

# **Repository Management with Nexus**

Ed. 4.0

---

# Contents

<b>1</b>	<b>Introducing Sonatype Nexus</b>	<b>1</b>
<b>2</b>	<b>Component Lifecycle Management and Repository Management</b>	<b>7</b>
<b>3</b>	<b>Installing and Running Nexus</b>	<b>26</b>
<b>4</b>	<b>Configuring Maven and Other Build Tools</b>	<b>57</b>
<b>5</b>	<b>Using the Nexus User Interface</b>	<b>70</b>
<b>6</b>	<b>Configuring Nexus</b>	<b>106</b>
<b>7</b>	<b>Nexus Smart Proxy</b>	<b>168</b>
<b>8</b>	<b>Nexus LDAP Integration</b>	<b>178</b>
<b>9</b>	<b>Atlassian Crowd Support</b>	<b>202</b>

<b>10 Nexus Procurement Suite</b>	<b>217</b>
<b>11 Improved Releases with Nexus Staging</b>	<b>235</b>
<b>12 Repository Health Check</b>	<b>288</b>
<b>13 Managing Maven Settings</b>	<b>296</b>
<b>14 OSGi Bundle Repositories</b>	<b>305</b>
<b>15 P2 Repositories</b>	<b>311</b>
<b>16 .NET Package Repositories with NuGet</b>	<b>315</b>
<b>17 Node Packaged Modules and npm Registries</b>	<b>326</b>
<b>18 Ruby, RubyGems and Gem Repositories</b>	<b>335</b>
<b>19 RPM Packages and YUM Repositories</b>	<b>344</b>
<b>20 Deploying Sites to Nexus</b>	<b>353</b>
<b>21 Nexus Best Practises</b>	<b>362</b>
<b>22 Nexus Plugins</b>	<b>365</b>
<b>23 Migrating to Nexus</b>	<b>370</b>

<b>24 Configuring Nexus for SSL</b>	<b>383</b>
<b>25 Evaluating Nexus Step by Step</b>	<b>394</b>
<b>26 Nexus Community</b>	<b>425</b>
<b>A Contributing to the Nexus Book</b>	<b>430</b>
<b>B Copyright</b>	<b>431</b>
<b>C Creative Commons License</b>	<b>433</b>

# Preface

This book covers the concepts of component lifecycle and repository management in general and specifically the usage of Sonatype Nexus Open Source and Sonatype Nexus Professional. It details all aspects of set-up and running Nexus with the features of the latest release version 2.11.2.

This book was last updated and published on 2015-03-20.

# Chapter 1

## Introducing Sonatype Nexus

### 1.1 Introduction

Nexus manages software components required for development, deployment, and provisioning. If you develop software, Nexus can help you share those components with other developers and end users. Nexus greatly simplifies the maintenance of your own internal repositories and access to external repositories. With Nexus you can completely control access to, and deployment of, every component in your organization from a single location.

Nexus is available in three editions:

- Nexus Open Source - Nexus OSS
- Nexus Professional - Nexus PRO
- Nexus Professional CLM Edition - Nexus PRO CLM

The basis of all Nexus versions is formed by Nexus Open Source. It is licensed under the Eclipse Public License version 1.0 and can be used to get started with component and repository management. It provides a plugin infrastructure for all its features and supports numerous repository formats out of the box.

Nexus Professional builds on top of the numerous features of Nexus Open Source and adds component

---

information integration, improvements for the release process, improved LDAP integration and other features typically required by enterprises and advanced users.

Nexus Professional CLM Edition adds integration with Sonatype CLM to Nexus Professional. This makes your component management policies and rules configurable and actionable and provides further automation and integration with numerous tools to advanced users.

Upgrades from Nexus Open Source can be easily performed. This makes Nexus Open Source an easy, yet powerful solution to get started with component and repository management. Nexus Professional adds further features as well as full support by Sonatype.

**TIP**

Check out Chapter 2 for more background on component lifecycle management and repository management.

## 1.2 Nexus Open Source

Nexus Open Source provides you with an essential level of control over the external repositories you use and the internal repositories you create. It provides infrastructure and services for organizations that use repository managers to obtain and deliver software. If you create software libraries or applications for your end users, you can use Nexus Open Source to distribute your software. If your software depends on open source software components, you can cache software components from remote repositories.

### 1.2.1 Nexus Open Source Features

#### Hosting Repositories

When you [host a repository](#) with Nexus Open Source, you can upload artifacts using the Nexus interface, or you can deploy artifacts to hosted repositories using Maven. Nexus will also create the standard Nexus Index for all of your hosted repositories, which will allow tools like m2eclipse to rapidly locate software artifacts for your developers.

#### Proxy Remote Repositories

When you [proxy a remote repository](#) with Nexus Open Source, you can control all aspects of the connection to a remote repository, including security parameters, HTTP proxy settings. You can configure from which mirrors Nexus will download, how long Nexus will store artifacts, and how it will expire artifacts which are no longer referenced by your build.

## Repository Groups

[Grouping repositories](#) allows you to consolidate multiple repositories into a single URL. This makes configuring your development environment very easy. All of your developers can point to a single repository group URL, and if anyone ever needs a custom remote repository added to the group, you can do this in a central location without having to modify every developer's workstation.

## Numerous Repository Formats

The concepts of hosted repositories, proxy repositories and repository groups are supported for a number of repository formats such as [Maven 2](#), [NuGet](#), [NPM](#), [RubyGems](#) or [YUM](#). This allows you to facilitate one Nexus server to bring the same advantages to all developers in a team relying on different technologies and build tools including [Apache Maven](#), Apache Ant with [Apache Ivy](#) or [Eclipse Aether](#), [Gradle](#), [SBT](#), [.Net](#), [Node.js](#), [Ruby](#) and many others.

## Hosting Project Web Sites

Nexus is a publishing destination for [project web sites](#). While you very easily generate a project web site with Maven, without Nexus, you will need to set up a WebDAV server and configure both your web server and build with the appropriate security credentials. With Nexus, you can deploy your project's web site to the same infrastructure that hosts the project's build output. This single destination for binaries and documentation helps to minimize the number of moving parts in your development environment.

## Fine-grained Security Model

Nexus Open Source ships with a very [capable and customizable security framework](#) that can be used to configure user interface as well as component access. Every operation in Nexus is associated with a privilege, and privileges can be combined into standard Nexus roles. Users can then be assigned both individual privileges and roles that can be applied globally or at a fine-grained level. You can create custom administrative roles that limit certain repository actions, such as deployment to specific groups of developers, and you can use these security roles to model the structure of your organization.

## Flexible LDAP Integration

If your organization uses an LDAP server, Nexus [can integrate with an external authentication and access control system](#). Nexus is smart enough to be able to automatically map LDAP groups to the appropriate Nexus roles, and it also provides a very flexible facility for mapping existing users and existing roles to Nexus roles.

## Component Search

Nexus Open Source provides [an intuitive search feature](#) which allows you to search for software components by identifiers, such as groupId, artifactId, version, classifier, and packaging, names of classes contained in Java archives, keywords, and component sha1 checksums. In addition Nexus can automatically download the index from remote repositories. This allows discovery of components available in these remote repositories without prior downloads.

## Scheduled Tasks

Nexus Open Source has the [concept of scheduled tasks](#): periodic jobs which take care of various repository management tasks, such as deleting old snapshots, evicting unused items, and publishing repository indexes.

---

## REST Services

Nexus Open Source is based on [a series of REST](#) services, and when you are using the Nexus web front-end UI, you are really just interacting with a set of REST services. Because of this open architecture, you can leverage the REST service to create custom interactions or to automate repository management with your own scripts.

## Integration with m2eclipse

When you use Nexus as a repository manager it creates indexes that support the [Maven integration for the Eclipse IDE -M2Eclipse](#). They are immediately available to the project creation wizards and are included in search results in the IDE and other operations with dependencies and plugins.

# 1.3 Nexus Professional

Nexus Professional is designed to meet the needs of the enterprise and builds upon solid foundation and features provided by Nexus Open Source. It is a central point of access to external repositories which provides and a central distribution point with the intelligence required to support the decision that go into making quality software.

## 1.3.1 Nexus Professional Features

### Rich Component Information

The Sonatype CLM data services provides up-to-date and accurate information about known component security vulnerabilities as well as license issues found by component source inspection. This information is [available in Nexus Professional](#) and helps your users with their component choice.

### Nexus Staging Suite

When was the last time you did a software release to a production system? Did it involve a QA team that had to sign off on a particular build? What was the process you used to re-deploy a new build if QA found a problem with the system at the last minute? The [Nexus Staging Suite](#) provides workflow support for the release process of binary software components. If you need to create a release component and deploy it to a hosted repository, you can use the Staging Suite to post a collection of related, staged components which can be tested, promoted, or discarded as a unit. Nexus keeps track of the individuals who are involved in a staged, managed release and can be used to support the decisions that go into producing quality software.

### Support for OSGi Repositories

Nexus Professional adds support for [OSGi Bundle repositories](#) and [P2 repositories](#) for those developers who are targeting OSGi or the Eclipse platform. Just like you can proxy, host, and group

---

Maven 2, NuGet or NPM repositories with Nexus Open Source, Nexus Professional allows you to do the same with OSGi repositories.

### Enterprise LDAP Support

Nexus Professional offers [LDAP support](#) features for enterprise LDAP deployments, including detailed configuration of cache parameters, support for multiple LDAP servers and backup mirrors, the ability to test user logins, support for common user/group mapping templates, and the ability to support more than one schema across multiple servers.

### Support for Atlassian Crowd

If your organization uses Atlassian Crowd, Nexus Professional can [delegate authentication and access control to a Crowd server](#) and map Crowd groups to the appropriate Nexus roles.

### Maven Settings Management

Nexus Professional along with the Nexus M2Settings Maven Plugin allows you to [manage Maven settings](#). Once you have developed a Maven Settings template, developers can then connect to Nexus Professional using the Nexus M2Settings Maven plugin which will take responsibility for downloading a Maven settings file from Nexus and replacing the existing Maven settings on a local workstation.

### Custom Repository Metadata

Nexus Professional provides a facility for user-defined [custom metadata](#). If you need to keep track of custom attributes to support approval workflow or to associate custom identifiers with software artifacts, you can use Nexus to define and manipulate custom attributes which can be associated with artifacts in a Nexus repository.

## 1.4 Nexus Professional CLM Edition

The Nexus Professional CLM Edition includes a Sonatype CLM server that can be used to define component usage policies and automate the enforcement during the release process with the Nexus Staging Suite.

### 1.4.1 Nexus Professional CLM Edition Features

#### Component Usage Policies

The Sonatype CLM server allows you to define component usage policies in terms of security vulnerabilities, license issues and many other characteristics of the used components.

#### Release Policy Enforcement

The Nexus Staging Suite can be configured to use [application-specific policies for automated release validation](#).

---

### Application Specific Component Information

The [component information](#) displayed in Nexus can take the application-specific policies of your organization into account and display the specific validation result to the users.

## Chapter 2

# Component Lifecycle Management and Repository Management

### 2.1 Introduction

Component Lifecycle Management (CLM) in general and, specifically, the subset Repository Management are two aspects of current software development best practices that are closely related to Nexus usage. In this chapter you will learn more about CLM and repository management and how you can take advantage of Nexus features to implement these best practices.

### 2.2 Component Lifecycle Management

Component lifecycle management can be defined as the practice of analysis, control, and monitoring of all components used in your software development lifecycle.

It has emerged as a new category of software development products, information services, and practices that help manage agile, collaborative, component-based development efforts. They allow you to ensure the integrity of the modern software supply chain, amplifying the benefits of modern development, while reducing risk.

---

### 2.2.1 Increasing Component Usage and Open Source Components

Modern software development practices have shifted dramatically from large efforts of writing new code to the usage of components to assemble applications. This approach limits the amount of code authorship to the business-specific aspects of your software.

A large number of open source components in the form of libraries, reusable widgets or whole applications, application servers and others are now available featuring very high levels of quality and feature sets that could not be implemented as a side effect of your business application development. For example creating a new web application framework and business workflow system just to create a website with a publishing workflow would be extremely inefficient.

Open source has become an integral part of modern applications in this form of components. A typical enterprise application is comprised of tens, if not hundreds, of components accounting for 80% and more of the application.

### 2.2.2 Security Vulnerability and License Compliance Risks

With the huge benefits derived from using open source as well as commercial components comes the complexity of understanding all the implications to your software delivery. These include security vulnerabilities, license compliance problems as well as quality issues that need to be managed through the whole life cycle starting at the inception of the software all the way through development, quality assurance, production deployments and onwards until the decommissioning of the software.

The number of components, their rapid change rate with new releases, as well as the ease of adding new dependencies, make the management and full understanding of all involved components a task, that cannot be carried out manually and requires the assistance of tools such as Nexus and Sonatype CLM.

### 2.2.3 Nexus and Component Lifecycle Management

Nexus provides a number of tools that can help you in your CLM efforts. Besides focusing on being a component repository manager it includes features, such as the display of security vulnerabilities as well as license analysis results within search results and the Repository Health Check reports for a proxy repository.

Specific examples about using Nexus for CLM related tasks can be found in Chapter [12](#).

---

## 2.3 Repository Management

Repository Management is a critical practice that is part of your Component Lifecycle Management implementation. Without repository management your component usage is effectively out of control and cannot be governed and managed. This makes it impossible to track security, license, and quality issues you are exposed to due to the components you use from your source code through your build environments and releases to production usage.

Repository managers serve two purposes: they act as highly configurable proxies between your organization and the public repositories and they provide an organization with a deployment destination for its own generated artifacts. Just as Source Code Management (SCM) tools are designed to manage source artifacts, repository managers have been designed to manage and track external dependencies and artifacts generated by your build. They are an essential part of any enterprise or open-source software development effort, and they enable greater collaboration between developers and wider distribution of software.

### 2.3.1 Proxying Public Repositories

Proxying and caching a remote public repository can speed up your builds by reducing redundant downloads over the public Internet. If a developer in your organization needs to download version 2.5 of the Spring Framework and you are using Nexus, the dependencies (and the dependency's dependencies) only need to be downloaded from the remote repository once.

With a high-speed connection to the Internet this might seem like a minor concern, but if you are constantly asking your developers to download hundreds of megabytes of third-party dependencies, the real cost savings are going to be the time it takes Maven to check for new versions of dependencies and to download dependencies over the public Internet.

Proxying and serving Maven dependencies from a local repository cache can save you hundreds of HTTP requests over the public Internet, and in very large multi-module projects, this can shave minutes from a build.

### 2.3.2 Managing Releases and Snapshots

If your project is relying on a number of snapshot dependencies, Maven will need to regularly check for updated versions of these snapshots. Depending on the configuration of your remote repositories, Maven will check for snapshot updates periodically, or it might be checking for snapshot updates on every build.

---

When Maven checks for a snapshot update it needs to interrogate the remote repository for the latest version of the snapshot dependency. Depending on your connection to the public Internet and the load on the Maven Central repository, a snapshot update can add seconds to your project's build for each snapshot dependency you rely upon.

When you host a local repository proxy with Nexus, you reduce the amount of time it takes for Maven to check for a newer version as your build interacts with a local repository cache. If you develop software with snapshot dependencies, using a local repository manager will save you a considerable amount of time, as your 5-10 second snapshot update checks against the public Central Repository are going to execute in hundreds of milliseconds (or less) when they are executed against a local resource.

### 2.3.3 Getting Control of Dependencies

In addition to the simple savings in time and bandwidth, a repository manager provides an organization with control over what is downloaded by Maven. You can include or exclude specific artifacts from the public repository, and having this level of control over what is downloaded from the Maven Central repository is a prerequisite for many organizations which have a need for strict standards for the quality and security of the dependencies used in an enterprise system.

If you want to standardize on a specific version of a dependency like Hibernate or Spring, you can enforce this standardization by only providing access to a specific version of an artifact in Nexus. You might be concerned with making sure that every external dependency has a license compatible with your legal standards for adopting and integrating open source libraries. If you are producing an application which is distributed, you might want to make sure that no one inadvertently adds a dependency on a third-party library covered under a copy-left license like the General Public License (GPL). All of this is possible with Nexus.

Repository managers are a central point of access to external binary software artifacts and dependencies upon which your systems rely. Nexus provides a level of control that is essential when you are trying to track and manage the libraries and frameworks your software depends upon.

### 2.3.4 Nexus for Collaboration

Aside from the benefits of mediating access to remote repositories, a repository manager also provides an important platform for collaborative software development. Unless you expect every member of your organization to download and build every single internal project from source, you will want to provide a mechanism for developers and departments to share binary artifacts (both snapshots and releases) for internal software projects. Internal groups often consume the APIs and systems which are generated by

other internal groups. When you adopt Nexus as a deployment platform for internal artifacts, you can easily share components and libraries between groups of developers.

Nexus provides you with a deployment target for your software components. Once you install Nexus, you can start using Maven to deploy snapshots and releases to internal repositories, which can then be combined with other repositories in repository groups. Over time, this central deployment point for internal projects becomes the fabric for collaboration between different development teams and operations. Nexus is the secret ingredient that allows an organization to scale its development effort without sacrificing agility.

