context highlighting. Clicking a row opens the details of the matching artifact.

Two buttons sit next to the search field.

The **Sort** button controls how the tree is ordered:

- **Sort by** — `Name`, `Date added`, or `Date modified`;
- **Sort direction** — `A to Z` or `Z to A`.

The filter button (funnel icon), available for proxy repositories, exposes a **Show uncached artifacts** toggle. By default the tree shows only what is already in the local cache. With the toggle enabled, Save also lists artifacts known to upstream but not yet cached — useful for browsing what's available in the proxied source without triggering a real pull. *This is **not** available for every repository format*, because the underlying protocols differ.

#Actions on a Single Artifact

After selecting an artifact, the right pane offers:

- **Download** — saves the file locally;
- **Lock artifact** — marks the artifact as `release`. Once locked, the artifact cannot be deleted or overwritten by a normal `upload` — this protects released versions from being silently replaced;
- **Delete artifact** — removes the file from the repository (for hosted) or from the cache (for proxy).

Lock is irreversible in the UI

The release flag can only be cleared through the API:
`PUT /api/v1/artifacts/release?id=<artifact-id>` with `{"is_release": false}`.

#Bulk Actions

Each tree row has a checkbox on its left. Selecting several files reveals an action bar at the top:

- **Lock** — locks the selected artifacts in bulk;
- **Delete** — deletes them in bulk.

Both operations are atomic for the entire selection.

#Connecting Clients and Authentication

Save supports several authentication methods to cover both interactive clients and CI pipelines. The client chooses the method itself through the `Authorization` header (or a format-specific header).

Method	Header	cs-auth type	When to use
Basic Auth (user)	`Authorization: Basic base64(<username>:<password>)`	`basic`	Interactive work from IDEs, manual `curl`, `mvn`/`pip`/`npm` runs as a specific user
Basic Auth (robot account)	`Authorization: Basic base64(sa$<robot-name>:<api-key>)`	`api_key`	**CI/CD pipelines**, build runners, and any service integration
Bearer JWT	`Authorization: Bearer <jwt>`	`bearer_jwt`	Session requests from the web interface and other services

Method	Header	cs-auth type	When to use
			after login via cs-auth
Bearer (opaque token)	`Authorization: Bearer <opaque-token>`	`npm_token`	Any non-JWT bearer token; in practice, the npm client after `npm adduser` / `npm login`
`X-NuGet-ApiKey`	`X-NuGet-ApiKey: <api-key>`	`nuget_key`	NuGet format only, for compatibility with `dotnet nuget push`
Docker v2 token	`Authorization: Bearer <docker-jwt>`	`docker_token`	OCI clients after going through the `401 → retry → 200` cycle

A Bearer token with three dot-separated segments is classified as a JWT; any other Bearer token is treated as an opaque npm token. The `sa$` prefix in Basic Auth marks a service account (robot).

#Robot Accounts for CI/CD

Personal passwords are not recommended for CI/CD integrations — they're tied to an employee and rotate often. The right tool is a **robot account**: a service account with its own role and a long-lived API key.

#Creating Through the Web Interface

1. In the sidebar, open `Settings -> Robot accounts`.
2. Click **Create robot account** in the top-right corner.
3. Fill in the fields:
    - **Login** — the service name used for authentication (for example, `ci-builder`). Save automatically adds the service-account prefix to the value you enter;
    - **Name** — a human-readable name (for example, `CI Builder Bot`);
    - **Description** — an optional description;
    - **Expires in** — the date when the key expires. Quick presets `Week`, `Month`, and `Year` are available below the field. To make the key permanent, tick the **Never** checkbox to the right of the date picker — the date input is then disabled automatically;
    - **Global permissions** — the robot's global permissions;
    - **Scoped permissions** — permissions limited to one, several, or all projects or repositories. The **Scope** switch selects the level

(Projects / Repositories), and each scope adds entries through `Add project permissions` / `Add repository permissions`.

4. Click **Create robot account**.

When the robot is created successfully, a modal window with the API key opens automatically — this is the password used for Basic Auth.

#The API Key Window

After clicking **Create robot account**, Save opens the API key window. The modal shows:

- **API key** — a read-only field with the key itself and a **Copy** button that puts it on the clipboard;
- **Key prefix** — a short prefix of the key. It lets you identify the key in the audit log without revealing the full secret;
- **Expires at** — the expiration date (if one was set);
- **Warning** — server-supplied text, for example `Store this API key securely — it cannot be retrieved again.`

Below the secret is an **I have copied the key** button that closes the window and takes you to the newly created robot account's detail page.

The API key is shown only once

After the window closes, the same key cannot be viewed or retrieved again — Save stores only its hash. Copy the key and save it into your CI secret storage (GitLab CI variables, GitHub Actions secrets, HashiCorp Vault) before closing the window. If the key is lost, rotate it (either through the robot account's edit form in the UI, or via `POST /admin/robots/<id>/rotate-key`) — once rotated, the old value becomes invalid.

#Creating Through the 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
  }'
```

The response returns `api_key`, which is exposed **only once** at creation time — save it in your CI secret storage.

#Client Usage

`username = sa$<robot-name>`, `password = <api-key>` — standard Basic Auth:

```
curl -u "sa\$ci-builder:<api-key>" https://save.example.com/...
```

Escaping

$ in shell In bash, $ inside single quotes is preserved as-is; inside double quotes it must be escaped as \$. No escaping is needed in YAML / TOML / XML configuration files.

NuGet API key — implementation detail

The `X-NuGet-ApiKey` header is supported directly: cs-auth classifies it as `nuget_key` and validates it by the 12-character key prefix. Both options work for NuGet, but we recommend Basic Auth with a robot account — it is consistent with the other formats, and the robot's identity is recorded explicitly in the audit log.

#Anonymous Read Access

When the global `AllowAnonymousRead` flag is enabled, unauthenticated requests receive a `PermissionSet` granting `projects.view`, `project_repos.view`, `project_artifacts.view`, and `project_artifacts.download` across all projects. For Docker / OCI this mode still requires the `401 → /v2/token → 200` cycle — otherwise the Docker client cannot work (see Working with OCI / Docker).

The current value of the flag is visible under `Settings -> Configuration`. Changing it is done through the 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"}'
```

#Cleanup Policies

To keep storage from growing without bound, repositories can have **cleanup policies** attached — rules for automatic deletion (or retention) of artifacts. Save supports five policy types; simple ones are enough for most cases, while complex AND/OR conditions use the `expression` type.

#Creating a Policy

1. In the sidebar, open **Cleanup**.
2. Click **Create policy**.
3. Fill in the common fields: **Display name**, **Description**, and optionally **Schedule** (a five-field cron expression: minutes, hours, day, month, weekday).
4. Choose **Policy type** — this defines the main cleanup rule:

Policy type	Purpose	`value` parameter
`delete-snapshots-older-than`	Delete SNAPSHOT/dev versions older than N days	number of days, for example `30`
`keep-latest-versions`	Keep only the N latest versions of each artifact, delete the rest	N, for example 5
`delete-by-age`	Delete any artifact older than N days	number of days
`delete-by-size`	Delete the oldest artifacts once a size limit is exceeded	size in gigabytes as an integer, for example `50`
`expression`	Flexible DSL rule combining criteria with AND/OR	an `expression` object is passed instead of `value`

5. Specify the **Formats** the policy applies to. Leaving this empty means it applies to every format.
6. Tick the **Active** flag for the policy to start working immediately.
7. For the `expression` type, build the criteria tree as well — each criterion combines a `type`, a comparison operator (`eq`, `ne`, `gt`, `lt`, `matches`, `contains`), and a value; criteria can be grouped with logical operators `AND` or `OR`.
8. Click **Create policy**.

#Available Criterion Types for `expression`

Full list: `GET /api/v1/enums/cleanup-criterion-types`.

Group	`type`	Purpose
Age	`created_before`	Artifact age (in days)
	`last_downloaded_days`	Days since the last download

Group	type	Purpose
	`not_downloaded_since`	Never downloaded, or not downloaded since the given date
Version	`version_pattern`	Glob pattern matched against the version
	`is_snapshot`, `is_prerelease`, `is_release`	Snapshot / pre-release / release markers
Name/path	`name_pattern`, `path_pattern`	Glob patterns
Size	`size_greater_than`, `size_less_than`	Artifact size
Docker	`docker_tag`, `docker_untagged`	Tag / no tag
Maven	`maven_classifier`, `maven_packaging`	Classifier / packaging
npm	`npm_scope`	Package scope
PyPI	`pypi_package_type`	`sdist`, `bdist_wheel`, ...
NuGet	`nuget_is_prerelease`	NuGet pre-release flag
OCI	`oci_artifact_type`	OCI artifact type (Helm, Cosign, SBOM, ...)
Retention	`keep_latest`	Keep N latest versions

#Assigning a Policy to a Repository

The assignment happens on the create or edit form of a repository (or of a project — in which case the policy cascades to every repository inside the project).

1. Open the repository and click **Edit** in the header.
2. Scroll down to the **Cleanup policy** section.
3. The left column (**Available**) lists every existing policy; the right column (**Applied**) lists the ones already attached. Move the policies you need between columns using the arrow.
4. Save the changes.

#Manual Run and Preview via API

```
## Dry run — see what would be deleted, without saving the
policy
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"
  }'
```

```
## Run saved policies (with an optional 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
  }'
```

#Access Management

Users, roles, and access rights for projects and repositories are managed by a separate service, **cs-auth**. Its API is proxied through Save under `/api/v1/auth/*` and `/api/v1/admin/*`. The authentication methods are described above in Connecting Clients and Authentication.

In the web interface, the relevant sections live under `Settings`:

- **Roles** — role definitions with their permission sets;
- **Users** — human accounts;
- **Robot accounts** — service accounts (see above).

cs-auth API

The exact endpoints for managing users, roles, and project membership are part of the cs-auth API surface. Save forwards them transparently but does not describe them in its own Swagger.

#Access Levels

Permissions in CodeScoring.Save are granular and follow the `<resource>.<action>` format. Roles in cs-auth are user-defined and consist of such permissions.

Repository level (artifacts inside a repository):

Permission	Allows
`artifacts.view`	Viewing the artifact list and metadata
`artifacts.download`	Downloading artifacts
`artifacts.upload`	Publishing artifacts (relevant for hosted)
`artifacts.delete`	Deleting artifacts
`artifacts.lock`	Lock/unlock one or all artifacts

Repository level (the repository itself):

Permission	Allows
`repos.view`	Viewing repository settings and statistics
`repos.edit`	Changing repository settings
`repos.delete`	Deleting the repository

Project level (cascade permissions that apply to every repository and artifact in the project):

Permission	Allows
`projects.view`, `projects.create`, `projects.edit`, `projects.delete`	Managing projects
`project_repos.view`, `project_repos.create`, `project_repos.delete`, `project_repos.edit`	Managing all repositories in a project
`project_artifacts.view`, `project_artifacts.upload`, `project_artifacts.download`, `project_artifacts.delete`, `project_artifacts.lock`, `project_artifacts.unlock`	All artifact operations in any repository of the project

Global (only available through a global-scope role):

Permission	Allows
`cleanup.view`, `cleanup.edit`, `cleanup.delete`	Managing cleanup policies
`roles.create`, `roles.delete`	Managing roles
`users.create`, `users.delete`	Managing users
`config.view`, `config.manage`	Viewing and editing configuration
`audit.view`	Viewing the audit log

Cascade logic

Project-scope permissions automatically cascade to the repository scope: a user with `project_artifacts.download` on a project gets `artifacts.download` on every repository inside that project. The * key grants wildcard access to all projects / repositories.

#Assigning a User or Robot Account to a Project

Membership ties a user or a robot account to a project with a specific role. The assigned member receives every permission of the role and shows up as a member in the project header.

#Through the Web Interface

Section in progress

The **Members** section on the project page is still being built. The backend supports every necessary operation (`/api/v1/admin/projects/<project>/members`), but the **Members** tab is not yet exposed in the project UI.

Until the tab is available, use the API operations below.

#Through the API

Adding a member:

```
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
  }'
```

For human users `role_id` is required; for robot accounts (`is_service=true`) it can be omitted.

Listing members:

```
curl -u "<admin-username>:<admin-password>" \
  "https://save.example.com/api/v1/admin/projects/backend-team/
members?limit=50"
```

Changing the role of an existing member:

```
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}'
```

Removing a member from the project:

```
curl -X DELETE https://save.example.com/api/v1/admin/projects/
backend-team/members/5 \
  -H "Authorization: Bearer <admin-jwt>"
```

#The Same Actions via API

Every action covered above for the web interface is also available through the REST API. This is useful for bootstrap scripts, infrastructure-as-code (Terraform / Ansible), and integration tests.

#Creating a Project

```
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"
  }'
```

#Creating a Proxy Repository

```
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
  }'
```

#Creating a Hosted Repository

```
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"
  }'
```

#Assigning a Cleanup Policy

```
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}
    ]
  }'
```

#Creating a Cleanup Policy

```
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": "Delete Maven SNAPSHOT versions older than
30 days",
    "policy_type": "delete-snapshots-older-than",
    "value": "30",
    "formats": ["maven"],
    "enabled": true,
    "schedule": "0 2 * * *"
  }'
```

#Creating an Expression-Based Policy

```
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
  }'
```

#Monitoring

#Audit Log

The audit log records every change to projects, repositories, artifacts, roles, and robot accounts. In the UI it is available:

- **Globally** — under `Settings -> Audit log`;
- **Per project** — through the journal icon on the right of the project header, which opens the audit log filtered to that project.

Each entry contains the event time, the initiator, the resource type and identifier, the action type, and a set of fields with details (the name of the uploaded artifact, changed repository settings, and so on). Entries are grouped by day and displayed as a timeline, and the links inside an entry are clickable (they open the corresponding project, repository, policy, user, or role).

The log can be exported for a specific period through the **Export** button in the header (formats: `JSON` or `HTML`).

```
## Audit log for a specific repository via API
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=maven-central&limit=50"
```

#Repository Statistics

Current aggregated repository metrics are available via the API:

```
curl https://save.example.com/api/v1/repos/<repository_id>/
stats \
  -u "<username>:<password>"
```

Response:

```
{
  "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"
}
```

Statistics fields

The endpoint returns aggregated values only. The `cache_hit_ratio` field is present only for proxy repositories.

#Service Status

```
curl https://save.example.com/health
```

A global service health check. There are no per-repository health endpoints: upstream availability is checked by background workers and exposed through Prometheus metrics.

#Logs

Logs are centralized and emitted as JSON. Filtering by a specific repository is done through standard log-aggregator tools.

#Working with Go

CodeScoring.Save implements **Go Module Proxy Protocol** with the `/go/<project>/<repository>/` prefix. It is compatible with the standard Go toolchain (`go mod`, `go build`, `go install`).

#Proxy Repository

```
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 Repository

```
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"
  }'
```

A hosted repository accepts PUT uploads of modules at URLs like:

```
PUT /go/<project>/go-hosted/<module>/@v/<version>.zip
```

Module paths use case encoding according to the Go Module Proxy spec: uppercase letters are replaced with `!` + the corresponding lowercase letter, for example `Acme` -> `!acme`.

#Client Configuration

```
## Set GOPROXY
export GOPROXY="https://save.example.com/go/<project>/go-proxy"

## If Basic Auth is used, credentials are passed
through .netrc:
```

```
cat >> ~/.netrc << EOF
machine save.example.com
login <username>
password <password>
EOF
chmod 600 ~/.netrc
```

Proxy chain

GOPROXY can contain several URLs separated by commas: `GOPROXY="https://save.example.com/go/<project>/go-proxy,direct"`. The Go toolchain tries each proxy in order; `direct` means direct access to VCS.

Robot accounts in CI

For CI/CD, use a robot account: `login = sa$<robot-name>`, `password = <api-key>` in `~/.netrc`. For details, see Authentication.

GOPRIVATE for internal modules

If the repository stores internal modules that are not present in public sumdb, add their patterns to `GOPRIVATE`.

```
export GOPRIVATE="git.example.com/*"
## If needed, sumdb can be disabled completely:
## export GOSUMDB=off
```

#Repository URL Migration

Use case: migrating a Go proxy from Nexus / Artifactory to CodeScoring.Save.

Source	GOPROXY before migration	GOPROXY after migration
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`
Official proxy	`https://proxy.golang.org`	`https://save.example.com/go/<project>/go-proxy`

#Troubleshooting

#Checking Version List

```
curl -u "<username>:<password>" \
  https://save.example.com/go/<project>/go-proxy/github.com/
gin-gonic/gin/@v/list
```

#Service Status

```
curl https://save.example.com/health
```

#Repository Audit

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=go-proxy&limit=50"
```

#Working with Maven

CodeScoring.Save supports Maven 2 layout and is compatible with JVM tools that use it: `mvn`, `gradle`, `ant + Ivy` / `maven-ant-tasks`, sbt, and others.

#Proxy Repository

```
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 Repository

```
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"
  }'
```

#Client Configuration

#mvn

In `~/.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>
```

To publish artifacts (deploy to hosted), add this to `pom.xml`:

```
<distributionManagement>
  <repository>
    <id>save-maven</id>
    <url>https://save.example.com/maven/<project>/internal-
maven/</url>
  </repository>
</distributionManagement>
```

```
mvn deploy
```

#gradle

In `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')
            }
        }
    }
}
```

Or in 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 can be stored in `~/.gradle/gradle.properties`:

```
saveUser=<username>
savePassword=<password>
```

Robot accounts in CI

For CI/CD, use a robot account: `username = sa$<robot-name>`, `password = <api-key>`. The structure of `settings.xml` and `gradle.properties` remains the same; only values change. For details, see Authentication.

#ant

Option 1: through `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>
```

Option 2: through Apache Ivy with `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

In `~/.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", "")
)
```

#Repository URL Migration

Use case: migrating a Maven repository from Nexus / Artifactory to CodeScoring.Save.

Source	URL in settings.xml before migration	URL in settings.xml after migration
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/`
Official repository	`https://repo.maven.apache.org/maven2`	`https://save.example.com/maven/<project>/maven-central-proxy/`

Authentication parameters (username, password) and the `<servers>` block are transferred unchanged.

#Troubleshooting

#Checking Package Metadata

```
curl -u "<username>:<password>" \
  https://save.example.com/maven/<project>/maven-central-proxy/
org/apache/commons/commons-lang3/maven-metadata.xml
```

#Service Status

```
curl https://save.example.com/health
```

#Repository Audit

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=maven-central-proxy&limit=50"
```

#Working with NPM

CodeScoring.Save implements npm Registry API with the `/npm/<project>/<repository>/` prefix. It is compatible with standard `npm`, `yarn`, and `pnpm` clients.

#Proxy Repository

```
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 Repository

```
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"
  }'
```

#Client Configuration

#npm

```
## Set registry in the current project
echo "registry=https://save.example.com/npm/<project>/npmjs-
proxy/" > .npmrc
```

```
## Authentication (npm 7+, _auth is 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

## Install dependency
npm install lodash
```

Robot accounts in CI

For CI/CD, use a robot account: `username = sa$<robot-name>`, `password = <api-key>`. The `.npmrc` structure remains the same; only `_auth` changes (base64 of `sa$<robot-name>:<api-key>`). For details, see Authentication.

Opaque bearer tokens

After `npm adduser` / `npm login`, the npm client stores the received opaque token and then uses it as `Authorization: Bearer <token>`. cs-auth classifies any non-JWT bearer token (a Bearer value without three dot-separated segments) as `npm_token` and validates it through service accounts linked to API keys.

Publishing to a hosted repository:

```
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 uses the same `.npmrc` as npm:

```
echo "registry=https://save.example.com/npm/<project>/npmjs-
proxy/" > .npmrc
yarn install
```

#Yarn 2+ (Berry)

In `.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 reads the same `.npmrc`:

```
echo "registry=https://save.example.com/npm/<project>/npmjs-
proxy/" > .npmrc
pnpm install
```

For scoped packages, configure a separate registry:

```
echo "@mycompany:registry=https://save.example.com/npm/
<project>/npm-hosted/" >> .npmrc
```

#Repository URL Migration

Use case: migrating an npm repository from Nexus / Artifactory to CodeScoring.Save.

Source	`.npmrc` `registry=` before migration	`.npmrc` `registry=` after migration
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/`
Official repository	`https://registry.npmjs.org`	`https://save.example.com/npm/<project>/npmjs-proxy/`

During migration, it is enough to replace the URL in `.npmrc`/`.yarnrc.yml`. Existing Nexus/Artifactory credentials can be reused as Basic Auth.

#Troubleshooting

#Checking Package Metadata

```
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
```

#Service Status

```
curl https://save.example.com/health
```

#Repository Audit

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=npmjs-proxy&limit=50"
```

#Working with NuGet

CodeScoring.Save implements **NuGet v3 API** with the `/nuget/<project>/<repository>/` prefix. It is compatible with standard `dotnet`, `nuget.exe`, Visual Studio, and Rider clients.

#Proxy Repository

```
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 for remote_url

Specify the full service index URL. For official nuget.org, it is `https://api.nuget.org/v3/index.json`. Save resolves nested resources (flatcontainer, registration, search) using this index.

#Hosted Repository

```
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"
  }'
```

After creation, the service index is available at `https://save.example.com/nuget/<project>/<repository>/v3/index.json` .

#Client Configuration

#dotnet CLI

In `nuget.config` or `NuGet.Config` (file name depends on the OS):

```
<?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>
```

Usage:

```
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
```

#Publishing (hosted)

Push packages to a hosted repository through standard clients:

```
## 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 for CI

`dotnet nuget push` supports either an API key (`X-NuGet-ApiKey` header) or Basic Auth. CodeScoring.Save supports both methods: `X-NuGet-ApiKey` is classified as `nuget_key`. Still, Basic Auth with a robot account is recommended for CI/CD. It is consistent with other formats and explicitly records the robot identity in the audit log.

#Repository URL Migration

Use case: migrating a NuGet repository from Nexus / Artifactory to CodeScoring.Save.

Source	URL in NuGet.Config before migration	URL in NuGet.Config after migration
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
Official repository	https://api.nuget.org/v3/index.json	https://save.example.com/nuget/<project>/nuget-proxy/v3/index.json

During migration, <packageSourceCredentials> remains unchanged.

#Troubleshooting

#Checking the Service Index

```
curl -u "<username>:<password>" \
  https://save.example.com/nuget/<project>/nuget-proxy/v3/
index.json | jq .
```

The response must contain a resources array with PackageBaseAddress, RegistrationsBaseUrl, SearchQueryService, and other resources.

#Package Version List

```
curl -u "<username>:<password>" \
  https://save.example.com/nuget/<project>/nuget-proxy/v3-
flatcontainer/newtonsoft.json/index.json
```

#Service Status

```
curl https://save.example.com/health
```

#Repository Audit

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=nuget-proxy&limit=50"
```

#Working with OCI / Docker

CodeScoring.Save implements **OCI Distribution Spec** under the `/v2/` prefix and supports Docker, Helm OCI charts, and other OCI artifacts such as `oras`.

#Proxy Repository

```
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
  }'
```

For single-name images without slashes, standard Docker Hub normalization `library/<image>` is applied.

#Hosted Repository

```
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 Scheme

The base scheme is path-based, as required by Docker / OCI:

```
<host>/<project>/<repository>/<image>:<tag>
```

Nexus-compatible flat URL routing (flat-alias) is also supported. For compatible repositories, the `<project>/<repository>` prefix is replaced with a short alias or omitted entirely.

#Client Configuration

#Docker / Podman

```
## Login
docker login save.example.com

## Pull through proxy
docker pull save.example.com/common/dockerhub-proxy/library/
nginx:latest

## Push to 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 and

`docker login` Endpoint `/v2/` always returns `401` with a `WWW-Authenticate: Bearer realm=...` challenge header, even if the repository allows anonymous read. This is required so the Docker client correctly performs the `401 -> retry -> 200` cycle and then sends Basic Auth if needed. This is an OCI Distribution Spec requirement.

Robot accounts in CI

For CI/CD, use a robot account:
`docker login -u 'sa$<robot-name>' -p '<api-key>' save.example.com`.
The same approach works for `helm registry login` and `oras login`. For details, see Authentication.

#Helm

CodeScoring.Save accepts Helm charts as regular OCI artifacts. Push/pull is performed with standard `helm` commands:

```
## Login (Helm 3.8+)
helm registry login save.example.com -u <username>

## Push
helm package ./mychart
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 for test stands

For test stands without TLS, use `--plain-http` (Helm 3.13+):

```
helm push mychart-0.1.0.tgz oci://save.example.com/common/
docker-hosted --plain-http
```

#oras

`oras` works with any OCI-compatible artifact: configs, policies, SBOMs, binaries, and others.

```
## Push an arbitrary file as an OCI artifact
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
```

#Flat URL Routing

Save supports **Nexus-compatible flat namespace** for migration from Nexus / Artifactory: a repository can be configured with a prefix alias so that it responds to requests without `<project>/<repository>` in the path.

Longest-prefix matching is used:

- `docker pull save.example.com/common/external/golang:1.24.1` goes to the repository with `flat_alias = common/external/`.
- `docker pull save.example.com/johnny-depp:v1` goes to a catch-all hosted repository if it is configured.

When creating a repository with a flat alias, use `flat_alias_prefix` and/or `is_catch_all`. Prefixes are validated for uniqueness.

#Repository URL Migration

Use case: migrating a Docker registry from Nexus / Artifactory to CodeScoring.Save.

Source	URL before migration	URL after migration (with project)	URL after migration (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`

The flat-alias option does not require client-side changes: existing scripts with a hardcoded `nexus.host.ru/library/nginx` path start working after the host is replaced.

#Troubleshooting

#Checking `/v2/`

```
curl -i https://save.example.com/v2/
## Expected: HTTP/1.1 401 Unauthorized with Www-Authenticate:
Bearer realm=...
```

#Checking the Token Endpoint

```
curl -u "<username>:<password>" \
  "https://save.example.com/v2/token?
service=save.example.com&scope=repository:common/dockerhub-
proxy/library/nginx:pull"
```

The response is JSON with the `token` field containing a short-lived Docker v2 JWT.

#Listing Tags

```
curl -u "<username>:<password>" \
  https://save.example.com/v2/<project>/<repository>/<image>/
tags/list
```

#Service Status

```
curl https://save.example.com/health
```

#Repository Audit

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?
resource_type=repository&q=dockerhub-proxy&limit=50"
```

#Working with PyPI

#Proxy Repository

```
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 Repository

```
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"
  }'
```

After creation, the repository is available at `https://save.example.com/pypi/<project>/<repository>/`. The simple index is available under `/simple/`; package upload (twine-compatible POST) uses the repository root.

#Client Configuration

#pip

```
## One-time installation through --index-url
pip install --index-url https://save.example.com/pypi/
<project>/pypi-proxy/simple/ <package>

## Persistent configuration through 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
```

pip.conf location

- Linux/macOS: `~/.config/pip/pip.conf` or `~/.pip/pip.conf`
- Windows: `%APPDATA%\pip\pip.ini`
- In virtualenv: `$VIRTUAL_ENV/pip.conf`

Robot accounts in CI

For CI/CD, use a robot account: `username = sa$<robot-name>`, `password = <api-key>`. The `pip.conf` structure remains the same; only `username` changes. For details, see Authentication.

#poetry

In `pyproject.toml`:

```
[[tool.poetry.source]]
name = "codescoring"
url = "https://save.example.com/pypi/<project>/pypi-proxy/
simple/"
priority = "primary"
```

Credentials are passed through `poetry config`:

```
poetry config http-basic.codescoring <username> <password>
```

#pipenv

In `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>
```

Or in `pyproject.toml`:

```
[[tool.uv.index]]
name = "codescoring"
url = "https://save.example.com/pypi/<project>/pypi-proxy/
simple/"
default = true
```

#Publishing (twine, hosted)

Packages are uploaded to a hosted repository with standard `twine upload`. The upload URL is the **repository root** without `/simple/`.

```
## ~/.pypirc
[distutils]
index-servers = codescoring

[codescoring]
repository = https://save.example.com/pypi/<project>/pypi-
hosted/
username = <username>
password = <password>
```

```
python -m build
twine upload -r codescoring dist/*
```

#Repository URL Migration

Use case: migrating a PyPI repository from Nexus / Artifactory to CodeScoring.Save.

Source	URL in pip.conf before migration	URL in pip.conf after migration
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/`
Official repository	`https://pypi.org/simple`	`https://save.example.com/pypi/<project>/pypi-proxy/simple/`

Authentication parameters (username / password) and the `pip.conf` format remain unchanged.

#Troubleshooting

#Checking the Simple Index

```
curl -u "<username>:<password>" \
  https://save.example.com/pypi/<project>/pypi-proxy/simple/<package>/
```

The response is an HTML page with links to package files. If the page is empty or returns 404, the package is not cached and upstream did not return it.

#Service Status

```
curl https://save.example.com/health
```

#Repository Audit

```
curl -u "<username>:<password>" \
  "https://save.example.com/api/v1/admin/audit?resource_type=repository&q=pypi-proxy&limit=50"
```

#Integration with CodeScoring.OSA

CodeScoring.Save can be used as an upstream repository for OSA Proxy. In this setup, package managers send requests to OSA Proxy, OSA Proxy checks requested components through CodeScoring.OSA, and allowed requests are routed to the Save proxy repository.

CodeScoring.Save is not connected to OSA Proxy through a separate integration API. To check downloaded components, package managers must be configured to work through **OSA Proxy**, and OSA Proxy must be configured for the corresponding upstream repository.

If the upstream repository is a CodeScoring.Save proxy repository, its URL is specified in the `registry` field of the OSA Proxy repository. In this scenario, traffic follows this chain:

```
Package manager -> OSA Proxy -> CodeScoring.Save proxy
repository -> external upstream
```

OSA Proxy scans packages and manifests, calls the CodeScoring API to check policies, and blocks unsafe components depending on the selected `work-mode`.

#What is configured in OSA Proxy

Configuration is performed in OSA Proxy. For each supported format, the corresponding section is enabled and the list of repositories is specified:

```
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
```

For PyPI, `packages-registry` is specified additionally; for Go, `sumdb-registry`; for Docker, `auth-token-url` if the upstream registry requires a separate token service.

#Operation Modes

Check behavior is controlled by the `work-mode` parameter. It can be set globally in the `codescoring` section or overridden for a specific repository.

Supported modes:

- `warmup` — loads data into the CodeScoring cache without blocking components;
- `spectator` — loads data into the CodeScoring cache without blocking components and saves component request results in the platform;
- `moderate` — blocks components that fail policy checks; unscanned components can still be downloaded;
- `strict` — blocks components that fail policy checks; unscanned components cannot be downloaded;
- `strict_wait` — blocks components that fail policy checks; unscanned components are checked with waiting.

#Limitations

Checks through OSA Proxy apply to traffic that passes through OSA Proxy. For hosted Save repositories, checks must be performed before publishing an artifact or through a separate control process.

Security policies are configured in CodeScoring. OSA Proxy uses the CodeScoring API to scan components, retrieve package information, and check policies, but does not create policies inside Save.

#See Also

- OSA Proxy general description
- OSA Proxy service configuration
- OSA Proxy supported protocols
- OSA Proxy Redis and cache configuration

#CodeScoring.OSA

The CodeScoring.OSA module provides supply chain protection through a set of integration components that work together to automatically scan downloaded artifacts and block unwanted packages. CodeScoring.OSA includes both repository manager plugins and a proxy service — all components operate in sync and complement each other:

- plugins are integrated into the *request*|*response* processing of repository managers (e.g., Sonatype Nexus and JFrog Artifactory), blocking the download of unwanted components at the storage level;
- the proxy service intercepts package manager requests to remote repositories, performs scanning, and, if necessary, modifies responses — a convenient option for centralized control or when installing a plugin is not possible.

Integration options:

- Sonatype Nexus Repository — plugin for Nexus (integrated into the repository's request|response pipeline);
- JFrog Artifactory — plugin for Artifactory (similar integration);
- Sfera — plugin for Sfera;
- OSA Proxy — proxy service that intercepts package manager requests to upstream repositories, performs automatic scanning, modifies responses, and manages component access according to security policies;
- OSA Proxy Go — Go version of the proxy service with support for an extended ecosystem set.

#Plugin for Nexus

Supported repository types: Alpine, CocoaPods, Composer, Conan, Conda, Debian, Docker, Go, Maven, NPM, NuGet, PyPI, RPM, RubyGems.

#Installing the plugin

The **CodeScoring.OSA** plugin comes as a JAR file and supports the following Sonatype Nexus Repository versions:

- `nexus-codescoring-plugin-{release}.jar` - for Nexus Repository Community Edition versions from **3.71** to **3.77** and Nexus Repository Pro versions from **3.33.1-01** to **3.77** (supports H2 and PostgreSQL);
- `nexus-codescoring-plugin-legacy-{release}.jar` - for Nexus Repository OSS versions from **3.33.1-01** to **3.70.X** (supports OrientDB).

To add a plugin to **NXRM** you must:

1. Copy the `nexus-codescoring-plugin.jar` file received from the vendor to the `/opt/sonatype/nexus/deploy` directory:

```
cp nexus-codescoring-plugin.jar /opt/sonatype/nexus/deploy/
nexus-codescoring-plugin.jar
```

If **NXRM** is running in a Docker container:

```
docker cp nexus-codescoring-plugin.jar nexus:/opt/sonatype/
nexus/deploy/nexus-codescoring-plugin.jar
```

1. Grant rights for the user and group nexus:

```
chown nexus:nexus /opt/sonatype/nexus/deploy/nexus-codescoring-
plugin.jar
```

If **NXRM** is running in a Docker container:

```
docker exec -it -u 0 nexus chown nexus:nexus /opt/sonatype/
nexus/deploy/nexus-codescoring-plugin.jar
```

1. Check that the user has a minimum set of privileges to work correctly with the plugin:

```
nx-repository-view-*-*-{read,browse}
```

After the completed operations, you need to restart **NXRM**.

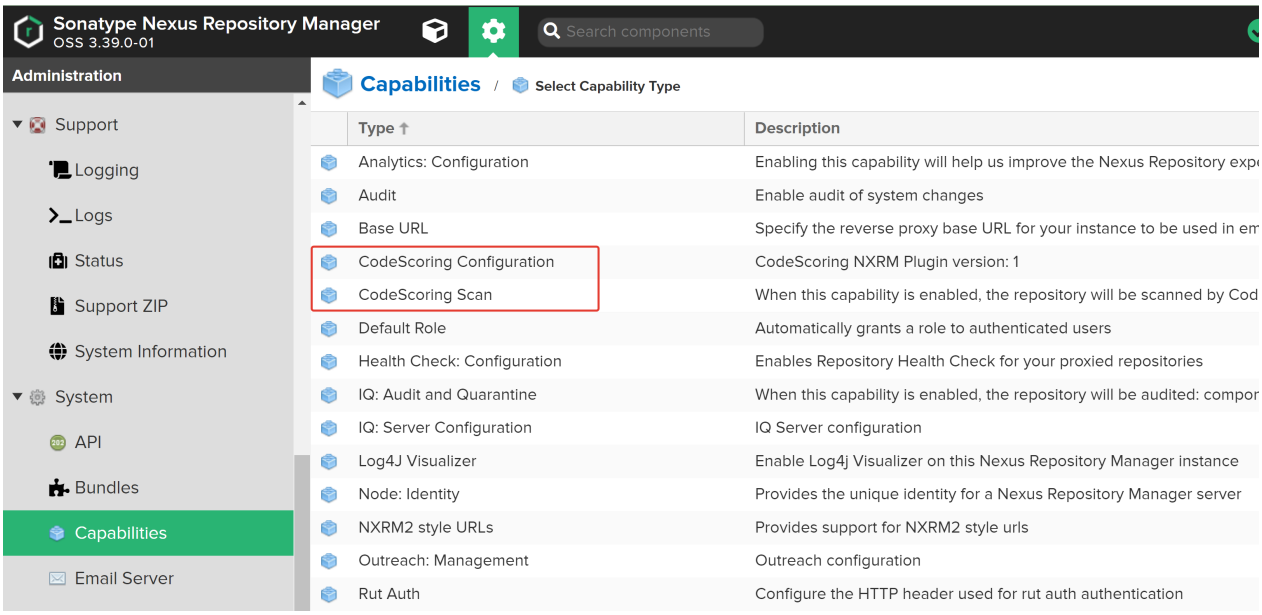
#Plugin setup

To use the **CodeScoring.OSA** plugin in further work, you must use the **Capability** mechanism provided by **NXRM**. **Capability** is a set of APIs and UI components for embedding into **NXRM**, allowing you to expand its functionality.

The **CodeScoring.OSA** plugin provides five new **Capabilities**:

- **CodeScoring Configuration** — setting up interaction with the on-premise platform of **CodeScoring**;
- **CodeScoring Scan** — setting up scanning for a separately selected proxy repository;
- **CodeScoring Docker Repository Scan** – setting up scanning for a separately selected hosted, proxy docker or virtual repository;
- **CodeScoring Repository Mask Scan** – setting up scanning for repositories whose names match a regex mask;
- **CodeScoring All Repositories Scan** – scan settings for all repositories.

After installing the **CodeScoring.OSA** plugin in the `System -> Capabilities` section, it will be possible to create **Capability** through the (`+ Create capability`) interface element.



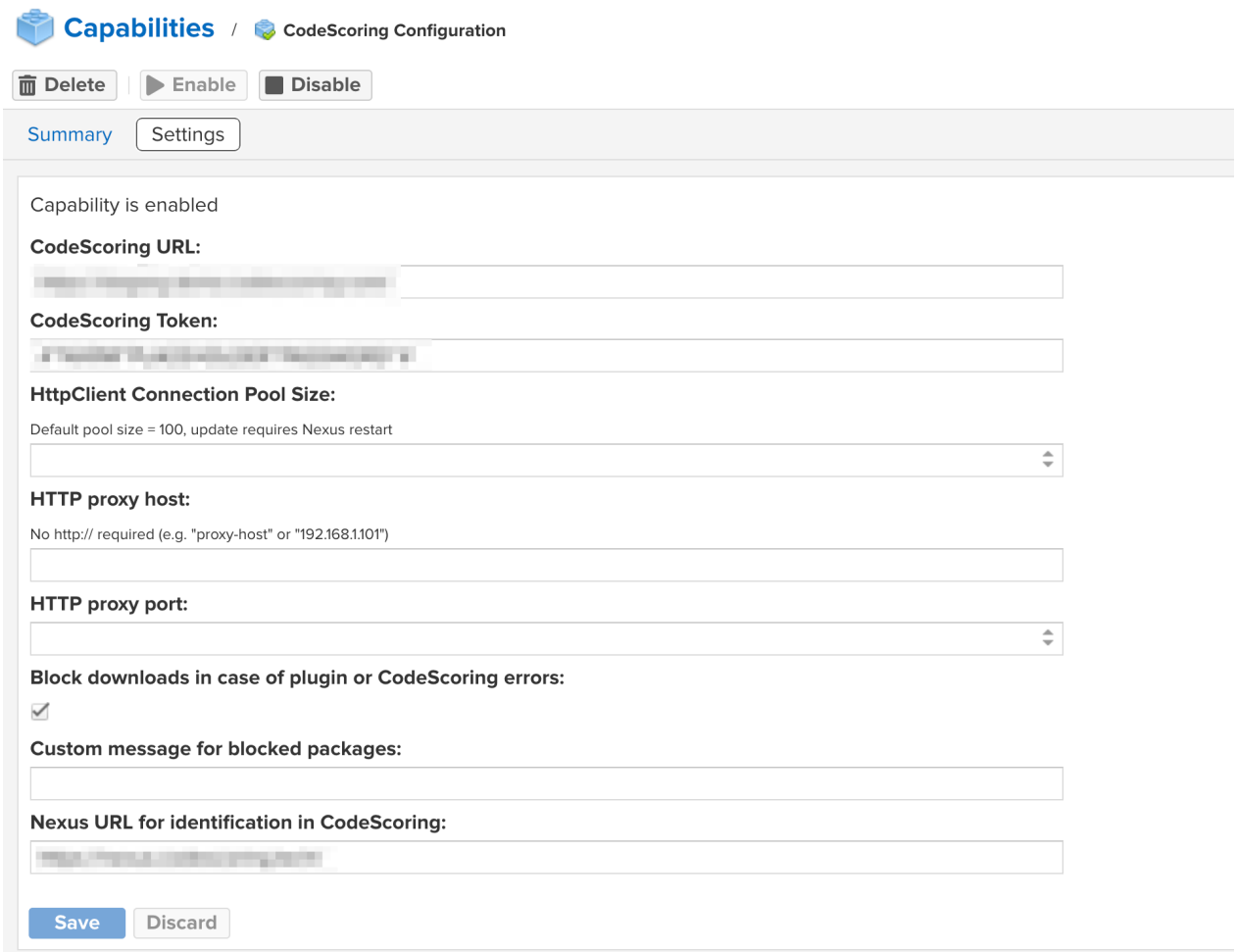
#CodeScoring Configuration

This extension allows you to set general plugin settings for working with the **CodeScoring** platform:

- **CodeScoring URL** – address of the on-premise platform of **CodeScoring**;
- **CodeScoring Token** – key for authorizing API calls (Created from CodeScoring section `Profile -> Home`);
- **HttpClient Connection Pool Size** – number of available connections. This parameter allows you to control the number of parallel requests to speed up scanning;
- **Timeout for CodeScoring requests in seconds** – time to wait for a response from the platform (in seconds, default value is **1800**);
- **HTTP Proxy Host** – proxy server address. Used if it is not possible to establish a direct connection between NXRM and CodeScoring;
- **HTTP Proxy Port** – proxy server port;
- **Block downloads in case of plugin or CodeScoring errors** – blocking the download of a component if there are errors from the plugin or CodeScoring API.
- **Custom message for blocked packages** – message for the user when components are blocked;
- **Append block URL to custom message** – adds a link to the block details page to the custom message;
- **Nexus URL for identification in CodeScoring** – address of the Nexus Repository Manager with the protocol for displaying the results on the platform.

Required fields

The **CodeScoring URL**, **CodeScoring Token**, **HttpClient Connection Pool Size**, **Timeout for CodeScoring requests in seconds**, and **Nexus URL for identification in CodeScoring** fields are required.

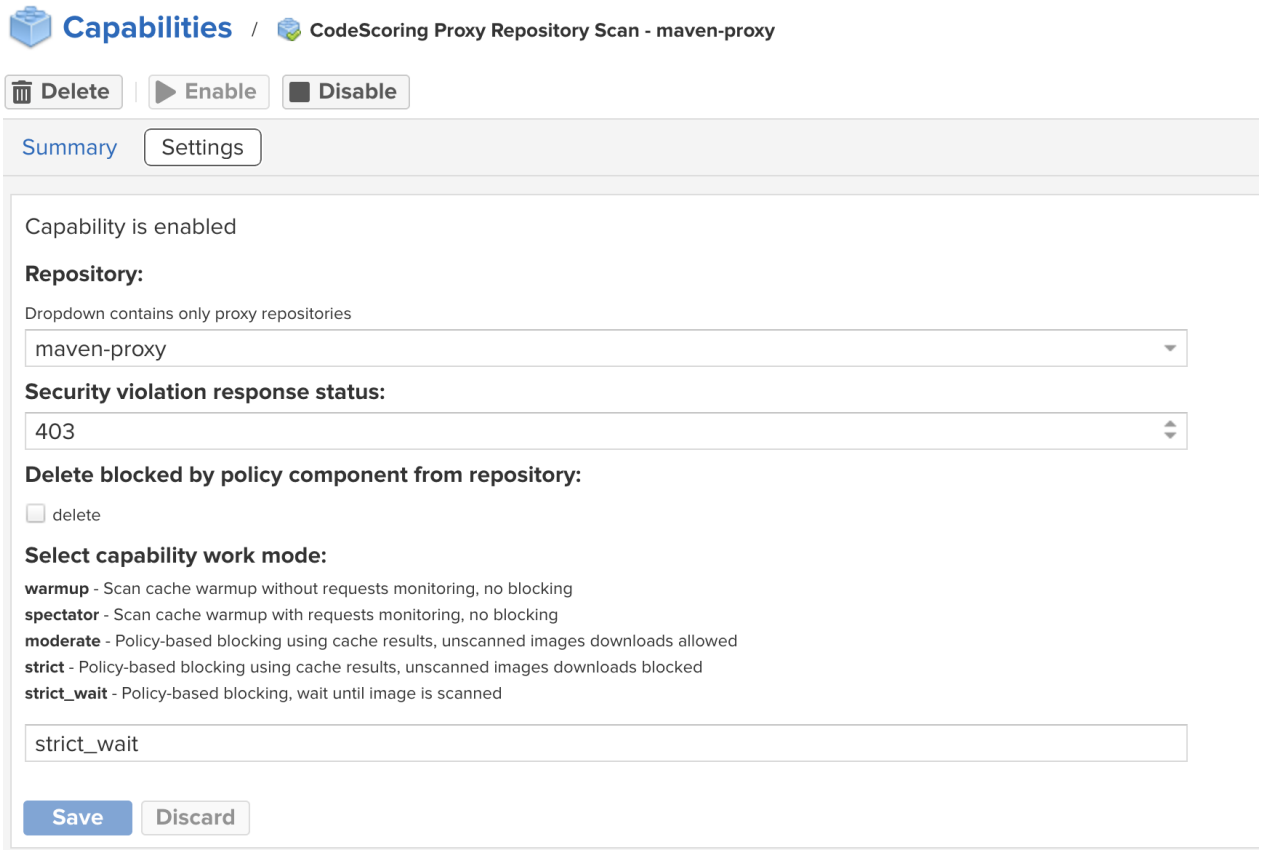


Attention: the specified settings will be used by all instances that check proxy repositories.

#CodeScoring Proxy Repository Scan

This extension allows you to enable component checking for the selected proxy repository with the following parameters:

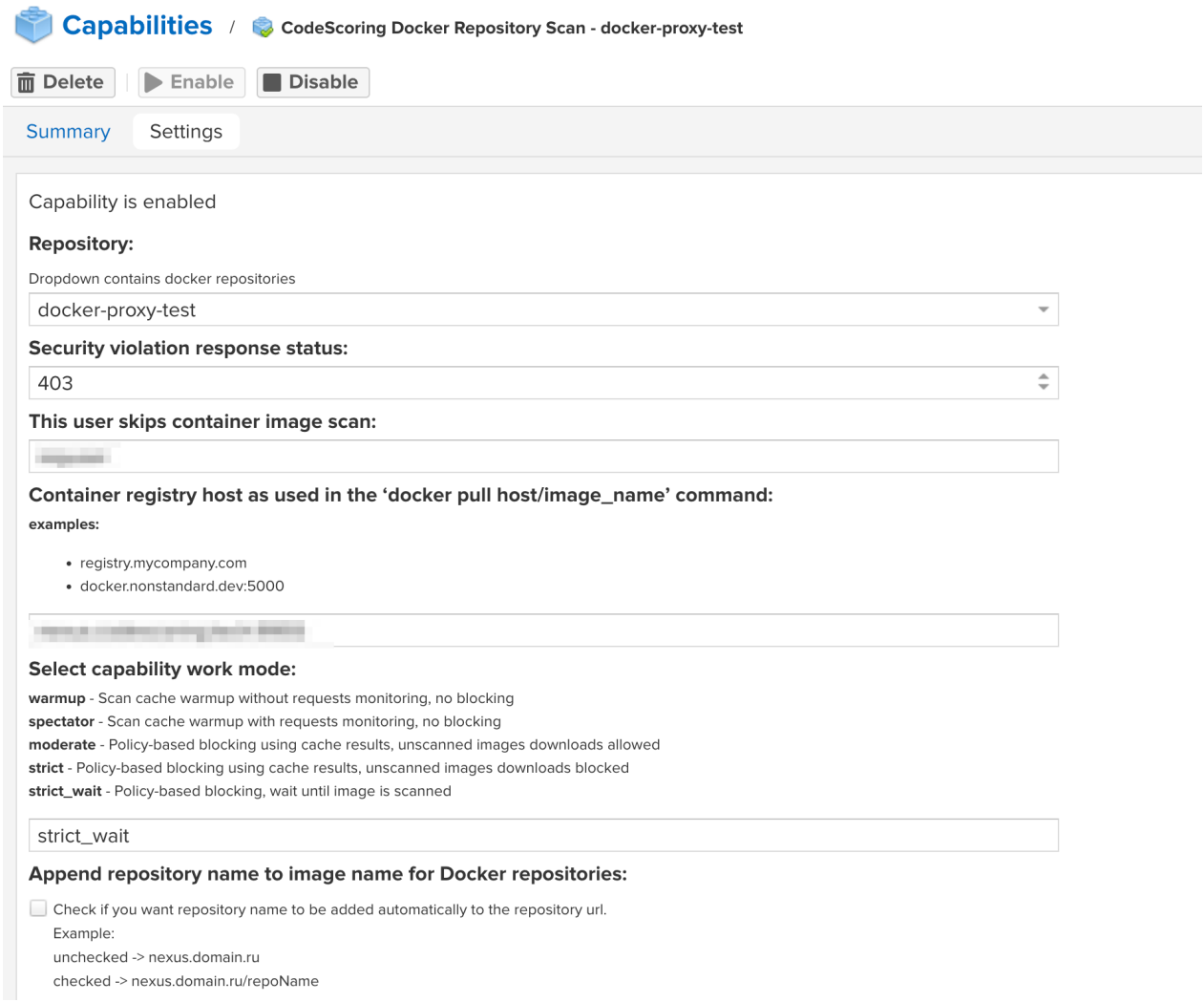
- **Repository** – select the repository for which the escaping function will be applied;
- **Security violation response status** – error code returned when security policies are triggered;
- **Delete blocked by policy component from repository** – forced removal of blocked components from the repository (*creating a “sterile” repository*);
- **Select capability work mode** – plugin operating mode.



#CodeScoring Docker Repository Scan

This extension allows you to enable component check for the selected hosted or proxy docker repository with the following parameters:

- **Repository** – select the repository for which the escaping function will be applied;
- **Security violation response status** – error code returned when security policies are triggered;
- **This user skips container image scan** – user for which image scanning is not applied. Used when loading and checking components by the console agent;
- **Container registry host as used in the `docker pull host/image_name` command** – address (without specifying the protocol) and port through which images for scanning will be downloaded. Used to communicate between Nexus and repository via Docker;
- **Select capability work mode** – plugin operating mode. The operating modes are described in the section below;
- **Append repository name to image name for Docker repositories** – adding the repository name to PURL for correct operation in **RepoPath** mode (in case of accessing the component via the `docker pull registry/repository/image_name` command).

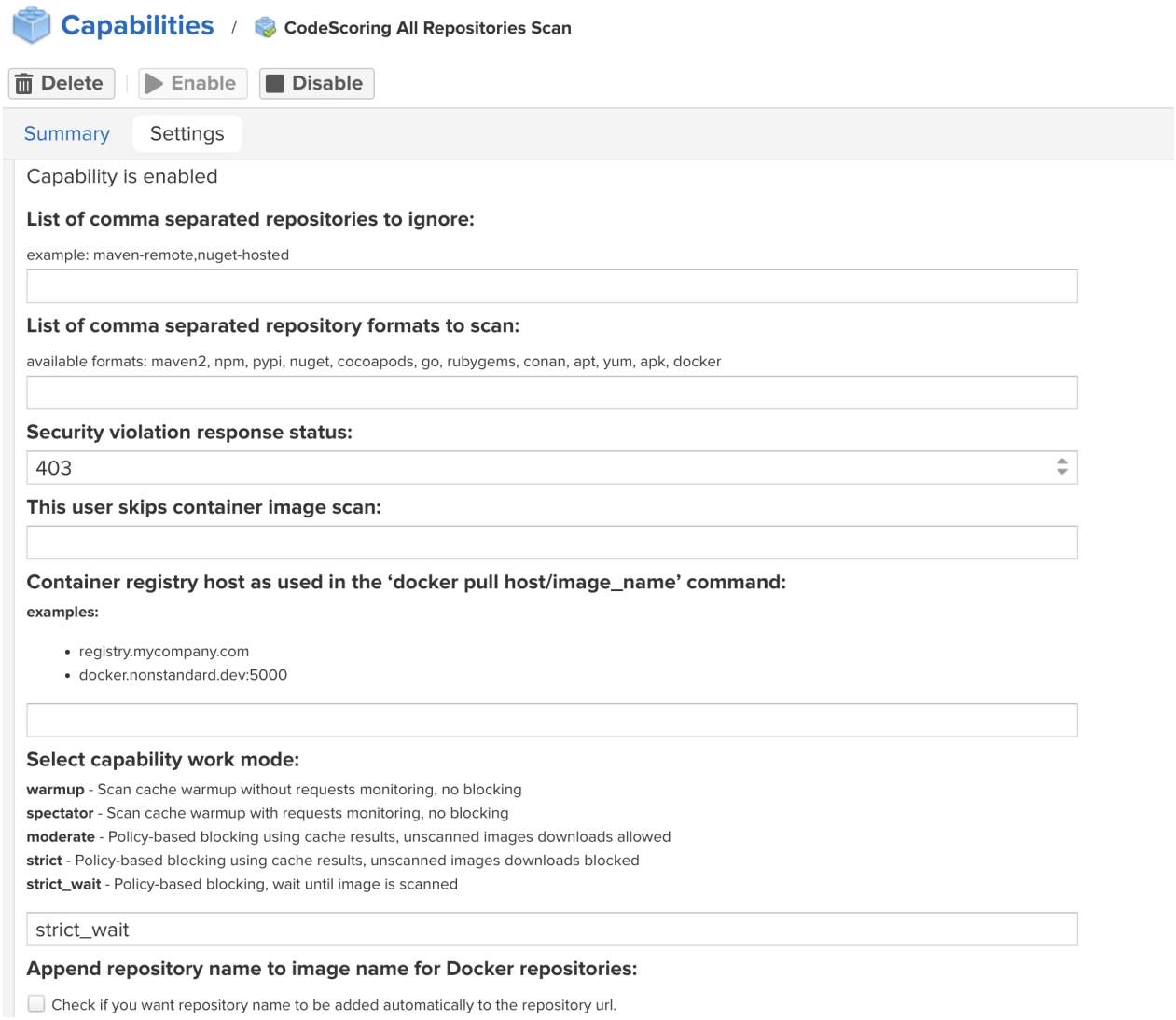


#CodeScoring All Repositories Scan

The extension allows you to enable component checking for all repositories within the Sonatype Nexus Repository Manager with the following parameters:

- **List of comma separated repositories to ignore** – list of repositories that will not be scanned;
- **List of comma separated repository formats to scan** – list of repository formats to scan. Available formats: `maven2`, `npm`, `pypi`, `nuget`, `cocoapods`, `go`, `rubygems`, `conan`, `apt`, `yum`, `apk`, `docker`, `conda`;
- **Repository exclude masks (regex, comma-separated)** – list of regex masks for repositories that must always be excluded from scanning. Masks have priority over other settings;
- **Security violation response status** – error code returned when security policies are triggered;
- **This user skips container image scan** – user for which image scanning is not applied. Used when downloading and checking components by the console agent;
- **Container registry host as used in the `docker pull host/image_name` command** – address (without specifying the protocol) and port through which images for scanning will

be downloaded. Used to communicate between Nexus and repository via Docker;
- **Select capability work mode** – plugin operating mode. The operating modes are described in the section below;
- **Append repository name to image name for Docker repositories** – adding the repository name to PURL for correct operation in **RepoPath** mode (in case of accessing the component via the `docker pull registry/repository/image_name` command).



#CodeScoring Repository Mask Scan

This extension allows you to enable component checking for repositories whose names match a Java regex pattern. The Capability has the following parameters:

- **Repository name pattern (regex)** – Java regex pattern for matching repository names. For example, `npm-.*`, `.*-remote`, or `maven-(release|snapshot)-.*`;
- **Security violation response status** – error code returned when security policies are triggered;

- **Delete blocked by policy component from repository** – forced removal of blocked components from the repository (*creating a “sterile” repository*);
- **Select capability work mode** – plugin operating mode. The operating modes are described in the section below.

#Setting the plugin operating mode

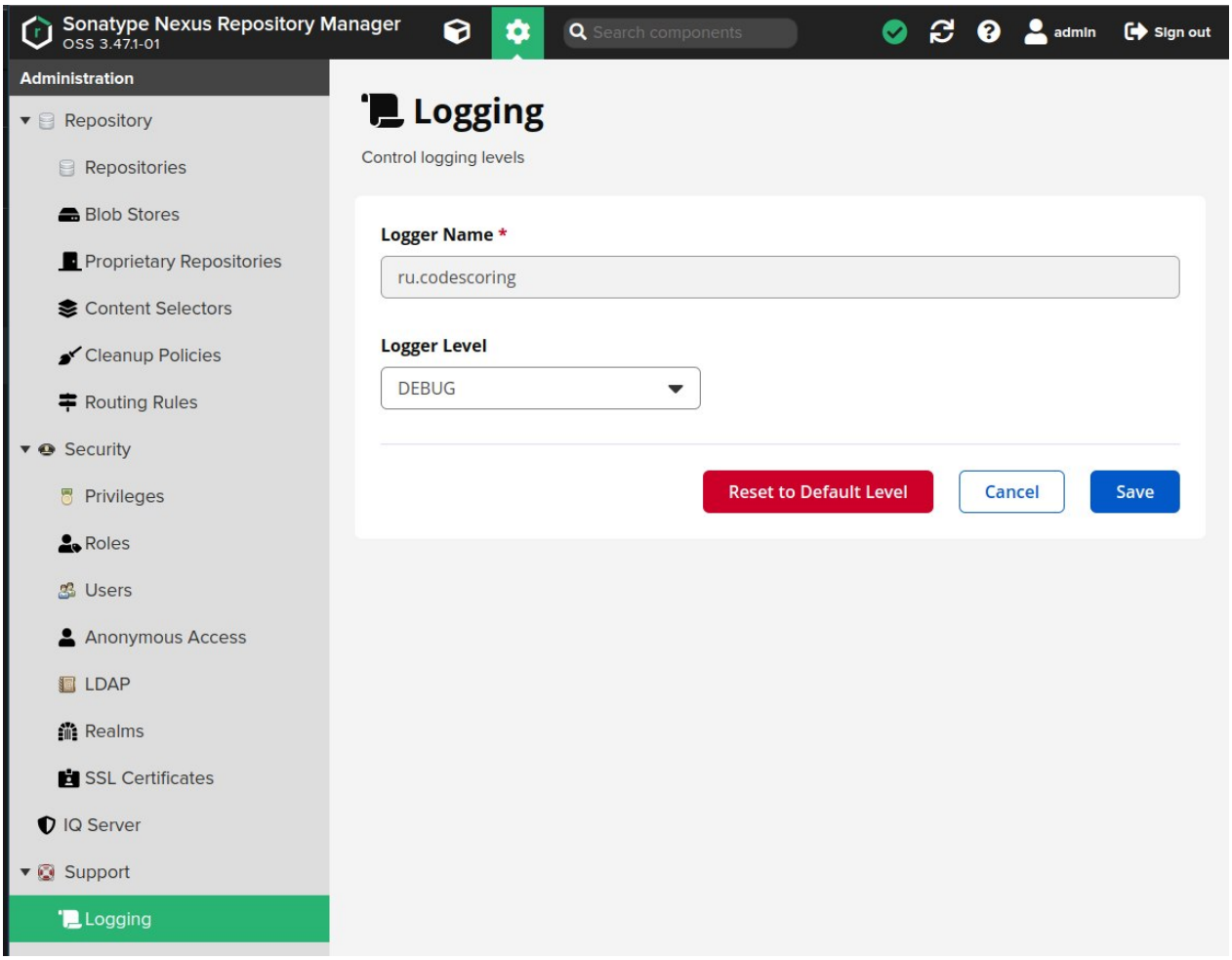
The operating mode of the plugin must be defined as a text string in the corresponding field of the **CodeScoring Proxy Repository Scan**, **CodeScoring Docker Repository Scan**, **CodeScoring All Repositories Scan**, or **CodeScoring Repository Mask Scan** Capability settings.

The plugin has 5 operating modes that determine the severity of component checking before loading.

- **warmup** – loading data into the CodeScoring cache without blocking components;
- **spectator** – loading data into the CodeScoring cache without blocking components, saving the results of component queries to the platform;
- **moderate** – blocking components that have not passed the policy check. Loading of unscanned components is allowed;
- **strict** – blocking components that do not pass the policy check. Loading of unscanned components is prohibited;
- **strict_wait** – blocking components that have not passed the policy check. Pending verification for unscanned components.

#Setting up logging

To configure logging of plugin events, you need to go to the `Support -> Logging` section and add a logger with the name **ru.codescoring** and logging level **DEBUG**.



Event logging results are available in the `Support -> Logs` section.

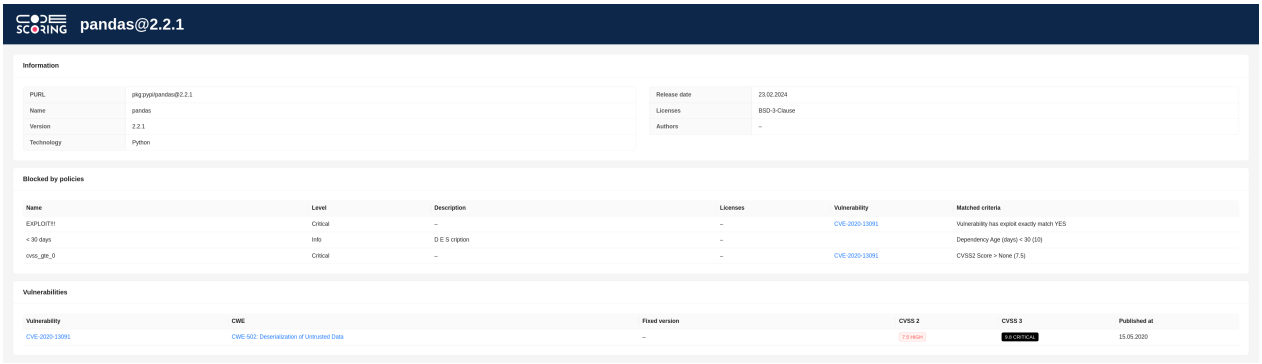
#Blocking components

When a component is blocked from downloading, the user console displays one of the following reasons for blocking:

- **"The download has been blocked in accordance with the policies configured in CodeScoring"** – blocking of the component according to the policies configured on the platform;
- **"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"** – blocking an unscanned component and then starting scanning. Used in `strict` mode;
- **"The download has been blocked due to the failure of the scan of the component in CodeScoring"** – the component could not be scanned;
- **"The download has been blocked due to the wrong mode of the plugin"** – incorrect [plugin operating mode] is used(#_3);
- **"The download has been blocked due to the timeout of the scan of the component in CodeScoring"** – the timeout for scanning the component has expired. Used in `strict_wait` mode;
- **"The download has been blocked, because registry is not configured in CodeScoring"** – there is no corresponding Registry in the platform.

When using the delayed blocking policy, the component receives the `delayed_block` status. Component download in this status is not interrupted.

The response also contains a link to the component page in CodeScoring with information about triggered security policies and found vulnerabilities. When a custom message is used, adding this link is controlled by the **Append block URL to custom message** setting:



#Setting up an SSL connection

To set up an SSL connection between the plugin and the platform, you need to import certificates into the Java Truststore.

#Finding the location of your Java platform

To import certificates into the Java Truststore, you first need to locate your Java installation. This can be done in one of the following ways:

1. **Using the `JAVA_HOME` environment variable**:

```
echo $JAVA_HOME
```

1. **Using the `java` command with the `-XshowSettings:properties` parameter**:

```
java -XshowSettings:properties 2>&1 > /dev/null | grep
'java.home'
```

1. **Using the `readlink` command with the path to `java`**:

```
readlink -f $(which java)
```

The following steps assume that the `$JAVA_HOME` variable is set.

#Downloading the certificate

You can download the certificate with the following command:

```
openssl s_client -connect <codescoring.domain.ru>:443 2>/dev/
null | openssl x509 > codescoring_ca.pem
```

Make sure you replace <codescoring.domain.ru> with the corresponding address of your platform.

#Importing a certificate

Once the certificate has been downloaded, it can be imported into the Java Truststore using the following command:

```
keytool -import -alias <mycert> -keystore $JAVA_HOME/lib/
security/cacerts -file <codescoring_ca.pem>
```

Notes:

- Replace <mycert> with a unique name for your certificate.
- Replace <codescoring_ca.pem> with the actual name of your certificate file.
- You may be asked to enter a password for the Truststore. The default password is changeit.

#Verifying the import

To verify that the certificate was successfully imported, use the keytool command to list the certificates in the Truststore:

```
keytool -list -keystore $JAVA_HOME/lib/security/cacerts
```

Note:

- You can use the grep command to filter the results by your certificate alias:

```
keytool -list -keystore $JAVA_HOME/lib/security/cacerts | grep
mycert
```

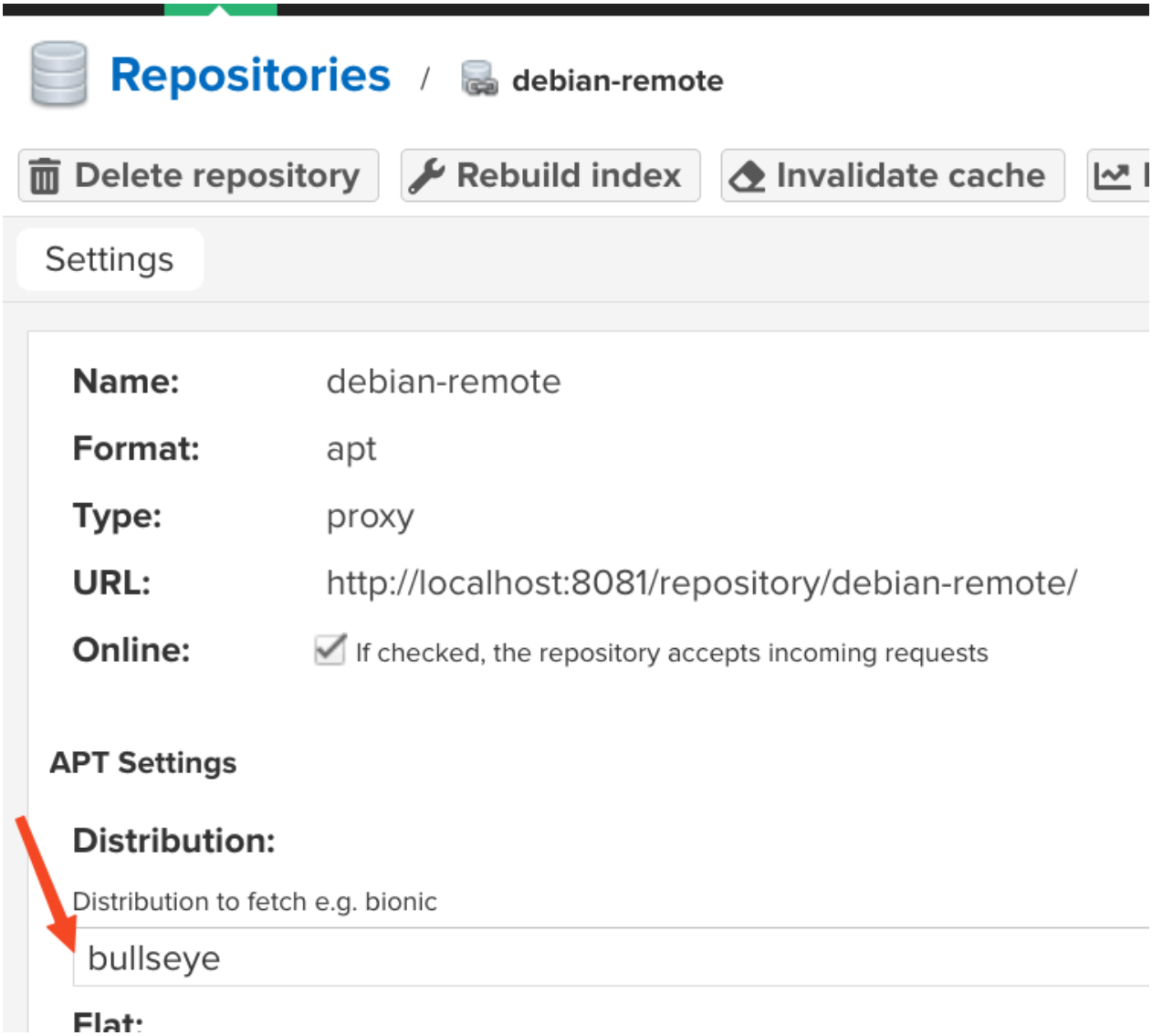
#Working with system packages

#Setting up repositories

For plugin to work correctly with system packages of some ecosystems, additional actions must be taken.

For the plugin to work correctly, you must specify the codename of the distribution from the remote repository, for example "bullseye" for Debian. This name is used in PURL (Package URL) to improve the accuracy of package analysis.

The codename should be in lowercase and without extra symbols.



List of supported Debian distributions:

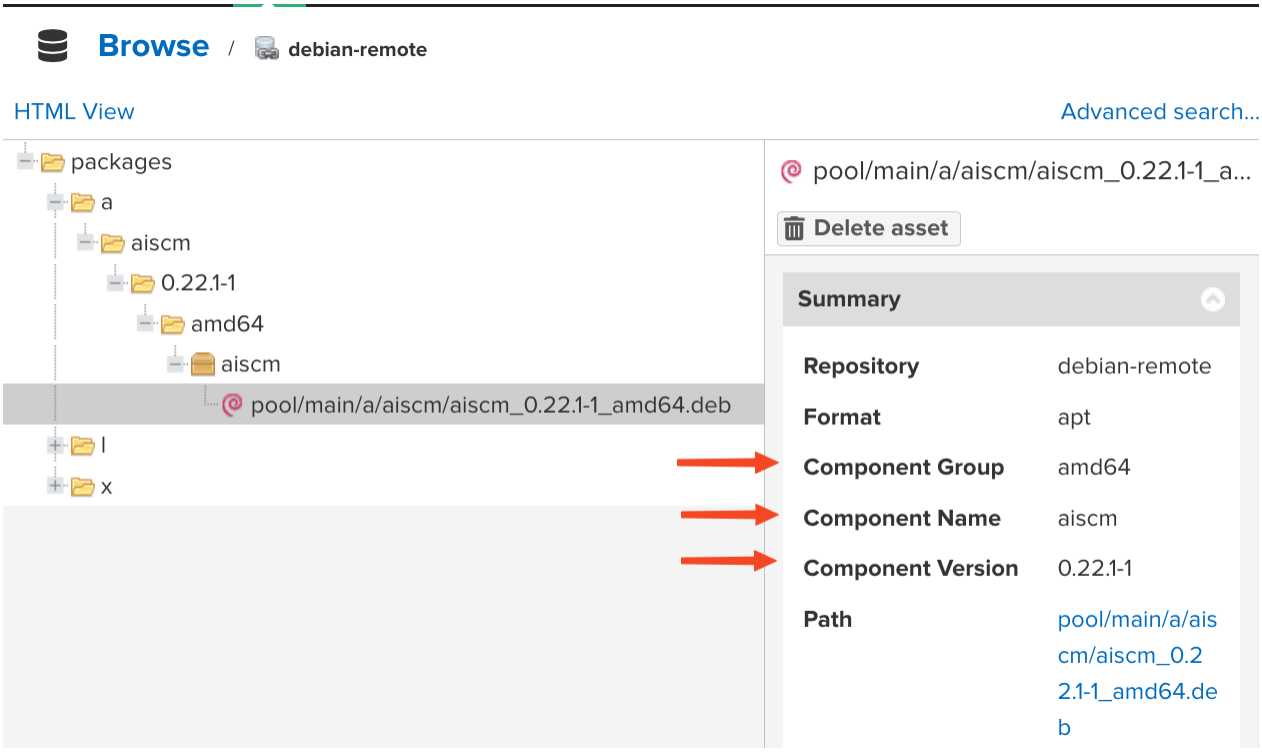
- **Debian 2.0** – *hamm*
- **Debian 2.1** – *slink*
- **Debian 2.2** – *potato*
- **Debian 3.0** – *woody*
- **Debian 3.1** – *sarge*
- **Debian 4** – *etch*
- **Debian 5** – *lenny*
- **Debian 6** – *squeeze*
- **Debian 7** – *wheezy*
- **Debian 8** – *jessie*
- **Debian 9** – *stretch*
- **Debian 10** – *buster*
- **Debian 11** – *bullseye*
- **Debian 12** – *bookworm*
- **Debian 13** – *trixie*
- **Debian 14** – *forky*

List of supported Ubuntu distributions:

- **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*

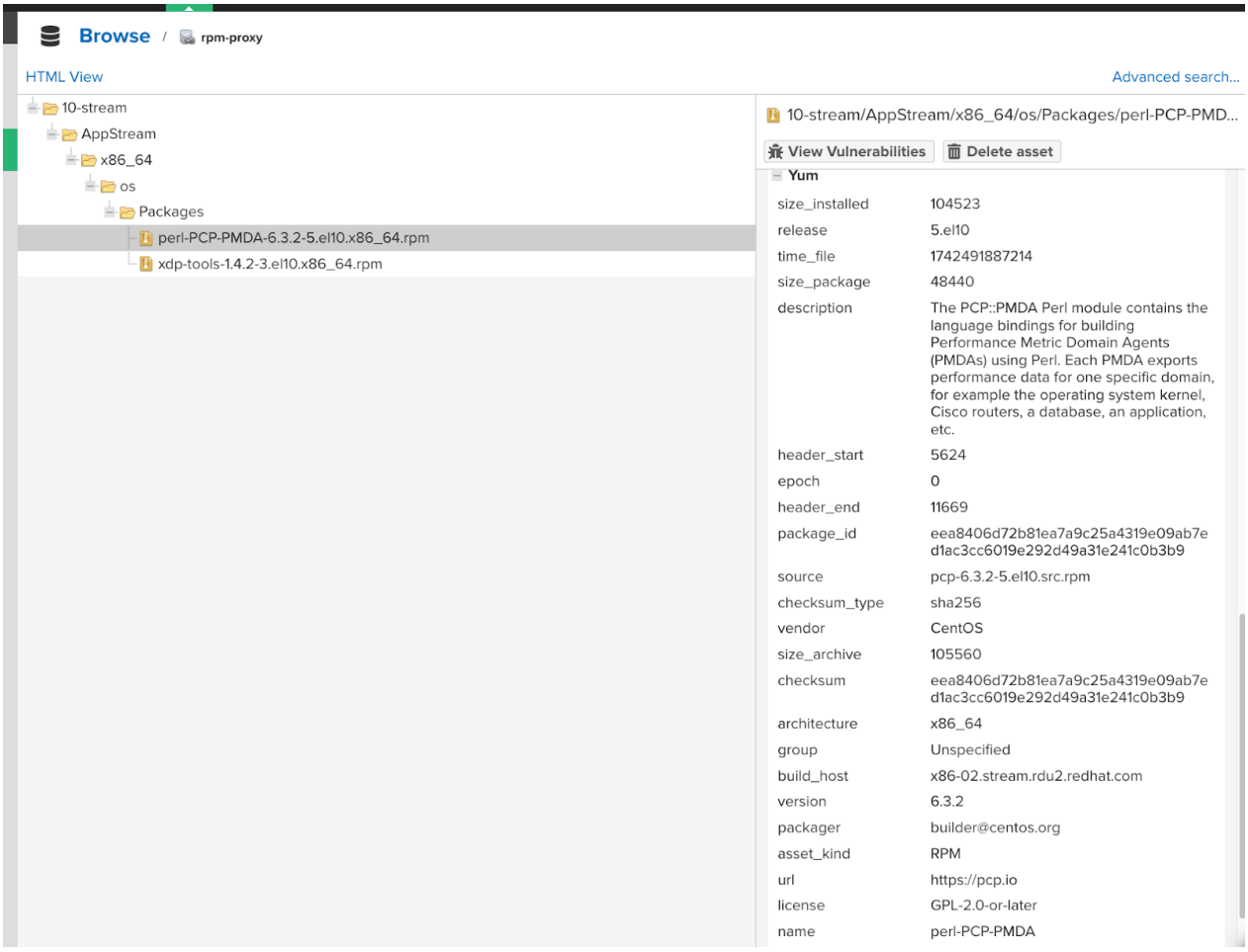
#Browsing Debian package information

Plugin extracts package information from various sources. Specifically, it gets the package name, version, and architecture from the asset's Summary field. If the Summary is missing, the data for the PURL is parsed from the Path.



#Browsing RPM package information

For RPM packages, plugin extracts data from the asset's attributes. This includes the package name, version, and architecture.



#Plugin for JFrog

Supported repository types: Alpine, Cargo, CocoaPods, Composer, Conan, Conda, Debian, Docker, Go, Maven, NPM, NuGet, PyPI, RPM, RubyGems, Swift.

#Installing the plugin

The **CodeScoring.OSA** plugin supports JFrog Artifactory Pro versions **7.43** and higher.

The plugin comes as an archive with the following structure:

CHANGELOG.md

codescoring.groovy

codescoring.yaml
›
lib

codescoring-plugin-jfrog.jar

To add a plugin to **JFrog** you need to:

1. Unpack the resulting archive into the `$JFROG_HOME/artifactory/var/etc/artifactory/plugins` directory.
2. Create a file to configure `codescoring.yaml` in the directory. An example of the content is in the supplied archive.
3. Make a request to **API JFrog Pro** to load the plugin `POST /api/plugins/reload`:

```
curl -X POST https://[JFROG_URL]/api/plugins/reload
```

#Checking the plugin installation

To check the installation of the plugin in the system, you need to check the service logs. Upon successful download and initialization, the following message will appear in the logs:

```
2023-08-08T09:41:35.105Z [jfrt ] [INFO ] [70be801ff583b741]
[r.c.p.codescoring:16 ] [art-init ] - CodeScoring:
Initialization of CodeScoringPlugin completed
```

#Plugin update

If you update an archive with a plugin, you must use the following API command for the updates to take effect:

```
curl -X POST https://[JFROG_URL]/api/plugins/reload
```

In case of updating the plugin configuration in the `codescoring.yaml` file, you must use the following API command:

```
curl -X POST https://[JFROG_URL]/api/plugins/execute/
codeScoringReload
```

#Plugin setup

To configure the plugin, use the `codescoring.yaml` file.

Example of file content:

```
## 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:
```

#Parameter description

- **disablePlugin** – disable the plugin;
- **codeScoringAPI** - settings for the interaction of the plugin with the CodeScoring platform;
- **url** – CodeScoring platform address (protocol must be specified);
- **token** – key for authorizing API calls (*Created from CodeScoring section `Profile -> Home`*);
- **connectionPoolSize** – size of the connection pool with the CodeScoring platform;
- **timeout** - response waiting time (in seconds). By default, if the CodeScoring API does not respond within 60 seconds, the request will be canceled;
- **proxy** - proxy server settings;
- **host** - host/IP;
- **port** - port;
- **blockedBuildResponseCode** – error code returned when security policies are triggered;
- **blockOnErrors** - blocking the loading of components in case of an error when interacting with the CodeScoring platform;
- **scanAllRepositories** - connection of all supported repositories except those specified in the **excludeRepositories** parameter;
- **storeScanProperties** - saving the blocking reason and scan time stamp to the artifact properties;
- **defaults** – default scanning settings for all connected repositories;
- **dockerRegistryUrl** – Docker registry address;
- **workMode** – plugin operating mode. The conditions of each operating mode are described in the section below;
- **skipScanUser** – Artifactory user for whom component scanning is skipped. Necessary so that CodeScoring can independently pick up the component for scanning. The same user must be specified as in the Repo Managers configuration;
- **stripRepoNameInDockerImageName** – remove the repository name from the image name. Used in the Repository Path approach when working with docker registry. By default, the repository name is appended to the image name;
- **repositoryManagerUrl** - Artifactory URL. The same URL must be specified in CodeScoring to apply policies across repositories.
- **deleteBlocked** - delete an artifact blocked by policies;
- **repositories** – list of repositories for which component scanning is operating. For each repository, you can specify parameters separately, as in the **defaults** parameter;

- **repositoryMasks** – list of masks for matching repositories by regular expressions. Each mask supports the same settings as **defaults**, plus a mandatory **pattern** field. See "Setting up repository masks" for details;
- **excludeRepositoryMasks** – list of regex masks for excluding repositories from scanning. Exclude masks take the highest priority;
- **excludeRepositories** - list of exact repository names excluded from processing by the plugin.

Important: for generic and VCS repositories, be sure to specify one of the following repository types in the Internal Description:

- maven
- npm
- pypi
- nuget
- cocoapods
- go
- gems
- debian
- yum
- alpine
- docker
- composer
- cargo
- conda
- conan
- swift

#Setting up repository masks

Masks allow you to configure scanning parameters for groups of repositories without explicitly listing each one, using regular expressions.

#repositoryMasks

The `repositoryMasks` parameter defines a list of masks with regex patterns. Each mask supports the same settings as the `defaults` section (work mode, skip user, etc.), plus the required `pattern` field.

Lookup order for a repository's settings:

1. Exact match in `repositories`
2. First matching mask in `repositoryMasks`
3. Default settings from `defaults`

When `scanAllRepositories: false`, a matching mask also acts as an inclusion criterion for scanning.

Example:

```
repositoryMasks:
  - pattern: "npm-.*"
```

```
    workMode: moderate
    skipScanUser: codescoring
  - pattern: "pypi-.*-remote"
    workMode: warmup
  - pattern: ".*-staging"
    workMode: spectator
    deleteBlocked: true
```

#excludeRepositoryMasks

The `excludeRepositoryMasks` parameter defines a list of regex patterns for excluding repositories from scanning. Exclude masks take the highest priority and are evaluated before `repositories`, `repositoryMasks`, and `scanAllRepositories`.

Example:

```
excludeRepositoryMasks:
  - pattern: ".*-snapshot"
  - pattern: "temp-.*"
```

#Setting up operating modes

The plugin's operating mode is determined by the **workMode** variable in the `codescoring.yaml` file.

The plugin has 6 operating modes that determine the severity of checking components before loading.

- **off** – component scanning is disabled;
- **warmup** – loading data into the CodeScoring cache without blocking components;
- **spectator** – loading data into the CodeScoring cache without blocking components, saving the results of component queries to the platform;
- **moderate** – blocking components that have not passed the policy check. Loading of unscanned components is allowed;
- **strict** – blocking components that do not pass the policy check. Loading of unscanned components is prohibited;
- **strict_wait** – blocking components that have not passed the policy check. Pending verification for unscanned components.

Important: the selected operating mode will affect **all** repositories specified in the `repositories` variable.

#Setting up logging

The file with logging settings is located at `$JFROG_HOME/artifactory/var/etc/artifactory/logback.xml`.