## 2.4 What is a Repository?

Maven developers are familiar with the concept of a repository: a collection of binary software artifacts and metadata stored in a defined directory structure which is used by clients such as Apache Ivy to retrieve binaries during a build process. In the case of the Maven repository, the primary type of binary artifact is a JAR file containing Java bytecode, but there is no limit to what type of artifact can be stored in a Maven repository. For example, one could just as easily deploy documentation archives, source archives, Flash libraries and applications, or Ruby libraries to a Maven repository. A Maven repository provides a platform for the storage, retrieval, and management of binary software artifacts and metadata.

In Maven, every software artifact is described by an XML document called a Project Object Model (POM). This POM contains information that describes a project and lists a project's dependencies—the binary software artifacts which a given component depends upon for successful compilation or execution.

When Maven downloads a dependency from a repository, it also downloads that dependency's POM. Given a dependency's POM, Maven can then download any other libraries which are required by that dependency. The ability to automatically calculate a project's dependencies and transitive dependencies is made possible by the standard and structure set by the Maven repository.

Maven and other tools, such as Ivy which interact with a repository to search for binary software artifacts, model the projects they manage and retrieve software artifacts on-demand from a repository. When you download and install Maven without any customization, Maven will retrieve artifacts from the Central Repository which serves millions of Maven users every single day. While you can configure Maven to retrieve binary software artifacts from a collection of mirrors, the best practice is to install Nexus and use it to proxy and cache the contents of Central on your own network.

In addition to Central, there are a number of major organizations, such as Red Hat, Oracle, and Codehaus which maintain separate repositories.

---

While this might seem like a simple, obvious mechanism for distributing artifacts, the Java platform existed for several years before the Maven project created a formal attempt at the first repository for Java artifacts. Until the advent of the Maven repository in 2002, a project's dependencies were gathered in a manual, ad-hoc process and were often distributed with a project's source code. As applications grew more and more complex, and as software teams developed a need for more complex dependency management capabilities for larger enterprise applications, Maven's ability to automatically retrieve dependencies and model dependencies between components became an essential part of software development.

### 2.4.1 Release and Snapshot Repositories

A repository stores two types of artifacts: releases and snapshots. Release repositories are for stable, static release artifacts. Snapshot repositories are frequently updated repositories that store binary software artifacts from projects under constant development.

While it is possible to create a repository which serves both release and snapshot artifacts, repositories are usually segmented into release or snapshot repositories serving different consumers and maintaining different standards and procedures for deploying artifacts. Much like the difference between a production network and a staging network, a release repository is considered a production network and a snapshot repository is more like a development or a testing network. While there is a higher level of procedure and ceremony associated with deploying to a release repository, snapshot artifacts can be deployed and changed frequently without regard for stability and repeatability concerns.

The two types of artifacts managed by a repository manager are:

#### Release

A release artifact is an artifact which was created by a specific, versioned release. For example, consider the 1.2.0 release of the commons-lang library stored in the Maven Central repository. This release artifact, commons-lang-1.2.0.jar, and the associated POM, commons-lang-1.2.0.pom, are static objects which will never change in the Maven Central repository. Released artifacts are considered to be solid, stable, and perpetual in order to guarantee that builds which depend upon them are repeatable over time. The released JAR artifact is associated with a PGP signature, an MD5 and SHA checksum which can be used to verify both the authenticity and integrity of the binary software artifact.

#### Snapshot

Snapshot artifacts are artifacts generated during the development of a software project. A Snapshot artifact has both a version number such as "1.3.0" or "1.3" and a timestamp in its name. For example, a snapshot artifact for commons-lang 1.3.0 might have the name commons-lang-1.3.0-20090314.182342-1.jar the associated POM, MD5 and SHA hashes would also have a similar name. To facilitate collaboration during the development of software components, Maven and

other clients that know how to consume snapshot artifacts from a repository also know how to interrogate the metadata associated with a Snapshot artifact to retrieve the latest version of a Snapshot dependency from a repository.

A project under active development produces snapshot artifacts that change over time. A release is comprised of artifacts which will remain unchanged over time.

## 2.4.2 Repository Coordinates

Repositories and tools like Maven know about a set of coordinates, including the following components: groupId, artifactId, version, and packaging. This set of coordinates is often referred to as a GAV coordinate, which is short for *Group, Artifact, Version coordinate*. The GAV coordinate standard is the foundation for Maven's ability to manage dependencies. Four elements of this coordinate system are described below:

### groupId

A group identifier groups a set of artifacts into a logical group. Groups are often designed to reflect the organization under which a particular software component is being produced. For example, software components being produced by the Maven project at the Apache Software Foundation are available under the groupId org.apache.maven.

### artifactId

An artifact is an identifier for a software component. An artifact can represent an application or a library; for example, if you were creating a simple web application your project might have the artifactId "simple-webapp", and if you were creating a simple library, your artifact might be "simple-library". The combination of groupId and artifactId must be unique for a project.

### version

The version of a project follows the established convention of Major, Minor, and Point release versions. For example, if your simple-library artifact has a Major release version of 1, a minor release version of 2, and point release version of 3, your version would be 1.2.3. Versions can also have alphanumeric qualifiers which are often used to denote release status. An example of such a qualifier would be a version like "1.2.3-BETA" where BETA signals a stage of testing meaningful to consumers of a software component.

### packaging

Maven was initially created to handle JAR files, but a Maven repository is completely agnostic about the type of artifact it is managing. Packaging can be anything that describes any binary software format including ZIP, SWC, SWF, NAR, WAR, EAR, SAR.

---

### 2.4.3 Addressing Resources in a Repository

Tools designed to interact with Maven repositories translate artifact coordinates into a URL which corresponds to a location in a Maven repository. If a tool such as Maven is looking for version 1.2.0 of the commons-lang JAR in the group org.apache.commons, this request is translated into:

```
<repoURL>/org/apache/commons/commons-lang/1.2.0/commons-lang-1.2.0.jar
```

Maven would also download the corresponding POM for commons-lang 1.2.0 from:

```
<repoURL>/org/apache/commons/commons-lang/1.2.0/commons-lang-1.2.0.pom
```

This POM may contain references to other dependencies which would then be retrieved from the same repository using the same URL patterns.

### 2.4.4 The Central Repository

The most useful Maven repository is the Central Repository. The Central Repository is the largest repository for Java-based components and the default repository built into Apache Maven. Statistics about the size of the Central Repository are available at <http://search.maven.org/#stats>. You can look at the Central Repository as an example of how Maven repositories operate and how they are assembled. Here are some of the properties of release repositories such as the Central Repository:

#### Artifact Metadata

All software artifacts added to the Central Repository require proper metadata, including a Project Object Model (POM) for each artifact which describes the artifact itself and any dependencies that software artifact might have.

#### Release Stability

Once published to the Central Repository, an artifact and the metadata describing that artifact never change. This property of release repositories guarantees that projects which depend on releases will be repeatable and stable over time. While new software artifacts are being published every day, once an artifact is assigned a release number on the Central Repository, there is a strict policy against modifying the contents of a software artifact after a release.

#### Repository Mirrors

The Central Repository is a public resource, and it is currently used by the millions of developers who have adopted Maven and other build tools that understand how to interact with the Maven repository structure. There are a series of mirrors for the Central Repository which are constantly

synchronized. Users are encouraged to query for project metadata and cryptographic hashes and they are encouraged to retrieve the actual software artifacts from one of Central's many mirrors. Tools like Nexus are designed to retrieve metadata from the Central Repository and artifact binaries from mirrors.

### Artifact Security

The Central Repository contains cryptographic hashes and PGP signatures, which can be used to verify the authenticity and integrity of software artifacts served from Central or one of the many mirrors of Central and supports connection to Central in a secure manner via HTTP.

## 2.5 What is a Repository Manager

If you use Maven, you are using a repository to retrieve artifacts and Maven plugins. In fact, Maven used a Maven repository to retrieve core plugins that implement the bulk of the features used in your builds. Once you start to rely on repositories, you realize how easy it is to add a dependency on an open source software library available in the Maven Central repository, and you might start to wonder how you can provide a similar level of convenience for your own developers. When you install a repository manager, you are bringing the power of a repository like Central into your organization, you can use it to proxy Central, and host your own repositories for internal and external use. In this section, we discuss the core functionality that defines what a repository manager does.

Put simply, a repository manager provides two core features:

- The ability to proxy a remote repository and cache artifacts saving both bandwidth and time required to retrieve a software artifact from a remote repository, and
- The ability to host a repository providing an organization with a deployment target for software artifacts.

In addition to these two core features, a repository manager also allows you to manage binary software artifacts through the software development lifecycle, search and catalogue software artifacts, audit development and release transactions, and integrate with external security systems, such as LDAP. The following sections define the feature sets of Nexus Open Source and Nexus Professional.

## 2.5.1 Core Capabilities of a Repository Manager

The base-line features of a repository manager are a description of the core capabilities of Nexus Open Source. Nexus Open Source provides for the:

### Management of Software Artifacts

A repository manager is able to manage packaged binary software artifacts. In Java development, this would include JARs containing bytecode, source, or javadoc. In other environments, such as Flex, this would include any SWCs or SWFs generated by a Flex build.

### Management of Software Metadata

A repository manager should have some knowledge of the metadata that describes artifacts. In a Maven repository this would include project coordinates (groupId, artifactId, version, classifier) and information about a given artifact's releases.

### Proxying of External Repositories

Proxying an external repository yields more stable builds as the artifacts used in a build can be served to clients from the repository manager's cache even if the external repository becomes unavailable. Proxying also saves bandwidth and time as checking for the presence of an artifact on a local network is often orders of magnitude faster than querying a heavily loaded public repository.

### Deployment to Hosted Repositories

Organizations that deploy internal snapshots and releases to hosted repositories have an easier time distributing software artifacts across different teams and departments. When a department or development group deploys artifacts to a hosted repository, other departments and development groups can develop systems in parallel, relying upon dependencies served from both release and snapshot repositories.

### Searching an Index of Artifacts

When you collect software artifacts and metadata in a repository manager, you gain the ability to create indexes and allow users and systems to search for artifacts. With the Nexus index, an IDE such as Eclipse has almost instantaneous access to the contents of all proxy repositories (including the Central repository) as well as access to your own internal and third-party artifacts. While the Central repository transformed the way that software is distributed, the Nexus index format brings the power of search to massive libraries of software artifacts.

### Infrastructure for Artifact Management

A repository manager should also provide the appropriate infrastructure for managing software artifacts and a solid API for extension. In Nexus, Sonatype has provided a plugin API, which allows developers to customize both the behavior, appearance, and functionality of the tool.

## 2.5.2 Additional Features of a Repository Manager

Once you adopt the core features of a repository manager, you start to view a repository manager as a tool that enables more efficient collaboration between development groups. Nexus Professional builds upon the foundations of a repository manager and adds capabilities such as Procurement and Staging.

### Managing Project Dependencies

Many organizations require some level of oversight over the open source libraries and external artifacts that are integrated into an organization's development cycle. An organization could have specific legal or regulatory constraints that require every dependency to be subjected to a rigorous legal or security audit before it is integrated into a development environment. Another organization might have an architecture group which needs to make sure that a large set of developers only has access to a well-defined list of dependencies or specific versions of dependencies. Using the Procurement features of Nexus Professional, managers and architecture groups have the ability to allow and deny specific artifacts from external repositories.

### Managing a Software Release

Nexus Professional adds some essential workflow to the process of staging software to a release repository. Using Nexus Professional, developers can deploy to a staging directory that can trigger a message to a Release Manager or to someone responsible for QA. Quality assurance (or a development manager) can then test and certify a release, having the option to promote a release to the release repository or to discard a release if it didn't meet release standards. Nexus Professional's staging features allow managers to specify which personnel are allowed to certify that a release can be promoted to a release repository giving an organization more control over what software artifacts are released and who can release them.

### Integration with LDAP

Nexus integrates with an LDAP directory, allowing an organization to connect Nexus to an existing directory of users and groups. Nexus authenticates users against an LDAP server and provides several mechanisms for mapping existing LDAP groups to Nexus roles.

### Advanced Security

Using Nexus Professional provides the User Token feature set. It removes the need for storing username and password credentials in the Maven settings file, replacing it with Nexus-managed tokens that can automatically be updated to the user's specific settings file with the Maven settings integration. The tokens do not allow any reverse engineering of the user name and password and, therefore, do not expose these on the file system in the settings file in any form.

### Settings Templates

Nexus Professional allows you to define Maven settings templates for developers. Developers can then automatically receive updates to Maven settings (`~/.m2/settings.xml`) using the Maven Nexus plugin. The ability to define Maven settings templates and to distribute customized Maven settings files to developers makes it easy for an organization to change global profiles or repository configuration without relying on developers to manually install a new settings file in a development environment.

---

## Support for Multiple Repository Formats

Nexus Professional supports the P2 and the OSGi Bundle repository format used by the new Eclipse provisioning platform and OSGi developers. You can use the P2 plugin to consolidate, provision, and control the plugins that are being used in an Eclipse IDE. Using Nexus procurement, repository groups, and proxy repositories to consolidate multiple plugin repositories, an organization can use Nexus Professional to standardize the configuration of Eclipse IDE development environments.

## Archive Browsing

Nexus Professional allows users to browse the contents of archives directly in the user interface as described in Section [5.6](#).

## 2.6 Reasons to Use a Repository Manager

Here are a few reasons why using a repository manager is imperative. While most people wouldn't even think of developing software without the use of a source code control system like Subversion or Perforce, the concept of using a repository manager is still something that needs development. There are many who have used Maven for years without realizing the benefits of using a repository manager. This section was written as an attempt to capture some of the benefits of using a repository manager.

### 2.6.1 Speed Up Your Builds

When you run your multimodule project in Maven, how do you think Maven knows if it needs to update plugins or snapshot dependencies? It has to make a request for each artifact it needs to test. Even if nothing has changed, if your project depends on a few snapshot or if you don't specify plugin version, Maven might have to make tens to hundreds of requests to a remote repository. All of these requests over the public internet add up to real, wasted time. We have found complex builds to cut build time by up to 75 percent after installing a local instance of Nexus. You are wasting time better spent coding waiting for your build to needlessly interrogate a remote Maven repository.

### 2.6.2 Save Bandwidth

The larger the organization, the more critical bandwidth savings can be. If you have thousands of developers regularly wasting good bandwidth to download the same files over and over again, using a repository manager to keep a local cache is going to save you a good deal of bandwidth. Even for smaller organizations with limited budgets for connectivity and IT operations, having to deal with a set of developers

---

maxing out your connection to the Internet to download the same things over and over again seems backwards.

### 2.6.3 Ease the Burden on Central

Running the Maven Central repository is no short order. It ain't cheap to serve the millions of requests and Terabytes of data required to satisfy the global demand for software artifacts from the Maven Central repository. Something as simple as installing a repository manager at every organization that uses Maven would likely cut the bandwidth requirements for Central by at least half. If you have more than a couple developers using Maven, install a repository manager for the sake of keeping Central available and in business.

### 2.6.4 Gain Predictability and Scalability

How often in the past few years has your business come to a crashing halt because of an outage? Depending on Central for your day-to-day operations also means that you depend on having Internet connectivity (and on the fact the Central will remain available 24/7). While Sonatype is confident in its ability to keep Central running 24/7, you should take some steps of your own to make sure that your development team isn't going to be surprised by some network outage on either end. If you have a local repository manager, like Nexus, you can be sure that your builds will continue to work, even if you lose connectivity.

### 2.6.5 Control and Audit Dependencies and Releases

So, you've moved over to Maven (or maybe Ivy that reads the same repository), and you now have a whole room full of developers who feel empowered to add or remove dependencies and experiment with new frameworks. We've all seen this. We've all worked in places with a developer who might be more interested in experimenting than in working. It is unfortunate to say so, but there are often times when an architect or an architecture group needs to establish some baseline standards that are going to be used in an organization. Nexus provides this level of control. If you need more oversight over the artifacts that are making it into your organization, take a look at Nexus. Without a repository manager, you are going to have little control over what dependencies are going to be used by your development team.

---

## 2.6.6 Deploy Third-Party Artifacts

How do you deal with that one-off JAR from a vendor that is not open source, and not available on the Maven Central repository? You need to deploy these artifacts to a repository and configure your Maven instance to read from that repository. Instead of handcrafting some POMs, download Nexus and take the two or three minutes it is going to take to get your hands on a tool that can create such a repository from third-party artifacts. Nexus provides an intuitive upload form that you can use to upload any random free-floating JAR that finds its way into your project's dependencies.

## 2.6.7 Collaborate with Internal Repositories

Many organizations require every developer to check out and build the entire system from source, simply because they have no good way of sharing internal JARs from a build. You can solve a problem like this by dividing projects and using Nexus as an internal repository to host internal dependencies.

For example, consider a company that has 30 developers split into three groups of 10 with each group focused on a different part of the system. Without an easy way to share internal dependencies, a group like this is forced either to create an ad-hoc filesystem-based repository or to build the system in its entirety so that dependencies are installed in every developer's local repository.

The alternative is to separate the projects into different modules that all have dependencies on artifacts hosted by an internal Nexus repository. Once you've done this, groups can collaborate by exchanging compiled snapshot and release artifacts via Nexus. In other words, you don't need to ask every developer to check out a massive multimodule project that includes the entire organization's code. Each group within the organization can deploy snapshots and artifacts to a local Nexus instance, and each group can maintain a project structure, which includes only the projects it is responsible for.

## 2.6.8 Distribute with Public Repositories

If you are an open source project, or if you release software to the public, Nexus can be the tool you use to serve artifacts to external users. Think about it this way... When was the last time you cut a release for your software project? Assuming it wasn't deployed to a Maven repository, you likely had to write some scripts to package the contents of the release, maybe someone special had to sign the release with a super-secret cryptographic key. Then, you had to upload it to some web server, and then make sure that the pages that describe the upload were themselves updated. Lots of needless complexity...

If you were using something like Nexus, which can be configured to expose a hosted repository to the

---

outside world, you could use the packaging and assembly capabilities of Maven and the structure of the Maven repository to make a release that is more easily consumed. This isn't just for JAR files and Java web applications. Maven repositories can host any kind of artifact. Nexus, and Maven repositories in general, define a known structure for releases. If you are writing some Java library, publishing it to your own Nexus instance serving a public repository will make it easier for people to start using your code right away.

## 2.7 Adopting a Repository Manager

Adopting a repository manager is not an all or nothing proposition, and there are various levels (or stages) of adoption that can be distinguished when approaching repository management. On one end of the adoption spectrum is the organization that installs a repository manager just to control and consolidate access to a set of remote repositories. On the other end of the spectrum is the organization that has integrated the repository manager into an efficient software development lifecycle, using it to facilitate decision points in the lifecycle, encouraging more efficient collaboration throughout the enterprise, and keeping detailed records to increase visibility into the software development process.

### 2.7.1 Stage Zero: Before Using a Repository Manager

While this isn't a stage of adoption, Stage Zero is a description of the way software builds work in the absence of a repository manager. When a developer decides that he needs a particular open source software component, he will download it from the component's web site, read the documentation, and find the additional software that his components rely on (referred to as "dependencies"). Once he has manually assembled a collection of dependencies from various open source project web sites and proprietary vendors, he will place all these components somewhere on the network so that he, his team members, the build script, the QA team, and the production support team can find it. At any time, other developers may bring in other components, sometimes with overlapping dependencies, placing them in different network locations. The instructions to bring all of these ad-hoc, developer-managed components libraries together in a software build process can become very complicated and hard to maintain.

Maven was introduced to improve this build process by introducing the concept of structured repositories from which the build scripts can retrieve the software components. In Maven language, these software components or dependencies are referred to as artifacts, a term which can refer to any generic software artifact including components, libraries, frameworks, containers, etc. Maven can identify artifacts in repositories, understand their dependencies, retrieve all that are needed for a successful build, and deploy its output back to repositories when completed.

---

Developers using Maven without a repository manager find most of their software artifacts and dependencies in Maven Central. If they happen to use another remote repository or if they need to add a custom artifact, the solution in Stage Zero is to manually manipulate the files in a local repository and share this local repository with multiple developers. While this approach may yield a working build for a small team, managing a shared local repository doesn't allow an organization to scale a development effort. There is no inherent control over who can set up a local repository, who can add to them or change or delete from them nor are there tools to protect the integrity of these repositories.

That is, until Repository Managers were introduced.

### **2.7.2 Stage One: Proxying Remote Repositories**

This is the easiest stage to understand both in terms of benefits to an organization and action required to complete this stage. All you need to do to start proxying a remote repository is to deploy Nexus and start the server with the default configuration. Configure your Maven clients to read from the Nexus public repository group, and Nexus will automatically retrieve artifacts from remote repositories, such as Maven Central, caching them locally.

Without a repository manager, your organization might have hundreds of developers independently downloading the same artifacts from public, remote repositories. With a repository manager, these artifacts can be downloaded once and stored locally. After Stage One, your builds run considerably faster than they did when you relied upon the Maven Central repository.

Once you've installed Nexus and you've configured all of your organization's clients to use it as a single point of access to remote repositories, you begin to realize that it now provides you with a central configuration point for the artifacts used throughout your organization. Once you've started to proxy, you can start to think about using Nexus as a tool to control policy and what dependencies are allowed to be used in your organization. Nexus Professional provides a procurement plugin which allows for fine-grained control over which artifacts can be accessed from a remote repository. This procurement feature is described in more detail in the section which deals with lifecycle integration.

### **2.7.3 Stage Two: Hosting a Repository Manager**

Once you have started to proxy remote repositories and you are using Nexus as a single, consolidated access point for remote repositories, you can start to deploy your own artifacts to Nexus hosted repositories. Most people approach repository management to find a solution for proxying remote repositories, and while proxying is the most obvious and immediate benefit of installing a repository manager, hosting internally generated artifacts tends to be the stage that has the most impact on collaboration within an

---

organization.

To understand the benefits of hosting an internal repository, you have to understand the concept of managing binary software artifacts. Software development teams are very familiar with the idea of a source code repository or a source code management tool. Version control systems such as Subversion, Clearcase, Git, and CVS provide solid tools for managing the various source artifacts that comprise a complex enterprise application, and developers are comfortable checking out source from source control to build enterprise applications. However, past a certain point in the software development lifecycle, source artifacts are no longer relevant. A QA department trying to test an application or an Operations team attempting to deploy an application to a production network no longer needs access to the source artifacts. QA and Operations are more interested in the compiled end-product of the software development lifecycle—the binary software artifacts. A repository manager allows you to version, store, search, archive, and release binary software artifacts derived from the source artifacts stored in a source control system. A repository manager allows you to apply the same systematic operations on binary software artifacts that you currently apply to your source code.

When your build system starts to deploy artifacts to an internal repository, it changes the way that developers and development groups can interact with one another in an enterprise. Developers in one development group can code and release a stable version of an internal library, deploy this library to an internal Nexus release repository, and so share this binary artifact with another group or department. Without a repository manager managing internal artifacts, you have ad-hoc solutions and the organizational equivalent of duct tape. How does the infrastructure group send a new library to the applications group without Nexus? Someone copies a file to a shared directory and sends an email to the team lead. Organizations without repository managers are full of these ad-hoc processes that get in the way of efficient development and deployment.

With a repository manager, every developer and every development group within the enterprise understands and interacts with a common collaborative structure—the repository manager. Do you need to interact with the Commerce team’s new API? Just add a dependency to your project and Maven will retrieve the library from Nexus automatically.

One of the other direct benefits of deploying your own artifacts to a repository such as Nexus is the ability to quickly search the metadata and contents of those artifacts both via a web UI and through IDE integration tools such as m2eclipse. When you start to deploy internal artifacts you can synchronize all development groups to a common version and naming standard, and you can use the highly configurable authentication and role-based access controls to control which developers and which development groups can deploy artifacts to specific repositories or paths within a repository.

---

## 2.7.4 Stage Three: Continuous Collaboration

Developing this collaborative model further, if your application is being continuously built and deployed using a tool like Hudson, a developer can check out a specific module from a large multimodule build and not have to constantly deal with the entire source tree at any given time. This allows a software development effort to scale efficiently. If every developer working on a complex enterprise application needs to checkout the entire source tree every time he or she needs to make a simple change to a small component, you are quickly going to find that building the entire application becomes a burdensome bottleneck to progress. The larger your enterprise grows, the more complex your application becomes, the larger the collective burden of wasted time and missed opportunities. A slow enterprise build prevents the quick turnaround or quick feedback loop that helps your developers maintain focus during a development cycle.

Once you are building with Maven, sharing binary artifacts with Nexus, continuously testing and deploying with Hudson, and generating reports and metrics with tools like Sonar, your entire organization gains a collaborative "central nervous system" that enables a more agile approach to software development.

## 2.7.5 Stage Four: Lifecycle Integration

Once you've configured a repository manager to proxy remote repositories and you are using a repository manager as an integration point between developers and departments, you start to think about the various ways your repository manager can be used to support the decisions that go into software development. You can start using the repository manager to stage releases and supporting the workflow associated with a managed release, and you can use the procurement features of a tool like Nexus Professional to give management more visibility into the origins, characteristics, and open source licenses of the artifacts used during the creation of an enterprise application.

Nexus Professional enables organizations to integrate the management of software artifacts tightly with the software development lifecycle: Provisioning, Compliance, Procurement, Enterprise Security, Staging and other capabilities that support the workflow that surrounds a modern software development effort.

Using Nexus Professional's Maven Settings management feature and integrated security features you can configure a developer's Maven settings by running a single, convenient Maven goal and downloading customized settings for a particular developer. When you use Maven and Nexus Professional together, developers can get up and running quickly, collaborating on projects that share common conventions without having to manually install dependencies in local repositories.

### Provisioning

Using Nexus as an integration point between Engineering and Operations means that Engineering

---

can be responsible for delivering solid, tested artifacts to Quality Assurance and Operations via a standard repository format. Often development teams are roped into the production deployment story and become responsible for building entire production environments within a build system. This blends the functions and roles of software engineering with those of systems administration thus blurring the lines between Engineering and Operations. If you use Nexus as an end point for releases from Engineering, Operations can then retrieve, assemble, and configure an application from tested components in the Nexus repository.

## Compliance

Procurement, staging, and audit logs are all features that increase the visibility into who and what is involved with your software development efforts. Using Nexus Professional, Engineering can create the reports and documents that can be used to facilitate discussions about oversight. Organizations subject to various regulations often need to produce a list of components involved in a software release. Legal departments often require a list of open source licenses being used in a particular software component, and managers often lack critical visibility into the software development process.

## Procurement

The ease with which today's developer can add a dependency on a new open source library and download this library from a Central repository has a downside. Organizations large and small are constantly wondering what open source libraries are being used in applications, and whether these libraries have acceptable open source licenses for distribution. The Procurement features of Nexus Professional give architects and management more oversight of the artifacts that are allowed into an organization. Using the Procurement features, a Nexus administrator or Procurement manager can allow or deny specific artifacts by group, version, or path. You can use the procurement manager as a firewall between your own organization's development environment and the 95,000 artifacts available on the Maven Central repository.

## Enterprise Security

Nexus' LDAP integration allows an enterprise to map existing LDAP groups to Nexus roles and provides Nexus administrators with a highly configurable interface to control which individuals or groups have access to a fine-grained set of Nexus permissions.

## Staging

Nexus Professional adds an important step to the software release workflow, adding the concept of a managed (or staged) release to a hosted repository. When a developer needs to perform a production release, Nexus Professional can isolate the artifacts involved in a release in a staged repository that can then be certified and tested. A manager or a quality assurance tester can then promote or discard a release. The staging feature allows you to specify the individuals that are allowed to promote a release and keeps an audit of who was responsible for testing, promoting, or discarding a software release.

---

# Chapter 3

# Installing and Running Nexus

## 3.1 Nexus Prerequisites

Nexus Open Source and Nexus Professional only have one prerequisite, a Java Runtime Environment (JRE) compatible with Java 7 or Java 8. Nexus is most often run with the JRE that is bundled with a Java Development Kit (JDK) installation, and it can be run with Oracle's JDK. To download the Oracle JDK, go to <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. At a minimum Java 7u2 is required, but we recommend to use the latest available version.

## 3.2 Downloading Nexus

There are two distributions of Nexus: [Nexus Open Source](#) and [Nexus Professional](#). Nexus Open Source is a fully-featured repository manager which can be freely used, customized, and distributed under the Eclipse Public License (EPL Version 1). Nexus Professional is a distribution of Nexus with features that are relevant to large enterprises and organizations which require complex procurement and staging workflows in addition to more advanced LDAP integration, Atlassian Crowd support, and other development infrastructure. The differences between Nexus Open Source and Nexus Professional are explored in the previous chapter.

---

### 3.2.1 Downloading Nexus Open Source

To download the latest Nexus Open Source distribution, go to <http://www.sonatype.org/nexus/go> and choose Nexus (TGZ) or Nexus (ZIP) shown in Figure 3.1. This will download a Gzip TAR (TGZ) or a ZIP with identical contents. Your download will be file named `nexus-2.11.2.-06-bundle.zip` or `nexus-2.11.2.-06-bundle.tar.gz`.

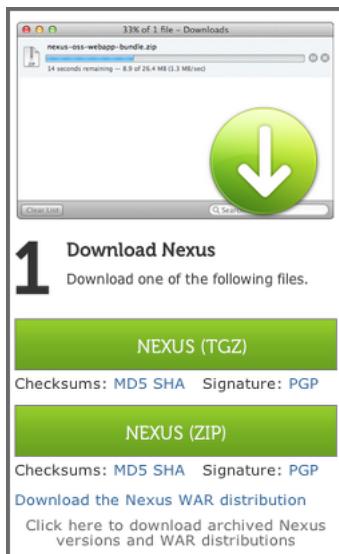


Figure 3.1: Downloading Nexus Open Source

Older versions can be downloaded following the link at the bottom of Figure 3.1 and selecting a version and archive type in the page displayed in Figure 3.2.

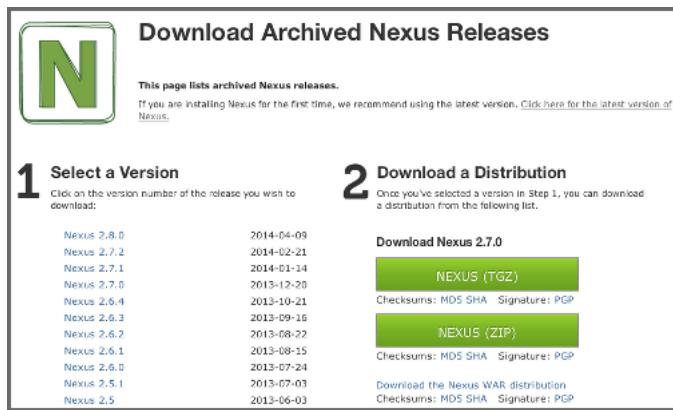


Figure 3.2: Selecting a Specific Version of Nexus Open Source to Download

### 3.2.2 Downloading Nexus Professional

Nexus Professional can be downloaded as `zip` or `tar.gz` archive from [the Nexus Professional support download page](#). Existing customers with access to the support system can also download it directly from the [Nexus Professional Support landing page](#).

---

**Tip**

Use the [Nexus Pro trial version](#) for an evaluation.

---

## 3.3 Installing Nexus

The following instructions are for installing Nexus Open Source or Nexus Professional as a stand-alone server. Nexus comes bundled with a Jetty instance that listens to all configured IP addresses on a host (0.0.0.0) and runs on port 8081 by default.

Installing Nexus is straightforward. Unpack the Nexus web application archive in a directory. If you are installing Nexus on a local workstation to give it a test run, you can install it in your home directory or wherever you like. Nexus doesn't have any hard coded directories. It will run from any directory. If you downloaded the ZIP

```
$ unzip nexus-2.11.2.-06-bundle.zip
```

And, if you download the GZip'd TAR archive, run:

```
$ tar xvzf nexus-2.11.2.-06-bundle.tar.gz
```

For Nexus professional the equivalent commands would be

```
$ unzip nexus-professional-2.11.2.-06-bundle.zip  
$ tar xvzf nexus-professional-2.11.2.-06-bundle.tar.gz
```



### Caution

There are some known incompatibilities with the version of the tar command provided by Solaris and the GZip TAR format. If you are installing Nexus on Solaris, you must use the GNU tar application, or you will end up with corrupted files.

---

### Note

If you are installing Nexus on a server, you might want to use a directory other than your home directory. On a Unix machine, this book assumes that Nexus is installed in /usr/local/nexus-2.11.2.-06 with a symbolic link /usr/local/nexus to the nexus directory. Using a generic symbolic link nexus to a specific version is a common practice which makes it easier to upgrade when a newer version of Nexus is made available.

---

```
$ sudo cp nexus-2.11.2.-06-bundle.tar.gz /usr/local  
$ cd /usr/local  
$ sudo tar xvzf nexus-2.11.2.-06-bundle.tar.gz  
$ sudo ln -s nexus-2.11.2.-06 nexus
```

Although it isn't required for Nexus to run, you may want to set an environment variable NEXUS\_HOME in your environment that points to the installation directory of Nexus. This chapter will refer to this location as \$NEXUS\_HOME.

---

**Note**

On Windows you should install Nexus outside Program Files to avoid problems with Windows file registry virtualization. If you plan to run Nexus as a specific user you could install into the AppData\Local directory of that users home directory. Otherwise simply go with e.g., C:\nexus or something similar.

---

The Nexus installation directory `nexus-2.11.2-06` or `nexus-professional-2.11.2-06` has a sibling directory named `sonatype-work`. This directory contains all of the repository and configuration data for Nexus and is stored outside of the Nexus installation directory to make it easier to upgrade to a newer version of Nexus.

By default, this directory is always a sibling to the Nexus installation directory. If you installed Nexus in the `/usr/local` directory it would also contain a `sonatype-work` subdirectory with a nested `nexus` directory containing all of the content and configuration. The location of the `sonatype-work` directory can be customized by altering the `nexus-work` property in `$NEXUS_HOME/conf/nexus.properties`.