To configure plugin event logging, you need to add the following content to the `logback.xml` file:

```
<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="en.codescoring" additivity="false">
 <level value="debug" />
 <appender-ref ref="FILE_CODESCORING" />
</logger>
```

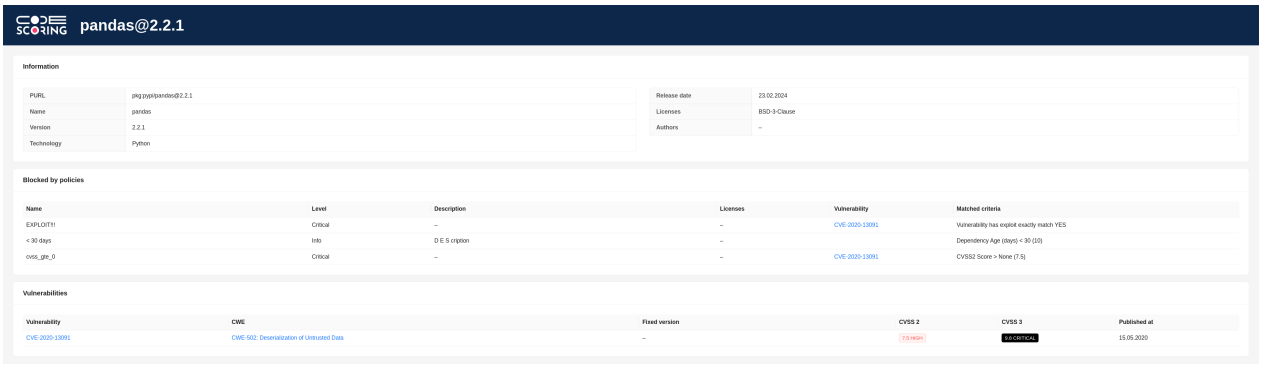
#Blocking components

When a component is blocked from downloading, the user console displays one of the following reasons for blocking:

- **"The download has been blocked in accordance with the policies configured in CodeScoring"** – blocking of the component according to the policies configured on the platform;
- **"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"** – blocking an unscanned component and then starting scanning. Used in `strict` mode;
- **"The download has been blocked due to the failure of the scan of the component in CodeScoring"** – the component could not be scanned;
- **"The download has been blocked due to the wrong mode of the plugin"** – incorrect plugin operating mode is used;
- **"The download has been blocked due to the timeout of the scan of the component in CodeScoring"** – the timeout for scanning the component has expired. Used in `strict_wait` mode;

- **"The download has been blocked, because registry is not configured in CodeScoring"** – there is no corresponding Registry on the platform.

The response also contains a link to the component page in CodeScoring with information about triggered security policies and found vulnerabilities:



Important: if a component does not contain a version, it is not sent for analysis to CodeScoring and, accordingly, is not blocked by the plugin.

#Working with system packages

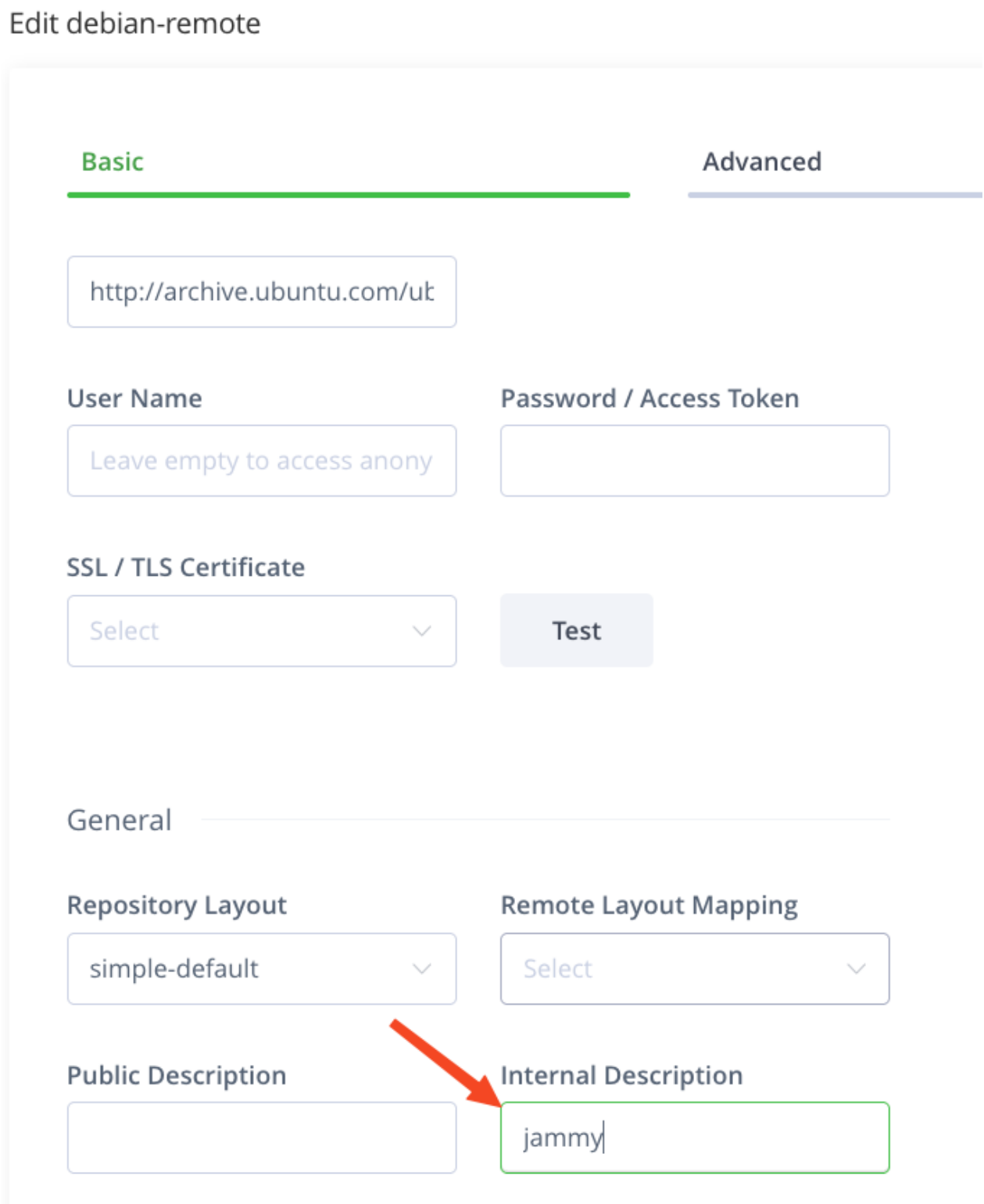
#Setting up repositories

For plugin to work correctly with system packages of some ecosystems, additional actions must be taken.

#Setting up a Debian repository

For correct package analysis, you must specify the codename of the distribution from the remote repository, for example "bullseye" for Debian. This name is entered in the **Internal Description** field. It is used in PURL (Package URL) to improve the accuracy of package analysis.

The codename should be in lowercase and without extra symbols.



List of supported Debian distributions:

- **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*

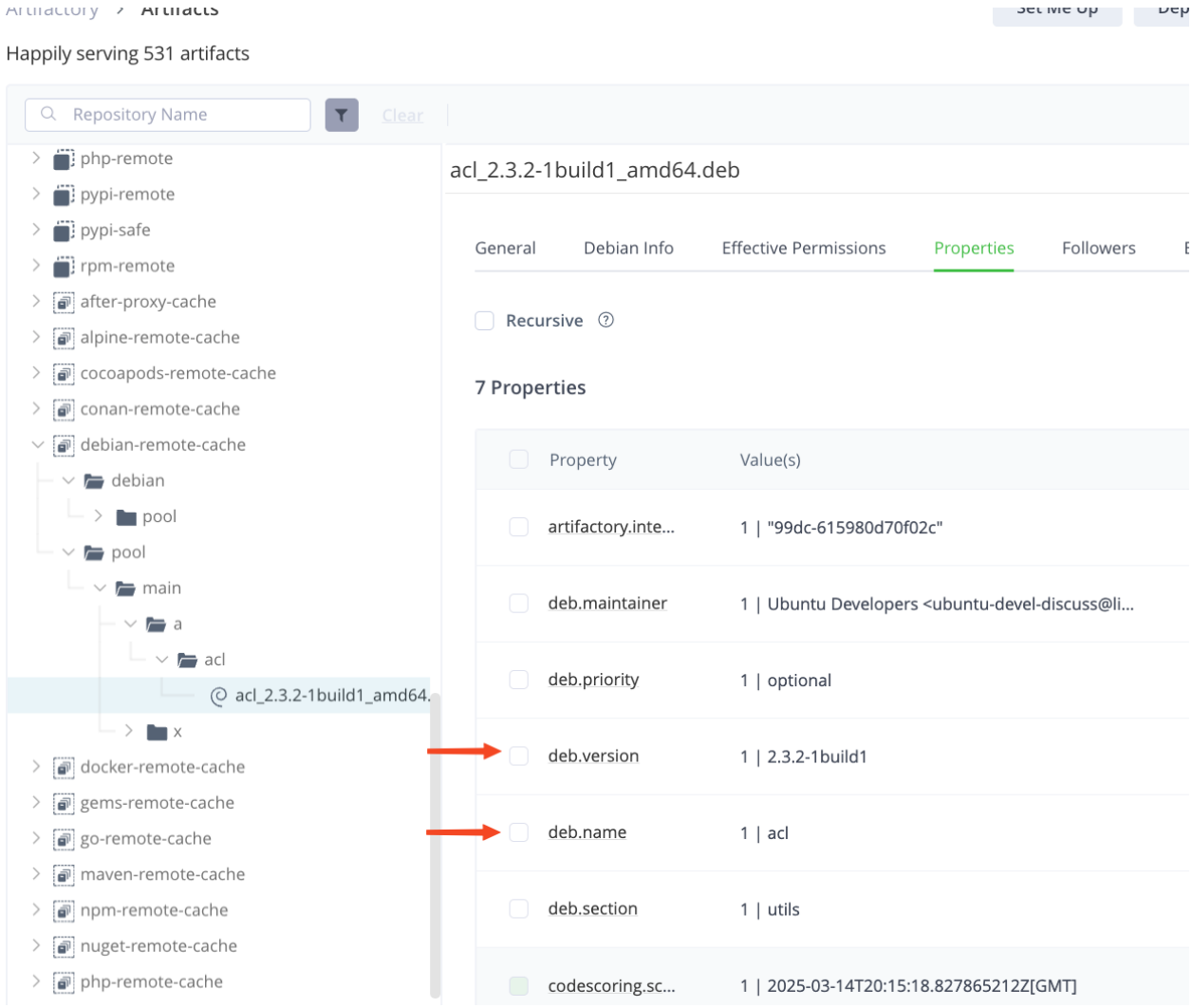
List of supported Ubuntu distributions:

- **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*

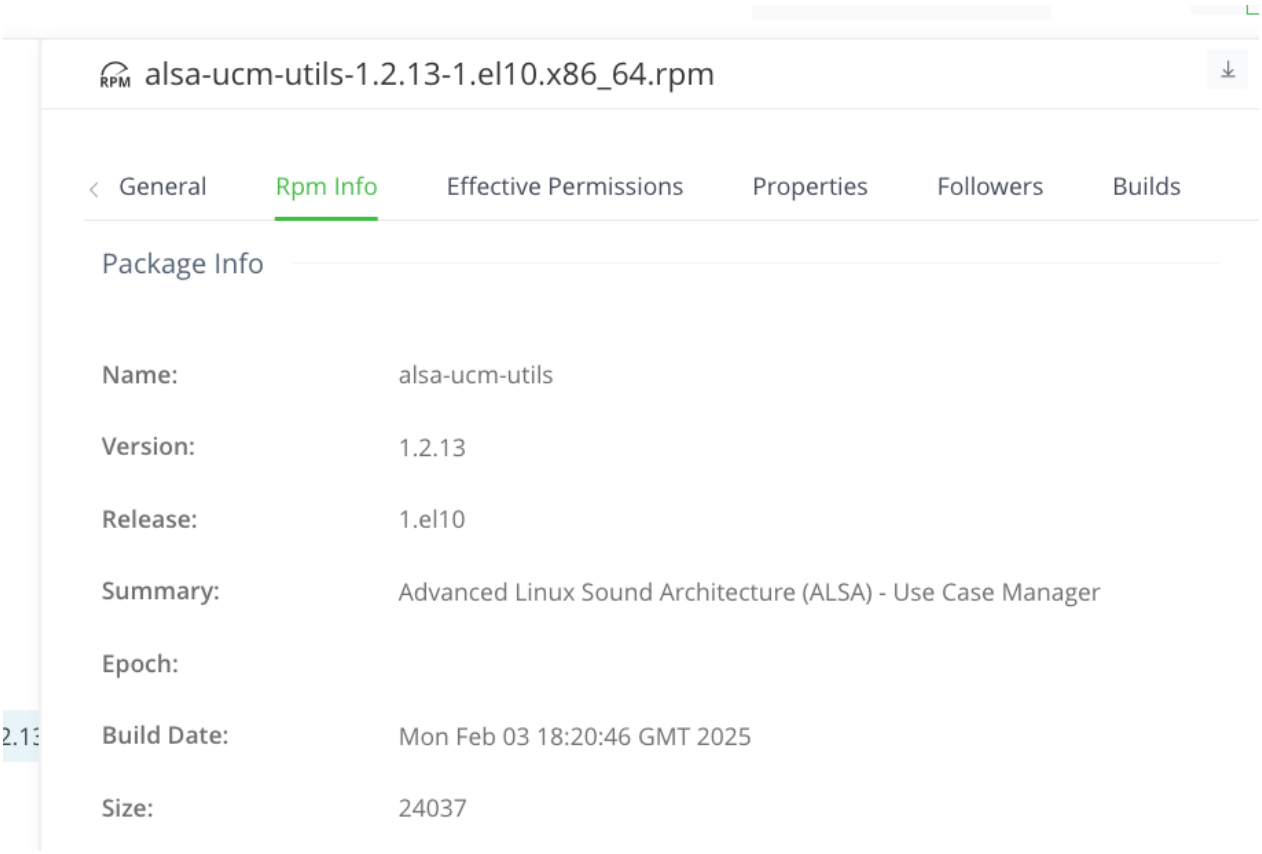
#Browsing Debian package information

Plugin extracts package information from several sources. First, it gets the package name, version, and architecture from the **Properties** of the artifact. If Properties are missing, the data for PURL is parsed from the **Repository Path**.



#Browsing RPM package information

For RPM packages, plugin gets the name, version, and architecture by parsing the **Repository Path**.



#Sfera.Distributions and licenses plugin

#Plugin installation

The plugin is delivered as a jar-file.

To install the plugin in **Sfera**, you need to:

1. Place the plugin jar file and config file `codescoring.yaml` into the `plugins` folder located in the application's PPDL working directory.
2. (*Optional*) To manage enabling/disabling of plugins, you can create `enabled.txt` or `disabled.txt` files in the `plugins` folder.
    - The file should list the names of the plugins to be enabled or disabled.
    - Enable/Disable logic:
        - plugin in `disabled.txt` - disabled;
        - `enabled.txt` is not empty and does not contain the plugin - disabled;
        - in other cases, the plugin is enabled.

#Plugin configuration

The `codescoring.yaml` file is used to configure the plugin.

Example file content:

```
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.
  connectionPoolSize: 50

  # 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:

## 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

## Default settings for all repositories. Can be overridden by
repositories.repo-name settings
defaults:
  dockerRegistryUrl: registry.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

  # Repository Manager url for CodeScoring to apply policies.
  # Value MUST BE equal to Repository Manager URL in
CodeScoring
  # Example: https://sfera.my.domain.ru
  repositoryManagerUrl:

## Settings per repository
## Example:
## repositories:
##   docker-remote:
##   docker-local:
##     dockerRegistryUrl: another-registry.my.domain
##     skipScanUser: anotheruser
##     workMode: spectator
##   pypi-remote:
##     workMode: warmup
repositories:

## List of the excluded repositories. 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, go, gems,
debian, yum, alpine, docker
## Example:
## repositoryTypes:
##   - npm
##   - go
repositoryTypes:
```

#Parameter Description

- **codeScoringAPI** - configuration parameters for plugin interaction with the CodeScoring platform;
  - **url** – CodeScoring platform address (protocol must be specified);
  - **token** – API key for authorization (*Created in CodeScoring section `Profile -> Home`*);
  - **connectionPoolSize** – connection pool size for CodeScoring platform;

        - **timeout** - response timeout (in seconds). By default, if CodeScoring API does not respond within 60 seconds, the request will be cancelled;
        - **proxy** - proxy server settings;
            - **host** - host/IP;
            - **port** - port;
- **blockOnErrors** - block component downloads in case of an error during interaction with the CodeScoring platform;
- **scanAllRepositories** - enable scanning for all supported repositories except those specified in the **excludeRepositories** parameter;
- **defaults** – default scanning settings for all connected repositories;
    - **dockerRegistryUrl** – docker registry address;
    - **workMode** – plugin operation mode. Conditions for each mode are described in the section below;
    - **skipScanUser** – Sfera user for whom component scanning is skipped. Necessary so that CodeScoring can independently pick up the component for scanning. The same user must be specified as in the Repo Managers configuration;
    - **repositoryManagerUrl** - Sfera URL. The same URL must be specified in CodeScoring to apply policies by repositories.
- **repositories** – list of repositories for which component scanning is active. Parameters can be specified separately for each repository, same as in **defaults**;
- **excludeRepositories** - list of repository names excluded from processing by the plugin.

#Work Mode Configuration

The plugin's operation mode is determined by the **workMode** variable in the `codescoring.yaml` file.

The plugin has 6 modes of operation that determine the strictness of component verification before download.

- **warmup** – load data into CodeScoring cache without blocking components;
- **spectator** – load data into CodeScoring cache without blocking components, save component request results on the platform;
- **moderate** – block components that fail policy checks. Downloading unscanned components is allowed;
- **strict** – block components that fail policy checks. Downloading unscanned components is forbidden;
- **strict_wait** – block components that fail policy checks. Wait for verification for unscanned components.

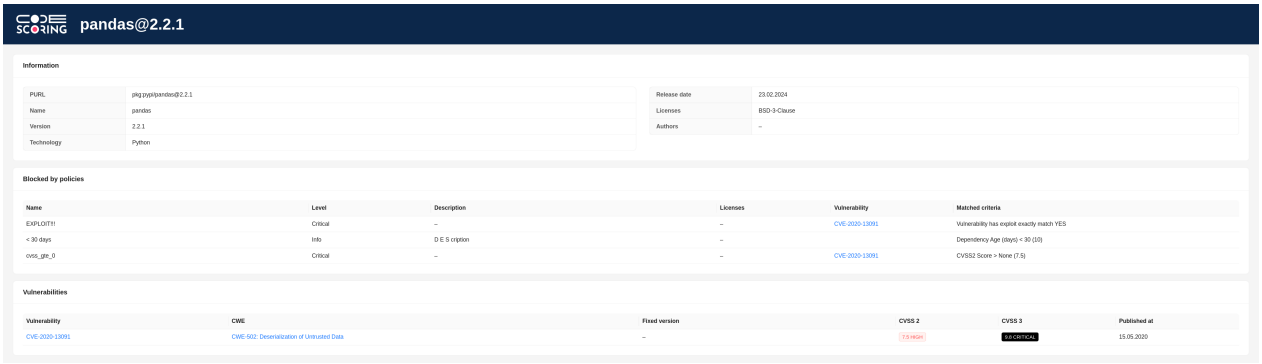
Important: the selected operation mode will affect **all** repositories specified in the `repositories` variable.

#Component Blocking

When a component download is blocked, one of the following blocking reasons is displayed in the user console:

- **"The download has been blocked in accordance with the policies configured in CodeScoring"** – component blocked according to policies configured on the platform;
- **"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"** – blocked unscanned component with subsequent scan initiation. Used in `strict` mode;
- **"The download has been blocked due to the failure of the scan of the component in CodeScoring"** – failed to scan the component;
- **"The download has been blocked due to the wrong mode of the plugin"** – incorrect plugin operation mode is used;
- **"The download has been blocked due to the timeout of the scan of the component in CodeScoring"** – component scan timeout expired. Used in `strict_wait` mode;
- **"The download has been blocked, because registry is not configured in CodeScoring"** – corresponding Registry is missing in the platform.

The response also contains a link to the component page in CodeScoring with information about triggered security policies and found vulnerabilities:



#Connecting a repository manager

Codescoring.OSA performs artifact check in the **Sonatype Nexus Repository** and **JFrog Artifactory** repository managers via OSA plugins. You can pre-configure a connection to a repository manager to make it easier to work with artifacts on your platform.

To add a new repository manager on an platform, follow these steps:

1. Go to `Settings -> Repo managers`.
2. Click on the **Setup new** button.

3. Fill in the fields in the form:
    - **Name** - name in the CodeScoring system;
    - **Is active** - an indication of an active repository manager;
    - **Type** - type (Sonatype Nexus Repository or JFrog Artifactory);
    - **URL** - address with the protocol specified. For example: `https://jfrog.example.com`;
    - **Username** - user with access to the repository manager;
    - **Password** - password.
4. Test the connection after filling in the data using the **Test it** button.

After creating a new connection by clicking **Setup now**, the repository manager will be displayed in the section list, with the possibility to view information about it (**View**), change connection parameters (**Edit**), or delete the connection (**Delete**).

On the view page you can see a list of repositories with the following parameters:

- **Name** - name;
- **Type** - type of repository manager (hosted, proxy and virtual types are currently supported);
- **Ecosystem** - type of stored artifacts (NPM, PyPI and others);
- **Is active** - presence of component requests in the last 24 hours;
- **Is available** - availability of the repository in API (checked once an hour);
- **Last request at** - date and time of the last request to check components.

Repository manager data update process

At the **17th minute of every hour**, the system checks all connected repository managers. If the connection is successful, it loads and updates the list of repositories for each available manager. Alternatively, the list can be updated manually.

You can see the list of packages and images from connected repositories in the `OSA -> Packages` and `OSA -> Container Images` sections, and the list of requests for verification in the `OSA -> Requests` section.

Within the sections, you can filter by individual components or the following fields:

- **Repository** – a repository within the artifact storage;
- **Technology** – a programming language or operating system;
- **License** – a component distribution license;
- **Block Status** – an indicator of a blocked component request;
- **Last Request** – the date of the last component request;
- **Repository Manager** – the name of the connected manager in CodeScoring;
- **Repository Ecosystem** – the type of stored artifacts (PyPI, NPM, etc.);

- **Contains Vulnerabilities** – the presence of vulnerabilities in the requested components;
- **Current** – a flag for an updated component. You can read more about this flag on the Updating Component Data page.

In addition, once the manager is connected, it is possible to customize security policies for individual repositories.

#Connecting registry with container images

#Supported Registries

CodeScoring supports integration with image registries in the following tools:

- Sonatype Nexus Repository;
- JFrog Artifactory;
- GitLab;
- Harbor;
- GitFlic;
- Others that use the Registry V2 API protocol.

#Image Loading Mechanism

CodeScoring loads the information about container images from registries the following way:

1. Lists all container image names;
2. For each container image name, lists tags;
3. For each container image name and tag, requests a manifest;
4. Parses the architecture and the sha256 digest of an image from a manifest;
5. Saves the information about container images currently present in the registry.

Note: CodeScoring identifies a combination of the following as a distinct image:

- Container registry;
- Image name;
- Image sha256 digest.

#Configuring an Integration

To work with Container Images, it is necessary to connect registry with images in `System -> Registries` section.

Go to the form of creating a new connection using the **Setup new** button. The following fields should be filled in the form:

- **Name** - name of the registry;

- **Type** - type of registry implementation (Sonatype Nexus Repository, JFrog Artifactory, JFrog Artifactory Repository Path or other);
- **Is active** - the sign of the active registry. For inactive registries the list of available images will not be updated;
- **Auth type** – authorization scheme (Basic, Bearer or Auto);
- **URL** - the address of the registry with the protocol. For example: `https://jfrog.example.com`;
- **Max concurrent connections** - the maximum number of connections concurrently made to the registry during the process of image loading;
- **Max keepalive connections** - the maximum number of connections that are kept alive for future reuse during the process of image loading;
- **Keepalive expiry, seconds** - the maximum time a keepalive connection can exist before expiring;
- **Page size** - the number of records to request during paginated listing of registry instances;
- **Connect timeout, seconds** - for how many seconds to wait before failing while establishing a connection;
- **Pool timeout, seconds** - for how many seconds to wait before failing while waiting for a connection from the connection pool. This often happens when the RPS or the maximum number of concurrent connections is significantly limited, therefore, in these cases, the configuration value must high;
- **Read timeout, seconds** - for how many seconds to wait before failing while reading from a connection;
- **Write timeout, seconds** - for how many seconds to wait before failing while writing to a connection;
- **Max RPS** - maximum number of requests per second the system will make to a registry during image loading process;
- **Skip TLS Verification?** - whether to disable certificate validation for TLS/SSL connections;
- **Username** - username with access to the registry;
- **Password** - password (for the GitFlic manager type, use the transport token);
- **Load full images list?** - whether to periodically load information about images present in the registry;
- **Repositories to load** - select repositories to load container images from (available for type of manager JFrog Artifactory Repository Path).

Important notes about GitLab Container Registry implementation:

- When setting up an integration with a GitLab Container Registry, you must select the authorization type **Bearer**;
- According to the GitLab Documentation, in order to list all container image names in `/v2/_catalog/`, administrator credentials are required.

You can test the connection after filling in the data by pressing the **Test it** button.

#Viewing Container Registries

After creating a new connection by the **Setup now** button, the registry will be displayed in the section list, with the possibility to view information about

it (**View**), change connection parameters (**Edit**), or delete the connection (**Delete**).

To update the list of available images, click **Update images list** on the view page. You can check the connection by clicking the **Refresh status** button.

#Configuring OSA policies

To decide whether to block or allow downloading of components, the plugin uses the CodeScoring policies mechanism.

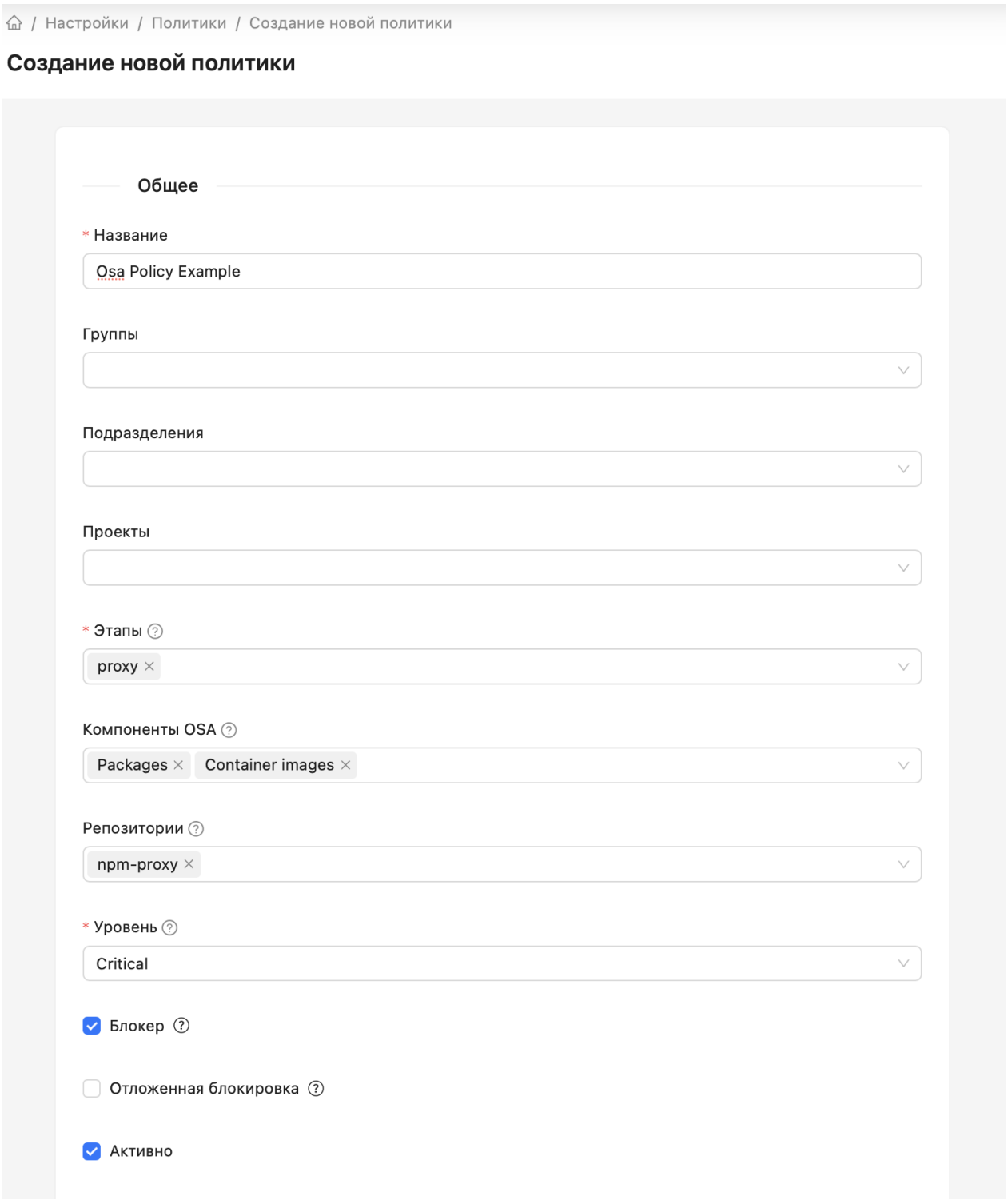
In order for the policy to be applied when called from the plugin, you must set the following settings for the selected policy in the `Policies` section:

- **Stages** – specify the *proxy* value;
- if it is necessary for the policy to block a component request, set the **Blocker** flag.
- set the *Is Active* flag.

In addition, you can clarify the scope of the policy using two parameters:

- **OSA Components** – type of components to which the policy will apply (packages or container images);
- **Repositories** – list of repositories with components.

Important: to select a repository, you must first connect the repository manager to the platform.



#Working with OSA components

Components that are checked by the plugin are displayed in the 0SA section of the CodeScoring UI.

#View package list

The list of scanned packages can be viewed in the `OSA -> Packages` subsection. The Packages table contains **all** packages that were scanned while the OSA plugin was running, with the following information:

- **Package** – name of the package (with a link to its page);
- **Technology** – technology (programming language or assembly tool);
- **Licenses** – licenses;
- **Authors** – authors;
- **Vulnerabilities** - number of vulnerabilities found;
- **Block status** – component blocking status at the time of the last request;
- **Released** – date and time when the package version was published;
- **Last requested at** – date and time of the last package request.

The package table can be filtered by technology, license, blocking status, or time of last component request.

By clicking on the name of the package, you can go to its individual page, where information about the found vulnerabilities and triggered policies is displayed.

#Package details page

The package details page contains the main component attributes:

- **PURL** – unique package identifier;
- **Actual** – component actuality status;
- **Technology** – package technology;
- **Licenses** – package licenses;
- **Version** – package version;
- **Authors** – package authors;
- **Home page** – link to the project page;
- **Index URL** – link to the package index page;
- **Released** – date and time when this version was published;
- **Last requested at** – date and time of the latest request;
- **First requested at** – date and time of the first request;
- **Policies updated** – date and time of the latest policy update for the component according to the update mechanism;
- **Status** – package revoked status;
- **Repository** – repository name where the package was received from;
- **Repository manager** – repository manager name;
- **Package link** – link to the package in the repository manager.

#View a list of container images

Container images are displayed in the `OSA -> Container images` subsection after connecting the corresponding registry. Each entry in the list contains the following information:

- **Name** – name of the image;

- **Container registry** – name of the registry that contains the image;
- **Dependencies** – number of dependencies found;
- **Vulnerabilities** - number of vulnerabilities found;
- **Scan status** – image scanning status;
- **Block status** – component blocking status at the time of the last request (for repositories with the OSA plugin);
- **Last scanned** – date and time of the last scan;
- **Last requested at** – date and time of the last package request.

The table with images can be filtered by registry name, as well as scanning and blocking status.

By default, the added image is not scanned. To carry out the analysis, you need to go to the image page and click on the **Run SCA** button. Based on the scanning results, information about the found dependencies and vulnerabilities, as well as a list of triggered security policies and image layers, will appear on the image page.

A SBOM and a PDF report are available for download, based on the data from the most recent successful image scan.

#Container image details page

The container image details page contains the main component attributes:

- **PURL** – unique image identifier;
- **Digest** – image digest;
- **Tags** – image tags;
- **Actual** – component actuality status;
- **Policies updated** – date and time of the latest policy update for the component according to the update mechanism;
- **Last requested at** – date and time of the latest request;
- **First requested at** – date and time of the first request;
- **Last scanned** – date and time of the latest scan;
- **Block status** – component blocking status at the time of the last request;
- **Container registry** – registry name where the image is stored;
- **Image links in registry** – links to the image in connected registries;
- **Base image name** – name of the image this image was built from;
- **Base image version** – version of the image this image was built from;
- **Repository** – repository name where the image was received from;
- **Repository manager** – repository manager name.

#View image layers list

Container image layers are displayed in the `OSA -> Image layers` subsection after images are scanned via an SCA project or from the `OSA -> Container images` section. The list helps you assess layer composition and size, as well as how many images use each layer.

The table can be filtered by layer ID, media type, and the image that contains the layers.

Each entry in the list contains the following information:

- **ID** – layer identifier (digest) and a short description of the build instruction that produced the layer; click it to view the full command;
- **Media type** – MIME type of the layer content;
- **Size, MB** – layer size in megabytes;
- **Number of images** – number of images that include this layer.

As in other list sections, pagination, rows per page, table display settings, and column sorting are available.

#View component requests

The list of component requests from proxy repositories with the plugin connected can be viewed in the `OSA -> Requests` subsection.

Package requests appear on the **Packages** tab and by default contain the following information:

- **Package** – name of the package (with a link to its individual page);
- **Technology** – technology;
- **Plugin mode** – plugin operating mode;
- **Block status** – blocking status;
- **Requested at** – date and time of the request;
- **Request initiator** - request initiator username from repository manager.

Container image requests appear on the **Container images** tab and by default contain the following information:

- **Container image** – name of the image (with a link to its individual page);
- **Registry** – name of the registry that contains the image;
- **Plugin mode** – plugin operating mode;
- **Block status** – blocking status;
- **Requested at** – date and time of the request;
- **Request initiator** - request initiator username from repository manager.

You can also display the **CodeScoring User** (CodeScoring user configured in the OSA plugin) and **Requested PURL** (PURL of the requested component) columns for both tables.

#Updating component data

CodeScoring.OSA automatically updates information about previously requested components in the background to ensure that the data is up-to-date and that performance is maintained.

#Update mechanism

Packages and images that have been successfully requested at least once in the last 14 days are considered up-to-date, the period can be changed in the settings. The system updates data on such packages every 2 hours, including information on vulnerabilities, licenses, and meta-information on packages.

If a component has not been requested for 14 days, it is automatically transferred to the **archived** status. Such components are no longer updated until the next request.

#Archiving and deletion of components

By default, data on archived components is saved in the system, but it is possible to enable their automatic deletion. This behavior is controlled by parameters in the application configuration (file `app.env`):

- `OSA_ARCHIVE_THRESHOLD_DAYS` — after how many days without requests the component is considered archived (default: 14);
- `OSA_ARCHIVE_AUTO_CLEANUP_ENABLED` — enable deletion of data about archived components (default: `False`);
- `OSA_ARCHIVE_RETENTION_PERIOD_DAYS` — retention period of data about archived components before deletion (default: `30`);
- `OSA_ARCHIVE_CHUNK_SIZE` — chunk size for batch processing of components during archiving and deletion (default: `1000`).

#Filtering by actuality

The `OSA → Packages`, `OSA → Images` and Alerts sections have a filter **Is actual**, which allows you to control the display of components:

- **Yes** — only actual (updated) components are displayed;
- **No** — only archived (not updated) components are displayed.

By default, all components are displayed.

#Deployment

Following the configuration of the `application.yml` file, the application can be deployed and executed either within a Docker container environment or orchestrated via a Helm chart for Kubernetes (k8s) deployments.

#Docker container deployment

To instantiate the application as a Docker container, execute the following command:

```
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>
```

#Kubernetes deployment (Helm Chart)

For Kubernetes environments, the application can be deployed using the provided Helm chart, accessible at `https://{REGISTRY_URL}/repository/helm/`.

Installation order:

1. Create a namespace.

   ```
   kubectl create namespace codescoring
   ```

2. Create a secret to access the private Docker registry, using the address (`REGISTRY_URL`), login (`USERNAME`) and password (`PASSWORD`) received from the vendor.

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

3. Install Helm using your preferred method.

4. Run the following commands to add the current Helm repository to the local machine:

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

5. Create a `values.yaml` file with the following content:

   ```
   env:
     javaOpts: "-Xmx4g" # Optional: JVM parameters
   configuration

   config: |
     # This field must be populated with the content of the
   application.yml file

   # There's an option to create an Ingress resource
   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. Run the command to install the chart

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

#Service configuration

#Main parameters

Configuration of **OSA Proxy** is done via the `application.yml` file:

Example configuration file

```
## CodeScoring parameters
codescoring:
  host: CodeScoring server URL
  token: authorization token ((with access level User or
higher)
  work-mode: operation mode (applies only to package scanning)
              # warmup | Warm up scan cache without monitoring
requests, no blocking
              # spectator | Warm up scan cache with request
monitoring, no blocking
              # moderate | Policy-based blocking using cache
results, unscanned components allowed
              # strict | Policy-based blocking using cache
results, unscanned components blocked
              # strict_wait | Policy-based blocking, wait until
the component is scanned
  proxy-manager-host: proxy server host
  enable-status-line: true/false (adds the block reason to the
status line)
```

```
  block-status-code: status code for blocking package downloads
  block-on-codescoring-errors: block package download on 5xx
status, scan_failed errors
  override-block-url: true/false (replaces URL in block reason
link with the one specified in codescoring.host)
  remove-blocked-versions: true/false
(default: true; when true – blocked versions are removed from
the manifest; when false – they are marked as deprecated)
## PyPI settings
pypi:
  enabled: true
  repository:
    - name: internet-pypi
      scan-manifest: true
      scan-package: 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
## Maven settings
maven:
  enabled: true
  repository:
    - name: internet-mvn
      scan-manifest: true
      scan-package: 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
## NPM settings
npm:
  enabled: true
  repository:
    - name: internet-npm
      scan-package: true
```

```
      scan-manifest: 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
## NuGet settings
nuget:
  enabled: true
  repository:
    - name: codescoring-nuget
      scan-package: 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
## GO settings
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/
## Debian settings
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
## Alpine (APK) settings
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
## RPM settings
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
## Docker Registry settings
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
```

Specifics of working in Nexus Repository and JFrog Artifactory

- For JFrog Artifactory, it is recommended to set a `Custom Base URL` and use it in the `registry` field to correctly replace package references within manifests.

- In the configuration `package manager -> jfrog -> OSA proxy -> internet`, in the additional JFrog repository settings, it is necessary to set the `Bypass HEAD requests flag`.
- There is no identical functionality for Nexus Repository; the host and port (if specified) from the request will be used in manifests. If a `reverse proxy` is available, it is recommended to use its link. For example: `registry: https://nexushost.ru/repository/pypi-proxy`.

#Additional settings

#Logging level settings

Example logging configuration

```
logging:
  level:
    ru:
      codescoring: info
```

#Viewing blocked packages in logs

To find blocked packages in the application logs, ensure the logging level for `ru.codescoring` is set to `info` or lower. The `PolicyLogger` component will output information about blocked packages in the following formats:

- For packages blocked by policies:
  `Policy '<policy_name>' blocked package '<package_name>' versions: [<versions>]`
- For OSA packages blocked by the platform:
  `Policy blocked package '<purl>' for endpoint '<endpoint>': <reason>`

#Logging external requests

External requests to third-party registries can be logged using the `ru.codescoring.proxy.logging.RegistryRequestResponseLogger` component. To do this, set its logging level to `trace`.

Example configuration for logging external requests

```
logging:
  level:

ru.codescoring.proxy.logging.RegistryRequestResponseLogger:
trace
```

#Blocked version handling mode in manifests

The `codescoring.remove-blocked-versions` parameter controls how blocked package versions are displayed in npm, PyPI, and NuGet manifests:

- `true` (default) — blocked versions are **fully removed** from the manifest. The package manager does not see them and will not offer them to the user.
- `false` — blocked versions **remain in the manifest** but are marked as deprecated, with the name of the triggered policy included in the relevant field:
  - **npm** — the `deprecated` field of the version entry contains the policy name;
  - **PyPI** — the `data-yanked` attribute of the package link contains the policy name;
  - **NuGet** — the `deprecation.message` field of the entry contains the policy name, and `listed` is set to `false`.

Example configuration

```
codescoring:
  remove-blocked-versions: false
```

#Buffer size for large manifests

Example buffer size configuration

```
spring:
  http:
    codecs:
      max-in-memory-size: 150MB (this is the default setting,
already included in the application; increase it if you
encounter very large manifests)
```

#Retry policies and circuit breaker for platform requests

#Retry configuration

This configuration defines the retry policy for the `codeScoringApi` service. It is designed to handle temporary failures by retrying up to 3 times.

Retries use an exponential backoff strategy, starting with a 1-second delay and doubling it with each attempt. This policy applies only to certain exceptions, such as `WebClientRequestException`.

#Circuit breaker configuration

The circuit breaker for `codeScoringApi` acts as a fail-fast mechanism. It tracks failure rates, and if they reach 50% (calculated over the last 20 calls),

it “opens” and prevents further requests for 30 seconds. This gives the downstream service time to recover. After the wait period, it switches to “half-open” state, allowing 5 trial calls to check if the service has recovered.

Retry and circuit breaker configuration can be overridden by setting the following properties, for example, for `codeScoringApi`.

#Adding truststore certificates

Example of adding truststore certificates in application.yml

```
spring:
  cloud:
    gateway:
      server:
        webflux:
          httpclient:
            ssl:
              trustedX509Certificates:
                - /usr/local/share/ca-certificates/
codescoring.crt
                - /etc/ssl/certs/ca-certificates.crt
```

#Adding http proxy

Example of http proxy configuration

```
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)'
```

#Redis and caching configuration

To improve performance and reduce load on the CodeScoring platform, caching of policy checking results (Judge service) is supported. Caching requires a Redis connection.

Redis and Caching Settings

```
spring:
  data:
    redis:
      host: localhost
      port: 6379
      database: 0         # Database number (optional)
      password: password  # Optional
      timeout: 2000ms

cache:
  judge:
    enabled: true  # Enable caching (default is false)
    ttl: 24h       # Time-to-live for cache entries
    refresh-after: 30m  # Time after which an entry is
considered stale and needs refresh (but can still be served
from cache)
    proactive-refresh-enabled: true # Enable proactive
(background) cache refresh
    proactive-refresh-interval: 2h  # Interval for background
refresh task
    key-prefix: "cs:judge:" # Prefix for Redis keys
```

Cache TTL prolongation specifics

Proactive refresh does not prolong the TTL (time-to-live) of a cache entry. The TTL is only extended when data is read from the cache by actual user requests. This ensures that only frequently requested components remain in the cache, while rarely used ones are automatically removed from Redis.

#Swagger UI

OSA Proxy provides Swagger UI for API documentation and cache management.

- **URL:** `http://<osa-proxy-host>:<port>/api/swagger`
- **Available operations:**
  - Clear cache by PURL
  - Clear cache by package type

#Supported protocols

This section contains data formats and response modification rules for each supported package manager in OSA Proxy.

#Maven

#Processed files

- `maven-metadata.xml` - version information manifest

- `.jar`, `.war`, `.ear` - package files

#maven-metadata.xml field modifications

```
<metadata>
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <versioning>
    <latest>updated to latest unblocked</latest>
    <release>updated to latest unblocked</release>
    <versions>
      <version>blocked versions are removed</version>
    </versions>
  </versioning>
</metadata>
```

#NPM

#Files processed

- Package JSON manifest (path `/{repository}/*`)
- `.tgz` - package archives

#NPM Manifest field modifications

```
{
  "name": "package-name",
  "dist-tags": {
    "latest": "updated to latest unblocked version"
  },
  "versions": {
    "1.0.0": "blocked versions are removed"
  },
  "time": {
    "1.0.0": "entries for blocked versions are removed"
  }
}
```

#PyPI

#Files and resources

- Simple API HTML pages (path `/{repository}/simple/*`)
- `.zip`, `.tar`, `.tgz`, `.tar.gz`, `.tar.bz2`, `.egg`, `.whl` - package files

#HTML page modifications

- Elements for blocked versions are removed

- URLs are rewritten for proxy downloading

```
<!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

#Handled resources

- `index.json` - service index
- Registration index JSON
- `.nupkg` - package files

#Registration index modifications

```
{
  "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

#Processed files

- Version list (`/@v/list`)
- `.zip` — module archives

#Version list modifications

- Blocked versions are removed from the version list

#Debian

#Processed files

- `.deb` — package files

Debian scanning specifics

For Debian, only package scanning is supported. Modification of manifests (Packages files) is not performed.

#Alpine

#Processed files

- `.apk` — package files

Alpine scanning specifics

For Alpine, only package scanning is supported. Modification of indexes (APKINDEX) is not performed.

#RPM

#Processed files

- `.rpm` — package files

RPM scanning specifics

For RPM, only package scanning is supported. Modification of metadata (repodata) is not performed.

#Docker

#Processed files

- Manifests (v2 API)

#Manifest modification

- Digests of blocked images are removed from multi-architecture manifests (Manifest Lists).

#Behavior in case of complete package blocking

In a scenario where all available versions of a requested package are blocked by security policies, OSA Proxy returns a message indicating that all versions are blocked.

Since some package manager clients might not display this specific blocking message in the user interface, it is recommended to use the `curl` utility for direct diagnosis of the package status. Below are examples of `curl` requests to check the blocking status for various package types:

#Pip

```
curl http://localhost:8080/codescoring-pypi/simple/package_name
```

#Maven

```
curl http://localhost:8080/codescoring-maven/groupid/
artifactid/maven-metadata.xml
```

#npm

```
curl http://localhost:8080/codescoring-npm/package_name
```

#NuGet

Although the NuGet client might display the reason for blocking all packages in the console, a direct `curl` request also allows for status confirmation:

```
curl http://localhost:8080/codescoring-nuget/nuget-api/v3/
registration5-gz-semver2/newtonsoft.json/index.json
```

#Go

```
curl http://localhost:8080/codescoring-go/module_name/@v/list
```

#Metric collection

Metrics are available in **OSA Proxy** at `{osa-proxy-url}/actuator/metrics` in JSON format, as well as in the Prometheus format `{platform-url}/actuator/prometheus`.

These metrics are collected for each repository type (`maven`, `pypi`, `nuget`, `npm`) and allow detailed monitoring of incoming requests to proxy repositories.

#Available metrics

- `gateway_route_<package-type>_requests_seconds_count` – total number of requests processed;
- `gateway_route_<package-type>_requests_seconds_sum` – total request processing time, used to calculate average response time;
- `gateway_route_<package-type>_requests_seconds_max` – maximum request processing time;
- `gateway_route_<package-type>_requests_seconds_bucket` – SLO (Service Level Objective) response time metrics with buckets: 10ms, 25ms, 50ms, 100ms, 250ms, 500ms, 1s, 2s, 5s.

When collecting metrics, `<package-type>` is replaced with the corresponding repository type: `maven`, `pypi`, `nuget`, `npm`, `go`, `debian`, `alpine`, `rpm`, `docker`. For example, for a Maven repository, the metric will be called `gateway_route_maven_requests_total`.

These metrics can be filtered by the following labels:

- **`operation`** – the type of operation performed on the package;
  - `scan_package` – package scanning;
  - `scan_manifest` – manifest scanning;
  - `other` – other operations (e.g., transferring files not subject to analysis).
- **`method`** – HTTP request method (`GET`, `POST`, `PUT`, etc.);
- **`repository`** – the name of the repository to which the request was made;
- **`status`** – HTTP response status code (e.g., `200`, `403`, `500`);
- **`outcome`** – request processing result;
  - `success` – request successfully processed;
  - `error` – an error occurred during processing (status 400 or higher, excluding the blocking code);
  - `blocked_by_policies` – the request was blocked by security policies.

#CodeScoring API Metrics

To monitor interaction with the CodeScoring platform, the following metrics are available:

- `codescoring_api_requests_seconds_count` – total number of requests to the CodeScoring API;
- `codescoring_api_requests_seconds_sum` – total execution time of requests to the API;
- `codescoring_api_requests_seconds_max` – maximum execution time of a request to the API;
- `codescoring_api_requests_seconds_bucket` – SLO metrics for API response time with buckets: 10ms, 25ms, 50ms, 100ms, 250ms, 500ms, 1s, 2s, 5s.

These metrics allow tracking:

- Performance of interaction with the CodeScoring platform
- Number of component scan requests
- API response time to identify connection issues
- Load on the platform from OSA Proxy

#Base64 URL Parameters

#Using Base64 URL Parameters for `osa-proxy`

In certain scenarios, interacting with `osa-proxy` requires explicitly specifying additional parameters in the URL path. This is achieved by encoding the required information in Base64 (URL-safe) format.

The primary purpose of using Base64-encoded parameters is to provide `osa-proxy` with the necessary context to correctly apply security policies, especially when `osa-proxy` acts as an intermediary for external repositories.

#Automatic Context Detection

When `osa-proxy` is placed between a client (package manager) and an internal repository manager (e.g., JFrog Artifactory or Nexus Repository Manager), `osa-proxy` can automatically extract the host and repository name from the downstream repository settings.

Example configuration where context is automatically determined:

```
npm:
  repository:
    - name: codescoring-npm
      # ...
      registry: https://nexus.test.ru/repository/npm-proxy
```

#Explicit Context Specification via Base64 Parameters

In cases where `osa-proxy` directly interacts with external, public repositories (e.g., `https://registry.npmjs.org`), it cannot independently retrieve information about the internal host and repository name. In this situation, it is critical for `osa-proxy` to receive this data to apply linked security policies and process the request correctly.

For this purpose, a Base64-encoded string is used, which contains a JSON object with parameters such as `repoManagerHost` and `repoName`. This string is embedded directly into the request URL, allowing `osa-proxy` to obtain the necessary context.

Example configuration requiring explicit context specification:

```
npm:
  repository:
    - name: codescoring-npm
      # ...
      registry: https://registry.npmjs.org # Parameter passing
is required here
```

#How it works:

The Base64-encoded parameter string is placed in the URL path immediately after the repository name. `osa-proxy` decodes this string, extracts the parameters, and uses them to perform its functions, including applying security policies associated with a specific internal repository.

General URL Structure:
`https://<osaproxy-host>/<repository-name>/<base64-parameters>/<rest-of-path>`

#Passing Context for Correct Operation of Security Policies Linked to Nexus and Artifactory Repositories

`Developer Client` -> `Nexus / Artifactory` -> `osa-proxy` -> `Internet`

To pass contextual information, including the host and repository name of your repository manager, this data should be integrated into a Base64-encoded parameter string. It is important to strictly adhere to the rule that this Base64 string must be placed immediately after the repository name in the URL.

#Configuration Update

It is necessary to mark the repository as compatible with Base64 parameters using `url-encoded-config: true`.

```
npm:
  repository:
    - name: codescoring-npm
      url-encoded-config: true
      # ...
      registry: https://registry.npmjs.org
```

#Nexus

1. Go to **Server Administration** -> **Repositories**.
2. Select the desired type (e.g., `maven2 (proxy)`).
3. In the **Remote storage** field, enter the URL of your `osa-proxy` instance, including the repository name and Base64-encoded parameters.

Example for a Maven proxy repository:
`https://osaproxy.example.com/internet-maven/eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL3JlcG8xLm1hdmVuLm9yZy9tYXZlbjIiLCJyZXBvTmFtZSI6ImludGVybmV0LW1hdmVuIn0/maven2`

#Artifactory

1. Go to **Administration** -> **Repositories** -> **Remote**.
2. In the configuration, set the **URL** field to the `osa-proxy` URL. This URL must include the repository name and the Base64-encoded string.

Example for a remote PyPI repository:
`https://osaproxy.example.com/internet-pypi/eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL3B5cGkub3JnL3NpbXBsZSIsInJlcG9OYW1lIjoiaW50ZXJuZXQtcHlwaSJ9/simple`

#Rule

The Base64-encoded parameter string must be placed in the URL path immediately after the repository name.

The general URL structure is as follows:
`https://<osaproxy-host>/<repository-name>/<base64-parameters>/<rest-of-path>`

Where:

- `<osaproxy-host>`: The hostname of the `osa-proxy` instance.
- `<repository-name>`: The name of the repository being accessed.
- `<base64-parameters>`: A URL-safe Base64-encoded JSON string containing the parameters.
- `<rest-of-path>`: The remaining part of the path from the package manager settings.

#Example

For example, to pass the following parameters as a JSON object:

```
{"repoManagerHost":"https://nexus.test.ru","repoName":"npm-
proxy"}
```

To do this, you should:

1. **Convert the JSON object to a string.**
2. **Encode the string using URL-safe Base64.**

The result of encoding the above JSON object in Base64:
`eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZXBv
TmFtZSI6Im5wbS1wcm94eSJ9`

#Configuring Package Managers

To permanently use the URL with Base64 parameters for all requests, update your package manager's configuration file.

#NPM

For NPM, edit `.npmrc` file and set the `registry` key.

The URL should include the repository name and the Base64 encoded string.

```
registry=https://osaproxy.example.com/npm-proxy/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6Im5wbS1wcm94eSJ9
```

#Maven

For Maven, edit `settings.xml` file. You can add a new `<mirror>` in the `<mirrors>` section.

The `<url>` tag should contain the full URL, including the repository name and the Base64 encoded string.

```
<settings>
  ...
  <mirrors>
    <mirror>
      <id>osa-proxy-mirror</id>
      <mirrorOf>*</mirrorOf>
      <url>https://osaproxy.example.com/my-maven-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6Im5wbS1wcm94eSJ9/maven2</url>
    </mirror>
```

```
  </mirrors>
  ...
</settings>
```

Make sure the <mirrorOf> value matches the repositories you want to proxy.

#Go

For Go, set the GOPROXY environment variable to include the repository name and the Base64-encoded string.

```
export GOPROXY="https://osaproxy.example.com/go-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6ImdvLXJlcG8ifQ"
```

#Debian

For Debian, edit /etc/apt/sources.list or a file in /etc/apt/sources.list.d/. Update the URIs field.

```
Types: deb
URIs: https://osaproxy.example.com/debian-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6ImRlYmlhbi1yZXBvIn0=
Suites: stable
Components: main
Signed-By: /path/to/key.gpg
```

#NuGet

For NuGet, edit NuGet.config file and add a new package source. The value attribute of the <add> tag should contain the full URL.

```
<configuration>
  <packageSources>
    <add key="osa-proxy" value="https://osaproxy.example.com/
nuget-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6Im51Z2V0LXJlcG8ifQ/index.json" />
  </packageSources>
  ...
</configuration>
```

#PyPI

For PyPI, edit pip.conf (Linux/macOS) or pip.ini (Windows) file and set the index-url.

```
[global]
index-url = https://osaproxy.example.com/pypi-repo/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLnRlc3QucnUiLCJyZX
BvTmFtZSI6InB5cGktcmVwbyJ9/simple
```

#Maven configuration

#Repository URL migration

Use case: migrating a Maven repository from Artifactory to OSA Proxy.

The following table provides a summary of repository URL redirection for Maven. Authentication parameters and other configurations, such as username and password, remain unchanged.

Source	settings.xml URL before migration	settings.xml URL after migration	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`
Official Repository	`https://repo.maven.apache.org/maven2`	`https://{osa-proxy-url}/inet-mvn`	`https://repo.maven.apache.org/maven2`

#Maven registry migration

Original `.m2/settings.xml` file:

```
<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>
```

The following repository definition needs to be added to the service's YAML configuration (the `application.yml` file) under the maven section. A service restart is required to apply the changes.

Configuration in `application.yml` file

```
maven:
  enabled: true
  repository:
    - name: jfrog-mvn
      scan-manifest: false
      scan-package: true
      registry: https://jfrog.host.ru/artifactory/maven-remote
```

Updated `.m2/settings.xml` file:

```
<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>
```

#NPM Configuration

#Repository URL migration

Use Case: Migrating an `npm` repository from Artifactory to OSA Proxy.

The following table provides a summary of repository URL redirection for NPM. Authentication parameters and other configurations, such as username and password, remain unchanged.

Source	.npmrc registry: Before migration	.npmrc registry: After migration	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`
Official Repository	`https://registry.npmjs.org`	`https://{osa-proxy-url}/inet-npm`	`https://registry.npmjs.org`

#NPM registry migration

Original `.npmrc` file:

```
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
```

The following repository definition needs to be added to the service's YAML configuration (the application.yml file) under the npm section. A service restart is required to apply the changes.

Configuration in `application.yml` file

```
npm:
  enabled: true
  repository:
    - name: arti-npm
```

```
      scan-package: true
      scan-manifest: true
      registry: https://artifactory.domain.ru/artifactory/api/
npm/npm-remote/
```

Updated .npmrc file:

```
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
```

#NuGet Configuration

#Repository URL migration

Use case: migrating a NuGet repository from Artifactory to OSA Proxy.

The following table provides a summary of repository URL redirection for NuGet. Authentication parameters and other configurations, such as username and password, remain unchanged.

Source	NuGet.config URL before migration	NuGet.config URL after migration	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	`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`
Official Repository	`https://api.nuget.org/v3/index.json`	`https://{osa-proxy-url}/inet-nuget/nuget-api/v3/index.json`	`https://api.nuget.org`

#NuGet registry migration

Original `NuGet.config` file:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <add key="Artifactory" value="https://jfrog.host.ru/
artifactory/api/nuget/v3/nuget-safe" protocolVersion="3"/>
  </packageSources>
  <packageSourceCredentials>
    <Artifactory>
      <add key="Username" value="your-username" />
      <add key="ClearTextPassword" value="your-password" />
    </Artifactory>
  </packageSourceCredentials>
</configuration>
```

The following repository definition needs to be added to the service's YAML configuration (the `application.yml` file) under the nuget section. A service restart is required to apply the changes.

Configuration in `application.yml` file

```
nuget:
  enabled: true
  repository:
    - name: arti-nuget
      scan-package: true
      scan-manifest: true
      registry: https://jfrog.host.ru/artifactory/api/nuget/v3/
nuget-safe
```

Updated `NuGet.config` file:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="OSA-Proxy" value="https://{osa-proxy-url}/arti-
nuget/nuget-api" protocolVersion="3"/>
  </packageSources>
  <packageSourceCredentials>
    <OSA-Proxy>
      <add key="Username" value="your-username" />
      <add key="ClearTextPassword" value="your-password" />
    </OSA-Proxy>
  </packageSourceCredentials>
</configuration>
```

#PyPI Configuration

#Repository URL migration

Use case: migrating a PyPI repository from Artifactory to OSA Proxy.

The following table provides a summary of repository URL redirection for PyPI. Authentication parameters and other configurations, such as username and password, remain unchanged.

Source	pip.conf / pip.ini URL before migration	pip.conf / pip.ini URL after migration	application.yml pypi.repository.registry
Nexus	`https://nexus.host.ru/repository/pip-remote`	`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`
Official Repository	`https://pypi.org/simple`	`https://{osa-proxy-url}/inet-pypi/simple`	`https://pypi.org`

#PyPI registry migration

Original `pip.conf` (Linux/macOS) or `pip.ini` (Windows) file:

```
[global]
index-url = https://jfrog.host.ru/artifactory/api/pypi/pypi-remote/simple
trusted-host = jfrog.host.ru

[install]
trusted-host = jfrog.host.ru
```

Or with authentication:

```
[global]
index-url = https://username:password@jfrog.host.ru/artifactory/api/pypi/pypi-remote/simple
```

```
trusted-host = jfrog.host.ru

[install]
trusted-host = jfrog.host.ru
```

The following repository definition needs to be added to the service's YAML configuration (the `application.yml` file) under the pypi section. A service restart is required to apply the changes.

Configuration in `application.yml` file

```
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
```

GitLab configuration example in `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
```

Updated `pip.conf` (Linux/macOS) or `pip.ini` (Windows) file:

```
[global]
index-url = https://{osa-proxy-url}/jfrog-pypi
trusted-host = {osa-proxy-url}

[install]
trusted-host = {osa-proxy-url}
```

Or with authentication:

```
[global]
index-url = https://username:password@{osa-proxy-url}/jfrog-
pypi
```

```
trusted-host = {osa-proxy-url}

[install]
trusted-host = {osa-proxy-url}
```

#Multiple package registries

Some PyPI repositories may serve packages from multiple hosts. For example, the `download.pytorch.org` index contains links to both its own CDN hosts and the standard `files.pythonhosted.org`.

To correctly proxy such repositories, use the `additional-packages-registries` parameter — a map where the key specifies the source host and the value specifies the package registry URL to route requests to.

PyTorch configuration example:

```
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
```

#Configuration file locations

- **Linux/macOS**: `~/.config/pip/pip.conf` or `~/.pip/pip.conf`
- **Windows**: `%APPDATA%\pip\pip.ini` or `%HOME%\pip\pip.ini`
- **Per-virtualenv**: `$VIRTUAL_ENV/pip.conf`

#Go Configuration

#Go Proxy Migration

Use case: Migrating Go to use an OSA Proxy instead of direct access or external public proxies.

The following table summarizes URL redirection for the Go proxy. Authentication parameters and other settings (if applicable, e.g., for private repositories requiring specific credentials) should be configured separately in accordance with your corporate policies (e.g., via `.netrc` or SSH keys).

Module Source / Repository	GOPROXY before migration	GOPROXY after migration
Nexus	`https://nexus.host.ru/repository/go-remote`	`https://{osa-proxy-url}/nexus-go`
Artifactory	`https://jfrog.host.ru/artifactory/api/go/go-virt`	`https://{osa-proxy-url}/arti-go`
Official Go Proxy	`https://proxy.golang.org`	`https://{osa-proxy-url}/inet-go`

Checksum Database (sum.golang.org)

The Checksum DB is not a separate GOPROXY endpoint. It is configured via the GOSUMDB environment variable. See the section below for details.

#Go Proxy Migration Details

#Environment Configuration Before Migration

Before migration, your GOPROXY might have been set to the public Go proxy (`https://proxy.golang.org`) or not set at all, which would default to using `proxy.golang.org`. The following repository definition needs to be added to the service's YAML configuration (the `application.yml` file) in the go section. A service restart is required for the changes to take effect.

Configuration in `application.yml` file

```
go:
  enabled: true
  repository:
    - name: inet-go
      scan-package: true
      scan-manifest: true
      registry: https://proxy.golang.org
      sumdb-registry: https://sum.golang.org
```

Example of current environment variable configuration (e.g., in `.bashrc`, `.zshrc` file, or in a CI/CD pipeline):

```
export GOPROXY=https://{osa-proxy-url}/inet-go
```

#Checksum Database Configuration

To proxy requests to `sum.golang.org` through OSA Proxy, use the GOSUMDB environment variable. Its value follows the format `<db-name> <proxy-url>`,

where the URL is constructed as `{osa-proxy-url}/{repo-name}/sumdb/sum.golang.org`:

```
export GOSUMDB="sum.golang.org https://{osa-proxy-url}/inet-go/
sumdb/sum.golang.org"
```

Full usage example:

```
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
```

#Debian Package Configuration

#Repository URL Migration

Use case: Migration of Debian repositories from direct sources to an OSA Proxy server.

The following table summarizes the redirection of repository URLs for Debian. Please note that the format of the deb or `deb-src` repository lines (distribution, components) remains unchanged; only the base repository URL changes.

Source	URL in `sources.list` before migration	URL in `sources.list` after migration	`application.yml` `apt.repository.registry`
Nexus	`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`
Official Debian	`https://deb.debian.org/debian/`	`https://{osa-proxy-url}/inet-debian`	`http://deb.debian.org/debian/`

#APT Repository Migration

Original file `/etc/apt/sources.list` or `/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
```

The following repository definition needs to be added to the service's YAML configuration (the `application.yml` file) in the debian section. The service must be restarted for the changes to take effect.

Configuration in `application.yml` file

```
debian:
  enabled: true
  repository:
    - name: debian-apt
      scan-package: true
      distro: bullseye
      registry: http://deb.debian.org/debian/
```

After configuring the proxy server and adding it to `application.yml`, your `sources.list` will look like this:

```
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
```

#Docker Registry Configuration

#Registry URL Migration

Use case: Migration of Docker registries from direct sources to an OSA Proxy server.

The following table summarizes the redirection of URLs for Docker.

Source	URL before migration	URL after migration	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

#Docker Client Migration

The following repository definition needs to be added to the service's YAML configuration (the `application.yml` file) in the docker section. The service must be restarted for the changes to take effect.

Configuration in `application.yml` file

```
docker:
  enabled: true
  repository:
    - name: codescoring-docker
      scan-package: true
      registry: https://registry-1.docker.io
      auth-token-url: https://auth.docker.io
```

After configuring the proxy server and adding it to `application.yml`, the command to pull an image will look like this:

```
docker pull {osa-proxy-url}/library/alpine:latest
```

#Using subdomains for access

If you want to use more than one Docker repository, you need to enable subdomains. The subdomain names must match the repository names specified in the `docker.repository` configuration.

In this case, the command to pull an image will look like this:

```
docker pull codescoring-docker.osaproxyhost.ru/library/postgres
```

For a single repository, there is no need to add a subdomain — Docker will be available directly via the OSA Proxy host:

```
docker pull osaproxyhost.ru/library/postgres
```

#Alpine Package Configuration

#Repository URL Migration

Use case: Migration of Alpine repositories from direct sources to an OSA Proxy server.

The following table summarizes the redirection of repository URLs for Alpine.

Source	URL in repositories before migration	URL in repositories after migration	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`
Official Alpine	`https://dl-cdn.alpinelinux.org/alpine`	`https://{osa-proxy-url}/inet-alpine`	`https://dl-cdn.alpinelinux.org/alpine`

#APK Repository Migration

Original file `/etc/apk/repositories`:

```
https://dl-cdn.alpinelinux.org/alpine
```

The following repository definition needs to be added to the service's YAML configuration (the `application.yml` file) in the alpine section. The service must be restarted for the changes to take effect.

Configuration in `application.yml` file

```
alpine:
  enabled: true
```

```
  repository:
    - name: codescoring-alpine
      scan-package: true
      registry: https://dl-cdn.alpinelinux.org/alpine
```

After configuring the proxy server and adding it to `application.yml`, your repositories file will look like this:

```
https://{osa-proxy-url}/codescoring-alpine
```

#RPM Package Configuration

#Repository URL Migration

Use case: Migration of RPM repositories (YUM/DNF) from direct sources to an OSA Proxy server.

The following table summarizes the redirection of repository URLs for RPM.

Source	`baseurl` in `.repo` before migration	`baseurl` in `.repo` after migration	`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`
Official Mirror	`https://repo.almalinux.org/almalinux`	`https://{osa-proxy-url}/inet-rpm`	`https://repo.almalinux.org/almalinux`

#YUM/DNF Repository Migration

Original file `/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
```

The following repository definition needs to be added to the service's YAML configuration (the `application.yml` file) in the rpm section. The service must be restarted for the changes to take effect.

Configuration in `application.yml` file

```
rpm:
  enabled: true
  repository:
    - name: codescoring-rpm
      scan-package: true
      registry: https://repo.almalinux.org/almalinux
```

After configuring the proxy server and adding it to `application.yml`, the repository configuration will look like this:

```
[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 is the Go version of the proxy service for package registries with CodeScoring OSA checks. The service accepts package manager requests, proxies them to upstream registries, checks manifests and downloaded artifacts, blocks forbidden components, and exposes operational endpoints for health checks, metrics, Swagger UI, and cache administration.

#Supported ecosystems

OSA Proxy Go supports the following registry types:

- npm;
- Composer;
- Maven;
- NuGet;
- PyPI;
- RubyGems;
- Go modules;
- Debian;
- Alpine;

- RPM;
- Docker Registry API v2.

::::note Alternative repositories

OSA Proxy Go can work not only with public registries, but also with repository managers that implement the corresponding ecosystem protocols, for example Sonatype Nexus Repository, JFrog Artifactory, or CodeScoring.Save. ::::

#Main functionality

#Manifest and package scanning

For supported ecosystems, two check levels are available:

- **manifest scanning** — analyzes package metadata/indexes and removes versions blocked by policies from responses returned to package managers;
- **package scanning** — checks downloaded archives, binary packages, or images before they are passed to the client.

Support depends on the ecosystem. For example, npm, Maven, NuGet, PyPI, Go, Composer, and RubyGems support metadata and package checks, while Debian, Alpine, and RPM support downloaded package checks without system index modification.

#Blocking unsafe components

If a component violates security policies, OSA Proxy Go can remove unsafe versions from metadata, block artifact downloads, and return a configurable block HTTP status code.

#Response modification

When manifest scanning is enabled, the service modifies upstream registry responses: removes blocked versions, rewrites download links through the proxy, and preserves the response format expected by the package manager.

#Verdict caching

To reduce CodeScoring load and speed up repeated requests, OSA Proxy Go supports Redis caching for Judge check results. The cache is disabled by default and configured in the cache section.

#Routes

For all ecosystems except Docker, the route name is taken from the `name` field in the `repository` section of `osa-proxy.yml`.

Registry type	Route form
npm, Composer, Maven, NuGet, PyPI, Ruby, Go, Debian, Alpine, RPM	`GET /{repository-name}/{path...}`
Docker	`/v2/{path...}` and `GET /token`

For example, an npm repository named `npm` is available at:

```
https://osa-proxy.example.com/npm/
```

Docker mode uses standard Docker Registry API v2 endpoints and does not add the repository name to the path:

```
docker pull osa-proxy.example.com/library/alpine:latest
```

#Operational endpoints

Endpoint	Purpose
`GET /healthz`	Checks that the OSA Proxy Go process is running.
`GET /metrics`	Prometheus metrics.
`GET /api/v3/api-docs`	OpenAPI JSON.
`GET /api/swagger/`	Swagger UI.
`DELETE /api/cache/purls`	Deletes specific PURLs from the verdict cache.
`DELETE /api/cache/packages/{packageType}`	Deletes cache records by package type, package name, or repository context.

#Work modes

The `work-mode` parameter controls check behavior. It can be set globally in `codescoring.work-mode` and overridden for a specific repository with `repository[*].work-mode`.

- `warmup` — warms up the cache without blocking components;
- `spectator` — warms up the cache and records request results without blocking;
- `moderate` — blocks by policies, but allows unscanned components;

- `strict` — blocks by policies and blocks unscanned components;
- `strict_wait` — blocks by policies and waits for checks of unscanned components.

#Service deployment

OSA Proxy Go uses the `osa-proxy.yml` configuration file. By default, the service looks for it in the working directory, but the path can be passed as the first startup argument or through the `OSA_PROXY_CONFIG_PATH` environment variable.

#Docker

Example container run with an external configuration file:

```
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>
```

Availability check:

```
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
```

If Redis verdict cache is enabled, add Redis to the Compose file and set its address in `cache.redis.address`.

.env file

When running with Docker Compose, `.env` is passed into the container only when it is listed in `env_file`. These variables can be used:

- directly as process environment variables;
- in `osa-proxy.yml` through `${VAR_NAME:default_value}` placeholders.

In the Compose example, the configuration path is set separately through `environment.OSA_PROXY_CONFIG_PATH`, so do not duplicate it in `.env`.

Example `.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
```

Example usage in `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}
```

Store secrets in `.env`, not as literal values in `osa-proxy.yml`.

#Proxy settings for outgoing HTTP requests

OSA Proxy Go uses standard Go HTTP clients. They automatically respect the `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` environment variables; lowercase variants `http_proxy`, `https_proxy`, and `no_proxy` can also be used.

Example `.env` for a corporate 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` should include addresses that the service must reach directly: local addresses, Redis, internal Kubernetes/Docker DNS names, internal CodeScoring domains, or internal package registries that should not go through the corporate proxy.

#Helm

Minimal `values.yaml` example:

```
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
```

After installation, check the endpoints:

```
curl https://osa-proxy.example.com/healthz
curl https://osa-proxy.example.com/metrics
```

#Service configuration

OSA Proxy Go is configured with the `osa-proxy.yml` file. The example below shows a typical working configuration with several ecosystems, CodeScoring settings, HTTP client settings, Redis cache, and logging.

#Configuration example

```
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
```

#codescoring section

Parameter	Purpose
`url`	CodeScoring platform URL.
`token`	CodeScoring access token.
`work-mode`	Global work mode when it is not overridden at repository level.
`osa-proxy-url`	External OSA Proxy Go URL used when building links and responses.
`enable-status-line`	Adds the block reason to the HTTP/1.1 status line when the client displays it.
`block-on-codescoring-errors`	Blocks downloads on CodeScoring errors or scan errors.
`remove-blocked-versions`	Removes blocked versions from manifests. If `false`, versions remain marked as blocked where the format supports it.
`block-status-code`	HTTP status code for blocked downloads. The default is `403`.
`judge-concurrency`	Number of parallel Judge requests. Used to limit load on Judge when checking large version lists and during background cache refresh.
`resilience.retry`	Retry settings for CodeScoring requests.
`resilience.circuit-breaker`	Circuit breaker settings for temporary degradation of external calls.

#Package manager sections

Each ecosystem contains the `enabled` flag and a `repository` list. The repository name becomes part of the OSA Proxy Go URL:

```
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
```

This repository is available at:

```
https://osa-proxy.example.com/company-npm/
```

The `scan-manifest` and `scan-package` fields enable checks for manifests and downloaded artifacts. Support for these modes depends on the ecosystem; see Supported protocols. Repository-level `work-mode` overrides global `codescoring.work-mode`.

#Ecosystem-specific options

`composer` and `pypi` support `packages-registry` and `additional-packages-registries` when artifacts are downloaded from separate hosts. `go` uses `sumdb-registry` when SumDB proxying is required. Docker uses `auth-token-url`.

#Verdict cache

Redis cache is disabled by default:

```
cache:
  judge:
    enabled: false
  redis:
    address: redis:6379
```

To enable caching:

```
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
```

#Logging

The log level is configured with `logging.level`. Supported values are `debug`, `info`, `warn`, and `error`.

```
logging:
  level: info
```

#Parameter reference

#Root sections

Parameter	Purpose
`pypi`	PyPI repository settings.
`maven`	Maven repository settings.
`nuget`	NuGet repository settings.
`npm`	npm repository settings.
`composer`	Composer/Packagist repository settings.
`ruby`	RubyGems repository settings.
`go`	Go module proxy settings.
`debian`	Debian repository settings.
`alpine`	Alpine APK repository settings.
`rpm`	RPM/YUM/DNF repository settings.
`docker`	Docker Registry API v2 settings.
`codescoring`	CodeScoring connection and check behavior.

Parameter	Purpose
`http`	HTTP server and HTTP client timeouts and limits.
`cache`	Redis Judge verdict cache.
`logging`	Service log level.

#Common package manager parameters

Parameter	Available in	Purpose
`enabled`	All package managers	Registers routes for the ecosystem. If `false`, repositories from this section are not served.
`repository`	All package managers	List of upstream repositories for the ecosystem.
`repository[*].name`	All package managers	Repository name. For non-Docker ecosystems it becomes the first URL segment: `/{name}/...`. Must be unique among enabled routes.
`repository[*].registry`	All package managers	Upstream registry URL where OSA Proxy Go proxies requests.
`repository[*].work-mode`	All package managers	Work mode for a specific repository. If empty, `codescoring.work-mode` is used.
`repository[*].scan-manifest`	`npm`, `composer`, `maven`, `nuget`, `pypi`, `ruby`, `go`	Enables manifest/metadata checking and modification.
`repository[*].scan-package`	All except `docker`	Enables checks for downloaded package files. For `docker`, image checks are handled by Docker Registry proxy logic.
`repository[*].url-encoded-config`	All except `docker`	Enables URL-safe Base64 context in the path for Nexus/JFrog scenarios and repository-context policy application.
	All except `docker`	Limits which files are sent to package scanning by extension.

Parameter	Available in	Purpose
`repository[*].file-type-filter`		If the parameter is omitted or disabled, filtering is not applied.

#Repository-specific parameters

Parameter	Available in	Purpose
`packages-registry`	`pypi`, `composer`	Base URL of a separate host used to download package files when it differs from the metadata registry.
`additional-packages-registries`	`pypi`, `composer`	Additional host -> registry map for packages that publish artifacts on multiple domains. Useful for correct routing of external package hosts.
`sumdb-registry`	`go`	Go checksum database URL, for example `https://sum.golang.org`, when SumDB requests should go through OSA Proxy Go.
`distro`	`debian`, `alpine`	Distribution or branch name used when processing metadata and package paths.
`auth-token-url`	`docker`	Docker Registry token service URL. For Docker Hub this is usually `https://auth.docker.io`; for JFrog/Nexus use the corresponding repository manager endpoint.

#file-type-filter

Parameter	Purpose
`additional-allowed-extensions`	YAML array of strings. The presence of this field enables the filter: after that, only extensions from the built-in preset and from `additional-allowed-extensions` pass, while other unknown extensions are blocked. Values can be written with or without the leading dot; they are normalized to lowercase and to the dotted form.
`scanned-extensions`	YAML array of strings. Enables the filter and makes the handler treat files with these extensions as package artifacts for scanning and short-lived scan-result caching. Values can be written with or without the leading dot.

The filter applies only to non-Docker repositories. If the `file-type-filter` section is omitted or set to `{}`, the filter is disabled: OSA Proxy Go behaves as if the filter did not exist, so requests follow the normal handler rules and compatibility with previous behavior is preserved. To enable the filter, add at least one of `additional-allowed-extensions` / `scanned-extensions`. The presence of the key is what matters: for example, `additional-allowed-extensions: []` enables the filter but does not add extensions beyond the built-in preset.

When the filter is enabled, OSA Proxy Go:

- lets metadata/manifest requests pass without extension checks;
- extracts the filename from the URL path, decodes URL-encoded characters, and compares extensions case-insensitively;
- allows a file when its extension is included in the built-in ecosystem preset or in `additional-allowed-extensions`;
- allows checksum and signature sidecar files (`.metadata`, `.sha256`, `.sha512`, `.sha1`, `.asc`, `.md5`) only when the base artifact is also allowed;
- immediately blocks all other package file requests before contacting the upstream registry or CodeScoring.

Built-in presets:

Ecosystem	Allowed extensions
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
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

For Debian, source tarballs matching `.orig-*.tar.gz`, `.orig-*.tar.xz`, and `.orig-*.tar.bz2` are also allowed.

`additional-allowed-extensions` only extends the filter allow-list. The presence of this field switches the repository to allow-list mode: OSA Proxy Go allows the built-in ecosystem extensions and the extensions from `additional-allowed-extensions`, and blocks all other unknown package file extensions. Use it when a repository contains acceptable files with non-standard extensions and the filter must not block them. This parameter does not make the handler send those files to package scanning: if the ecosystem's default strategy does not treat the extension as scannable, the request continues as normal passthrough. To make a new file type participate in package scanning, also add the extension to `scanned-extensions`.

`scanned-extensions` controls the second behavior: files with these extensions are treated as scannable package artifacts even if the ecosystem's default strategy does not recognize them. For these extensions, a short-lived scan-result cache is enabled so related files with the same base name can reuse one verdict. For example, Maven can use `scanned-extensions: [.jar, .pom]` so `demo-1.0.0.jar` and `demo-1.0.0.pom` are grouped by the `demo-1.0.0` base name.

Example:

```
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]
```

In this example, `.tgz` is allowed by the npm preset and participates in package scanning, while `.license` is additionally allowed by the filter but does not become a scannable artifact.

#npm behavior example

Without the `file-type-filter` section, the filter is disabled. The npm handler follows its standard logic: the package tarball `left-pad-1.0.0.tgz` is sent to package scanning, while other requests are handled as metadata or passthrough depending on the route.

```
npm:
  enabled: true
  repository:
    - name: npm
```

```
      registry: https://registry.npmjs.org
      scan-package: true
```

An empty section also keeps the filter disabled:

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-package: true
      file-type-filter: {}
```

To enable the filter without adding new extensions, set an empty list. For npm, only the built-in `.tgz` preset and sidecar files for allowed artifacts are allowed. A request for `left-pad-1.0.0.tgz` passes and is checked, while `left-pad-1.0.0.exe` is blocked before the upstream registry and CodeScoring are contacted.

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-package: true
      file-type-filter:
        additional-allowed-extensions: []
```

To allow a non-standard file without sending it to package scanning, add the extension only to `additional-allowed-extensions`:

```
npm:
  enabled: true
  repository:
    - name: npm
      registry: https://registry.npmjs.org
      scan-package: true
      file-type-filter:
        additional-allowed-extensions: [license]
```

With this configuration, `.tgz` is scanned as an npm package, `.license` passes the filter as an allowed file, and `.exe` is blocked by the filter.

#codescoring

Parameter	Default	Purpose
`url`	Required	CodeScoring platform URL.
`token`	Required	CodeScoring API access token.
`work-mode`	Not set	Global work mode: `warmup`, `spectator`, `moderate`, `strict`, `strict_wait`.
`osa-proxy-url`	Not set	External OSA Proxy Go URL. Used when generating links and rewriting URLs in responses.
`enable-status-line`	`false`	Adds the block reason to the HTTP/1.1 status line. Does not affect HTTP/2 or HTTP/3; Docker clients read the JSON body.
`block-on-codescoring-errors`	`false`	Blocks downloads when CodeScoring returns an error or a package cannot be checked.
`remove-blocked-versions`	`true`	Removes blocked versions from manifests. If `false`, versions remain in metadata with a block marker when the format supports it.
`legacy-judge`	`false`	Enables compatibility with Judge versions before `2026.20.0`. Use only for CodeScoring installations with an older Judge service version.
`stage`	Not set	Stage/context value passed to CodeScoring checks. Usually `proxy`.
`block-status-code`	`403`	HTTP status code returned for blocked packages.
`judge-concurrency`	`10`	Limits the number of parallel Judge requests. Lower values make OSA Proxy Go send fewer concurrent requests to Judge when checking large version lists and during background cache refresh. Versions are sent in chunks of up to `900` PURLs, so this parameter becomes visible when a single package has

Parameter	Default	Purpose
		more than 900 versions and the check is split into multiple requests.
resilience	See below	Resilience settings for CodeScoring requests.

#codescoring.resilience.retry

Parameter	Default	Purpose
max-attempts	3	Maximum number of request attempts.
wait-duration	1s	Delay between attempts.
exponential-backoff-multiplier	2	Exponential backoff multiplier for increasing retry delays.

#codescoring.resilience.circuit-breaker

Parameter	Default	Purpose
failure-rate-threshold	50	Error percentage that opens the circuit breaker.
minimum-number-of-calls	10	Minimum number of calls required to calculate the error rate.
sliding-window-size	20	Window size used for error statistics.
wait-duration-in-open-state	30s	Time before moving from open to half-open state.
permitted-number-of-calls-in-half-open-state	5	Number of trial requests allowed in half-open state.

#http.server

Parameter	Default	Purpose
`read-timeout`	`2m`	Maximum time for reading the whole incoming request.
`read-header-timeout`	`5s`	Maximum time for reading HTTP headers.
`idle-timeout`	`120s`	Time to keep an idle keep-alive connection.
`shutdown-timeout`	`10s`	Graceful shutdown timeout.

#http.client

Parameter	Default	Purpose
`connection-timeout`	`10s`	Timeout for establishing connections to upstream registries and CodeScoring.
`response-timeout`	`30s`	Response wait timeout.
`max-manifest-body-size`	`200mb`	Maximum manifest/metadata response body size the service will process. Values such as `200mb` are supported.
`max-idle-conns`	`100`	Maximum number of idle HTTP connections.
`max-idle-conns-per-host`	`10`	Maximum number of idle HTTP connections per host.
`idle-conn-timeout`	`90s`	Lifetime of an idle connection in the HTTP client.

#cache.judge

Parameter	Default	Purpose
`enabled`	`false`	Enables Redis cache for Judge check results.

Parameter	Default	Purpose
`ttl`	`24h`	Cache entry lifetime.
`refresh-after`	`30m`	Entry age after which it can be refreshed in the background.
`proactive-refresh-enabled`	`false`	Enables proactive background refresh for stale entries.
`proactive-refresh-interval`	`2h`	Background refresh interval.
`proactive-refresh-workers`	`10`	Number of background refresh workers.
`key-prefix`	Not set	Redis key prefix, for example `cs:judge:`.

#cache.redis

Parameter	Purpose
`address`	Redis address in `host:port` format.
`password`	Redis password.
`db`	Redis database number.

#logging

Parameter	Default	Purpose
`level`	`info`	Log level: `debug`, `info`, `warn`, `warning`, `error`. Unknown values are treated as `info`.

#Redis and Caching Configuration

OSA Proxy Go supports Redis caching for Judge verdicts to speed up repeated requests and reduce CodeScoring load. The cache is disabled by default.