## 3.4 Upgrading Nexus

Since Nexus separates its configuration and data storage from the application, it is easy to upgrade an existing Nexus installation.

To upgrade Nexus, unpack the Nexus archive in the directory that contains the existing Nexus installation. Once the archive is unpacked, the new Nexus application directory should be a sibling to your existing `sonatype-work/` directory.

If you have defined a symbolic link for the version of Nexus to use, stop the server and change that to point at the new Nexus application directory. When you start the new instance of Nexus it will read the existing repository configuration from the `sonatype-work` directory. Depending on the version you upgrade from and to, some maintenance tasks like rebuilding the internal indices can be necessary. Please refer to the [upgrade notes](#) of the new release for more information on this. In addition, a review of the [release notes](#) can be very useful to get a better understanding of potential, additional steps required.

If you are using any additional plugins supplied by Sonatype, the new version of Nexus you downloaded will contain a newer version of the plugin. Be sure to copy the new version from the `optional-plugins` folder to the `plugin-repository` folder, as documented in Section [22.1](#), and restart Nexus.

Externally supplied plugins are updated by simply replacing the folder with the plugin with the new

---

version.

This automatic upgrade of Nexus works for nearly all update ranges. All 2.x versions can directly upgrade to the latest version. All 1.x version can upgrade to 2.7.x maximum. If you need to upgrade from 1.x to a newer version, you need to perform an intermediate upgrade step to a 2.x version.

---

**Note**

The same upgrade process can be used to change from the open source to the professional version of Nexus.

---

## 3.5 Running Nexus

When you start Nexus, you are starting a web server on the default port `0.0.0.0:8081`. Nexus runs within a servlet container called Eclipse Jetty, and it is started with a native service wrapper called the [Tanuki Java Service Wrapper](#). This service wrapper can be configured to run Nexus as a Windows service or a Unix daemon. Nexus ships with generic startup scripts for Unix-like platforms called `nexus` and for Windows platforms called `nexus.bat` in the `$NEXUS_HOME/bin` folder. To start Nexus on a Unix-like platform like Linux, MacOSX or Solaris use

```
cd /usr/local/nexus  
./bin/nexus console
```

Similarly, starting on Windows can be done with the `nexus.bat` file. Starting Nexus with the `console` command will leave Nexus running in the current shell and display the log output.

On Unix systems, you can start Nexus detached from the starting shell with the `start` command even when not yet installed as a service.

```
./bin/nexus start
```

When executed you should see a feedback message and then you can follow the startup process viewing the log file `logs/wrapper.log` changes.

```
Starting Nexus Repository Manager...  
Started Nexus Repository Manager.  
$ tail -f logs/wrapper.log
```

At this point, Nexus will be running and listening on all IP addresses (0.0.0.0) that are configured for the current host on port 8081. To use Nexus, fire up a web browser and type in the URL <http://localhost:8081/-nexus>. You should see the Nexus user interface as displayed in Figure 3.7.

While we use `localhost` throughout this book, you may need to use the IP Loopback Address of `127.0.0.1`, the IP address or the DNS hostname assigned to the machine running Nexus.

When first starting Nexus Professional you are presented with a form that allows you to request a trial activation. This page displayed in Figure 3.3 contains a link to the license activation screen in Figure 3.4.

The screenshot shows a web form titled "Sonatype Nexus Professional". At the top right is a link "Already have a license?". The main section is titled "Trial Activation" and contains the following text: "Just complete this brief form to activate your trial. We'll email your license key and you'll be up and running in minutes." Below this is another section titled "About Your Trial" with the text: "Once you activate your trial, it will be active for 14 days. If you have any questions, or if you need to extend your trial, just email us at [info@sonatype.com](mailto:info@sonatype.com)". The form fields include:

- \* First name: John
- \* Last name: Smith
- \* Email: user@domain.com
- \* Organization: (empty field)
- \* Country: Please select a country (dropdown menu)
- Region: Please select a region (dropdown menu)

A blue "Submit Activation Request" button is located at the bottom right of the form area.

Figure 3.3: Nexus Trial Activation Form

After submitting the form for your trial activation, you will receive a license key via email that you can use in the license activation screen to activate Nexus Professional. If you already have a license key or license file, you can use the same screen to upload the file and register your license.



Figure 3.4: Nexus License Activation

Once you have agreed to the End User License Agreement you will be directed to the Sonatype Nexus Professional Welcome screen displayed in Figure 3.5.

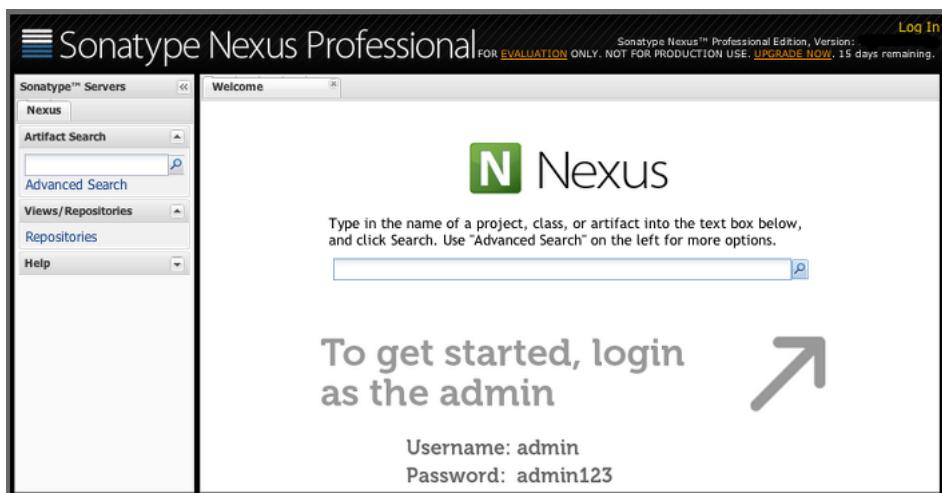


Figure 3.5: Sonatype Nexus Professional Welcome Screen

Click on the *Log In* link in the upper right-hand corner of the web page, and you should see the login dialog displayed in Figure 3.6.

---

**Tip**

The default administrator username and password combination is `admin` and `admin123`.

---

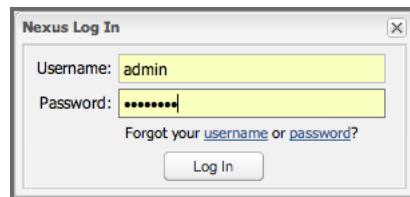


Figure 3.6: Nexus Log In Dialog (default login/password is admin/admin123)

When you are logged into your evaluation version of Nexus Professional, you will see some helpful links to the Nexus Pro Evaluation Guide, Sample Projects and the Knowledgebase below the search input on the Welcome screen.

With a full license for Nexus these links will be removed and you will get the Nexus Application Window displayed in Figure 3.7.

Nexus Open Source will not need to be activated with a license key and will display a number of links to resources and support on the Welcome screen to logged in users.



Figure 3.7: Nexus Application Window

The files from Java Service Wrapper used for the start up process can be found in \$NEXUS\_HOME/bin/jsw and are separated into generic files like the wrapper.conf configuration file in conf and a number of libraries in lib. An optional wrapper.conf include allows you to place further configuration optionally in \$NEXUS\_HOME/conf/wrapper-override.conf.

The platform-specific directories are available for backwards compatibility with older versions only and should not be used. A full list of directories follows:

```
$ cd /usr/local/nexus/bin/jsw
$ ls -1
conf
lib
license
linux-ppc-64
linux-x86-32
linux-x86-64
macosx-universal-32
macosx-universal-64
solaris-sparc-32
solaris-sparc-64
solaris-x86-32
windows-x86-32
windows-x86-64
```

The wrapper.conf file is the central configuration file for the startup of the Jetty servlet container running Nexus on a Java virtual machine and therefore includes configuration for things such as the java command to use, Java memory configuration, logging configuration and other settings documented in the configuration file.

Typical modifications include adapting the maximum memory size to your server hardware and usage requirements e.g. 2000 MB up from the default 768 and other JVM related configurations.

```
wrapper.java.maxmemory=2000
```

Additional configuration in the wrapper.conf file includes activation of further Jetty configuration file for monitoring Nexus via [JMX](#) or using [HTTPS](#).

---

**Tip**

The startup script nexus supports the common service commands start, stop, restart, status, console and dump.

---

## 3.6 Post-Install Checklist

Nexus ships with some default passwords and settings for repository indexing that need to be changed for your installation to be useful (and secure). After installing and running Nexus, you need to make sure that you complete the following tasks:

### 3.6.1 Step 1: Change the Administrative Password and Email Address

The administrative password defaults to *admin123*. The first thing you should do to your new Nexus installation is change this password. To change the administrative password, login as *admin* with the password *admin123*, and click on *Change Password* under the *Security* menu in the left-hand side of the browser window. For more detailed instructions, see Section [5.16](#).

### 3.6.2 Step 2: Configure the SMTP Settings

Nexus can send username and password recovery emails. To enable this feature, you will need to configure Nexus with a SMTP Host and Port as well as any necessary authentication parameters that Nexus needs to connect to the mail server. To configure the SMTP settings, follow the instructions in Section [6.1.1](#).

### 3.6.3 Step 3: Configure Default HTTP and HTTPS Proxy Settings

In many deployments the internet, and therefore any remote repositories that Nexus needs to proxy, can only be reached via a HTTP and HTTPS proxy server internal to the deployment company. In these cases the connection details to that proxy server need to be configured in Nexus, as documented in Section [6.1.5](#) in order for Nexus to be able to proxy remote repositories at all.

### 3.6.4 Step 4: Enable Remote Index Downloads

Nexus ships with three important proxy repositories for the Maven Central repository, Apache Snapshot repository, and the Codehaus Snapshot repository. Each of these repositories contains thousands (or tens of thousands) of artifacts and it would be impractical to download the entire contents of each. To that end, most repositories maintain an index which catalogues the entire contents and provides for fast and

---

efficient searching. Nexus uses these remote indexes to search for artifacts, but we've disabled the index download as a default setting. To download remote indexes:

1. Click on *Repositories* under the *Views/Repositories* menu in the left-hand side of the browser window.
2. Select each of the three proxy repositories and change *Download Remote Indexes* to *true* in the *Configuration* tab. You'll need to load the dialog shown in Figure 6.9 for each of the three repositories.

This will trigger Nexus to re-index these repositories, during which the remote index files will be downloaded. It might take Nexus a few minutes to download the entire index, but once you have it, you'll be able to search the entire contents of the Maven repository.

Once you've enabled remote index downloads, you still will not be able to browse the complete contents of a remote repository. Downloading the remote index allows you to search for artifacts in a repository, but until you download those artifacts from the remote repository they will not show in the repository tree when you are browsing a repository. When browsing a repository, you will only be shown artifacts which have been downloaded from the remote repository.

### **3.6.5 Step 5: Change the Deployment Password**

The deployment user's password defaults to *deployment123*. Change this password to make sure that only authorized developers can deploy artifacts to your Nexus installation. To change the deployment password, log in as an administrator. Click on *Security* to expand the security menu. When the menu appears, click on *Users*. A list of users will appear. At that point, right-click on the user named *Deployment* and select *Set Password*.

### **3.6.6 Step 6: If Necessary, Set the LANG Environment Variable**

If your Nexus instance needs to store configuration and data using an international character set, you should set the `LANG` environment variable. The Java Runtime will adapt to the value of the `LANG` environment variable and ensure that configuration data is saved using the appropriate character type. If you are starting Nexus as a service, place this environment variable in the startup script found in `/etc/init.d/nexus`.

---

### 3.6.7 Step 7: Configure Routes

A route defines patterns used to define and identify the repositories in which the artifacts are searched for. Typically, internal artifacts are not available in the Central Repository or any other external, public repository. A route, as documented in Section 6.4, should be configured so that any requests for internal artifacts do not leak to external repositories.

## 3.7 Configuring Nexus as a Service

When installing Nexus for production usage you should configure Nexus as a service, so it starts back up after server reboots. It is good practice to run that service or daemon as a specific user that has only the required access rights. The following sections provide instructions for configuring Nexus as a service or daemon on various operating systems.

### 3.7.1 Running as a Service on Linux

You can configure Nexus to start automatically by copying the `nexus` script to the `/etc/init.d` directory. On a Linux system perform the following operations as the root user:

1. Create a `nexus` user with sufficient access rights to run the service
2. Copy `$NEXUS_HOME/bin/nexus` to `/etc/init.d/nexus`
3. Make the `/etc/init.d/nexus` script executable and owned by the root user -

```
chmod 755 /etc/init.d/nexus
chown root /etc/init.d/nexus
```
4. Edit this script changing the following variables:
  - a. Change `NEXUS_HOME` to the absolute folder location (e.g., `NEXUS_HOME="/usr/local/nexus"`)
  - b. Set the `RUN_AS_USER` to `nexus` or any other user with restricted rights that you want to use to run the service. You should not be running Nexus as root.
  - c. Change `PIDDIR` to a directory where this user has read/write permissions. In most Linux distributions, `/var/run` is only writable by root. The property you need to add to customize the PID file location is `wrapper.pidfile`. For more information about this property and how it would be configured in `wrapper.conf`, see: <http://wrapper.tanukisoftware.com/doc/english/properties.html>.

5. Change the owner and group of your Nexus-related directories, including `nexus-work` configured in `nexus.properties` defaulting to `sonatype-work/nexus`, to the `nexus` user that will run the application.
6. If Java is not on the default path for the user running Nexus, add a `JAVA_HOME` variable which points to your local Java installation and add a `$JAVA_HOME/bin` to the `PATH`.

**Warning**

We recommend to avoid running Nexus as the root user or a similar privileged user, as this practice poses serious security risks to the host operating system unnecessarily. Instead we suggest to follow system administration best practice and use a service specific user with the minimum required access rights only.

---

### 3.7.1.1 Add Nexus as a Service on Red Hat, Fedora, and CentOS

This script has the appropriate `chkconfig` directives, so all you need to do to add Nexus as a service is run the following commands:

```
$ cd /etc/init.d  
$ chkconfig --add nexus  
$ chkconfig --levels 345 nexus on  
$ service nexus start  
Starting Sonatype Nexus...  
$ tail -f /usr/local/nexus/logs/wrapper.log
```

The second command adds `nexus` as a service to be started and stopped with the `service` command. `chkconfig` manages the symbolic links in `/etc/rc[0-6].d` which control the services to be started and stopped when the operating system restarts or transitions between run-levels. The third command adds `nexus` to run-levels 3, 4, and 5. The `service` command starts Nexus, and the last command tails the `wrapper.log` to verify that Nexus has been started successfully. If Nexus has started successfully, you should see a message notifying you that Nexus is listening for HTTP.

### 3.7.1.2 Add Nexus as a Service on Ubuntu and Debian

The process for setting up Nexus as a service on Ubuntu differs slightly from the process used on a Red Hat variant. Instead of running `chkconfig`, you should run the following sequence of commands once you've configured the startup script in `/etc/init.d`.

---

```
$ cd /etc/init.d  
$ update-rc.d nexus defaults  
$ service nexus start  
Starting Sonatype Nexus...  
$ tail -f /usr/local/nexus/logs/wrapper.log
```

### 3.7.2 Running as a Service on Mac OS X

The standard way to run a service on Mac OS X is by using launchd, which uses plist files for configuration. An example plist file for Nexus installed in /opt is shown [A sample com.sonatype.nexus.plist file](#).

#### A sample com.sonatype.nexus.plist file

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"  
      "http://www.apple.com/DTDs/PropertyList-1.0.dtd">  
<plist version="1.0">  
<dict>  
    <key>Label</key>  
    <string>com.sonatype.nexus</string>  
    <key>ProgramArguments</key>  
    <array>  
        <string>/opt/nexus/bin/nexus</string>  
        <string>start</string>  
    </array>  
    <key>RunAtLoad</key>  
    <true/>  
</dict>  
</plist>
```

After saving the file as com.sonatype.nexus.plist in /Library/LaunchDaemons/ you have to change the ownership and access rights.

```
sudo chown root:wheel /Library/LaunchDaemons/com.sonatype.nexus.plist  
sudo chmod 644 /Library/LaunchDaemons/com.sonatype.nexus.plist
```

---

#### Tip

Consider setting up a different user to run Nexus and adapt permissions and the RUN\_AS\_USER setting in the nexus startup script.

---

With this setup Nexus will start as a service at boot time. To manually start it after the configuration you can use

```
sudo launchctl load /Library/LaunchDaemons/com.sonatype.nexus.plist
```

### 3.7.3 Running as a Service on Windows

The startup script for Nexus on Windows platforms is `bin/nexus.bat`. Besides the standard commands for starting and stopping the service, it has the additional commands `install` and `uninstall`. Running these commands with elevated privileges will set up the service for you or remove it as desired. Once installed as a service with the `install` command, the batch file can be used to start and stop the service. In addition, the service will be available in the usual Windows service management console as a service named *nexus*.

## 3.8 Running Nexus Behind a Reverse Proxy

The Nexus installation bundle is based on the high-performance servlet container Eclipse Jetty running the Nexus web application. This achieves a very high performance of Nexus and make installation of a separate proxy for performance improvements unnecessary.

However, in many cases organizations run applications behind a proxy for security concerns, familiarity with securing a particular proxy server or to consolidate multiple disparate applications using tools like `mod_rewrite`.

Some brief instructions for establishing such a setup with Apache httpd follow as an example. We assume that you've already installed Apache 2, and that you are using a virtual host for `repo.example.com`.

Let's assume that you wanted to host Nexus behind Apache httpd at the URL `http://repo.example.com`. To do this, you'll need to change the context path that Nexus is served from.

1. Edit `nexus.properties` in `$NEXUS_HOME/conf`. You'll see an element named `nexus-webapp-context-path`. Change this value from `/nexus` to `/`
-

2. Restart Nexus and Verify that it is available on `http://localhost:8081/`
3. Set the Base URL in Nexus as shown in Figure 6.4 under *Application Server Settings* to the URL that will be the externally available URL of Nexus e.g. `http://repo.example.com`

At this point, edit the httpd configuration file for the `repo.example.com` virtual host. Include the following to expose Nexus via mod\_proxy at `http://repo.example.com/`.

```
ProxyRequests Off
ProxyPreserveHost On

<VirtualHost *:80>
    ServerName repo.example.com
    ServerAdmin admin@example.com
    ProxyPass / http://localhost:8081/
    ProxyPassReverse / http://localhost:8081/
    ErrorLog logs/repo.example.com/nexus/error.log
    CustomLog logs/repo.example.com/nexus/access.log common
</VirtualHost>
```

If you just wanted to continue to serve Nexus at the `/nexus` context path, you would not change the `nexus-webapp-context-path` and you would include the context path in your `ProxyPass` and `ProxyPassReverse`

```
ProxyPass /nexus/ http://localhost:8081/nexus/
ProxyPassReverse /nexus/ http://localhost:8081/nexus/
```

For the user interface to work via the proxy reliably you also need to configure a `ProxyPassReverseCookiePath`.

```
ProxyPass /nexus http://localhost:8081/
ProxyPassReverse /nexus http://localhost:8081/
ProxyPassReverseCookiePath / /nexus
```

When your reverse proxy is configured to serve https, but it proxies with plain http to your Nexus instance, an additional header is required. This will ensure Nexus renders absolute URLs using the correct protocol. When setting this header, make sure that in Figure 6.4 *Force Base URL* is not checked.

```
RequestHeader set X-Forwarded-Proto "https"
```

Apache configuration is going to vary, based on your own application's requirements and the way you intend to expose Nexus to the outside world. If you need more details about Apache httpd and mod\_proxy, please see the documentation at <http://httpd.apache.org> and specifically [http://httpd.apache.org/docs/current/mod/mod\\_proxy.html](http://httpd.apache.org/docs/current/mod/mod_proxy.html).

---

A similar setup can be configured with [nginx](#). The following configuration is a simplified example for an nginx server running port 80. This server proxies Nexus running on the same server (127.0.0.1 = localhost) at the default port 8081 on the default context /nexus:

```
http {  
    ...  
    proxy_send_timeout 120;  
    proxy_read_timeout 300;  
    proxy_buffering off;  
    keepalive_timeout 5;  
    tcp_nodelay on;  
    ...  
    server {  
        listen 80;  
        server_name localhost;  
  
        location /nexus {  
            proxy_pass http://127.0.0.1:8081;  
            proxy_set_header Host $host;  
            proxy_set_header X-Real-IP $remote_addr;  
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
        }  
    }  
}
```

The [nginx documentation](#) contains further details for setting this up for HTTP as well as for HTTPS, if desired.

If the components proxied include larger files be sure to set `client_max_body_size` to an appropriate value to ensure uploads and downloads can succeed through nginx.

## 3.9 Installing a Nexus Professional License

When starting a Nexus Professional trial installation you can upload your license file as described in Section [3.5](#) on the license screen visible in Figure [3.4](#).

If you are currently using an evaluation license or need to replace your current license with a new one, click on Licensing in the Administration menu. This will bring up the panel shown in Figure [3.8](#). To upload your Nexus Professional license, click on Browse..., select the file, and click on Upload.

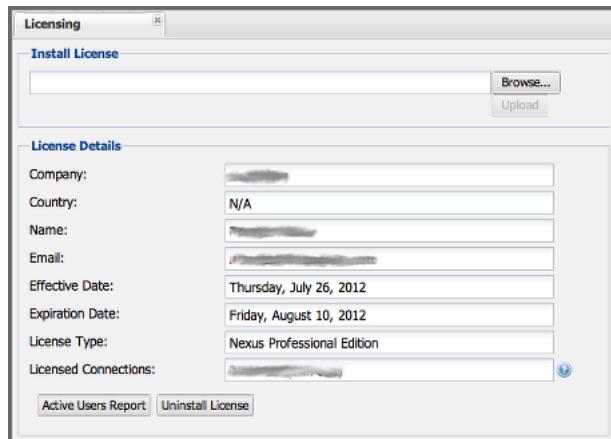


Figure 3.8: Nexus Professional Licensing Panel

Once you have selected a license and uploaded it to Nexus, Nexus Professional will display a dialog box with the Nexus Professional End User License Agreement as shown in Figure 3.9. If you agree with the terms and conditions, click on "I Agree".

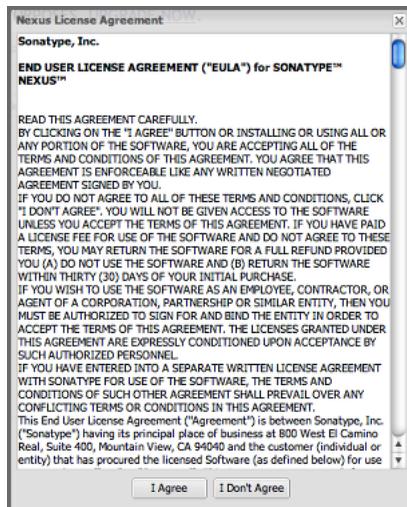


Figure 3.9: Nexus Professional End User License Agreement

Once you have agreed to the terms and conditions contained in the End User License Agreement, Nexus

Professional will then display a dialog box confirming the installation of a Nexus Professional license, as shown in Figure 3.10.

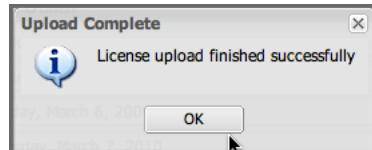


Figure 3.10: License Upload Finished Dialog

If you need to remove your Nexus Professional license, you can click on the "Uninstall License" button at the bottom of the Licensing Panel. Clicking on this button will show the dialog in Figure 3.11, confirming that you want to uninstall a license.

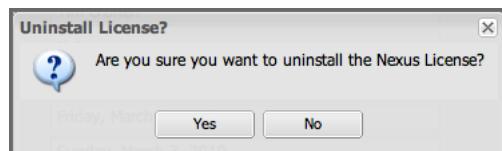


Figure 3.11: Uninstall License Confirmation Dialog

Clicking Yes in this dialog box will uninstall the license from Nexus Professional and display another dialog which confirms that the license has been successfully uninstalled.

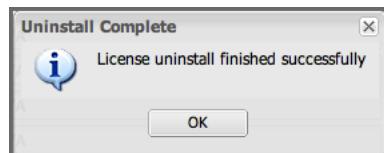


Figure 3.12: License Uninstall Completed Dialog

### 3.9.1 License Expiration

When a Nexus Professional license expires, the Nexus user interface will have all functionality disabled except for the ability to install a new license file.

## 3.10 Nexus Directories

The following sections describe the various directories that are a part of any Nexus installation. When you install Nexus Open Source or Nexus Professional, you are creating two directories: a directory containing the Nexus runtime and application often symlinked as `nexus` and a directory containing your own configuration and data - `sonatype-work/nexus`. When you upgrade to a newer version of Nexus, you replace the Nexus application directory and retain all of your own custom configuration and repository data in `sonatype-work/`.

### 3.10.1 Sonatype Work Directory

The Sonatype Work directory `sonatype-work` is created as a sibling to the `nexus` application directory, and the location of this directory can be configured via the `nexus.properties` file which is described in Section [3.10.2](#).

The Sonatype Work Nexus directory `sonatype-work/nexus/` contains a number of subdirectories. Depending on the plugins installed and used, some directories may or may be not present in your installation:

#### **access/**

This directory contains a log of all IP addresses accessing Nexus. The data can be viewed by clicking on Active Users Report in the Administration - Licensing tab in the Nexus user interface.

#### **aether-local-repository/ or maven2-local-repository**

This holds temporary files created when running Maven dependency queries in the user interface.

#### **backup/**

If you have configured a scheduled job to back up Nexus configuration, this directory is going to contain a number of ZIP archives that contain snapshots of Nexus configuration. Each ZIP file contains the contents of the `conf/` directory. (Automated backups are a feature of Nexus Professional.)

---

**broker/**

The broker directory and its subdirectories contains the storage backend for the Smart Proxy messaging component.

**conf/**

This directory contains the Nexus configuration. Settings that define the list of Nexus repositories, the logging configuration, the staging and procurement configuration, and the security settings are all captured in this directory.

**conf/keystore/**

Contains the automatically generated key used to identify this Nexus instance for Smart Proxy usage

**db/**

Contains the database storing the User Token information, if that feature is enabled.

**error-report-bundles/**

Used to contain the bundled archives of data assembled for problem reporting. Since this feature has been removed this folder can be safely deleted.

**felix-cache/**

This directory holds the cache for the OSGi framework Apache Felix, which is used for the Nexus plugin architecture.

**health-check/**

Holds cached reports from the Repository Health Check plugin.

**indexer/ and indexer-pro/**

Contains a Nexus index for all repositories and repository groups managed by Nexus. A Nexus index is a Lucene index which is the standard for indexing and searching a Maven repository. Nexus maintains a local index for all repositories, and can also download a Nexus index from remote repositories.

**logs/**

The nexus.log file that contains information about a running instance of Nexus. This directory also contains archived copies of Nexus log files. Nexus log files are rotated every day. To reclaim disk space, you can delete old log files from the logs directory.

**nuget/**

Contains the database supporting queries against NuGet repositories used for .NET package support in Nexus.

**p2/**

If you are using the P2 repository management features of Nexus Professional, this directory contains a local cache of P2 repository artifacts.

**plugin-repository/**

This directory contains any additionally installed plugins from third parties as documented in Section [22.1](#).

---

**proxy/**

Stores data about the files contained in a remote repository. Each proxy repository has a subdirectory in the `proxy/attributes/` directory and every file that Nexus has interacted with in the remote repository has an XML file that captures the last requested time stamp, the remote URL for a particular file, the length of the file, the digests for a particular file, and others. If you need to backup the local cached contents of a proxy repository, you should also back up the contents of the proxy repository's directory under `proxy/attributes/`

**storage/**

Stores artifacts and metadata for Nexus repositories. Each repository is a subdirectory that contains the artifacts in a repository. If the repository is a proxy repository, the storage directory will contain locally cached artifacts from the remote repository. If the repository is a hosted repository, the storage directory will contain all artifacts in the repository. If you need to back-up the contents of a repository, you should back up the contents of the storage directory.

**support/**

The support zip archive documented in Section [5.15](#) is created and stored in this folder.

**template-store/**

Contains the Maven settings template files documented in detail in Chapter [13](#).

**timeline/**

Contains an index which Nexus uses to store events and other information to support internal operations. Nexus uses this index to store feeds and history.

**tmp/**

Folder used for temporary storage.

**trash/**

If you have configured scheduled jobs to remove snapshot artifacts or to delete other information from repositories, the deleted data will be stored in this directory. To empty this trash folder, view a list of Nexus repositories, and then click on the Trash icon in the Nexus user interface.

The `conf/` directory contains a number of files which allow for configuration and customization of Nexus. All of the files contained in this directory are altered by the Nexus administrative user interface. While you can change the configuration settings contained in these files with a text editor, Sonatype recommends that you modify the contents of these files using the Nexus administrative user interface. Depending on your Nexus version and the installed plugins, the complete list of files may differ slightly.

**broker.groovy**

A groovy script for configuring low-level properties for Smart Proxy.

**capabilities.xml**

Further Smart Proxy backend configuration.

---

**healthcheck.properties**

Configuration for the Repository Health Check.

**logback.properties, logback.xml and logback-\* .xml**

Contains logging configuration. If you need to customize the detail of log messages, the frequency of log file rotation, or if you want to connect your own custom logging appenders, you should edit the logback-nexus.xml configuration file as desired. If you find log4j.properties files as well, you can safely remove them since they are remnants from an old version and are not used anymore.

**lvo-plugin.xml**

Contains configuration for the latest version plugin. This XML file contains the location of the properties file that Nexus queries to check for a newer version of Nexus.

**nexus.xml**

The bulk of the configuration of Nexus is contained in this file. This file maintains a list of repositories and all server-wide configuration like the SMTP settings, security realms, repository groups, targets, path mappings and others.

**pgp.xml**

Contains PGP key server configuration.

**nexus-obr-plugin.properties**

Contains configuration for the Nexus OSGi Bundle repository plugin in Nexus Professional.

**procurement.xml**

Contains configuration for the Nexus Procurement plugin in Nexus Professional.

**security-configuration.xml**

Contains global security configuration.

**security.xml**

Contains security configuration about users and roles.

**staging.xml**

Contains configuration for the Nexus Staging Plugin in Nexus Professional.

### 3.10.2 Nexus Configuration Directory

After installing Nexus and creating the `nexus` symlink as described earlier, your `fnexus` folder contains another `conf` directory. This directory contains configuration for the Jetty servlet container. You will only need to modify the files in this directory if you are customizing the configuration of Jetty servlet container or the behavior of the scripts that start Nexus.

The files and folders contained in this directory are:

---

+ nexus.properties:: This file contains configuration variables which control the behavior of Nexus and the Jetty servlet container. If you are customizing the port and host that Nexus will listen to, you would change the application-port and application-host properties defined in this file. If you wanted to customize the location of the sonatype-work directory, you would modify the value of the nexus-work property in this configuration file. Changing nexus-webapp-context-path allows you to configure the server context path Nexus will be available at.

### jetty.xml and jetty-\*.xml

Configuration files for the Eclipse Jetty servlet container running Nexus. Jetty users are used to providing a list of jetty XML config files which are merged to form the final configuration. As an advanced configuration option, Nexus supports this merging concept in its launcher code as of Nexus 2.8.

You can specify additional jetty XML configuration files to load to form the final configuration. For the standard distribution bundle, these files can be specified using special properties located in NEXUS\_HOME/bin/jsw/conf/wrapper.conf.

```
wrapper.app.parameter.1=./conf/jetty.xml  
wrapper.app.parameter.2=./conf/jetty-requestlog.xml  
# add more indexed app parameters...
```

Any of the files located at NEXUS\_HOME/conf/jetty-\*.xml can be specified as part of the wrapper.app.parameter.n property, where n is the next highest number not already used. The [Java Service Wrapper](#) documentation contains more information about this property. This setup allows for a simple method to add configuration for https, JMX and others by adjusting a few properties.

---

#### Warning

 Nexus version prior to 2.8 loaded all of the Jetty configuration from one jetty.xml file, typically found at NEXUS\_HOME/conf/jetty.xml and required modifications to this file for configuration changes. Examples were available in NEXUS\_HOME/conf/examples. These files cannot be used in Nexus 2.8 or higher, as they were intended to be standalone files that could not be merged into other files.

---

## 3.11 Monitoring Nexus

Now that your Nexus instance is up and running, you need to ensure that it stays that way. Typically this is done on a number of levels and each organization and system administration team has its own preferences and tools.

---

In general you can monitor:

+ \* hardware values like CPU, memory or diskspace utilization and many more \* operating system level values like processes running \* Java Virtual Machine specific values \* application specific value

For the hardware and operating system values, a large number of dedicated tools exist. Many of these tools can be configured to work with application-specific logs and other events. The following section discusses some of the available information in Nexus. It can potentially be integrated into the usage of the more generic tools for monitoring, log capturing and analysis.

A host of information from the operating system, the Java Virtual Machine and Nexus itself is available via the [Support Tools](#), which allow you to inspect the value directly in the Nexus user interface.

### 3.11.1 General Logging

Nexus logs events in the `sonatype-work/nexus/logs/nexus.log` file. In addition a dedicated user interface to configure and inspect the log is available. Further information about this interface can be found in Section [6.10](#).

### 3.11.2 Request Access Logging

Logging all access requests to Nexus allows you to gain a good understanding of the Nexus usage in your organization and the sources of these requests.

For example, you will be able to tell if the main load is due to a CI server cluster or from your developers, based on the IP numbers of the requests. You can also see the spread of requests and load across different time zones. Also available for review are the URLs, API calls, and features that are used in Nexus.

Requests access logging is enabled by default in Nexus 2.8 or higher and uses a performant and flexible LogBack implementation with built-in log rotation already configured for 90 days of log file retention. The log is written to the file `sonatype-work/nexus/logs/request.log` and contains all requests and the username for authenticated requests.

The configuration is located in `NEXUS_HOME/conf/logback-access.xml` and can be changed to suit your requirements. If you change the file, a restart of Nexus is required for these changes to take effect.

---

If you do not want to run access logging, you can disable it by commenting out the line

```
wrapper.app.parameter.2=conf/jetty-requestlog.xml
```

in bin/jsw/conf/wrapper.conf.

**Warning**

Older versions of Nexus require different customization of the Jetty configuration files. Instructions for these customizations can be found on the [support site](#).

---

### 3.11.3 Using Java Management Extension JMX

JMX is a common tool for managing and monitoring Java applications with client software like the free [VisualVM](#) and many others available. It can be performed locally on the server as well as remotely.

Nexus can be configured to support JMX by adding

```
wrapper.app.parameter.3=./conf/jetty-jmx.xml
```

to the list of `wrapper.app` parameters in `NEXUS_HOME/bin/jsw/conf/wrapper.conf` and set the parameters `jmx-host` and `jmx-port` in `NEXUS_HOME/conf/nexus.properties`.