```
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
```

#Parameters

Parameter	Purpose
`cache.judge.enabled`	Enables Redis cache for Judge check results.
`cache.judge.ttl`	Cache entry lifetime. Default is `24h`.
`cache.judge.refresh-after`	Entry age after which it can be refreshed in the background. Default is `30m`.
`cache.judge.proactive-refresh-enabled`	Enables periodic background refresh for stale entries. Default is `false`.
`cache.judge.proactive-refresh-interval`	Background refresh interval. Default is `2h`.
`cache.judge.proactive-refresh-workers`	Number of background refresh workers. Default is `10`.
`cache.judge.key-prefix`	Redis key prefix.
`cache.redis.address`	Redis address in `host:port` format.
`cache.redis.password`	Redis password.
`cache.redis.db`	Redis database number.

::::note TTL and background refresh

Background refresh does not extend an entry TTL by itself. TTL is extended when data is read from the cache by real requests, so rarely used entries are eventually removed from Redis. ::::

#Cache Management

The operational API is available through Swagger UI:

```
https://osa-proxy.example.com/api/swagger/
```

Main operations:

- `DELETE /api/cache/purls` — delete specific PURLs from the verdict cache;
- `DELETE /api/cache/packages/{packageType}` — delete entries by package type, package name, or repository context.

#Supported Protocols

This section describes which resources OSA Proxy Go checks and which responses it can modify for each ecosystem.

#Summary

Ecosystem	Manifest scanning	Package scanning	Response modification
Maven	Yes	Yes	Removes blocked versions from `maven-metadata.xml`, updates `latest` and `release`.
npm	Yes	Yes	Removes blocked versions from metadata, updates `dist-tags` and tarball links.
PyPI	Yes	Yes	Removes blocked links from Simple API pages, rewrites download URLs through the proxy.
NuGet	Yes	Yes	Modifies service index and registration metadata,

Ecosystem	Manifest scanning	Package scanning	Response modification
			removes blocked versions.
Go modules	Yes	Yes	Removes blocked versions from `@v/list`, proxies module zip and SumDB.
Composer	Yes	Yes	Modifies Packagist/ Composer metadata and rewrites dist URLs through the proxy.
RubyGems	Yes	Yes	Checks RubyGems metadata and downloaded `.gem` packages.
Debian	No	Yes	Packages indexes are not modified.
Alpine (APK)	No	Yes	`APKINDEX` indexes are not modified.
RPM	No	Yes	`repodata` metadata is not modified.
Docker	Yes	No	Checks image manifests; manifest lists are used to resolve image manifests and are not sent for checking as separate components. Blob/layer requests are proxied without separate layer checks. Several Docker repositories use subdomains.

#scan-manifest and scan-package

`scan-manifest` enables checking and modification of metadata from which the package manager selects available versions. When a blocking policy is triggered, unsafe versions are removed from the response or marked as blocked when the format supports it.

`scan-package` enables checking of downloaded artifacts: archives, binary packages, module zip files, `.gem`, `.deb`, `.apk`, or `.rpm`. If a policy blocks the component, the download is interrupted with the HTTP code from `codescoring.block-status-code`.

For Debian, Alpine, and RPM, only `scan-package` is used: system indexes are not modified, so the package manager can still see the version in the index, but downloading the specific package is blocked when it violates a policy.

For Docker, repository configuration does not use `scan-manifest` or `scan-package`. OSA Proxy Go checks Docker image manifests. Manifest lists are used to resolve image manifests and are not sent for checking as separate components; blob/layer requests are proxied to the registry.

#Maven

- Metadata: `maven-metadata.xml`.
- Packages: `.jar`, `.war`, `.ear`, and other Maven artifacts.
- When metadata is modified, blocked versions are removed and `latest`/`release` are updated to the latest allowed version.

#npm

- Metadata: package JSON metadata.
- Packages: `.tgz`.
- Blocked versions and related `time` entries are removed from metadata, and `dist-tags` are recalculated to allowed versions.

#PyPI

- Metadata: Simple API pages.
- Packages: `.whl`, `.tar.gz`, `.zip`, and other Python package archives.
- Links to blocked versions are removed, and download URLs are rewritten so downloads go through OSA Proxy Go.

#NuGet

- Metadata: service index and registration index.
- Packages: `.nupkg`.
- The client uses the `/nuget-api/v3/index.json` route; metadata is rewritten to OSA Proxy Go URLs.

#Go modules

- Metadata: version list `@v/list`.
- Packages: module `.zip`.
- Blocked versions are removed from the version list. SumDB uses `sumdb-registry` and the `GOSUMDB` client setting.

#Composer

- Metadata: Composer/Packagist metadata.
- Packages: dist archives `.zip`, `.tar`, `.tgz`, `.tar.gz`, `.tar.bz2`, `.tar.xz`.
- Dist URLs are rewritten to OSA Proxy Go routes. For external dist hosts, use `packages-registry` and `additional-packages-registries`.

#RubyGems

- Metadata: RubyGems indexes.
- Packages: `.gem`.
- Metadata and downloaded gem packages are checked.

#Docker

- Metadata: Docker image manifest. Manifest lists are used to resolve image manifests.
- Packages: there is no separate `scan-package`; blob/layer requests are proxied to the registry.
- Docker uses standard `/v2/...` and `/token` endpoints, so several Docker repositories are separated by subdomains, not by the first path segment.

#Blocking Diagnostics

If a package manager does not display the blocking reason, check the corresponding metadata endpoint directly:

```
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
```

#Base64 URL Configuration

Base64 URL is used when OSA Proxy Go must receive repository manager context from the request URL. This is needed for policies bound to a specific

repository manager and repository name when the upstream in `osa-proxy.yml` points directly to a public registry.

Enable this mode for a repository with `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
```

The Base64 parameter is placed immediately after the repository name:

```
https://<osa-proxy-url>/<repository-name>/<base64-parameters>/
<path...>
```

The encoded JSON contains repository context:

```
{"repoManagerHost":"https://nexus.example.com","repoName":"npm-
proxy"}
```

Example URL:

```
https://osa-proxy.example.com/npm/
eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL25leHVzLmV4YW1wbGUuY29tIi
wicmVwb05hbWUiOiJucG0tcHJveHkifQ/lodash
```

Docker does not use this mechanism in the client URL: Docker Registry API v2 works through `/v2/...` and `GET /token`.

#Maven Configuration

```
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
```

Example `settings.xml`:

```
<settings>
  <mirrors>
    <mirror>
      <id>osa-proxy</id>
      <mirrorOf>*</mirrorOf>
      <url>https://osa-proxy.example.com/maven/</url>
    </mirror>
  </mirrors>
</settings>
```

#npm Configuration

```
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
```

Client URL:

```
npm config set registry https://osa-proxy.example.com/npm/
npm view lodash version
```

Equivalent `.npmrc` entry:

```
registry=https://osa-proxy.example.com/npm/
```

During migration, replace the `registry` value in `.npmrc` from Nexus, Artifactory, or `https://registry.npmjs.org` with `https://osa-proxy.example.com/npm/`. Credentials remain in npm settings.

#NuGet Configuration

```
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
```

Add a package source:

```
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
```

Equivalent `NuGet.Config`:

```
<configuration>
  <packageSources>
    <clear />
    <add key="osa-proxy" value="https://osa-proxy.example.com/
nuget/nuget-api/v3/index.json" />
  </packageSources>
</configuration>
```

#PyPI Configuration

```
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
```

Example `pip.conf`:

```
[global]
index-url = https://osa-proxy.example.com/pypi/simple/
```

Persistent configuration with `pip config`:

```
python -m pip config set global.index-url https://osa-
proxy.example.com/pypi/simple/
```

One-off install:

```
pip install --index-url https://osa-proxy.example.com/pypi/
simple/ requests
```

For PyTorch, use a dedicated repository:

```
pip install --index-url https://osa-proxy.example.com/pytorch-
pypi/whl/cu121 torch
```

#Go Configuration

```
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
```

Configure the Go toolchain. Use `go env -w` for persistent configuration:

```
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` is required when requests to `sum.golang.org` must also go through OSA Proxy Go.

For a one-off run, set environment variables:

```
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
```

#Composer Configuration

```
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
```

Configure a Composer repository in the project:

```
composer config repositories.osa-proxy composer https://osa-
proxy.example.com/composer
composer config repo.packagist false
composer require monolog/monolog
```

Equivalent `composer.json` section:

```
{
  "repositories": [
    {
      "packagist.org": false
    },
    {
      "type": "composer",
      "url": "https://osa-proxy.example.com/composer"
    }
  ]
}
```

`additional-packages-registries` is used for dist archives that Composer metadata serves from separate hosts, for example GitHub or GitLab.

#RubyGems Configuration

```
ruby:
  enabled: true
  repository:
    - name: ruby
      registry: https://rubygems.org
      scan-manifest: true
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

For gem, replace the RubyGems source:

```
gem sources --add https://osa-proxy.example.com/ruby/
gem sources --remove https://rubygems.org/
gem install rails
```

For Bundler, set the source in `Gemfile`:

```
source "https://osa-proxy.example.com/ruby/"

gem "rails"
```

#Debian Configuration

```
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
```

Example `/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
```

#Docker Configuration

```
docker:
  enabled: true
  repository:
    - name: docker
      registry: https://registry-1.docker.io
      auth-token-url: https://auth.docker.io
      work-mode: strict_wait
```

Docker uses standard Registry API v2 endpoints. The repository name from the configuration is not added to the client path.

```
docker pull osa-proxy.example.com/library/alpine:latest
```

For Docker Hub, OSA Proxy Go can be configured as a registry mirror in `/etc/docker/daemon.json`:

```
{
  "registry-mirrors": ["https://osa-proxy.example.com"]
}
```

Restart the Docker daemon after changing the file.

If several Docker repositories are enabled, use subdomains where the subdomain matches `repository[*].name`:

```
docker pull docker.osa-proxy.example.com/library/alpine:latest
```

This is required by Docker Registry API v2 semantics: the client always uses fixed paths such as `/v2/...` and `/token`, so the OSA Proxy Go repository name cannot be added as the first path segment as it is for npm, Maven, or PyPI. With a single Docker repository, OSA Proxy Go can serve it through the main host. With several Docker repositories, the service selects the repository configuration by the request host.

For example, for this configuration:

```
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
```

clients must use different hostnames:

```
docker pull dockerhub.osa-proxy.example.com/library/
alpine:latest
docker pull company.osa-proxy.example.com/team/image:latest
```

For this setup, configure wildcard DNS or separate DNS records for the subdomains, a TLS certificate that covers these names, and a reverse proxy/load balancer that forwards requests to OSA Proxy Go while preserving the original `Host`.

#Alpine (APK) Configuration

```
alpine:
  enabled: true
  repository:
    - name: alpine
      registry: https://dl-cdn.alpinelinux.org/alpine
      scan-package: true
      work-mode: strict_wait
      url-encoded-config: true
```

Example `/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
```

#RPM Configuration

```
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
```

Example `.repo` file:

```
[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
```

#Migrating to OSA Proxy Go

This section describes how to migrate configuration from the legacy Java/Spring version of OSA Proxy to OSA Proxy Go. The legacy configuration used `application.yml`; the Go version uses `osa-proxy.yml`.

#Main changes

Legacy OSA Proxy	OSA Proxy Go
`application.yml`	`osa-proxy.yml`
`codescoring.host`	`codescoring.url`
`codescoring.proxy-manager-host`	`codescoring.osa-proxy-url`
Spring Boot logging with `logging.level.ru.codescoring`	`logging.level: debug/info/warn/error`
Actuator endpoints	`/healthz`, `/metrics`, `/api/v3/api-docs`, `/api/swagger/`
Global `codescoring.work-mode`	`codescoring.work-mode` plus `repository[*].work-mode` override

The `codescoring.enable-status-line`, `codescoring.block-status-code`, `codescoring.block-on-codescoring-errors`, and `codescoring.remove-blocked-versions` fields keep the same meaning, but must be moved to the new `osa-proxy.yml` structure.

#New capabilities

- Composer and RubyGems support.
- Repository-level work mode override with `repository[*].work-mode`.
- Repository-level file type filtering with `repository[*].file-type-filter`.
- Explicit HTTP server and HTTP client configuration in `http.server` and `http.client` instead of Spring Boot / WebFlux properties.

#npm migration example

Legacy configuration:

```
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
```

Configuration for 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
```

After migration, the npm client uses the same route name:

```
npm config set registry https://osa-proxy.example.com/internet-
npm/
npm view lodash version
```

#Migration checklist

1. Create a new `osa-proxy.yml` file.
2. Move the CodeScoring platform URL from `codescoring.host` to `codescoring.url`.
3. Move the token to `codescoring.token`.
4. Move the external proxy URL from `codescoring.proxy-manager-host` to `codescoring.osa-proxy-url`.
5. Choose the global `codescoring.work-mode` and set `repository[*].work-mode` for specific repositories when needed.
6. Move package manager sections and check `name`, `registry`, `scan-manifest`, `scan-package`, and `url-encoded-config`.
7. Enable `cache.judge.enabled` only when Redis is available and verdict caching is required.
8. Update package manager configuration to use OSA Proxy Go URLs.
9. Check `GET /healthz`, `GET /metrics`, and one test request for each enabled repository type.

#What to remove from the old configuration

Spring Boot parameters, JVM options, and actuator settings are not part of `osa-proxy.yml`. The Go version does not use `/actuator/metrics` or `/actuator/prometheus`; metrics are exposed directly at `/metrics`.

#Metrics

OSA Proxy Go exposes Prometheus metrics at:

```
GET /metrics
```

Example:

```
curl http://localhost:8080/metrics
```

The Go version does not provide Spring Boot actuator endpoints. Use `/metrics` for scraping, not `/actuator/metrics` or `/actuator/prometheus`.

#Prometheus scrape config example

```
scrape_configs:
  - job_name: osa-proxy-go
    metrics_path: /metrics
    static_configs:
      - targets:
          - osa-proxy.example.com:8080
```

#Post-installation check

1. Check that the service responds to the health check:

   ```
   curl http://localhost:8080/healthz
   ```

2. Check that the metrics endpoint returns Prometheus data:

   ```
   curl http://localhost:8080/metrics
   ```

3. Configure scraping in Prometheus or through the Helm chart ServiceMonitor if the service is deployed in Kubernetes.

#CodeScoring.SCA

#General description

The **CodeScoring.SCA** module solves the problems of software inventory and vulnerability search in open source components. The main functional capabilities of the module include:

- **Control at different stages of development lifecycle** with the ability to check projects in the version control system;
- **Integration of checks into the CI pipeline** with blocking security policies using the console agent Johnny;
- **Generating SBOM** and visualization of the dependency graph;
- **Analysis at different levels**: manifest parsing, resolving transitive dependencies, build interception for C and C++, scanning Docker images;
- **Tracking scan history** with the ability to export results for reporting;
- Vulnerability reachability analysis.

#Setting up and running the analysis

#Setting up the analysis

When running the analysis on the platform, you can select parameters for individual projects.

Management is performed on the project page in the `Settings -> Projects` section.

- **Scan with hashes** - scans the project using the hash sums of files to search for direct inclusion of dependencies;
- **Analysis path exclusions** - a list of directories that will be ignored during scanning;
- **Disable recursive scanning** - disables directory depth traversal during scanning. Only manifests in the root directory will be found;
- **Activate cloud resolve** - enable dependency resolution in the cloud. Warning! Using cloud resolve may give inaccurate results and increase the analysis time;
- **Exclude from SCA analysis** - exclude this project from SCA analysis;

#Manual launch of analysis

Compositional analysis (SCA) is launched automatically immediately after adding a project. To manually launch the analysis for a project, use the **Run SCA** button on the project page. In this case, you can select a separate branch or tag for analysis, which will be taken into account in the scan history.

You can also launch the analysis for all projects and for each analysis type (SCA, Quality, Authors) separately. You can manage the launch of the general analysis in the `Settings -> Workmode` section.

Important

To obtain correct results, you need to launch the analysis sequentially for each module, having previously waited for the completion of the previous run. Launch order:

1. Software Composition Analysis (SCA)
2. Authors Analysis
3. Quality Analysis

The progress of the analysis can be monitored by messages in the `Settings -> Audit log` section.

The first launch of the authors analysis can take a noticeable time, since the entire repository history is traversed. Subsequent runs will only analyze the difference in commits for updates since the last run.

#Scheduled analysis

In addition to manual launch, you can set up scheduled analysis of individual projects. This is managed on the project page in the `Settings -> Projects` section.

By default, the **SCA scan schedule** parameter is set to **Off**. To activate scheduled analysis, select **On** and specify the time and days of the week.

Note: The scan time will be calculated according to UTC +3.

#Running Version Analysis

If necessary, you can run an analysis of the selected version. To do this, use the additional action of the scan start button **Scan another version**.

In the modal window that appears, you can select the version to scan or create a new version and run the scan.

Important

It is possible to create a new version only if the user has the authority to edit the project.

#Overviewing dependencies

#Viewing dependencies list

The list of scanned open source dependencies can be viewed in the `SCA -> Dependencies` subsection. The table in this section contains **all** dependencies that were checked during the operation of the SCA module, with the following information:

- **Dependency** – dependency name (with a link to its individual page);
- **Technology** – technology (programming language or build tool);
- **Licenses** – license ID from the package index;
- **Authors** – developer of the component the package index;
- **Vulnerabilities** – number of vulnerabilities found in the dependency;
- **Found** – dependency definition type: by resolve, by manifest or by content (when the component code is included in the project code base);
- **Relationship** – dependency type (direct or transitive);
- **Environment** – development environment;
- **Parent Dependencies** – related upstream dependencies;
- **Project** – project that uses the dependency;
- **Maximum fix version** - the version of the dependency to which an update should be performed in order to fix the vulnerabilities currently detected by the SCA module, while only dependencies with the specified fixed version are taken into account;
- **Release date** - the date and time of the dependency release.

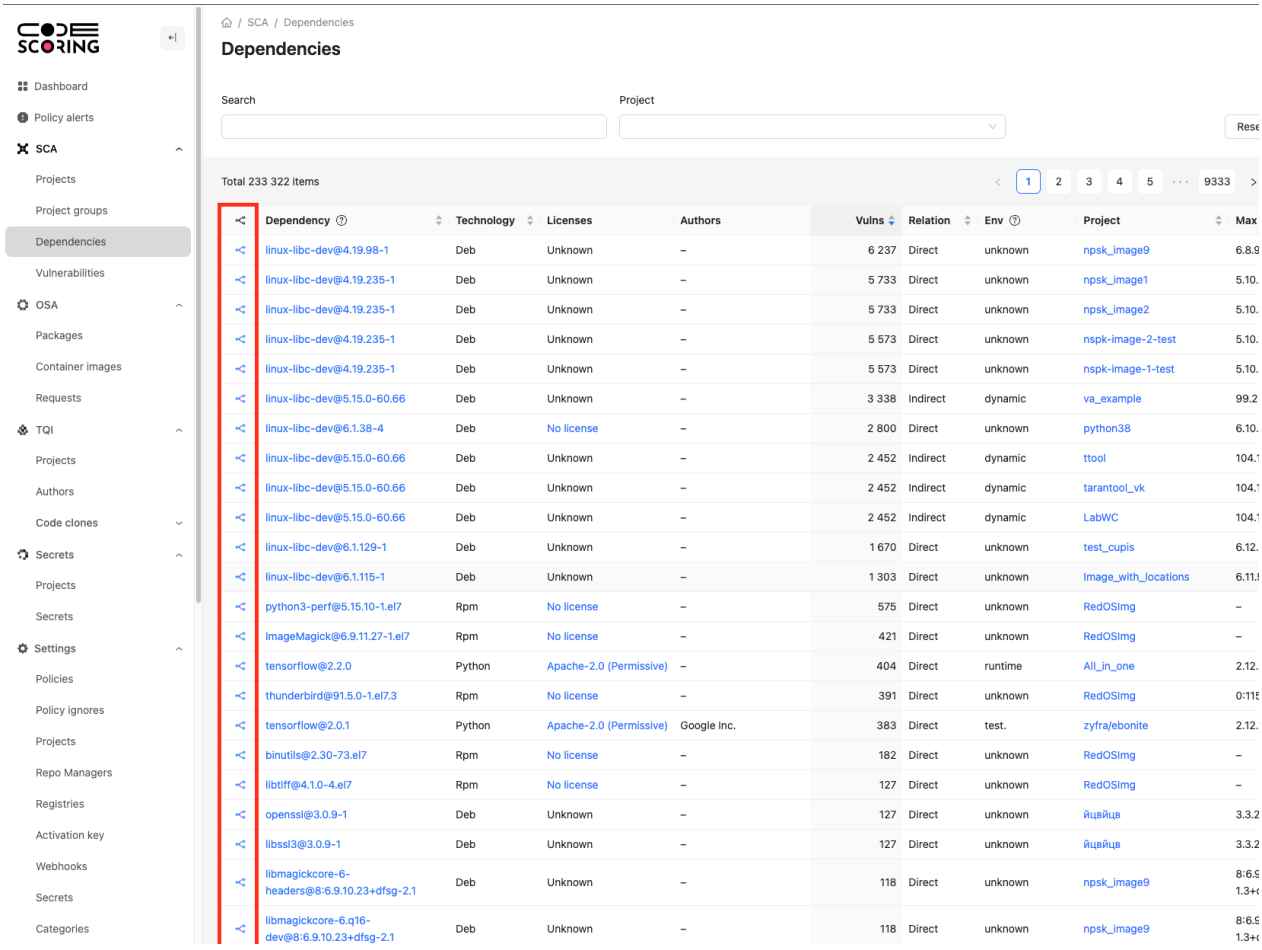
The dependency table can be filtered by project, department, project category, project groups, technology, license, license category, as found, relationship, parents, environment, release time period.

Clicking on the dependency name takes you to its individual page, where information about its use in projects and found vulnerabilities is displayed.

#Working with dependency graph

Open source dependencies of software projects represent a graph – a structure in which individual components are depicted as nodes, and the connections between them as edges.

You can see the visualization of the dependency graph in the Dependencies section or on the project page by clicking on the corresponding icon in the list of dependencies.



On the page with interactive visualization, components are displayed according to their nesting level - from the root dependency to the deepest level of transitive dependencies. Hovering the cursor over an object reveals more detailed information about the component, such as version, environment, technology and number of vulnerabilities.

The visualization is interactive and scalable. By selecting a component, you can trace its path into the project. Components with identified vulnerabilities are highlighted.

Components on the graph can be filtered according to the following parameters:

- technology;
- development environment;
- severity of vulnerability.

After selecting a graph component, you can configure the displayed relationships:

- direction (root/descendants)
- nesting level for descendants
- selection of a specific path to the root

#Working with vulnerabilities

#Viewing the list of vulnerabilities

The list of detected vulnerabilities is available in the `SCA -> Vulnerabilities` subsection. This section displays **all vulnerabilities** detected by the SCA and OSA modules during their operation.

The vulnerability table contains the following information:

- **Vulnerability** – the vulnerability identifier (e.g., CVE) with a link to its individual page;
- **Dependency** – the component in which the vulnerability was detected, including the version;
- **Relationship** – the type of dependency in which the vulnerability was detected (direct or transitive);
- **Environment** – the environment in which the dependency is used (e.g., runtime, dev, main);

- **Project** – the project in which the vulnerable dependency was detected;
- **Status** — vulnerability triage status;
- **CVSS 2** – CVSS v2 threat score;
- **CVSS 3** — CVSS v3 threat score;
- **CVSS 4** — CVSS v4 threat score;
- **Exploitation** — the present state of exploitation of the vulnerability according to SSVC;
- **Automatable** — possibility for an attacker to automatically exploit the vulnerability according to SSVC;
- **Technical impact** — the technical impact of the vulnerability according to SSVC;
- **EPSS** — the possibility of vulnerability exploitation according to EPSS;
- **CWE** — Common Weakness Enumeration categories to which the vulnerability belongs;
- **Exploitable** — indicates the presence of a publicly known exploit;
- **Reachable** — information about the vulnerability's reachability in the context of component usage;
- **Impact** — type of potential impact of the vulnerability (e.g., XSS, DoS, RCE, etc.);
- **Fixed version** — version of the dependency in which the vulnerability is fixed;
- **Found** — date and time when the vulnerability was discovered.

For convenience of analysis, the list of vulnerabilities can be filtered by the following parameters:

- project;
- proprietor;
- project category and group;
- vulnerability publication timeframe;
- detection date;
- CVSS v2, CVSS v3 and CVSS v4 rating and threat level;
- technology;
- dependency environment;
- dependency relationship type (direct or transitive);
- exploit presence;
- reachability;
- fixed version;
- SSVC rating presence;
- EPSS percentage;
- CWE;
- vulnerability impact;
- status;
- justification;
- response.

Text search by vulnerability ID and related data is also available.

#Triage

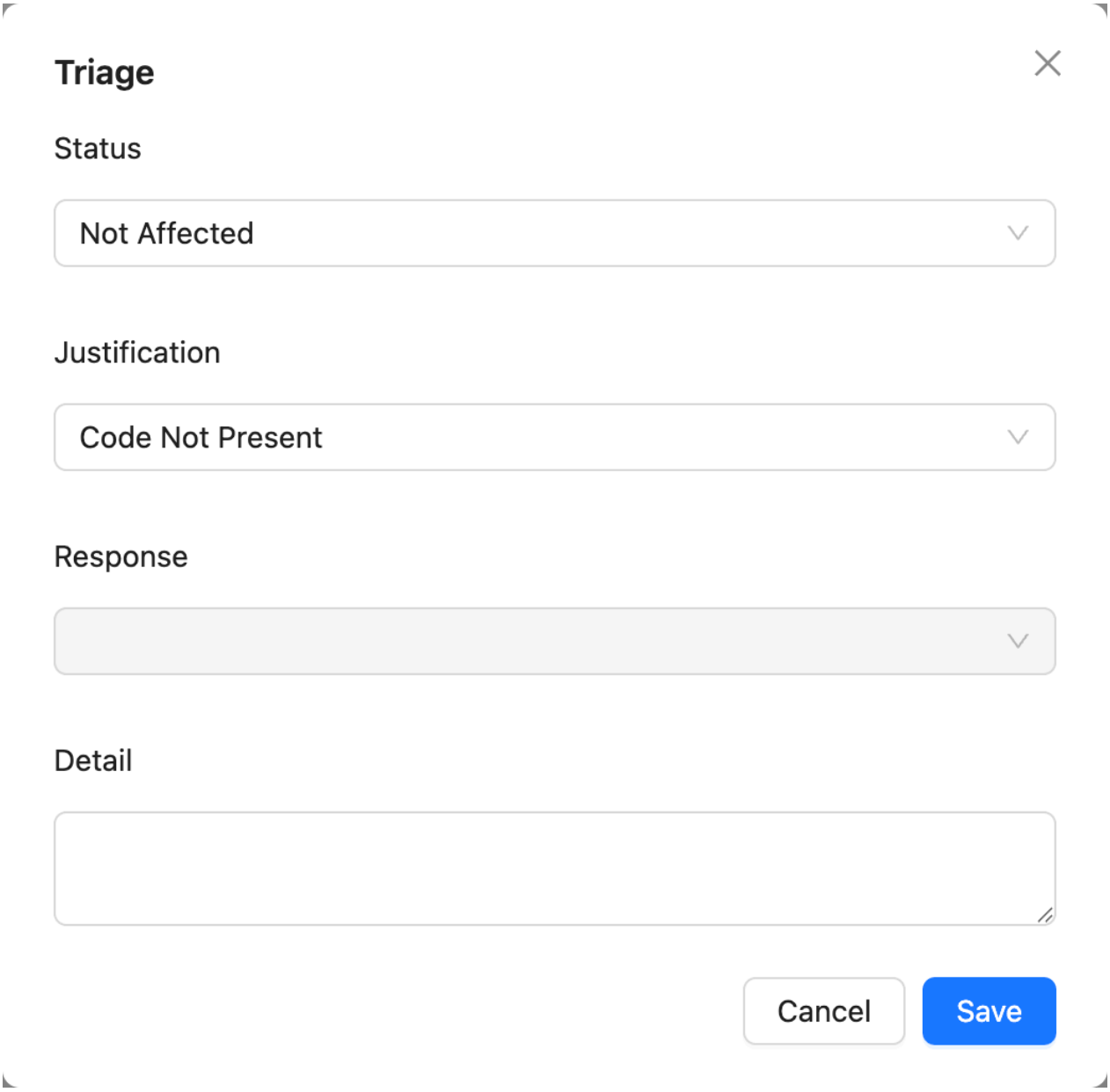
Triage allows users to manually assign a vulnerability status that reflects its actual impact on the project and the team's planned response. The feature is

implemented in accordance with the VEX (Vulnerability Exploitability eXchange) standard.

To perform triage, select one or more vulnerabilities in the list and click the **Triage** button above the table. A modal dialog will open with the following fields:

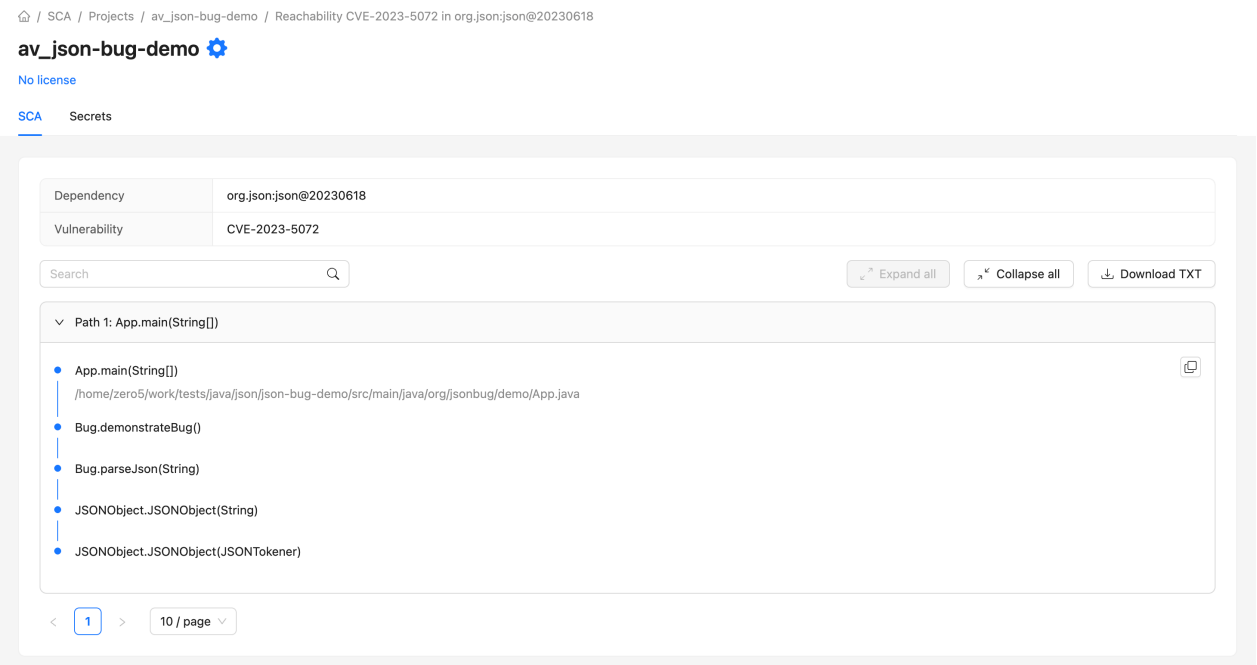
- **Status** — the vulnerability status reflecting its applicability to the project (e.g., *Not Affected*, *Active*, *Confirmed*, *False Positive*, *No Status*);
- **Justification** — the reason for assigning the *Not Affected* status (e.g., *Code Not Present*, *Code Not Reachable*);
- **Response** — the planned or implemented response to the vulnerability (e.g., *Can Not Fix*, *Will Not Fix*);
- **Detail** — a free-text field for additional comments or context.

After saving, the assigned status is displayed in the **Status** column of the vulnerability list and can be used as a filter.



#Reachability

For reachable vulnerabilities, it is possible to view the visualization of paths and download their text representation.



#Vulnerability page

The individual vulnerability page is designed for detailed analysis of a specific vulnerability and all related information within the platform.

Deduplication of vulnerabilities

The page displays a **single deduplicated vulnerability**, even if it was discovered by multiple data sources (e.g., NVD, GitHub Advisories, FSTEC BDU, etc.). The user can also view the original data from each source separately.

#General vulnerability information

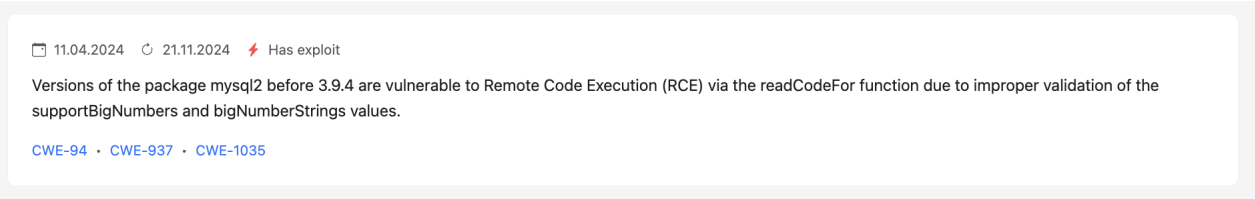
The top of the page displays summary information about the vulnerability:

- vulnerability ID (e.g., CVE);
- publication, withdrawal (if exists) and update dates in the data source;
- presence of a publicly known exploit;
- is vulnerability a protestware;
- whether the vulnerability was used in Ransomware;
- a brief description of the vulnerability;
- associated CWE categories.

Publication and withdrawal dates are the earliest dates among all the sources. Update date is the latest one.

Vulnerability ID and description are displayed from the sources in the following order:

- CVE.ORG;
- GHSA;
- Kaspersky;
- BDU;
- Other sources in alphabetical order.

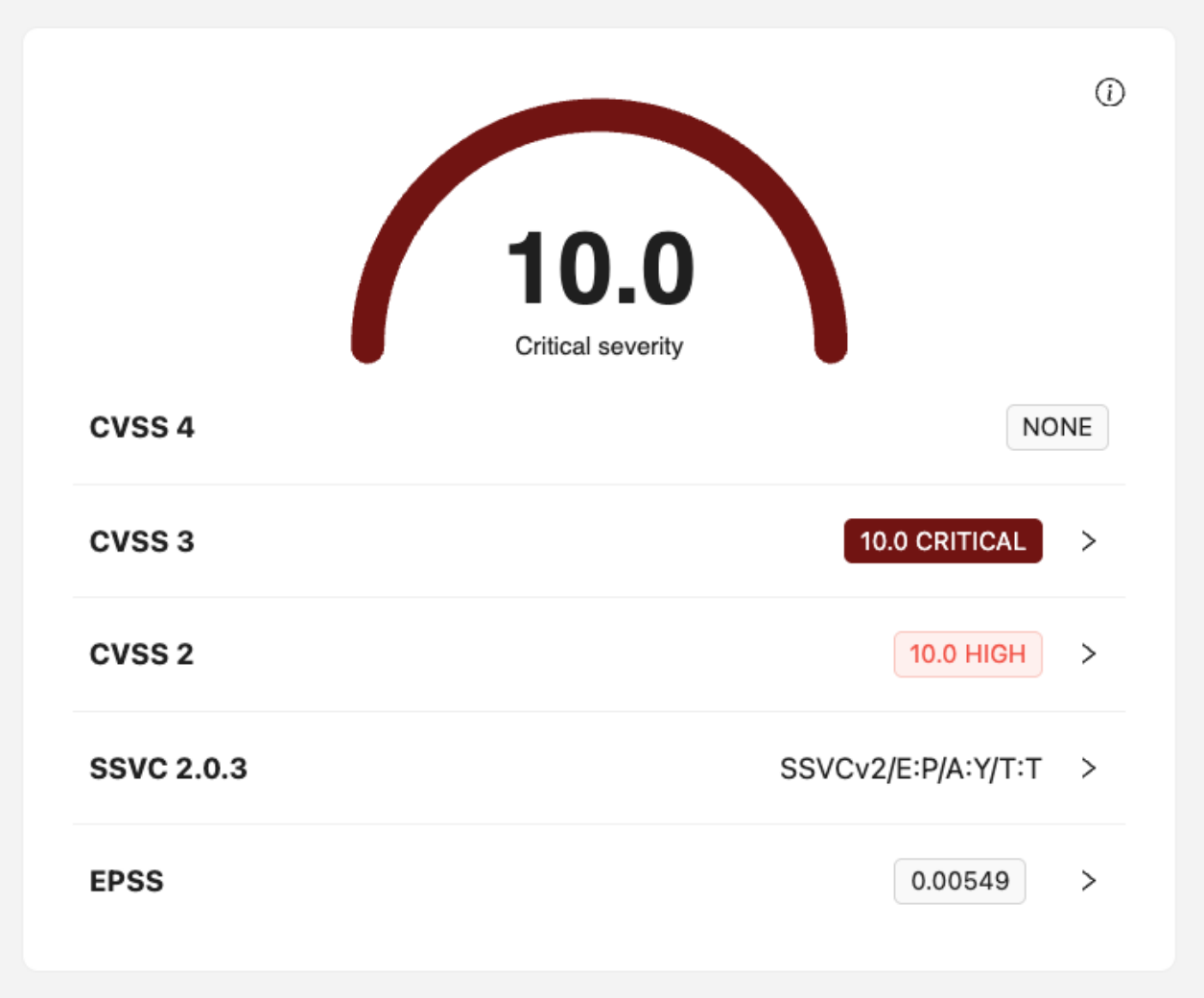


The highest threat level score for the most recent CVSS version, taking all sources into account, is displayed on the right.

Score, severity level and other metrics are displayed from the source with the highest score for each CVSS version.

Below you can also see the following information:

- breakdown by CVSS level, indicating the version and corresponding threat level;
- SSVC vector for vulnerability categorization;
- EPSS probability of vulnerability exploitation.

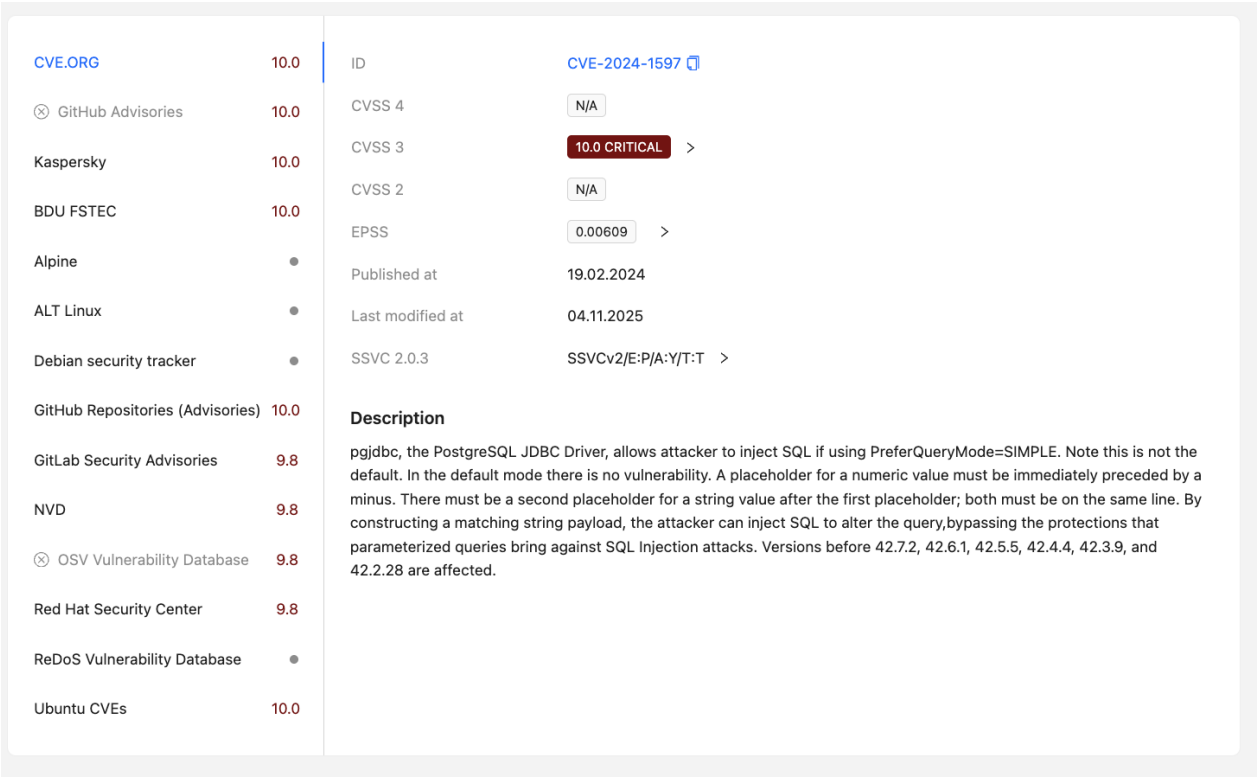


#Data sources and scores

A list of sources where a vulnerability was reported is displayed. For each source, the following can be provided:

- its own CVSS score;
- CVSS version;
- SSVC vector for vulnerability categorization;
- EPSS probability of vulnerability exploitation;
- additional attributes and metadata of the source.

This allows to compare data from different sources and take into account discrepancies in scores during risk analysis.

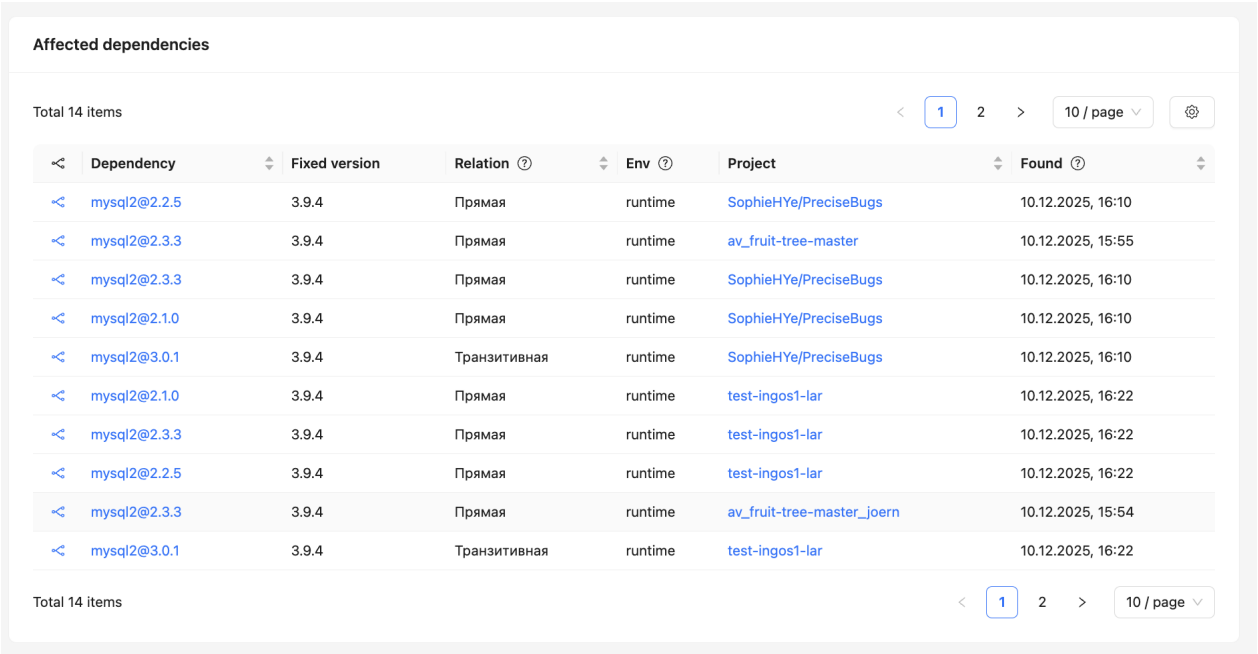


#Affected dependencies and images

This page displays lists of affected components:

- dependencies detected in SCA projects;
- packages and images scanned by the CodeScoring.OSA module.

This separation simplifies vulnerability analysis in different usage contexts and helps to more accurately assess its impact.



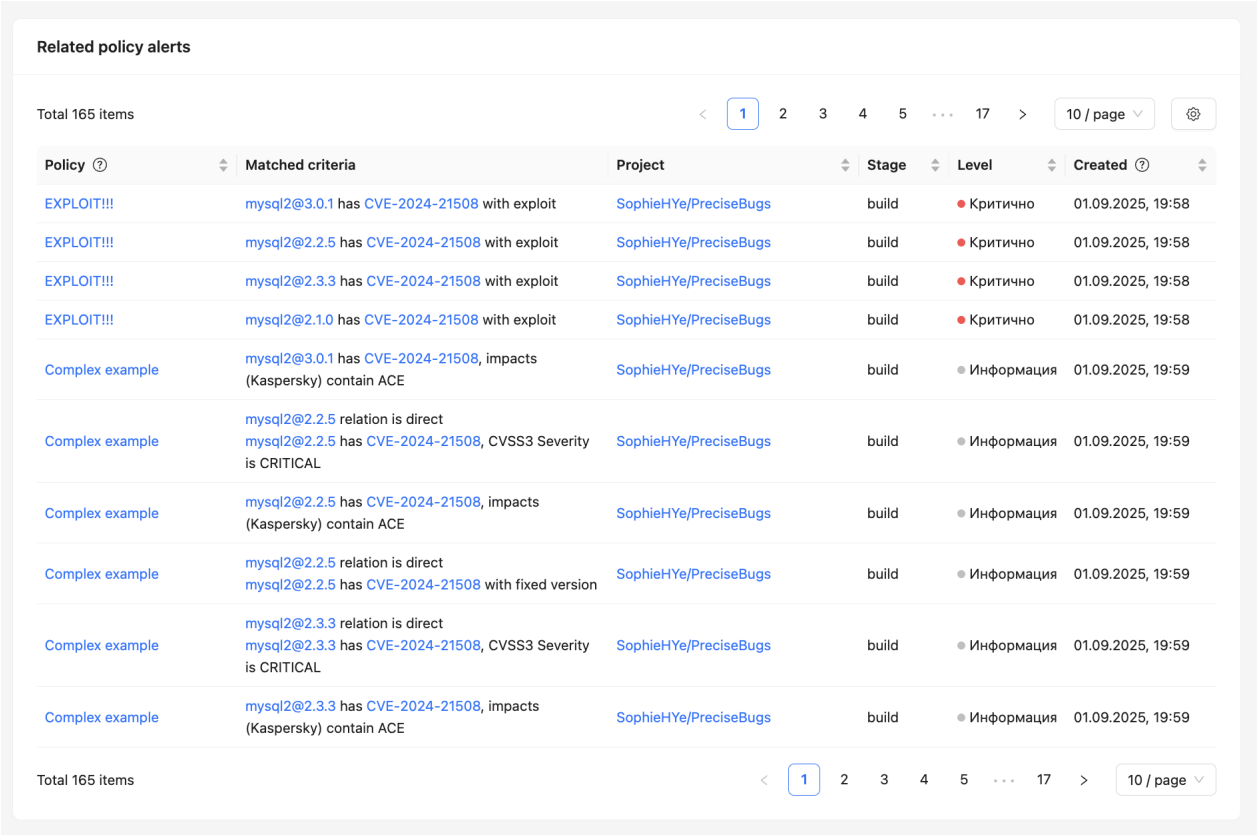
#Related alerts

At the bottom of the page, a list of related alerts generated by security policies is displayed.

For each alert, the following is displayed:

- the policy under which it was created;
- triggering conditions;
- project and development stage;
- severity level;
- creation date and time.

This allows to quickly understand which security policies the vulnerability affects and where exactly it impacts the project.



Related policy alerts

Total 165 items

1 2 3 4 5 ... 17 > 10 / page

Policy	Matched criteria	Project	Stage	Level	Created
EXPLOIT!!!	mysql2@3.0.1 has CVE-2024-21508 with exploit	SophieHYe/PreciseBugs	build	Критично	01.09.2025, 19:58
EXPLOIT!!!	mysql2@2.2.5 has CVE-2024-21508 with exploit	SophieHYe/PreciseBugs	build	Критично	01.09.2025, 19:58
EXPLOIT!!!	mysql2@2.3.3 has CVE-2024-21508 with exploit	SophieHYe/PreciseBugs	build	Критично	01.09.2025, 19:58
EXPLOIT!!!	mysql2@2.1.0 has CVE-2024-21508 with exploit	SophieHYe/PreciseBugs	build	Критично	01.09.2025, 19:58
Complex example	mysql2@3.0.1 has CVE-2024-21508, impacts (Kaspersky) contain ACE	SophieHYe/PreciseBugs	build	Информация	01.09.2025, 19:59
Complex example	mysql2@2.2.5 relation is direct mysql2@2.2.5 has CVE-2024-21508, CVSS3 Severity is CRITICAL	SophieHYe/PreciseBugs	build	Информация	01.09.2025, 19:59
Complex example	mysql2@2.2.5 has CVE-2024-21508, impacts (Kaspersky) contain ACE	SophieHYe/PreciseBugs	build	Информация	01.09.2025, 19:59
Complex example	mysql2@2.2.5 relation is direct mysql2@2.2.5 has CVE-2024-21508 with fixed version	SophieHYe/PreciseBugs	build	Информация	01.09.2025, 19:59
Complex example	mysql2@2.3.3 relation is direct mysql2@2.3.3 has CVE-2024-21508, CVSS3 Severity is CRITICAL	SophieHYe/PreciseBugs	build	Информация	01.09.2025, 19:59
Complex example	mysql2@2.3.3 has CVE-2024-21508, impacts (Kaspersky) contain ACE	SophieHYe/PreciseBugs	build	Информация	01.09.2025, 19:59

Total 165 items

< 1 2 3 4 5 ... 17 > 10 / page

#Additional information

The right side of the page also displays:

- Links to external resources (NVD, CVE.org, GitHub, and others);
- Internal vulnerability identifier in CodeScoring;
- Date of the last data update.

References

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

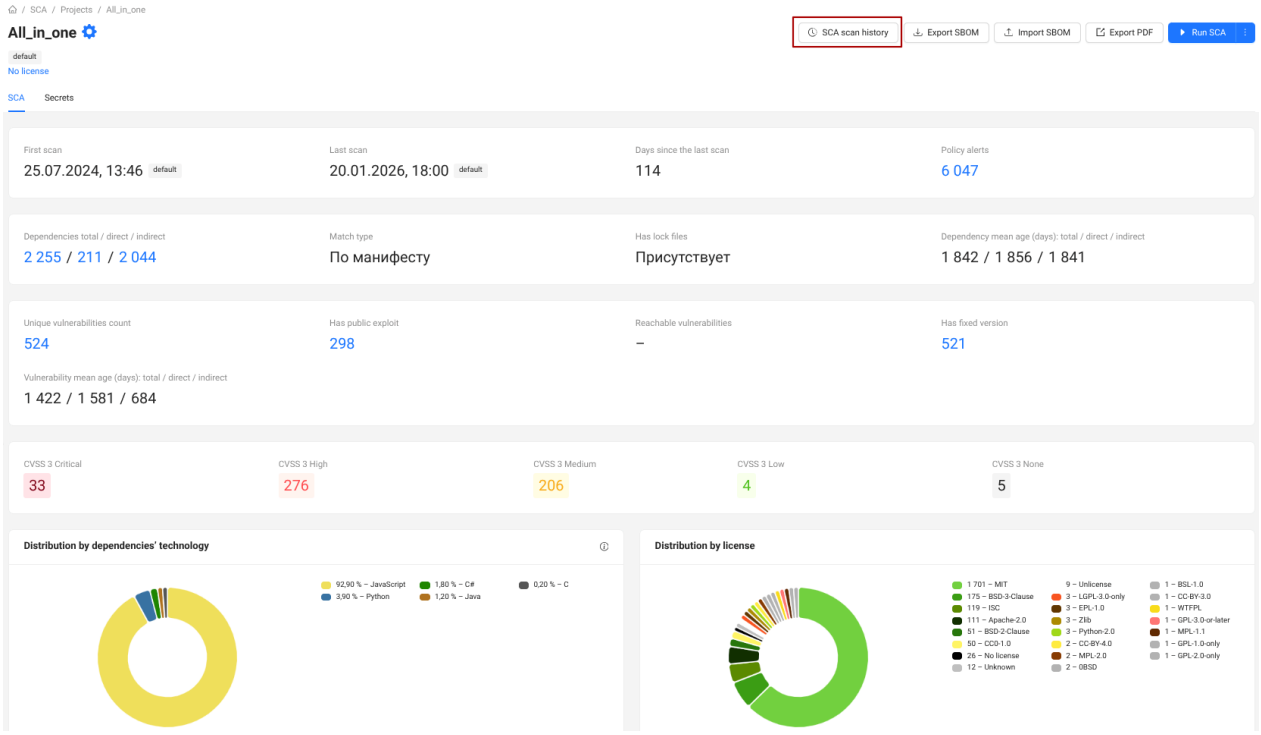
https://www.cve.org/CVERecord?id=CVE-2022-21724

https://поддержка.нппкт.рф/bin/view/ОСнова/Обновления/2.6/

CodeScoring ID UNCS-2022-9732

#Tracking scan history

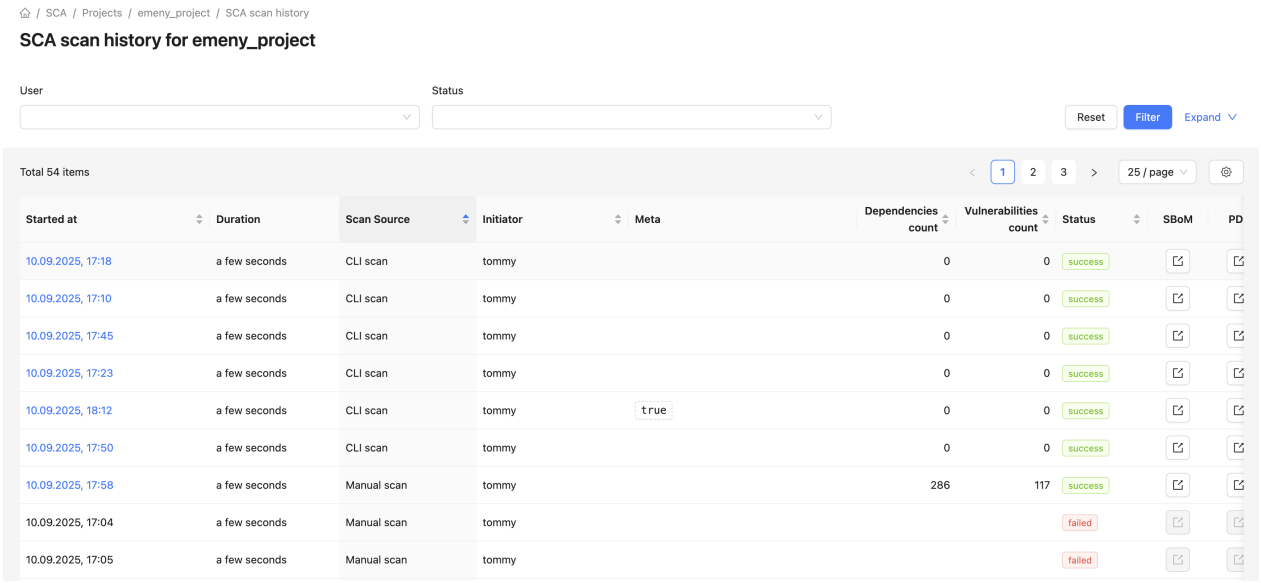
For each project analysis within the SCA module, a snapshot of the results of found dependencies and vulnerabilities is saved. To see the list of available snapshots, you should go to the SCA tab on the project page and click on the **SCA scan history** button.



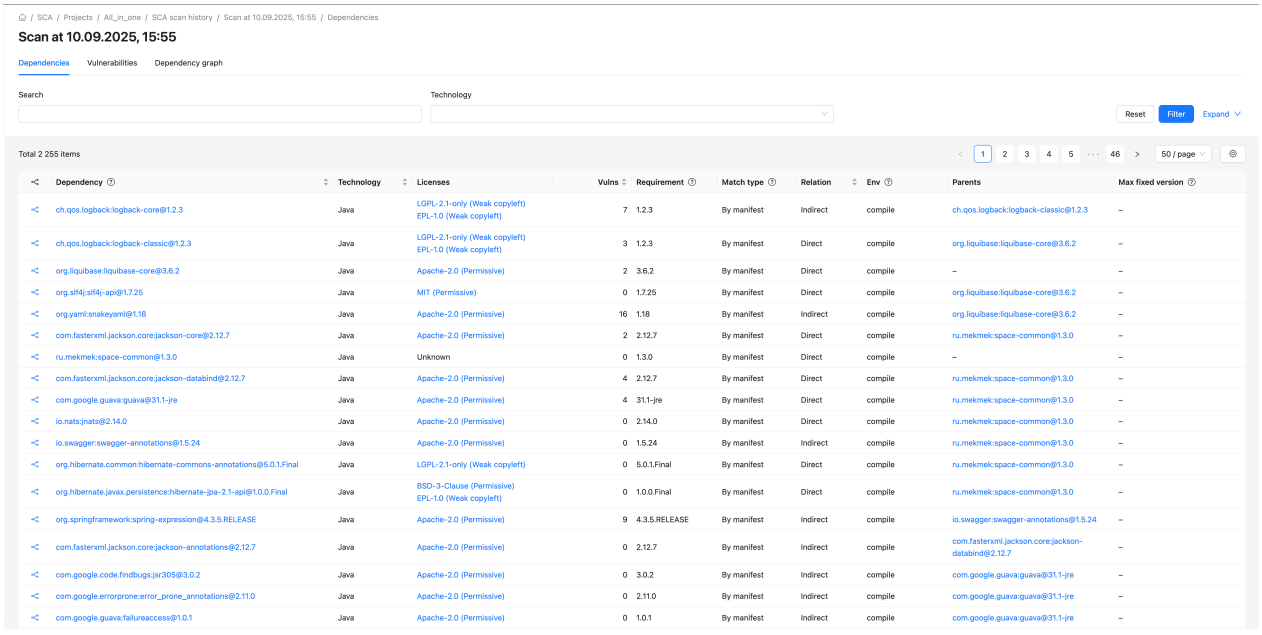
The snapshot contains the following data:

- **Started at** - the date the project was scanned. The date of the latest scan is marked with a green label;
- **Duration** – the time the analysis lasted;
- **Scan Source** - startup type, manual or scheduled;
- **Initiator** - user who started the scan. For scheduled runs the value is set to "system";
- **Version** – version information, name and metadata of the git branch (for VCS projects);
- **Dependencies count** - number of found dependencies;
- **Vulnerabilities count** - number of found vulnerabilities;
- **Status** - status of completed scanning. It can have three possible values - success, failed or canceled.

The buttons on the right side of the list allow to download a SBOM generated during the analysis and export a PDF report.



To view scan details, click on the link with the scan date in the first column. A list of project dependencies and vulnerabilities, as well as dependency graph for the particular scan are available on this page.



#Exporting analysis results

#Exporting to CSV

Each table with analysis results in CodeScoring can be exported in CSV format using the **Export** button in the upper right corner of the interface.

The CSV table will take into account the filters used at the time of export.

#Generating a PDF report on a project

After performing composition analysis, the generation of a PDF report with summary information on the project becomes available on the project page.

You can export a PDF report with the latest analysis data on the project page using the **Export PDF** button. You can export an analysis report for a specific date on the `SCA Scan History` page.

The report will be generated in the language set in the user profile.

The resulting report contains the following data by default:

- general information on the project (name, VCS branch, time of the last analysis, commit hash);
- distribution of vulnerabilities by CVSS;
- distribution of vulnerabilities by technology;
- table of found dependencies divided by technologies and development environment;
- table of found vulnerabilities divided by technologies and development environment;
- table of active policy alerts;
- table of policy ignores;
- dependency graph.

It's also possible to define a custom filename, choose required data blocks and filter data before export. By default, the report will list only effective policy ignores. Effective ignores are those that are applied to policies that are currently active. To display all policy ignores in the list, you need to uncheck the "Only effective policy ignores" checkbox.

If a filename is not set, it will be generated automatically according to the following rules:

- For projects: `report_<project name>.pdf`
- For container images:
  `report_<image name>_<first 8 hash chars>.pdf`

PDF export

Filename

* Content

- [x] Summary
- [x] Dependencies
- [x] Licenses
- [x] Vulnerabilities
- [x] Policy alerts
- [x] Policy ignores
- [x] Dependency tree
- [x] Triage details

Export all data | Filter data

- [x] Only effective policy ignores

Environment

Technology

Match type

Relation

#Working with SBOM within a project

After performing composition analysis of the project it becomes possible to export the obtained list of used components (SBOM) in CycloneDX format.

You can export the obtained SBOM on the project page in the `Projects` section by clicking the **Export SBOM** button.

SBOM export is supported in the following formats:

- CycloneDX v1.4 JSON;
- CycloneDX v1.5 JSON;
- CycloneDX v1.6 JSON;
- CycloneDX v1.6 Ext JSON – extended CycloneDX format containing additional properties: `GOST:attack_surface`, `GOST:security_function`, `GOST:source_lang`, `GOST:provided_by`. The format is adapted to additional requirements for the SBOM from the FSTEC of Russia.
- CycloneDX v1.7 JSON;
- CycloneDX v1.7 Ext JSON – extended CycloneDX format containing additional properties: `GOST:attack_surface`, `GOST:security_function`, `GOST:source_lang`, `GOST:provided_by`. The format is adapted to additional requirements for the SBOM from the FSTEC of Russia.
- SPDX v2.3 JSON

SBOM export also supports additional export customization, like the PDF one. Automatic SBOM filename generation rules are following:

- For projects: `bom_<project name>_<SBOM format>.json`
- For container images:
  `bom_<image name>_<first 8 hash chars>_<SBOM format>.json`

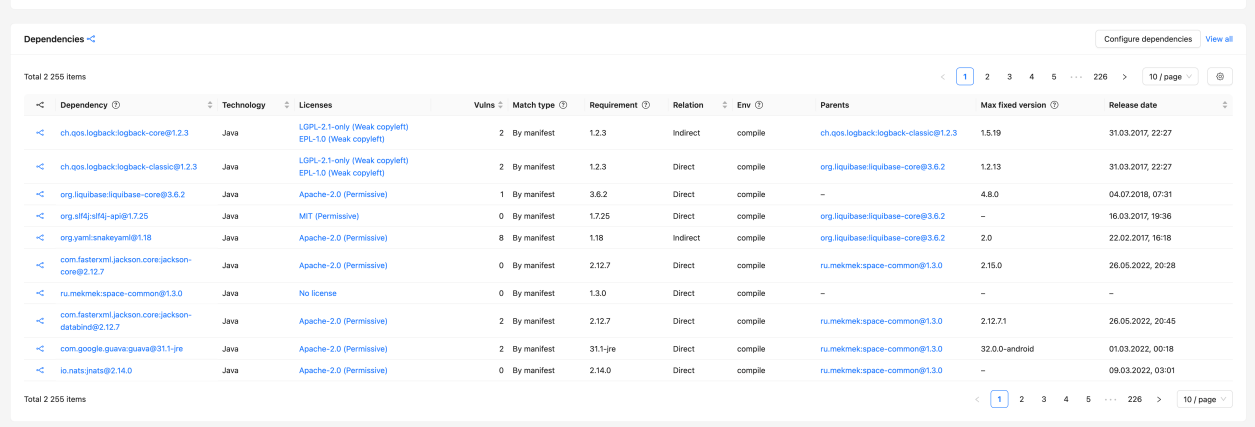
#SBOM import

For CLI projects, SBOM can also be loaded via the interface using the **Import SBOM** button. The loaded SBOM must be in CycloneDX v1.4, 1.5, 1.6, 1.6_ext, 1.7 or 1.7_ext format and have the `.json` extension.

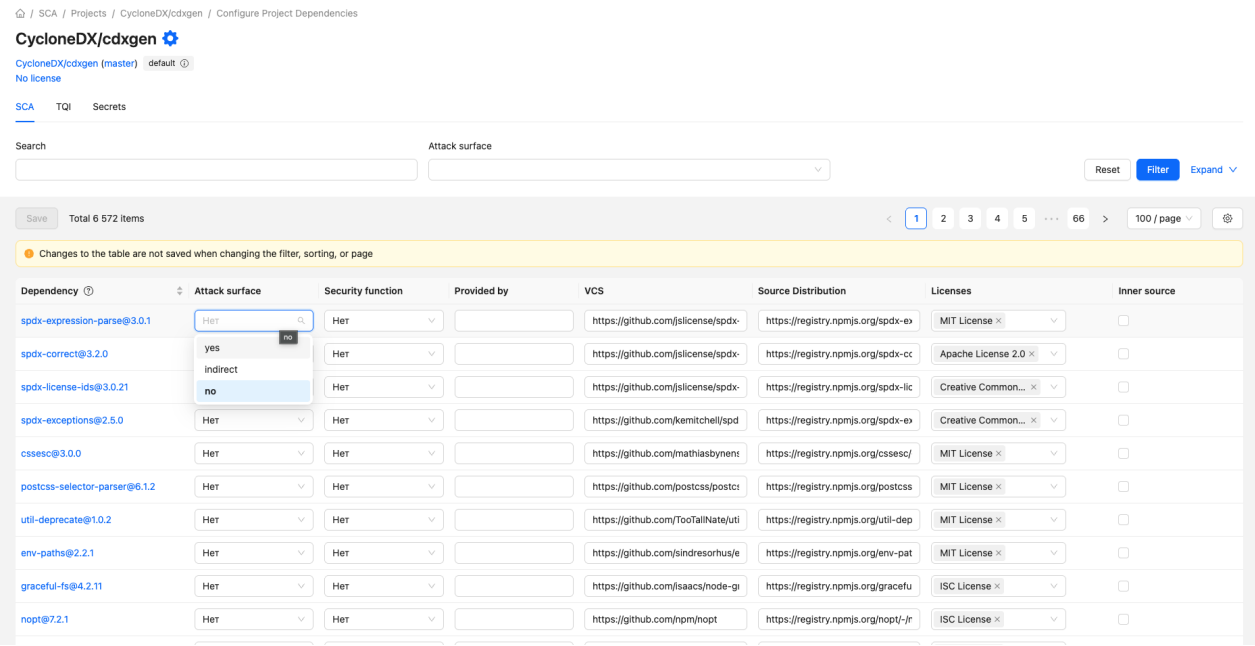
Importing SBOM you can also add branch or tag name as a meta-information.

#Configuring dependency properties for SBOM export

To configure dependency properties, go to the page by clicking the `Configure dependencies` button in the **Dependencies** table of the project.



The page allows you to specify the attack surface (**Attack surface**), security function (**Security function**), version control system (**VCS**) and licenses for each project component.



The values entered are taken into account:

- when exporting SBOM from the project page;
- when exporting SBOM from the scan results history page (for the most recent successful SCA scan);
- during subsequent scans of the project;
- when scanning the project via the Johnny console agent;
- in the project dashboard;
- on the license page.

Important: Changes to values do not apply to previous scans of the project and only relate to the SBOM of the current project, even if a dependency is used in multiple projects.

#VCS

The **VCS** field allows you to specify the URL of the repository where the dependency code is stored. When exporting SBOM, the selected value is taken into account in the externalReferences field.

#Source Distribution

The **Source Distribution** field contains the URL of the package's source code. When exporting SBOM, the selected value is taken into account in the externalReferences field.

#Attack surface

The **Attack surface** field allows you to specify whether the component belongs to an attack surface. You can select one of the following values:

- yes — the component is part of the direct attack surface;
- `indirect` — the component is part of the indirect attack surface;
- no — otherwise (default value).

When exporting SBOM in the `CycloneDX v1.6 Ext JSON` and `CycloneDX v1.7 Ext JSON` formats, the selected value is taken into account in the `GOST:attack_surface` property of the component.

#Security function

The **Security function** field allows you to specify whether the component belongs to the security functions of the information security tool. You can select one of the following values:

- yes — if the component functions directly implement security functions;
- `indirect` — if the component functions participate in the implementation of security functions, interacting with components implementing security functions;
- no — if the component functions do not participate in the implementation of security functions (default value).

When exporting SBOM in the `CycloneDX v1.6 Ext JSON` and `CycloneDX v1.7 Ext JSON` formats, the selected value is taken into account in the `"GOST:security_function"` property of the component.

#Provided by

The **Provided by** field lets you specify affiliation of the component to the information security tool from which this component was borrowed. You can enter any arbitrary text value.

When exporting SBOM in the `CycloneDX v1.6 Ext JSON` and `CycloneDX v1.7 Ext JSON` formats, the specified value is taken into account in the "GOST:provided_by" property of the component.

#Licenses

The **Licenses** field allows you to specify the licenses of the component.

If an empty list is specified, the values found during the last SCA analysis are selected.

When exporting SBOM, the selected values are taken into account in the `licenses` field of the component.

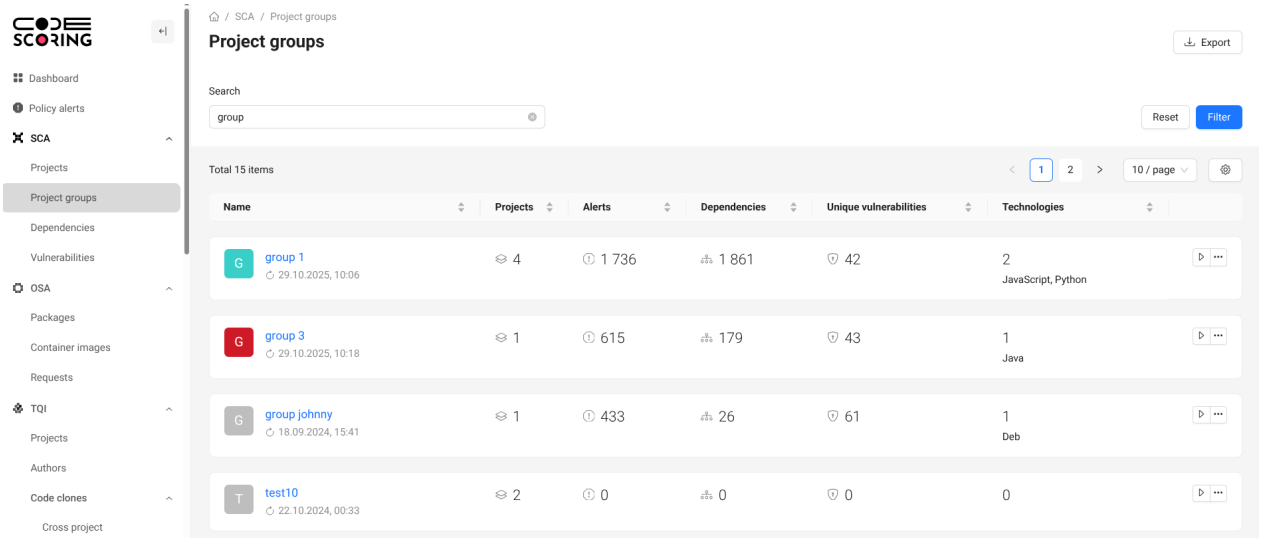
#Organizing project groups

#Viewing the list of groups

The list of groups can be viewed in the section `SCA -> Project Groups`. This section displays all the groups available for viewing by the user.

The following information is displayed for each group:

- **Name** - name of the group and the date of the last update of the project group settings;
- **Projects** - number of projects in the group;
- **Alerts** - total number of alerts related to the group's projects;
- **Dependencies** - total number of dependencies related to the group's projects;
- **Unique vulnerabilities** - number of unique vulnerabilities associated with the group's projects;
- **Technologies** - number and list of technologies related to the group's projects.



Access

The group displays only the projects included in the group and to which the user has access.

The following actions are available for each entry in the project list:

- **Start scanning group projects** - starts a scan for all projects in the group;
- **Group editing** - opens the page **Settings -> Groups**;
- **Deleting a group** - deletes a group with a preliminary confirmation of the operation from the user;
- **Export to CSV** – exports aggregated group data to csv-file;
- **Export to SBOM** - exports the list of used components (SBOM) obtained after composition analysis for all projects included in the group.

Access

To perform actions, the user must have the appropriate permissions. A description of permissions is provided in section Managing user accounts.

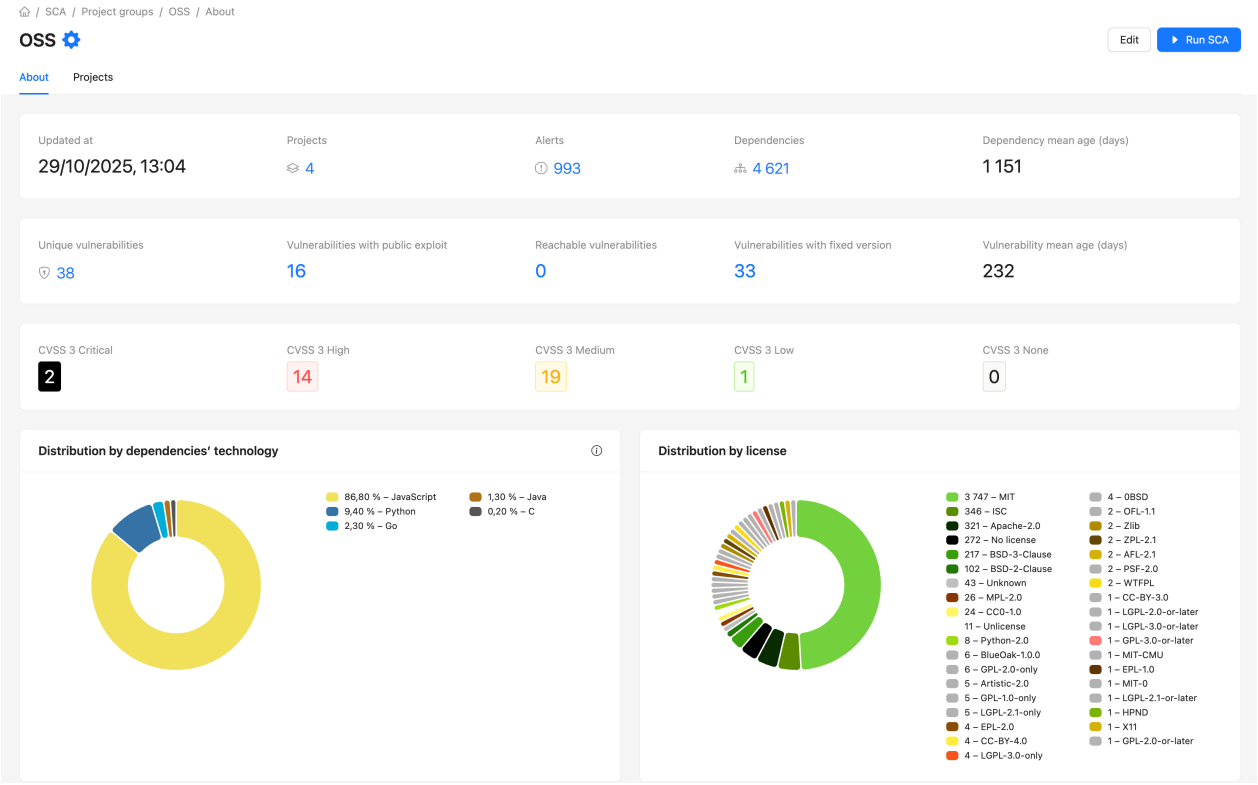
Export to SBOM

Export SBOM requires that each project has successfully completed a composition analysis.

#Project group page

#General statistics for the group

The "About" section provides summarized information.



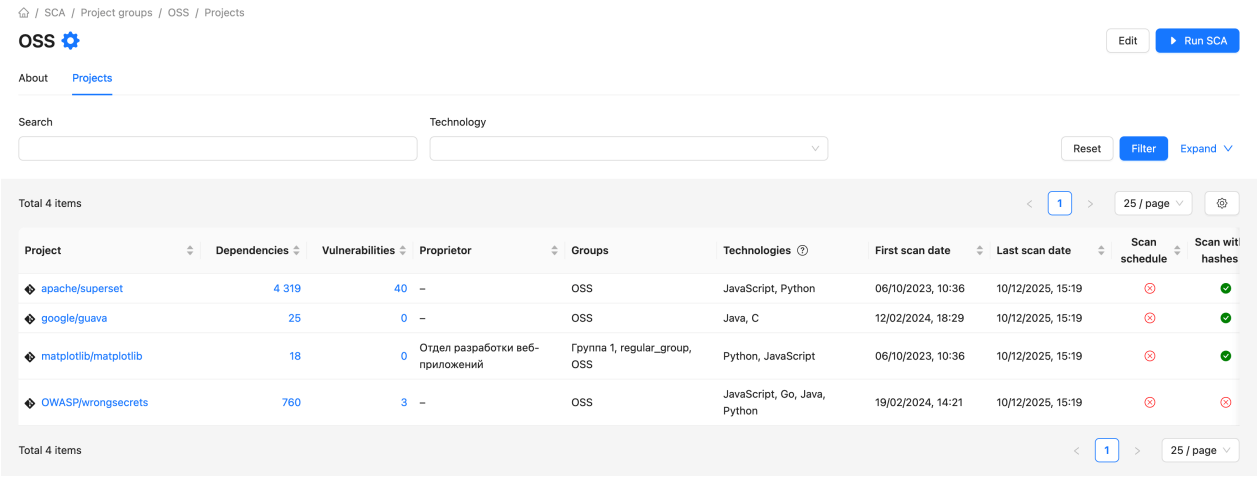
The section also contains the sections **Alerts**, **Vulnerabilities** and **Dependencies** related to the projects of this group.

For convenience, there are standard filters in each section.

#Group composition

The Projects section contains a list of projects that belong to this group.

For ease of operation, it is possible to filter the list of projects.



#CLI agent

The CLI agent **Johnny** is provided with the on-premise version of CodeScoring.

Agent is an executable binary file that parses manifests of well-known package managers, scans Docker images, analyzes C and C++ builds, parses archives, and searches for direct inclusions of Open Source libraries by hashes. The agent works in tandem with the platform, receiving enriched data on vulnerabilities, licenses, and configured policies from it, and saving scan results to CLI projects.

By default, the agent build is provided for Linux-compatible systems. Builds for Windows and MacOS are available upon request. You can download the agent executable file via platform using the following addresses:

- `[platform-url]/download/` – page with a list of available executable files;
- `[platform-url]/download/johnny_version` – current version of the binary agent;
- `[platform-url]/download/<file-name>` – download the executable file.

To view the current version and download the file, you need to authorize using an API token.

#Mechanism of operation

When working in source code directory scanning mode, the agent recursively traverses the directory specified in the launch parameters and searches for and parses manifests of known package managers.

In the image scanning mode, the agent examines the file system of the specified image, making an inventory of its component content.

At the end of the work, a **SBOM** file is generated, and information about the vulnerabilities found and triggered policies is displayed in the console.

Example output of found vulnerabilities:

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

Example output of triggered policies:

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

Example output of reachable vulnerabilities:

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()

#Using configuration file

You can manage the parameters of the CLI agent by adding the configuration file `codescoring-johnny-config.yaml` to the directory with the agent. Below is a list of available parameters and an example of config file.

#List of parameters

#Composition analysis parameters

- **project** – name of the project in the CodeScoring platform;
- **save-results** – saving the results in the CodeScoring platform. Used in conjunction with the project name. The default value is `false`;

- **license** – license of the analyzed project, for example `mit`;
- **stage** – development stage. Possible values: `build`, `dev`, `source`, `stage`, `test`, `prod`, `proxy`;
- **bom-path** – path (with file name) where the generated `bom.json` file will be saved;
- **bom-format** – format of the generated SBOM. Possible values: `cyclonedx_v1_6_json`, `cyclonedx_v1_5_json`, `cyclonedx_v1_4_json`, `cyclonedx_v1_6_ext_json`, `cyclonedx_v1_7_json`. Default value: `cyclonedx_v1_6_json`;
- **timeout** – limit on analysis waiting time (in seconds);
- 2024.52.0 **branch-or-tag** – a reference to a repository branch or tag, such as `refs/tags/v1.0` (for the `scan dir` and `scan file` commands);
- 2024.52.0 **commit** – a commit hash in the version control system (for the `scan dir` and `scan file` commands);
- 2024.52.0 **hash** – an image hash (for the `scan image` command);
- 2025.7.0 **cloud-resolve** – use dependency resolution in cloud. Default value is `false`.

#General scan parameters

- **ignore** – directories that will be ignored during scanning;
- **no-summary** – hide summary information on the scan in the console. Default value is `false`;
- **only-hashes** – search for **only** direct inclusions of Open Source libraries by hashes. Default value is `false`;
- **with-hashes** – search for direct inclusions of Open Source libraries by hashes. Default value is `false`;
- **no-recursion** – disable recursive scanning for the `scan dir` command. The default value is `false`.
- **block-on-empty-result** – blocking the build when an empty result is returned. When activated, the agent returns exit code **3** if there are no artifacts for analysis;
- 2026.20.0 **include-envs** – include only the listed dependency environments in the result, comma-separated, e.g. `compile,runtime`. Mutually exclusive with `exclude-envs`;
- 2026.20.0 **exclude-envs** – exclude the listed dependency environments from the result, comma-separated, e.g. `test,dev`. Mutually exclusive with `include-envs`.

#Docker image scanning parameters

- **scan-files** – scanning the file system inside the image. Default value is `false`;
- **insecure-skip-tls-verify** – skip TLS verification when connecting to the image registry. Default value is `false`;
- **insecure-use-http** – use the HTTP protocol when connecting to the image registry. Default value is `false`;
- **registries** – list of configurations for connecting to multiple image registries. Each list item may contain:
  - **authority** – registry URL (for example, `docker.io`, `localhost:5000`);

    - **login** – username for connecting to the registry;
    - **password** – password for connecting to the registry;
    - **token** – token for connecting to the registry. Mutually exclusive with the `login` and `password` parameters.

#C and C++ build scanning parameters

- **build-result** – flag indicating that the input data is the results of a previous build, including compiled artifacts. Default value is `false`;
- **lib-versions** – path to a JSON file with the list of versions of the libraries being analyzed;
- **output** – path to the file where the build analysis results will be saved;
- **unresolved-file** – path to the file where information about libraries with unresolved versions will be saved.

#Parsing parameters for different technologies

#Common parameters

- **enabled** – enables parsers for the given technology;
- **parsers** – set of parsers for manifests.

#Parser parameters

- **enabled** – enables this parser;
- **match** – condition for matching suitable manifests, can be by name (`equal`) or extension (`extension`);
- **properties** – additional properties for environment parsers, such as the path to executables;
- **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** – paths to package managers for dependency resolution in the environment;
- 2026.20.0 **pdm-path** – path to `pdm` for dependency resolution in the environment;
- 2025.45.0 **pipdeptree-path** – path to `pipdeptree` for dependency resolution in the environment;
- 2026.3.0 **bun-path** – path to `bun` for dependency resolution in the environment;
- 2025.45.0 **uv-path** – path to `uv` for dependency resolution in the environment;
- 2025.13.0 **swift-path** – path to `swift` for dependency resolution in the environment;
- **resolve-enabled** – enables dependency resolution in the environment. Default value is `false`;
- **dotnet-args**, **gradle-args**, **maven-args**, **sbt-args**, **npm-args**, **yarn-args**, **pnpm-args**, **composer-args**, **pip-args**, **poetry-args**, **conda-args** – arguments to pass to the corresponding package managers when resolving dependencies in the environment;
- 2026.20.0 **pdm-args** – arguments to pass to `pdm` during dependency resolution in the environment;

- 2025.45.0 **pipdeptree-args** – arguments to pass to `pipdeptree` during dependency resolution in the environment;
- 2026.3.0 **bun-args** – arguments to pass to bun during dependency resolution in the environment;
- 2025.45.0 **uv-args** – arguments to pass to uv during dependency resolution in the environment;
- 2025.13.0 **swift-args** – arguments to pass to `swift` during dependency resolution in the environment;
- **configuration** – configuration for the `gradle-dependency-tree_txt` parser;
- **depth** – parsing depth for the `jar` parser. Default value is 1;
- **python-version** – Python version used for dependency resolution in the environment.

#Archive scanning parameters

- **scan** – scan archives. Default value is `false`;
- **depth** – archive scanning depth. Default value is 1.

#Output parameters

- 2023.48.0 **format** – output format for found vulnerabilities. Default value is `coloredtable`. Export to formats `table`, `text`, `junit`, `sarif`, `csv`, `gl-dependency-scanning-report`, `gl-code-quality-report` is available;
- 2023.48.0 **group-vulnerabilities-by** – field used to group vulnerabilities in the table;
- 2023.48.0 **sort-vulnerabilities-by** – order of fields for sorting vulnerabilities in the table;
- 2025.29.0 **alerts-format** – output format for the policy alerts report. Supported formats: `coloredtable`, `table`, `text`, `json`, `csv`. Default value is `coloredtable`.

#Platform parameters

- **api_url** – platform address;
- **api_token** – token for access to the platform;
- 2026.3.0 **localization** — CLI output localization language. Possible values: en, ru. The default value is en.

#Secret scanning launch parameters

- 2025.13.0 **gitleaks-path** – path to the Gitleaks executable file to be used during scanning;
- 2025.13.0 **gl-secrets-report** – enables the generation of a secrets report in GitLab format. Default: `false`;
- 2025.13.0 **gl-secrets-report-filename** – name of the generated GitLab secrets report file. Default: `gl-secrets-report.json`.

#Gitleaks secret scanning tool parameters

- 2025.13.0 **baseline-path** – path to the Gitleaks baseline report file. All previously detected secrets recorded in this file will be ignored during rescanning;
- 2025.13.0 **enable-rule** – list of rule IDs to be **enabled** during scanning;
- 2025.13.0 **gitleaks-ignore-path** – path to the `.gitleaksignore` file or the directory containing it. Default: `.` (current directory);
- 2025.13.0 **ignore-gitleaks-allow** – ignores `gitleaks:allow` comments. Default: `false`;
- 2025.13.0 **log-level** – logging level. Possible values: `trace, debug, info, warn, error, fatal`. Default: `info`;
- 2025.13.0 **max-decode-depth** – maximum depth for recursive decoding. A value of `0` disables decoding;
- 2025.13.0 **max-target-megabytes** – maximum file size (in megabytes) to be processed. Files exceeding this size will be skipped. Default: `0` (no limit);
- 2025.13.0 **no-banner** – disables the Gitleaks startup banner. Default: `false`;
- 2025.13.0 **no-color** – disables colored output in verbose mode. Default: `false`;
- 2025.13.0 **redact** – masks detected secrets in logs and console output. A value of 0 fully reveals secrets, while 100 completely hides them. Intermediate values, such as 20, mask 20% of the secret. Default: `0`;
- 2025.13.0 **verbose** – enables verbose output during scanning. Default: `false`.

#Example file

```
## 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
```

#Priority of settings

Since agent startup parameters can be configured in multiple ways, if two or more methods are used at the same time, the agent will accept parameters in the following priority order:

1. Value of the scan-technology command (if used);
2. Value of the command flag;
3. Value of environment variable;
4. Value from config file.

#Configuration via environment variables

The console agent parameters can be configured via environment variables. The configuration file structure is used for configuration via environment variables.

#Formation of environment variables

1. **Variable prefix**: All environment variables begin with the `JOHNNY_` prefix.
2. **Section path**: The variable is formed based on the section path in the configuration file. Section separators are replaced with the _ symbol.
3. **Symbol replacement**: The "." and "-" symbols in section names are also converted to the _ symbol.

#Example

Let's look at an example of setting the `block-on-empty-result` flag to block the build when an empty result is received:

- **Path in the configuration file**: `scan.general.block-on-empty-result`;
- **Environment variable**: `JOHNNY_SCAN_GENERAL_BLOCK_ON_EMPTY_RESULT`;

Thus, to change the value of this parameter through environment variables, you need to set the variable `JOHNNY_SCAN_GENERAL_BLOCK_ON_EMPTY_RESULT` with the desired value.

#Priority of settings

Since agent startup parameters can be configured in multiple ways, if two or more methods are used at the same time, the agent will accept parameters in the following priority order:

1. Value of the scan-technology command (if used);
2. Value of the command flag;
3. Value of environment variable;
4. Value from config file.

#Scan command

The agent is launched using the `scan` command with possible scanning options:

- `scan dir` – scanning a directory;
- `scan file` – scanning a file;
- `scan image` – scanning a container image;
- `scan bom` – scanning a SBOM;
- `scan <technology>` - scanning a directory using settings for selected technology;
- `scan build` – scanning a build.

#Launch options

Available and required options for launching the agent for scanning can be viewed using the `help` flag.