```
jmx-host=192.168.10.12  
jmx-port=1099
```

`jmx-host` is the host name, or commonly the IP address, to remotely access Nexus using JMX from another host and `jmx-port` is the network port used for the connection. It is important to ensure that the port is not blocked by any network setup, when connecting remotely. The value of 1099 is the default port used for JMX, but any other available port can be used as well.

---

**Warning**

Nexus versions older than 2.8 require different procedures, depending on the specific version.

---

Once Nexus is restarted with JMX enabled you can inspect the running JVM in detail. Figure 3.13 and Figure 3.14 show some example screenshots of VisualVM connected to a Nexus instance running on localhost.

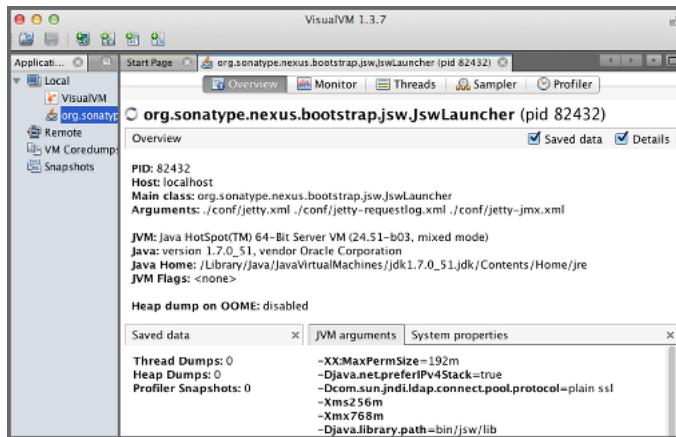


Figure 3.13: Overview of Nexus Monitored via JMX in VisualVM

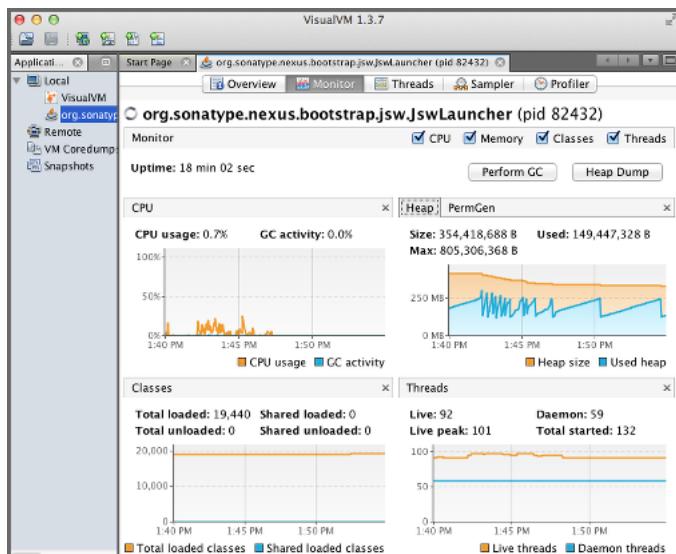


Figure 3.14: CPU, Memory and Other Visualizations of Nexus Monitored via JMX in VisualVM

Depending on the tool used to connect, a number of monitoring, analysis and troubleshooting actions can be performed. Please refer to the documentation about your specific tool for more information.

### 3.11.4 Analytics

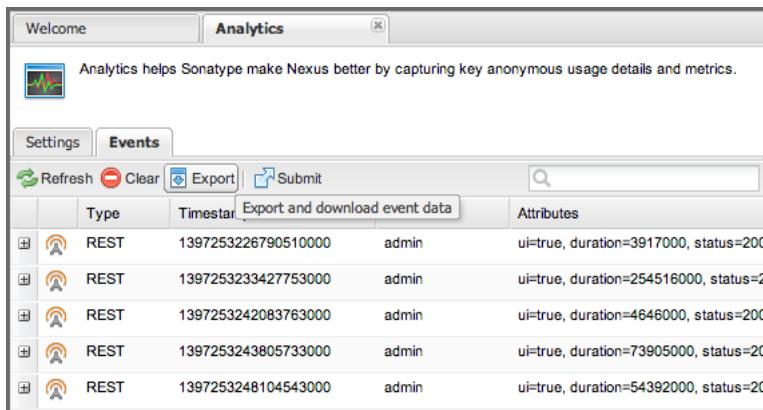
The analytics integration of Nexus allows you to gather a good understanding of your Nexus usage, since it enables the collection of event data in Nexus. It collects non-sensitive information about how you are using Nexus. It is useful to you from a compatibility perspective, since it gathers answers to questions such as what features are most important, where are users having difficulties, and what integrations/APIs are actively in use.

The collected information is limited to the use of the Nexus user interface and the Nexus REST API, the primary interaction points between your environment and Nexus. Only the user interface navigation flows and REST endpoints being called are recorded. None of the request specific data (e.g., credentials or otherwise sensitive information) is ever captured.

You can enable the event logging in the *Settings* section of the *Analytics* tab available via *Analytics* menu item in the *Administration* menu in the left side *Nexus* navigation. Select the checkbox beside *Enable analytics event collection* and press the *Save* button.

You can choose to provide this data automatically to Sonatype by selecting the checkbox beside *Enable automatic analytics event submission*. It enables Sonatype to tailor the ongoing development of the product. Alternatively, you can submit the data manually or just use the gathered data for your own analysis only.

Once enabled all events logged can be inspected in the *Events* tab in the *Analytics* section displayed in Figure 3.15.



The screenshot shows the 'Analytics' tab in the Nexus interface. At the top, there's a message: 'Analytics helps Sonatype make Nexus better by capturing key anonymous usage details and metrics.' Below this is a toolbar with 'Settings' and 'Events' tabs, and buttons for 'Refresh', 'Clear', 'Export', and 'Submit'. A search bar is also present. The main area is a table with columns: Type, Timestamp, User, and Attributes. There are five rows, each representing a REST event. Each row has a '+' icon in the first column.

	Type	Timestamp	Export and download event data	Attributes
+	REST	1397253226790510000	admin	ui=true, duration=3917000, status=200
+	REST	1397253233427753000	admin	ui=true, duration=254516000, status=200
+	REST	1397253242083763000	admin	ui=true, duration=4646000, status=200
+	REST	1397253243805733000	admin	ui=true, duration=73905000, status=200
+	REST	1397253248104543000	admin	ui=true, duration=54392000, status=200

Figure 3.15: List of Events in the Analytics Tab

The list of events shows the *Type* and the *Timestamp* of the event as well as the *User* that triggered it and any *Attributes*. Each row has a + symbol in the first column that allows you to expand the row vertically. Each attribute will be expanded into a separate line allowing you to inspect all the information that is potentially submitted to Sonatype. The *User* value is replaced by a salted hash so that no username information is transmitted. The *Anonymization Salt* is automatically randomly generated by Nexus and can optionally be configured in the *Analytics: Collection* capability manually. This administration area can additionally be used to change the random identifier for the Nexus instance.

---

**Tip**

More information about capabilities can be found in Section [6.6](#).

---

If you desire to further inspect the data that is potentially submitted, you can select to download the file containing the JSON files in a zip archive by clicking the *Export* button above the events list and downloading the file. The *Submit* button can be used to manually submit the events to Sonatype.

When you select to automatically submit the analytics data, a scheduled task, named *Automatically submit analytics events*, is automatically created. This task is preconfigured to run at 1:00 AM every day. If desired the recurrence can be changed in the scheduled tasks administration area documented in Section [6.5](#).

**Important**

 Sonatype values your input greatly and hopes you will activate the analytics feature and the automatic submission to allow us to ensure ongoing development is well aligned with your needs. In addition, we appreciate any further direct contact and feedback in person and look forward to hearing from you.

---

## Chapter 4

# Configuring Maven and Other Build Tools

### 4.1 Introduction

Historically Nexus started as a repository manager supporting the Maven repository format. While it supports many other repository formats now, the Maven repository format is still the most common and well supported format for build and provisioning tools running on the JVM and beyond.

This chapter shows example configurations for using Nexus with a Maven and number of other tools. The setups take advantage of Nexus merging many repositories and exposing them via a repository group. Setting this up is documented in the chapter in addition to the configuration used by specific tools.

### 4.2 Apache Maven

To use Nexus with [Apache Maven](#), we configure Maven to check Nexus instead of the default, built-in connection to the Central Repository.

To do this, you add a `mirror` configuration and override the default configuration for the `central` repository in your `~/.m2/settings.xml` as shown in [Configuring Maven to Use a Single Nexus Group](#).

---

## Configuring Maven to Use a Single Nexus Group

```
<settings>
  <mirrors>
    <mirror>
      <!--This sends everything else to /public -->
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://localhost:8081/nexus/content/groups/public</url>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>nexus</id>
      <!--Enable snapshots for the built in central repo to direct -->
      <!--all requests to nexus via the mirror -->
      <repositories>
        <repository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <!--make the profile active all the time -->
    <activeProfile>nexus</activeProfile>
  </activeProfiles>
</settings>
```

In [Configuring Maven to Use a Single Nexus Group](#), we have defined a single profile called `nexus`. It configures a `repository` and a `pluginRepository` with the id `central` that overrides the same repositories in the super pom. The super pom is internal to every Apache Maven install and establishes default values. These overrides are important since they change the repositories by enabling snapshots and replacing the URL with a bogus URL. This URL is overridden by the `mirror` setting in the same `settings.xml` file to point to the URL of your single Nexus group. This Nexus group can, therefore, contain `release` as well as `snapshot` artifacts and Maven will pick them up.

---

The `mirrorOf` pattern of `*` causes any repository request to be redirected to this mirror and to your single repository group, which in the example is the `public` group.

It is possible to use other patterns in the `mirrorOf` field. A possible valuable setting is to use `external : *`. This matches all repositories except those using `localhost` or file based repositories. This is used in conjunction with a repository manager when you want to exclude redirecting repositories that are defined for integration testing. The integration test runs for Apache Maven itself require this setting.

More documentation about mirror settings can be found in the [mini guide on the Maven web site](#).

As a last configuration the `nexus` profile is listed as an active profile in the `activeProfiles` element.

## 4.3 Adding Repositories for Missing Dependencies

If you've configured your Maven `settings.xml` or other build tool configuration to use the Nexus `public` repository group as a mirror for all repositories, you might encounter projects that are unable to retrieve artifacts from your local Nexus installation.

This usually happens because you are trying to build a project that has defined a custom set of repositories and snapshot repositories or relies on the content of other publically available repositories in its configuration. When you encounter such a project all you have to do is

- add this repository to Nexus as a new proxy repository
- and then add the new proxy repository to the public group.

The advantage of this approach is that no configuration change on the build tool side is necessary at all.

## 4.4 Adding a New Repository

To add a repository, log into Nexus as an administrator, and click on the *Repositories* link in the left-hand navigation menu in the *Views/Repositories* section as displayed in Figure 4.1.

Clicking on this link should bring up a window that lists all of the repositories that Nexus knows about.

---

You'll then want to create a new proxy repository. To do this, click on the *Add* link that is directly above the list of repositories. When you click the *Add* button, click the down arrow directly to the right of the word *Add*, this will show a drop-down which has the options: *Hosted Repository*, *Proxy Repository*, *Virtual Repository*, and *Repository Group*. Since you are creating a proxy repository, click on *Proxy Repository*.

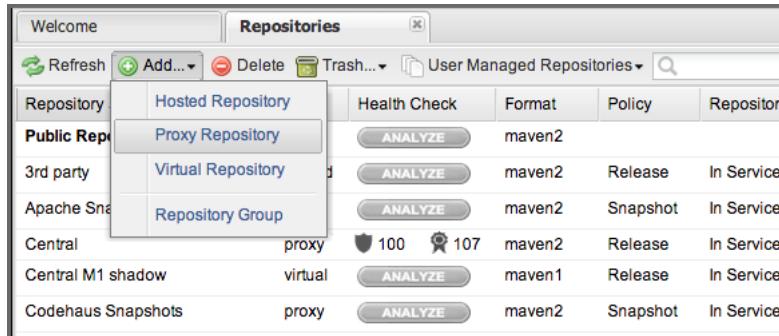


Figure 4.1: Creating a New Proxy Repository

Once you do this, you will see a screen resembling Figure 4.2. Populate the required fields *Repository ID* and the *Repository Name*. The *Repository ID* will be part of the URL used to access the repository, so it is recommended to avoid characters that could cause problems there or on the filesystem storage. It is best to stick with lowercase alphanumerics. Set the *Repository Policy* to *Release*, and the *Remote Storage Location* to the public URL of the repository you want to proxy.

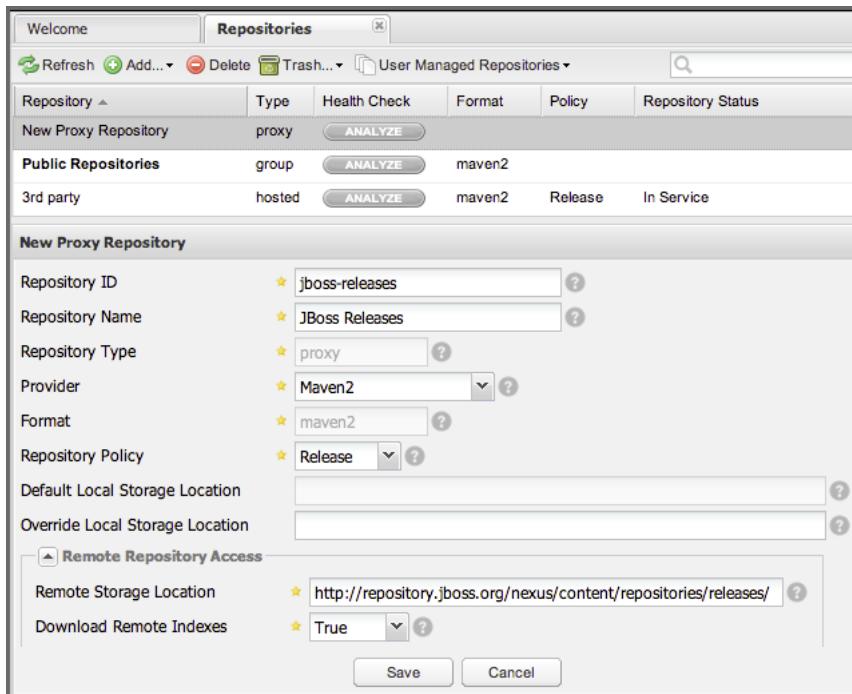


Figure 4.2: Configuring a Proxy Repository

Once you've filled out this screen, click on the *Save* button. Nexus will then be configured to proxy the repository. If the remote repository contains snapshots as well as release components, you will need to repeat the process creating a second proxy repository and setting the policy to *Snapshots*.

## 4.5 Adding a Repository to a Group

Next you will need to add the new repositories to the *Public Repositories* Nexus repository group. To do this, click on the *Repositories* link in the left-hand Nexus menu in the *Views/Repositories* section. Nexus lists Groups and Repositories in the same list so click on the public group. After clicking on the *Public Repositories* group, you should see the *Browse* and *Configuration* tabs in the lower half of the Nexus window.

**Note**

If you click on a repository or a group in the *Repositories* list and you do not see the *Configuration* tab, this is because your Nexus user does not have administrative privileges. To perform the configuration tasks outlined in this chapter, you will need to be logged in as a user with administrative privileges.

Clicking on the *Configuration* tab will bring up a screen which looks like Figure 4.3.

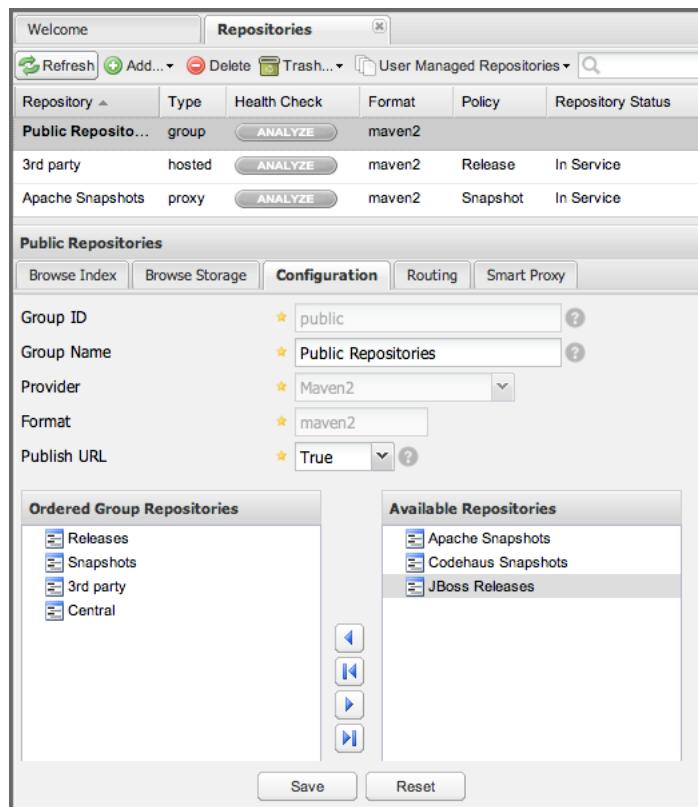


Figure 4.3: Adding New Repositories to a Nexus Group

To add the new repository to the public group, find the repository in the *Available Repositories* list on the right, click on the repository you want to add and drag it to the left to the *Ordered Group Repositories* list. Once the repository is in the *Ordered Group Repositories* list you can click and drag the repository within that list to alter the order in which a repository will be searched for a matching artifact.

**Note**

Nexus makes use of the Javascript widget library [ExtJS](#). ExtJS provides for a number of UI widgets that allow for rich interaction like the drag-drop UI for adding repositories to a group and reordering the contents of a group.

---

In the last few sections, you learned how to add a new custom repositories to a build in order to download artifacts that are not available in the Central Repository.

If you were not using a repository manager, you would have added these repositories to the repository element of your project's POM, or you would have asked all of your developers to modify `~/.m2/settings.xml` to reference two new repositories. Instead, you used the Nexus repository manager to add the two repositories to the public group. If all of the developers are configured to point to the public group in Nexus, you can freely swap in new repositories without asking your developers to change local configuration, and you've gained a certain amount of control over which repositories are made available to your development team. In addition the performance of the artifact resolving across multiple repositories will be handled by Nexus and therefore be much faster than client side resolution done by Maven each time.

## 4.6 Apache Ant and Apache Ivy

[Apache Ivy](#) is a dependency manager often used in Apache Ant builds. It supports the Maven repository format and can be configured to download dependencies that can be declared in the `ivy.xml` file. This configuration can be contained in the `ivysettings.xml`. A minimal example for resolving dependencies from a Nexus server running on `localhost` is shown in [Minimal Apache Ivy Settings](#).

### Minimal Apache Ivy Settings

```
<ivysettings>
    <settings defaultResolver="nexus"/>
    <property name="nexus-public"
              value="http://localhost:8081/nexus/content/groups/ ←
                  public">
    <resolvers>
        <ibiblio name="nexus" m2compatible="true" root="${nexus-public}" />
    </resolvers>
</ivysettings>
```

These minimal settings allow the `ivy:retrieve` task to download the declared dependencies.

---

To deploy build outputs to a Nexus repository with the `ivy:publish` task, user credentials and the URL of the target repository have to be added to `ivysettings.xml` and the `makepom` and `publish` tasks have to be configured and invoked.

Full example projects can be found in the `ant-ivy` folder of the [Nexus book examples project](#). A full build of the `simple-project`, including downloading the declared dependencies and uploading the build output to Nexus can be invoked with

```
cd ant-ivy/simple-project  
ant deploy
```

Further details about using these example projects can be found in Chapter [25](#).

## 4.7 Apache Ant and Eclipse Aether

[Eclipse Aether](#) is the dependency management component used in Apache Maven 3+. The project provides Ant tasks that can be configured to download dependencies that can be declared in `pom.xml` file or in the Ant build file directly.

This configuration can be contained in your Ant `build.xml` or a separate file that is imported. A minimal example for resolving dependencies from a Nexus server running on `localhost` is shown in [Minimal Setup for Aether Ant Tasks](#).

### Minimal Setup for Aether Ant Tasks

```
<project xmlns:aether="antlib:org.eclipse.aether.ant" ....>  
  <taskdef uri="antlib:org.eclipse.aether.ant" resource="org/eclipse/ ↵  
    aether/ant/antlib.xml">  
    <classpath>  
      <fileset dir="${aether.basedir}" includes="aether-ant-tasks-*.jar" ↵  
        />  
    </classpath>  
  </taskdef>  
  <aether:mirror id="mirror" url="http://localhost:8081/nexus/content/ ↵  
    groups/public/" mirrorOf="*"/>  
  ...  
</project>
```

These minimal settings allow the `aether:resolve` task to download the declared dependencies.

To deploy build outputs to a Nexus repository with the `aether:deploy` task, user authentication and details about the target repositories have to be added .

Full example projects can be found in the `ant-aether` folder of the [Nexus book examples project](#). A full build of the `simple-project`, including downloading the declared dependencies and uploading the build output to Nexus can be invoked with

```
cd ant-aether/simple-project  
ant deploy
```

Further details about using these example projects can be found in Chapter [25](#).

## 4.8 Gradle

**Gradle** has a built in dependency management component that supports the Maven repository format. In order to configure a Gradle project to resolve dependencies declared in `build.gradle` file, a maven repository as shown in [Minimal Gradle Setup](#) has to be declared

### Minimal Gradle Setup

```
repositories {  
    maven {  
        url "http://localhost:8081/nexus/content/groups/public"  
    }  
}
```

These minimal settings allow Gradle to download the declared dependencies.

The above setup is specific to each project. Alternatively an `init.gradle` file placed e.g., in `~/.gradle` can establish Nexus the source for dependencies in all projects. A simple implementation could look like

```
allprojects {  
    ext.RepoConfigurator = {  
        maven { url = uri('http://localhost:8081/nexus/content/groups/public') ←  
            }  
    }  
    buildscript.repositories RepoConfigurator  
    repositories RepoConfigurator
```

Other setup could be an expansion of the following example allowing file system based repositories:

```
/**  
 * init.gradle file for development using Nexus as proxy repository  
 *  
 * @author Manfred Moser <manfred@simpligility.com  
 */  
  
apply plugin:NexusRepositoryPlugin  
  
class NexusRepositoryPlugin implements Plugin<Gradle> {  
  
    final static String LOG_PREFIX = "init.gradle/NexusRepositoryPlugin:"  
  
    final Closure NexusConfig = {  
        maven {  
            name = 'standard-nexus'  
            url = 'http://localhost:8081/nexus/content/groups/public'  
        }  
        // if required you can add further repositories or groups here and ←  
        // they will  
        // be left intact if the name starts with standard-  
        // although it is better to just add those repositories in Nexus  
        // and expose them via the public group  
    }  
  
    final Closure RepoHandler = {  
        all { ArtifactRepository repo ->  
            if (repo.name.toString().startsWith("standard-") ) {  
                println "$LOG_PREFIX $repo.name at $repo.url activated as ←  
                     repository."  
            } else {  
                if (repo instanceof MavenArtifactRepository) {  
                    remove repo  
                    println "$LOG_PREFIX $repo.name at $repo.url removed."  
                } else {  
                    println "$LOG_PREFIX $repo.name kept (not a Maven repository) ←  
                        ."  
                }  
            }  
        }  
    }  
  
    void apply(Gradle gradle) {  
        // Override all project specified Maven repos with standard defined in ←  
        // here  
        gradle.allprojects{ project ->  
            println "$LOG_PREFIX Reconfiguring repositories and buildscript ←
```

```
        repositories."
project.repositories RepoHandler
project.buildscript.repositories RepoHandler

    project.repositories NexusConfig
    project.buildscript.repositories NexusConfig
}
}
}
```

Gradle init scripts can be much more powerful and customized and are explained with more examples in the [official Gradle documentation](#).

To deploy build outputs to a Nexus repository with the `uploadArchives` task, user authentication can be declared in e.g., `gradle.properties`:

```
nexusUrl=http://localhost:8081/nexus
nexusUsername=admin
nexusPassword=admin123
```

and then used in the `uploadArchives` task with a `mavenDeployer` configuration from the Maven plugin:

```
uploadArchives {
    repositories {
        mavenDeployer {
            repository(url: "${nexusUrl}/content/repositories/releases") {
                authentication(userName: nexusUsername, password: ←
                    nexusPassword)
            }
            snapshotRepository(url: "${nexusUrl}/content/repositories/←
                snapshots") {
                authentication(userName: nexusUsername, password: ←
                    nexusPassword)
            }
        }
    }
}
```

Full example projects can be found in the `gradle` folder of the [Nexus book examples project](#). A full build of the `simple-project`, including downloading the declared dependencies and uploading the build output to Nexus can be invoked with

```
cd gradle/simple-project
gradle upload
```

Further details about using these example projects can be found in Chapter [25](#).

## 4.9 SBT

[sbt](#) has a built in dependency management component and defaults to the Maven repository format. In order to configure a sbt project to resolve dependencies declared in `build.sbt` file, a `resolver` as shown in [Minimal SBT Configuration](#) has to be declared

### Minimal SBT Configuration

```
resolvers += "Nexus" at "http://localhost:8081/nexus/content/groups/public" ->
  "
```

These minimal settings allow sbt to download the declared dependencies.

To deploy build outputs to a Nexus repository with the `publish` task, user credentials can be declared in the `build.sbt` file:

```
credentials += Credentials("Sonatype Nexus Repository Manager",
  "nexus.scala-tools.org", "admin", "admin123")
```

and then used in the `publishTo` configuration:

```
publishTo <=> version { v: String =>
  val nexus = "http://localhost:8081/nexus/"
  if (v.trim.endsWith("SNAPSHOT"))
    Some("snapshots" at nexus + "content/repositories/snapshots")
  else
    Some("releases" at nexus + "content/repositories/releases")
```

Further documentation can be found in the [sbt documentation on publishing](#).

## 4.10 Leiningen

[Leiningen](#) has a built in dependency management component and defaults to the Maven repository format. As a build tool it is mostly used for projects using the Clojure language. Many libraries useful for these

projects are published to the Clojars repository. If you want to use these, you have to create two proxy repositories with the remote URL `http://clojars.org/repo/`. This repository is mixed and you therefore have to create a release and a snapshot proxy repository and then add both to the public group.

In order to configure a Leiningen project to resolve dependencies declared in the `project.clj` file, a `mirrors` section overriding the built in `central` and `clojars` repositories as shown in [Minimal Leiningen Configuration](#) has to be declared

### Minimal Leiningen Configuration

```
:mirrors {  
  "central" {:name "Nexus"  
    :url "http://localhost:8081/nexus/content/groups/public"  
    :repo-manager true}  
  # "clojars" {:name "Nexus"  
  #   :url "http://localhost:8081/nexus/content/groups/public"  
  #   :repo-manager true}  
}
```

These minimal settings allow Leiningen to download the declared dependencies.

To deploy build outputs to a Nexus repository with the `deploy` command, the target repositories have to be add to `project.clj` as `deploy-repositories`. This avoids Leiningen checking for dependencies in these repositories, which is not necessary, since they are already part of the Nexus public repository group used in `mirrors`.