```
$ ./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.
```

The `--api_url` parameter must contain the full address of the on-premise platform. The value for `--api_token` can be taken from the platform user profile.

Specifying the `--project` parameter will allow scanning to apply policies related to the selected project.

To specify the path to a SBOM save file, you must add the `--bom-path` parameter to the request or assign the `bom-path` variable in the config file. By default, the SBOM is saved in the startup directory in the file `bom.json`.

#Run results

Depending on the run result and launch parameters, the agent returns the corresponding exit code:

- **0** – successful run, no issues were detected;
- **1** – as a result of the scan, problems were found that correspond to the configured security policies, user action is required;
- **2** – run failure;
- **3** – empty result, no artifacts were found for analysis. Returned only if the `--block-on-empty-result` parameter is set to `true`.

#Priority of settings

Since agent startup parameters can be configured in multiple ways, if two or more methods are used at the same time, the agent will accept parameters in the following priority order:

1. Value of the scan-technology command (if used);
2. Value of the command flag;
3. Value of environment variable;
4. Value from config file.

#Launch without the platform

If the `--api_url` and `--api_token` parameters are not specified, the scan will be launched without interacting with the CodeScoring platform. The scan will generate an SBOM file containing only a list of components and their versions without any additional information.

#Scanning a directory

Scanning a directory is done using the `scan dir` subcommand.

During launching the agent:

1. Recursively goes through the entire contents of the specified directory (if a particular manifest is specified, it processes only that one)
2. Identifies manifest files and parses them
3. Hashes each file (when ran with `--with-hashes`)
4. Generates a request for platform
5. After receiving the result, it shows summary information on the found manifests, dependencies, vulnerabilities, triggered policies and more detailed information on each vulnerability and triggered policy
6. Additionally, a file `bom.json` is generated in the current directory, containing the complete Software Bill of Materials in the **CycloneDX** format.

Depending on the result of work and launch parameters, the agent returns the corresponding exit code.

By default, the agent traverses the contents of a directory recursively (including subdirectories). For non-recursive scanning, you need to add the `--no-recursion` option to the `scan dir` command.

Request example

```
./johnny scan dir . \
--api_token <api_token> \
--api_url <api_url> \
--ignore .tmp --ignore fixtures --ignore .git
```

#Command parameters

The **scan dir** command has three unique parameters, in addition to the general scan command settings:

- `--branch-or-tag` – a reference to a branch or repository tag. Format `^refs/(heads|tags)/.+` (e.g. `refs/tags/v1.0`);
- `--commit` – specifying the commit hash;
- `--no-recursion` – disabling recursive directory scanning.

For a summary of available command parameters and usage instructions, you can call the command with the `-h, --help` flag.

#Scanning a file

If you need to scan a single manifest within a directory, you can use the `scan file` command.

During launching the agent:

1. Identifies the format of the specified file and parses the contents.
2. Generates a request to the platform to analyze the contents.
3. After receiving the results, displays general information about the found manifests, dependencies, vulnerabilities, and policies that have been triggered.
4. Additionally, a `bom.json` file is created in the current directory, containing the full Software Bill of Materials in the **CycloneDX** format.

Depending on the launch parameters, the agent returns the appropriate exit code:

- **0** – successful scanning, no issues were detected;
- **1** – as a result of the scan, problems were found that correspond to the configured security policies, user action is required;
- **2** – scanning error;
- **3** – empty result, no artifacts were found for analysis. Returned only if `--block-on-empty-result` parameter is `true`.

Example of running command

To scan only one file, without processing nested directories or other manifests, you must specify the file path when running the command.

```
./johnny scan file path/to/file \
--api_token <api_token> \
--api_url <api_url>
```

#Command parameters

The **scan file** command has three unique parameters, in addition to the general scan command settings:

- `--branch-or-tag` – a reference to a branch or repository tag. Format `^refs/(heads|tags)/.+` (e.g. `refs/tags/v1.0`);
- `--commit` – specifying the commit hash;
- `--parser` – the parser to use.

For a summary of the available command parameters and usage instructions, you can call the command with the `-h, --help` flag.

#Available parsers

Language	Parsers
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`
Objective-C	`objective_c.podfile`, `objective_c.podfile_lock`, `objective_c.podspec`
Rust	`rust.cargo_toml`, `rust.cargo_lock`

Language	Parsers
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

#Scanning an archive

To scan archives for manifests, use the `--scan-archives` flag.

By default, scanning archives only works for one nesting level. To specify scanning depth, you need to add the `--scan-depth` parameter to the command or specify the `depth` variable in the `scan-archives` section in the config file.

Request example for scanning archives:

```
./johnny scan dir . \
--api_token <api_token> \
--api_url <api_url> \
--ignore .tmp --ignore fixtures --ignore .git \
--scan-archives\
--scan-depth 2
```

Supported archive formats:

- .jar
- .rar
- .tar
- .tar.bz2
- .tbz2
- .tar.gz
- .tgz
- .tar.xz
- .txz
- .war
- .zip
- .aar
- .egg

- `.hpi`
- `.nupkg`
- `.whl`

#Scanning a container image

The agent supports image scanning functionality in the OCI and Docker standards and can be launched in one of the following ways, indicating:

- path to the **tar** archive created using **docker save**:

```
./johnny scan image ./my_own.tar \
--api_url <api_url> \
--api_token <api_token>
```

- image name located in the **Docker** daemon, **Podman**:

```
./johnny scan image docker:python:3.9 \
--api_url <api_url> \
--api_token <api_token>
```

- image name from the public **Docker HUB**:

```
./johnny scan image python:3.9 \
--api_url <api_url> \
--api_token <api_token>
```

- image name from private **registry**:

Before working with a private repository, you need to run the command `docker login`

```
./johnny scan image pvt_registry/johnny-depp:<version> \
--api_url <api_url> \
--api_token <api_token>
```

Alternatively, you can log in to the private registry using environment variables:

- `JOHNNY_REGISTRY_AUTH_AUTHORITY` - URL to the registry (for example "docker.io", "localhost:5000", etc.);
- `JOHNNY_REGISTRY_AUTH_LOGIN` - login;
- `JOHNNY_REGISTRY_AUTH_PASSWORD` - password;
- `JOHNNY_REGISTRY_AUTH_TOKEN` - token;

or through similar variables in the config file:

- `authority`;

- `login`;
- `password`;
- `token`.

Note: token and login with password are interchangeable.

#Scanning the file system inside a Docker image

To scan files inside an image, you need to add the `--scan-files` parameter to the command or specify the `scan-files` variable in the `image` section in the config file.

When scanning a file system, you can use the `--ignore` option to exclude specific files from analysis. For example:

```
./johnny scan image ./my_own.tar \
--api_url <api_url> \
--api_token <api_token> \
--scan-files \
--ignore "**/node_modules"
```

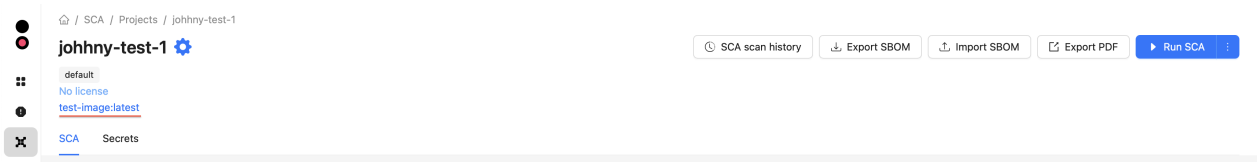
#Command parameters

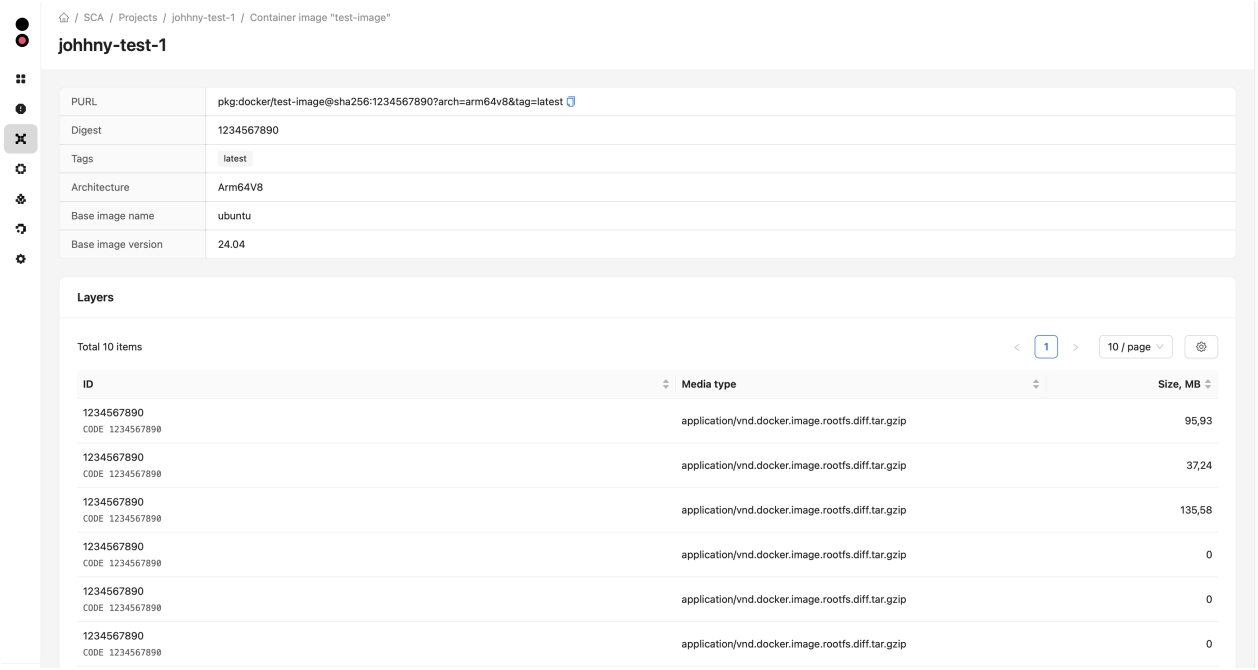
The **scan image** command has four unique parameters, in addition to the general scan command settings:

- `--hash` – specifying the image hash;
- `--scan-files` – scanning files in the image.
- `--branch-or-tag` – a reference to a branch or repository tag. Format `^refs/(heads|tags)/.+` (e.g. `refs/tags/v1.0`);
- `--commit` – specifying the commit hash.

For a summary of the available command parameters and usage instructions, you can call the command with the `-h, --help` flag.

When image scan results are saved, you can view detailed information about the image and its layers in the project:





#Scanning a build for C and C++ languages

If the Conan package manager and corresponding manifests are not used to build a C/C++ project, Johnny can be run in a special mode to analyze the output of the build process to get a list of used libraries.

Johnny analyzes the build process, using the compiler flags and identifying the libraries used. Then, using the system cache, the location of the libraries and their source are determined.

The version of the used library can be found if the library was statically linked. In this case, the agent analyzes the `.pc` file, which contains metadata about the component.

#Example of work

A `buildConfig` JSON file is added to the project, describing the sequence of commands for the build. For example:

```
{
 //  Commands for execution
 "commands": [
    {
   // Command
   "command": "make",
   // Command execute parameters
   "flags_and_args": "clean"
  },
  {
   "command": "./configure"
  },
```

```
  {
   "command": "make",
   // analyze this command
   "do_analyze": true
  }
 ]
}
```

Then the build analysis command is called and the path to the config file is specified:

```
./johnny scan build ./buildConfig.json
```

As a result of the work, the agent displays one of the codes:

- **0** – successful analysis, no problems were found in the build;
- **1** – as a result of the scan, problems were found that correspond to the configured security policies, user action is required;
- **2** – analysis failed;
- **3** – empty result, no artifacts were found for analysis. Returned only if `--block-on-empty-result` parameter is `true`.

Important: at the moment the command supports only Debian-based and RPM-based Linux distributions.

#Command parameters

The **scan build** command has six unique parameters, in addition to general scan command settings:

- `--build-result` – input is the result of the previous build process, including compiled artifacts;
- `--lib-versions` – path to a JSON file with a list of versions of the libraries being analyzed;
- `--output` – path to a file where the scan results will be saved;
- `--unresolved-file` – path to a file where information about libraries with unresolved versions will be saved.
- `--branch-or-tag` – a reference to a branch or repository tag. Format `^refs/(heads|tags)/.+` (e.g. `refs/tags/v1.0`);
- `--commit` – specifying the commit hash.

For a summary of available command options and usage instructions, you can call the command with the `-h, --help` flag.

#Scanning with eBPF

eBPF (extended Berkeley Packet Filter) is a technology in the Linux kernel that allows to safely run user code in response to system events, such as network traffic, system calls, or process actions.

The feature of the `scan build ebpf` command is that the analysis is performed not according to the protocol of the build tool, but by monitoring the launched processes and their parameters via the eBPF mechanism.

Important: to ensure work with the eBPF mechanism, the command must be run with **root privileges**.

#Example of work

The parameters and results of the command are similar to the `scan build` command.

```
./johnny scan build ebpf ./buildConfig.json
```

#Scanning a SBOM

In case of scanning an existing Software Bill of Materials (SBOM) in the **CycloneDX** format, you can use the `scan bom` command.

When the agent is launched, it does the following:

1. Validates SBOM with the specified version json schema.
2. Parses the specified SBOM.
3. Generates a request to the platform to analyze the contents.
4. After the analysis is complete, it displays summary information about the results in the console, as well as tables with the vulnerabilities found and the policies that were triggered.
5. Additionally, the `bom.json` file is created in the current directory, containing the supplemented SBOM.

Depending on the launch parameters, the agent returns the appropriate exit code:

- **0** – successful scanning, no problems were detected;
- **1** – as a result of the scan, problems were found that correspond to the configured security policies, user action is required;
- **2** – scanning error;
- **3** – empty result, no artifacts were found for analysis. Returned only if `--block-on-empty-result` parameter is `true`.
- **5** - SBOM validation error.

Command example

To scan SBOM, you must specify the path to it when running the command.

```
./johnny scan bom path/to/bom \
--api_token <api_token> \
--api_url <api_url>
```

#Command parameters

The **scan bom** command has two unique parameters, in addition to the general scan command settings:

- `--branch-or-tag` – a reference to a branch or repository tag. Format `^refs/(heads|tags)/.+` (e.g. `refs/tags/v1.0`);
- `--commit` – specifying the commit hash.

#Scanning a technology

To make it easier to work with different ecosystems, the agent allows to scan individual technologies with a set of pre-defined settings. Scanning in this case is performed using the `scan <technology>` subcommand.

The behaviour of the agent during scanning is similar to that of the `scan dir` command, but has the following differences:

1. Directory traversal is always performed non-recursively (as when the `--no-recursion` flag is used in the `scan dir` command);
2. Only manifests belonging to the selected technology are processed;
3. All parsers of the selected technology are used, including dependency resolution in the environment. Other settings are ignored;

#List of supported technologies

Technologies are listed as they are used in the `scan <technology>` command:

- clang
- conda
- csharp
- go
- java
- js
- objective_c
- php
- python
- ruby
- rust

Example of running the command

```
./johnny scan java . \
--api_token <api_token> \
--api_url <api_url>
```

You can call the command with the `-h, --help` flag to summarise the available command parameters and usage instructions.

#Scanning for secrets

Important

To use this functionality, the platform must have active CodeScoring.Secrets module.

Scanning for secrets is performed using the following commands:

- `johnny secrets gitleaks dir` — scan files in the specified directory;
- `johnny secrets gitleaks git` — scan the commit history of a local git repository;
- `johnny secrets trufflehog dir` — scan files in the specified directory using Trufflehog;
- `johnny secrets trufflehog git` — scan the commit history of a local git repository using Trufflehog.

Important: The agent only works with Gitleaks version 8.19.0 and above, and with Trufflehog version 3.93.8 and above.

When launched, the agent:

1. Analyzes files in the specified directory or the commit history of the repository for secrets (passwords, tokens, access keys, etc.).
    - excludes files and directories specified in `.gitleaksignore`;
    - ignores secrets recorded in the Gitleaks report if `baseline-path` is specified.
2. Generates results for the found secrets, saves them to the CodeScoring platform if necessary, and creates a report in GitLab format.

#Scanning a git repository

The `git` mode allows scanning the commit history of a local git repository. Unlike the `dir` mode, which analyzes the current state of files, the `git` mode checks for secrets across all commits in the repository or within a limited range.

#Example of running the command for 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
```

#Example of running the command for 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
```

#Git mode parameters

The `johnny secrets gitleaks git` and `johnny secrets trufflehog git` commands support the following additional parameters:

- `--git-ref` – git branch, tag, or commit ref to scan. If not set, all refs are scanned. Examples: `main`, `v1.0.0`, `a1b2c3d`;
- `--git-depth` – limit the scan to this many commits from the tip of the history. The value `0` means no limit.

#Gitleaks config example

Example of Gitleaks config which extends a default configuration and adds a new rule with regex

```
title = “Custom gitleaks config”

[extend]
useDefault = true

[[rules]]
id = “custom-generic-password”
description = “Detected a Generic password”
regex = ‘’‘passw(?:or)d.+’‘’
entropy = 1
```

#Example of running the command

```
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
```

This command starts scanning secrets in the current directory, ignoring files listed in `.gitleaksignore`, sends the results to the CodeScoring platform, and generates a report in GitLab format, saving it to `secrets-report.json`.

#Command Parameters

The **johnny secrets gitleaks dir**, **johnny secrets gitleaks git**, **johnny secrets trufflehog dir**, and **johnny secrets trufflehog git** commands have the following unique parameters:

#Secrets search startup parameters

- `--commit` – commit hash to associate with found secrets when saving results. Used only for `dir` commands when the tool does not determine the commit on its own (e.g.: `--commit a1b2c3d`);
- `--gl-secrets-report` – enable generation of a report on found secrets in GitLab format.
- `--gl-secrets-report-filename` – name of the output file for the report in GitLab format (by default `gl-secrets-report.json`).

#Gitleaks parameters

- `--gitleaks-path` – path to the Gitleaks executable file that will be used during scanning. If not set system command `gitleaks` will be executed.
- `--gitleaks-config` - path to the Gitleaks config file;
- `--baseline-path` – path to the Gitleaks report file, which is used as a baseline for ignoring previously found secrets;
- `--enable-rule` – list of rule IDs that will be **enabled** during scanning;
- `--gitleaks-ignore-path` – path to `.gitleaksignore` file or directory containing it, to add fingerprints of secrets discovered earlier;
- `--ignore-gitleaks-allow` – ignore `gitleaks:allow` comments that mark lines as safe to ignore;
- `--log-level` – logging level that controls the verbosity of output messages. Possible values: `trace`, `debug`, `info`, `warn`, `error`, `fatal`;
- `--max-decode-depth` – maximum depth of recursive decoding when searching for secrets. The value `0` disables decoding;
- `--max-target-megabytes` – maximum size of analyzed files in megabytes. Files larger than this size will be skipped;
- `--no-banner` – disables the Gitleaks banner that is displayed when the tool is launched;
- `--no-color` – disables color output for verbose mode;
- `--redact` – masks found secrets in logs. You can specify intermediate values (for example, `20` to hide 20% of the secret);
- `--verbose` – enables verbose output, providing more information about the scanning process.

#Trufflehog parameters

- `--trufflehog-path` – path to the Trufflehog executable file that will be used during scanning. If not set, system command `trufflehog` will be executed;
- `--trufflehog-config` – path to the Trufflehog configuration file;
- `--concurrency` – number of concurrent workers used during scanning;
- `--no-verification` – do not verify found secrets;
- `--only-verified` – output only verified secrets;
- `--include-paths` – path to a file with newline-separated regex patterns for files to include in scanning;
- `--exclude-paths` – path to a file with newline-separated regex patterns for files to exclude from scanning;
- `--trufflehog-log-level` – Trufflehog logging level. Possible values: `debug`, `info`, `warn`, `error`.

For a summary of available command options and usage instructions, you can call the command with the `-h, --help` flag.

#Vulnerability reachability analysis

What is reachability

Reachability analysis checks whether a potentially vulnerable piece of code can actually be executed when using the application. This allows filtering out the "noise" and focusing on exploitable issues.

Johnny CLI agent can analyze whether vulnerabilities are reachable from source code. To use this feature, you need to set two parameters:

- `cg-path` — path to the call graph file: for `java`, `python`, `go`, `kotlin`, and `csharp` (C#), use the Svace format; for `javascript`, use JSON produced by Joern (see below);
- `cg-lang` — programming language for which the call graph was built. Supported values are `java`, `python`, `go`, `kotlin`, `csharp` (C#), and `javascript`.

#Building the call graph

#Using Svace

1. Download the Svace module `https://REGISTRY_USERNAME:REGISTRY_PASSWORD@REGISTRY_URL/#browse/browse:files:codescoring%2Fsvace-callgraph`

2. Obtain a user token in CodeScoring (`/cabinet/profile`)

3. Run Svace on the project's source code. This step is best performed within or after the build stage in your CI/CD pipeline.

    1. Initialization

```
svace init
```

2. Instrumented build

   ```
   svace build <build command>
   ```

   Example of building Java project:

   ```
   svace build mvn clean package
   ```

   Example for Go:

   ```
   svace build go build -a main.go
   ```

   Example for Python:

   ```
   svace build --python .
   ```

   Example for Kotlin:

   ```
   svace build ./gradlew clean build
   ```

   Example for C#:

   ```
   svace build dotnet build
   ```

3. Analyze results and generate the call graph

   ```
   svace analyze --build-call-graph-only --license-
   server-url "http(s)://<codescoring_host>" --license-
   server-token "<token from step 2>"
   ```

4. Upon successful completion of all steps, a file named `.svace-dir/analyze-res/call-graph-results/<project_name>-graph-order.json` containing the call graph will appear in the project directory.

   Saving a file

   Prior to Svace version 5.0.260311, the call graph file was saved to the folder `.svace-dir/analyze-res/call-graph`

5. Launch the scan using Johnny, for example:

```
johnny-linux-amd64 scan dir . --api_url "http(s)://
<codescoring_host>" --api_token "<token from step 1>"  --
cg-path .svace-dir/analyze-res/call-graph-results/<projec
t_name>-graph-order.json  --cg-lang java
```

#Using Joern (JavaScript)

For JavaScript, the reachability call graph is built with Joern. Run the following in the directory that contains your project source code:

```
joern-parse .
joern-slice usages cpg.bin
```

The `joern-parse` command creates a CPG in `cpg.bin`. The `joern-slice usages` command writes JSON output. By default the file is `slices.json` in the current folder. Pass the path to that file in Johnny's `--cg-path` argument. You can set the output file explicitly with `joern-slice`'s `-o` option; see Joern CPG slicing documentation.

Example scan invocation:

```
johnny-linux-amd64 scan dir . --api_url "http(s)://
<codescoring_host>" --api_token "<token from profile>" --cg-
path <path_to_joern_json> --cg-lang javascript
```

#Remote analysis

In **Svace 5.0.260311**, it is now possible to use a remote analysis server. This allows you to take the construction of the call graph beyond the CI/CD pipeline, reduce the load on the assembly servers and optimize the continuous integration process.

#Server configuration steps

1. Specify environment variables

```
SVACE_LIC_SERVER_URL=http(s)://<codescoring_host>
SVACE_LIC_SERVER_TOKEN=<токен CodeScoring>
```

1. Initialization

```
svace server init
```

1. Start the server

```
svace server start
```

1. Submit the project for analysis. After configuring the server, you do not need to specify the `--license-server-url` and `--license-server-token` license parameters.

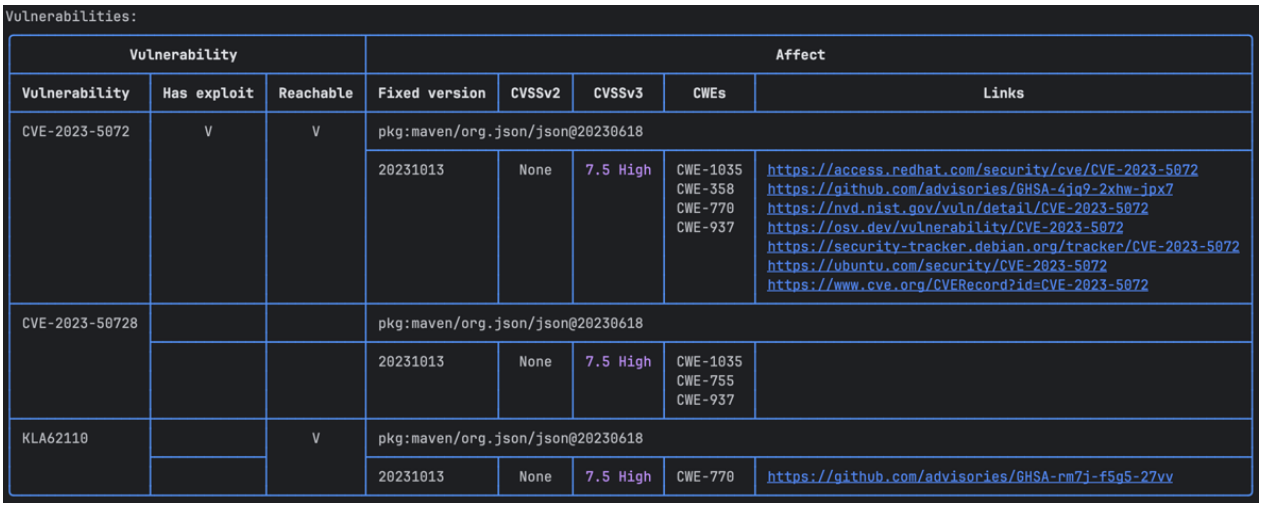
   ```
   svace remote --host <server address> analyze --build-call-graph-only
   ```

   If necessary, you can add the parameters: `--port`, `--login`, `--pass`, etc.

For more information about the remote analysis server settings, see the Svace documentation.

#Viewing results

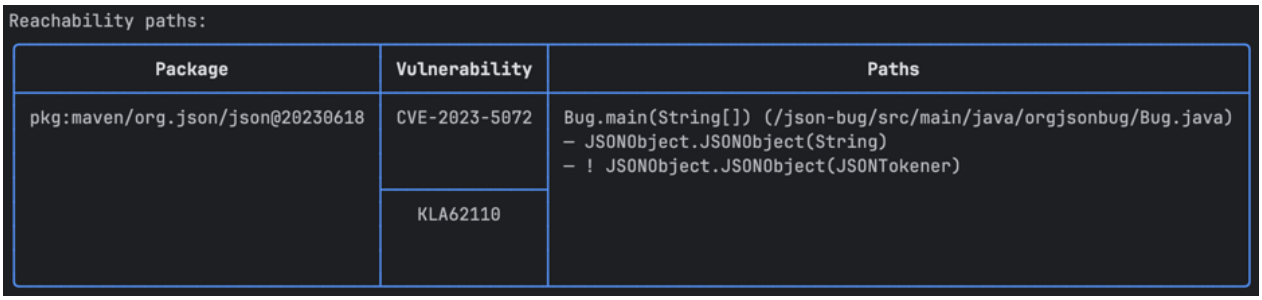
In the vulnerabilities table, vulnerabilities with identified reachable call paths will be marked in the corresponding column:



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

Additionally, at the end of the report, another table listing the call trees for vulnerabilities will be available:



Reachability paths:

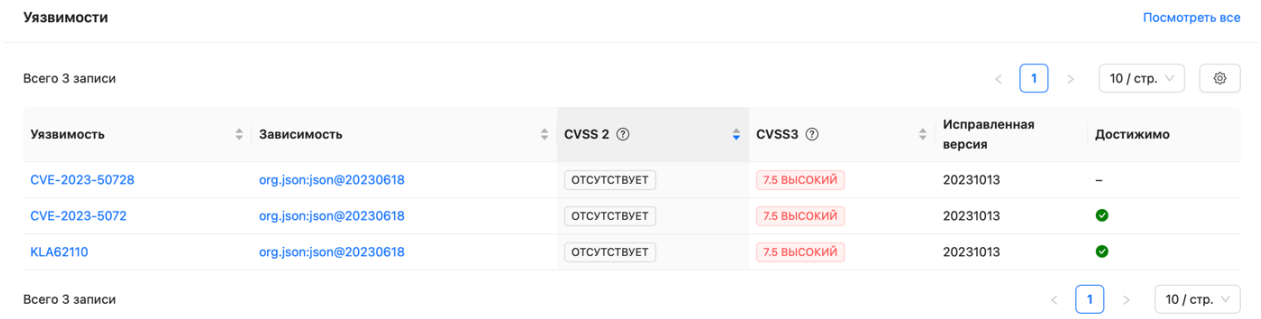
Package	Vulnerability	Paths
pkg:maven/org.json/json@20230618	CVE-2023-5072	Bug.main(String[]) (/json-bug/src/main/java/orgjsonbug/Bug.java) – JSONObject.JSONObject(String) – ! JSONObject.JSONObject(JSONTokener)
	KLA62110	

Example for a larger project:

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()

If the `--save-results` flag was specified, reachability results will appear in the "Reachable" column of the vulnerabilities table:



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.

#SBOM signing and verification

Signing and verification using RSA SHA256 digital signatures is supported to confirm the integrity and authenticity of SBOM files. This functionality is available starting with binary agent version **2025.29.0**.

Both commands can be used without setting parameters `--api_url` and `--api_token`.

Important: only RSA keys are supported, and they must be in PEM format.

#SBOM file signing

The `sign bom` command is used to create a digital signature for an SBOM file. It has the following parameters:

- `--private-key <path>` - path to the RSA private key in PEM format (**required**);
- `--include-public-key` - include the public key in the SBOM file (optional).

Launch examples

```
## Signing a file with a private key
./johnny sign bom <bom_json> \
--api_token <api_token> \
--api_url <api_url> \
--private-key <private_key_pem>

## Signing a file with a public key included in SBOM
./johnny sign bom <bom_json> \
--api_token <api_token> \
--api_url <api_url> \
--private-key <private_key_pem> \
--include-public-key
```

#Verifying the signature of an SBOM file

The `verify bom` command is used to verify the signature of an SBOM file. It has the following parameters:

- `--public-key <path>` - path to the RSA public key in PEM format (optional). If this parameter is set and SBOM file contains public key, key from SBOM file will be used.

Run examples

```
## Verification using a public key from a file
./johnny verify bom <bom_json> \
--api_token <api_token> \
--api_url <api_url> \
--public-key <public_key_pem>

## Verification using a key from an SBOM file
./johnny verify bom <bom_json> \
--api_token <api_token> \
--api_url <api_url>
```

#Results

The agent returns the following exit code:

- 0: successful run;
- 4: signature verification error.

#Launching in Docker

To run the agent via Docker, you currently need active authorization in the registry with system images.

Example request on the current directory

```
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
```

The `-a stdout` parameter is required to correctly display the **Vulnerabilties** and **Policy Alerts** tables when running the agent via Docker.

#Integration into Gitlab CI

The CLI agent supports integration into Gitlab CI using the `.gitlab-ci.yaml` file and is supplied as both a Docker image and a binary file.

#Johnny Docker image

Example .gitlab-ci.yaml file (Docker image)

<version> must be replaced with the agent version. A list of current versions with descriptions is available in 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
```

#Johnny binary file

To use the agent binary file, you must first perform the following steps on the gitlab-runner machine:

1. Download the file with the command

```
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
```

or

```
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` must be replaced with the agent version. A list of current versions with descriptions is available in Changelog. `REGISTRY_URL`, `REGISTRY_USERNAME` and `REGISTRY_PASSWORD` should be replaced with the URL, login and password received from the vendor. 2. Allow file execution

```
chmod +x /usr/local/bin/johnny
```

Example .gitlab-ci.yaml (binary file)

```
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
```

#Integration into Jenkins Pipeline

The CLI agent supports integration into Jenkins in two ways: through `Jenkinsfile` and a specialized plugin.

#Adding agent to Jenkinsfile

#Using a Docker image

Example of adding agent to a pipeline (Docker image)

```
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 .
 """
 }
 }
```

```
 }
}
```

#Using a binary file

To use the console agent binary file, you must first perform the following steps on the Jenkins machine:

1. Download the file with the command

```
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
```

or

```
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
```

The `JOHNNY_VERSION` variable must be replaced with the agent version. A list of current versions is available in the Changelog section.

The `REGISTRY_URL`, `REGISTRY_USERNAME` and `REGISTRY_PASSWORD` variables should be replaced with the URL, login and password received from the vendor.

1. Allow file execution

```
 chmod +x /usr/local/bin/johnny
```

Example of running agent in pipeline (binary file)

```
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 .
 """
 }
 }
 }
}
```

#Integration into Gitflic CI

Using the johnny console agent, you can configure component scanning in GitFlic CI. The supported runner types are GitFlic shell and GitFlic docker.

#Using the agent with the GitFlic shell runner type

To use the console agent with the GitFlic Shell runner type, you must first perform the following steps on the machine with the agent:

1. Download the file using the command

   ```
   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
   ```

   or

   ```
   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` should be replaced with the agent version. The list of current versions with descriptions is available in the Changelog section. `REGISTRY_URL`, `REGISTRY_USERNAME` and `REGISTRY_PASSWORD` should be replaced with the URL, login and password received from the vendor.

2. Allow file execution

```
chmod +x /usr/local/bin/johnny
```

Example of requesting the agent binary in `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"
```

The results of the scan can be managed in the **Security** tab in the project interface.

#Using an agent with the GitFlic docker runner type

To use the console agent with the GitFlic docker runner type, you must first perform the following steps on the machine with the agent:

1. Download the file with the command

   ```
   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
   ```

   or

   ```
   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` should be replaced with the agent version. The list of current versions with descriptions is available in the Changelog section. `REGISTRY_URL`, `REGISTRY_USERNAME` and `REGISTRY_PASSWORD` should be replaced with the URL, login and password received from the vendor.

2. Copy the agent to the container that you plan to use it in

   ```
   docker cp ./johnny CONTAINER:/usr/bin
   ```

3. Allow file execution

```
docker exec CONTAINER chmod +x /usr/bin/johnny
```

4. Save changes to the container

```
docker commit <container name><repository>:<tag>
```

Important: if necessary, save the container to a remote repository.

An example of requesting the agent binary file in `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"
```

You can manage the results of the scan in the **Security** tab of the project interface.

#Connecting to the registry and scanning images

Example of an image scan using the agent 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}"
```

Important: access to the file `/v2/\_catalog` in GitFlic is disabled for security reasons. Currently, it is not possible to recurse through all images in the registry.

#Configuring security policies during scanning

1. Configure policies in the CodeScoring platform
2. Run the pipeline using the default scan settings

```
    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. When the policies are triggered, the agent will exit with an error code and the GitFlic runner will automatically stop the pipeline.

Important: GitFlic currently does not have a mechanism for receiving artifacts when a task completes with an error. Due to this, viewing a report on the artifact that caused the pipeline to stop is not possible in the GitFlic web interface.

#Saving scan results

To save scan results in an on-premise platform, you need to add the `--save-results` and `--project` parameters to the command or specify the following variables in the config file:

- `project` — the name of the CLI project in the system into which the results will be saved;
- `save-results` — flag for saving results, default value is **false**.

If the CLI project has not been created in the system in advance, you can specify the `--create-project` parameter in the call command or in the config file. For a new project the following parameters can be specified:

- `--project-group` — the group name. If a group with this name does not exist and the `--create-project-group` flag is provided, it will be created before adding the project to the group;
- `--project-proprietor`;
- `--project-categories`.

An example of a request to save scan results to a new project

```
./johnny scan dir . \
--api_token <api_token> \
--api_url <api_url> \
--save-results \
--create-project \
--project "project-name" \
--project-group "group" \
--project-proprietor "proprietor"
```

#Exporting scan results

The Johnny console agent supports exporting scan results in various formats. This allows to adapt reporting to different needs, including integration with vulnerability management systems. By default all reports are rendered in English but you can use `--localization ru` flag to switch to Russian.

#Report on found vulnerabilities

#Available formats

- **coloredtable** – colored table in the console. Default format;
- **table** – simple table;
- **text** – text report;
- **junit** – used in CI/CD (Jenkins, GitLab CI, GitHub Actions);
- **sarif** – exported to DefectDojo and other vulnerability management systems;
- **csv** – used in BI systems, Excel, Pandas, SQL;

- **gl-dependency-scanning-report** – report format for GitLab Dependency Scanning;
- **gl-code-quality-report** – report format for GitLab Code Quality;
- **gl-secrets-report** – report format for GitLab Secret Detection.

#Usage example

If necessary, you can specify multiple formats, separating them with commas, for example:

```
./johnny scan file path/to/file \
--api_token <api_token> \
--api_url <api_url> \
--format "coloredtable, junit>>junit.xml"
```

In this example, the output will be in the `coloredtable` format to the console, and also saved to the `junit.xml` file in the `junit` format.

#Report on triggered policy alerts

#Available formats

- **coloredtable** – colored table in the console. Default format;
- **table** – simple table;
- **text** – text report;
- **json** – a structured format based on JavaScript Object Notation, suitable for data processing;
- **csv** – text format for table data;

#Usage example

If necessary, you can specify multiple formats, separating them with commas, for example:

```
./johnny scan file path/to/file \
--api_token <api_token> \
--api_url <api_url> \
--alerts-format "coloredtable, json>>alerts.json"
```

In this example, the output will be in the `coloredtable` format to the console, and also saved to the `alerts.json` file in the `json` format.

#Report on policy ignoring

The report is generated using the `--policy-ignores` flag. The report includes policy ignores for the project specified in `--project`.

#Available formats

- **coloredtable** – colored table in the console. Default format;
- **table** – simple table;
- **text** – text report;
- **json** – a structured format based on JavaScript Object Notation, suitable for data processing;
- **csv** – text format for table data;

#Usage example

If necessary, you can specify multiple formats, separating them with commas, for example:

```
./johnny scan file path/to/file \
--api_token <api_token> \
--api_url <api_url> \
--project <project> \
--policy-ignores \
--ignores-format "coloredtable, json>>ignores.json"
```

In this example, the output will be in the `coloredtable` format to the console, and also saved to the `ignores.json` file in the `json` format.

#Dependency resolution in the build environment

Package managers of some ecosystems do not include transitive dependencies in manifests by default. For high-quality composition analysis when working with them, it is recommended to use the dependency resolution mechanism in the build environment.

When resolving dependencies in the environment, the agent checks for the absence of a lock file. If a lock file is detected, the resolution is not performed even if the corresponding flags are present, and results are taken from the detected lock file. An exception applies to technologies where the name and location of the lock file are not fixed by the package manager and can be passed as a parameter to the resolution.

#Configuring dependency resolution

Dependency resolution parameters in the environment, paths to the package manager, and execution parameters are controlled by the following options in the `scan` command:

- `--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`

Example command:

```
./johnny \
scan dir . \
--api_token <api_token> \
--api_url <api_url> \
--dotnet-resolve \
--dotnet-path <path/to/dotnet>
```

If necessary, the listed parameters can be added to the agent configuration file.

#Supported ecosystems

#.NET

Main manifest used for dependency resolution: `.csproj`.

For .NET projects the agent runs:

```
dotnet restore
```

After that, the file `obj/project.assets.json`, which contains complete information about dependencies and their versions, is analyzed. Execution is performed in the directory where the `.csproj` file is located.

In case `.sln` manifest is detected dependency resolution will be executed in the context of the solution. Resolution is performed only if the `obj/project.assets.json` file does not exist for one or more components of the solution.

#Go

Main manifests used for dependency resolution: `go.mod`, `go.sum`.

The agent uses data from the `go.mod` and `go.sum` files, adding entries from `go.sum` that are missing in `go.mod` (only lines without the `/go.mod` suffix). Then it runs:

```
go mod graph
```

The resulting list of `parent → child` pairs is used to build the dependency tree. If the parent is not specified, the following command is used:

```
go mod why <package>
```

If the parent relationship is not established, the dependency is excluded with a warning.

#Gradle

Main manifests used for dependency resolution: `build.gradle`, `build.gradle.kts`.

For Gradle dependency resolution by default you must specify:

```
--gradle-path ./gradlew
```

With this value set, the following custom task is run first:

```
./gradlew CodeScoring_All_Dependencies --configuration <config
uration>
```

If the task is missing, the standard command is used:

```
./gradlew dependencies --configuration <configuration>
```

The agent analyzes the console output and forms the dependency graph.

#Additional info

When a gradle-dependency-tree.txt file is present in the directory, the existing file will be used without creating a new one.

#Maven

Main manifest used for dependency resolution: `pom.xml`.

For Maven projects the command is:

```
mvn dependency:tree -f <pom.xml> -DoutputFile=<tmpdir/mdt.json>
```

The agent parses the `mdt.json` file, which contains the complete dependency structure. If the `maven-dependency-tree.txt` file exists, it will be processed as a standalone lock file, and resolution will not be performed.

#npm

Main manifest used for dependency resolution: `package.json`.

The agent runs:

```
npm install
```

Then it analyzes the generated `package-lock.json` file, which records the dependency tree and the versions used.

#pnpm

Main manifest used for dependency resolution: `package.json`.

The command executed:

```
pnpm install
```

The lock file `pnpm-lock.yaml` is analyzed, which contains data about all dependencies.

#yarn

Main manifest used for dependency resolution: `package.json`.

The command executed:

```
yarn install
```

The agent parses `yarn.lock`, which contains information about dependencies and their versions.

#bun

Main manifest used for dependency resolution: `package.json`.

The command executed:

```
bun install --lockfile-only
```

The agent parses `bun.lock`, which contains information about dependencies and their versions.

Presence of a bun.lockb file

To resolve dependencies in a project that contains `bun.lockb`, the agent must be provided with the following flag:
`--bun-args '--save-text-lockfile --frozen-lockfile'`

Package manager behavior

Due to the specifics of how the bun package manager implements the creation of the `bun.lock` file, the set of deep transitive dependencies may change between two runs.

#pip

For Python projects the command used is:

```
pip freeze
```

The output records the list of installed dependencies and their versions. The results reference a pseudo file `codescoring_pip_for_freeze`.

#pipdeptree

For Python projects the command used is:

```
pipdeptree
```

The output records the list of installed dependencies and their versions from the dependency tree. For projects organized around `requirements.txt` and `Pipfile`, the environment scanned by pipdeptree will be filtered based on the contents of these manifests to display the relevant set of dependencies. For projects with `pyproject.toml`, filtering will be performed by `project.name`. The results reference a pseudo file `codescoring_pipdeptree`.

Other manifests in results

In order to prevent the dependencies of the main project manifest (e.g., requirements.txt) from appearing in the analysis results alongside the pipdeptree analysis results, it is recommended to exclude this manifest from scanning:

```
    johnny scan python . \
      --pipdeptree-resolve \
      --ignore "requirements.txt"
```

#Poetry

Main manifest used for dependency resolution: `pyproject.toml`.

Two commands are executed:

```
poetry debug resolve --tree
poetry debug resolve
```

The first output contains the tree with constraints, the second — the specific versions. The agent matches the data and forms the final graph.

#uv

Main manifest used for dependency resolution: `pyproject.toml`.

The command executed:

```
uv lock
```

The agent parses the `uv.lock` file, which contains information about dependencies and their versions.

#pdm

Main manifest used for dependency resolution: `pyproject.toml`.

The command executed:

```
pdm lock
```

The agent parses the `pdm.lock` file, which contains information about dependencies and their versions. If the alternative format is used, the agent will parse the `pylock.toml` file.

#sbt (Scala)

Main manifest used for dependency resolution: `build.sbt`.

For Scala the command used is:

```
sbt dependencyTree
```

The console output is analyzed, containing the project's dependency structure.

#Swift

Main manifest used for dependency resolution: `Package.swift`.

For the Swift ecosystem the agent runs:

```
swift build --package-path <path_to_Package.swift>
```

Then the `Package.resolved` file is parsed, which contains the locked package versions.

#Composer (PHP)

Main manifest used for dependency resolution: `composer.json`.

The command executed:

```
composer install
```

The agent analyzes `composer.lock`, which records the dependency state.

#Conda

Main manifests used for dependency resolution: `environment.yml`, `environment.yaml`, `meta.yml`, `meta.yaml`.

For projects using Conda, the agent runs:

```
conda-lock -f <environment.yml> --filename <tmpdir/conda-lock.yml>
```

After that, the `conda-lock.yml` file is analyzed, which contains all project dependencies.

#CodeScoring IDE integrations

#Overview

CodeScoring provides Software Composition Analysis (SCA) directly within your development environment through specialized IDE plugins. These integrations enable developers to identify and fix vulnerable dependencies in real-time, shifting security left in the development lifecycle.

By integrating vulnerability detection into the IDE, developers can:

- Detect security issues as they write code, not after deployment
- Get instant feedback on dependency vulnerabilities
- Learn about safe versions and use them right in their development environment
- Maintain secure code from the earliest stages of development
- Track components over time
- Explore list of alerts for triggered policies

#Available integrations

CodeScoring offers plugins for the most popular development environments:

- Visual Studio Code Extension
- Plugin for IntelliJ based IDEs

Both integrations provide vulnerable component scanning, visual indicators in code, and one-click remediation options, ensuring developers can maintain secure dependencies without disrupting their workflow.

#CodeScoring.SCA Extension for Visual Studio Code

The extension provides Software Composition Analysis (SCA) capabilities for VS Code, highlighting vulnerable dependencies in your project files and providing detailed information about vulnerabilities through integration with Johnny CLI.

The **CodeScoring.SCA** extension supports Visual Studio Code versions **1.95.0** and higher.

#Supported Ecosystems

#Languages and Package Managers

Ecosystem	Manifest Files	Lock Files
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

Ecosystem	Manifest Files	Lock Files
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

#Default File Detection

- **Automatic**: Scans all supported files
- **Recursive**: Searches in subdirectories

Fine-tuning the scan is done by modifying the config.yaml file

#Getting Started

#Prerequisites

Before you begin, ensure you have:

- Visual Studio Code installed on your system
- Access to a CodeScoring platform with active credentials
- The codescoring-sca extension distribution for vscode (.vsix)

#Required Permissions

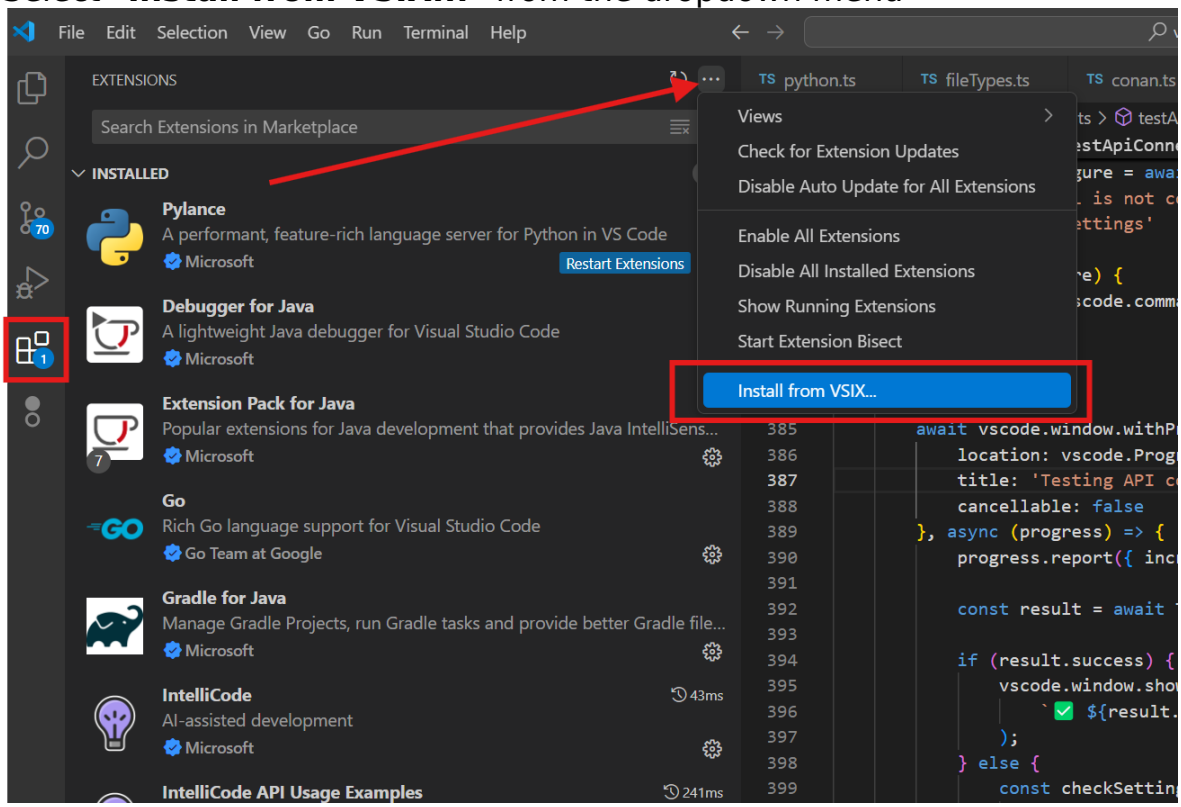
- **File system**: Read project files, write .codescoring files, download executable, execute downloaded CLI
- **Network**: Communication with CodeScoring API
- **VS Code API**: Editor integration

#Step 1: Download the Extension

The extension is provided as a platform-independent file `codescoring-sca-<version>.vsix`.

#Step 2: Install the Extension from VSIX File

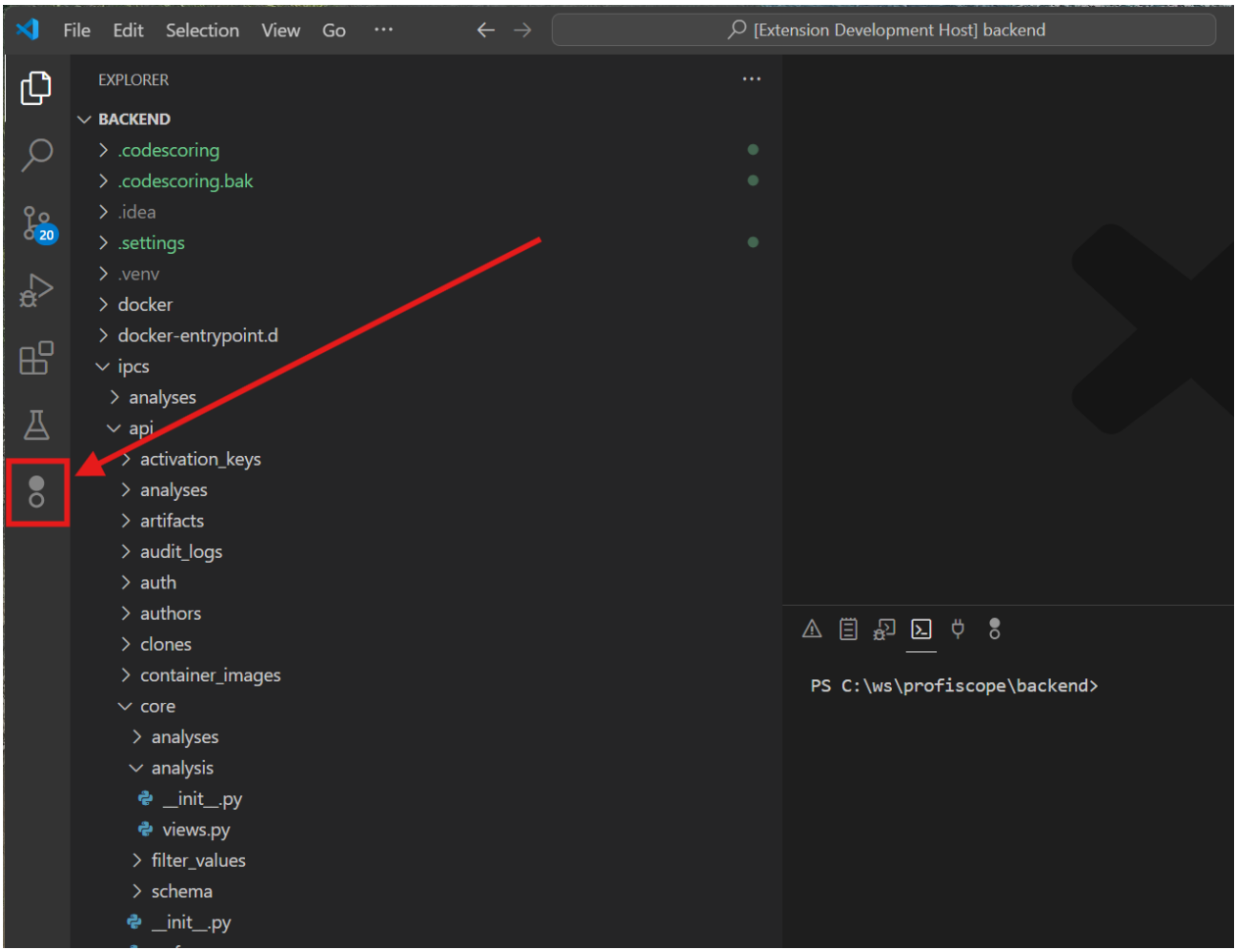
1. Open Visual Studio Code
2. Go to **File → Preferences → Extensions** (or press `Ctrl+Shift+X` or `Cmd+Shift+X` on Mac)
3. Click on the **three dots (...)** menu in the top-right corner of the Extensions panel
4. Select **"Install from VSIX..."** from the dropdown menu



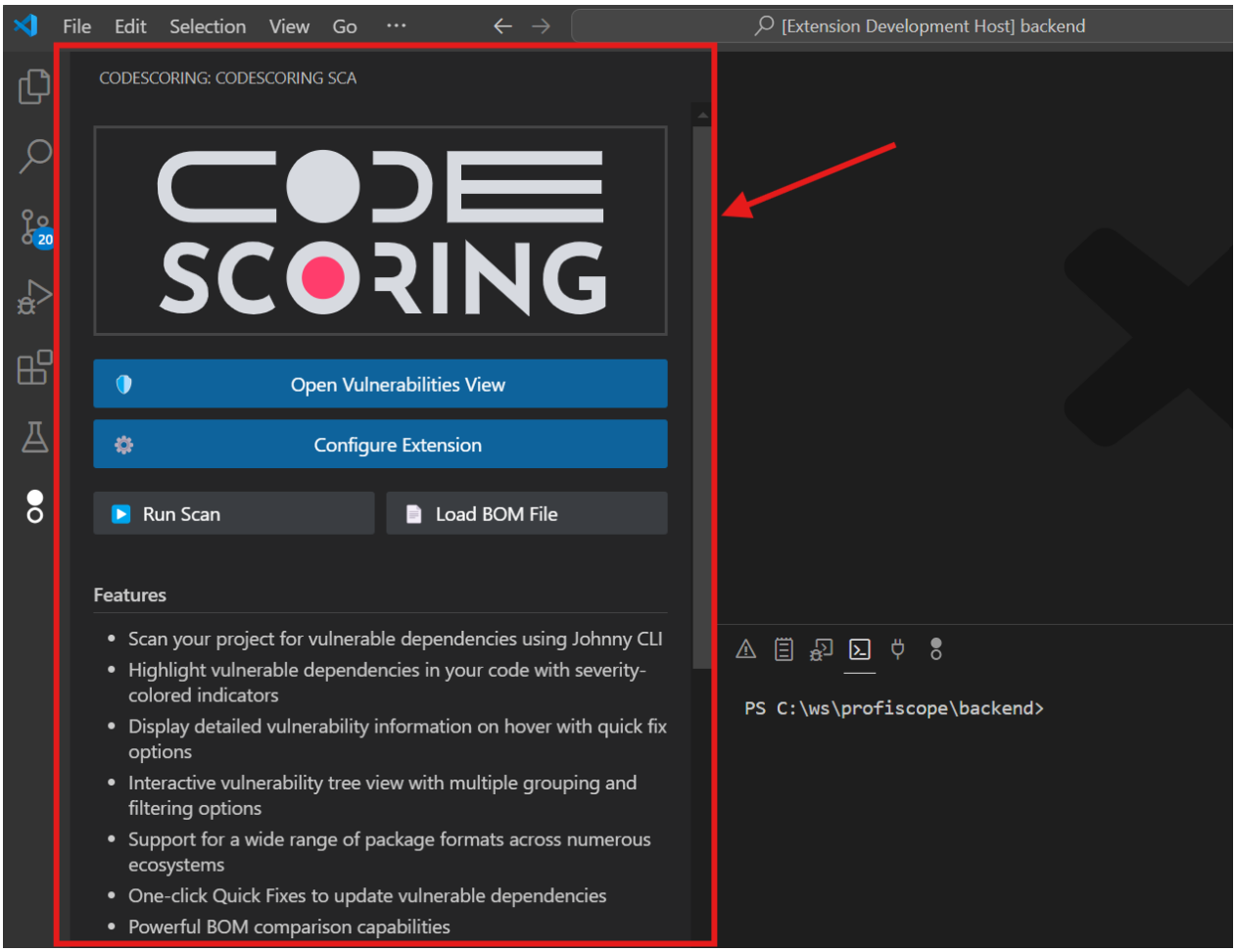
5. Navigate to where you saved the `.vsix` file
6. Select the file and click **"Install"**
7. Wait for the installation to complete

#Step 3: Locate the CodeScoring Extension

After installation, you should see the CodeScoring logo in the VS Code Activity Bar (left sidebar).

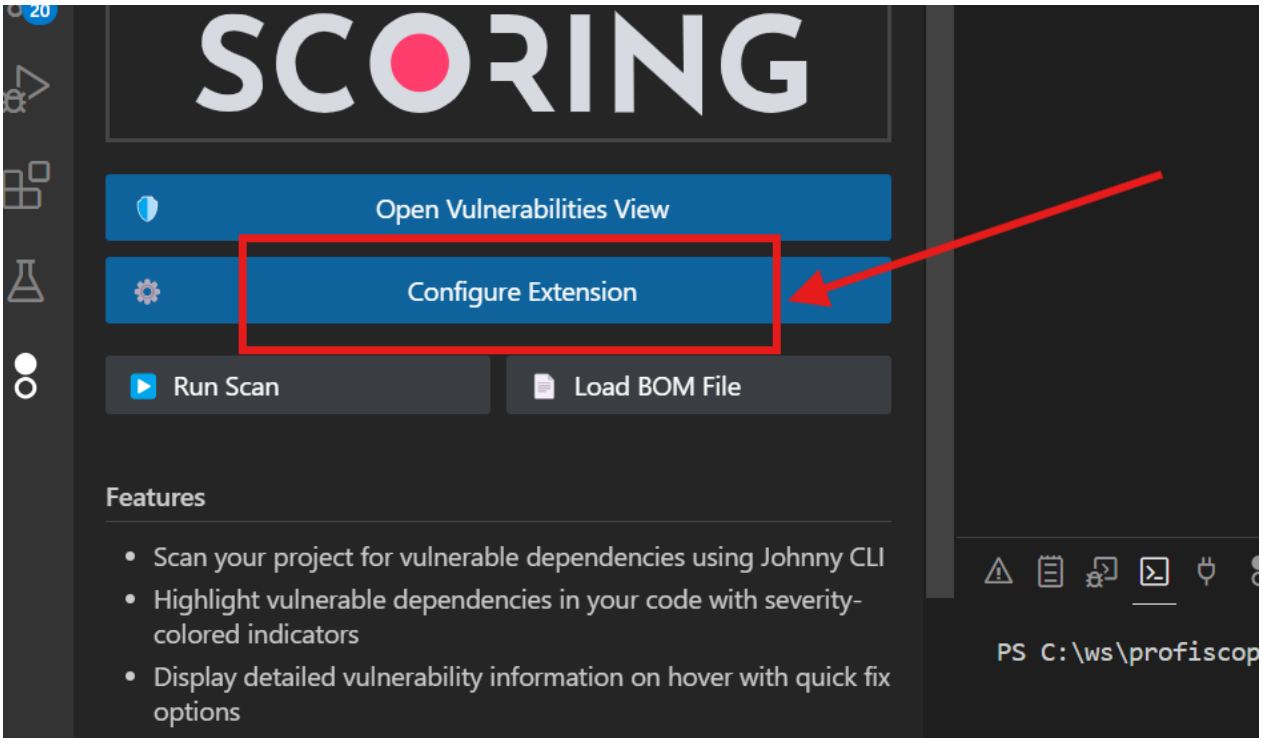


1. Click on the **CodeScoring icon** in the Activity Bar
2. The **"CODESCORING: CODESCORING SCA"** sidebar panel will open



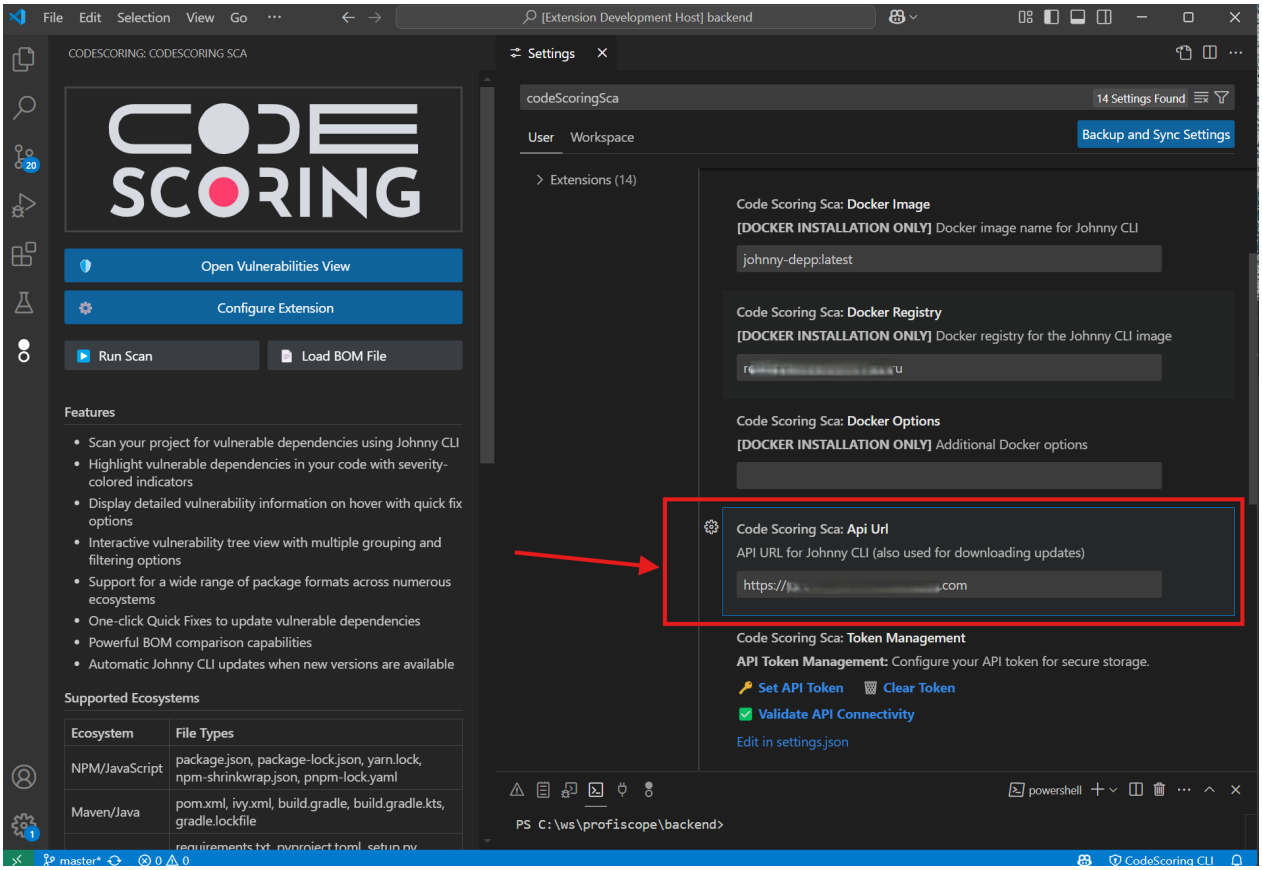
#Step 4: Configure the Extension

1. In the CodeScoring SCA sidebar panel, click the **"Configure Extension"** button
2. The CodeScoring SCA settings page will open



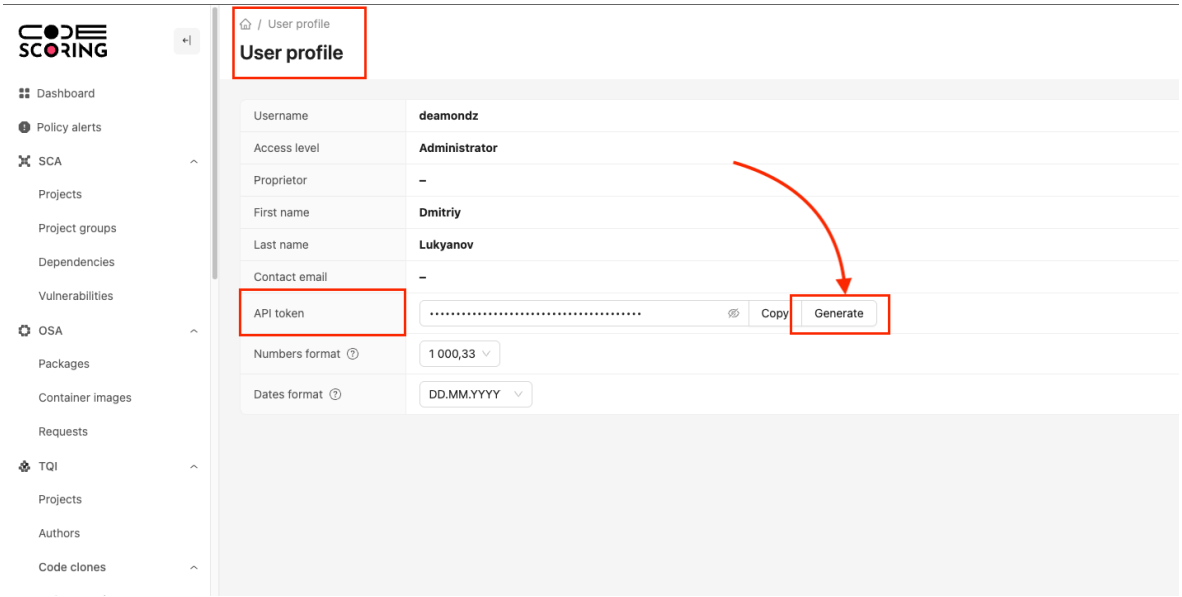
#4.1 Verify the API URL

1. In the settings, locate the **API URL** field
2. You need to use the URL for CodeScoring installed in your organization. Please, contact your administrator.

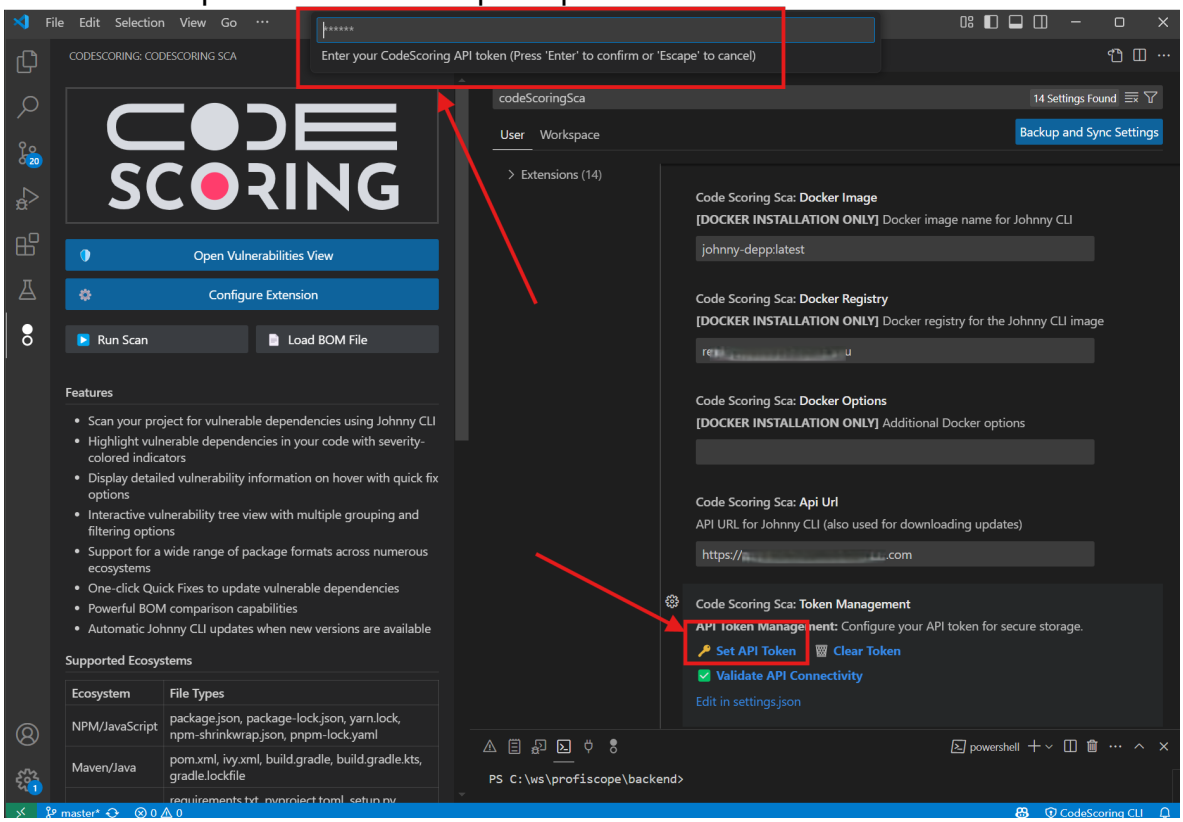


#4.2 Generate and Set the API Token

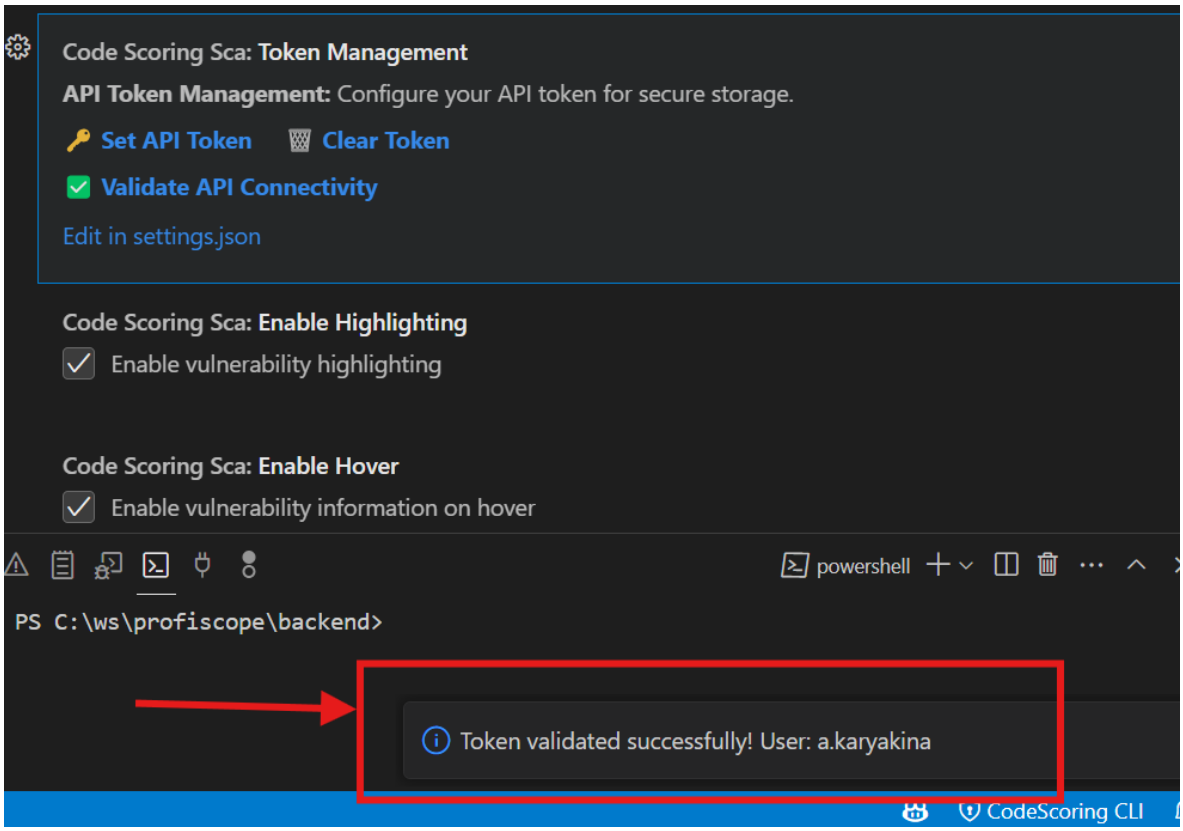
1. Open a web browser and navigate to: `<API URL>/cabinet/profile`
2. Log into your CodeScoring account
3. Ensure you're on your profile page
4. Find the **"API token"** field
5. Click the **"Generate"** button next to the API token field



6. Copy the generated API token value
7. Return to VS Code settings
8. Click on the **"Set API Token"** link in the plugin settings
9. Paste the copied token when prompted

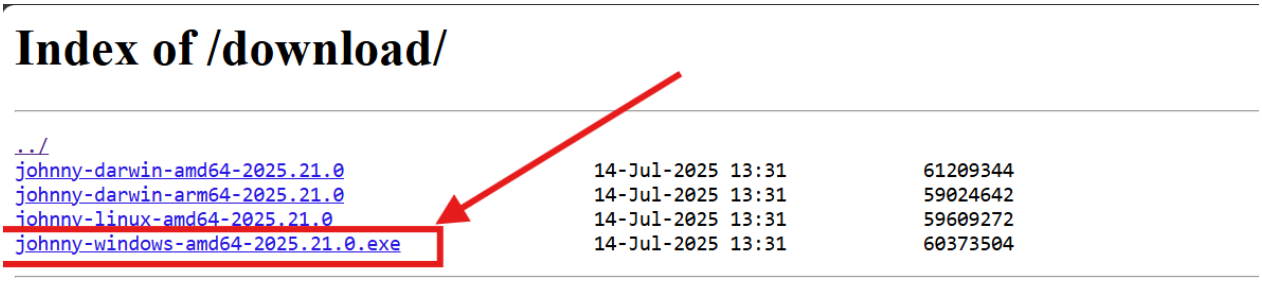


10. The plugin should display confirmation that the token is valid



#4.3 Download Johnny CLI (Optional)

1. Navigate to the downloads page: `<API URL>/download/` (note the trailing slash)
2. Download the latest release executable for your operating system



#4.4 Configure Johnny CLI

There are three ways to obtain Johnny CLI for analyzing your dependencies with our service.

4.4.1 Local platform

Prerequisites:

- Johnny CLI must be downloaded and the file made executable on the system
- The operating system must allow running the executable file (to verify, run the file once in the console with the --help parameter manually)

Setup Steps:

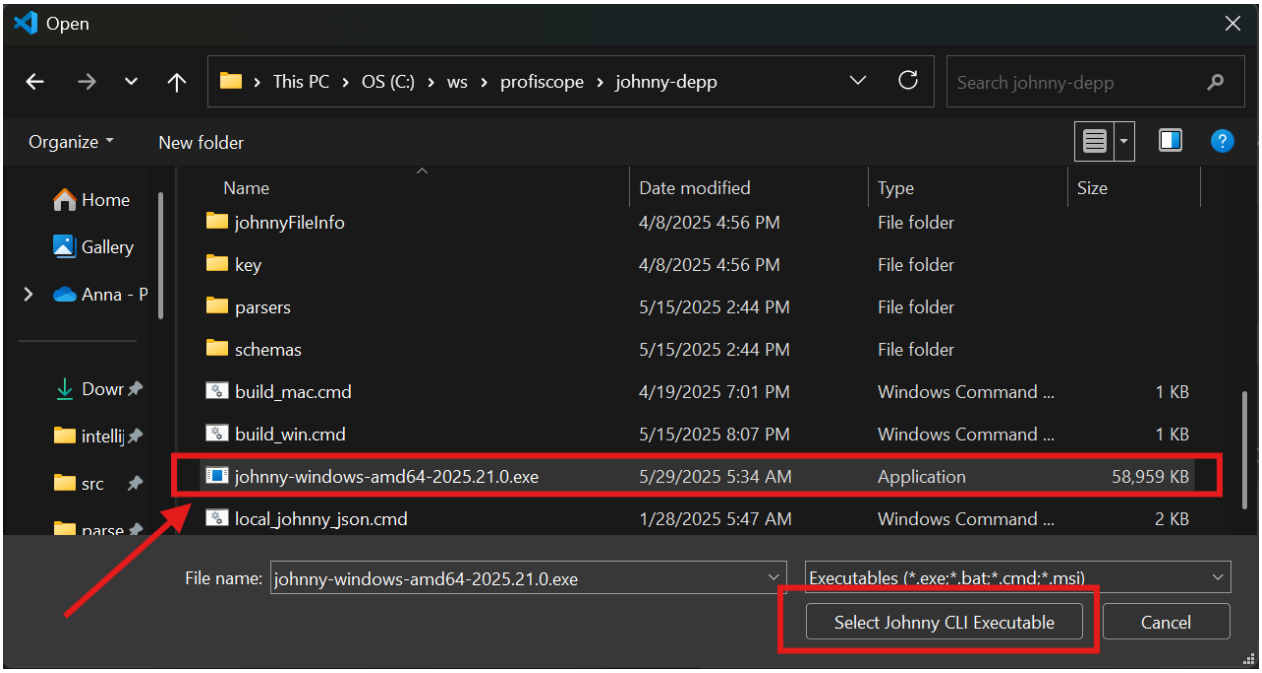
1. Set the platform type to Local

- In VS Code settings, change `installationType` to `local`
- Or in settings.json:

```
"codescoringSca.installationType": "local"
```

1. In the settings, find the `Codescoring Sca: Johnny Cli Path` field marked **[LOCAL INSTALLATION ONLY]**
2. Click on the **"Browse..."** action link
3. Navigate to where you previously downloaded Johnny CLI
4. Select the Johnny CLI executable file

Note: If the plugin asks about changing file permissions (chmod +x), click **"Yes"** to allow the plugin to make the file executable.



4.4.2 Automatic Client Download

Prerequisites:

- API URL must be configured
- API token must be configured and validation must pass successfully

Configuration Steps:

1. Set the platform type to Local

- In VS Code settings, change `installationType` to `local`
- Or in settings.json:

```
"codescoringSca.installationType": "local"
```

1. In the settings, find the `Codescoring Sca: Johnny Cli Path` field marked **[LOCAL platform ONLY]**
2. Clear the value in this field if necessary (empty value is the default)
3. Now on the first scan request, Johnny CLI will be downloaded from the API URL. This will allow you to automatically receive client updates as soon as they become available. The downloaded client will be saved in the following location:
    1. **Linux**:
       `~/.config/Code/User/globalStorage/CodeScoring.codescoring-sca/johnny`
    2. **Windows**:
       `%APPDATA%\Code\User\globalStorage\CodeScoring.codescoring-sca\johnny.exe`
    3. **MacOS**:
       `~/Library/Application Support/Code/User/globalStorage/CodeScoring.codescoring-sca/johnny`

4.4.3 Using Docker

Docker installation allows running Johnny CLI in an isolated container, which is useful when you don't want to install it directly on your system.

Prerequisites:

- Docker must be installed and running on your system
- Your user must have permissions to execute Docker commands

Configuration Steps:

1. Set the platform type to Docker

- In VS Code settings, change `installationType` to `docker`
- Or in settings.json:

```
"codescoringSca.installationType": "docker"
```

1. Configure the Docker image

- **Image**: `johnny-depp:2025.29.0` (default)
- **Registry**: `<codescoring-registry-host>`

- Sample full image path: `sample-codescoring-registry.com/johnny-depp:2025.29.0`

1. **Optional: Additional Docker Options** Add custom Docker run options if needed:

   ```
   "codescoringSca.dockerOptions": "--memory=2g --cpus=2"
   ```

How it works:

- The extension automatically mounts your project directory into the container
- Scanning is performed inside the container, with results saved to your project
- No manual Docker commands are required - the extension handles everything automatically

Example Docker Configuration:

```
{
  "codescoringSca.installationType": "docker",
  "codescoringSca.dockerImage": "johnny-depp:2025.29.0",
  "codescoringSca.dockerRegistry": "sample-codescoring-
registry.com",
  "codescoringSca.dockerOptions": ""
}
```

Troubleshooting Docker installation:

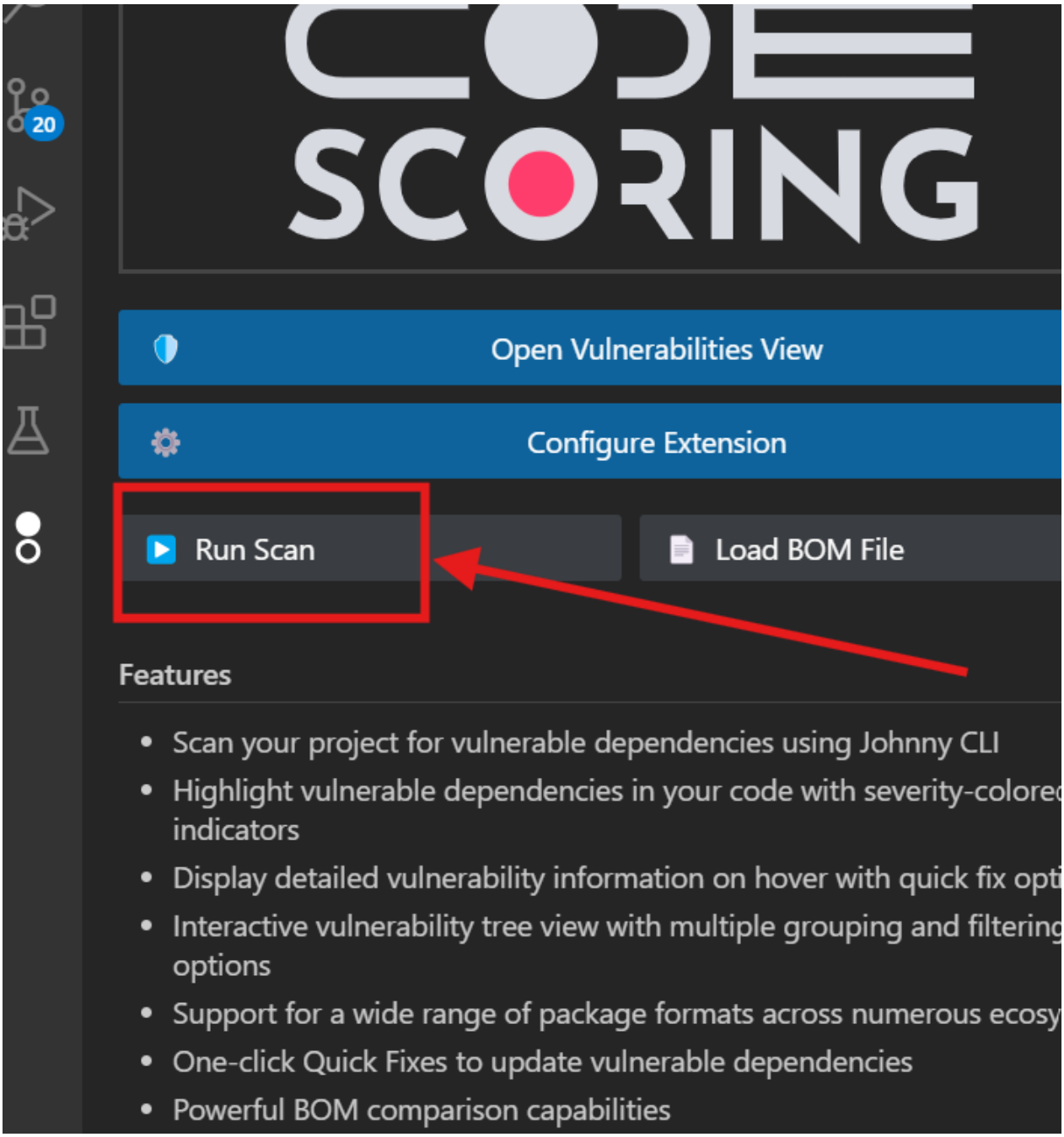
- **"Docker not found"**: Ensure Docker is installed and the `docker` command is in your PATH
- **Permission denied**: Add your user to the docker group: `sudo usermod -aG docker $USER`
- **Image pull failed**: Check registry credentials and network connection
- **Container exits immediately**: Check VS Code Output panel for detailed error messages

#Step 5: Run Your First Scan

Now that the extension is configured, you can run your first scan for vulnerable dependencies:

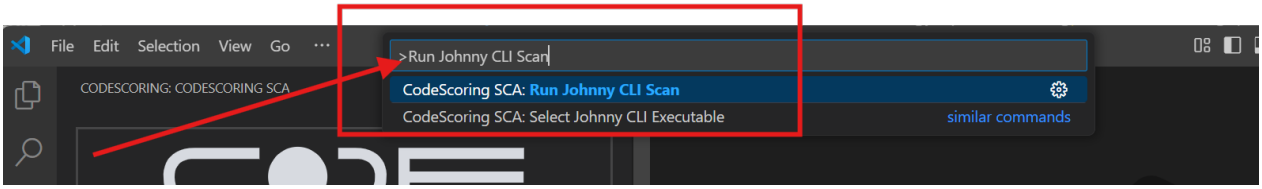
#Method 1: Using the Sidebar Panel

1. Open a project folder in VS Code
2. Click on the CodeScoring icon in the Activity Bar
3. In the "CODESCORING: CODESCORING SCA" sidebar panel, click the **"Run Scan"** button



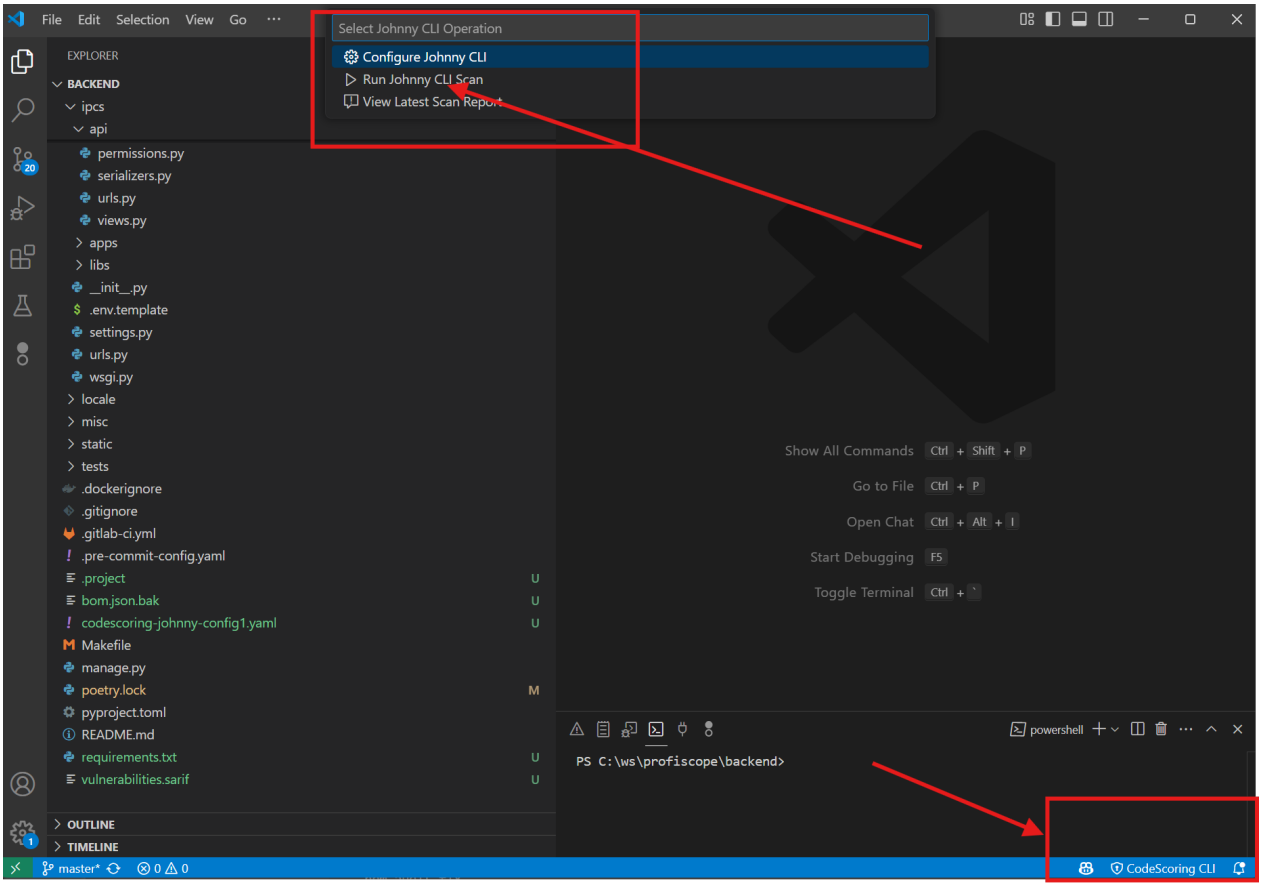
#Method 2: Using the Command Palette

1. Press `Ctrl+Shift+P` (or `Cmd+Shift+P` on Mac) to open the Command Palette
2. Type "Run Johnny CLI Scan"
3. Select **"CodeScoring SCA: Run Johnny CLI Scan"** from the list



#Method 3: Using the Status Bar

1. Look at the bottom status bar of VS Code
2. Find the **"CodeScoring CLI"** indicator
3. Click on it to open the Johnny CLI menu
4. Select **"Run Johnny CLI Scan"** from the list



#Step 6: Fine-tuning the Scan Configuration

After the scan completes, you'll see a new `.codescoring` directory containing:

1. config.yaml - configuration file for Johnny CLI, you can read about it in this section. You can modify this file as it will never be overwritten.
2. report.html - report generated in html format containing Johnny CLI output in colored table format. Overwritten during each scan.
3. bom.json - scan results file created by Johnny CLI in cyclone-dx 1.6 format, it will be loaded automatically and shown in the vulnerabilities panel for any directory opened in VSCode that has .codescoring/bom.json
4. bom.json.N - where N is the scan revision, i.e., 0 is the previous scan, and 5 (for example) is the very first scan, and bom.json.0 will be used for comparison with bom.json (and shown in the DIFF tree) if it exists when opening a new folder

›
your-project

›
.codescoring

config.yaml# Scan configuration

report.html# Latest scan report

bom.json# Current vulnerabilities

bom.json.0# Previous scan (comparison)

bom.json.1# Older scans...

package.json# Your dependencies

...your code ...

#Example Usage of CodeScoring Scan Configuration

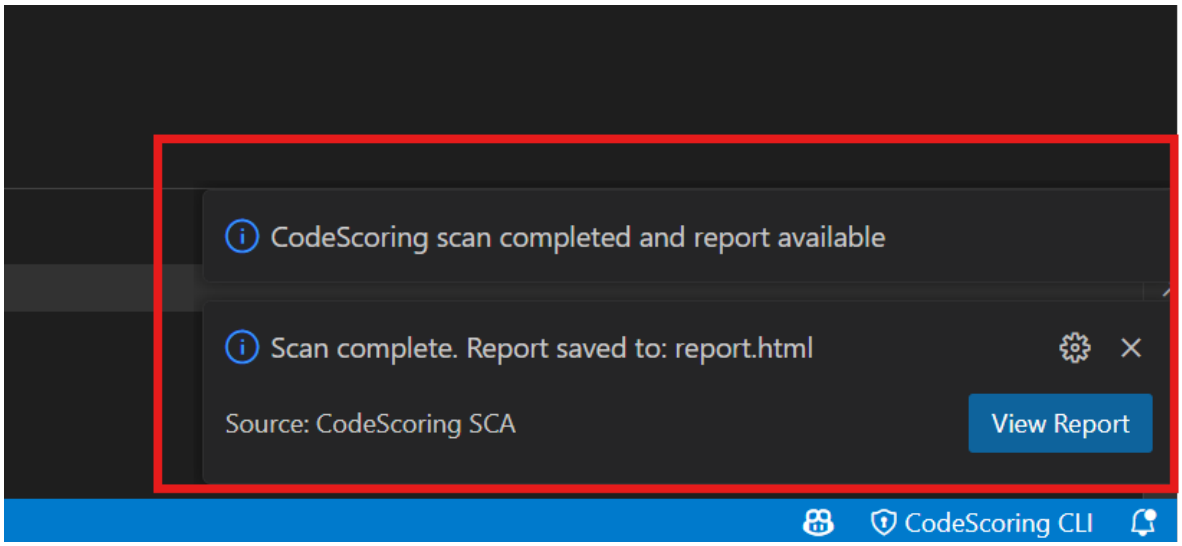
Edit `.codescoring/config.yaml` to customize:

```
scan:
  general:
    ignore:           # Ignore paths
      - node_modules
      - .git
      - test
    with-hashes: true # Include file hashes for exact matching
    only-hashes: false # Use only hash-based detection
  dir:
    no-recursion: false # Prevents from recursively scan
directories
```

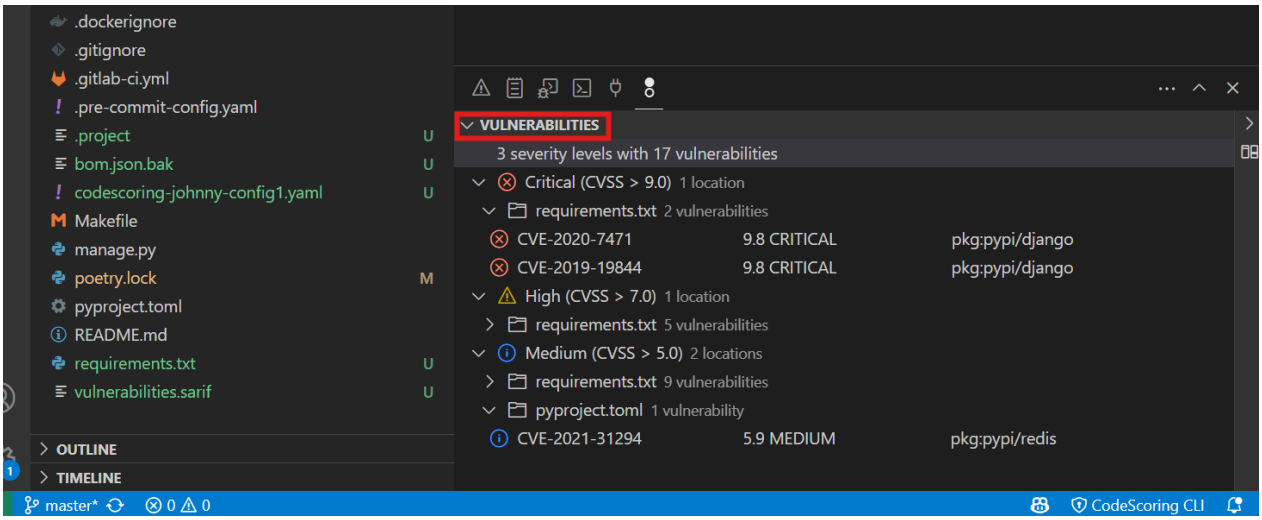
#Step 7: View Scan Results

After the scan completes:

1. **Check notifications**: Look at the notifications in the bottom-right corner of VS Code, click **"View Report"**



2. **Open the vulnerabilities tree** (opens automatically by default): Use the **"Open Vulnerabilities View"** button from the sidebar panel
3. **Review vulnerabilities**: The vulnerabilities panel will show all detected security issues in your dependencies
4. **Explore details**: You can click on individual vulnerabilities to see detailed information
5. **Apply fixes**: Use quick fix options to update vulnerable dependencies
6. **Open Alerts View**: Use **Open Alerts View** button from the sidebar to view alerts list of triggered policies



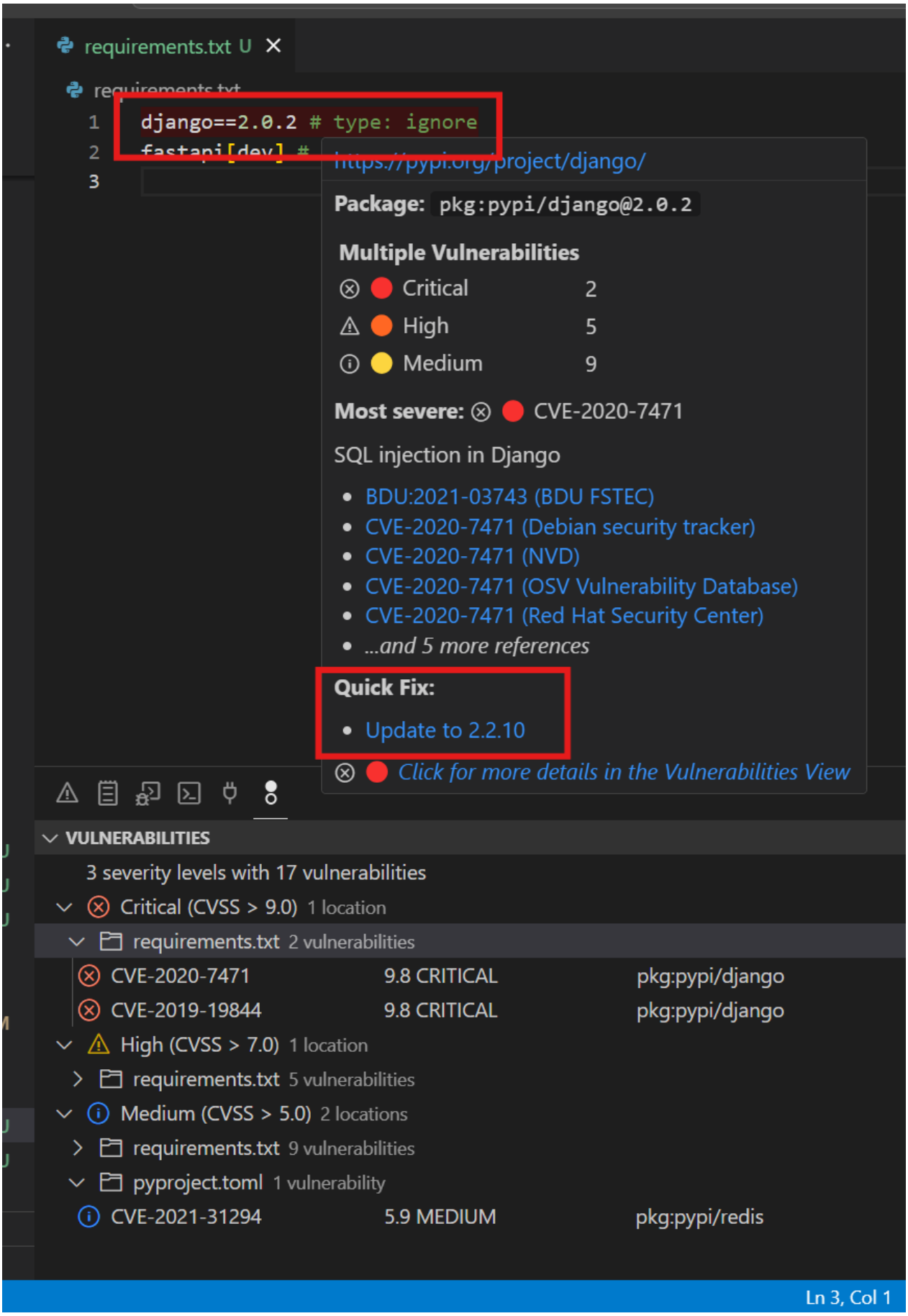
#7.1 Vulnerability Highlighting

- **In-code highlighting**: Vulnerable dependencies are highlighted as you type
- **Severity colors**:
  - Critical (red)
  - High (orange)
  - Medium (yellow)
  - Low (blue)
- **Multi-file support**: Works with all supported file types
- **Hover** over highlighted dependencies to see vulnerability details

#7.2 Hover Information

When hovering over highlighted dependencies, it displays:

- **Vulnerability ID**: CVE number with link
- **Severity**: CVSSv3 score and level
- **Description**: What the vulnerability does
- **Source links**: Official vulnerability registration information
- **Quick fix**: One-click update option


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 Vulnerabilities Panel

7.3.1 Tree View Structure

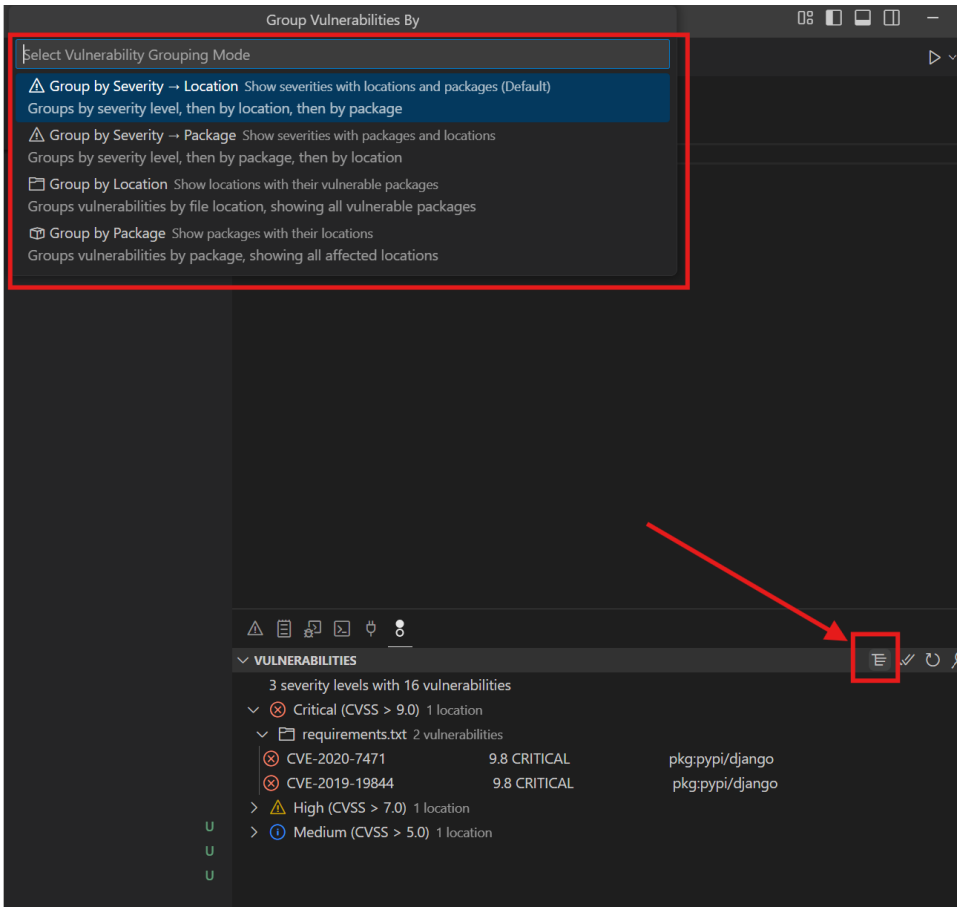
```
 Vulnerabilities (247)
├── 🔴 Critical (12)
│   ├── CVE-2023-1234 - Remote Code Execution
│   │   ├── lodash@4.17.20 → 4.17.21
│   │   └── package.json:15
│   └── ...
├── 🟠 High (45)
├── 🟡 Medium (89)
└── 🔵 Low (101)
```

7.3.2 Grouping Options

- Use the vulnerabilities panel to filter by severity, package, or other criteria
- Group vulnerabilities by different categories for better organization

Change grouping via toolbar button or command:

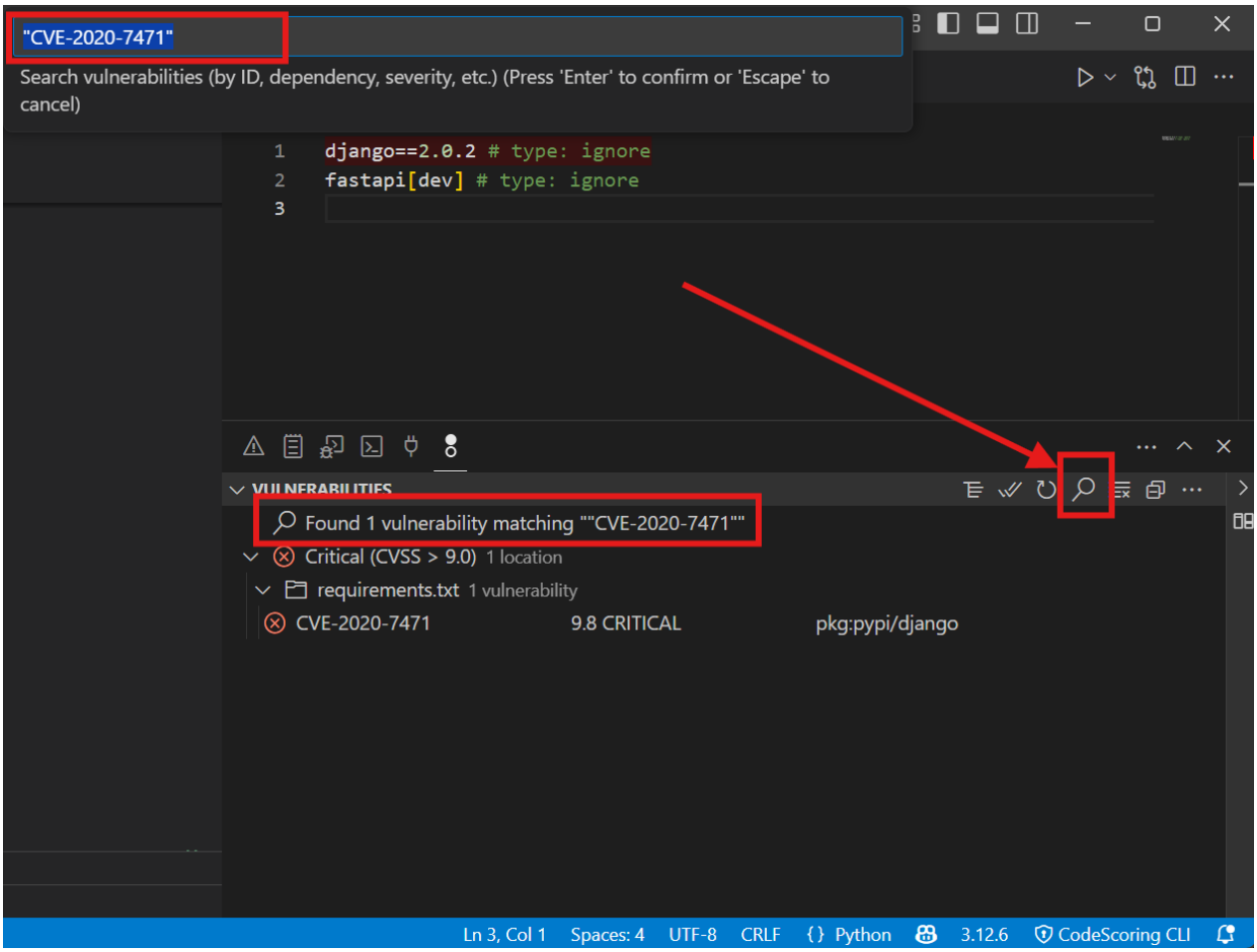
- **By Severity -> Location** (default): By severity level, then by package
- **By Severity -> Package (PURL)**: By severity level, then by package name alphabetically
- **By Location**: Group by file path
- **By Package (PURL)**: Group by package



#7.4 Search and Filtering

Search Capabilities

- **Multiple fields**: Search by:
  - Package name (e.g., "lodash")
  - CVE ID (e.g., "CVE-2023")
  - File path (e.g., "frontend/")
  - Severity level
- **Fuzzy matching**: Finds partial matches unless quotes are used
- **Exact match**: Use quotes
- **Case insensitive**: No exact case required



#7.5 Quick Fixes

Quick Fixes

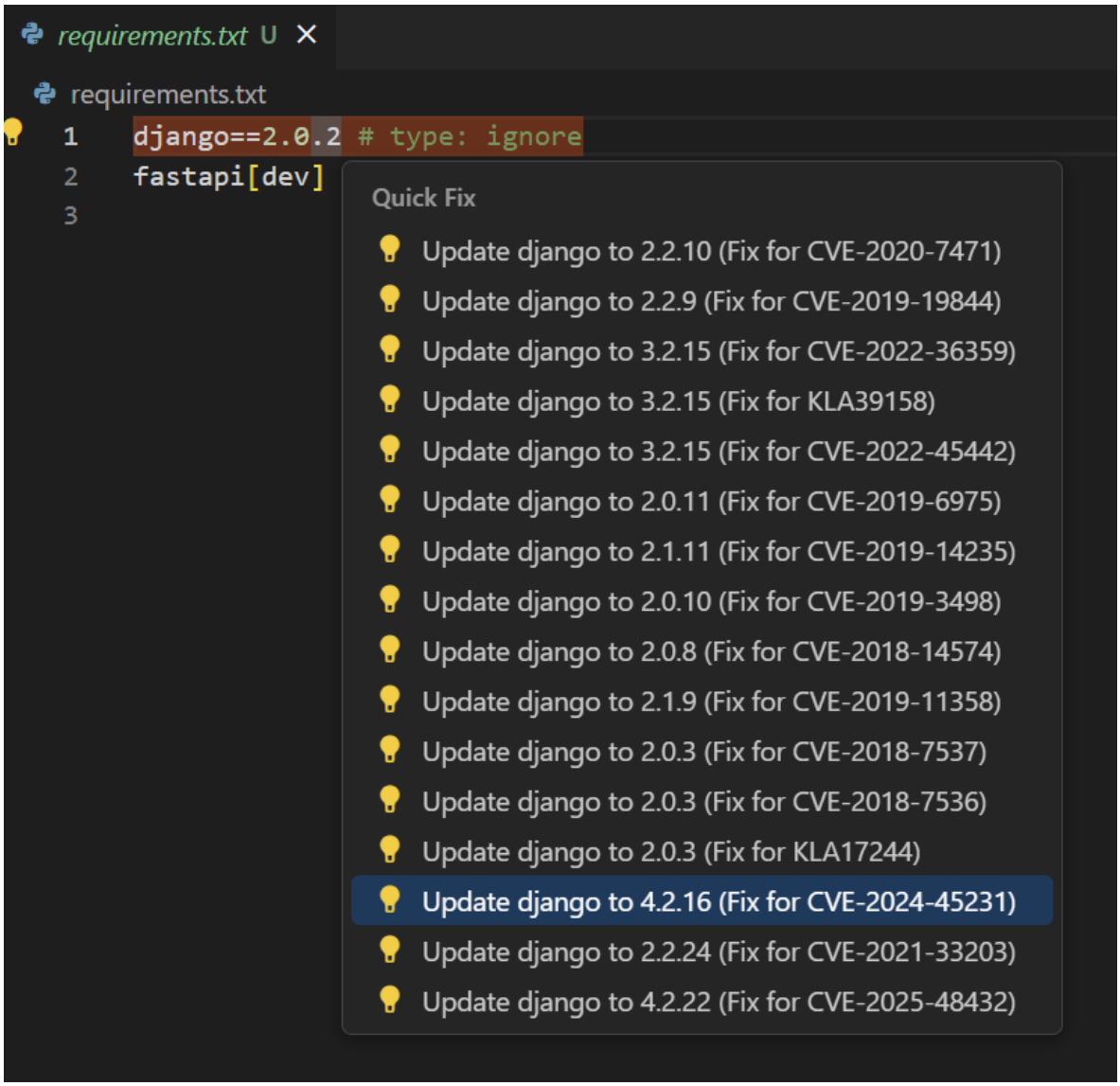
- Click on suggested version updates when hovering over a vulnerable component to automatically update to the latest dependency version
- When pressing `Ctrl+.` (`Cmd+.` on Mac) while the cursor is on a vulnerable component, you'll be prompted to select a specific version to update to (if multiple are available)

Individual Fixes

- Hover your mouse over a vulnerable dependency
- Click on the suggested version in the tooltip
- After updating, you'll see a success message

or

- Hover your keyboard cursor over a vulnerable dependency
- Press `Ctrl+.` (`Cmd+.` on Mac)
- Select the appropriate version from the list if there are multiple



or

- **Fix Selected Button**: Select one vulnerable component or a vulnerability belonging to that component, and the component will be updated to the latest safe version

Bulk Fixes

- **Fix All Button**: Updates all vulnerable components with available fixes that are currently visible in the tree, taking into account applied filters (magnifying glass)

#7.6 Working with BOM Files

Auto-loading

The extension automatically loads BOM files from:

1. `.codescoring/bom.json` (primary)
2. `bom.json` (root directory)

Manual Operations

- **Load BOM**: `Ctrl+Shift+P` (or `Cmd+Shift+P` on Mac) → "Load BOM File"
- **Close BOM**: Clears all vulnerability data and frees memory

#7.7 BOM Comparison

Automatic Comparison

When opening a project:

- Loads current BOM (`bom.json`)
- Compares with previous (`bom.json.0`)
- Shows notification of changes

Manual Comparison with Open BOM

1. Have a loaded BOM (target)
2. Execute "Compare BOMs" command
3. Select base BOM file
4. Comparison result will be displayed in DIFF panel

Manual Comparison of Two Arbitrary BOMs

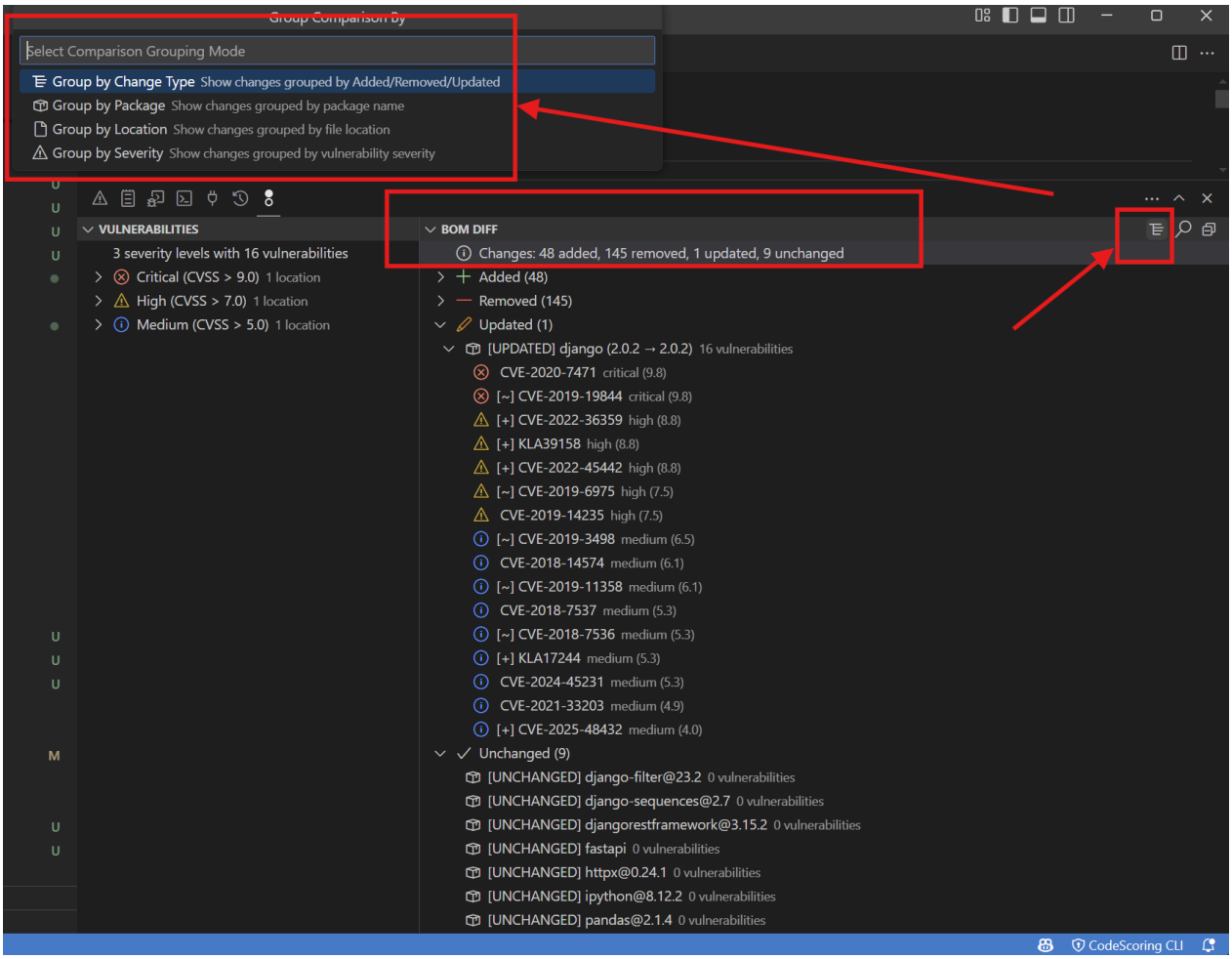
1. Close BOM if necessary
2. Execute "Compare BOMs" command
3. Select base BOM file for comparison
4. Select target BOM file to compare with base
5. Comparison result will be displayed in DIFF panel

Comparison Views

```
 BOM DIFF (Changes: 23 added, 15 removed, 45 updated, 73
unchanged)
├── + Added (23)
│   ├── [ADDED] react@18.1.3 0 vulnerabilities
│   └── ...
├── − Removed (15)
├──  Updated (45)
│   ├── [UPDATED] lodash: 4.17.20
│   └── ...
└── ✓ Unchanged (73)
```

Comparison Grouping Options

- **By Change Type**: Added/Removed/Updated/Unchanged (note: updated means the vulnerability count was updated for a component)
- **By Package**: Alphabetical package grouping without version. This is the most useful grouping for viewing updated components (added version and removed version will be grouped together)
- **By Location**: Group by file path
- **By Severity**: Group by vulnerability impact



Comparison Filtering

You can also search/filter the BOM DIFF tree to focus on components of interest. To clear the search, click the "Clear Comparison Search" button in the top right of BOM DIFF.

#7.8 Reports

Scan Reports

- **Automatically generated**: Created after each scan
- **Location**: `.codescoring/report.html`
- **Format**: Colored HTML
- **Contents**:
  - Scan status
  - Command executed
  - Results summary
  - Found vulnerabilities
  - Policy warnings
  - Error messages

Viewing Reports

- **Command**: "View Latest Scan Report"
- **Opens in**: HTML Preview in VS Code

- Adapts to the selected VSCode color scheme at generation time

#7.9 Alerts panel

The **Alerts** panel displays information about alerts for triggered policies based on the analysis results. For each alert, the following is shown:

- Policy name
- Alert level
- Blocking status
- List of packages with criteria

#Step 8: Settings and Customization

Available Settings List

Setting	Description	Default
`apiUrl`	Your CodeScoring platform URL	
`apiToken`	API token. Securely stored	*(set via command)*
`installationType`	local or docker	`local`
`johnnyCliPath`	Path to Johnny CLI (empty for auto-download)	*(auto-download)*
`dockerImage`	Docker image name with Johnny CLI	`johnny-depp:2025.29.0`
`dockerRegistry`	Registry with Johnny CLI image	*(provided by support)*
`dockerOptions`	Additional docker options	
`enableHighlighting`	Show in-code highlighting	`true`
`enableHover`	Show hover tooltips	`true`
`enableQuickFixes`	Allow one-click fixes	`true`
`showVulnerabilityHeaders`	Display headers for vulnerability columns	`false`
`paginationSize`	Number of items per page	`100`
`batchProcessingSize`		`100`

Setting	Description	Default
	Number of items to process at once	
`severityColors`	Custom color mapping	*(default colors)*

Keyboard Shortcuts

Configure in VS Code settings, for example:

```
{
  "key": "ctrl+shift+s",
  "command": "codescoring-sca.runJohnnyCLI"
}
```

#Troubleshooting

Extension Logs

- For detailed understanding of plugin functionality, check the extension log file located at:
  - Windows: `%USERPROFILE%\.vscode\extensions\codescoring-sca-[version]\out\logs\extension.log`
  - macOS/Linux: `~/.vscode/extensions/codescoring-sca-[version]/out/logs/extension.log`

#Common Issues

Scanning Issues

Issue	Solution
Scan hangs	Try running the cli file manually, ensure the operating system and antivirus allow it to work, check internet connection, verify API token
No results	Ensure project has dependency files, check report.html output
Partial results	Check ignore patterns in configuration

Display Issues

Issue	Solution
No highlighting	Enable in settings, reload window
Wrong colors	Check theme compatibility
Missing panel	View → Open View → Vulnerabilities
Hovers not showing	Enable hover in settings

Fix Issues

Issue	Solution
Fix fails	Check write permissions
Wrong version	Manually specify in package file
Breaks project	Use version control, rollback
Conflicts	Fix one by one

Contact support: support@codescoring.ru

#Security and Privacy, Data Handling

- **Local scanning**: Code is not sent to servers
- **API communication**: Only metadata is transmitted (your package manager configuration files)
- **Token storage**: VS Code credential secure storage

#Best Practices, Workflow Recommendations

1. **Initial setup**: Full scan when starting a project
2. **Reviews**: Compare BOMs between versions
3. **CI/CD**:
    - **Before commit**: Run full scan
    - **Share configuration**: Commit `.codescoring/config.yaml`
    - **Ignore reports**: Add `report.html` to `.gitignore`
    - **Track BOM changes**: Version control `bom.json` for history
    - **Standardize**: Create team-wide VS Code settings

#CodeScoring.SCA Plugin for IntelliJ based IDEs

The plugin provides Software Composition Analysis (SCA) capabilities for IntelliJ IDEA and other IntelliJ Platform based IDEs, highlighting vulnerable dependencies in your project files and providing detailed vulnerability information through Johnny CLI integration.

The **CodeScoring.SCA** plugin supports IntelliJ IDEA versions **2024.1** and above, as well as all IDEs based on IntelliJ Platform (OpenIDE, GIGA IDE, PyCharm, WebStorm, PhpStorm, RubyMine, GoLand, CLion, Rider, Android Studio).

#Supported Ecosystems

#Languages and Package Managers

Ecosystem	Manifest Files	Lock Files	Features
Java/JVM	pom.xml, *.gradle, *.gradle.kts, ivy.xml	gradle.lockfile, gradle-dependency-tree.txt, maven-dependency-tree.txt	Full support for Maven and 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	Gemfile, gems.rb, *.gemspec	Gemfile.lock, gems.locked	Bundler and RubyGems
Go	go.mod	go.sum	Go modules
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 and Paket

Ecosystem	Manifest Files	Lock Files	Features
Swift	Package.swift	Package.resolved	Swift Package Manager
Objective-C	Podfile, *.podspec	Podfile.lock	CocoaPods for iOS/macOS
C/C++	conanfile.txt, conanfile.py	conan.lock	Conan package manager
Conda	environment.yml, meta.yml, environment.yaml, meta.yaml	conda-lock.yml	Conda environments

#Default File Detection

- **Automatic**: Scans all supported files
- **Recursive**: Searches in project subdirectories

Fine-tuning the scan is done by modifying the config.yaml file

#Getting Started

#Prerequisites

Before you begin, ensure you have:

- IntelliJ IDEA 2024.1 or newer (or any compatible supported IntelliJ based IDE)
- Access to the CodeScoring platform with active credentials
- The codescoring-intellij plugin distribution (.zip file)

#Required Permissions

- **File system**: Read project files, write .codescoring files, download executable file, execute downloaded CLI
- **Network**: Communication with CodeScoring API
- **VS Code API**: Editor integration

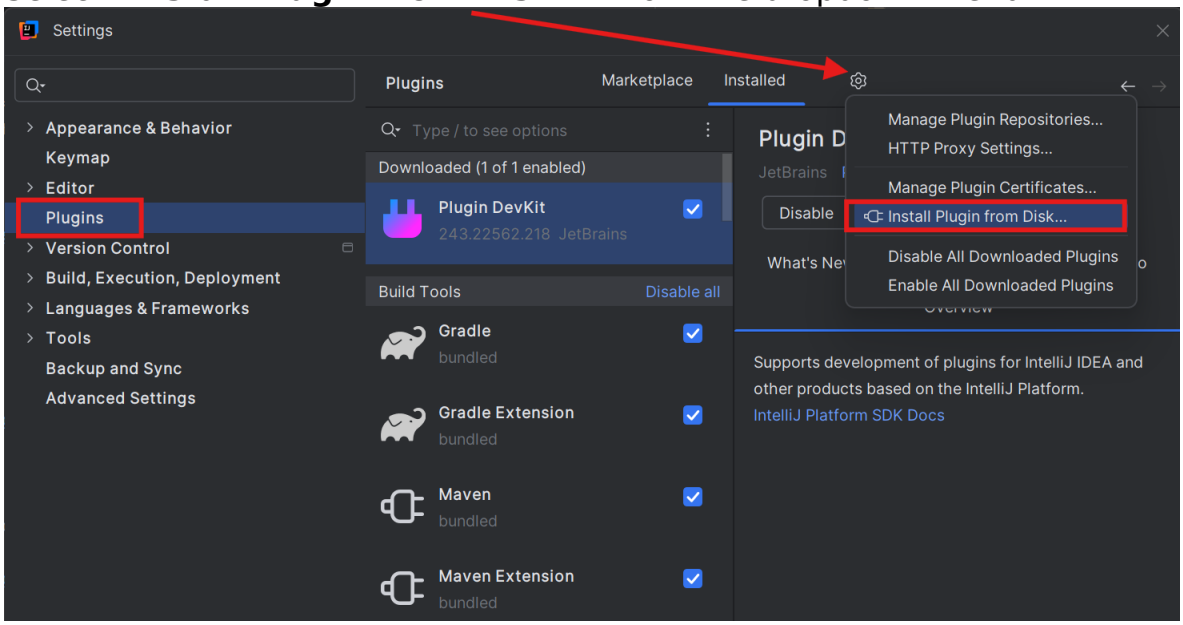
#Step 1: Download the Plugin

The plugin is provided as a `codescoring-intellij-<version>.zip` file.

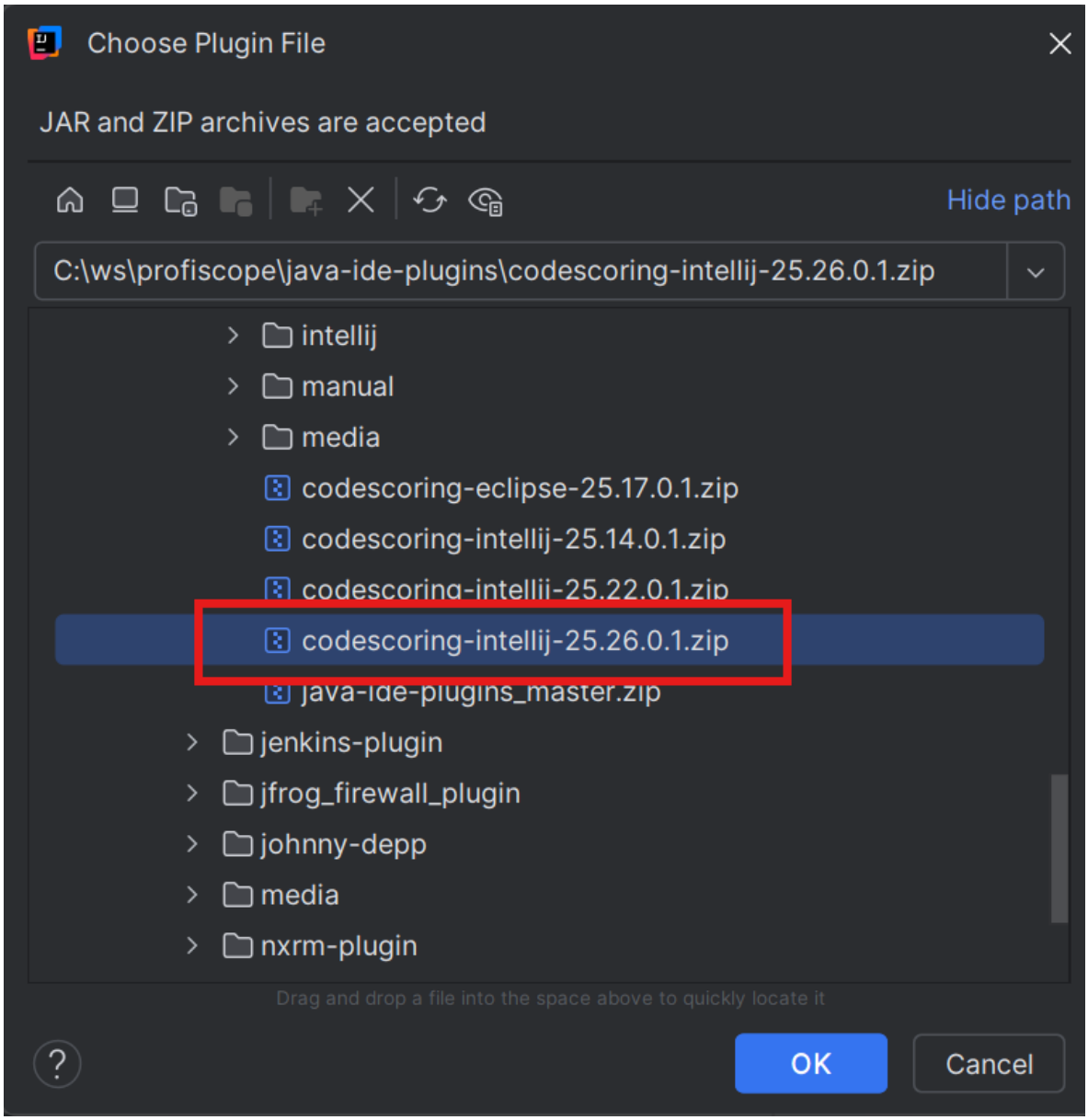
#Step 2: Install the Plugin from ZIP File

1. Open IntelliJ based IDE
2. Go to **File** → **Settings** (or **<IDE Name>** → **Preferences** on macOS)

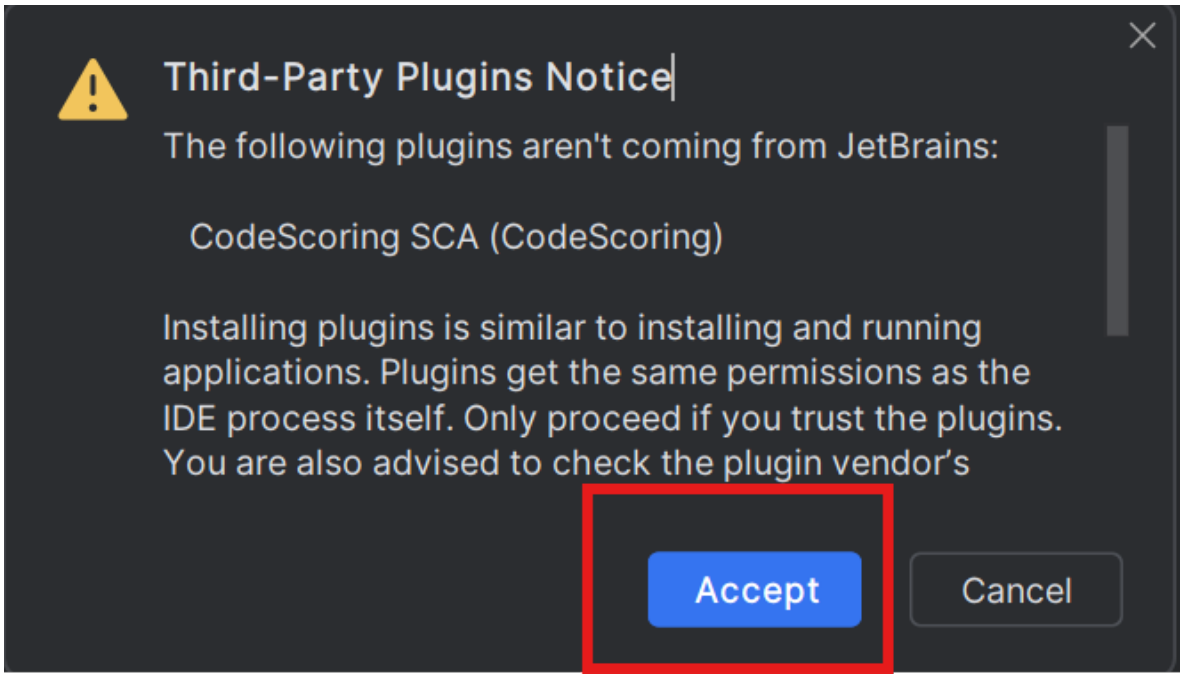
3. In the settings dialog, select **Plugins** in the left sidebar
4. Click the **gear icon (⚙)** at the top of the plugins panel
5. Select "**Install Plugin from Disk...**" from the dropdown menu



6. Navigate to where you downloaded the `.zip` file
7. Select the file and click "**OK**"



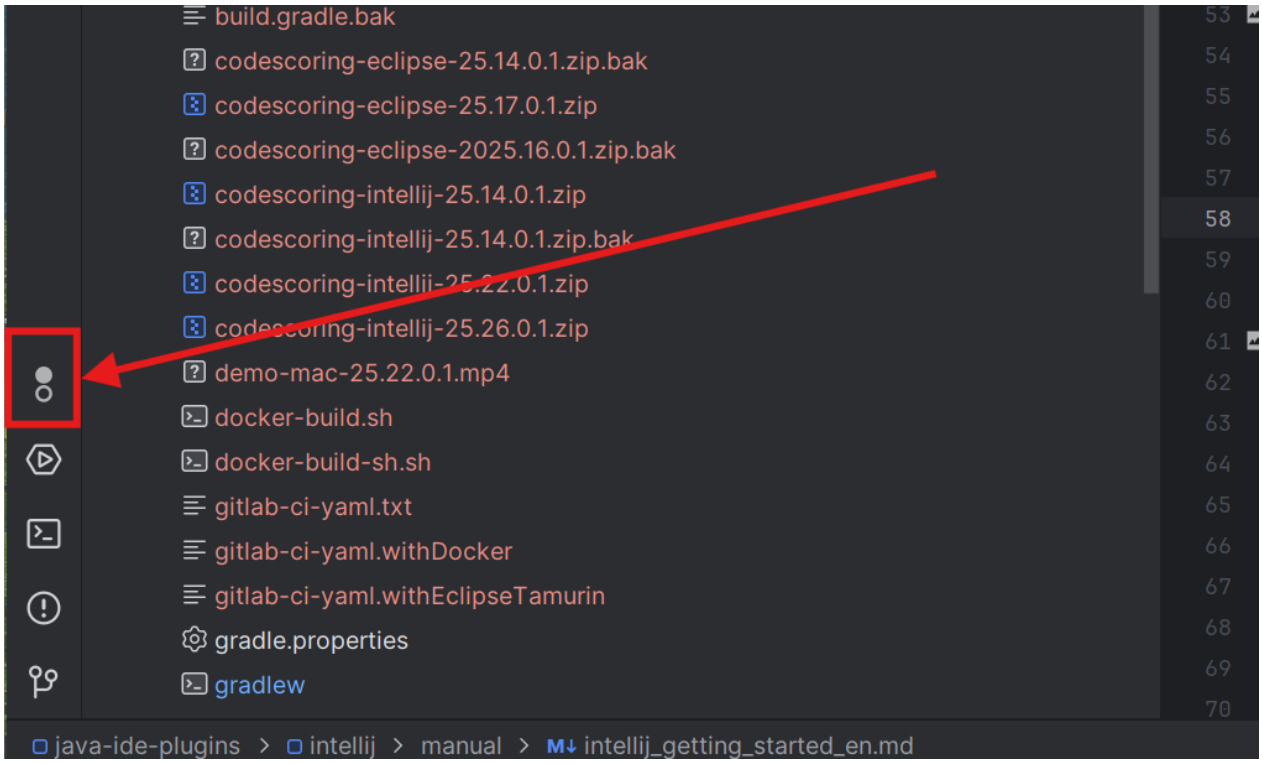
8. Wait for the installation to complete
9. Confirm the installation of the plugin from CodeScoring



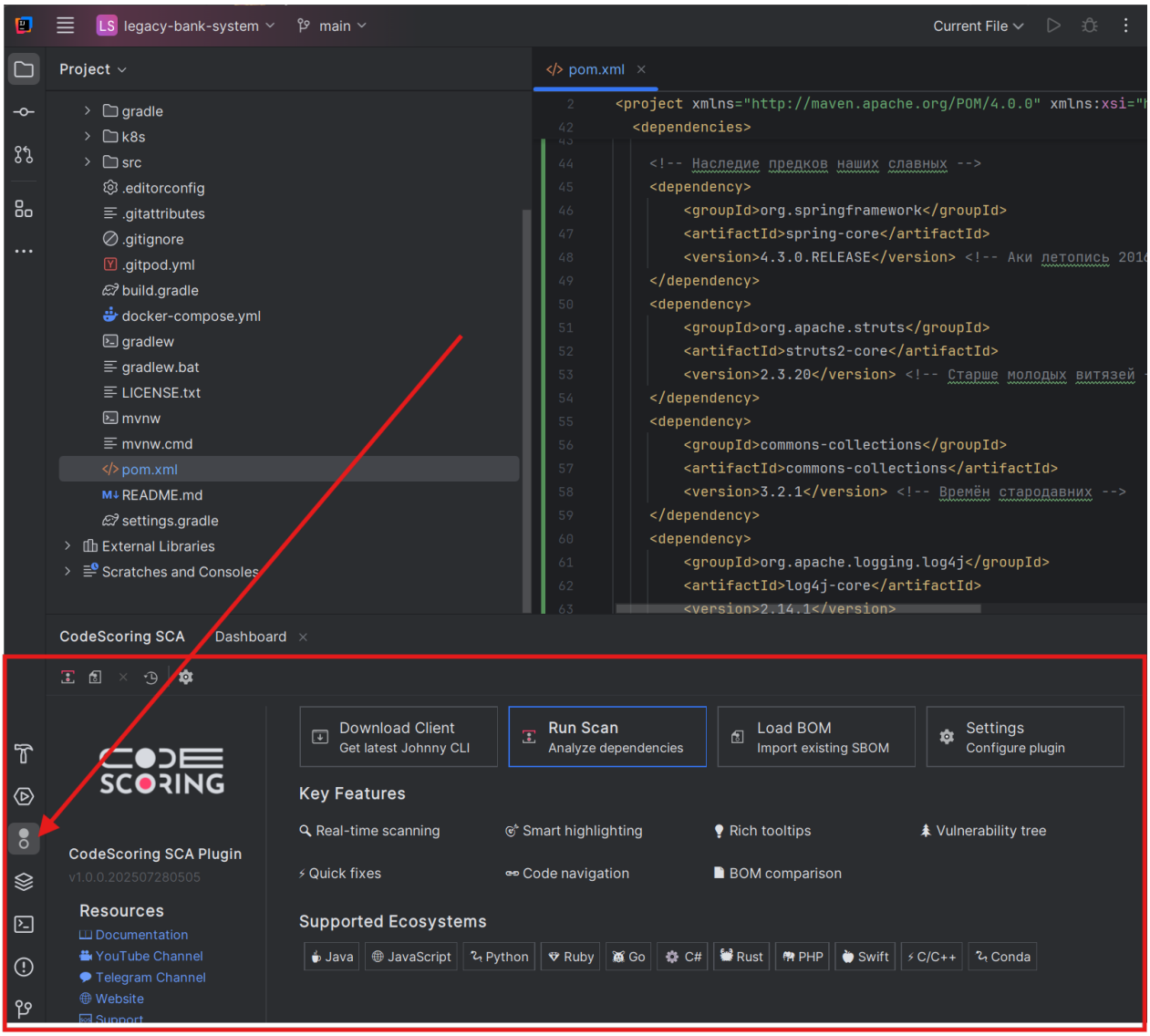
10. Restart IntelliJ based IDE when prompted

#Step 3: Locate the CodeScoring Plugin

After installation and restart, you should see the CodeScoring tool window.

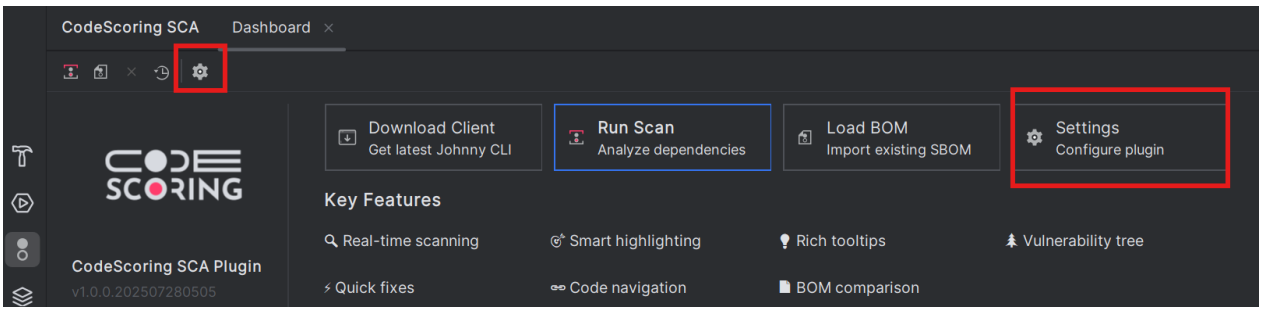


1. Look for the **"CodeScoring SCA"** tool window tab (usually at the bottom or left of the IDE)
2. If not visible, go to **View → Tool Windows → CodeScoring SCA**
3. This will open the **CodeScoring SCA** tool window with the Dashboard panel



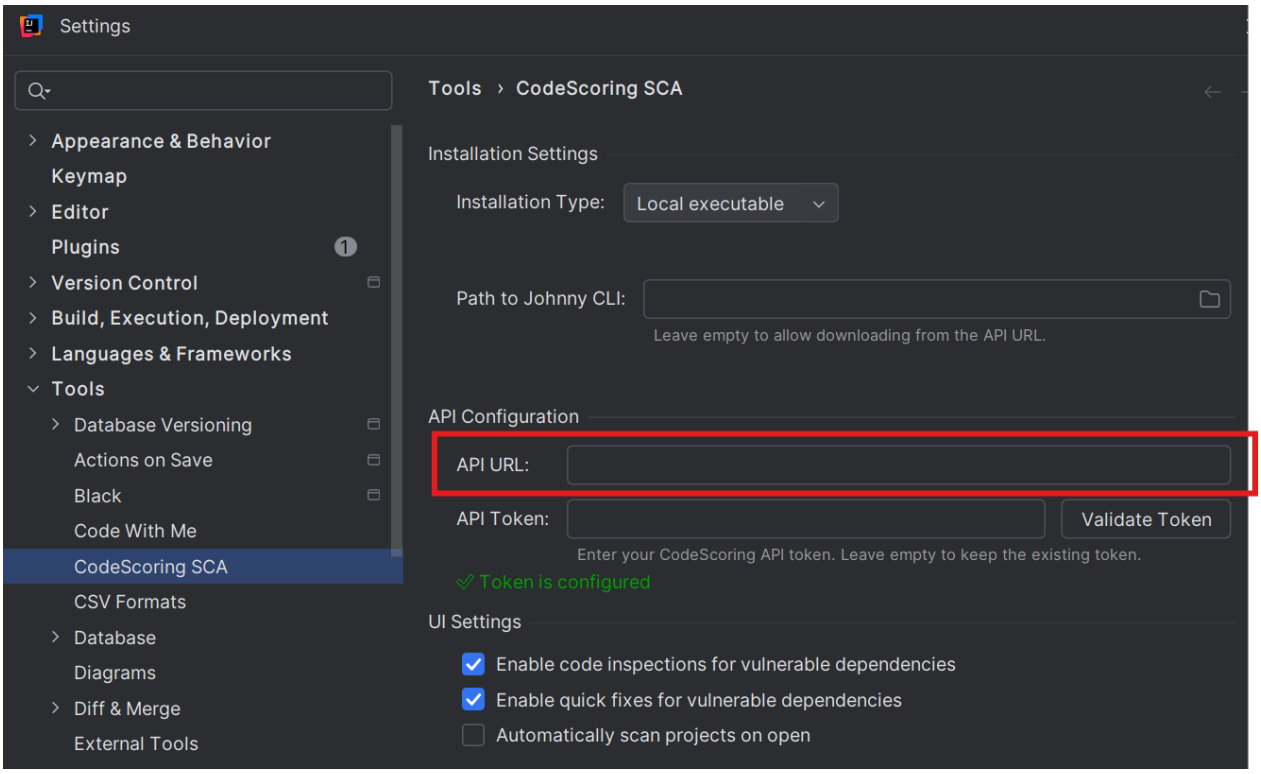
#Step 4: Configure the Plugin

1. In the CodeScoring SCA tool window, click the **"Settings"** button (gear icon) in the toolbar
2. This will open the CodeScoring SCA settings page



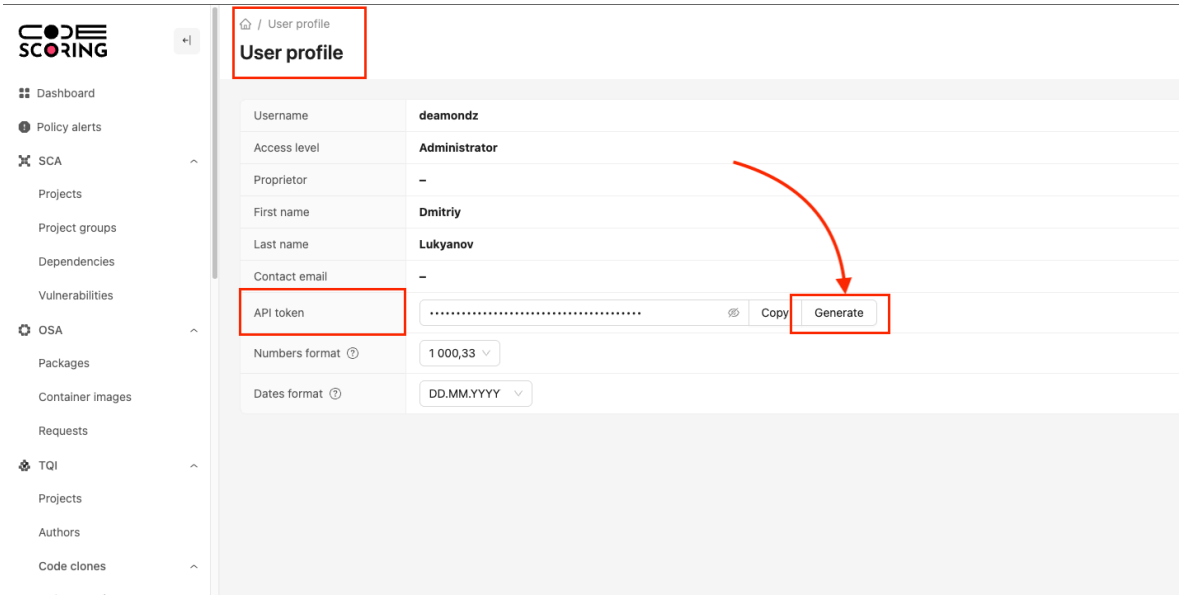
#4.1 Verify the API URL

1. In the settings, locate the **API URL** field
2. You need to use the URL of CodeScoring installed in your organization. Please, contact your administrator.

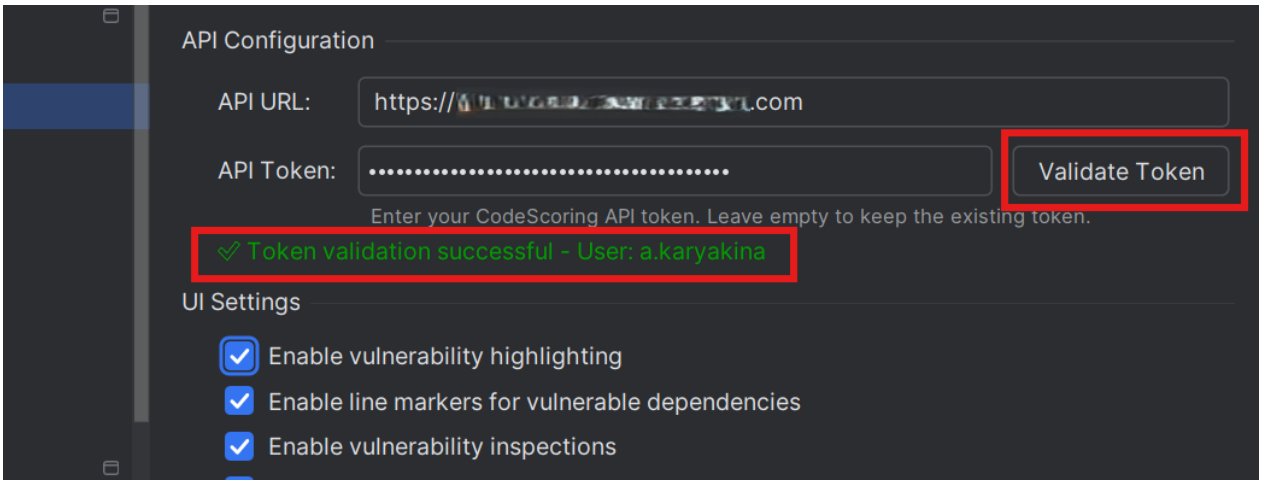


#4.2 Generate and Set API Token

1. Open your web browser and navigate to: `<API URL>/cabinet/profile`
2. Log in to your CodeScoring account
3. Ensure you're on your profile page
4. Locate the **"API token"** field
5. Click the **"Generate"** button next to the API token field

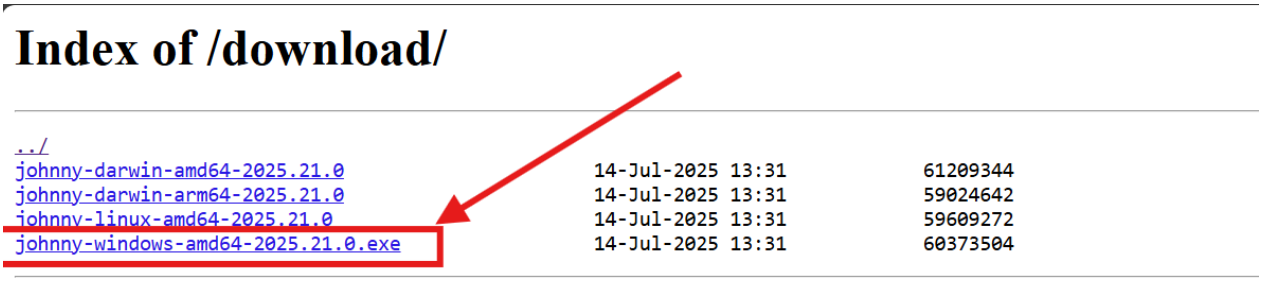


6. Copy the generated API token value
7. Return to IntelliJ based IDE settings
8. Paste the token into the **"API Token"** field
9. Click **"Validate Token"** to verify the token works
10. The plugin should display confirmation that the token is valid



#4.3 Download Johnny CLI (Optional)

1. Navigate to the releases page: `<API URL>/download/` (note the trailing slash)
2. Download the latest release executable for your operating system



#4.4 Configure Johnny CLI

There are three ways to get Johnny CLI for analyzing your dependencies with our service.

4.4.1 Local installation

Prerequisites:

- Johnny CLI must be downloaded and the file made executable on the system
- Operating system must allow running the executable file (to verify, run the file once in the console manually with the --help parameter)

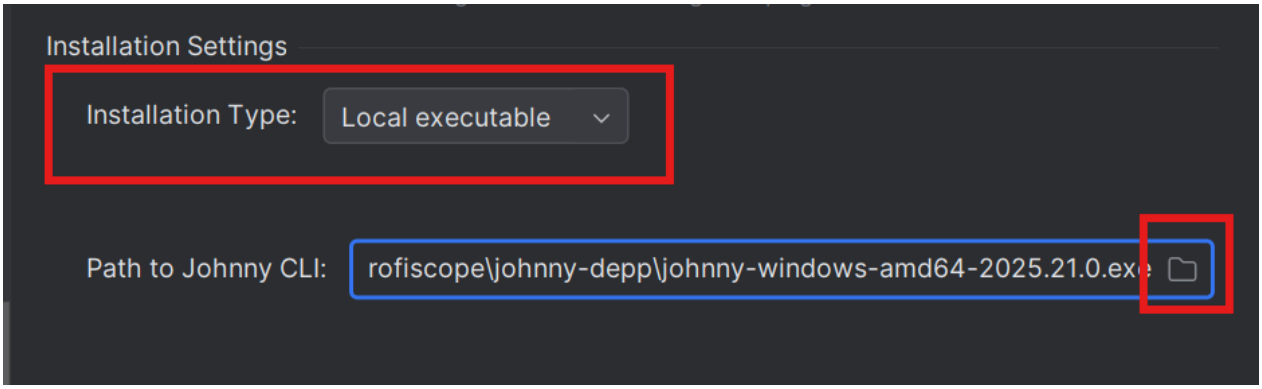
Setup Steps:

1. Set installation type to Local

- In settings, select **"Local executable"** from the **platform Type** dropdown

1. In the **Johnny CLI Path** field, click the folder selection button
2. Navigate to where you previously downloaded Johnny CLI
3. Select the Johnny CLI executable file
4. Click **"OK"** to save settings

Note: If the plugin asks about changing file permissions (chmod +x), click **"Yes"** to allow the plugin to make the file executable.



4.4.2 Automatic Client Download

Prerequisites:

- API URL must be configured
- API token must be configured and validation must pass successfully

Setup steps:

1. Set installation type to Local

- In settings, select **"Local executable"** from the **platform Type** dropdown

1. Leave the **Johnny CLI Path** field empty
2. Now when scanning is first requested, Johnny CLI will be downloaded from the API URL. This will allow you to automatically get client updates as soon as they are available. The downloaded client will be saved in the following location:
    - **Linux/MacOS**: `~/.codescoring/johnny`
    - **Windows**: `%USERPROFILE%\.codescoring\johnny.exe`

4.4.3 Using Docker

Docker installation allows running Johnny CLI in an isolated container, which is useful when you don't want to install it directly on your system.

Prerequisites:

- Docker must be installed and running on your system
- Your user must have permissions to execute Docker commands

Setup steps:

1. Set installation type to Docker

- In settings, select **"Docker"** from the **Installation Type** dropdown

1. Configure Docker image

- **Docker Image**: `johnny-depp:2025.29.0` (default)
- **Docker Registry**: `<codescoring-registry-host>`
- Sample full image path: `sample-codescoring-registry.com/johnny-depp:2025.29.0`

1. **Optional: Additional Docker Options** Add custom Docker run options if needed in the **Additional Docker Options** field:

```
--memory=2g --cpus=2
```

How it works:

- The client automatically mounts your project directory into the container
- Scanning is performed inside the container, and results are saved in your project
- No manual Docker commands are required - the plugin handles everything automatically

Troubleshooting Docker installation:

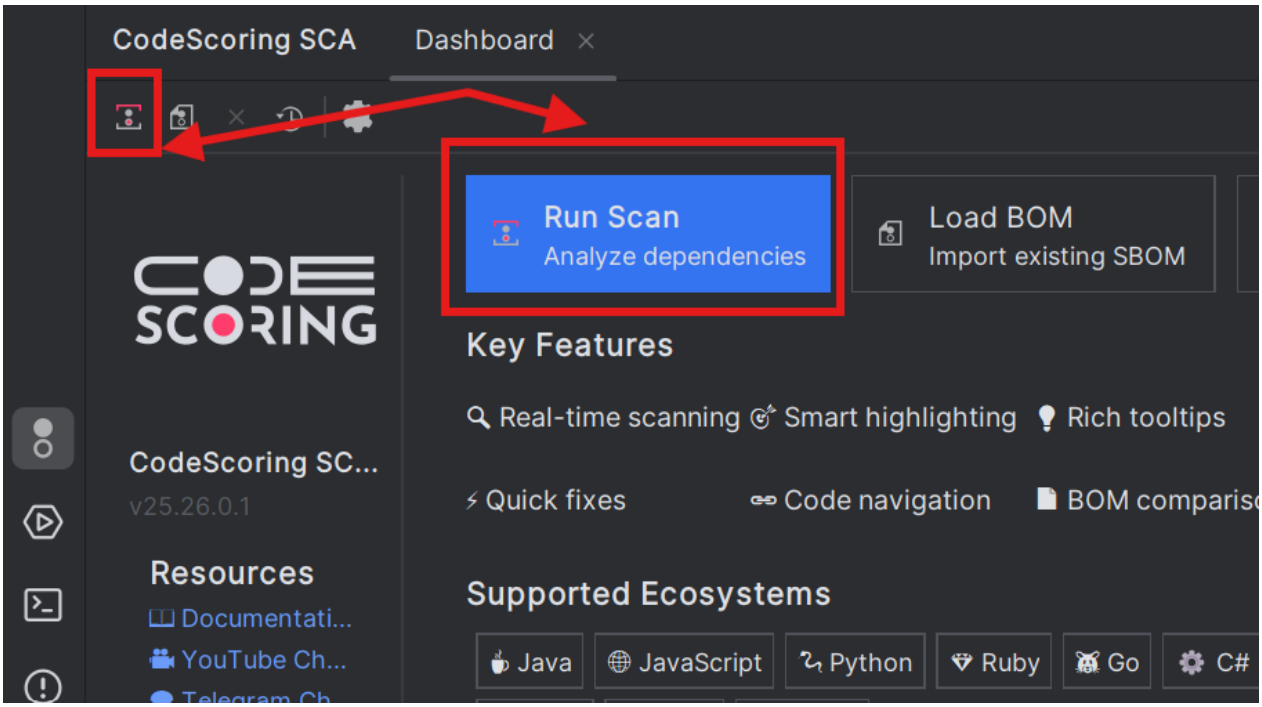
- **"Docker not found"**: Ensure Docker is installed and the `docker` command is in your PATH
- **Permission denied**: Add your user to the docker group: `sudo usermod -aG docker $USER`
- **Image pull failed**: Check registry credentials and network connection
- **Container exits immediately**: Check Event Log for detailed error messages

#Step 5: Run Your First Scan

Now that the plugin is configured, you can run your first dependency scan:

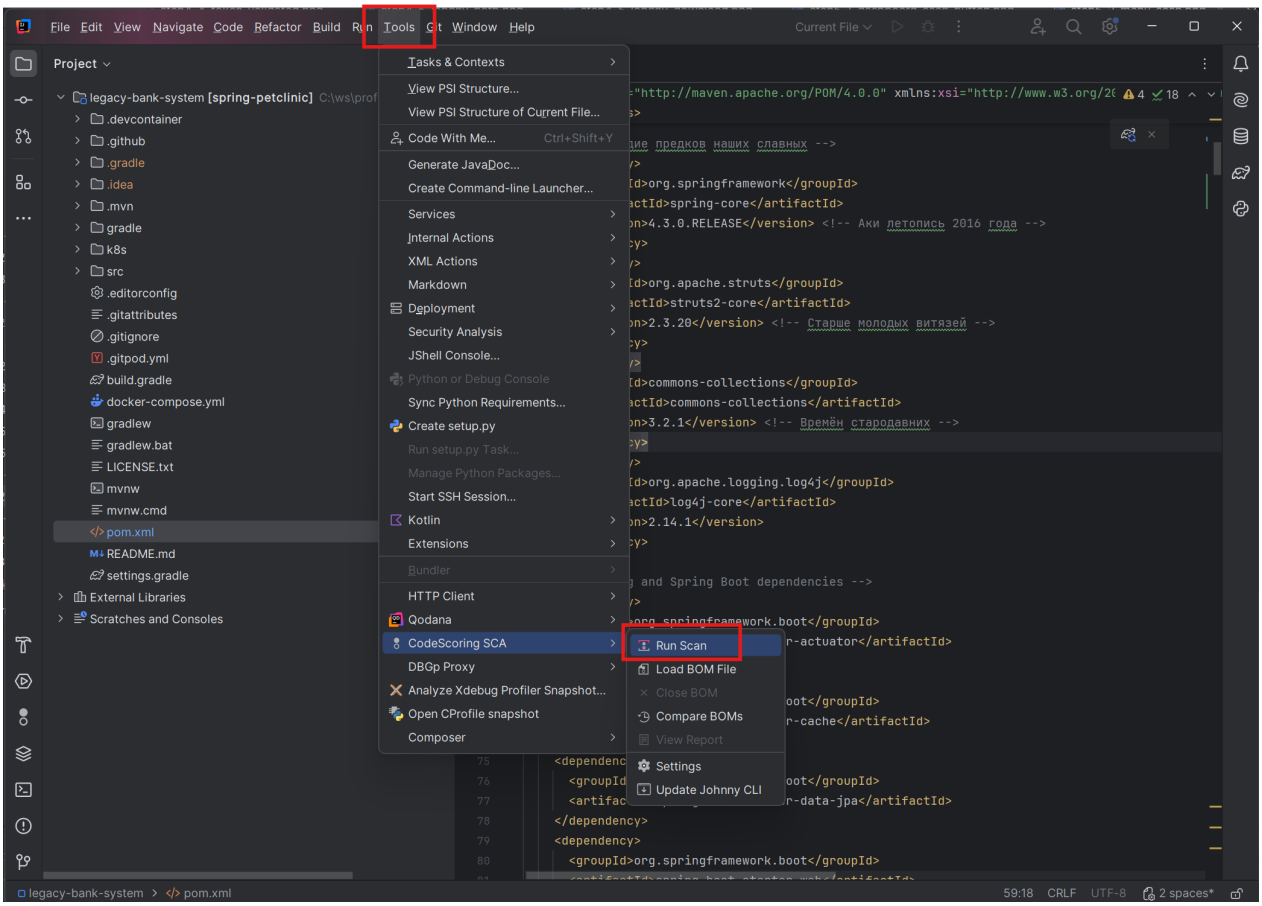
#Method 1: Using the Dashboard

1. Open a project in IntelliJ based IDE
2. Open the CodeScoring SCA tool window
3. In the Dashboard panel, click the **"Run Scan"** button



#Method 2: Using the Main Menu

1. Go to **Tools → CodeScoring SCA → Run Scan**



#Method 3: Using the Toolbar

1. Find the **CodeScoring SCA** plugin tab and its main toolbar

2. Click the **"Run Scan"** button (see screenshot from method 1)

#Step 6: Fine-tuning Scan Configuration

After the scan completes, you'll see a new `.codescoring` directory containing:

1. **config.yaml** - configuration file for Johnny CLI, you can read about it in this section. You can modify this file as it will never be overwritten.
2. **donotfix.yaml** - configuration file listing filename patterns that should be excluded from Quick Fix actions. You may modify this file.
3. **report.html** - report generated in html format containing Johnny CLI output in colored table format. Overwritten during each scan.
4. **bom.json** - scan results file created by Johnny CLI in cyclone-dx 1.6 format, it will be loaded automatically and shown in the vulnerabilities panel for any project opened in IntelliJ that has .codescoring/bom.json
5. **bom.json.N** - where N is the scan revision, i.e. 0 is the previous scan, and 5 (for example) is the very first scan, and bom.json.0 will be used for comparison with bom.json (and shown in the DIFF tree) if it exists at the time of project opening

›
your-project
  ›
  .codescoring

    config.yaml# Scan configuration

    donotfix.yaml# QuickFix configuration

    report.html# Latest scan report

    bom.json# Current vulnerabilities

    bom.json.0# Previous scan (comparison)

    bom.json.1# Older scans...

  pom.xml# Your dependencies

  ...your code ...

#Example of Using CodeScoring Scan Configuration

Edit `.codescoring/config.yaml` to customize:

```
scan:
  general:
    ignore:           # Directories to skip
      - target
```

```
      - build
      - .idea
    with-hashes: true # Include file hashes for exact matching
    only-hashes: false # Use only hash-based detection
  dir:
    no-recursion: false # Prevents recursive scanning of root
directory
```

#Configuring Quick Fix Exclusions

The `.codescoring/donotfix.yml` file controls which files should not be modified by Quick Fix actions. This is particularly useful for generated files (like lock files) that should be regenerated rather than manually patched.

Example of using QuickFix Exclusion configuration:

```
## Lock files and generated files that should not be directly
modified
patterns:
  - go.sum
  - package-lock.json
  - yarn.lock
  - Cargo.lock
  - composer.lock
  - "*.generated.*"
  - "**/generated/**"
```

Pattern syntax:

- Exact filename: `go.sum`
- Wildcard patterns: `*.lock`, `*-lock.json`
- Directory patterns: `**/node_modules/**`
- Multiple extensions: `*.{lock,generated}`

The file is created automatically on:

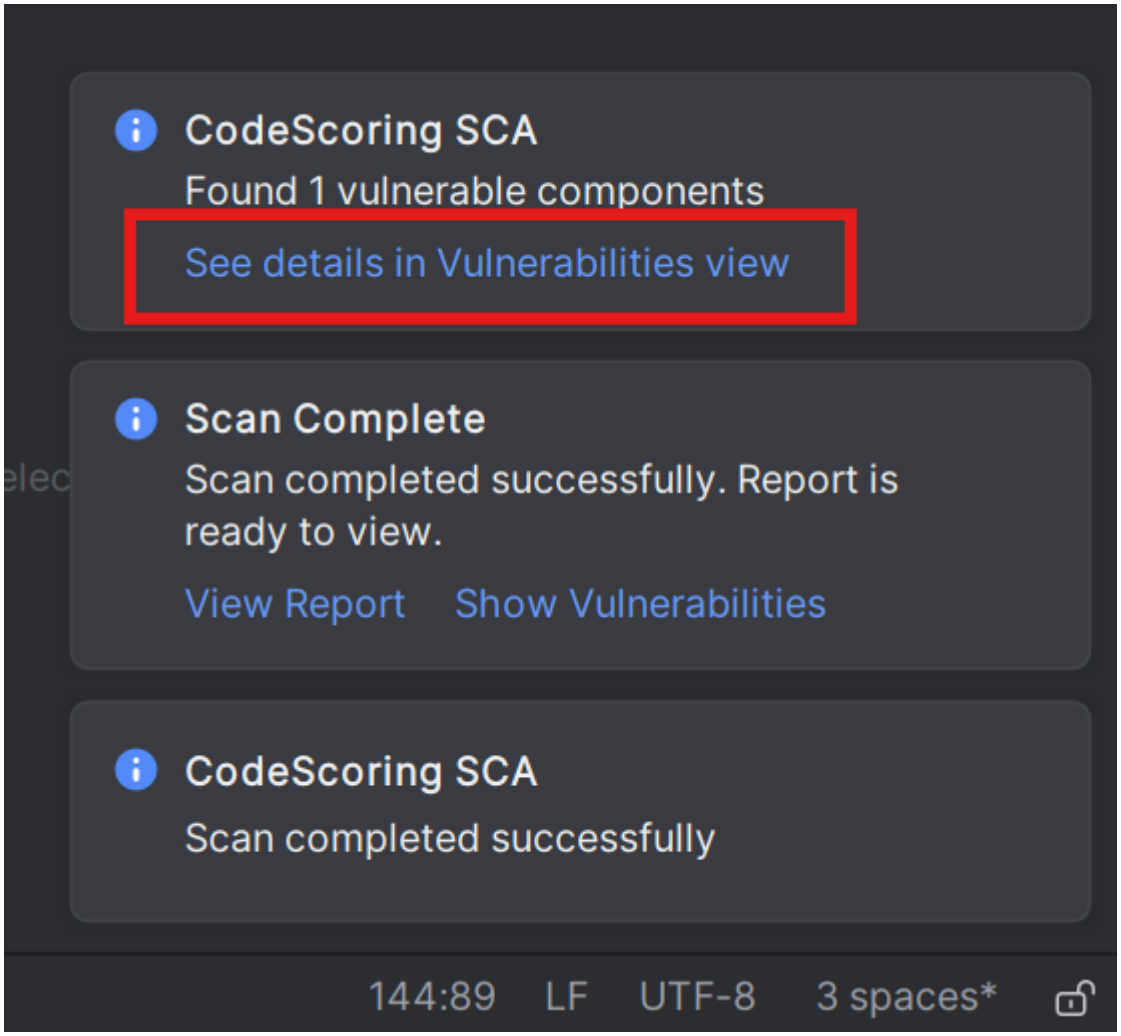
- First scan in a project
- Opening a project with existing `.codescoring` directory but no `donotfix.yml`

Note: Files matching these patterns will still be scanned for vulnerabilities and shown in the results, but the Quick Fix actions (both individual and bulk) will skip them. Users should regenerate these files using their package manager commands instead.

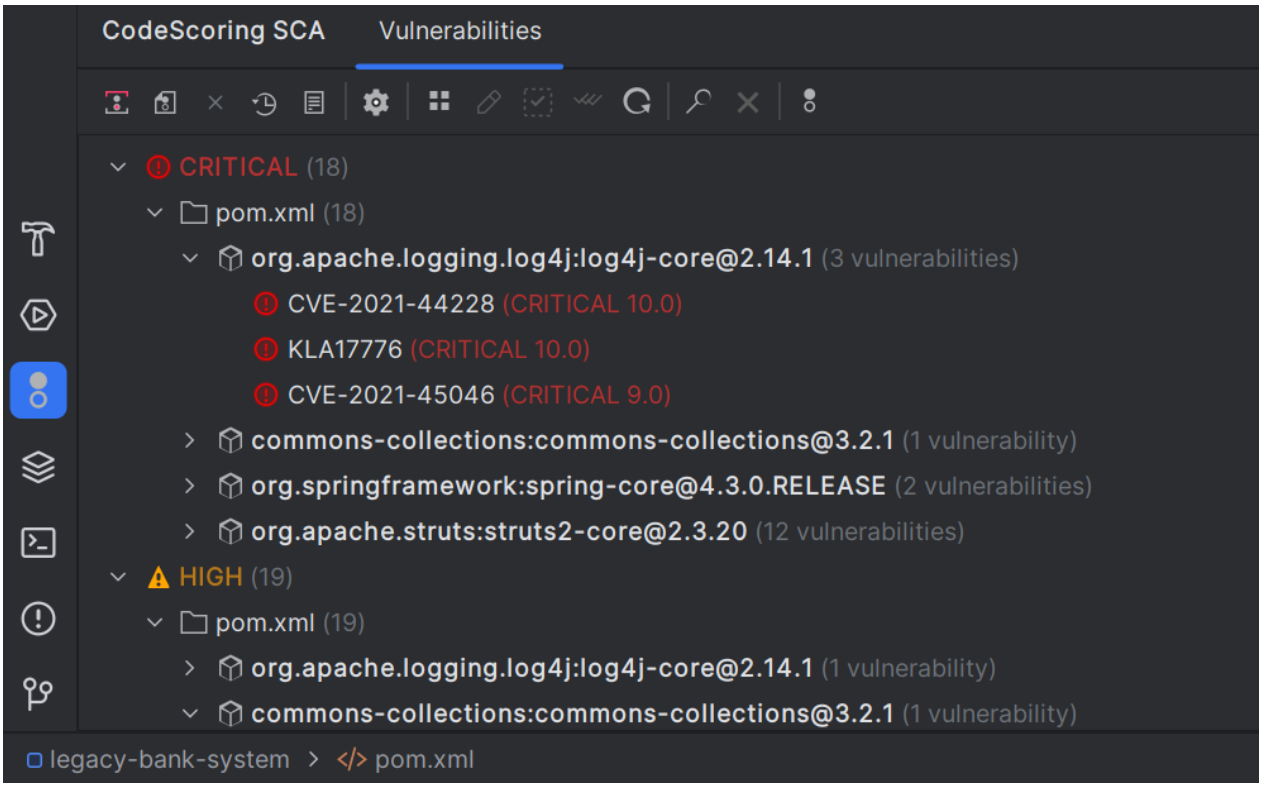
#Step 7: View Scan Results

After the scan completes:

1. **Check notifications**: Look at the notifications in the bottom right corner of IntelliJ IDEA, clicking on **"View Report"** and **"See details in Vulnerabilities view"**

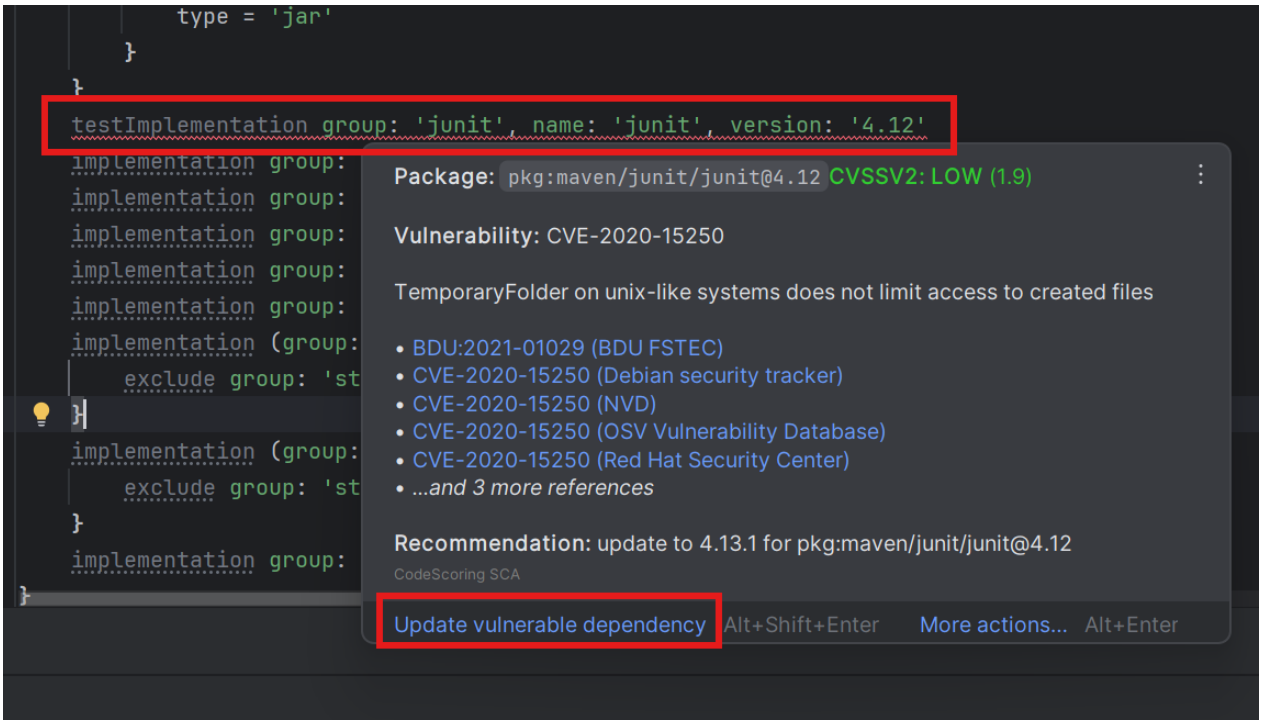


2. **Open vulnerabilities tree**: The plugin window will automatically switch to the **Vulnerabilities** panel
3. **Review vulnerabilities**: The vulnerabilities panel will show all detected security issues in your dependencies
4. **Explore details**: You can click on individual vulnerabilities to see detailed information in the details panel
5. **Apply fixes**: Use the **"Fix All"** or **"Fix Selected"** buttons or individual quick fixes to update vulnerable dependencies
6. **View alerts**: You can click on the **Alerts** panel to view alerts list of triggered policies



#7.1 Vulnerability Highlighting

- **In-code highlighting**: Vulnerable dependencies are highlighted directly in code files (build.gradle, pom.xml, package.json, etc.)
- **Severity colors**:
  - Critical (red)
  - High (orange)
  - Medium (yellow)
  - Low (blue)
- **Multi-file support**: Works with all supported file types
- **Hover** over highlighted dependencies to see vulnerability details



#7.2 Hover Information

When hovering over highlighted dependencies, displays:

- **Vulnerability ID**: CVE number with link
- **Severity**: CVSSv3 score and level
- **Description**: What the vulnerability does
- **Source links**: Official vulnerability registration information
- **Recommendations**: Suggested versions to upgrade to
- **Quick fix**: One-click update option

#7.3 Vulnerabilities Panel

7.3.1 Tree Structure

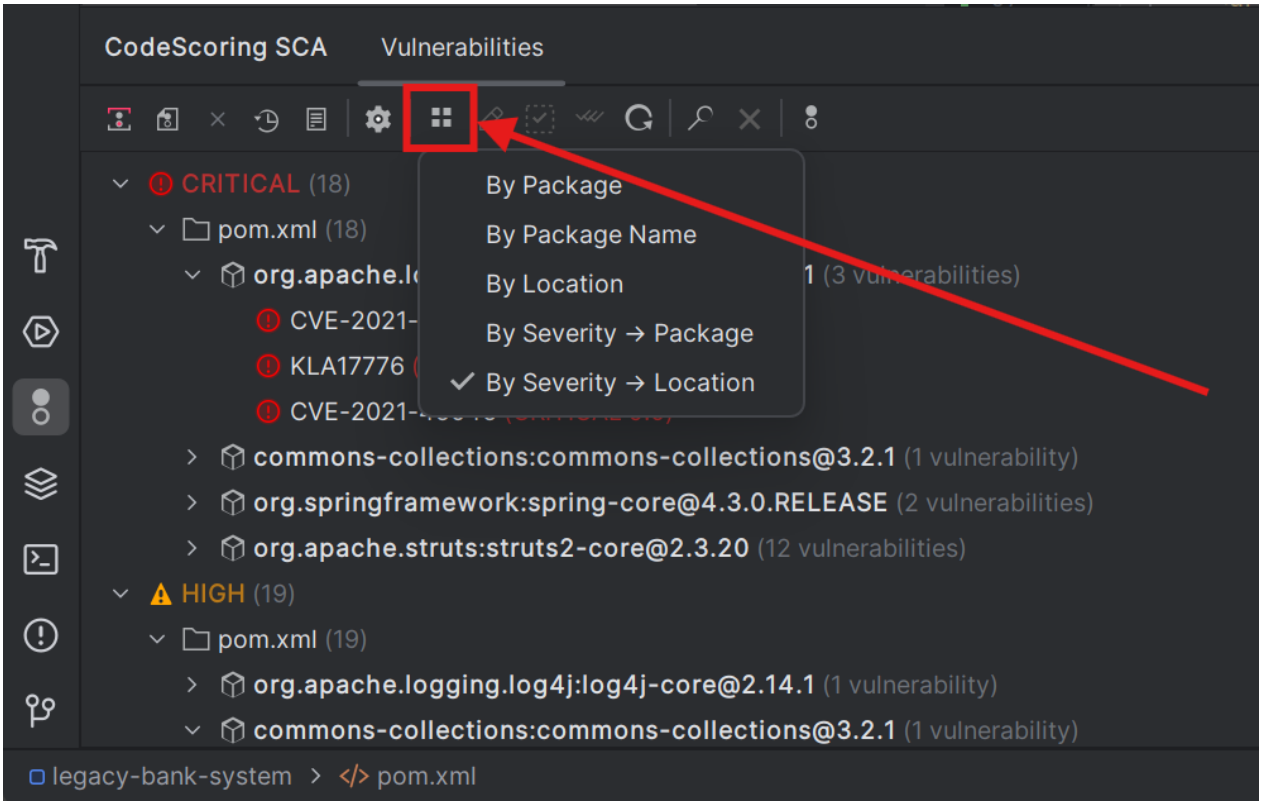
```
 Vulnerabilities (247)
├──  Critical (12)
│   ├── CVE-2023-1234 - Remote Code Execution
│   │   ├── lodash@4.17.20
│   │   └── pom.xml:15
│   └── ...
├──  High (45)
├──  Medium (89)
└──  Low (101)
```

7.3.2 Grouping Options

- Use the vulnerabilities panel to filter by severity, package, or other criteria
- Group vulnerabilities by different categories for better organization

Change grouping via toolbar button or command:

- **By Severity → Location → Component** (default): By severity level, then by file then by package
- **By Severity → Component**: By severity level, then by package name alphabetically
- **By Location → Component**: Grouped by file path
- **By Component Name → Component**: Groups different versions of the same package together alphabetically
- **By Component**: By component (with version if known)



#7.4 Search and Filtering

Search Capabilities

- **Multiple fields**: Search by:
  - Package name (e.g., "lodash")
  - CVE identifier (e.g., "CVE-2023")
  - File path (e.g., "frontend/")
  - Severity level
- **Fuzzy matching**: Finds partial matches
- **Exact matching**: Search for exact word match in quotes
- **Case insensitive**: Exact case not required

Search Toolbar

For searching and filtering, the plugin has:

- **Search field**: For entering search queries
- **Clear button**: Reset search

- **Results indicator**: Shows number of found items as first element of vulnerable components tree

#7.5 Quick Fixes

In-code Quick Fixes

- **IntelliJ lightbulb**: Click the lightbulb next to a vulnerable dependency

or

- **Alt+Enter**: Use the hotkey when cursor is on a vulnerable dependency

or

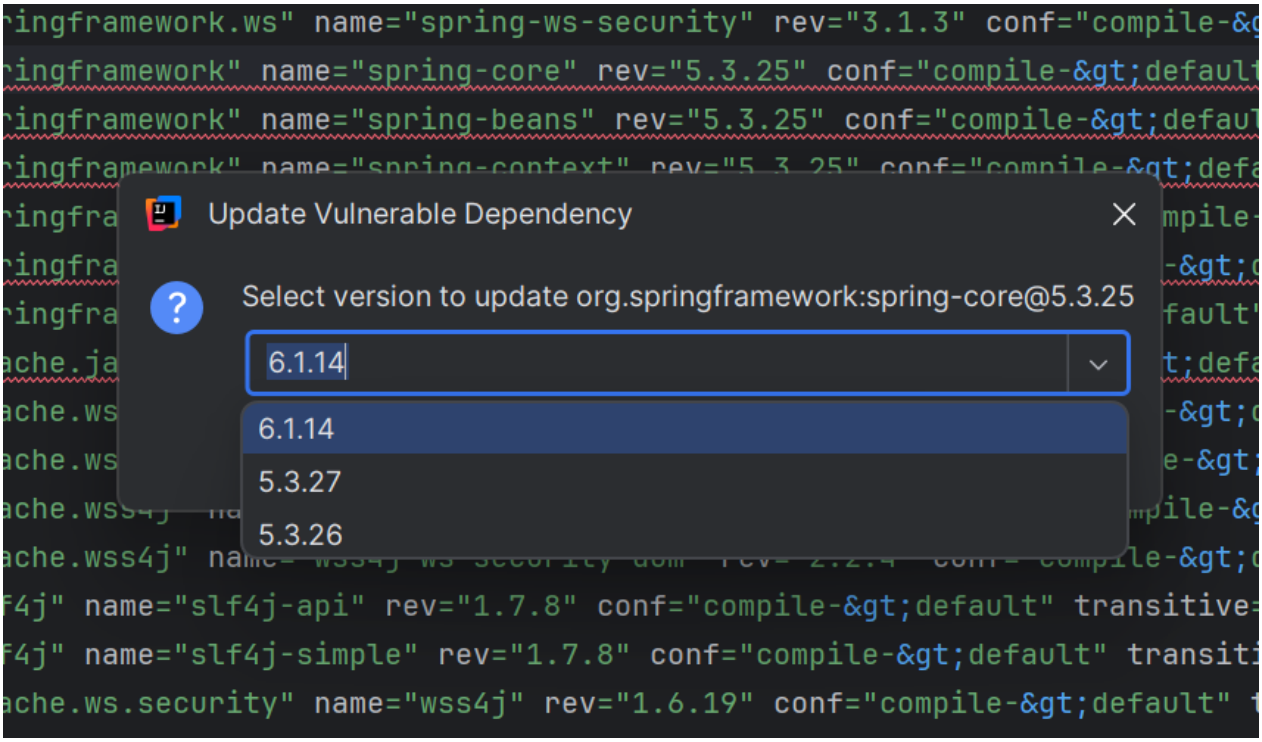
- **Update vulnerable dependency** at the bottom of the vulnerable component hover card

or

- **Fix Selected** button on the toolbar,

then

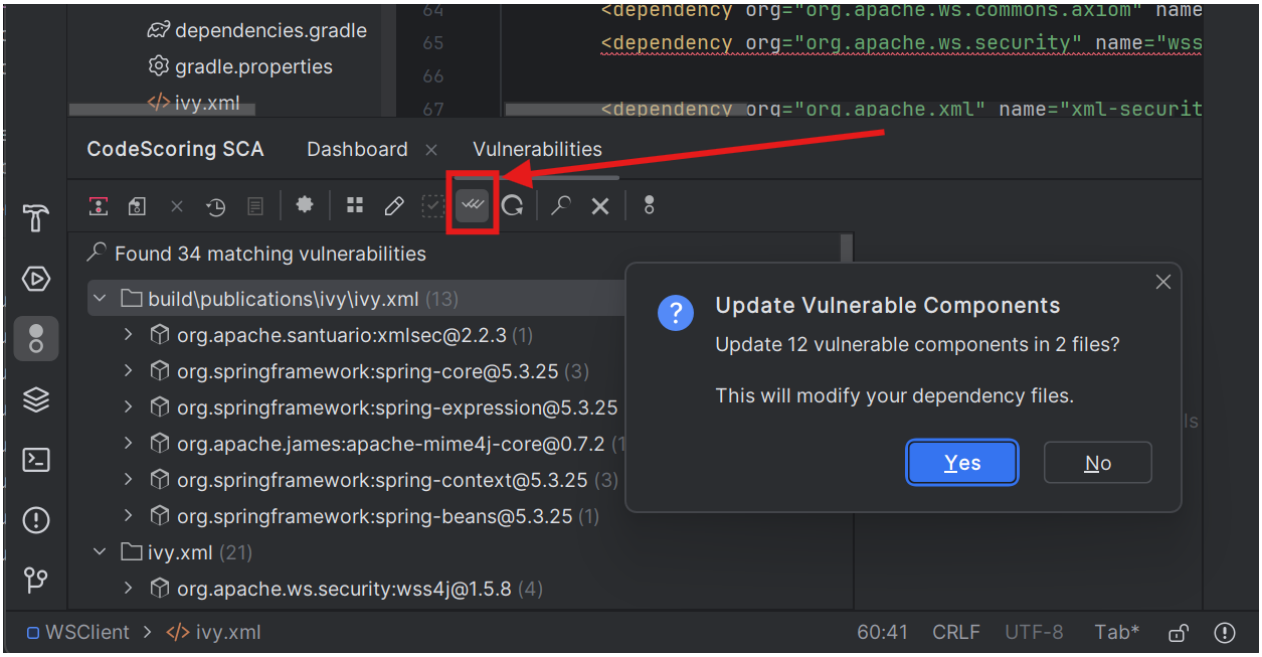
- **Choose version**: If multiple safe versions are available, select the appropriate one



Bulk Fixes (doesn't work with severity groupings)

- **Fix All button**: Updates all vulnerable components with available fixes
- **Intelligent update**: Automatically selects the most appropriate safe version
- **Change report**: Shows how many dependencies were updated

- **Exclusions**: Respects patterns defined in `.codescoring/donotfix.yml`



Important Note: Files matching patterns in `donotfix.yml` will be skipped during Quick Fix operations. For these files (typically lock files), use your package manager to regenerate them after updating the main dependency files.

#7.6 Working with BOM Files

Auto-loading

The plugin automatically loads BOM files from:

1. `.codescoring/bom.json` (primary)
2. `bom.json` (project root directory)

Manual Operations

- **Load BOM**: **Tools → CodeScoring SCA → Load BOM File**
- **Close BOM**: **Tools → CodeScoring SCA → Close BOM**

#7.7 BOM Comparison

Automatic Comparison

When opening a project:

- Loads current BOM (`bom.json`)
- Compares with previous (`bom.json.0`)
- Shows change notification

Manual Comparison

1. Select **Tools → CodeScoring SCA → Compare BOMs**

2. Select base BOM file
3. If BOM is already loaded, it will be used as target
4. If BOM is not loaded, select target file
5. Comparison result will be displayed in DIFF panel

Comparison Views

```
 BOM DIFF (Changes: 23 added, 15 removed, 45 updated, 73
unchanged)
├── + Added (23)
│   ├── [ADDED] react@18.1.3 0 vulnerabilities
│   └── ...
├── − Removed (15)
├──  Updated (45)
│   ├── [UPDATED] lodash: 4.17.20
│   └── ...
└── ✓ Unchanged (73)
```

Comparison Grouping Options

- **By change type**: Added/Removed/Updated/Unchanged (default)
- **By package**: Alphabetical package grouping, most useful view for tracking changed package versions
- **By location**: Grouped by file path
- **By severity**: Grouped by vulnerability impact

Comparison Filtering

Use the search field to filter comparison results by:

- Package name
- Change type
- File path

#7.8 Reports

Scan Reports

- **Automatically generated**: Created after each scan
- **Location**: `.codescoring/report.html`
- **Format**: Colored HTML with detailed information
- **Content**:
  - Scan status
  - Executed command
  - Results summary
  - Found vulnerabilities
  - Policy warnings
  - Error messages

Viewing Reports

- **Command**: **Tools → CodeScoring SCA → View Report**
- **Opens in**: Choice of external browser, internal preview, code editor

#7.9 Alerts panel

The **Alerts** panel displays information about alerts for triggered policies based on the analysis results. For each alert, the following is shown:

- Policy name
- Alert level
- Blocking status
- List of packages with criteria

#Step 8: Settings and Customization

#Available Settings List

Setting	Description	Default
API Configuration		
`API URL`	URL of your CodeScoring platform	
`API Token`	API token. Securely stored	*(set via UI)*
Installation Settings		
`platform Type`	Local executable or Docker	`Local executable`
`Path to Johnny CLI`	Path to Johnny CLI (empty for auto-download)	*(auto-download)*

Setting	Description	Default
`Docker Image`	Docker image name	`johnny-depp:2025.29.0`
`Docker Registry`	Docker registry	*(provided by support)*
`Additional Docker Options`	Additional Docker options	
UI Settings		
`Enable vulnerability inspections`	Enable code inspections	`true`
`Enable quick fixes for vulnerable dependencies`	Allow quick fixes	`true`
`Automatically scan projects on open`	Run scan when opening project	`true`
Severity Colors		
`Critical Color`	Color for critical vulnerabilities	*(red)*
`High Color`	Color for high vulnerabilities	*(orange)*
`Medium Color`	Color for medium vulnerabilities	*(yellow)*
`Low Color`	Color for low vulnerabilities	*(blue)*
`Unknown Color`	Color for unknown severity	*(gray)*

#Troubleshooting

#Plugin Logs

- For detailed understanding of plugin operation, check IDE logs:
  - **Help → Show Log in Explorer/Finder**
  - Look for entries with "CodeScoring" in `idea.log`

#**Common Issues**

Installation Issues

Issue	Solution
Plugin not visible after installation	Fully restart IntelliJ IDEA, check **Settings → Plugins**
Compatibility error	Ensure IDE version is 2024.1 or newer, check for issues in log file
Installation hangs	Check internet connection, try reinstalling

Configuration Issues

Issue	Solution
Token validation fails	Check API URL, generate new token, check proxy/VPN
Johnny CLI not found	Check path, access permissions, antivirus
Docker not working	Ensure Docker is running, check user permissions

Scanning Issues

Issue	Solution
Scan hangs	Check Event Log, try running Johnny CLI manually
No results	Ensure project contains dependency files
Partial results	Check configuration in .codescoring/config.yaml
Token errors	Check token expiration, access permissions

Display Issues

Issue	Solution
No highlighting	Enable in settings, reload files
Wrong colors	Check severity color settings

Issue	Solution
Missing panel	**View → Tool Windows → CodeScoring SCA**
Slow performance	Reduce pagination size in settings

Fix Issues

Issue	Solution
Fix fails	Check file write permissions
Fix is ignored	Check plugin logs and .codescoring/config.yaml
Wrong version	Manually specify version in file
Version conflicts	Fix one component at a time
Rollback changes	Use version control system

#Getting Help

1. **Check Event Log**: **View → Tool Windows → Event Log**
2. **Enable debug logs**:
    - **Help → Diagnostic Tools → Debug Log Settings**
    - Add `com.codescoring.intellij`
3. **Error reports**: Review `.codescoring/report.html` for scan details

Contact support: support@codescoring.ru

#Security and Privacy, Data Handling

- **Local scanning**: Code is not sent to servers
- **API communication**: Only metadata is transmitted (your package manager configuration files)
- **Token storage**: Secure VS Code credential storage

#Best Practices

#Workflow Recommendations

1. **Initial setup**: Full scan when starting project
2. **Reviews**: Compare BOMs between versions
3. **CI/CD**:
    - **Before commit**: Run full scan

    - **Share configuration**: Commit `.codescoring/config.yaml` and `.codescoring/donotfix.yaml`
    - **Ignore temporary files**: Add to `.gitignore`:

```
  .codescoring/report.html
  .codescoring/bom.json.*
```

```
- **Track main BOM**: Version `.codescoring/bom.json`
- **Standardize**: Standardize plugin settings
```

#Integration with Development Processes

1. **Code Review**:
    - Check dependency changes
    - Require critical vulnerability fixes
    - Document accepted risks
2. **Release Management**:
    - Generate reports for each release
    - Track security improvements
    - Plan dependency updates
3. **Compliance**:
    - Export BOM for audit
    - Track component licenses
    - Maintain scan history

#Working with Lock Files

1. **Understanding Lock Files**:
    - Lock files are generated by package managers
    - They should not be manually edited
    - Changes should be made to manifest files instead
2. **Quick Fix Behavior**:
    - Files matching `donotfix.yml` patterns are skipped
    - Update manifest files, then regenerate lock files
    - Use appropriate package manager commands, for instance:
        - **Go**: `go mod tidy`
        - **NPM**: `npm install`
        - **Yarn**: `yarn install`
        - **Cargo**: `cargo update`
3. **Best Practices**:
    - Review `donotfix.yaml` and adjust patterns as needed

    - Document your regeneration process
    - Automate lock file updates in CI/CD

#Working with dependencies

In some cases, when working with dependencies, additional actions are required to improve the accuracy of composition analysis.

This section contains instructions and best practices for working with dependencies in different ecosystems, as well as a description of how CodeScoring works in non-standard dependency processing scenarios.

#Working with dependencies in Java

When working with Java, you can create additional artifacts that contain the full dependency structure of the project.

#Apache Maven:

#Creating a `maven-dependency-tree.txt` file

```
mvn dependency:tree -DoutputFile=maven-dependency-tree.txt
```

#Gradle:

#Creating a `gradle-dependency-tree.txt` file

```
./gradlew dependencies > gradle-dependency-tree.txt
```

#Creating a `gradle-dependency-tree.txt` file for multi-project builds

To analyze dependencies in Gradle projects, Johnny uses the `gradle-dependency-tree.txt` file. In regular projects, it is generated automatically. However, in multi-project builds, its correct generation is possible only if the project has a special task with the expected name.

To get all dependencies in this case, you need to do the following:

#Manifest Coupling

- If a directory contains build.gradle and gradle.lockfile without name matching, they will be coupled;
- When all three manifests are present in one directory (build.gradle, gradle.lockfile, gradle-dependency-tree.txt), priority in coupling is given to gradle-dependency-tree and gradle.lockfile is parsed separately;

- When multiple lock files for one build.gradle are present in one directory without name matching, any one of them will be used for coupling. The remaining lock files are parsed separately.

#Groovy

Add to `build.gradle` file code:

```
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

Add to `build.gradle.kts` file code:

```
subprojects {
    afterEvaluate {

tasks.register<DependencyReportTask>("CodeScoring_All_Dependen
cies")
    }
}

tasks.register("CodeScoring_All_Dependencies") {
    dependsOn(subprojects.mapNotNull {
        it.tasks.findByName("CodeScoring_All_Dependencies")
    })
}
```

Execute command:

```
./gradlew CodeScoring_All_Dependencies > gradle-dependency-
tree.txt
```

After generating artifacts, you need to use the console agent command scan file on the resulting artifact, for example:

```
./johnny \
scan file ./gradle-dependency-tree.txt \
--api_token <api_token> \
--api_url <api_url>
```

#Working with dependencies in Scala

#sbt

#Generating `scala-dependency-tree.txt` or `sbt-dependency-tree.txt` files

1. **Setting the width of the dependency graph**

   To generate a full dependency graph, add the following line to the `build.sbt` file:

   ```
   ThisBuild / asciiGraphWidth := 999999999
   ```

   Alternatively, you can set the `asciiGraphWidth` value globally.

2. **Generating dependency tree**

   Run the following command to generate dependency tree:

   ```
   sbt clean compile "dependencyTree::toFile target/tree.txt"
   ```

   Make sure to save the file with the name `scala-dependency-tree.txt` or `sbt-dependency-tree.txt`, as these are the only names supported for correct parsing.

3. **Scanning the generated file**

   The console agent option `--sbt-resolve` in the scan command is not needed in this case, as it scans the already generated tree with the full dependency structure.

#Working with dependencies in Go

#Go Modules

#Creating a `go.sum` file

1. Initialize the module:

```
go mod init <module_name>
```

2. Install dependencies:

```
go get <package>
```

3. After installing dependencies, `go.mod` and `go.sum` files are automatically created and updated.

4. Fixate dependency versions:

```
go mod tidy
```

#Working with dependencies in JavaScript

#NPM

#Creating a `package-lock.json` file

1. Initialize the project:

```
npm init -y
```

1. Install dependencies:

```
npm install
```

#Support for the NPM package alias mechanism

The NPM package alias mechanism allows you to install packages under different names, which is convenient for using multiple versions of a library at the same time, replacing a dependency without changing its name in the code, and working with forks.

Instead of the standard version specification, a syntax is used that explicitly specifies which package and its version to install under the desired name. This simplifies testing, updates, and dependency compatibility.

In `package.json`, the dependencies section may contain the following entry:

```
"dependencies": {
"@babel/legacy-core": "npm:@babel/core@=7.12.0"
}
```

The Johnny console agent handles this entry correctly, recognizing that **@babel/legacy-core** is an alias for **@babel/core** version 7.12.0. The original package is taken into account during dependency analysis, preventing errors related to non-existent names.

#Support for the NPM overrides mechanism

The NPM overrides mechanism allows to replace the version of a transitive dependency. It is helpful when replacing a version with a known security issue or replacing an existing dependency with a fork.

In `package.json`, the overrides section may contain the following entry:

```
"overrides": {
  "foo": "1.0.0"
}
```

`package-lock.json` contains single version of package `foo`:

```
"node_modules/foo": {
"version": "1.0.0",
```

The Johnny console agent handles this entry correctly, recognizing package **foo** version 1.0.0.

#Support for the NPM workspaces mechanism

The NPM workspaces mechanism allows to support managing multiple packages from your local file system within a singular top-level, root package.

In `package.json`, the workspaces section may contain the following entry:

```
"workspaces": [ "packages/a", "packages/b" ]
```

The Johnny binary agent processes all valid `package.json` from workspaces together.

#PNPM

#Creating a `pnpm-lock.yaml` file

1. Initialize the project:

   ```
   pnpm init -y
   ```

2. Install dependencies:

```
pnpm install
```

#Support for the PNPM package alias mechanism

The PNPM package alias mechanism allows to install packages under different names, which is convenient for using multiple versions of a library at the same time, replacing a dependency without changing its name in the code, and working with forks.

Instead of the standard version specification, a syntax is used that explicitly specifies which package and its version to install under the desired name. This simplifies testing, updates, and dependency compatibility.

In `package.json`, the dependencies section may contain the following entry:

```
"dependencies": {
"lodash-old": "npm:lodash@3.10.1"
}
```

The Johnny binary agent handles this entry correctly, recognizing that **lodash-old** is an alias for **lodash** version 3.10.1. The original package is taken into account during dependency analysis, preventing errors related to non-existent names.

#Support for the PNPM overrides mechanism

The PNPM overrides mechanism allows to replace the version of a transitive dependency. It is helpful when replacing a version with a known security issue or replacing an existing dependency with a fork.

In `package.json`, the pnpm/overrides section may contain the following entry:

```
"pnpm": {
  "overrides": {
    "example-package": "^1.3.0"
  }
}
```

`pnpm-lock.yaml` contains package `example-package` with version equal or high 1.3.0:

```
example-package@1.3.0: {}
```

The Johnny console agent handles this entry correctly, recognizing package **example-package** version 1.3.0.

#Support for the PNPM workspaces mechanism

The PNPM workspaces mechanism allows you support managing multiple packages from your local file system from within a singular top-level, root package.

In `pnpm-workspace.yaml` in folder with root package `package.json` may contain the following entry:

```
packages:
- 'packages/*'
```

The Johnny binary agent process all valid `package.json` from workspaces together.

#Yarn

#Creating a `yarn.lock` file

1. Initialize the project:

```
yarn init -y
```

1. Install dependencies:

```
yarn install
```

#Support for the Yarn package alias mechanism

The Yarn package alias mechanism allows to install packages under different names, which is convenient for using multiple versions of a library at the same time, replacing a dependency without changing its name in the code, and working with forks.

Instead of the standard version specification, a syntax is used that explicitly specifies which package and its version to install under the desired name. This simplifies testing, updates, and dependency compatibility.

In `package.json`, the dependencies section may contain the following entry:

```
"dependencies": {
"lodash-old": "npm:lodash@3.10.1"
}
```

In `yarn.lock` :

```
"lodash-old@npm:lodash@3.10.1":
```

The Johnny console agent handles this entry correctly, recognizing that **lodash-old** is an alias for **lodash** version 3.10.1. The original package is taken into account during dependency analysis, preventing errors related to non-existent names.

#Support for Yarn's selective dependency resolution mechanism

Yarn supports selective version resolution via the `resolutions` field in `package.json`, allowing you to specify specific versions of dependencies without editing `yarn.lock`.

This mechanism is useful if you need to update a sub-dependency that is not updated frequently, fix a vulnerability in a transitive dependency, or lock a version due to a problematic update.

CodeScoring supports handling this mechanism in the Johnny console agent. Here are some scenarios for its operation:

#Replacing a package

To replace a package via the resolutions mechanism, the following entry is added to `package.json`. In this example, the **parcel/watcher** package is replaced with the **favware/skip-dependency** package.

```
"resolutions": {
"@parcel/watcher": "npm:@favware/skip-dependency@latest"
}
```

The corresponding entry in the `yarn.lock` file for this package will be as follows:

```
dependencies:
"@parcel/watcher": "npm:2.1.0"
```

When installing, the build uses the **favware/skip-dependency** package version 1.2.2. The console agent correctly identifies this mechanism and analyzes the final package.

```
"@parcel/watcher@npm:@favware/skip-dependency@latest":
version: 1.2.2
resolution: "@favware/skip-dependency@npm:1.2.2"
```

#Fixing the version of a transitive dependency

To fix the version via the resolutions mechanism, the following entry is added to `package.json`. In this example, the version of the **http-signature** package is fixed to **1.3.4**.

```
"resolutions": {
"http-signature": "1.3.4"
}
```

The entries in the `yarn.lock` file corresponding to this package will be as follows:

```
dependencies:
http-signature "~1.2.0"
```

When installing, the build will use version **http-signature 1.3.4**. The console agent analyzes the final version of the package.

```
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"
```

#Fixating the version with multiple dependencies

To fixate the version with multiple dependencies, the following entry is added to `package.json` via the resolutions mechanism. In this example, the version of the **yaml** package is fixated to **2.2.2**.

```
"resolutions": {
"yaml": "2.2.2"
}
```

The entries in `yarn.lock` corresponding to this package will be as follows:

```
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
```

When installing, the build will use version **2.2.2**. The console agent analyzes only the package version fixated in `resolutions`.

```
"yaml@npm:2.2.2":
version: 2.2.2
resolution: "yaml@npm:2.2.2"
```

#Bun

#Creating a `bun.lock` file

```
bun install
```

If the project already uses the binary lockfile format `bun.lockb`, use the following command to generate `bun.lock`:

```
bun install --save-text-lockfile --frozen-lockfile --lockfile-only
```

#Working with dependencies in .NET

#NuGet

#Creating a `packages.lock.json` file

1. Enable lock-file support (for .NET 5 and above):

   ```
   dotnet nuget locals all --clear
   ```

2. Install dependencies:

   ```
   dotnet restore --use-lock-file
   ```

#Creating a `paket.lock` file

1. Create a lock-file:

   ```
   paket install
   ```

#Working with an `sln` manifest

In case an *.sln manifest is detected while scanning a directory the list of target manifests will be composed of the components described in it. The components which are present in the solution directory but not included will be ignored.

#Support for `Directory.Packages.props`

The agent automatically detects the `Directory.Packages.props` file when scanning `.csproj` projects and uses the package versions defined in it. The

file is located by walking up the directory tree from the project location to the scan root.

#General Information

- Within the .NET ecosystem, the main manifest is `*.csproj`, with `packages.lock.json` serving as the lock file;
- The `deps.json` file is considered a separate lock file that is automatically generated and is not linked to any other manifests. It specifies dependencies required for the target runtime.

#Working with dependencies in PHP

#Composer

#Creating a `composer.lock` file

1. Initialize the project:

   ```
   composer init
   ```

2. Install dependencies:

   ```
   composer install
   ```

   or create a lock-file directly:

   ```
   composer update
   ```

#Working with dependencies in Python

#pip

#Creating `requirements.txt`

1. Install dependencies and save them to a lock file:

   ```
   pip freeze > requirements.txt
   ```

#pipenv

#Creating `Pipfile.lock`

1. Install pipenv:

```
pip install pipenv
```

2. Create `Pipfile.lock`:

```
pipenv install
```

#poetry

#Creating `poetry.lock`

If `poetry.lock` does not already exist, Poetry will create it automatically when installing dependencies. If the file already exists, it will be updated. To do this, run the command:

```bash
poetry lock
```

This command will update the dependencies specified in `pyproject.toml` and create or update the `poetry.lock` file.

#pipdeptree

#Creating the `pipdeptree.txt` file

When a `pipdeptree.txt` file is detected, the agent will analyze its contents as the output of the pipdeptree utility in the standard dependency tree format. To create the file, you can use the following commands:

```
pipdeptree > pipdeptree.txt
```

To filter the output by specific packages:

```
pipdeptree --packages "example1,example2" > pipdeptree.txt
```

Other manifests in results

In order to prevent the dependencies of the main project manifest (e.g., requirements.txt) from appearing in the analysis results alongside the pipdeptree analysis results, it is recommended to exclude this manifest from scanning:

```
    johnny scan python . \
      --ignore "requirements.txt"
```

#uv

#Creating `uv.lock`

If `uv.lock` does not already exist, uv will create it automatically when installing dependencies. If the file already exists, it will be updated. To do this, run the command:

```bash
uv lock
```

This command will update the dependencies specified in `pyproject.toml` and create or update the `uv.lock` file.

#UV workspaces mechanism Support

The UV workspaces mechanism allows centralized management of multiple packages.

In `pyproject.toml`, the following entry can be specified in the workspaces section:

```
[tool.uv.workspace]
members = [
    "packages/core",
    "packages/api"
]
```

In this case, the Johnny agent will process the root `pyproject.toml` and all `pyproject.toml` files of all packages from the workspace as a single entity.

#pdm

#Creating `pdm.lock`

If `pdm.lock` does not already exist, pdm will create it automatically when installing dependencies. If the file already exists, it will be updated. To do this, run the command:

```bash
pdm lock
```

This command will update the dependencies specified in `pyproject.toml` and create or update the `pdm.lock` file.

#Creating the `pylock.toml` file

In addition to the standard format, pdm allows generating a lock file in the `pylock.toml` format. To lock dependencies in this format, run the following command before executing `pdm lock`:

```
pdm config lock.format pylock
```

#Working with dependencies in Ruby

#Bundler

#Creating a `Gemfile.lock` file

1. Initialize the project:

   ```
   bundle init
   ```

2. Install dependencies:

   ```
   bundle install
   ```

   or create a lock-file directly:

   ```
   bundle lock
   ```

#CodeScoring.Secrets

#General description

CodeScoring.Secrets is a module that searches for sensitive information in code (passwords, API keys, tokens), while using its own machine learning model to significantly reduce the number of false positives during scanning.

Secrets are searched using open analysis tools, currently the Gitleaks engine is used.

#Setting up engine configuration

1. To start working with the Secrets module, you must first create a VCS or CLI project in the `Settings -> Projects` section.

2. After creating the project, you must set the secrets engine configuration in the `Settings -> Secrets` section by opening the form using the **Setup new** button.

3. In the configuration form, specify a name, select an engine for searching for secrets in the code, and provide its standard configuration. The configuration will be passed to the selected engine during scanning.

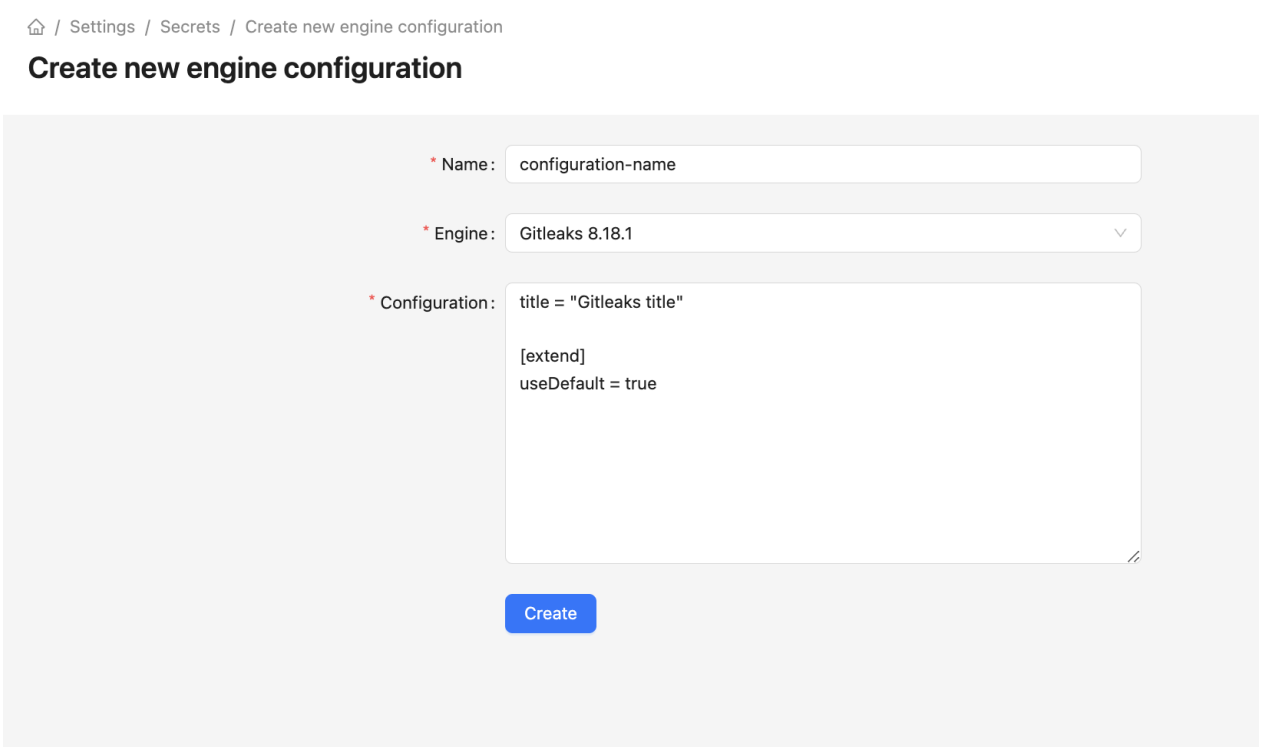
In the **Verification tool** field, you can select one of the supported engines:

- **Gitleaks 8.27.0**;
- **TruffleHog 3.93.8**.

Example configuration for Gitleaks:

```
title = "Gitleaks title"

[extend]
useDefault = true
```



More details on configuring the Gitleaks engine can be found in the tool documentation.

Example configuration for TruffleHog:

```
detectors:
  - name: generic-api-key
    keywords:
      - key
      - api
```

```
      - token
      - secret
      - client
      - passwd
      - password
      - auth
      - access
    regex:
      # generic-api-key regular expression from 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|;]|$)"
```

More details on configuring the TruffleHog engine can be found in the tool documentation.

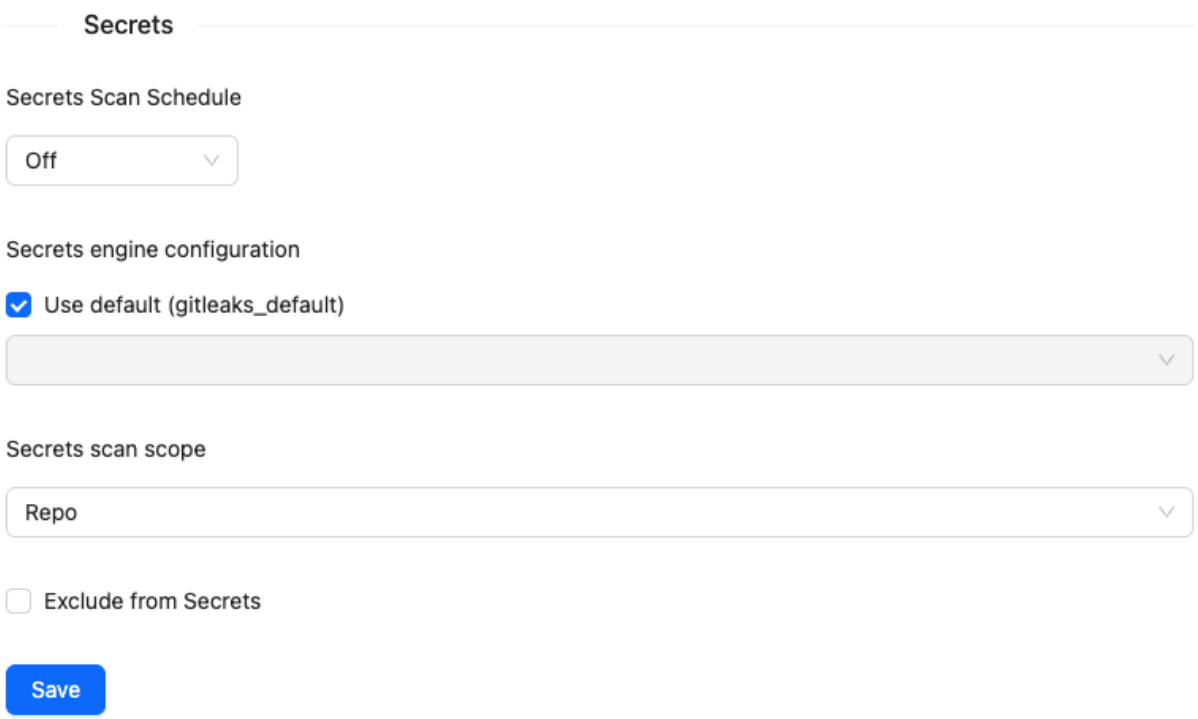
#Setting up default engine configuration

To set the default configuration you need to click the **Use by default** button in the configuration settings.

Editing the default configuration

You cannot set more than one default configuration, nor can you delete a configuration that is set as default.

To use the default configuration for a project check the **Use default** flag in the **Secrets** section of the project settings. The configuration currently used by default will be shown in parentheses.



Changing default configuration

When setting a new default configuration, all projects with the **Use default** flag selected will use the new configuration.

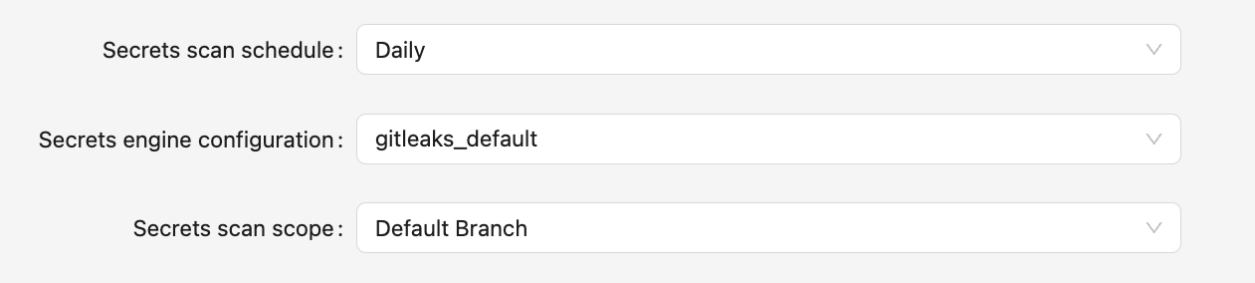
Configuration secrets engine for the new project

When creating a new project, a default engine configuration is automatically set. This configuration can be changed in the project settings in the **Secrets** section.

#Configuring VCS project for secret search

For the Secrets module to work within a project, you need to set the scanning parameters on the project settings page in the `Settings -> Projects` section:

- **Secrets scan schedule** - schedule for scanning for secrets (is set by time and days of week);
- **Secrets engine configuration** - configuration of the secrets engine;
- **Secrets scan scope** - scope of scanning:
  - **Repo** - for scanning all the branches within repository;
  - **Default branch** - for scanning the default branch in the project settings;
- **Exclude from Secrets** - exclude current project from SCA analysis;



#Launching secret search

#Search for secrets in a single project

To launch the search for secrets, go to the `Secrets` tab of the selected project and click the **Run secrets analysis** button, after which the search for secrets in this project will start.

Depending on the size of the project, the analysis can last from a few seconds to a few minutes.

#Scheduled search for secrets (VCS Projects)

In addition to manual runs, you can set up scheduled analysis of individual projects.

This is controlled on the project page in `Settings -> Projects`.

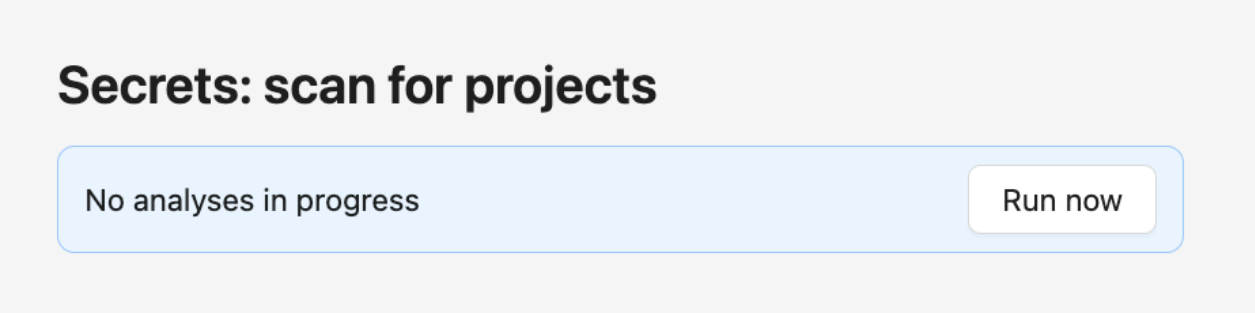
By default, the **Secrets Scan Schedule** parameter is set to **Off**. To activate scheduled analysis, you need to select **On** and select time and days of week.

Note: Selected time will be treated as UTC +3.

Note: Search won't start if no secrets engine configuration for project is selected.

#Search for secrets in all projects

To launch the search for secrets in all projects, go to the `Workmode` section and click the **Run now** button under the **Secrets: scan for projects** heading.



#Scheduled analysis

In addition to manual launch, you can set up scheduled analysis of individual projects. This is managed on the project page in the `Settings -> Projects` section.

By default, the **Secrets Scan Schedule** parameter is set to **Off**. To activate scheduled analysis, select **On** and specify the time and days of the week.

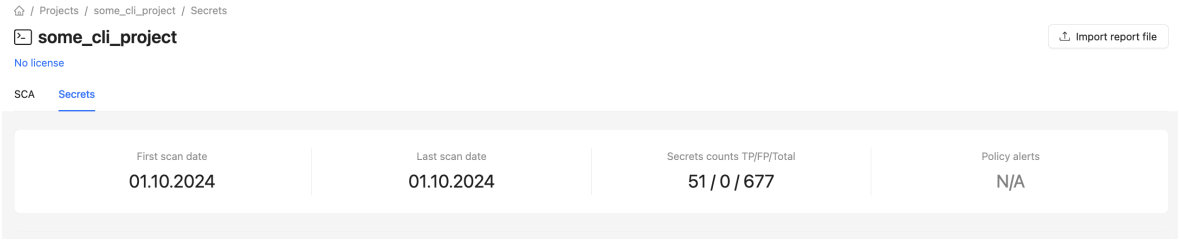
Note: The scan time will be calculated according to UTC +3.

#Searching secrets in CLI projects

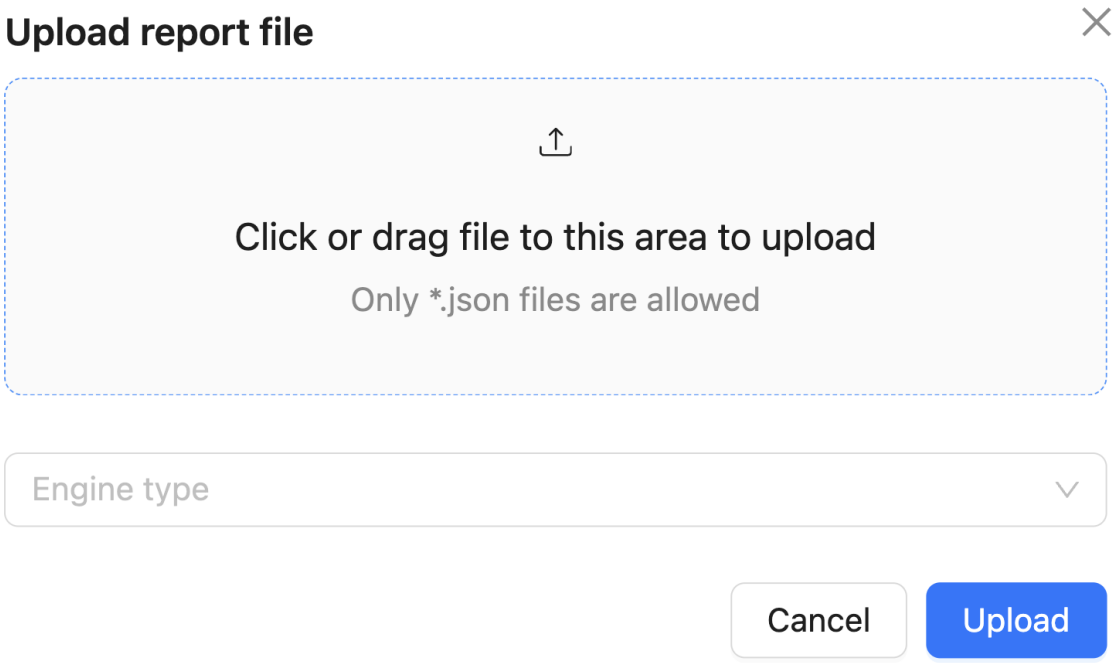
#Importing a report into a CLI project

You can import a separately created report in `json` format into a CLI project. You can read about using **Gitleaks** in the command line in the tool documentation.

1. After receiving a report from the tool, you need to go to the Secrets tab of the selected project.



2. Click the **Import report file** button.



3. Select the report file and specify the tool it was created with, then click the **Upload** button.

#Scanning CLI projects

To scan secrets in CLI projects and integrate scans into the CI/CD pipeline, use the `secrets gitleaks dir` command in the Johnny console agent.

Important: The agent only works with Gitleaks version 8.19.0 and above.

You can read more about using the command in the agent's documentation.

#Working with findings

#Viewing secrets in a separate project

To view data on secrets in the context of a project, go to the `Projects -> Project -> Secrets` page.

The page displays the following summary information:

- Date of first scan;
- Date of last scan;
- Number of found secrets by category (true positive, false positive, all);

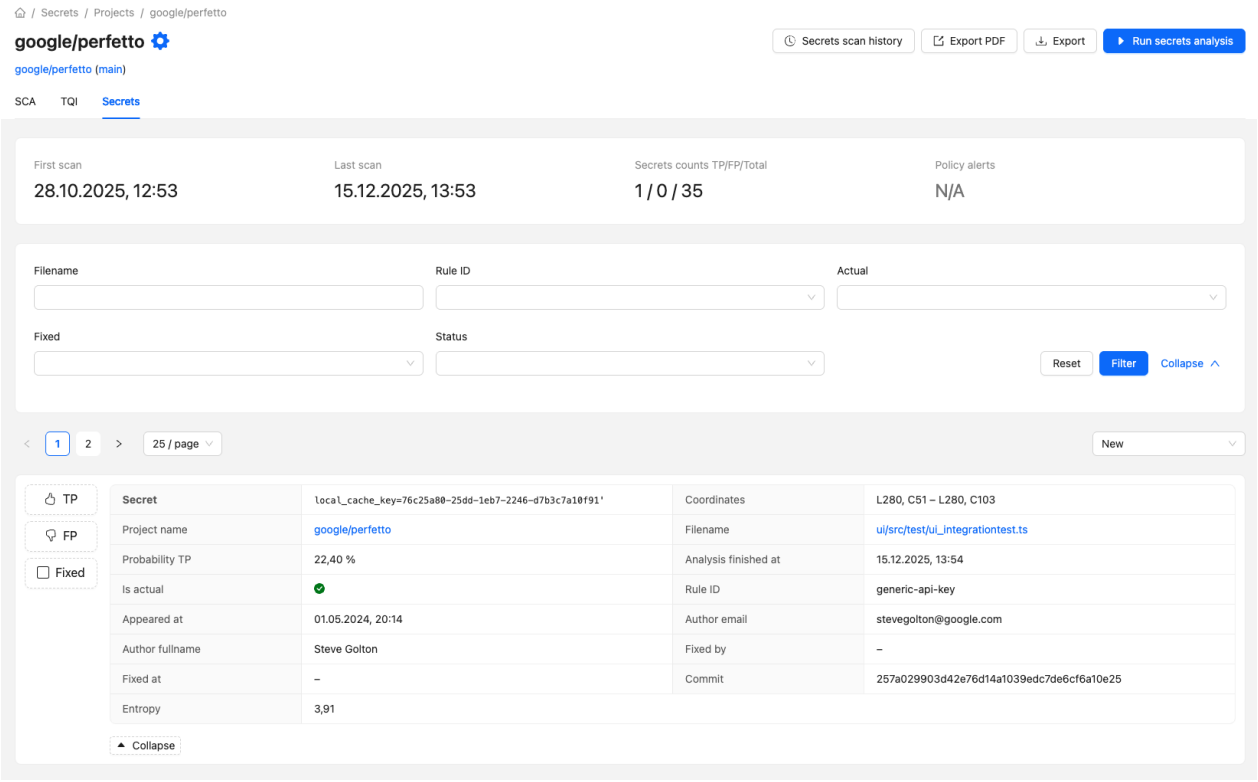
The table with found secrets has the following fields:

- **Secret** – secret content;
- **Project name** – name of the project in which the secret was found;
- **Filename** – name of the file in which the secret was found;
- **Coordinates** - line and column of the beginning and end of the secret in the file;

- **Probability TP** – probability of a true find;
- **Analysis finished at** – date and time of scan completion;
- **Is actual** – whether the secret was found during the last scan;
- **Rule ID** – secret search rule identifier within the used configuration engine;
- **Appeared at** – date and time the secret was added to the code;
- **Author email** – email of the author responsible for adding the secret;
- **Author fullname** – name of the author responsible for adding the secret;
- **Fixed by** – name of the user who marked the secret as fixed;
- **Fixed at** – date of fix;
- **Commit** – hash of the commit in which the secret was added;
- **Entropy** - entropy of the found secret.

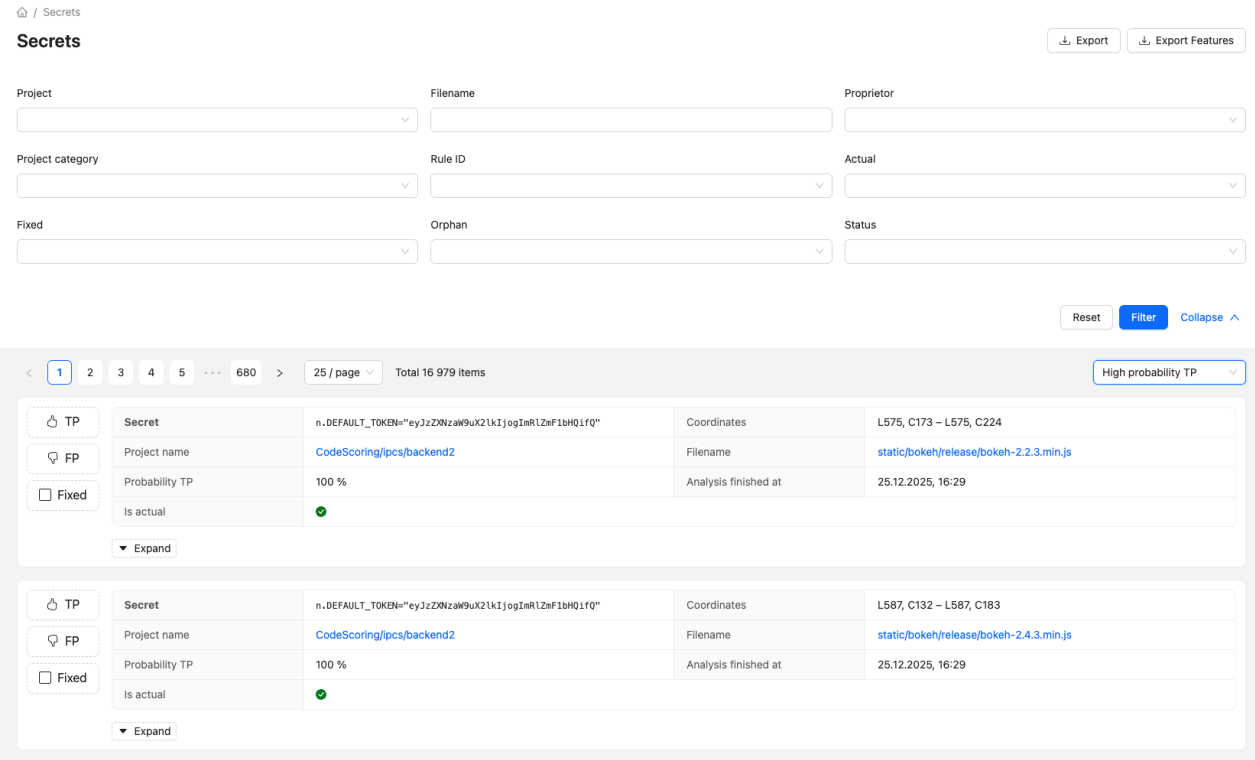
Entropy

This parameter is the Shannon entropy and can be used in rules as a threshold value.



#Viewing secrets

To study the found secrets for all projects, go to the Secrets section in the system menu.



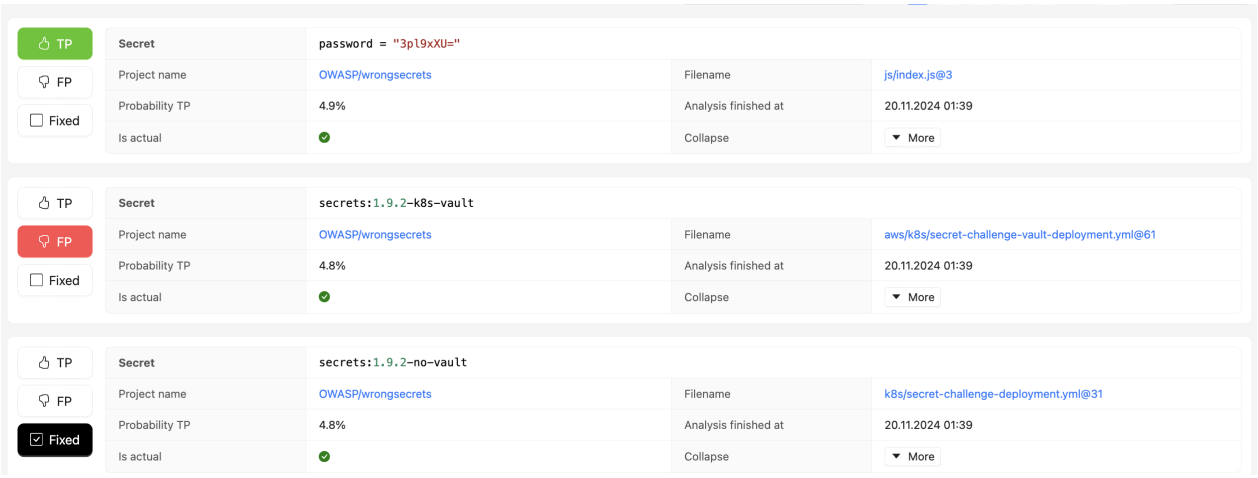
The table with secrets can be filtered by the following criteria:

- **Project** – name of the project where the secret was found;
- **Filename** – name of the file where the secret was found;
- **Proprietor** – division of the organization to which the project with the found secret belongs;
- **Category** – category of the project with the found secret;
- **Rule ID** – identifier of the secret search rule within the used configuration engine;
- **Actual** – secret found during the last scan;
- **Fixed** – secret fixed;
- **Orphan** – secret has no project;
- **Status** - finding status (true positive, false positive, no status).

#Markup of true and false positives.

Each secret can be marked as true positive, false positive or fixed. To do this, use the **TP**, **FP** and **Fixed** buttons in the table of found secrets.

Manual markup will be used for additional training of the machine learning model.



#Unloading data on secrets

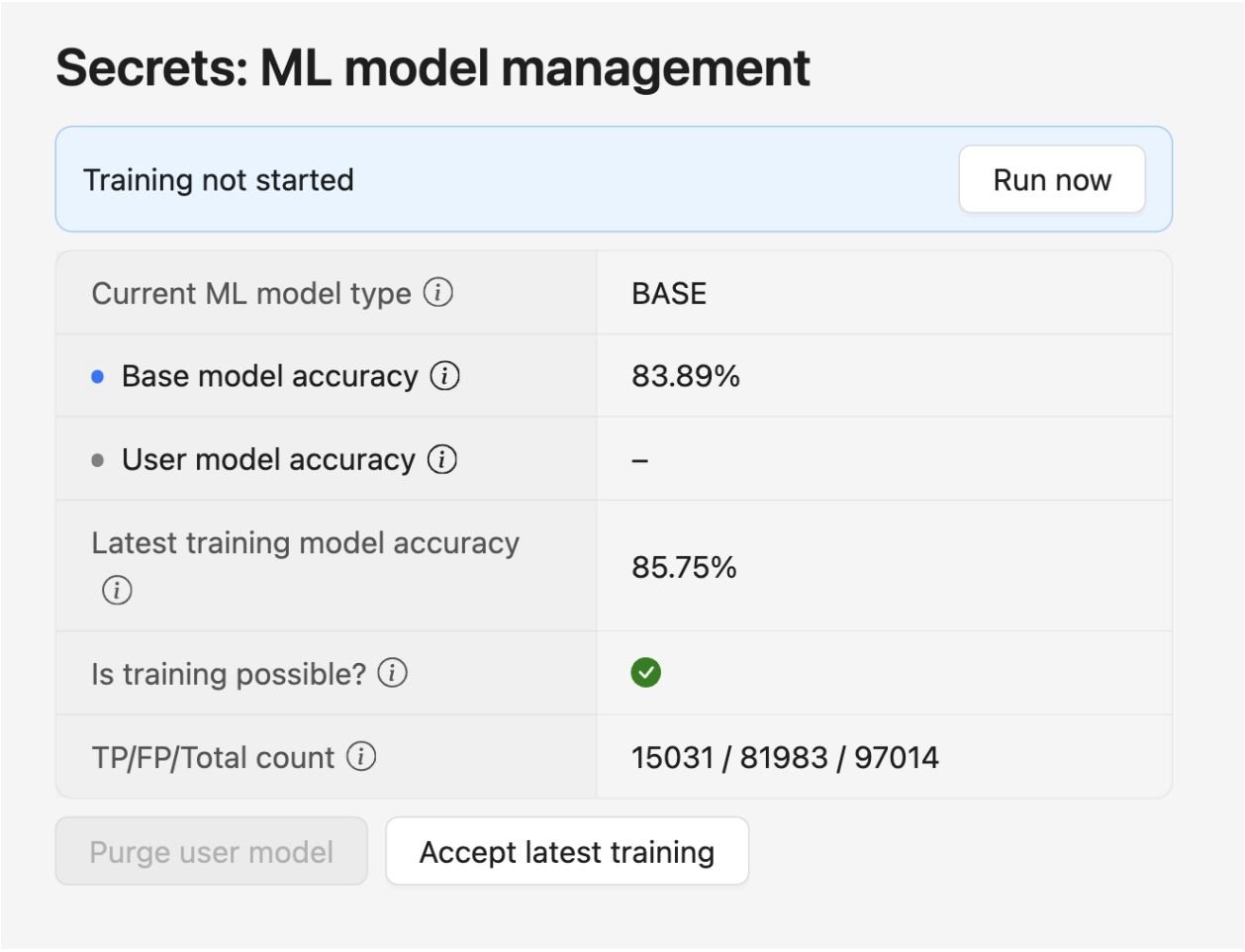
To get an export of data on found secrets, you can use the **Export** button in the upper right corner of the section.

#Machine learning model management

By default, CodeScoring uses its own machine learning model to reduce the number of false positives when searching for secrets. By manually marking the found secrets, you can further train the model and improve the search results on your own source code.

To further train the model, go to the `Settings -> Workmode` section and click the **Run now** button in the **Secrets ML model management** section. To activate the ability to further train the model, it is necessary to mark up at least 1000 found secrets as true positives or false positives.

After further training, you can compare the results of the secret search and, based on them, either accept the user model (**Accept training**) or return to the base model (**Purge user model**).



The control section displays information about the current state of the model to the user:

- **Current ML Model Type** – type of the model used (base or user);
- **Base model accuracy** – search accuracy based on marked findings. True positive findings are taken as 1, false positive findings as 0. The final accuracy is the average value of all results, presented as a percentage;
- **User model accuracy** – search accuracy using the user model;
- **Latest training model accuracy** – search accuracy using the latest additional training;
- **Is training possible** – possibility of additional training of the model based on the current labeling;
- **TP/FP/Total count** – true positives, false positives and all findings.

Important: if additional training of the model is impossible, this means that the markup is not complete enough. In this case, it is necessary to designate a larger number of findings as true positives or false positives.

#CodeScoring.TQI

#General description

CodeScoring.TQI is a module for analyzing the quality of an organization's proprietary source code, determining the main parameters of technical debt.

The main functional capabilities of the module allow to:

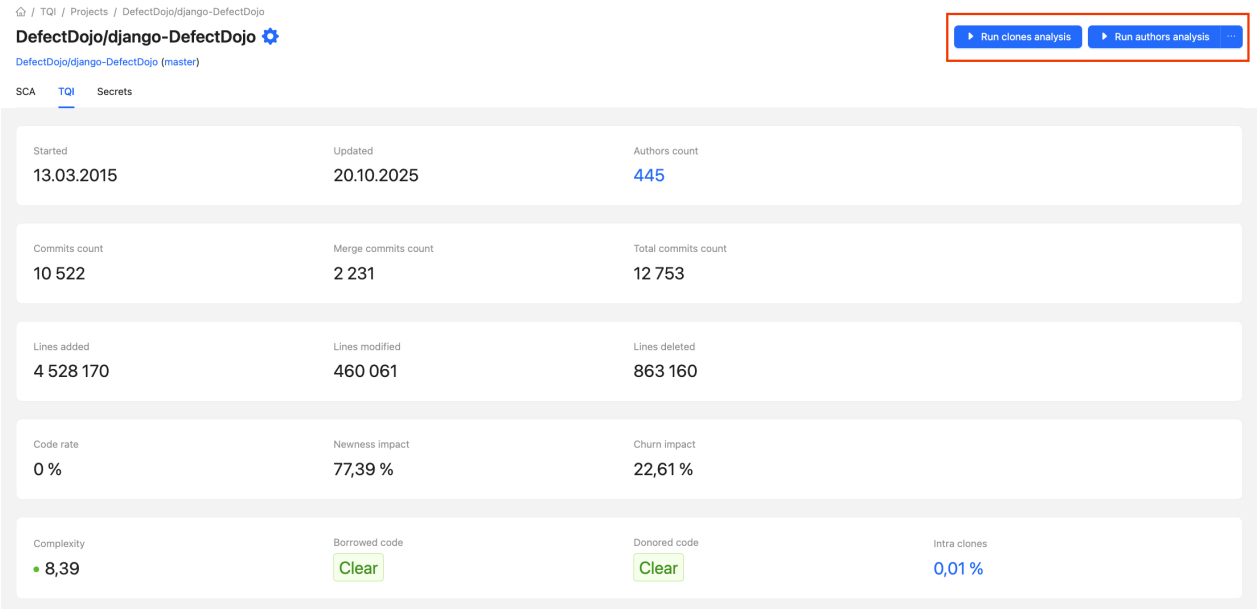
- View project dynamics with calculation of cyclomatic complexity of the code;
- Build author profiles with comparison of similarity;
- Track code clones within and between projects.

#Launching TQI analysis

#Analysis of a single project

After a successful cloning of a project from VCS on its page in the `TQI -> Projects` section, it becomes possible to launch two types of analysis:

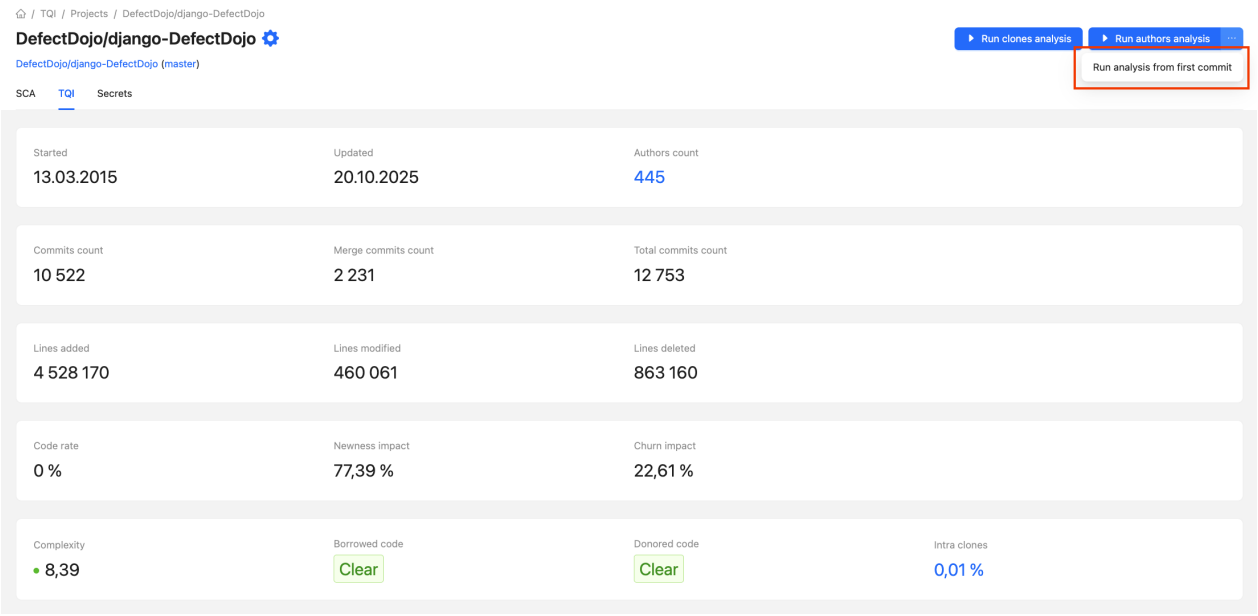
1. Duplicate analysis
2. Author analysis



After launching the analysis, the process runs in the background, and upon its completion, the results become available on the project page.

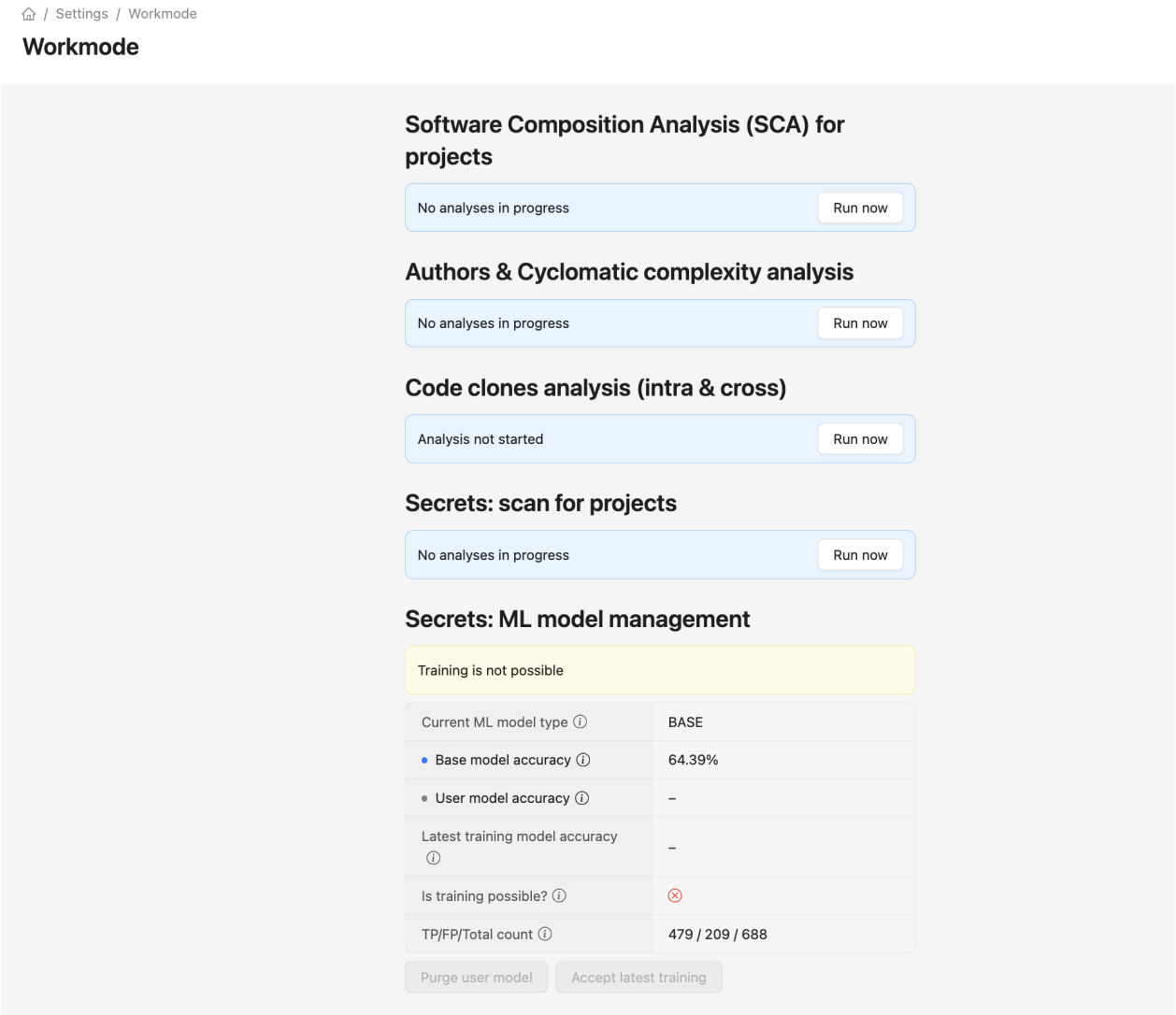
#Analysis from first commit

It's also possible to run authors analysis from first commit. Analysis will process all commits in project repository and update data in CodeScoring.



#Analysis of all projects

To launch the analysis for all VCS projects in the system, go to the `Settings -> Work mode` section and launch one of the two types of analysis.



#Scheduled analysis

In addition to manual launch, you can set up scheduled analysis of individual projects. This is managed on the project page in the `Settings -> Projects` section.

By default, the schedule is not configured and turned off.To activate scheduled analysis, you need to enable it and specify the time and days of the week. Scheduled analysis of duplicates and authors can be configured independently.

Note: The scan time will be calculated according to UTC +3.

#Viewing TQI analysis results

#Project page

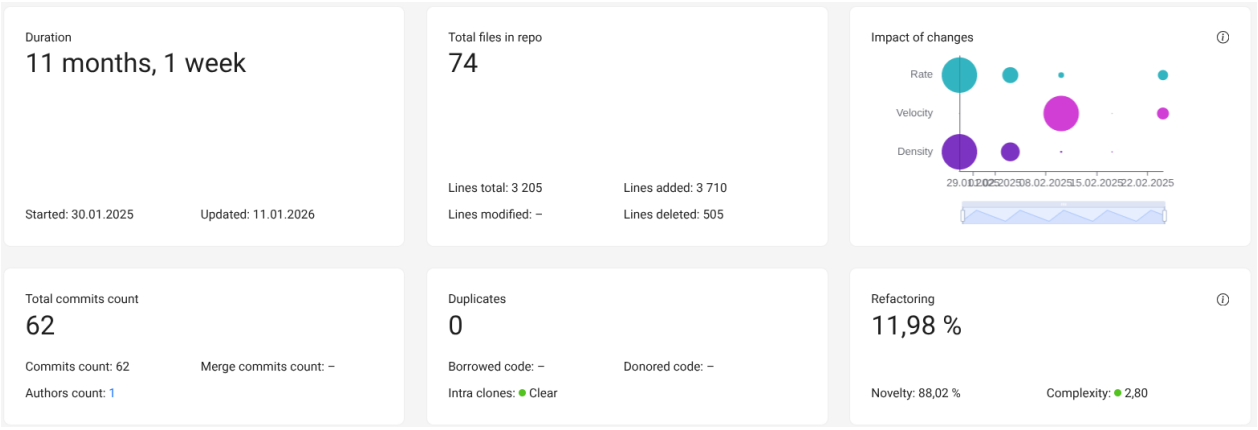
After the analysis is complete, a detailed report containing key metrics, author information, change dynamics, and a list of commits becomes available on the project page in the `TQI -> Projects` section.

#General project statistics

The beginning of the report records key indicators based on the analysis results:

- **Started** – records the moment of the first commit in the repository;
- **Duration** - number of months from the time of the first commit to the time of the last change;
- **Updated** – time of the last committed change;
- **Authors count** – number of developers who made changes to the code base;
- **Commits count** - number of commits in the repository;
- **Merge commits count** number of merge commits in the repository;
- **Total commits count** – total number of commits (commits + merge commit) in the repository;
- **Lines added** – number of added lines of code;
- **Lines modified** - number of modified lines of code;
- **Lines deleted** – number of removed lines of code;
- **Code rate** - mean value of commit changes relative to total LOC in the repository;
- **Novelty** - added lines share relative to total LOC in the repository;
- **Refactoring** - modified and removed lines share relative to total LOC in the repository;
- **Complexity** – a code complexity indicator based on the number of branches in the program logic;
- **Borrowed code** – identifies sections of code that were copied from other projects within the organization;
- **Donored code** – identifies code fragments transferred from other projects within the organization;

- **Intra clones** – records duplicate sections of code within the project.



#Authors

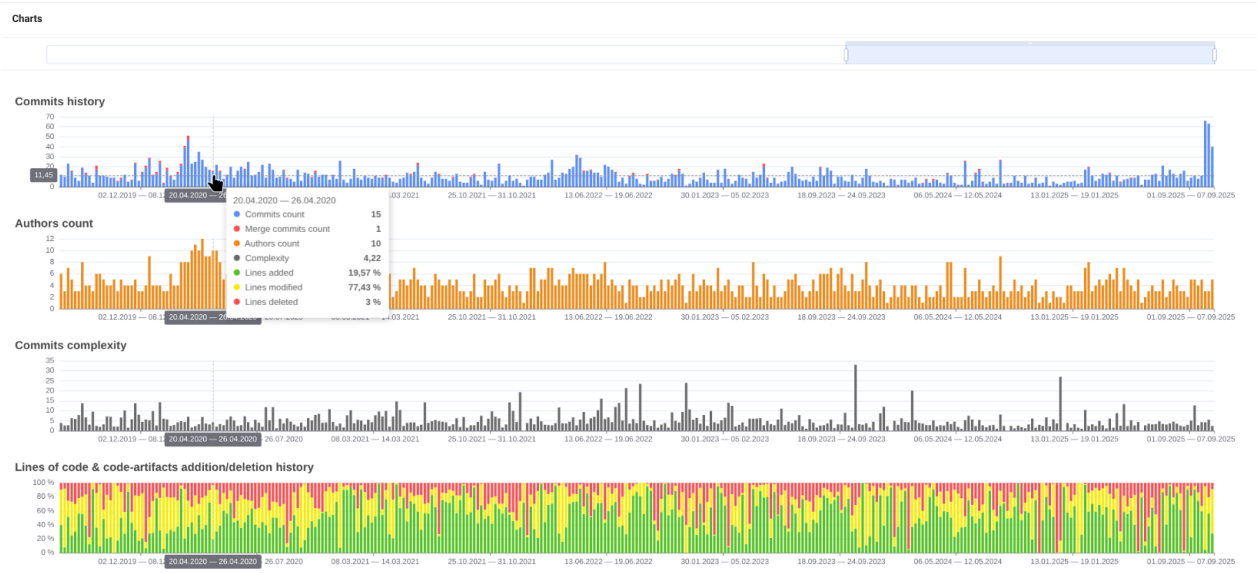
The list of authors can be viewed as a table. It is possible to change the display of columns and export the table in CSV format.

- **Author** – author's name and email;
- **Start date** – date of the author's first commit;
- **Last seen** – date of the author's last commit in the project;
- **Activity, months** – number of months during which the author actively committed changes;
- **Total commits** – total number of commits made by the author in the project. Details are available on value hover;
- **Complexity** – average cyclomatic complexity of the author's commits in the project;
- **Clones made** – number of borrowed code fragments made by the author;
- **Technologies** – programming languages the author works with (determined by his commits).

#Project dynamics

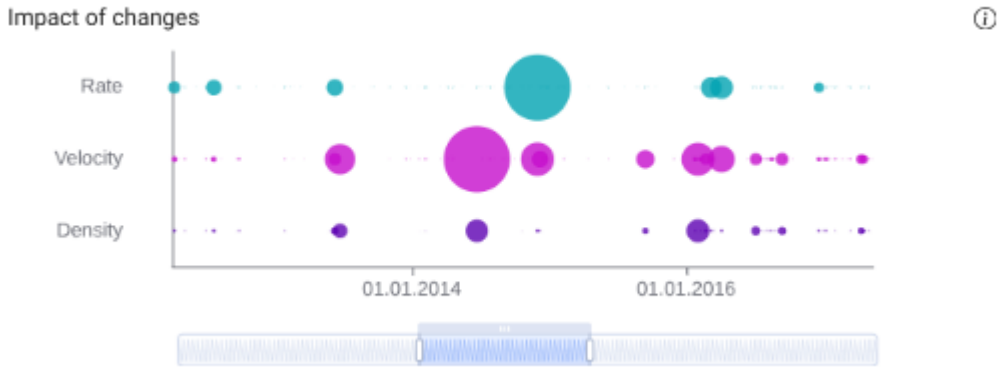
The history of the project can be tracked using graphs showing the dynamics of the project according to the following parameters:

- Lines of code & code artifacts addition/deletion history;
- Commits history;
- Authors count;
- Commits complexity.



In addition, the impact of changes on the project can be assessed using the following parameters::

- Rate;
- Velocity;
- Density.



The time interval on the graphs can be changed using the slider, choosing the period of interest for analysis.

#List of commits with cyclomatic complexity calculation

For each commit, cyclomatic complexity is calculated, and its context is shown:

- **Hash** – unique commit identifier with a link to the version control system;
- **Commit message** – a short description of the changes made, specified by the author when committing;
- **Commit date** – date and time when the change was made;
- **Lines added** – number of lines of code added in the commit;
- **Lines deleted** – number of lines removed in the commit;

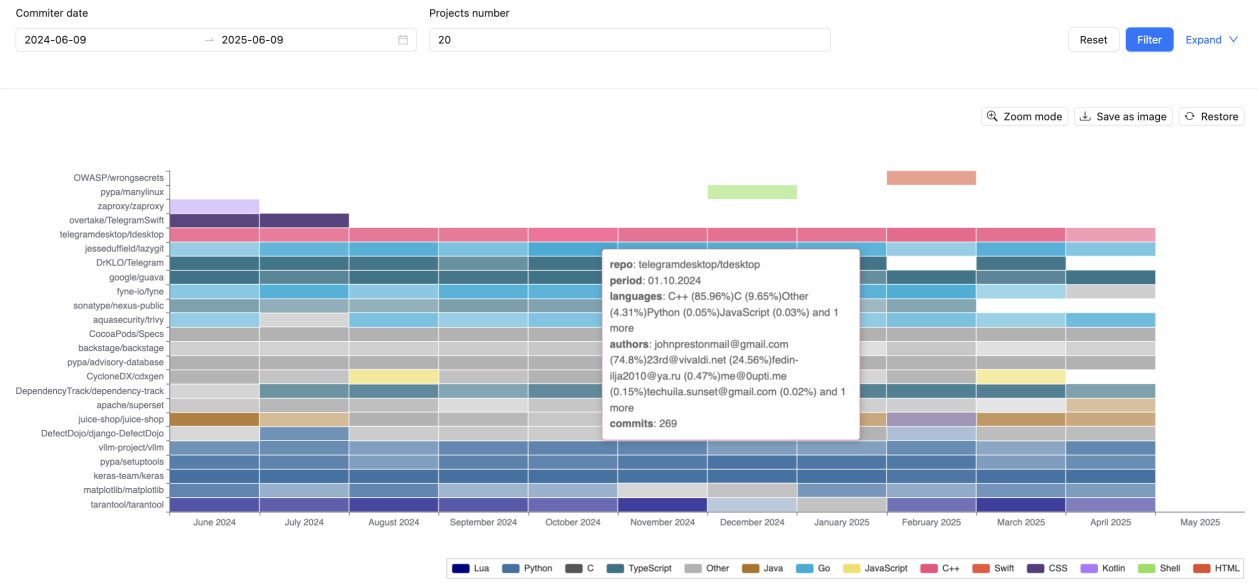
- **Complexity** – cyclomatic complexity value calculated based on the changes made;
- **Author** – name of the developer who made the commit.

#Results visualization

#Activity map

The activity map is available in the `TQI -> Projects` section on the **Activity map** tab. It displays all the contributions of authors for the selected time period for a set of projects, which can be filtered by the following parameters:

- **Commit Date** – period during which the commit was made in the version control system;
- **Number of Projects** – total number of projects displayed on the map;
- **Proprietor** – part of the organization that manages the project;
- **Project category** – category assigned within the CodeScoring system;
- **Technologies** – programming languages used in the creation of the project.

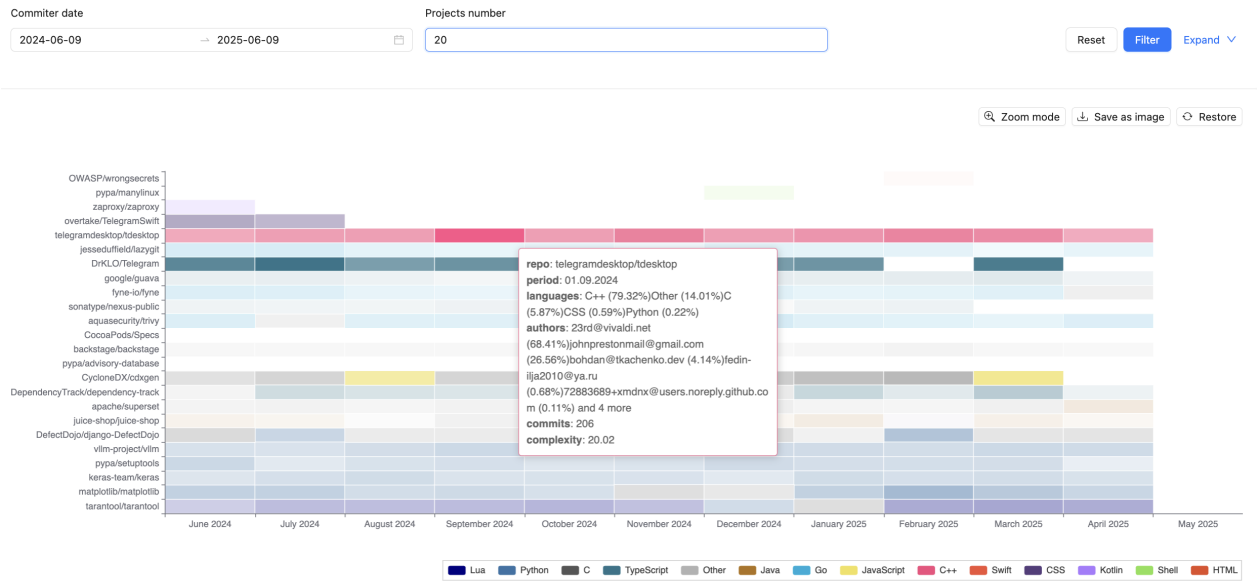


The map can also be saved as a PNG image.

#Complexity map

The complexity map is available in the `TQI -> Projects` section on the **Complexity map** tab. It displays the change in the complexity of a set of projects, which can be filtered by the following parameters:

- **Commit Date** – period during which the commit was made in the version control system;
- **Number of Projects** – total number of projects displayed on the map;
- **Proprietor** – part of the organization that manages the project;
- **Project Category** – category assigned within the CodeScoring system;
- **Technologies** – programming languages used within the project.



The map can also be saved as a PNG image.

#Tracking code clones

CodeScoring.TQI allows you to track code fragments that were copied from one project of the organization to another, or duplicated within a single project.

#Cross-project clones

The `TQI -> Code clones -> Cross project` section displays a list of projects between which copying was performed. Clicking on the number of clones will display a table with detailed information about the copying performed, indicating the following fields:

- Excerpts of copied code with a link to the commit in the version control system;
- Direction of copying;
- Commit date;
- Author;
- Number of copied lines;
- Clone occurrence level (low, medium, high);
- Technology.

#Intra-project clones

The `TQI -> Code clones -> Intra project` section displays a list of projects where code fragments were copied, as well as the percentage of duplicates from the total project code and the occurrence level. Clicking on

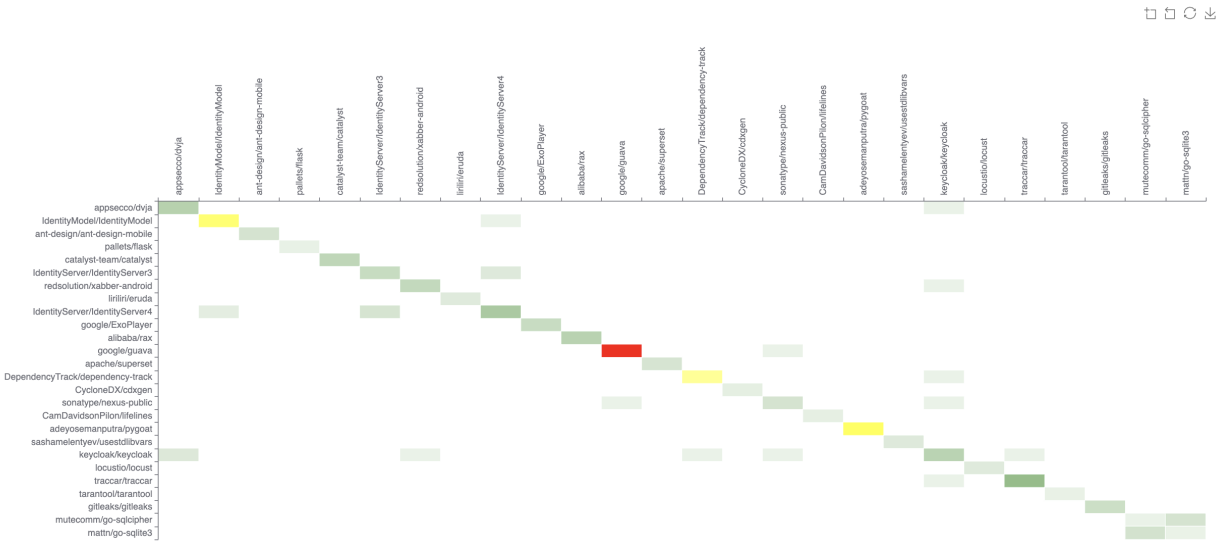
the number of duplicates will display a table with detailed information about the copying performed, indicating the following fields:

- Excerpts of copied code with a link to the commit in the version control system;
- Copy direction;
- Commit date;
- Author;
- Number of copied lines;
- Clone occurrence level (low, medium, high);
- Technology.

#Clones map

The `TQI -> Code clones -> Clones map` section displays a visualization of borrowings between projects. The clones map can be filtered by the following fields:

- **Proprietor** – part of the organization to which the project belongs;
- **Project category** – project category within the CodeScoring system.



Clicking on the intersection between two projects will transfer user to a page with a detailed description of the clones. The map can also be saved as a PNG image.

#Calculating technical debt metrics

CodeScoring.TQI tracks several technical debt metrics. The main ones are **cyclomatic complexity**, **clone occurrence**, **code rate**.

#Calculating cyclomatic complexity

Cyclomatic complexity is an indicator that reflects the number of independent paths in the code. The higher this value, the more difficult it is to maintain and test the code.

Calculation is based on the formula:

```
M = E - N + 2P
```

Where:

- **M** – cyclomatic complexity;
- **E** – number of edges (transitions between operators);
- **N** – number of nodes (operators, conditions);
- **P** – number of connectivity components (usually 1).

Example:

```
def is_even(x):
print("Even" if x % 2 == 0 else "Odd")
```

Cyclomatic complexity:

- **N = 3** (input, `if-else`, `print`).
- **E = 3** (input -> `if-else`, `if-else` -> `print`, `print` -> output).
- **P = 1** (function).

So cyclomatic complexity is M = E - N + 2P = 3 - 3 + 2 = **2**

Complexity levels:

- **Low**: < 10 (simple code, easy to read and maintain);
- **Medium**: 10–20 (moderately complex code, requires attention when changing);
- **High**: > 20 (complex code, possible problems with testing and support).

#Calculating the percentage of intra-project clones

This indicator reflects what part of the project's code is duplicated. It is calculated using the following formula:

```
Percentage of duplicates = (number of duplicated lines of
code / total number of lines of code) * 100%
```

The higher this percentage, the more code can be optimized with refactoring.

#Calculating the occurrence of clones

This indicator evaluates the extent to which duplicated code is distributed throughout the project.

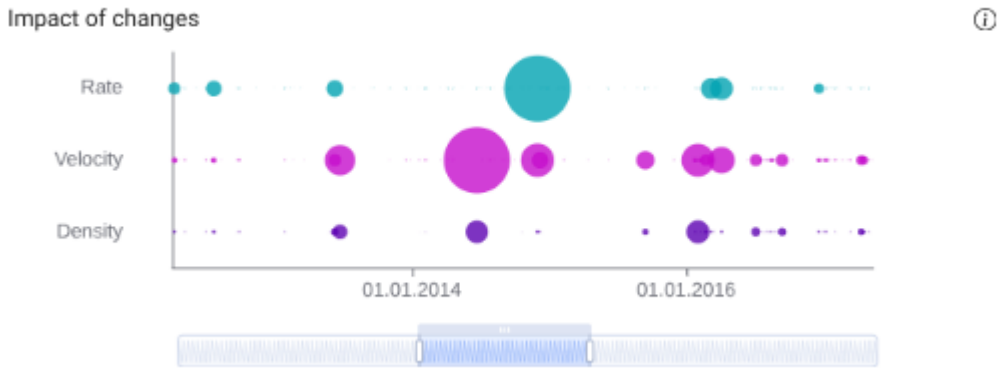
Categories:

- **Low level** - if there are less than 50 duplicated lines;
- **Medium level** - if there are from 50 to 300 duplicated lines;
- **High level** - if there are more than 300 duplicated lines.

#Impact of changes

You can evaluate the impact of changes on the project using the following parameters on the general graph:

- Rate;
- Velocity;
- Density.



The calculation is performed for the period. The minimum period is one week.

The sizes of the elements in the sample are normalized. The sample is normalized using the formula:

```
X’ = (X–Xmin)/(Xmax–Xmin)
```

Where:

- **Xmin** is the minimum value in the sample;
- **Xmax** is the maximum value in the sample;
- **X’** is the normalized value.

#Calculating the code rate

Code rate shows the amount of code change relative to the total number of lines of code. A high rate of code change indicates frequent changes in requirements or an unstable rhythm of the development team.

Calculation is based on the formula:

```
R = 100 * L / T
```

Where:

- **R** - code rate;
- **L** – number of lines of code entered (added, deleted, modified) during the billing period;
- **T** - total number of lines of code in the project repository at the beginning of the billing period.

#Calculating the velocity rate

Velocity rate shows the amount of changes in the code relative to the number of commits. At high values, the workload increases during code review, testing, and maintenance.

Calculation is based on the formula:

```
V = T / C
```

Where:

- **D** – density of changes;
- **L** – number of lines of added, deleted, modified code during the reporting period;
- **F** – number of modified files.

#Calculation of the density rate

Density rate shows how many changes were made to the code relative to the number of modified files. The high density value indicates the modification of a large number of modules. The burden of code review, testing and maintenance increases.

Calculation is based on the formula:

```
D = T / F
```

Where:

- **D** – density of changes;
- **L** – number of lines of code entered (added, deleted, modified) during the billing period;
- **F** – number of modified files.

#Connect a project and run the first SCA analysis

#Context

The first practical step in CodeScoring.SCA is to connect a repository as a VCS project and wait for the first SCA analysis. This scenario uses GitLab as an example, but the overall flow is the same for other supported version control systems: create the connection, create the project, and wait for the first analysis.

#What you will get

After completing this scenario, you will have a connected project in CodeScoring with the first completed SCA analysis. The project page will show the initial results, and scan history will contain metadata for the run if you need to verify branch and commit details.

#Requirements

Before you start, make sure you have:

- access to the GitLab repository;
- a GitLab account that can create a `Personal Access Token`;
- access to CodeScoring with `VCS: adding repositories` and `Projects: creating projects` permissions;
- a CodeScoring license with the **SCA** module enabled.

#Steps

#Step 1. Create a GitLab access token

CodeScoring uses this token to read the repository content during analysis.

1. Sign in to your GitLab account.
2. Open `Edit profile`.
3. In the left menu, open **Access Tokens**.
4. Set a token name, for example `codescoring-demo`.
5. In the `scopes` section, enable `read_api` and `read_repository`.
6. Click **Create personal access token**.
7. Copy the generated token and store it securely.

The token is now ready to be used in the GitLab connection settings in CodeScoring.

#Step 2. Add a GitLab connection in CodeScoring

At this step, GitLab becomes available to the platform as a source of code for future analysis runs.

1. In CodeScoring, go to `Settings -> VCS`.
2. Click **Add**.
3. Fill out the connection form:
    - **Name** — a clear connection name, for example `GitLab main`;
    - **Connection type** — HTTPS;
    - **Type** — `Gitlab`;
    - **Address** — the GitLab address, for example `https://gitlab.com`;
    - **Access token** — the token created in the previous step.
4. Click **Check connection**.
5. If the check is successful, click **Add**.

After a successful connection check, the GitLab connection can be selected when creating projects.

#Step 3. Create a VCS project

Now you can add the repository itself to CodeScoring and immediately trigger the first SCA run after cloning.

1. Go to `Settings -> Projects`.
2. Click **Create** and open the **VCS projects** tab.
3. Fill out the project form:
    - **Repository** — the GitLab repository URL;
    - **VCS** — the GitLab connection created in the previous step;
    - **Name** — the project name in CodeScoring.
4. Keep the **Run SCA after cloning** option enabled.
5. Click **Create**.

Once the project is saved, CodeScoring starts cloning the repository and then launches the first SCA analysis automatically.

#Step 4. Wait for the first analysis to finish

During the first run, the platform downloads the source code and builds the initial view of project dependencies and vulnerabilities.

1. Open the project page.
2. If needed, monitor progress in `Settings -> Audit log`.
3. Wait until the analysis is complete.

When the analysis finishes, the project page starts showing SCA results.

#Step 5. Check the analysis results

At this point, confirm that the project has really been analyzed and that the results are already usable.

1. Open the SCA tab on the project page.
2. Check that analysis results are already visible on the tab.
3. If needed, open **SCA scan history** to verify that the latest run has the `success` status.
4. Open the latest run by its start date and check that the results include:
    - the number of detected dependencies;
    - the number of detected vulnerabilities;
    - VCS metadata, including the branch and commit SHA.

At this stage, the project is connected to the platform and the first SCA results are ready for review.

#Result

The scenario is complete if you can:

- open `Settings -> VCS` and confirm the GitLab connection is present;
- open `Settings -> Projects` and confirm the project exists;
- open the project page and confirm that SCA results are already visible;
- if needed, open `SCA -> SCA scan history` and verify that the latest run finished with `success`.

After that, the project is ready for dependency review, vulnerability analysis, and policy setup.

#What next

After the first analysis, the next common steps are:

- review dependencies;
- inspect detected vulnerabilities;
- configure security policies and recurrent analysis.

#Configure a basic set of security policies

#Context

After the first analysis in CodeScoring.SCA, you usually get a lot of information about dependencies and vulnerabilities. A basic set of rules helps separate the most important cases from the general flow of results and makes it easier to understand what deserves attention first.

For VCS projects at the `source` stage, a practical starting point is to create three separate policies:

- one for vulnerabilities that already have a public exploit and a fix;

- one for dependencies with critical vulnerabilities;
- one for components that were published less than a month ago.

These rules help set priorities faster without overwhelming the team with noisy alerts or introducing hard blocks too early.

Why this starter set works

OpenSSF recommends prioritizing dependency risk instead of treating all findings as equal, and OWASP highlights the value of handling the most dangerous and actionable cases separately. CodeScoring research also shows that a useful starter set separates three different queues: vulnerabilities that are already exploited and already fixable, dependencies with critical vulnerabilities, and very young components that still need additional verification before wider use.

#What you will get

By the end of this scenario, CodeScoring will contain three active policies for VCS projects at the `source` stage:

- a policy for vulnerabilities with a public exploit and a fixed version at the `source` stage;
- a policy for dependencies with critical vulnerabilities at the `source` stage;
- a policy for very young components at the `source` stage.

After a repeated SCA run, you will be able to see which of these rules already produce useful alerts.

#Requirements

Before you start, make sure you have:

- access to CodeScoring with the `Administrator` or `Security Manager` role;
- at least one connected VCS project that can be rescanned with SCA;
- a clear decision on whether this starter set should be applied to all projects immediately or only to a pilot project first.

#Steps

#Step 1. Create a policy for vulnerabilities with an exploit and a fix

This rule helps surface not just any vulnerabilities, but the ones that are already practical to exploit and also have a clear remediation path.

1. Go to `Settings -> Policies`.
2. Click **Create**.
3. Fill in the policy context:
    - **Name** — for example, `Exploit and fix`;
    - **Stages** — `source`;

    - **Level** — choose the severity level that fits your process;
    - **Is Active** — enable it;
    - **Blocker** — keep it disabled.
4. If you want to roll the set out gradually, specify a pilot project in **Projects**. If the rule should apply to all active projects, leave **Proprietors**, **Groups**, and **Projects** empty.
5. In the default top-level conditions group with the **AND** expression, add:
    - **Vulnerability has exploit**;
    - **Vulnerability has fixed version**.
6. Click **Create**.

Why Blocker should stay off at first

For a starter set it is more useful to see what the rule really catches and how noisy it is before turning it into an enforcement point. That makes it easier to shape the process without stopping scans or everyday work too early.

After that, the platform will have a rule that creates alerts for the most actionable vulnerability cases in VCS projects.

#Step 2. Create a policy for dependencies with critical vulnerabilities

Creating a separate rule for critical vulnerabilities helps move the most severe cases into their own review queue. For a starter set, this makes it easier to track them separately and avoid mixing them with lower-risk findings.

1. Open the policy created in the previous step.
2. Click **Create a copy**.
3. Change the main fields:
    - **Name** — for example, `Critical vulnerability in a dependency`;
    - **Stages** — keep `source`;
    - **Is Active** — keep it enabled;
    - **Blocker** — keep it disabled.
4. Remove the previous conditions.
5. In the default top-level conditions group, select the **OR** expression and add:
    - **CVSS2 Severity** = `critical`;
    - **CVSS3 Severity** = `critical`;
    - **CVSS4 Severity** = `critical`.
6. Click **Create**.

At this point, the starter set also includes a separate rule for dependencies that have at least one critical severity signal in one of the CVSS versions.

#Step 3. Create an informational policy for very young components

This rule helps isolate dependencies that were published very recently. For a starter set, this is more useful than making the policy blocking right away, because the team gets a separate signal for additional verification of new

packages and versions without mixing it with vulnerability handling or routine upgrades.

Why this rule should stay informational at first

For a starter set it is more useful to observe these components as a separate signal and understand how often they appear in your projects. Turning this rule into a blocker too early can interrupt routine workflows because of normal library updates before the team has agreed on how new versions should be reviewed.

1. In `Settings -> Policies`, click **Create** again.
2. Fill in the policy context:
   - **Name** — for example, `Component younger than 30 days`;
   - **Stages** — `source`;
   - **Is Active** — enable it;
   - **Blocker** — keep it disabled.
3. If you want a gradual rollout, specify a pilot project in **Projects** if needed.
4. In the default top-level conditions group, select the **OR** expression and add:
   - **Dependency age (days)** < `30`;
   - optionally, a condition for dependencies where age information is missing.
5. Click **Create**.

If you want to refine the threshold later or extend the rule, the full list of supported criteria is described in policy setup.

What to keep in mind about dependency age

The platform determines the dependency publication date for supported ecosystems. Before rolling this rule out to all projects, it is better to validate it on a pilot project first and confirm that the criterion behaves as expected for your packages and sources.

After that, very recently published components start appearing in a separate alert stream, making it easier to review them before they spread widely across projects.

#Step 4. Rerun the analysis and check the first alerts

Policies start working during analysis, so after configuring them it is worth verifying immediately that the `source` rules already participate in the process.

1. Open one of the projects to which the new policies should apply.
2. Click **Run SCA** on the project page.
3. Wait for the analysis to finish.
4. Open the `Policy alerts` section.
5. Check whether new alerts appear for:
   - vulnerabilities with an exploit and a fix;
   - direct dependencies with critical vulnerabilities;

    - very young components.

At this point, the starter set is already in use: all three rules produce alerts after analysis and separate the most important cases into different review tracks.

#Result

The scenario is complete if:

- three active policies from this set are visible in `Settings -> Policies`;
- all three policies use the `source` stage and contain the expected criteria;
- the young-component policy uses the **Dependency age (days) < 30** condition and remains non-blocking;
- after a repeated analysis, the first alerts for each policy type can be reviewed in `Policy alerts`.

After that, the platform already has a minimal set of guardrails that helps stop the riskiest components and focus the team on findings that matter first.

#What next

- review triggered alerts;
- configure exceptions for acceptable cases;
- connect policy notifications;

#Automatically check component security in a build and generate a report

#Context

If composition analysis runs only manually, vulnerabilities and policy violations are easy to notice too late, after a failed build or even after a release. Experienced teams embed this kind of check directly into the build pipeline so every run immediately shows the component set, detected vulnerabilities, and policy matches.

This is what the Johnny CLI agent is used for: it analyzes dependencies, saves results to CodeScoring, and saves an SBOM and reports for further work.

The scenario below uses GitLab CI as an example, but the same idea works for other pipelines where the Johnny agent can run as part of the build.

#What you will get

By the end of this scenario, the `.gitlab-ci.yml` file will contain a dedicated `sca` job that:

- runs the Johnny agent against the repository contents;
- saves the results to a CodeScoring project;

- produces `bom.json` and `report.sarif` as build artifacts.

#Requirements

Before you start, make sure you have:

- a GitLab repository with a working GitLab CI pipeline;
- GitLab CI variables `JOHNNY_API_URL` and `JOHNNY_API_TOKEN`;
- a CodeScoring license with the **SCA** module enabled;
- access to CodeScoring where a project for Johnny results can be created automatically with `--create-project` or an existing project can be used;
- a clear understanding of which policies should be applied to Johnny runs at the `build` stage.

#Steps

#Step 1. Prepare the agent binary in the build environment

Before adding the job to the pipeline, make sure the agent itself is available in the environment where the build runs.

1. Open the `[platform-url]/download/` page in your CodeScoring installation.
2. If needed, check the current agent version at `[platform-url]/download/johnny_version`.
3. Download the required executable file into the build environment, for example to `/usr/local/bin/johnny`.
4. Allow it to run:

```
chmod +x /usr/local/bin/johnny
```

If you need a more detailed GitLab CI-specific example, use the Johnny Agent GitLab CI guide.

After that, the pipeline can call `johnny` as a regular executable command.

#Step 2. Add a dedicated SCA job to `.gitlab-ci.yml`

The goal of this step is to move dependency checking into a standalone job and define which files should remain after the build finishes.

Example job in

```
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
```

Key points in this job:

- `scan dir .` runs analysis against the repository directory;
- `--project`, `--save-results`, and `--create-project` save results to CodeScoring;
- `--stage build` applies policies intended for Johnny Agent runs;
- `--format "coloredtable,sarif>>report.sarif"` keeps console output and writes a `sarif` report;
- the full list of flags and launch modes is available in the Johnny Agent launch guide;
- `bom.json` is generated automatically by the agent and stored as an artifact together with `report.sarif`.

After this change, the pipeline gets a dedicated job that not only runs analysis but also leaves files that are ready for reporting and automation.

#Step 3. Run the pipeline with the new job

Now the first real run should confirm three things at once: the command works, the artifacts are created, and the results are sent to the platform.

1. Save the changes in `.gitlab-ci.yml`.
2. Commit them and push the branch to GitLab.
3. Wait for the `sca` job to start.

How to read the agent exit code

Exit code 1 means the Johnny Agent completed the scan and found issues that match security policies. It does not mean the agent crashed. The full description of exit codes and output formats is available in the Johnny agent launch guide.

After the first run, it becomes clear whether the current variables and permissions are enough for the job to reach the analysis and reporting steps.

#Step 4. Review how a typical Johnny run looks and which files it leaves behind

Below is an interactive demo with an example job and a typical agent run.

After such a run, the build already leaves behind an SBOM and a machine-readable `sarif` report.

#Step 5. Verify that the results are immediately usable

Automatic checking is only useful when the result can be consumed right away by the build, the platform, and later reporting steps.

Check that after the job finishes:

1. The GitLab job artifacts contain:
    - `bom.json`;
    - `report.sarif`.
2. A CodeScoring project named `billing-service-cli` exists if it was not created earlier.
3. The latest run results are saved in that project.

Why these two files are enough

`bom.json` works well as an SBOM artifact for the build, while `report.sarif` is convenient for CI/CD consumers, application security tooling, and later automation.

At this stage, the build does not just perform a check; it leaves behind a practical set of outputs for analysis and reporting.

#Result

The scenario is complete if:

- `.gitlab-ci.yml` contains a dedicated `sca` job that launches Johnny;
- the job produces `bom.json` and `report.sarif` as artifacts;
- the Johnny Agent saves results to a CodeScoring project through `--save-results`;
- the team understands that exit code 1 means policy matches were found rather than an agent crash.

After that, component security is checked automatically during the build, and the reports remain available both in CI and in the platform.

#What next

- add more output formats when needed;

- tune Johnny agent launch options and flags;
- roll the same pattern out to other projects.

#Prepare and export an SBOM for FSTEC requirements

#Context

For certification-related work it is not enough to export a generic SBOM. You need a machine-readable software component list that follows additional FSTEC expectations and still reflects the full composition of the project, including dependencies that are not always exposed by manifests by default.

CodeScoring can export SBOM files in the extended `CycloneDX v1.6 Ext JSON` and `CycloneDX v1.7 Ext JSON` formats adapted for those requirements. To make such an export useful, first build a complete component inventory and then fill in the dependency properties that are not determined automatically.

The example below uses a Python project with `pip`, because it is a clear way to show why a complete component list sometimes requires dependency resolution in the local environment and an explicit package manager path. The same approach applies to other ecosystems where dependency resolution is needed to build a complete inventory.

#What you get

After completing the scenario:

- CodeScoring will contain a project with a full component inventory saved from a local agent run;
- key dependencies will have the properties needed for the exported component list;
- you will be able to download `CycloneDX v1.6 Ext JSON` or `CycloneDX v1.7 Ext JSON` from the interface.

#Requirements

Before you start, make sure you have:

- access to an on-premise CodeScoring installation;
- a CodeScoring license with the **SCA** module enabled;
- a platform API token for the local agent run;
- a local checkout of the project for which the SBOM has to be prepared;
- the required package manager installed locally and its path known, for example `/usr/local/bin/pip3`;
- permissions to view the project and export its SBOM.

#Steps

#Step 1. Prepare the Johnny agent for a local run

First make sure the local run can be started without last-minute setup.

1. Open `[platform-url]/download/` in your CodeScoring installation.
2. If needed, check the current version at `[platform-url]/download/johnny_version`.
3. Download the agent binary for your system.
4. Make the file executable:

```
chmod +x ./johnny
```

After that the agent can be used for local inventory collection and for saving results to the platform.

#Step 2. Build the full component inventory locally first

To get a full dependency graph including transitive dependencies, you need either a lock file from the corresponding package manager or a run with dependency resolution in the build environment. With `pip`, a separate lock file is usually not available, so this example uses the second option and passes the package manager path explicitly.

Example of a local run that also saves results to CodeScoring

```
./johnny scan dir . \
  --api_token <api_token> \
  --api_url <api_url> \
  --project "ppk-fstec-demo" \
  --save-results \
  --create-project \
  --localization en \
  --pip-resolve \
  --pip-path /usr/local/bin/pip3 \
  --bom-path bom-local.json
```

What matters in this command:

- `--pip-resolve` enables dependency resolution in the environment;
- `--pip-path` tells the agent which local `pip` should be used;
- `--save-results` and `--create-project` save the result to a separate CLI project in CodeScoring;
- `--bom-path bom-local.json` keeps a local SBOM copy next to the project;
- the full list of flags and run modes is available in the Johnny scan command guide.

When

`resolve` is actually needed In some ecosystems package managers do not expose transitive dependencies in manifests by default. In such cases CodeScoring recommends dependency resolution in the environment. If a lock file already exists, the agent uses it and does not run a separate `resolve` step.

Below is a shortened interactive demo. It shows the difference between a manifest-only run and a run with `resolve`. With `pip`, this mode should be used only in an isolated project environment, because otherwise extra packages from the local environment can leak into the result even if they are not part of the scanned application.

After this run the platform contains a CLI project with the full component inventory, and `bom-local.json` is saved locally.

#Step 3. Annotate dependency properties before export

An extended SBOM for FSTEC differs from a generic one because extra component properties must be added. Some of them require manual annotation in the project.

1. Open the saved project in CodeScoring.
2. Go to the dependencies table.
3. Click **Configure dependencies**.
4. For the components that need more precise representation in the SBOM, fill in the required fields:
   - **VCS** — a repository or source archive link;
   - **Attack surface** — `yes`, `indirect`, or `no`;
   - **Security function** — `yes`, `indirect`, or `no`;
   - **Provided by** — if the component was borrowed from another product.
5. Save the changes.

What is not filled automatically

The fields related to attack surface, security function, and product origin require expert judgment and should not be treated as fully automatic. That is why it is worth checking and filling them before export.

These values apply only to the current project and are taken into account in its future exports.

After that these values are taken into account in subsequent SBOM exports from the project.

#Step 4. Export the SBOM in the extended CycloneDX format

Now you can get the machine-readable file that already includes the project annotations.

1. On the project page click **Export SBOM**.

2. In the list of formats choose:
    - `CycloneDX v1.6 Ext JSON`, or
    - `CycloneDX v1.7 Ext JSON`.
3. If needed, set the file name.
4. Confirm the export.

Why the

`Ext` format matters A regular CycloneDX JSON export is not enough for this scenario. You need the extended `Ext` format, where the additional component properties are included in `properties`.

After that the local machine receives a component list file in a format adapted to the additional FSTEC requirements.

#Step 5. Check the downloaded file

The final check is needed to confirm that the result is not just an SBOM export, but a file that includes the extra properties and is ready for further review.

1. Open the downloaded JSON file in a text editor or JSON viewer.
2. Find one of the components annotated in the previous step.
3. Confirm that:
    - the component contains a `properties` block;
    - the block includes values for the properties related to attack surface, security function, and product origin;
    - the repository or source archive link appears in `externalReferences`.

After this check you can be confident that the file contains not only the base component list, but also the additional annotations needed for this SBOM workflow.

#Result

You can consider the scenario complete if:

- the local agent run produced the full component inventory and saved the result to a CodeScoring project;
- the required dependency properties are filled in for the components that matter for export;
- `CycloneDX v1.6 Ext JSON` or `CycloneDX v1.7 Ext JSON` has been downloaded from the interface;
- the downloaded file contains additional component properties and source code links where they were specified.

At this point the component list can be used as a machine-readable result for internal review and further preparation of supporting materials.

#What next

- adjust dependency resolution for other ecosystems;
- refine dependency properties for future exports;
- optionally validate the file with the ISP RAS SBOM Checker.

#Block malicious components with OSA Proxy

#Context

If an unsafe package reaches the internal repository manager first and the check only happens afterwards, the team has to investigate the issue later and clean the cache separately. A safer approach is to stop such components before they enter the perimeter and become available to developers and build pipelines.

OSA Proxy is the CodeScoring proxy service that intercepts package-manager requests to remote repositories, checks the requested components, and blocks them when needed. In this scenario it is deployed in front of JFrog Artifactory so that the internal PyPI repository receives only the packages that pass the configured security policies.

#What you get

After completing the scenario:

- OSA Proxy will be deployed and configured for PyPI in `strict` mode;
- JFrog Artifactory will fetch packages through OSA Proxy instead of reaching the external source directly;
- a blocking policy at the `proxy` stage will stop malicious components by the **Dependency is dangerous** condition;
- the result can be checked in `OSA -> Requests`.

#Requirements

Before you start, make sure you have:

- a host or test environment where OSA Proxy can be deployed;
- access to Docker or Kubernetes together with the OSA Proxy image for the selected installation path;
- access to the OSA Proxy `application.yml` file and the ability to restart the service after configuration changes;
- JFrog Artifactory with a remote PyPI repository that can be reconfigured;
- a CodeScoring license with the **OSA** module enabled;
- access to CodeScoring with permissions to create policies and review OSA data;
- a pilot package flow that can be used to validate the setup before rolling it out to all teams.

#Steps

#Step 1. Deploy OSA Proxy

If the service is not installed yet, start by deploying it in a separate environment. For an initial validation, the Docker option is the simplest one.

Example OSA Proxy run in 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>
```

If Kubernetes is used in your environment, the service can also be deployed with a Helm Chart. Both installation options are described in the OSA Proxy deployment guide.

After deployment, the service should be reachable by URL and ready to load its configuration.

#Step 2. Configure OSA Proxy for PyPI and enable strict mode

Now configure the service so that unscanned and forbidden components do not pass any further into the internal repository manager.

Example OSA Proxy configuration for 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
```

What matters in this configuration:

- `work-mode: strict` blocks unscanned components;
- `scan-manifest: true` allows OSA Proxy to remove forbidden versions from the Simple API response;
- `scan-package: true` enables package archive scanning;
- `enable-status-line: true` makes diagnosis easier when the client sees a blocked response;
- `url-encoded-config: true` is needed when OSA Proxy receives internal repository context through the URL.

After updating `application.yml`, restart OSA Proxy.

Roll out strict mode gradually

New requests are filtered immediately, but packages that already reached the JFrog Artifactory cache earlier do not disappear on their own. It is safer to validate the setup on a separate remote repository first and only then move the main traffic to `strict`.

#Step 3. Put OSA Proxy in front of JFrog Artifactory

Now the remote PyPI repository in JFrog Artifactory has to stop reaching the external source directly and fetch packages only through OSA Proxy.

1. Open **Administration -> Repositories -> Remote** in JFrog Artifactory.
2. Find the remote PyPI repository used by your teams for external Python packages.
3. Replace the direct PyPI URL in the **URL** field with the OSA Proxy address.
4. If the remote repository in JFrog Artifactory points directly to the public PyPI source, pass the internal repository context to OSA Proxy through Base64 parameters and use a URL like this:

   ```
   https://osaproxy.example.com/internet-pypi/
   eyJyZXBvTWFuYWdlckhvc3QiOiJodHRwczovL3B5cGkub3JnL3NpbXBsZ
   SIsInJlcG9OYW1lIjoiaW50ZXJuZXQtcHlwaSJ9
   ```

   In this example the Base64 string contains the following JSON:

   ```
   {"repoManagerHost":"https://pypi.org/simple","repoName":"
   internet-pypi"}
   ```

   OSA Proxy uses these parameters to understand which internal repository the request belongs to and which security policies should be applied to it.
5. In the additional JFrog Artifactory settings, enable **Bypass HEAD requests**.

6. Save the changes.

After that new requests for external PyPI packages will go through OSA Proxy.

#Step 4. Create a blocking policy for malicious components

Now create a rule that works at the `proxy` stage and blocks a component when it matches the **Dependency is dangerous** condition.

1. Go to `Settings -> Policies`.
2. Click **Create**.
3. Fill in the main fields:
   - **Name** — for example, `Dangerous dependency at the proxy stage`;
   - **Stages** — `proxy`;
   - **OSA Components** — Packages;
   - **Is Active** — enabled;
   - **Blocker** — enabled.
4. If the rule should apply only to a single delivery channel, choose the corresponding repository. If it should protect all proxy repositories, leave the repository field empty.
5. Add the condition:
   - **Dependency is dangerous**.
6. Click **Create**.

Validate blocking rules on a pilot repository first

At the `proxy` stage, a blocker affects the actual package retrieval rather than just producing a report. If the rule is attached to the main repository without a pilot check, everyday developer and CI requests can suddenly stop.

After the policy is saved, OSA Proxy can apply it to new package requests.

#Step 5. Verify that the client no longer sees forbidden versions

For this check it is more useful to look at the index response returned to the client than to simulate an installation of a single package version.

In this example the client requests the list of available `requests` versions through OSA Proxy and receives an already filtered Simple API response. If a dangerous version is absent from that response, the package manager cannot choose it for installation.

#Step 6. Check the result in `OSA -> Requests`

The final step is needed to confirm that the platform recorded the request and its status, rather than only changing the package-manager response.

1. Open `OSA -> Requests`.
2. Switch to the **Packages** tab.
3. Find the request made through the pilot PyPI repository.

4. Check that the table shows:
    - the package name;
    - the operation mode;
    - the block status;
    - the request date;
    - the request initiator.

At this point you can verify not only the client-side behavior, but also that the event is visible in the platform itself.

#Result

You can consider the scenario complete if:

- OSA Proxy is deployed and runs in `strict` mode for the PyPI source;
- JFrog Artifactory fetches packages through OSA Proxy;
- CodeScoring contains an active blocking policy at the `proxy` stage with the **Dependency is dangerous** condition;
- the client receives a filtered index response, and `OSA -> Requests` shows the recorded request and its status.

After that malicious and forbidden components can be stopped before they enter the internal repository and become available to development teams.

#What next

- review request details and block statuses in OSA;
- refine OSA Proxy work modes and block response settings;
- extend the same pattern to other package managers.

#Find current secrets in project code

#Context

This scenario uses a VCS project with the Secrets module enabled. The video below shows the short path from launching the scan to opening the findings list and using the built-in ML model to focus on secrets that are more likely to require attention.

#Requirements

Before you start, make sure you have:

- a CodeScoring license with the **Secrets** module enabled;
- a VCS project where secrets analysis can be launched;
- access to the project and to the secrets analysis results in CodeScoring.

The embedded video is in Russian

The workflow in the interface is the same. Use it as a visual walkthrough of where to click and what to look at after the scan.

When you watch the walkthrough, pay attention to the **Probability TP** field and to the way the model helps filter out false positives. That makes it easier to move quickly to findings that are more likely to deserve review first.

#What next

- review found secrets and their statuses;
- adjust secrets scanning parameters for the VCS project;
- enable regular secrets scanning.

#Codescoring On-premise Changelog

#[2026.20.2] - 2026-06-02

#Changed

- Updated the Frontend `nginx` image from `nginx:1.29.5-alpine` to `nginx:1.31.1-alpine`
- Secrets Updated the Go base version for Gitleaks from `golang:1.25.4` to `golang:1.26.3`
- Updated the Backend base image from `alpine:3.23.3` to `alpine:3.23.4`
- Updated the Go base version for the policy processing service from `golang:1.25.1` to `golang:1.25.10`
- TQI Updated the JSCPD JavaScript library component set
- Updated the Index Proxy base image from `alpine:3.23.3` to `alpine:3.23.4`
- OSA Updated the OSA API base image from `alpine:3.23.3` to `alpine:3.23.4`
- Updated the Redis image from `7.4.6` to `7.4.9-alpine3.21`

#Fixed

- Fixed policy alert action execution when a specific project is selected in settings
- Secrets Fixed the TruffleHog engine when working with authenticated VCS connections

#[2026.20.1] - 2026-05-25

#Changed

- SCA Renamed the buttons for starting SCA scans

#Fixed

- SCA Fixed a crash on the vulnerabilities page when selecting multiple filters
- Fixed policy processing for the OSA Proxy service
- TQI Fixed an error when starting authors analysis for a new project from the first commit
- OSA Fixed behavior where OSA API did not account for recently updated or created policies
- Fixed project technology display
- Reduced database load during migration to the new project versions model
- Fixed template saving when creating an alert sending action in policies

#[2026.20.0] - 14-05-2026

Minimum PostgreSQL version

CodeScoring 2026.20.0 and later requires PostgreSQL 15 or later for correct operation. If PostgreSQL is not upgraded, correct installation operation is not guaranteed. The upgrade procedure is described in the documentation.

Migration timing

Migration when upgrading to version 2026.20.0 may take longer than usual. This is expected behavior and is caused by the need to migrate projects to the new version control model.

#Added

- Data Added CISA KEV Catalog as a vulnerability source
- SCA Added CISA KEV source information to the vulnerability page
- SCA Added the ability to manage vulnerability VEX statuses
- Secrets Added support for secret scanning with the TruffleHog engine
- Secrets Added the list of secrets engines used to find a secret to the secret card
- SCA OSA Added processing of container image layer information from the Johnny agent. Container images now include base image name and version fields
- SCA Added scanned image information to SCA projects
- SCA Added a page for an image linked to an SCA project
- SCA OSA Added a Layers page to the OSA section, as well as layer tables to the scanned image page in a project and the OSA container image page
- OSA Added commands for creating container image layers
- Added a check for linked policies and policy ignores before deleting a project. If linked objects exist, deletion is blocked
- SCA Added EPSS and SSVC columns and filters to vulnerability tables
- SCA Added a tooltip with the vulnerability description source to the vulnerability page
- Added a last update time column to the alerts table. The value changes when an alert is triggered again

- Added the "Dependency is ransomware" policy
- Added the "Dependency is revoked" policy
- OSA Added support for previously unavailable policies in CodeScoring.OSA
- SCA Implemented version management in project settings
- SCA Added a dependency filter to the vulnerabilities page
- SCA Added the `Inner source` field for dependencies
- SCA Added SBOM package filtering by the `Inner source` field
- SCA OSA Added revocation status to package pages and dependency pages
- SCA Added the ability to select GitFlic as a VCS type
- Added the "Name", "Contact name", and "Contact email" filters to the proprietors page
- OSA API Added the ability to run the `api/osa_components/packages/{id}/scan/` endpoint by package ID
- SCA Added `source-distribution` export to the `externalReferences` section when generating SBOM
- SCA Added output of the scan result link to the `metadata.tools.components.externalReferences` section when generating SBOM
- SCA Added the `percent` and `percentile` parameters for EPSS when exporting SBOM to `vulnerabilities.ratings`
- SCA Added the `is_dangerous` and `is_protestware` parameters when exporting SBOM to `components.properties`
- SCA Added logging of an invalid VCS project access token to the audit log and `tasks` container logs
- Added the ability to clear task priority in the task creation and editing form when editing a policy
- SCA Added SBOM export for project groups
- Secrets Added multiple selection for the "Rule ID" filter on the project "Secrets" tab

#Changed

- TQI API Changed the permissions required to get a Svace key through API. The key can now be obtained by a user with the `Administrator`, `Security Manager`, or `User` role with `Owner` or `Developer` permissions in at least one project
- SCA Changed EPSS and SSVC item display: they are now always shown in the vulnerability score block and CVE.ORG feed information regardless of data availability
- SCA Changed vulnerability revocation display logic: the revocation date is shown in the top block only if the vulnerability is revoked in all sources. In a source, a vulnerability is considered revoked if there is at least one revoked entry
- Secrets Implemented deduplication of findings from different secrets engines
- Changed system requirements: the minimum PostgreSQL version is now 15+
- Renamed the "Dependency authors list" policy condition to "Dependency authors email list" in the policy creation and editing form

- OSA Changed the minimum allowed timeout value to 1 when configuring a container registry
- Improved loading speed for dashboard data with top 5 policies and alert counts
- Improved security policy check performance in the OSA Proxy service

#Fixed

- Data Fixed incorrect publication dates shown as 1970 for packages from NuGet and Packagist
- Data Fixed author information collection for packages from crates.io
- Optimized the periodic `check_vcs_availability` task by removing an extra database query
- SCA Fixed email notifications for SCA analysis when the project is excluded from SCA analysis
- SCA Fixed SBOM export without licenses: manually changed licenses no longer appear in the report when licenses are excluded from export
- SCA Fixed policy ignore export in PDF reports: the "Only active ignores" checkbox was renamed to "Only effective ignores" and now excludes ignores for inactive policies by default
- Secrets Fixed the "Rule ID" filter when exporting a Secrets PDF report
- API Fixed the Swagger schema for PDF report export endpoints
- Fixed the "Vulnerability Age is empty" policy
- Fixed the "Dependency Version is empty" policy
- OSA Fixed OSA policy checks for vulnerabilities that were missing from the CodeScoring database at scan time
- SCA Fixed the vulnerable function counter when scanning from the UI
- Fixed a deadlock when updating the repository list
- SCA Fixed displayed dependency names and versions in project dependency graphs and dependency detail pages
- OSA Fixed duplicate image tags during scans via OSA
- API Fixed the validation error response when adding a project to a group: the response now matches the API contract
- SCA Optimized the project scan history page loading
- SCA Optimized vulnerable node marking in the dependency graph
- Fixed access permissions for project group settings
- Fixed an OIDC login error
- API Aligned access to some endpoints with the role model
- Secrets Fixed an incorrect custom secrets model status after deleting the model
- SCA Fixed sorting in SCA projects

#[2026.11.2] - 2026-04-03

#Added

- OSA Added the `DATABASE_CONNECTION_POOL_TIMEOUT` environment variable in OSA API to configure the timeout for acquiring a connection from the pool. The default value is `30` seconds

- OSA Expanded database connection pool metrics in OSA API by adding `requests_queued`, `requests_wait_ms`, `requests_errors`, `connections_errors`, `connections_lost`, and others

#Changed

- OSA Separated scan result waiting from database connection holding in OSA API, so connections are no longer blocked while a scan is running

#Fixed

- SCA Fixed license parsing for components in `SBOM` files generated by third-party tools
- OSA Fixed `Redis` availability checks in OSA API
- OSA Fixed the use of the `SCAN_WAIT_TIMEOUT` scan timeout when waiting for scan results in OSA API
- SCA Fixed a `403` error when rerunning `SBOM` scans from `CLI` project history for users with the `USER` role
- Fixed an issue that made the vulnerabilities section unavailable for users with the `user` role

#[2026.11.1] - 2026-03-24

#Fixed

- Fixed Index API being unavailable for installations running via a proxy
- Fixed the audit log message format for vulnerability updates when the previous update task has not yet completed
- Fixed saving SSVC metrics when receiving vulnerabilities from Index API

#[2026.11.0] – 2026-03-13

#Added

- Added a dedicated alert page
- Added Kaiten as a task manager integration
- SCA Added EPSS data and related security policies to the vulnerability page
- SCA Added CVSSv4 scores and related security policies to the vulnerability page
- SCA Added the SSVCv2.0.3 score and related security policies to the vulnerability page
- SCA Added protestware categories and related security policies to the vulnerability page
- SCA Added publication, revocation, and update dates for vulnerabilities in a specific feed
- SCA Added the ability to copy vulnerability identifiers on the vulnerability page
- SCA Added a tooltip for the top description block on the vulnerability page

- SCA Added a tooltip for the "Scan with hashes" field to the project settings
- SCA Added the "Ignore note" field to the policy ignores table in the PDF report
- SCA Added the "Note" column to the CSV export on the alerts page
- SCA Added sorting by the "Found at" column in "Affected dependencies" on the vulnerability page
- SCA Added display of the number of successful SCA analyses in a project group
- SCA Added highlighting of the "VCS" and "Licenses" fields on the project dependency settings page when changed manually
- SCA Added an image hash filter on the CLI project scan history page
- SCA Added a licenses column on the dependency page
- SCA Linked the "Environment" and "Technology" filters on the Dependencies page
- SCA Added the "Blocking status" field for alerts in PDF and CSV reports
- SCA Added a column with a link to the alert page in the alerts table on the SCA project page
- OSA Linked the "Repository" and "Repository manager" filters on the requests and packages pages
- OSA Added an "Available" filter to the "Repository managers" page
- OSA Added a CodeScoring user filter on the OSA requests page
- OSA Added a field to configure the timeout for checking image availability in the registry before analysis
- OSA Added the last scan date to the container page
- OSA Added logging of successful completion and container image load statistics in the audit log
- OSA Added marking an image as “Outdated” when a scan fails due to a missing image and cleanup of outdated images in `periodic_cleanup_osa_components`
- OSA Added the `USE_JSON_LOG_FORMAT` environment variable in the OSA API that controls log format (default `true`)
- API Added the `projects_isnull` filter to `/api/policies/`, `/api/policy_alerts_v2/`, `/api/settings/policy_ignores/`
- License and vulnerability update runs from the Index API now create an audit log entry

#Changed

- Secrets Updated the model for assessing secret validity
- Secrets Changed the fine-tuning sampling methodology
- OSA Changed container image loading logic to use explicitly specified media types
- SCA Renamed a field in PDF report filters
- SCA Changed the numbering field to the alert identifier in the "Policy Ignores" table of the PDF report
- SCA Changed CVSS scores, metrics, and severity display: data now comes from the source with the highest score for each CVSS version
- SCA Changed publication/revocation/update date display on the vulnerability page: shows the earliest publication and revocation dates and the latest update date across sources. Policies with conditions on

these properties will be updated on the next project, image, or package scan
- SCA Changed the provider priority for vulnerability metadata: CVE.ORG, GHSA, Kaspersky, BDU, then others in alphabetical order
- SCA Changed the color for the "critical" CVSS severity level
- SCA Updated the "Impact" field text on the vulnerability page for the Kaspersky feed
- SCA Improved criticality display on the vulnerability page
- SCA OSA Simplified SBOM parameter editing for SCA projects and container images: settings open in edit mode by default and can be saved with a single action
- API Changed the `impacts` field format for vulnerabilities in the relevant API methods to `pk-name`
- Changed the release-date policy logic: only the date (no time) is now considered, and time is no longer shown in the UI
- Updated base images for backend / tasks-*, judge, index-proxy, osa, and frontend (Alpine 3.23.3, nginx 1.29.5)
- Removed default scopes when configuring OIDC

#Fixed

- OSA Fixed scanning images with missing alerts
- OSA Optimized security policies processing for requests with more than 100 packages
- OSA Optimized OSA API: standardized access checks and request context handling, improved token and key caching
- OSA Fixed desynchronization between the policy service and OSA when checking dependency policies
- SCA Fixed dependency graph scaling
- SCA Fixed saving vulnerability reachability results for existing projects
- SCA Fixed vulnerability source names in SBOM export
- SCA Fixed duplicate rows in the "Affected images" table on the vulnerability page
- SCA Fixed counting vulnerabilities with missing CVSS3 severity on SCA project and project group pages
- SCA Fixed dependency, vulnerability, and alert counts in project groups when no SCA analysis has been performed
- SCA Fixed the missing `"language"` property for dependencies in SBOM export formats `1.6_ext` or `1.7_ext`
- SCA Fixed OSS Index integration in SCA
- SCA Fixed updating the "fixed version" field for vulnerabilities where the latest fixed version was revoked
- SCA Fixed vulnerability counts for dependencies in SCA scan history
- SCA Optimized dependency page loading
- TQI Fixed author count errors in audit reports when repositories returned 404
- TQI Optimized author comparison algorithm and reduced false negatives
- Secrets Fixed a secrets serialization bug when analyzing a project connected via the "Other Git VCS" integration
- SCA Fixed SBOM export filters behavior that caused the file to be exported in full regardless of selected content

- OSA Fixed image scanning via OSA API when the specified repository manager had not been created
- OSA Fixed duplicate image tags appearing when loading images from the registry
- API Fixed the `/api/commits` schema: the `author.id` field is now `nullable`
- Fixed report file downloads getting stuck in the "In progress" status when a save request fails
- Fixed group rule creation being blocked when LDAP is unavailable
- Fixed incomplete audit logging for policy actions
- Removed usage of the `public` schema in migrations, which caused errors when migrating to PostgreSQL 15+ with a non-`public` schema

#Removed

- Secrets Deleting a project now removes the secrets found in it

#[2026.3.3] - 2026-03-12

#Added

- Added separate environment variables for Index API timeouts: `INDEX_API_POOL_TIMEOUT`, `INDEX_API_CONNECT_TIMEOUT`, `INDEX_API_WRITE_TIMEOUT`, `INDEX_API_READ_TIMEOUT`

#Changed

- Split the single Index API request timeout into pool, connect, write, and read

#Deprecated

- Deprecated the `INDEX_API_TIMEOUT` environment variable

#[2026.3.2] - 2026-02-20

#Fixed

- Fixed a bug that prevented the `vulnerability_is_reachable` tag from being applied to vulnerabilities in exported SBOMs.
- Fixed an error scanning system packages with the same names and versions but different qualifiers
- Fixed an error analyzing projects with the `scan with hashes` option
- Optimized the project group section

#[2026.3.1] - 2026-01-30

#Added

- PostgreSQL upgrade configuration has been added. The upgrade procedure is described in the documentation

Upgrading to PostgreSQL 15

Until the end of 2025, CodeScoring shipped with PostgreSQL major version 13. The final release of PostgreSQL 13 (13.23) took place on November 13, 2025. To ensure secure and supported operation of CodeScoring, you must upgrade to PostgreSQL 15. Administrators of installations deployed in Docker Compose must specify the POSTGRES_IMAGE variable in the .env file during the upgrade.

#Fixed

- SCA Fixed a behavior where Johnny considered policies that triggered delayed alerts as blocking
- Fixed a behavior where ignored alerts sent email notifications and created issues in Jira
- OSA Fixed an error scanning some Maven packages
- Fixed a behavior where the unique author count would fail after 60 seconds

#[2026.3.0] - 2026-01-16

#Added

- SCA Added support for CycloneDX versions `1.7` and `1.7 ext`
- SCA Added a vulnerability reachability visualization page
- SCA OSA Added the `GOST:provided_by` field to the dependency edit table and to SBOM
- SCA Added the ability to specify a branch or tag when importing an SBOM
- SCA Added validation of the uploaded SBOM file format and version
- SCA Added an API endpoint with a list of supported SBOM specifications and versions
- SCA Added highlighting of vulnerable dependencies in the dependency graph when exporting a project PDF report
- SCA Added “Affected packages” and “Affected images” lists on the vulnerability page
- SCA Added the “Requirement” column to the table in SCA → Dependencies
- TQI Added filters by start date and last update date to the project list in the TQI section
- TQI Added code density and code churn metrics to charts on the TQI project page
- TQI Added display of project duration in months, lines of code count, and file count on the TQI project page

- TQI Added merge commit icons, commit type filtering, and the ability to collapse commit messages in the commits table on the TQI project page
- TQI Added scheduling configuration for author and clone analyses for a project
- TQI Added removal of outdated commits when running author analysis from the first commit
- OSA Added support for repository managers of the “Sfera.Distributions and Licenses” type
- OSA Added the `Virtual` repository type for JFrog and Nexus repositories
- OSA Added additional fields for container images (request time, policies, blocking status, registry, link)
- OSA Added the ability to specify Docker repositories for policies with OSA components and to pass the repository name when requesting an image
- OSA Added uniqueness validation when creating repository managers
- OSA Added recogninition of `cargo`, `composer`, and `huggingface` ecosystem formats in Sonatype Nexus Repository requests
- OSA Added recognition of `ai editor extensions`, `ansible`, `bazel modules`, `helm oci`, `huggingface`, `jetbrains plugins`, `nim model`, and `oci` ecosystem formats in JFrog Artifactory requests
- OSA Added PURL display on the container image page
- OSA Added filtering by OSA components in the policy list
- OSA Added the ability to pass host headers when generating a link to a blocked component in OSA
- OSA Added support for loading image data from JFrog Artifactory registries via the Repository Path API
- OSA Added updating of the last request date and scan status when scanning an image or package via the UI
- Secrets Added saving of secret coordinates
- Secrets Added secret coordinates to secret links
- Secrets Added a field for configuring the default secrets engine
- Data Added left boundary for ranges of vulnerable versions from the FSTEC Database by major version, which eliminates range overlap and reduces False Positives
- Data Added collection of metrics (CVSS v3 Rating and Threat Level) for vulnerabilities with Astra Linux
- Data Added fixed-version collection for components from the 'Possible Mitigation Measures' section of the FSTEC BDUs
- Added a bulk action for alerts: unlinking Jira issues
- Added navigation links between project group settings and group view pages
- Added the ability to use lists in policy conditions
- Added informational messages when excluding a project from analysis
- Added fields for affected OSA components, groups, and repositories on the policy view page
- Added display of scan execution time in project metadata
- Added display of system component versions in the “About” modal
- Added escaping of special characters in LDAP search

#Changed

- SCA Redesigned the vulnerability page interface

- SCA Changed the display of alert counts and dependency and vulnerability information on the project page when SCA analysis has not yet been performed
- SCA Changed the PURL comparison logic used when deciding whether to ignore an alert. See Policy ignores for details
- SCA SBOM export from the UI is now performed asynchronously (similar to CSV and PDF reports)
- SCA Updated CVSS rating filters: allowed values are now limited to the [0,10] range
- SCA Removed the has_exploit field from SBOM when no exploit is available
- TQI Redesigned the UI on TQI project and author pages
- Updated the loading indicator in the platform UI
- Changed user access with the User role to alert-related actions
- API Converted string values `True` / `False` to boolean `true` / `false` in API filter responses
- Changed the HTTP method for the UserInfo request in OIDC to `GET`
- Changed the base image in the judge service from Alpine 3.20 to Alpine 3.23.2
- Changed the base image in the index-proxy service from Alpine 3.20 to Alpine 3.23.2
- Changed the base image in the osa service from Alpine 3.20 to Alpine 3.23.2
- Changed the base image in the backend service: /tasks-* from Alpine 3.20 to Alpine 3.23.2
- Changed the base image in the frontend service from nginx:1.26.3 (Alpine 3.20.6) to nginx:1.29.4

#Fixed

- SCA Optimized the SBOM generation process
- SCA OSA Optimized SCA and OSA performance
- SCA Optimized vulnerability export to CSV
- SCA Fixed duplicate alerts when filtering by dependency relation or dependency environment
- SCA Fixed the `Has VCS specified` filter on the project dependency edit page
- SCA Fixed incorrect export of the match type field in the dependency report
- SCA Fixed vulnerability count calculation on the packages page
- SCA Fixed severity display when a CVSS score is present
- SCA Reduced memory consumption during intensive SCA usage
- SCA Removed alerts with delayed blocking when scanning via Johnny with the --save-results flag if the blocking time has not yet been reached
- SCA OSA Optimized API operation and UI page loading for all OSA section pages, alert list page, and SCA project groups
- TQI Optimized loading of the “Similar authors” tab on the author page
- TQI Optimized chart loading on the TQI project page
- TQI Fixed possible duplicate commits in TQI projects
- TQI Fixed the change rate calculation formula for the chart on the TQI project page
- TQI Fixed technology share calculation in project groups

- OSA Fixed display of packages from deleted repository managers in the dashboard
- Secrets Fixed saving of identical secrets with different coordinates in a file
- Secrets Fixed audit log output and logging of missing model errors when starting secrets analysis
- Data Fixed incorrect VCS links with `/sponsor` and github.io
- Data Fixed display of licenses by package versions - eliminated cases where licenses belonged to a different version
- Data Fixed incorrect FSTEC BDU versions with "X" symbol and commas
- Data Fixed incorrect VCS links in npm with /issues at the end
- Fixed unlocking of the PDF export button after scanning an image or project
- Fixed number formatting on dashboard and TQI project charts
- Fixed display of platform sections when a module expires in the activation key
- Adjusted project statistics display for projects with no scans yet

#Deprecated

- OSA API endpoints for container image scan history have been marked as deprecated

#Removed

- OSA Removed the container image scan history page
- TQI Removed time selection from start date and last activity date filters in the author and organization project lists
- TQI Removed the “Change rate” metrics from TQI project and author pages
- Removed navigation history persistence (in the browser) on internal tabs

#[2025.45.7] - 2025-12-19

#Added

- OSA Added `INDEX_API_TIMEOUT` and `LOG_LEVEL` settings to the Index Proxy service

#Fixed

- OSA Fixed policy behavior in OSA Proxy without specifying `repository_id`
- SCA Fixed versioning of npm packages in SBOM files

#[2025.45.6] - 2025-12-13

#Added

- OSA Implemented policy mechanism using cache in the OSA Proxy service

#Fixed

- SCA Fixed SBOM file generation for projects with protestware

#[2025.45.5] - 2025-12-05

#Added

- OSA Added automatic authorization type selection for container image registries

#Fixed

- SCA Optimized the query for getting a list of projects on the dependency page

#[2025.45.4] - 2025-12-02

#Fixed

- SCA Optimized PDF report export
- SCA Optimized dependency graph generation

#[2025.45.3] - 2025-11-24

#Fixed

- OSA Fixed package display when the repository manager has been deleted
- OSA Optimized loading of the packages list
- Secrets Updated the ML-based secret detection strategy in the Secrets module

#[2025.45.2] - 2025-11-18

#Added

- SCA Added saving of metadata when rescanning a project from the scan history

#Fixed

- OSA Optimized package scanning performance in the OSA module
- Fixed an issue with ignore handling during OSA Proxy policy evaluation

#Changed

- OSA Updated logging detail level in the OSA module

#[2025.45.1] - 2025-11-11

#Fixed

- Fixed a bug with self-signed SSL certificates
- OSA Fixed incorrect display of package blocking status

#[2025.45.0] - 2025-11-07

#Added

- SCA Added new section `SCA → Project Groups` with aggregated metrics and bulk analysis
- SCA Added CSV export of aggregated SCA project data within a group
- SCA OSA Added component analysis for the RedOS ecosystem
- Added new filtering options for the dependency graph: filtering by parents/children and by technology
- TQI Added display of merge commits count and changed lines in project and author tables, on project/author pages and in charts
- TQI Added option to run author analysis from the first commit
- TQI Added metrics “Code Rate”, “Newness Impact” and “Churn Impact”
- TQI Added “Code Rate” chart on the project page
- Added autofocus for forms: authentication, policy settings, policy ignores, repository managers, registries, projects
- Added indicator showing applied filters
- SCA Added API set for aggregated SCA project data (see `/api/swagger` for details)
- OSA Added column and filter “Request initiator” in `OSA → Requests`
- Added sample queries with real filters for `/bulk_actions/` endpoints in Swagger
- SCA Added “Requirement” column in dependency and scan lists
- SCA Added “Requirement” field to SCA PDF report
- SCA Added “Groups” column to project list and CSV export
- SCA Added “Max fixed version” field on dependency page
- Added filters “Authors’ emails” and “Merged with” on the author merge rules page
- Secrets Added “Status” filter on project page
- Secrets Added “Status” filter on secrets list page
- Added policy ignore section to PDF report
- SCA Added “Parents” filter on dependencies page
- Reworked `/health` endpoints in REST services and added checks for celery-beat and celery-worker

- Added links to specific CSPW vulnerabilities in Johnny reports
- SCA Added filters on dependency vulnerabilities page
- Added project link to notification about project creation
- Added ability to duplicate a policy
- Added ability to collapse the sidebar
- OSA Added links to policies on the blocked component page
- Added “Blocker” column to alert tables and delayed blocking details in block status
- Added expand/collapse button in the secret card
- Added support for missing trailing slash on Johnny download page (`/download`)
- Added ability to automatically prefill OIDC settings using the **Get settings** button (see OIDC configuration for details)

#Changed

- Moved technologies column on project lists grouped by author
- Renamed field `Host` to URL on container registry configuration page
- Updated filter panel behavior: state no longer persists across sections; panel stays open after applying filters
- OSA Removed initial relevance filters from OSA section lists
- Removed password change option for LDAP/OIDC-based users
- Secrets Removed syntax highlighting in Secrets section
- Added multi-select support in filters of vulnerabilities and alerts pages
- Updated display formats of “LDAP Server” and “OIDC Server” fields on user list page
- Fixed API content-type definitions in Swagger
- Updated permissions for project analysis and view in CodeScoring.SCA and CodeScoring.Secrets, see the User Management) section for more details
- TQI Improved tooltip data display on project metric charts
- Updated content of “New Version” modal
- Changed side menu structure and content
- Changed policy blocking delay calculation to exact hours instead of days
- Improved Russian translation quality
- SCA Moved `sca-tasks` queue to celery worker
- SCA Removed unicode encoding of text in SBOM files
- Updated Redis to version 7.4.6

#Fixed

- Optimized retrieval of fixed-value filters
- Optimized loading of author activity map
- Fixed fields “Type” and “Authorization Type” on container registry edit page
- Fixed formatting of dependency/vulnerability tables in PDF report
- OSA Fixed deadlock when scanning multiple packets with the same name but different qualifiers
- SCA Improved SCA analysis performance
- SCA Improved SBOM generation performance during SCA analysis
- Fixed reset of “Source files” field after rescan

- Fixed policy actions edit behavior
- Secrets Fixed secrets list display
- Fixed display of vulnerable packages in dependency graph
- Fixed filtering of container images by blocking status
- Fixed link to active policies on dashboard

#Deprecated

- Endpoint `/api/tqi/projects/:id/by_complexity` marked as deprecated. Use `/api/tqi/projects/:id/by_commits` instead
- Environment variable `PLATFORM_API_MAX_PAGE_SIZE` marked as deprecated. Use `CODESCORING_API_MAX_PAGE_SIZE`

#Removed

- Removed queue `high-priority`
- Removed problematic links from author detail page that caused UI jumps

#[2025.37.5] - 2025-10-27

#Added

- The maximum number of elements per page in the API is now controlled by the environment variable `PLATFORM_API_MAX_PAGE_SIZE`, the default value is 100. Exceeding this value in the request will cause an error.

#Fixed

- OSA Optimized loading of images from registries
- SCA Optimized the API request for getting project dependencies

#[2025.37.4] - 2025-10-15

#Fixed

- Fixed an issue where duplicate email notifications and Jira issues were generated for policies with the same project in the group and in policy actions
- Fixed processing of policy conditions based on the CVSS3 Severity criterion
- Fixed a 500 error for certain policy stage parameters
- Fixed policy behavior for the "PURL exactly matches" condition
- Fixed an issue with Jira tasks priority setting

#[2025.37.3] - 2025-10-07

#Fixed

- SCA Fixed an error determining paths to the `.csproj` and `project.assets.json` manifests when scanning .NET projects

#[2025.37.2] - 2025-10-03

#Fixed

- Fixed an issue where erroneous notifications were sent for policies without configured actions
- SCA Added a database migration to correct the archived flag for alerts generated by SCA analyses

#[2025.37.1] - 2025-09-25

#Added

- OSA Added ability to deactivate the repository manager
- Added `media-cleaner` queue in the `tasks-media` service
- Updated `celery` library version

#Changed

- Tasks related to cleaning up obsolete media files from the filesystem moved to a separate `media-cleaner` queue

#Fixed

- OSA Fixed issue with ignoring alerts for OSA packages
- Fixed Index Proxy server error when the activation key owner’s name contains Cyrillic characters
- Optimized repository manager removal
- SCA Fixed issue in the "PURL contains" policy: if the checked element was not a valid PURL, the policy did not trigger
- SCA Fixed issue when SCA scanning projects with dependencies having identical name, version, and language
- Fixed issue with enabling inactive policies
- Secrets Fixed error when exporting secrets PDF report
- Fixed error when the `error` parameter is missing in the OpenID Connect response

#[2025.37.0] - 2025-09-12

#Added

- SCA Added vulnerability reachability analysis for Java, powered by the Svace call graph builder
- SCA Added policy "Vulnerability is reachable"
- SCA Added UI display of the reachability attribute for vulnerabilities
- SCA Added ability to specify `component manufacturer` data to be placed in the relevant section of SBOM versions `CycloneDX 1.6` and `CycloneDX 1.6 ext`, both at platform level and per project
- Added new environment variables `DEFAULT_PROJECT_MANUFACTURER_NAME`,

DEFAULT_PROJECT_MANUFACTURER_EMAIL,
DEFAULT_PROJECT_MANUFACTURER_HOMEPAGE

- SCA Added saving of annotated data when importing SBOM
- SCA Added support for SBOM export in SPDX format
- SCA Added suspicious commit links for vulnerabilities with CSPW identifiers in SBOM
- Added integration with external identity providers implementing the OpenID Connect protocol
- SCA Added localization of PDF reports in the SCA module
- Added saving of links for Jira issues and emails created by automatic actions with alerts
- SCA Added "Requirement" field displaying the required version range from the manifest in the project dependencies section
- Added optional "Priority" field in the Jira task creation policy action form
- Added ability to create custom templates for emails and Jira tasks in alert actions (default template is built-in)
- Added script for re-encrypting sensitive data when changing the SECRET_KEY token
- SCA Added "Release Date" field to the project dependencies list
- Added grouping of options in the condition dropdown when configuring a policy
- Added rule dragging on the policy configuration page, and updated their appearance
- Added ability to automatically update the audit log list
- SCA Added project search and filters by project, relation, detection type, and environment for individual vulnerabilities on the dependencies page
- TQI Added ability to calculate the number of unique authors in GitLab outside of analysis runs
- SCA Added ability to run SBOM scans from scan history to check for new vulnerabilities in historical component data
- SCA Added "Max fixed version" field to the dependencies section of the project PDF report
- TQI Added filtering by multiple authors on the "List" and "Activity Map" tabs in the `TQI -> Authors` section
- TQI Added tooltip popup with author’s projects when hovering over author’s project count in the `TQI -> Authors` section
- TQI Added author filter on the project list page in the TQI section
- TQI Added "Number of Authors" chart on the project page in the TQI section
- TQI Added "Author’s Commits" and "Author’s Projects" charts on the author page in the TQI section
- Added ability to rename projects
- Added ability to perform bulk actions on certain entities in the Settings section
- Added ability to stop report generation
- Secrets Added ability to export a CSV report with secrets on the project tab
- Secrets Added ability to generate a PDF report with secrets on the project tab

#Changed

- SCA Improved parsing of VCS links in CycloneDX SBOM
- TQI Improved the mechanism for merging authors by email
- Unified the UI for running alert actions manually and automatically
- SCA Commit hash in the project PDF report is now displayed in full
- Reduced maximum number of items in paginated API responses to 100
- Blocked launch of SCA and TQI analyses until code cloning is completed for VCS projects
- Improved messages about inability to run SCA analysis for VCS and CLI projects
- OSA Changed detailed package info display on the OSA module package page
- TQI Fixed display of "Change author mapping" button when TQI module is disabled
- Changed `GOST:source_lang` field to `GOST:source_langs` in all exported files
- TQI Added information on the number of months an author committed to the TQI section
- TQI Project parameter "Duration (in months)" now shows integer number of months of project activity
- Removed `internal` field from `/api/activation_keys/` API response
- Split project list API into modules (SCA, TQI, Secrets)
- Migrated base images of Index Proxy, `frontend` service, and `backend / tasks-*` services from Debian bookworm to Alpine Linux
- Improved performance of scan history
- SCA Optimized loading of dependency and vulnerability lists
- Secrets Changed display and content of tooltips about inability to train ML model in Secrets module
- Changed "Deferred blocking" filter type on the policies page

#Fixed

- SCA Fixed alert table format in project PDF report
- Fixed link to refactoring candidate projects in the "Dashboard" section
- Fixed reset of selected number of items per page in tables when changing sorting
- Fixed pagination reset when navigating to the same page
- SCA Fixed link handling in CycloneDX SBOM
- SCA Implemented missing logic for handling alerts when comparing PURL values with case sensitivity
- Fixed auto-generated type names in OpenAPI schema
- Fixed verb forms of policy conditions in English locale
- TQI Fixed project links in "Organization Projects" tab on the author page
- TQI Fixed links to code duplicates in the duplicates map
- TQI Fixed CSV export of author’s projects
- Fixed slow loading of alerts when filter "Repository Manager = N/A" is selected
- Fixed Swagger schema for `/api/activation_keys/` endpoint
- SCA Fixed missing options in dependency environment filter when `USE_SMART_FILTERS` is set to `False`

- SCA Fixed error when scanning a project with dependencies containing invalid PURLs
- Fixed search in dropdowns for Russian locale
- TQI Fixed "Refactoring candidate projects" and "Intra-project duplicates" widgets on the dashboard
- Fixed search within condition dropdowns on the policy editing page
- SCA Fixed behavior of "Dependency is dangerous" policy in SCA module
- SCA Fixed policy execution error with condition `PURL exactly_match`

#[2025.29.4] - 2025-08-22

#Added

- Added granular timeout settings for container image registries: connect, pool acquisition, read, and write
- Added a per-registry requests-per-second (RPS) rate-limit setting for connected container registries

#Changed

- Changed how parameters for connected container images registries are configured: environment-variable configuration has been removed; all settings are now managed in the UI.

#[2025.29.3] - 2025-08-15

#Added

- Added task results storage time setting in Redis `TASK_RESULT_EXPIRATION_PERIOD`

#Changed

- Changed the version of the osa-api Alpine base image from 3.21 to 3.20 to support the latest versions of Dynatrace monitoring

#Fixed

- Fixed the lack of policy alerts in the PDF report download for the first SCA scan
- Optimized database queries when calculating policy ignores
- Fixed the behavior of archiving OSA packages. OSA Packages that were loaded but never requested are now sent to the archive
- Fixed saving the checkbox "Can create CLI projects via API" on the CodeScoring user settings form
- Optimized regular updates of current OSA packages

#[2025.29.2] - 2025-08-05

#Fixed

- Fixed background package update error in OSA module
- Fixed duplication of vulnerabilities in PDF project report
- Added timeout settings for Redis connections via variables `REDIS_SOCKET_CONNECT_TIMEOUT`, `REDIS_SOCKET_TIMEOUT`, `REDIS_SOCKET_KEEPALIVE`

#[2025.29.1] - 2025-07-22

#Fixed

- SCA Optimized loading of dependencies page in SCA module
- Fixed error when changing LDAP settings

#[2025.29.0] - 2025-07-18

#Added

- SCA Added the ability to view the results of SCA analysis of a project from the scan history
- Added the ability to customize the export of SBOM and PDF files
- Added the ability to manually create a task in the task manager and send letters for selected alerts
- Added the ability to configure deferred blocking in policies
- Added the ability to apply policy ignores to project groups
- Added a new Security Manager role
- SCA Added a condition for the `Dependency is a descendant` policy to search for child dependencies of a selected package at any level of the dependency graph
- SCA Added the `Max fixed version` column in the dependency table
- SCA Added the ability to download the current version of the Johnny binary agent directly from the platform
- Added the "Technology" column to the alert list export
- Added pop-up notifications with the analysis result upon its completion
- Added search in drop-down list of criteria in policy creation and editing forms
- Added duplicate block of buttons after the group of conditions in policy creation and editing forms
- Added ability to expand the policy conditions management block
- Added settings for the format of date and number output in the UI
- SCA Added legend for the project dependency graph
- SCA Added analysis identifier to webhooks associated with the completion of SCA analysis
- OSA Added output of the archive/activity flag for OSA packages, container images and alerts
- Added check for data presence when exporting a PDF report
- Added error handling when trying to download a file that was deleted due to expiration of the storage period

#Changed

- The `tasks-media` queue has been transferred to Celery. The number of workers is controlled by the variables `CELERY_MEDIA_WORKER_CONCURRENCY` (minimum, default is 2) and `CELERY_MEDIA_WORKER_MAX_CONCURRENCY` (maximum, default is 4). The variable `HUEY_MEDIA_WORKERS` has been removed
- OSA The OSA background package update mechanism has been optimized. Only relevant packages are updated. By default, packages requested in the last 14 days are considered relevant, the parameter is configured in the platform settings
- Improved error messages when checking the availability of repositories via SSH
- Improved display of the list of events in the "Webhooks" section
- Secrets Improved logic for displaying the ML model management section in the Secrets module
- Optimized the algorithm for launching policy recalculation when updating vulnerabilities: launch occurs only when data that affects policies changes
- Optimized loading of pages with images and alerts
- Changed the choice of a secure protocol for connecting to a mail server from checkboxes to a field with a drop-down list
- Unified action buttons in sections with entity tables
- Updated the OpenAPI specification in terms of error handling
- OSA Changed the base image in the OSA API service from Debian bookworm to Alpine
- Updated the pgbouncer image to switch from libevent to c-ares as a DNS backend for support SOA record resource types and EDNS0 protocol
- Secrets Updated gitleaks version for Secrets module to 8.27.0
- Updated Redis image from 7.0.12 to 7.4.3
- Updated PostgreSQL image from 13.4 to 13.21
- SCA Updated Johnny version on platform to 2025.29.1

#Removed

- Removed `HUEY_MEDIA_WORKERS` variable

#Fixed

- Fixed search on author profile page in "Organization Projects" section
- SCA Fixed image hash validation error when passing value via `--hash` parameter when scanning image via Johnny
- Fixed some API schema inaccuracies in Swagger
- OSA Fixed error processing list of entities from Docker Registry when receiving `null` instead of empty list
- Fixed error processing image signature files with `.sig` extension and `.att` when working with container registries
- TQI Fixed an error on the "Similar Authors" tab on the author profile page in the case of authors without certain technologies
- Fixed an error with incomplete output of connected version control systems when creating a VCS project

- Fixed the output of a long file name in the list of exported files
- Fixed the output of the "Until" field in the policy ignore modal window
- SCA Fixed the reset of the dependency graph state when switching to another browser tab
- Fixed problems updating data on the page after editing some entities
- OSA Fixed deadlocks when using multiple instances of the OSA Registration service
- Fixed an error when editing the connection to the Jira task manager
- Fixed incorrect hiding of sensitive data in an error when cloning a project
- Fixed the display of selected parameters in the policy condition
- OSA Fixed the output of the list of authors on the page with detailed information about the package blocking in the OSA module
- Fixed saving policies when changing certain conditions

#[2025.21.2] - 2025-06-18

- Optimized data migration process when updating the project scan schedule

#[2025.21.1] - 2025-06-04

- Optimized CSV report export in the "Dependencies" section
- Optimized memory utilization when exporting PDF reports

#[2025.21.0] - 2025-05-21

- Added new policy condition "Dependency is Protestware". Threats that are associated with protest software are marked with the CSPW identifier
- Added interface adaptability for different screen sizes, the system is now easier to use on tablets and mobile devices
- Added the ability to specify the name of the VCS project and create multiple VCS projects for one repository
- Added the ability to run SCA analysis for a VCS project by selecting a specific branch or tag without changing the default branch
- Changed creation and editing of policies: now it is possible to specify a project regardless of selected groups and owners, the policy will work for all selected groups and projects
- Added a link to the repository manager to the package page in OSA
- Added filter by technology and corresponding column in the "Alerts" section, the column is hidden by default
- Added filter by project group in "Alerts" and "Dependencies" sections
- Added multiple selection for "Attack Surface", "Security Function", "Found", "Dependency Environment" filters in project dependencies settings
- Added "Note" column to the "Policy Ignores" section, the column is hidden by default
- Added filters "Type", "Authorisation type", "Active" and search by name and address to "Registries" section
- Added time zone in PDF report generation date
- Added limitation of the number of login requests from the same user, 10 attempts per minute by default

- Added TLS encryption support for PostgreSQL and PgBouncer when installing via docker compose
- Added filters to SCA scan history page
- Updated project and author activity maps, as well as complexity and duplicates map: changed image file name when downloading, removed captions in cells, improved scaling, fixed rendering errors
- Changed handling of sensitive data such as tokens, keys and passwords in API and UI of the system
- Changed the logic of how filters work throughout the system. Filters are now loaded on demand (lazy load), optimised part of requests. When returning to the page, filters are not loaded again
- Changed adding a user or project to a group: existing ones will not be offered for selection
- Updated OpenAPI specification for the References field in the `VulnerabilitySummaryDetail` type
- Added information to metadata tools section when uploading SBOM in CycloneDX format
- Changed autovacuum settings to lower thresholds for tables with frequent updates
- Added `max_client_conn` setting for pgbouncer, the parameter regulates the total number of connections, increased default value
- Changed validation of phone number field to support international numbers
- Changed output of parent dependencies in the project dependencies table, only the first 5 values are shown
- Changed output of events in the webhooks table, only the first 5 values are shown
- Fixed sorting of image vulnerabilities by Fixed Version
- Fixed export of project data to CSV, reduced memory consumption
- Added masking of sensitive data in platform logs
- Improved Russian localisation
- Fixed error output in UI when trying to create an existing project
- Fixed access rights restriction errors
- Fixed animation when switching between tabs of the project editing form

#[2025.13.3] - 2025-05-07

- The behavior of policy ignores has been fixed when alerts outside the scope of ignoring projects and images were ignored
- Fixed an issue where a project could not be saved due to the new format of project scan schedules

#[2025.13.2] - 2025-04-23

- Added view of detailed information on vulnerabilities for platforms that only have the OSA module
- Added checking for connection to Gerrit via SSH
- Optimized memory utilization when calculating policies

#[2025.13.1] - 2025-04-08

- Added the ability to specify days of the week and time in the analysis launch schedule for the SCA and Secrets modules
- Changed the approach to specifying the date of the last artifact request in OSA to avoid increased disk load

#[2025.13.0] - 2025-03-28

- Support for specifying a database schema other than `public` via the `DATABASE_SCHEMA` environment variable has been discontinued. If this configuration is used, please refer to the how-to.
- The CodeScoring configuration in Docker Compose has been deeply rewritten and modernized. Please read the how-to before upgrading.
- Added support for Swift Package Manager ecosystem manifests
- Added granular projects configuration and groups in policy actions to send notifications to different email addresses or different projects in Jira within a single policy
- Added modes for sending email notifications and creating issues in Jira within policy actions: one per alert or digest per scan
- Added processing of secrets analysis results when working with a module via CLI using the johnny console agent
- Added the ability to recalculate secrets information in the ML model management section
- Added basic work with the history of secrets scans
- Added the "does not match" operator in dictionary policies
- Added module icons to the system menu
- Added hiding of the API token on the user settings page
- Added returning of the uuid of the blocked component in the OSA API in a separate field
- Added detailed display of the password validation error when creating a new user
- Added detailed display of the password validation error in the password change form
- Fixed the filter by project name in the `Settings -> Policy ignores` section
- Fixed display of package links in policy conditions in the Alerts section
- Fixed system behavior when receiving results from johnny without the `--save-results` key specifying a project, now the results will not be saved
- Fixed incorrect sorting by project name in project lists
- Added hiding of mutually exclusive Access Token and SSH Key fields in different connection settings to VCS to avoid incorrect validation
- Fixed incorrect display of the environment on the dependency graph
- Fixed the activity of the analysis start button for CLI projects without loaded dependencies
- Fixed an error in the logic of applying policies when using groups
- Optimized the speed of the policy page
- Fixed an error with possible duplication of vulnerabilities
- Optimized the mechanism for updating vulnerability information to reduce the number of entries in the database

#[2025.7.2] - 2025-03-14

- Fixed OSA API service crash when accessing a non-existent key in Redis in version 2025.7.0
- Fixed display of pages with OSA module scan results if the package was requested periodically with an interval of less than a week
- Increased the period for requesting code updates in VCS projects to reduce the load on VCS

#[2025.7.1]-2025-03-07

- Fixed error in the operation of the mechanism of ignoring the Alerts
- Optimized background policy re-scan for OSA when updating information about vulnerabilities
- Fixed error in the operation of integration with OSSIndex
- Optimized the work of updating the date of the last request of the package or image to reduce the load on the disk

#[2025.7.0] - 2025-02-14

- Added ability to select a license in the dependency fields management UI
- Added ability to select a group when creating a policy
- Added ability to select projects in a policy without first selecting an owner
- Added a setting for enabling and disabling cloud resolve for projects in the SCA module
- Added advanced settings for VCS projects: ignores, enabling/disabling recursive search
- Changed the grouping and display of project settings
- Added the output of the Index API availability metric to the standard platform metrics tracking mechanism
- Added a setting via the environment variable `INDEX_API_FAILURE_RATE_THRESHOLD`, which determines how much failed requests to the Index API in the OSA module must occur before the system considers the index unreachable
- Added the `Skip TLS Verification` setting when creating a connection to the image registry
- Added webhooks for Secrets module events
- Added pre-connection to Postgres when the connection is lost in the osa-registration service
- Updated maps in the TQI module. Rendering has been moved to the frontend, more convenient navigation has been implemented, and additional filters by period and number of projects have been added
- Optimized dependency list in SCA modules
- Optimized request list in the OSA module
- Fixed saving of filter state and pagination settings in the dependency editing table
- Fixed a validation error when autofilling the Instance URL field when creating connections to VCS
- Fixed an error setting up columns in the project list in the SCA module

- Fixed translation errors when using numerals
- Fixed links to dependencies and vulnerabilities in the Email digest and Jira Issue
- Fixed incorrect behavior while testing connection configuration for Email server settings
- Removed the API method `get_package_info`

#[2024.52.2] - 2025-02-26

- Fixed the operation of the alert ignoring mechanism

#[2024.52.1] - 2025-01-24

- Fixed a bug that could lead to a sharp increase in container image scanning time
- Changed the approach to background policy recalculation for container images in the OSA module to optimize speed and resource consumption
- Added a separate queue for background package rescanning in the OSA module to unload the main queue

#[2024.52.0] - 2024-12-28

- Added separation of modules in the menu
- Added separate project lists in the SCA, TQI and Secrets modules
- Added support for the Conda ecosystem
- Added editing of container image dependencies for SBOM export
- Added multiple selection of projects and images in the creation of Policy Ignore
- Added the ability to specify a policy stage when creating a CLI project
- Added the ability to filter lists in the Vulnerabilities, Policy Alerts and Projects sections by multiple Severity, Policy and Technology values
- Added saving and displaying SBOM editing in the audit log
- Added displaying the name of CLI projects in the audit log
- Added filter by image tag to Container Images section
- Added dates of first and last SCA scan to projects list
- Implemented ability to add projects to existing groups via API, interface and console agent options for users with active flag **Can create CLI projects via API **
- The full display of the secret in the section has been moved to a separate Secrets window
- Updated translation into Russian
- Added validation of API token update
- Changed the format of the recommendation field in the SBOM CycloneDX format export to correctly handle cases where a vulnerability affects several versions of the same library
- Fixed an error creating a task in Jira when a policy is triggered
- Fixed an error filtering by status in Policy Alerts section when resetting filters
- URL input errors are now shown after input is complete

#[2024.48.1] - 2024-12-04

- Fixed a bug when starting an image scan from the UI

#[2024.48.0] - 2024-11-30

- Added the ability to send webhooks for key events in the system
- Added the ability for the admin user to specify values for the SBOM fields `GOST:attack_surface`, `GOST:security_function` and links to VCS, the values will be taken into account when unloading SBOM in the `CycloneDX 1.6 Ext` format
- Updated display of matched criteria in alerts
- Added the ability to display the Source files column in the Vulnerabilities section table and in the Affected dependencies table on the vulnerability page
- Added hints for the user in the policy creation and editing form
- Added links from the project scan results page to the project settings page and back
- Improved link typification in the `externalReferences` section when unloading SBOM in CycloneDX
- Accelerated loading of the license distribution graph
- Changed the technology distribution graph on the main page of the system and on the SCA tab for VCS projects, the calculation is based on the project dependency technologies based on the results of compositional analysis
- Fixed the logic of policies when combining several conditions for the environment (env) of a dependency
- Fixed import of SBOM files in CycloneDX format containing information in the `components[i].evidence.identity` fields
- Fixed translations into Russian for numerals and some dictionaries of the system
- In emails with alert notifications, the vulnerability identifier is now a hyperlink

#[2024.44.3] - 2024-11-13

- Fixed CVSS Score and CVSS Severity inconsistency when a vulnerability is present in multiple data feeds

#[2024.44.2] - 2024-11-07

- Accelerated loading of `Components -> Packages` and `Components -> Container Images` sections

#[2024.44.1] - 2024-11-05

- Added beta version of interface localization into Russian, language switching is available on the user profile page
- Added support for CycloneDX 1.6 specification for SBOM import and export

- Added export into CycloneDX 1.6 Ext format with the addition of the fields `GOST:source_lang`, `GOST:attack_surface` and `GOST:security_function` to comply with FSTEC of Russia requirements. The fields are filled with the default value
- For new SCA analysis results, the ability to select the CycloneDX version when downloading SBOM has been added
- Improved SBOM export into all CycloneDX versions: added information about the scanned application to `metadata->component`, added information about the platform version to `metadata->tools`, updated the outdated format for indicating the authorship of components for CycloneDX versions 1.5 and 1.6, fixed the format of the component license. Changes are available for new SCA analysis results
- Added display of dependency tree in PDF reports
- Added collection of data on malware from GitHub Security Advisory
- Added “Dangerous package” classification and corresponding policy for OSA module. Packages with known Malware and certain types of CWE in vulnerabilities are marked as dangerous
- Added additional dates to the package view page in the OSA module: dates of the first and last request to the package, date of the last policy calculation, and date of updating information on the package
- Added the `Source files` value to the vulnerability dump in the Vulnerabilities section
- Added policy conditions for case-sensitive search of a string in the package name `contains (case sensitive)`, and changed the names of case-insensitive conditions from `icontains` to `contains (case insensitive)`
- Added the `Has vulnerabilities` filter and a column with the number of vulnerabilities when viewing the list in the Components and Container images sections of the OSA module
- Added the ability to run mass analysis of secrets in Workmode
- Added processing of the new manifest type `application/vnd.docker.distribution.manifest.list.v2+json` when analyzing container images
- Added a table with projects that use the component to the component view page in the OSA module
- Added a new template `%USER_DN%` for the filter by groups when configuring LDAP
- Added the ability to start a package analysis from its page in the Components section
- Added a notification about the expiration of the activation key
- Fixed key columns in tables during horizontal scrolling
- Implemented a periodic restart of background tasks to optimize memory consumption
- Stabilized the launch time of scheduled analyzes
- Optimized updating of information on the secrets list page when marking up results
- Fixed errors in the behavior of some lists with multiple selection
- Fixed the display of user group records in the LDAP integration diagnostics section
- Fixed loading a list of container images from registries if metadata on some images could not be obtained
- Fixed errors in the operation of filters in the Secrets section table

- Fixed an error when trying to filter dependencies by `License Category = N/A`
- Fixed display of paginators on the SCA and TQI tabs on the project page
- Changed the configuration of connection pools to PostgreSQL. To optimize the memory consumption of the platform, a division of connections to Postgres into connections through connection pools operating in session and transaction mode has been implemented. If the system is installed via docker compose, it is necessary to update the `docker-compose.yml` file. When using custom connection pool configurations, please consult with the support service on the update process.

#[2024.40.1] - 2024-10-09

- Fixed display of repository branch/tag when selecting it in project settings

#[2024.40.0] - 2024-10-04

- Added beta version of the new Secrets module, access for testing can be requested from the vendor
- Implemented the dependence of filters on each other in the sections Policy Alerts, Dependencies and Vulnerabilities
- Added a display of Has Exploit to the vulnerabilities table and on the vulnerability page
- Implemented the ability to hide the filter block in sections with filtering
- Added Policy Alerts to PDF report on the project
- Added the possibility of sorting by the Fixed Version column in the vulnerabilities table
- Added sorting vulnerabilities in the PDF report
- Fixed recalculation of the dates for the criteria of the Dependency Age (Days) policy
- Fixed timing issue of the launch of daily analysis
- Fixed LDAP group matching if several LDAP groups are matched with one group in Codescoring
- Fixed incorrect display of the active tab on the dependency page
- Fixed the display of licenses in the Policies settings
- Fixed Last Updated field value for CLI projects
- Optimized the search for the names of groups in LDAP

#[2024.38.0] - 2024-09-18

- Fixed CSV export in the Dependencies section

#[2024.37.0] - 2024-09-12

- Added team table to the project page
- Added ability to export to CSV in the Components section
- Accelerated SCA analysis in cases where the same package occurs multiple times within a manifest

- Fixed an error when generating a project report if there are incorrect PURLs in SBOM.

#[2024.35.0] - 2024-08-29

- Added a new condition for policies - Dependency Vulnerabilities Count
- Added OSA metrics to the dashboard
- Added the number of blocking Policy Alerts to the dashboard
- Added the ability to run a project scan in the UI in parallel with the scan in Johnny
- Added a column with the number of Policy Alerts in the `Settings -> Policies` subsection
- Added a filter by the date of the last request in the `Components -> Container Images` subsection
- Implemented background generation of downloaded files and accelerated their download
- Implemented pagination when interacting with LDAP
- Accelerated work of the Dependencies section
- Supplemented the list of repository types so that all repositories, including unsupported ones, are displayed in the Repository Managers section

#[2024.32.0] - 2024-08-09

- Added support for Bearer authorization for Container Registries
- Added implementation of searching for LDAP groups through user attribute records
- Added repository type output in the Repository Managers section in cases where it is not supported
- Improved tools for diagnosing LDAP integration
- Fixed comparison of system package versions when calculating policies
- Speeded up the work of the `Components -> Packages`, `Components -> Requests` and Repository Managers sections
- Improved OSA requests cleanup

#[2024.29.1] - 2024-07-17

- Added an ability to specify a license for CLI projects when creating via the API
- Fixed an error while displaying Matched Criteria in the Policy Alerts section
- Fixed the color of the warning level on the blocked component page

#[2024.28.0] - 2024-07-09

- Implemented a new mechanism of policy conditions with the formation of logical expressions
- Optimized memory consumption when generating PDF reports
- Searching LDAP user attributes is now case insensitive
- Fixed sorting in the vulnerability table on the Container Image viewing page

- Fixed calculation of block status in OSA for selected components in a policy

#[2024.25.0] - 2024-06-18

- Added setting for matching LDAP group and user access level
- Added the ability to ignore Policy Alerts by PURL
- Added checking the availability of the Container Registry before starting an image scan in the interface
- Added description of OSA API methods in Swagger
- Improved platform stability in older browsers
- Accelerated loading of the `Components -> OSA Packages` section
- Fixed display of graphs and pagination of the commit table on the TQI tab of the project page
- Fixed the request date filter in the `Components -> Packages` section
- Fixed changing of an obsolete token for VCS
- Removed deprecated API method `/api/dashboard/security/top5_vulnerable_projects/`

#[2024.24.0] - 2024-06-11

- Optimized memory usage when loading vulnerability descriptions

#[2024.22.2] - 2024-05-28

- Added the ability to run platform with the readOnlyRootFilesystem option in Kubernetes
- Improved search speed in the Dependencies, Policy Alerts and Components sections
- Improved sorting by the Dependency/Package field in the Policy Alerts section
- Added cleaning of temporary files after forced completion of image scanning
- Fixed missing Distribution by technology graph in CLI projects after the first scan
- Fixed an error in generating a PDF report if there were links in PURL

#[2024.22.1] - 2024-05-27

- Changed the status in OSA plugins when receiving an unknown Repository Manager

#[2024.22.0] - 2024-05-27

- Fixed a bug when displaying the `Settings -> Workmode` section

#[2024.21.0] - 2024-05-24

- Added the ability to connect repository managers through the interface

- Added OSA package filter for repository managers and individual repositories
- Added policy settings for specific repositories
- Improved LDAP group mapping

#[2024.17.2] - 2024-04-27

- Added the package release date to the page in the `Components -> Packages` section
- A hint has been displayed for the Plugin mode column in the `Components -> Requests` section
- Fixed an error in opening the Policy Ignore editing page with a specified Container Image
- Fixed generation of links to commits in Gitlab

#[2024.17.1] – 2024-04-23

- Added the ability to specify the OSA type of the component for which the policy will be applied
- Added Group mapping section for configuring LDAP groups
- Added mapping of LDAP groups to internal CodeScoring groups
- Added links to vulnerabilities and licenses in the Matched criteria field of the Policy Alerts section
- Added search to the policy condition by the Technology field
- Added the Matched criteria field to the Policy Alerts table on the project page
- Returned display of the SBOM loading icon
- The License and Vulnerability columns in the Policy Alerts tables are now hidden by default
- Accelerated loading of the `Components -> Requests` section
- Improved periodic cleaning of OSA queries from the database
- Fixed redirect when deleting a user
- Fixed formation of Dependency Name from PURL

#[2024.15.0] - 2024-04-09

- Added the ability to configure multiple LDAP integrations
- Added Matched criteria output in the Container images section
- Added columns and filters Scan schedule, Scan with hashes to the Projects section
- Added columns and filters Scan schedule, Excluded from analysis to the `Settings -> Projects` section
- Added information about the previous user access level to the Audit Log
- Added support for non-standard ports for SSH when adding VCS
- Added percentage of similarity between authors in the Authors section
- Implemented periodic cleaning of OSA queries in the database
- Returned the dev policy stage to the project settings

#[2024.13.0] – 2024-03-27

- Fixed re-running of periodic tasks in the queue

- Fixed long wait for response from Index API in case of connection interruption

#[2024.12.0] – 2024-03-22

- Added Components section, which includes lists of OSA components and requests
- Added the Vulnerabilities field to the CSV project report
- Added pool_used metric to view busy connections from the pool
- Optimized launching analysis and unloading SBOM CLI project
- Fixed ignoring the load full images list flag when creating or updating a Registry
- Fixed display of the number of active policies in the Dashboard section
- Fixed missing options for CVSS2 Authentication policy
- Fixed an error when calculating Policy Alerts of a policy with the CVSS Score condition

#[2024.11.0] – 2024-03-18

- Added information about the package to the Policy alerts and Vulnerabilities sections on the page of the image blocked in OSA
- Added Group filter in the Vulnerabilities section
- Renamed the Blocks build policy attribute to Blocker
- Optimized analysis of CLI projects in situations where the dependencies of the analyzed project are often found in other projects
- Accelerated loading of information about similar authors
- Fixed display of connections at the same level in the project dependency graph

#[2024.10.0] – 2024-03-06

- Fixed the disappearance of the HTML upload button for charts
- Changed the formatting of commit messages in the project commits table
- Fixed display of metrics on project pages for which analysis was not carried out
- Fixed availability of downloading the PDF report and SBOM depending on the status of the project analysis

#[2024.9.1] – 2024-02-29

- Fixed case insensitivity when creating users from LDAP

#[2024.9.0] – 2024-02-28

- Added display of analysis status in history for projects and images
- Added display of commit messages in projects for past periods on the TQI page
- Expanded SCA metrics on the project page
- Fixed behavior when loading and scanning images without tags

- Fixed updating information about Container registry after saving changes
- Fixed inability to view project commits for the last week
- Fixed display of metrics on project pages for which analysis was not carried out
- Fixed integration with LDAP
- Reduced timeout when checking Registry availability

#[2024.8.0] – 2024-02-22

- Fixed launch of general analysis of clones
- Fixed display of paths to files inside clones
- Fixed transitions to lists of clones from the clone map
- Fixed display of the author when there is no email field
- Fixed display of hidden columns in tables
- Fixed display of the `block_status` variable in the list of the Container Images section
- Fixed updating the list of images when there are images in the registry without specifying the platform
- Removed empty value when selecting in filters
- Removed automatic restart of scheduled scanning in case of a crash

#[2024.7.0] – 2024-02-16

- Added vulnerabilities from the FSTEC BDU
- Implemented a page to display information about a component blocked in OSA
- Implemented launch of author analysis for one project
- Implemented the launch of clone analysis for one project
- Added a table of commits on the TQI tab of the project page
- Fixed filter for vulnerabilities in the Policy Alerts section
- Fixed transition to the policy page from the Dashboard section

#[2024.5.0] – 2024-02-02

- Updated system UI
- Added new SCA metrics on the project page
- Added policy condition to match PURL regular expression
- Added policy condition for searching a text string in PURL
- Added a check for hidden characters when entering the activation key
- Fixed filter by technology in the Projects section
- Fixed display of technologies for CLI projects in the Projects section
- Optimized policy recalculation mechanism
- Clarified error messages about connecting to GitLab
- Changed the orientation of the Vulnerabilities section in the PDF project report

#[2024.2.0] – 2024-01-12

- Added information about the architecture of scanned images
- Added Group filter on the projects page

- Added support for CycloneDX 1.5 specification when importing SBOM
- Added new blocking statuses for OSA components
- Improved OSA performance
- Accelerated loading of Complexity map in the Projects section
- Removed scans stuck in in progress status
- Fixed report export to PDF when actively scanning a project and for CLI projects

#[2023.49.0] – 2023-12-08

- Added the ability to export a PDF report with the results of scanning a project
- Added support for Rust dependency analysis via Cargo manifests
- Added the ability to clone a repository via SSH
- Added new VCS type "Other Git"
- Added display of the number of vulnerabilities found in the list of the Projects section
- Added search for nested vulnerabilities (those with entries in different feeds) in the Vulnerabilities section
- Added setting for displaying project ID in the `Settings -> Projects` section
- Added null values to Prometheus OSA API metrics
- Automatic analysis after cloning a VCS project has become optional
- Fixed an error when sorting the table by the CVSS3 Attack Complexity field in the Vulnerabilities section
- Fixed incorrect operation of OSA with packages that have uppercase in the version

#[2023.48.0] – 2023-11-22

- Reduced the size of supplied Docker images CodeScoring
- Added the Matched criteria parameter with the reason for the triggered policy to the Policy alerts section
- Added a new type of Container registries "Other"
- Added the right for users with User access level to create CLI projects via the API
- Added Prometheus metrics on scan status and component blocking status in OSA
- Added password and token hiding in Jira connection settings
- Removed duplicate vulnerabilities in SBOM that have different affected versions
- Removed obsolete endpoint API `/integration_api/v1/`
- Analysis work with a large number of Policy ignores has been accelerated
- Fixed UnsafeOption error when working with Azure and Bitbucket
- Fixed display of available values for the Container images filter in the Policy alerts section
- Fixed search by dependency in the Policy alerts section
- Fixed display of the CWE filter in the Vulnerabilities section

#[2023.44.0] – 2023-10-31

- Added saving filters between tabs in the Policy alerts section
- Fixed display of resolved Policy alerts on the project page

#[2023.43.0] - 2023-10-27

- Added support for scanning proxy Docker repositories based on Sonatype Nexus and JFrog Artifactory
- Added Digest and tags on the container image page
- Added background update of vulnerabilities and background operation of policies for components from container images
- Added metrics of number and time of requests for CodeScoring OSA
- The platform queue metrics have been supplemented - now you can separately view the types of analyzes in the queue
- Added a new policy condition – age of vulnerability
- Added the ability not to automatically download the list of images when adding a Container Registry
- Fixed opening the Policy Ignore view page
- Fixed display of git error 128 when working with VCS

#[2023.41.0] - 2023-10-09

- Running CodeScoring no longer requires superuser rights inside the container. Instructions for migrating from root containers to rootless are available from the vendor

#[2023.40.0] - 2023-10-04

- Added the ability to scan images from hosted Docker repositories based on Sonatype Nexus and JFrog Artifactory
- Added the ability to block downloading images from Sonatype Nexus Repository if they do not comply with security policies

#[2023.38.0] - 2023-09-20

- Added the ability to specify special characters in the password for connecting to the database
- Fixed an issue with displaying ignored alerts in the Policy Ignores section
- Fixed display of Source files and Parents fields on the dependency page
- Instant removal of an alert from the Active list after creating an ignore rule has been restored
- Added update scheme to the current version of the product.

#[2023.35.0] - 2023-08-31

- Added history of SCA scans in the project
- On the project page, information for the SCA and TQI modules is now displayed in separate tabs
- Added display of parts of CVSS vectors in the list of vulnerabilities

- Added a new policy condition "Vulnerability has fixed version"
- Added the CWE field to the CSV vulnerability table export
- Added CodeScoring banner and version logging to the console when launching a Docker container with platform
- Added metrics for the number of running analyzes for Prometheus
- Removed deprecated endpoint `/api/policy_alerts/ignored/`
- Removed deprecated endpoint `/api/policy_alerts/resolved/`
- Removed deprecated endpoint `/api/policy_alerts/`
- Fixed display of technologies on the Policy ignores list
- Fixed purl and CVSS policy for rpm
- Removed license text from generated SBOMs for the project

#[2023.31.0] - 2023-08-04

- Speeded up CodeScoring OSA

#[2023.30.0] - 2023-07-27

- Added fixed version in the Affected dependencies table on the vulnerability page
- Added queue status metrics for Prometheus
- Moved the platform version from the footer to the side menu
- Fixed inconsistency between Swagger schema and API
- Removed the ability to click the Upload SBOM button without an attached file

#[2023.26.0] - 2023-06-30

- Added support for system packages for the OSA plugin in NXRM
- Added new policy conditions - on the presence of a vulnerability exploit and parts of the CVSS vector
- Added output of policy name, stage and level to the Policy alerts list on the project page
- Added a filter by project type in the list of projects
- Marked the `/api/dependencies/csv/` method as deprecated. Now you need to use `/api/dependencies/by_project/csv/`
- Fixed an error when adding a project without first creating a VCS
- Fixed display of information about stages on the policy description card for the User access level

#[2023.24.0] - 2023-06-13

- Added a filter for searching by policy name in the Policies section
- Fixed dependency export with an empty filter

#[2023.22.0] - 2023-06-01

- Fixed loading SBOM in CycloneDX format obtained from `Microsoft.SBOMTool`

- Fixed processing of fractional vulnerability ratings when generating SBOM

#[2023.21.3] - 2023-05-30

- Fixed loading SBOM in CycloneDX format obtained using third-party tools
- Improved loading of Code Clones pages

#[2023.21.2] - 2023-05-23

- Added the ability to upload and download SBOM for a project in CycloneDX format
- Added the ability to perform analysis on a CLI project through the interface
- Added manual update of project code in settings
- Added the ability to connect several VCS with the same address, but different tokens
- Added basic auth support for integration with Jira
- Fixed display of the list of licenses on the chart
- Fixed the appearance of duplicate alerts under heavy load

#[2023.15.0] - 2023-04-14

- Added a new type of projects without reference to a repository - CLI
- Added Policy alerts grouping to digests for notifications via email
- Added search by policy name and dependency name in Policy alerts
- Added recommendations for eliminating vulnerabilities (fixed version)
- Added support for parsing packages in deb/apk/rpm formats
- Added a prompt to refresh the page if the client version is different from the server version
- Added minor improvements to the UI

#[2023.11.0] - 2023-03-16

- Fixed long loading of the Distribution by license graph on the Dashboard
- Added Note field to Policy ignore
- Added Description field to Policy
- Added a new Auditor access level
- Added session lifetime setting via environment variable
- Added links to the dependency graph

#[2023.6.0] - 2023-02-10

- Added project dependency graphs (link is on the project page)
- Added option to disable hash collection during SCA on platform
- Added Index API response cache for OSA (by default from 1 hour to 1.5 hours, configured through environment variables)
- Added information about restrictions on using OSSIndex
- Launch of mass SCA is now logged in Audit log
- Swagger no longer requires internet

- Changed the path to the statics from the backend (you need to fix `docker-compose.yaml`)
- Fixed a bug due to which in packages of the same name (with different versions) located in different manifests, information about the file in which the package was found was incorrectly displayed

#[2022.49.1] - 2022-12-07

- Fixed the work of the project page

#[2022.49.0] - 2022-12-07

- Added login events to Audit log
- Added processing of the unresolved node
- Added a new condition in policies for missing component version
- Changed platform API
- Fixed analysis when there is no env in the dependency graph

#[2022.48.2] - 2022-12-04

- Added support for modern yarn
- Changed the mechanism for classifying dependencies as direct
- Added unresolved section for dependencies that are in the lock file, but do not have a parent component

#Johnny Changelog

#[2026.20.1] - 2026-05-28

#Fixed

- Fixed SARIF report generation for results with no vulnerabilities
- Fixed `gl_dependency_scanning` report generation: added `severity` block filling

#[2026.20.0] - 2026-05-12

#Added

- Added support for trufflehog for secret scanning
- Added `--commit` flag for secrets when scanning directories
- Added Git mode for secrets
- Added flags for passing gitleaks and trufflehog configurations
- Added support for the ld.lld linker in scan build commands
- Added handling of the `GOST:provided_by` component property during SBOM import
- Added handling of the `inner_source` component property during SBOM import

- Added handling of the `source-distribution` component property during SBOM import
- Added support for `Directory.Packages.props` files for NuGet projects
- Added `--include-envs` and `--exclude-envs` flags to filter direct dependencies involved in analysis by environment
- Added support for `pdm` manifests: `pdm.lock` and `pylock.toml`
- Added dependency resolution support for `pdm`
- Added support for the Streebog algorithm when importing SBOM in CycloneDX 1.6 format
- Added support for the Joern callgraph for the JavaScript language

#Changed

- Updated package filtering during SBOM import so that unsupported and unknown ecosystems now have generic type
- Changed PURL generation for components from the `--lib-versions` file in C/C++ build analysis: now uses the generic type
- Updated root process PID detection to account for userspace in the scan build ebpf command

#Fixed

- Fixed report generation format for secrets
- Fixed error output during SBOM validation
- Fixed error message when attempting to add a project to a group it already belongs to
- Fixed calculation of delayed blocking policy status
- Fixed detection of root dependencies when scanning pipdeptree output
- Fixed error when importing SBOM with a license not in the list of supported ones: updated the license list

#[2026.11.1] - 2026-04-16

#Fixed

- Fixed group creation when a group name was matching with name prefixes of a number of already created groups
- Fixed parent package discovery for go resolve
- Fixed a csv policy alerts report creation error

#[2026.11.0] - 2026-03-10

#Added

- Added CVSSv4 support in the analysis results
- Added handling of deferred policy blocking
- Added output of ignored components during SBOM scanning
- Added vulnerability reachability analysis for C#
- Added a Windows build for Svace module (since version 5.0.260311)
- Added support for remote analysis for Svace module (since version 5.0.260311)

#Changed

- Changed behavior of the `--project-group` flag: when used with an existing project, that project is now added to the specified group
- Improved library lookup on RPM-based operating systems in `scan build` and `scan build ebpf` commands

#Fixed

- The agent no longer attempts to create a project if the project already exists and the user has no permissions to interact with it
- Fixed a SARIF generation issue where the project path was set when a component location was missing
- Fixed dependency environment detection in `package-lock.json`
- Fixed incorrect `security-severity` output when generating results in SARIF format
- Fixed output of vulnerable functions when a vulnerable function calls itself
- Fixed output field names from `line_start` and `line_end` to `start_line` and `end_line` according to the GitLab schema for CI/CD Secrets vulnerabilities
- Fixed a cache handling issue for Svace module (since version 5.0.260311)
- Fixed an analysis error when a server address is specified with a trailing slash for Svace module (since version 5.0.260311)

#[2026.3.2] - 2026-01-28

#Fixed

- Fixed display of policy conditions for license incompatibility alerts
- Fixed workspaces field parsing while scanning `package-lock.json`
- Removed redundant 'saved to' info when saving reports to files
- Fixed secrets scanning related failure
- Fixed root manifest handling in the uv workspaces `pyproject.toml` block

#[2026.3.1] - 2026-01-14

#Changed

- Disabled checking of `externalReference.url` values for compliance with the iri-reference format during SBOM validation before scanning

#[2026.3.0] - 2026-01-13

#Added

- Added support for the lockfile text format for the bun package manager (1.2 and later)

- Added support for the Svace module call graph for the Kotlin language
- Added the `--create-project-group` flag for creating a group and adding a project to it
- Added the `--project-categories` flag for specifying the categories of the project being created
- Added the `--localization` flag for localizing the agent's output (coloredtable, table, text, csv formats)
- Added the `--gitleaks-config` flag of the `secrets gitleaks` command to pass the path to the gitleaks config file
- Added the `--policy-ignores` and `--ignores-format` flags for outputting policy ignores in the specified format
- Added a build of the Johnny console agent for Linux with ARM processors
- Added validation for SBOM scanning (supported Formats: 1.4, 1.5, 1.6, 1.6_ext, 1.7)
- Added support for exporting SBOM in 1.7 format
- Added correct handling of transitive dependencies without purl when importing SBOM
- Added coordinates of the detected and monitored secret
- Added output of the vulnerability impact when exporting the results to CSV format

#Changed

- Changed the severity in the secrets report to `Critical` for each secret, for compatibility with the gitlab report
- Changed paths in secret scan results from absolute to relative
- Changed the output of the list of agent processed manifests. All manifests fall into it, even if no dependencies are found in them
- The output of general information on the analysis result was changed. Information is always output to the console, even when the output is redirected to a file
- Improved Linux OS detection logic to better identify RHEL family OS in `scan build` command
- Changed handling of libraries that failed ldconfig validation in `scan build` command. They are added to the unresolved list

#Removed

- Discontinued publishing of console agent images with the `{VERSION}-busybox` tag

#Fixed

- Fixed a panic when importing some SBOMs

#[2025.45.4] - 2025-12-12

#Fixed

- Fixed a loop on some JavaScript projects when processing the workspace

#[2025.45.3] - 2025-12-01

#Changed

- Optimized gradle-dependency-tree processing

#[2025.45.2] - 2025-11-19

#Added

- Added output of the Summary field when exporting scan results in CSV format

#[2025.45.1] - 2025-11-14

#Fixed

- Fixed relative paths to manifests in scan results for .NET, uv, and npm/pnpm projects
- Fixed `packages.lock.json` association mechanism in projects using sln
- Fixed image scan results output

#[2025.45.0] - 2025-11-05

#Added

- Added vulnerability reachability analysis for Go
- Added vulnerability reachability analysis for Python
- Added support UV package manager for Python ecosystem
- Added printing matched criteria in policy alerts (starting from platform version **2025.45.0**)
- Added flags `--branch-or-tag` and `--commit` in `scan bom` command
- Added support workspaces mechanism for npm and pnpm
- Added unloading of the HasExploit flag to the junit and gitlab formats
- Added using dpkg for define information about static libraries in `scan build` и `scan build ebpf` commands
- Added support `pipdeptree` for Python ecosystem via the `--pipdeptree-resolve` option
- Added printing paths to manifest inside scanned image

#Changed

- Changed printing of reachability paths on table format. Printing top 5 paths and hidden count
- Optimized printing of reachability paths

#Removed

- `scan build` and `scan build ebpf` commands removed from not Linux binary

- Building scratch images of the CLI agent discontinued. All tags links to busybox version

#Deprecated

- Building images of the CLI agent with tag `*-busybox` will be discontinued in release 2026.3.0

#Fixed

- Fixed different result in processing npm with `--npm-resolve` and with existed `package-lock.json`
- Removed duplicates vulnerabilities in reachability graph
- Fixed printing path to lock file when using local resolve
- Fixed printing policy alerts: added validation to exclude empty objects
- Fixed processing of `gradle-dependency-tree.txt`
- Fixed processing of comments in some manifest types
- Removed duplicates in `scan image` command with `--scan-files` flag

#[2025.37.2] - 2025-10-06

#Fixed

- Fixed an error determining paths to the `.csproj` and `project.assets.json` manifests when scanning .NET projects

#[2025.37.1] - 2025-09-22

#Fixed

- Fixed local gitleaks version validation issues

#[2025.37.0] - 2025-09-08

#Added

- Added vulnerability reachability analysis for Java and Svace call graph, as well as output of reachability examples in SARIF (starting from platform version **2025.37.0**)
- Added the ability to run local scans without specifying activation parameters (`--api_url` and `--api_token`), generating an SBOM based on detected manifests without enrichment and policy application
- Added support for `deps.json` and `sln` manifests for the .NET stack
- Added support for all component types from the PURL specification, including the generic type, as well as components with empty or invalid PURLs when analyzing SBOM with the `scan bom` command
- In the `scan bom` command, added pre-conversion of the file to UTF-8 encoding for correct further processing
- Added the ability to run `sign bom` and `verify bom` commands without specifying activation parameters (`--api_url` and `--api_token`)

- Added transfer of annotated data (`GOST:attack_surface`, `GOST:security_function`, `GOST:source_langs`, `VCS`, `licenses`) to the platform when importing annotated SBOM (starting from platform version 2025.37.0)
- Added the ability to specify branch/tag and commit when scanning an image
- Added the option to select the progress bar type: `spinner` or `text` (default: `spinner`)

#Changed

- Changed the logging level for policy triggers from error to warning
- Changed the display of the number of alerts for triggered policies
- Changed the resolution logic in the JavaScript environment: local resolution is not performed if any of the known lock files (`package-lock.json`, `yarn.lock`, `pnpm-lock.yaml`) are present
- Improved linking mechanism for Gradle when using arbitrary manifest names. Now `build.gradle` and `gradle.lockfile` are automatically linked even if their names differ. If `gradle-dependency-tree.txt` is present, it takes priority, and other files are processed separately
- Changed the order of checking the ability to run a command and obtain analysis results according to the platform license. This now happens before the scan is performed

#Fixed

- Fixed processing of `requirements.txt` containing a comment about another dependency source

#Deprecated

- Building scratch images of the CLI agent will be discontinued in release 2025.45.0

#[2025.29.3] - 2025-08-22

#Fixed

- Fixed argument forwarding to package managers when using the local resolve mechanism in the absence of a lockfile.

#[2025.29.2] - 2025-07-25

#Fixed

- Fixed crash when processing dependencies for which the environment could not be determined

#[2025.29.1] - 2025-07-18

#Fixed

- Fixed error when exporting results in `junit` format

#[2025.29.0] - 2025-07-16

#Added

- Added parsing of dependencies declared in unified format in `build.gradle`
- Added `sign bom` command to sign SBOM files
- Added `verify bom` command to verify the authenticity of the SBOM file signature
- Added work with the predefined `CodeScoring_All_Dependencies` task for correct dependency resolution in multi-module projects of the gradle environment
- Added the `project-proprietor` parameter to link the scanned project to the department (since version **2025.29.0** of the platform)
- Added support for aliases for `yarn.lock` and `pnpm-lock.json`
- Added support for reports for alerts in the following formats: coloredtable, table, text, json, csv. The format is controlled by the `--alerts-format` parameter
- Added `--branch-or-tag` and `--commit` flags to the `scan build` and `scan build ebpf` commands
- Added unloading of the HasExploit flag to the sarif format
- Added output of license information to the text, table, coloredtable formats
- Added unloading of Relation, Parents, Match type, Env data to the CSV format
- Added the ability to pass flags to package managers when resolving dependencies
- Added output of a warning about parsing errors during scanning
- Added support for dependency groups with an arbitrary name in `pyproject.toml`
- Added the ability to pass the SHA hash of an image in the `--hash` parameter of the `scan image` command (since version **2025.29.0** of the platform)
- Added a check for the availability of the `dir` command of the local version of `gitleaks`

#Changed

- Disabled running the parser in the pip environment by default in the `scan python` command. Enabled explicitly with the `--pip-resolve` flag
- Improved the performance of the `scan build ebpf` command

#Fixed

- Fixed ignoring behavior for empty `--ignore` parameter values

- Fixed an error in determining the relation dependency when parsing a pair of `package.json` and `yarn.lock` manifests
- Fixed determining the environment in cases where a dependency of the same version is present in multiple environments
- Fixed `poetry-core` from the `build-system` section getting into the list of dependencies when parsing the `pyproject.toml` manifest

#[2025.21.0] - 2025-05-21

- Added `scan build ebpf` command to scan C/C++ project builds using eBPF
- Added export to sarif of data on dependency relationships within a project, direct or transitive, in the format: `results.properties.relation: direct|indirect`
- Added ignoring of commented lines when parsing `conanfile.py` files
- Fixed version detection from requirements of the type `==3.0.0.post1` in Python manifests
- Fixed upload of vulnerabilities to sarif that have criticality specified without a numerical assessment
- Fixed parsing in the Go environment: transitive dependencies for which it was not possible to determine the parent package are excluded from the scanning results

#[2025.13.0] - 2025-03-28

- Added support for parsing Swift ecosystem manifests: `Package.swift` and `Package.resolved` (starting with version 2025.13.0 of the platform)
- Added a beta version of the console agent with the Secrets module (starting with version 2025.13.0 of the platform)
- Added processing of corrupted `scala-dependency-tree.txt` files
- Added parsing of dependencies declared in an unmerged format in the `build.gradle.kts` file
- Added ignoring `.nuspec` files in the `scan csharp` command
- Added support for AltLinux operating systems in the `scan build` command
- Excluded Java archives from scanning when the `--scan-archives` flag is inactive
- Fixed import SBOM in which the library has multiple values of the env property
- Fixed processing of SBOM files in CycloneDX format containing information about components within components

#[2025.7.0] - 2025-02-13

- Added commands for scanning a directory with predefined settings depending on the selected technology (for example, `./johnny scan java`)
- Added output of information about the presence of Exploit for vulnerabilities in the agent's results
- Added the `--cloud-resolve` parameter to activate cloud resolver (compatible with platform of version 2025.7.0 and higher)

- Added support for the Selective dependency resolutions mechanism for Yarn
- Added support for the NPM Package Aliases mechanism for `package-lock.json`
- Optimized processing of large `gradle-dependency-tree.txt` files
- Fixed a bug in determining package versions in the `gradle.lockfile` file if there are suffix versions

#[2024.52.2] - 2025-01-23

- Fixed agent behavior that caused platform's tasks-policy queue to grow

#[2024.52.1] - 2025-01-16

- Fixed handling of multiple `gradle.lockfile` files per module

#[2024.52.0] - 2024-12-24

- Added scan build command for build analysis for C and C++ languages
- Added new formats for exporting work results: GitLab Dependency Scanning Report and GitLab Code Quality Report
- Added handling of the `--ignore` parameter when scanning archives and files inside images
- Added the ability to specify a link to a branch/tag and commit using the `branch-or-tag` and `commit` parameters when scanning a file and directory (when interacting with platform version `2024.52.0` and higher)
- Added the ability to specify a hash using the `hash` parameter when scanning images (when interacting with platform version `2024.52.0` and higher)
- Added the ability to specify a policy stage when creating a CLI project (when interacting with platform version `2024.52.0` and higher)
- Added specifying paths to manifests inside scanned images where information about a vulnerable package was found
- Added paths to manifests where a vulnerable package was found in `sarif` format
- Fixed crash when processing an incorrect file in `yaml` format
- Added handling of an error that occurs when a file was deleted during scanning
- Fixed the presence of extra characters when unloading in `sarif` format
- Fixed environment detection when parsing Poetry manifests
- Fixed scanning of RedHat-based images

#[2024.48.2] - 2024-12-13

- Fixed crash when processing some `gradle-dependency-tree.txt` files
- Fixed parsing of npm and yarn lock files paired with a manifest

#[2024.48.1] - 2024-12-10

- Added support for multiple versions of a single package in the `poetry.lock` file
- Added the ability to run without waiting for analysis results (`--no-wait` parameter)

#[2024.48.0] – 2024-11-29

- Added support for parsing Conda ecosystem manifests: `environment.yml`, `meta.yml`, `conda-lock.yml`
- Added support for parsing Conda components in the build environment
- Added warning output for packages with an invalid name
- Improved dependency graph construction for formats that allow multiple versions of a single package
- Improved dependency graph construction when both files of a manifest-lockfile pair are present
- Fixed errors in generating PURL and go package versions when scanning Docker images
- Fixed handling of SBOM files in CycloneDX format containing information in the `components[i].evidence.identity` fields
- Changed the logic for generating the distro property for PURL ALT Linux packages when scanning Docker images
- Added information about Location and Fixed Version of a vulnerability to the `sarif` format export

#[2024.44.1] - 2024-11-15

- Fixed a bug with skipping gem packages in the `scan bom` command
- Fixed the `ignore` flag on Windows OS
- Fixed a bug in the parser in the Go environment on projects without dependencies

#[2024.44.0] - 2024-11-02

- Added parsing of `pnpm-lock.yaml` manifests. Supported versions: 5.0-5.4, 6.0, 9.0
- Added parsing in the pnpm environment
- Takes into account the use of the `pnpm-workspaces.yaml` configuration file when parsing `package.json`
- Added the ability to specify a group when creating a CLI project, for admin role only
- Added the ability to specify the format of the generated SBOM using the `--bom-format` parameter (starting with on-premise version 2024.44.1)
- Implemented parsing in the pip environment
- Implemented parsing in the composer environment
- When resolving dependencies in the go environment, the mechanism for determining the parent library for transitive dependencies obtained from the test environment has been improved
- Fixed the `unsupported type` error for composer components in the `scan bom` command

#[2024.40.2] - 2024-10-18

- Fixed dependency graph construction in cases where a component occurs multiple times with different `bom-ref`

#[2024.40.1] - 2024-10-10

- Added merging of `pom.xml` and `mvn-dependency-tree.txt` parsing results to avoid unnecessary dependency resolution
- Fixed an error in checking for the presence of a lock file when using dependency resolution in the environment

#[2024.40.0] - 2024-10-02

- Added workspaces parsing when working with npm manifests
- Fixed `Gemfile.lock` parser for cases with multiple Gem sections

#[2024.39.0] - 2024-09-23

- Separated tags when unloading in sarif format to display all versions of the found package in DefectDojo
- Changed export of severity in sarif format to correctly display CVSS3 in DefectDojo
- Fixed issues with scanning SBOM containing Go packages
- Fixed panic when parsing empty `cargo.lock`
- Removed duplication of vulnerabilities in sarif format for cases of multiple versions of the same package
- Removed the ability to simultaneously use the `format` and `no-summary` flags

#[2024.36.0] - 2024-09-05

- Added the ability to configure the used parsers via the configuration file
- Added the ability to specify the parser used in the scan file command
- Fixed parsing of multi-project/module gradle-dependency-tree

#[2024.35.0] - 2024-08-20

- Fixed gradle-dependency-tree kotlin parsing

#[2024.32.0] - 2024-08-09

- Added analysis of standard go libraries to the parser in the go environment (`--go-resolve`)
- Added the ability to specify a license when creating a project
- Fixed an error when parsing `pom.xml`, which contains variables like `xxx.xxx.xxx.xxx`
- Fixed `scala-dependency-tree.txt` parser
- Fixed an error when scanning SBOM without a component section

#[2024.29.0] – 2024-07-19

- Added export of links and CWE to sarif format

#[2024.26.0] - 2024-06-24

- Added parsing in npm environment
- Added parsing in dotnet environment
- Added parsing in the poetry environment
- Added launch parameter `--block-on-empty-result` (returns code 3 if the scan result is empty)
- Added `--python-version` flag to specify the python version in the pypi manifest family
- Fixed the construction of the dependency graph on the pair `package.json` and `package-lock.json`
- Improved parsing of `project.assets.json`

#[2024.21.0] - 2024-05-24

- Improved yarn.lock parsing
- Fixed parsing in yarn environment

#[2024.17.0] – 2024-04-27

- Added Johnny build for Mac with Intel processors
- Fixed scala-dependency-tree parser

#[2024.15.0] – 2024-04-11

- Added support for uploading scan results in CSV format
- The path to the source file in which the dependency was found has been added to the scan result upload
- Improved search for .net packages when scanning images

#[2024.13.0] – 2024-03-28

- Added support for uploading scan results in SARIF and XML formats

#[2024.10.2] – 2024-03-07

- Fixed merging lock files with manifests on Windows

#[2024.9.0] – 2024-02-29

- Fixed crash when parsing go.sum

#[2024.7.0] – 2024-02-12

- Reduced the size of the Docker image with the agent

- Fixed a bug when hashing empty files

#[2024.5.0] – 2024-01-31

- Added Scala support
- Added dependency resolution in the go environment (`--go-resolve`)
- Added dependency resolution in the maven environment (`--maven-resolve`)
- Added dependency resolution in yarn environment (`--yarn-resolve`)
- Improved error messages in query parameters
- Added platform variables URL (`cli.api_url`) and TOKEN (`cli.api_token`) to the config
- The summary now counts the number of vulnerabilities, not packages
- Increased the width of tables when it is impossible to determine the width of the terminal

#[2023.49.0] – 2023-12-08

- Added support for parsing Rust manifests `cargo.lock` and `cargo.toml`
- Added `--no-recursion` parameter to disable recursive scanning of the scan dir command

#[2023.48.0] – 2023-11-22

- Added setting the output format of the table with results `-f --format` (with the ability to turn off colors)
- Added setting for grouping vulnerabilities in the output of `-g --group-vulnerabilities-by`
- Added setting for sorting vulnerabilities in the output of `-s --sort-vulnerabilities-by`
- Added setting of analysis timeout limit `-t --timeout`

#[2023.43.0] – 2023-10-27

- Added summary information about the severity of vulnerabilities to the console output
- Fixed parsing of `.gradle.kts` manifests

#[2023.38.0] - 2023-09-20

- Improved parsing of package.json and composer.json manifests

#[2023.35.0] - 2023-08-31

- Improved parsing of the environment field for the `Gemfile` and `Gemfile.lock` manifests
- Removed automatic merging of cells with the same CVSS value in the table with vulnerabilities

#[2023.33.0] - 2023-08-17

- Optimized output of tables to the console on small screens

#[2023.30.0] - 2023-07-27

- Added `conanfile.py` parsing for Conan
- Added indication of the active analysis process in the form of a progress bar
- Added a table display for displaying alerts and vulnerabilities in the console
- Unified processing of slash at the end of a line for the `scan dir` command

#[2023.27.0] - 2023-07-06

- Fixed panic when analyzing some Go projects
- Fixed scanning of images in terms of incorrect detection of components that are not dependencies

#[2023.26.0] - 2023-06-30

- Improved parsing of `gradle-dependency-tree` in terms of working with classPath strings
- Fixed output of Policy Alerts to the console

#[2023.23.0] - 2023-06-08

- Added parsing of different versions of the `conan.lock` format
- Fixed the parser flag being reset when reaching an empty line in `conanfile.txt`
- Fixed `yarn.lock` parsing

#[2023.21.0] - 2023-05-23

- Added launch parameter `--scan-depth` to configure archive scanning depth
- Added `--scan-files` flag to the scan image command to scan files inside a docker image
- Improved detection of nested dependencies of jar packages
- Fixed `Gemfile` parsing

#[2023.15.0] - 2023-04-14

- Added Fixed version output
- Added the ability to save scan results
- Added the ability to create a project
- Added search for system dependencies in the docker image
- Optimized parsing of `package-lock` v3 manifest for NPM
- Fixed some bugs

#[2023.11.0] - 2023-03-16

- Added support for console commands
- Improved parsing of `pyproject.toml`
- Added cleaning of the /tmp directory after scanning the docker image

#[2023.5.0] - 2023-02-01

- Added scanning of docker images
- Changed behavior when starting without a project
- Updated Golang to 1.19
- Fixed hashing in archives with the `--only-hashes` option
- Fixed detection of broken and password-protected archives

#[2023.3.0] - 2023-01-20

- Fixed a bug when parsing `gradle-dependency-tree`

#[2023.2.0] - 2023-01-13

- Added scanning of archives, flag to run `--scan-archives`

#[2022.52.0] - 2022-12-30

- Fixed parsing of `pom.xml` in terms of working with the dependencyManagement section

#[2022.50.0] - 2022-12-12

- Added support for parsing `conan.lock` files
- Fixed passing additional data for the resolver from Nuget manifests

#Nexus OSA Changelog

#[2026.19.0] - 2026-05-06

#Features

- Added the `CodeScoring Repository Mask Scan` Capability for configuring repository scanning by regex mask
- Added the `Repository exclude masks (regex, comma-separated)` field to the `CodeScoring All Repositories Scan` Capability to exclude repositories from scanning by regex masks

#[2026.8.0] - 2026-02-20

#Features

- Added handling of non-blocking `delayed_block` status
- Added `Append block URL to custom message` setting to control adding a block URL to a custom message
- Added support for conda repositories

#Fixes

- Fixed an issue with saving `Configuration` Capability: if `Pool Size` and `Timeout` fields are empty, changes are now saved correctly after disabling the `Block downloads in case of plugin...` flag
- Fixed an issue with scanning Maven packages where artifactId contains the version

#[2025.32.0] - 2025-08-06

#Changed

- Plugin now allows image downloads when registry is not configured in CodeScoring platform if the `Block downloads in case of plugin or CodeScoring errors` flag is disabled.

#[2025.23.0] - 2025-06-02

- The general plugin configuration now includes the option to set a timeout for waiting for a response from the platform

#[2025.12.0] - 2025-03-17

- Improved accuracy of analysis in RPM and Debian repositories: improved detection of namespaces, distribution and architecture qualifiers to PURLs
- Added Circuit Breaker mechanism to the plugin to prevent Nexus performance degradation in case of unavailability or timeouts on the platform side

#[2025.9.0] - 2025-02-25

- Fixed the Docker image link sent by the plugin for analysis in the CodeScoring platform. The link format changed in Nexus Repository 3.75 release

#[2025.5.2] - 2025-01-31

- Fixed a 500 error when attempting to scan an unknown repository format plugin

#[2025.5.1] - 2025-01-27

- Fixed the missing namespace in PURL for NPM packages in Nexus Repository 3.75 release

#[2025.5.0] - 2025-01-27

- Fixed an issue with spaces in the ignored repository names list and the repository formats list for scanning

#[2024.49.0] - 2024-12-05

- Added sending to CodeScoring a link to the package and the user downloading it. Compatible with platform version 2024.48.0 and higher
- Added support for Nexus 3.75

#[2024.46.0] - 2024-11-15

- Added support for PHP Composer repositories. Available when using community plugin
- Updated support for plugin compatibility with different versions of Nexus Repository: -for Nexus Repository OSS **3.71+** and Nexus Repository Pro versions from **3.33.1-01** to **3.71+**, a version of the plugin with H2 and PostgreSQL support has been released -for Nexus Repository OSS versions from **3.33.1-01** to **3.70.X**, a legacy version of the plugin with OrientDB support has been allocated

#[2024.42.0] - 2024-10-16

- Added ability to specify format of repositories to analyze using the `All Repositories` Scan capability

#[2024.34.0] - 2024-08-19

- Added support for Nexus 3.71

#[2024.28.0] - 2024-07-10

- Added scanning of archives in `.gem` format for Ruby repositories

#[2024.26.0] - 2024-06-27

- Added the `Append repository name to image name for Docker repositories` setting to allow working with the RepoPath approach for Docker registries
- Removed the ability to manually start scanning the entire repository from the plugin side (flag `Run manual scan on save`)

- The `Block downloads in case of plugin or CodeScoring errors` setting now takes into account scanning failures (status `blocked_scan_failed`)

#[2024.21.0] - 2024-05-24

- Added Capability, which activates scanning for all repositories (`CodeScoring All Repositories Scan`)
- Added the ability to specify your own blocking message
- Added a link to the reason for blocking in the component properties
- Lowered the logging level for the message about skipping a repository scan
- Added repository manager URL setting

#[2024.7.0] – 2024-02-17

- Added a link to the component page in CodeScoring in the blocking message
- Added blocking message for situations where registry is not added to CodeScoring

#[2024.2.0] – 2024-01-12

- The list of image architectures has been expanded
- Added support for new component blocking statuses

#[2023.48.0] – 2023-11-22

- Added new operating mode `spectator`
- Added support for scanning multi-platform Docker images

#[2023.44.0] – 2023-10-31

- Added support for scanning proxy Docker repositories based on Sonatype Nexus and JFrog Artifactory

#[2023.43.0] – 2023-10-27

- Added various plugin operating modes

#[2023.40.0] - 2023-10-04

- Added Capability for checking images from Hosted Docker Repository

#[2023.37.0] - 2023-09-11

- Fixed an error with saving scan results when required fields were missing

#[2023.36.0] - 2023-09-05

- Added an option to block the build when receiving an error interacting with the CodeScoring platform

#[2023.33.0] - 2023-08-15

- Added the option to save artifact scan results to the Nexus database
- Added a method to the API to retrieve artifact scan results

#[2023.26.0] - 2023-06-30

- Added support for scanning open-source components in hosted repositories

#[2023.6.2] - 2023-02-09

- Added proxy server setting in the plugin configuration

#[2023.6.0] - 2023-02-07

- Added the HTTP Client Connection Pool Size parameter in the plugin configuration to control the number of platform requests

#[2023.4.0] - 2023-01-24

- Improved parsing of RubyGems components

#[2022.51.0] - 2022-12-20

- Improved logging of platform responses

#JFrog plugin changelog

#[2026.16.0] - 2026-04-14

#Added

- Added the ability to configure repository scanning parameters using regular expression masks (`repositoryMasks` and `excludeRepositoryMasks`). See documentation for details

#Fixed

- Fixed a compatibility issue affecting several JFrog Artifactory versions, including 7.77.x, caused by incorrect ClassLoader behavior in those versions

#[2026.8.0] - 2026-02-20

#Added

- Added support for Conda repositories
- Added support for Conan repositories
- Added support for Swift repositories

#Changed

- Changed the application of policies depending on the type of repository to which the request was sent (virtual/remote): the actual repository name is now sent in the request to the platform

#[2026.4.0] - 2026-01-20

#Fixed

- Fixed a compatibility issue with JFrog Artifactory version 7.125.x and higher

#[2025.43.0] - 2025-10-23

#Added

- Implemented non-blocking `delayed` status

#Fixed

- Fixed a bug with duplicated requests for non-blocked packages
- Fixed an incorrect transition of Circuit Breaker to the OPEN state when the server silently closes connections from the plugin's connection pool

#[2025.32.0] - 2025-08-06

#Changed

- Plugin now allows Docker image downloads when registry is not configured in CodeScoring platform if the `blockOnErrors` configuration flag is set to `false`

#[2025.12.0] - 2025-03-17

- Improved accuracy of analysis in RPM and Debian repositories: improved detection of namespaces, distribution and architecture qualifiers to PURLs

#[2025.11.0] - 2024-03-13

- A Circuit Breaker mechanism has been added to the plugin to prevent Artifactory performance degradation in case of unavailability or timeouts from the on-premise platform

#[2024.49.0] - 2024-12-05

- Added new execution `codeScoringVersion`, allowing to find out the plugin version. To do this, run the command:
  `curl http://localhost:8082/artifactory/api/plugins/execute/codeScoringVersion` .
  If the version is not displayed, restart JFrog

#[2024.48.0] - 2024-11-29

- Added sending to CodeScoring a link to the artifact and the user downloading it. Compatible with platform version 2024.48.0 and higher
- Added support for JFrog Artifactory v7.90+
- For deb packages, improved information retrieval in case of using non-standard separators in the file name
- Visually highlighted plugin initialization and config reload in the logs

#[2024.42.0] - 2024-10-16

- Packages that do not contain a version are no longer sent for analysis to CodeScoring and will be skipped by the plugin
- Fixed parsing of deb versions for alternative separators in the format `package-version-architecture.type`

#[2024.39.0] - 2024-09-26

- Added a setting to scan the selected repository type `repositoryTypes` (maven, npm, etc)
- Fixed crash in Artifactory 7.49

#[2024.33.0] - 2024-08-12

- Added support for `cargo` and `composer` repositories
- Added a flag for saving scan results to artifact properties

#[2024.28.0] - 2024-07-10

- Added scanning of archives in `.gem` format for Ruby repositories

#[2024.26.0] - 2024-06-28

- Added the ability to connect the plugin to all supported repositories without having to list them (`scanAllRepositories`)

- Added the ability to exclude repositories from the list of connected ones using the `scanAllRepositories` option
- Added `deleteBlocked` flag to delete a component if it is blocked by policies
- Added setting of component properties: scan date, blocking reason, link to the package page
- Detailed description of the configuration file
- Improved logging when skipping a scan
- The `blockOnErrors` flag now takes into account scan failure (status `blocked_scan_failed`)

#[2024.11.0] – 2024-03-15

- Added a link to the component page in CodeScoring in the blocking message
- Added a new message about component blocking in situations where registry is not added to CodeScoring
- Added setting for working with Docker registry Repository path (`stripRepoNameInDockerImageName`)
- Improved artifact name and version detection for PyPI, NPM, Debian and Alpine repositories
- Fixed scanning error for uncached images in remote docker repositories

#[2024.5.0] – 2024-02-02

- Added processing of zip archives in golang repositories
- Improved work with debian packages
- Fixed error checking alpine packages
- Fixed scanning of artifacts in virtual repositories

#[2024.2.0] – 2024-01-12

- The list of image architectures has been expanded
- Added support for new component blocking statuses
- Added logging of the request body for loading a component and the response
- Fixed bug blocking packages in debug mode

#[2023.50.0] – 2023-12-15

- Added new operating mode `spectator`
- Added support for scanning multi-platform Docker images
- The plugin configuration file now uses the `.yaml` format
- Added the ability to specify the operating mode for each repository separately
- Added the ability to change default values in the configuration file
- Added HTTP Client Connection Pool Size parameter to control the number of platform requests
- Added option to disable plugin

#[2023.48.0] – 2023-11-23

- Added config output to log when plugin starts

#[2023.43.0] – 2023-10-27

- Added various plugin operating modes
- Added support for scanning local/remote Docker repositories

###[2023.28.2]

- Added `CodeScoring:` prefix to all logs

#[2023.28.1]

- Reduced the level of all errors interacting with the CodeScoring platform to `[INFO]`

#[2023.28.0]

- Fixed a bug with the `blockDownloads` flag not working in properties

###[2023.27.0]

- Added the `blockDownloads` flag to properties, which allows you to control the download of a package if there are errors in the CodeScoring API or plugin

#[2023.26.0]

- Fixed saving the properties of a package when downloading it from a virtual repository

#[2023.22.0]

- Fixed a bug with the required `project_name` field in the OSA API

#[2023.21.0]

- The plugin delivery format has been changed: it is now supplied as a jar file
- Changed plugin initialization: instead of reloading for each request, there is a one-time initialization at startup

#Sfera.Distributions and licenses plugin changelog

#[2026.7.0] - 2026-02-13

#Fixed

- Fixed passing a tag instead of sha256 in PURL when using `docker pull` command

#[2025.52.0] - 2025-12-30

#Added

- Initial release of the Sfera plugin
- Support for the following repository formats:
  - Maven
  - NPM
  - NuGet
  - PyPI
  - RubyGems
  - Go
  - APT (Debian)
  - YUM (RPM)
  - Docker

#Visual Studio Code Extension Changelog

#[2026.8.0] - 2026-02-18

#Added

- Added a panel for viewing policy alerts

#Fixed

- Fixed errors in automatic fix application

#[2025.33.0] - 2025-08-14

- The extension can scan all CodeScoring-supported manifests, display vulnerability information, offer automatic updates to fixed versions, and provides tools for working with SBOM in CycloneDX format. Initial version of the extension is distributed as a standalone .vsix file.

#IntelliJ Plugin Changelog

#[2026.8.0] - 2026-02-18

#Added

- Added a panel for viewing policy alerts

#Fixed

- Fixed errors in automatic fixes
- Fixed compatibility issues with IntelliJ versions 2024 and earlier

#[2025.33.0] - 2025-08-14

- The plugin can scan all CodeScoring-supported manifests, display vulnerability information, offer automatic updates to fixed versions, and provides tools for working with SBOM in CycloneDX format. Initial version of the plugin is distributed as a standalone .zip file

#OSA Proxy Changelog

#[2026.22.0] - 2026-05-27

Note

The new features listed below are available only in the Go implementation of OSA Proxy.

#Added

- Added a new Go implementation of OSA Proxy. See documentation
- Added support for Composer/Packagist repositories: metadata checks, dist archive checks, and dist URL rewriting through OSA Proxy Go. See documentation
- Added support for RubyGems repositories: RubyGems metadata checks and downloaded `.gem` package checks. See documentation
- Added host/subdomain-based routing for multiple Docker Registry API v2 repositories in the Go implementation: the subdomain name matches `repository[*].name`, while Docker client requests keep the standard `/v2/...` and `/token` paths. See documentation
- Added repository-level file type filtering through `repository[*].file-type-filter` to limit which artifacts are sent to package scanning.

#Changed

- In the Go implementation, Prometheus metrics are exposed directly at `/metrics`; Spring Boot Actuator endpoints `/actuator/metrics` and `/actuator/prometheus` are not used.

#[2026.13.2] - 2026-04-30

#Added

- Added support for multiple package registries in PyPI via the `additional-packages-registries` parameter: allows configuring additional package sources with automatic host-based routing. Relevant for repositories that serve packages from multiple hosts (e.g., PyTorch). See documentation

#Fixed

- Fixed handling of PyPI WHL index pages without package versions (e.g., subdirectories like `/whl/cu124/`): such pages are no longer forwarded to CodeScoring for scanning
- Fixed an issue where the proxy did not follow HTTP redirects when downloading packages outside of debug mode

#[2026.13.1] - 2026-04-16

#Added

- Added configurable blocked version handling in npm, PyPI, and NuGet manifests via the `codescoring.remove-blocked-versions` parameter (default: `true`): when `true`, blocked versions are fully removed from the manifest; when `false`, they are marked as deprecated with the name of the triggered policy
- Added Micrometer Tracing support — trace IDs are now included in log output
- Attestation manifests in Docker are now skipped during scanning: manifests with type `attestation-manifest` are no longer sent to CodeScoring for analysis

#Fixed

- Fixed a `Registry path cannot be blank` error when proxying Go packages with an empty registry path

#[2026.13.0] - 2026-04-02

#Fixed

- Fixed installation of packages from PyTorch PyPI subdirectories (`/whl/cu124/`, `/whl/cu130/`, etc.)

- Caching headers (`Last-Modified`, etc.) are now stripped from HEAD request responses. This fixes an issue where JFrog Artifactory considered the manifest up-to-date and did not re-fetch it — updated manifests with applied policies are now retrieved promptly

#Added

- Added support for `sum.golang.org` for Go module proxying. See documentation

#Changed

- Instead of removing vulnerable versions from npm, PyPI, and NuGet manifests, they are now marked as deprecated. Installation behavior for packages with unbound versions remains unchanged, while blocked versions gain added clarity — the triggered policy is displayed directly in the manifest

#[2026.5.3] - 2026-03-02

#Fixed

- Fixed an issue where package downloads were still blocked even when the "do not block downloads on errors" flag was enabled; the incorrect behavior occurred for `blocked_scan_failed` responses
- Fixed an error when accessing `jitpack.io`

#Changed

- Unified logging for blocked versions in manifests and blocked package downloads: both are now logged through a single entry point, `ru.codescoring.proxy.logging.PolicyLogger`

#[2026.5.2] - 2026-02-20

#Added

- Added support for the `https://download.pytorch.org` repository

#[2026.5.1] - 2026-02-11

#Added

- Implemented `docker.io` proxying via the `docker pull` command without the `library` prefix

#Changed

- Replaced the KEYS command with SCAN for searching keys in Redis, allowing Redis to run in restricted privilege mode (`+@all -@dangerous`)

- Expanded manifest parsing logging: blocked package versions and applied blocking policies are now logged

#Fixed

- Disabled forwarding of caching headers for HEAD requests, thus speeding up manifest refreshes in JFrog Artifactory and policy application

#[2026.5.0] - 2026-01-30

#Added

- Added metrics for CodeScoring requests
- Added the ability to replace the URL in the blocking reason link with the one specified in the `OSA-Proxy` config `codescoring.host`. See configuration
- Added support for APK and RPM repositories
- Added support for configuring `JAVA_OPTS` in the Helm chart. See installation
- Added support for Docker repositories

Pulling official images

Pulling official images without the `library` prefix (e.g., `docker pull <osa-proxy-host>/postgres`) will be available in the next patch release. Currently, the full path must be used: `docker pull <osa-proxy-host>/library/postgres`

#Changed

- Changed `gateway_route_<package_type>_requests_seconds` metrics from Histogram to SLO (10ms, 25ms, 50ms, 100ms, 250ms, 500ms, 1s, 2s, 5s)

#Fixed

- Fixed operation with remote repositories behind **Cloudflare**, which resulted in error code 1101 from Cloudflare
- Fixed an error with adding the repository to the package name in PURL for Go

#Removed

- Removed duplicate `gateway_route_` metric `http_server_requests`

#[2025.52.2] - 2026-01-19

#Fixed

- Fixed the display of empty paths in the dependency tree that caused incorrect behavior in NuGet-cli. If component versions are blocked, their corresponding parent nodes are now also removed

#[2025.52.1] - 2026-01-15

#Fixed

- Fixed an issue with Redis health check when cache is disabled

#[2025.52.0] - 2025-12-23

#Added

- Added caching of policy checking results in Redis with TTL and background refresh support. See documentation
- Added a proactive cache refresh mechanism based on a timer
- Added Swagger UI for API documentation (`/api/swagger`)
- Added REST API for cache management (`/api/cache`)
- Added header updates (`Last-Modified`, ETag) for manifests. This ensures manifests stay up to date with the latest security policies

#Changed

- Increased default value of `max-in-memory-size` config variable from 50MB to 150MB (for processing large manifests)
- Optimized NPM manifest processing using a stream processor to increase processing speed for large files and reduce garbage collector load

#[2025.48.3] - 2025-12-08

#Fixed

- Fixed the resource leak, which manifested immediately before the Garbage Collection (GC) cycle
- Fixed a behavior that caused the error `Connection prematurely closed BEFORE response` in cases where the connection was closed on the server side or by a reverse proxy

#[2025.48.2] - 2025-12-02

#Fixed

- Fixed an issue where packages with a release date were blocked when the `Dependency release date is empty` policy was active

#[2025.48.0] - 2025-11-28

#Fixed

- Fixed how security policies were applied to package versions whose data was missing from the CodeScoring Index at the time of scanning

#[2025.47.0] - 2025-11-19

#Added

- Implemented support for Go and Debian packages
- Introduced handling for DELAYED status
- Extended proxy server functionality to pass context via URL, enabling repository-bound policies to be applied in the `jfrog/nexus -> OSA proxy -> internet` configuration. For more detailed information, please refer to documentation
- When requesting a manifest, if all package versions are blocked, a list of corresponding blocking policies is now displayed in the response

#Fixed

- Resolved an issue that prevented policies from being applied to repositories in the `OSA proxy -> jfrog/nexus -> internet` configuration

#[2025.39.2] - 2025-10-22

#Added

- Implemented metrics for each package manager type, available at `gateway.route.[maven|npm|nuget|pipy].requests`
- When all package versions in a manifest are blocked by the policies, the service now returns a correct error status and the reason for the block instead of an empty manifest
- Implemented warning (`Warn`) logging in case of incorrect configuration if the manifest does not contain links to packages using URLs from `repository.registry`. In rare cases, false positives may appear in the logs, which will be addressed if detected

#Fixed

- Fixed an error that occurred during JFrog Artifactory requests to the actual `pypi` package index via the `/simple/` route
- In `warmup` & `spectator` modes, version blocking in manifests is no longer performed

#[2025.39.1] - 2025-09-23

#Package scanning

Two levels of scanning:

- **Manifest scanning** — analyzes and excludes versions blocked by security policies from the manifest
- **Package scanning** — analyzes downloaded package

#Blocking Unsafe Components

- Policy-blocked versions are excluded from the manifest.
- Downloading of policy-blocked archives is prevented.
- A configurable status code with a blocking message in the status-line is returned.

#CodeScoring.Save Changelog

#[2026.21.0] - 2026-05-26

#Added

- Initial CodeScoring.Save release — artifact repository manager with proxy and hosted modes, web UI, RBAC, cleanup policies, audit log, metrics, and OSA.Proxy integration
- Support for Maven, npm, NuGet, PyPI, Go, Docker / OCI, and raw formats
- Deployment with OSA.Proxy through a Helm chart

Learn more in Functional characteristics, Architecture, and Repository management.