```
:deploy-repositories [  
  ["snapshots" "http://localhost:8081/nexus/content/repositories/snapshots"]  
  ["releases" "http://localhost:8081/nexus/content/repositories/releases"]  
]
```

User credentials can be declared in `~/.lein/credentials.clj.gpg` or will be prompted for.

Further documentation can be found on the [Leiningen website](#).

## Chapter 5

# Using the Nexus User Interface

### 5.1 Introduction

Nexus provides anonymous access for users who only need to search repositories, browse repositories, and peruse the system feeds. This anonymous access level changes the navigation menu and some of the options available when you right-click on a repository. This read-only access displays the user interface shown in Figure 5.1.

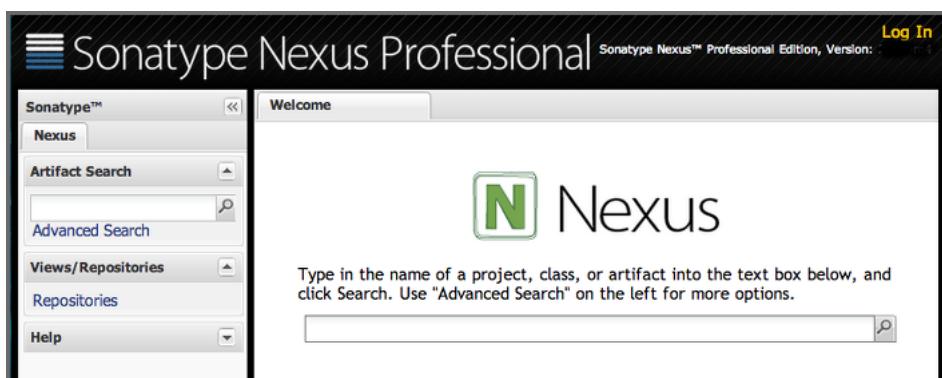


Figure 5.1: Nexus Interface for Anonymous Users

---

The Nexus user interface is used with a web browser and works best with modern browsers. Older versions such as Microsoft Internet Explorer 7 or earlier are not supported and actively blocked from using Nexus to avoid an unsatisfactory user experience. Internet Explorer 8 works up to Nexus 2.8 and is not supported for newer releases.

The user interface is separated into a number of different sections.

## **Header**

The top of the page contains the header and on the right-hand side the *Log In* button, which is replaced with a drop-down to log out, as well as navigate to the users profile. The header displays the version of Nexus and potentially the availability of a newer version.

## **Nexus Menu**

The left-hand side of the application features the Nexus menu, with its numerous submenus. The panel itself can be horizontally collapsed and expanded with the button in the top right-hand corner of the panel. Each submenu can be vertically collapsed and expanded with the button beside the title for each submenu. Depending on the access rights for the current user, different submenus and menu items are displayed.

## **Main Panel**

The main panel of the application to the right of the Nexus menu can host different tabs for different selections on the submenus in the navigation. Each tab can be closed individually and selected as the active tab.

Figure 5.2 shows a typical user interface appearance of Nexus with multiple tabs in the main panel. The activated panel *Repositories* shows a list of repositories with the current selection highlighted. The panels underneath the list show details for the selected list item.

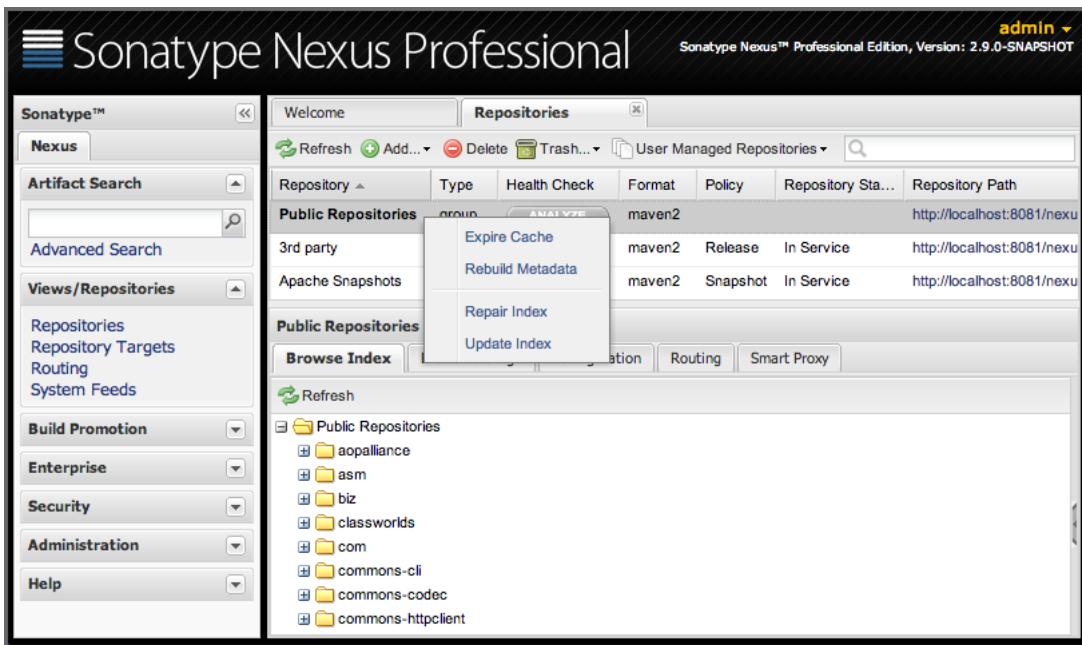


Figure 5.2: Typical Example Nexus Interface with Repository List and Details

The list header features buttons for various operations as well as an input box that allows you to filter the list by any terms used in any column. Figure 5.3 shows an example use case where a user typed "snap" in the filter box and the list of repositories only shows snapshot repositories. This filtering works for all columns in a list and can be used in most list displays in Nexus. For example you can use it to filter the users list to find disabled users, filter the routing list, the roles list and many more.

The column headers in most lists can be clicked to invoke a sorting of the list by the respective column.

Repository	Type	Health Check	Format	Policy	Repository Status
Apache Snapshots	proxy	ANALYZE	maven2	Snapshot	In Service
Codehaus Snapshots	proxy	ANALYZE	maven2	Snapshot	In Service
Snapshots	hosted	ANALYZE	maven2	Snapshot	In Service
Sonatype Grid Snapshot	proxy	ANALYZE	maven2	Snapshot	In Service

Figure 5.3: Filtering the Repository List to Display Only Snapshot Repositories

**Tip**

A right mouse button click on list items exposes a context sensitive menu of operations in some lists.

---

## 5.2 Browsing Repositories

One of the most straightforward uses of Nexus is to browse the structure of a repository. If you click on the *Repositories* menu item in the *Views/Repositories* menu, you should see the following display. The top half of Figure 5.4 shows you a list of groups and repositories along with the type of the repository and the repository status. To browse the artifacts that are stored in a local Nexus instance, click on the Browse Storage tab for a repository as shown in Figure 5.4.

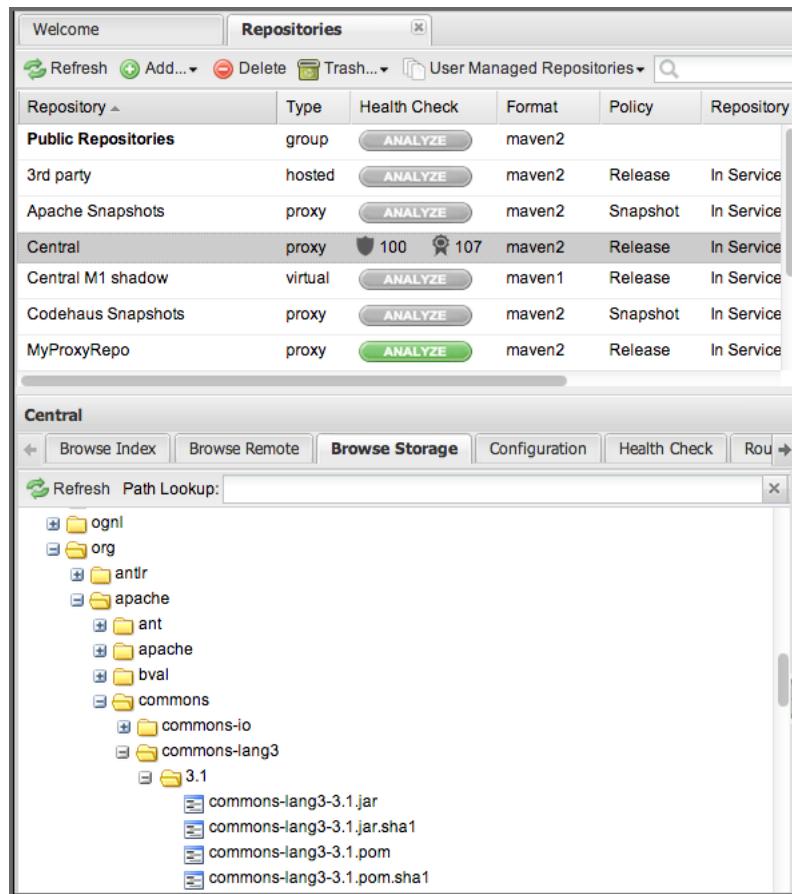


Figure 5.4: Browsing a Repository Storage

When you are browsing a repository, you can right-click on any file and download it directly to your browser. This allows you to retrieve specific artifacts manually or examine a POM file in the browser. In addition, artifacts as well as directories can be deleted using right-click.

**Note**

When browsing a remote repository you might notice that the tree doesn't contain all of the artifacts in a repository. When you browse a proxy repository, Nexus is displaying the artifacts that have been cached locally from the remote repository. If you don't see an artifact you expected to see through Nexus, it only means that Nexus has yet to cache the artifact locally. If you have enabled remote repository index downloads, Nexus will return search results that may include artifacts not yet downloaded from the remote repository. Figure 5.4, is just an example, and you may or may not have the example artifact available in your installation of Nexus.

A Nexus proxy repository acts as a local cache for a remote repository, in addition to downloading and caching artifacts locally, Nexus will also download an index of all the artifacts stored in a particular repository. When searching or browsing for artifacts, it is often more useful to search and browse the repository index. To view the repository index, click on the Browse Index tab for a particular repository to load the interface shown in Figure 5.5.

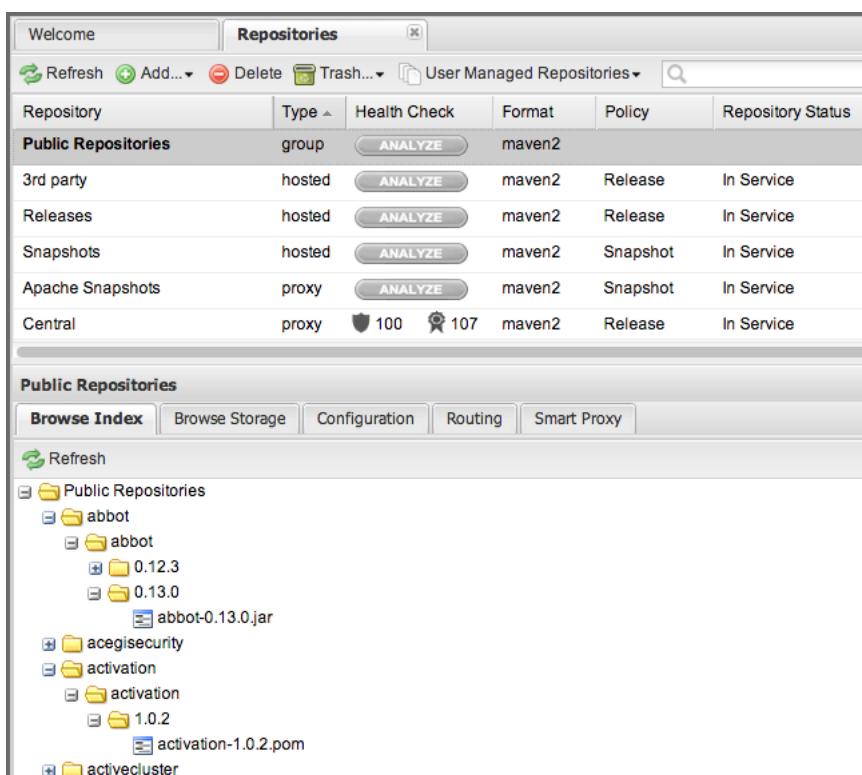


Figure 5.5: Browsing a Repository Index

## 5.3 Viewing the Artifact Information

Once you located an archive in the repository index or storage or via a search the right-hand panel will at minimum show the *Artifact Information* tab as visible in Figure 5.6. Besides showing details like the *Repository Path*, *Size*, *Checksums*, location of the artifact and other details, you are able to download and delete the artifact with the respective buttons.

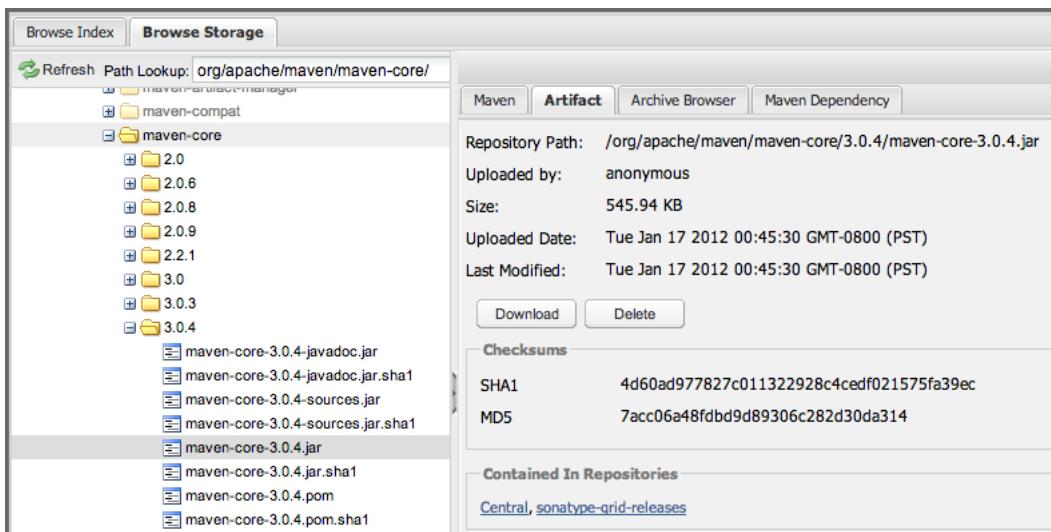


Figure 5.6: Viewing the Artifact Information

## 5.4 Viewing the Maven Information

If the artifact you are examining is a Maven-related artifact like a pom file or a jar, you will see the *Maven Information* tab in the right-hand panels. As visible in Figure 5.7, the GAV parameters are displayed above an XML snippet identifying the artifact that you can just cut and paste into a Maven pom.xml file.

The screenshot shows the Nexus Maven interface. At the top, there are tabs: 'Maven' (which is selected), 'Artifact', 'Archive Browser', and 'Maven Dependency'. Below the tabs, there are four data rows: 'Group:' (org.apache.maven), 'Artifact:' (maven-core), 'Version:' (3.0.4), and 'Extension:' (jar). Under the 'XML:' row, there is a code block containing the XML representation of the dependency:

```
<dependency>
<groupId>org.apache.maven</groupId>
<artifactId>maven-core</artifactId>
<version>3.0.4</version>
</dependency>
```

Figure 5.7: Viewing the Maven Information

## 5.5 View and Editing Artifact Metadata

Support for custom metadata for components in Maven 2 repositories is part of Nexus Professional. You can view, edit, and search for additional metadata associated to any artifact in your Nexus repositories.

The features for custom metadata usage need to be activated by adding and enabling the *Custom Metadata* capability as described in Section 6.6.

Prior to Nexus 2.7 custom metadata support was an optional plugin that needed to be installed, following the instructions in Section 22.1. The directory containing the plugin code is called nexus-custom-metadata-plugin-X.Y.Z. Install the plugin

Security privileges allow you to define "read only" as well as "write" access for custom metadata as well as grant or disallow access.

When viewing a specific artifact from browsing repository storage or a repository index or from a search, the *Artifact Metadata* tab displays the interface shown in Figure 5.8.

Key	Value	Namespace
approved	1	urn:nexus/user#
approvedBy	tobrien	urn:nexus/user#
artifactId	activemq-core	urn:maven#
baseVersion	5.3.0	urn:maven#
extension	jar	urn:maven#
groupId	org.apache.activemq	urn:maven#
path	/org/apache/activemq/activemq-cor...	urn:maven#
repositoryId	central	urn:maven#
type	urn:maven#artifact	http://www.w3.org/1999/02/22-rdf-syntax-ns#
version	5.3.0	urn:maven#

Figure 5.8: Viewing Artifact Metadata

Artifact metadata consists of a key, a value, and a namespace. Existing metadata from an artifact's POM is given a `urn:maven` namespace, and custom attributes are stored under the `urn:nexus/user` namespace.

To add a custom attribute, click on an artifact in Nexus, and select the *Artifact Metadata* tab. Click *Add...* there and a new row will be inserted into the list of attributes. Supply a *Key* and *Value* and click *Save* to update the artifact's metadata. Figure 5.9 shows the *Artifact Metadata* panel with two custom attributes: "approvedBy" and "approved".

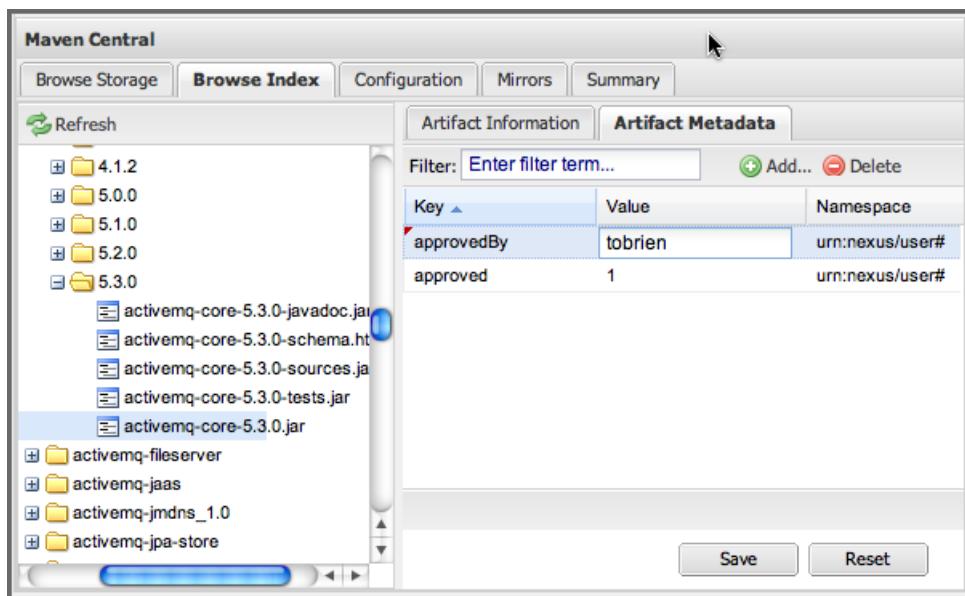


Figure 5.9: Editing Artifact Metadata

You can upload custom metadata data into Nexus using an xml file. The file will be processed as artifact metadata if it meets the following criteria:

- file extension is .n3 or .xml
- the artifact classifier is metadata

Here are example contents of a metadata file which adds additional custom metadata to an artifact with GAV of "test:project:1.0" and packaging of "jar":

```
<urn:maven:artifact#test:project:1.0::jar>
<urn:mycustomspace#repositoryId> "releases" ;
<urn:mycustomspace#mavenVersion> "2.2.1" ;
<urn:mycustomspace#releaseManager> "myusername" ;
<urn:mycustomspace#codeCoverage> ".99" .
```

A file with the above metadata content and a name of `metadata.n3` can e.g., be attached as an additional project output artifact with the build helper maven plugin.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifactId>
  <version>1.7</version>
  <executions>
    <execution>
      <id>attach-artifacts</id>
      <phase>package</phase>
      <goals>
        <goal>attach-artifact</goal>
      </goals>
      <configuration>
        <artifacts>
          <artifact>
            <file>metadata.n3</file>
            <type>n3</type>
            <classifier>metadata</classifier>
          </artifact>
        </artifacts>
      </configuration>
    </execution>
  </executions>
</plugin>
```

The metadata in the file is consumed by the Nexus custom metadata plugin and becomes available in the user interface for inspection and search. By default this metadata available for read operations only. If the repository deployment policy allows redeploys, the custom metadata can be changed.

## 5.6 Using the Artifact Archive Browser

For binary artifacts like jar files Nexus displays an *Archive Browser* panel, as visible in Figure 5.10 that allows you to view the contents of the archive. Clicking on individual files in the browser will download them and potentially display them in your browser. This can be useful for quickly checking out the contents of an archive without manually downloading and extracting it.

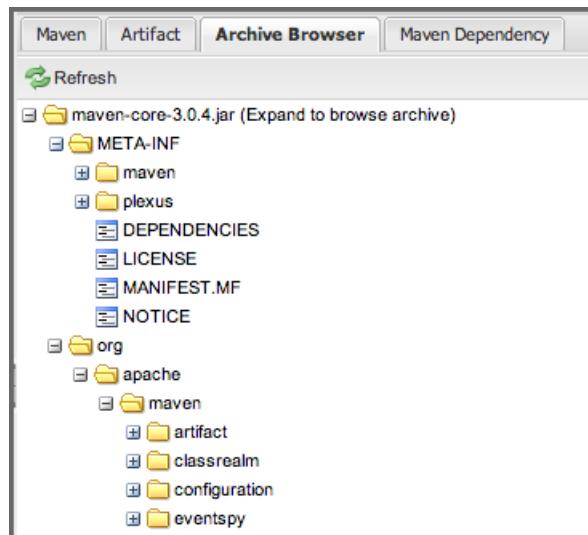


Figure 5.10: Using the Archive Browser



#### Important

The archive browser is a feature of Nexus Professional.

---

## 5.7 Inspecting the Component Dependencies

Nexus Professional provides you with the ability to browse an artifact's dependencies. Using the artifact metadata found in an artifact's POM, Nexus will scan a repository or a repository group and attempt to resolve and display an artifact's dependencies. To view an artifact's dependencies, browse the repository storage or the repository index, select an artifact (or an artifact's POM), and then click on the *Maven Dependency* tab.

On the *Maven Dependency* tab, you will see the following form elements:

### Repository

When resolving an artifact's dependencies, Nexus will query an existing repository or repository

---

group. In many cases it will make sense to select the same repository group you are referencing in your Maven settings. If you encounter any problems during the dependency resolution, you need to make sure that you are referencing a repository or a group that contains these dependencies.

## Mode

An artifact's dependencies can be listed as either a tree or a list. When dependencies are displayed in a tree, you can inspect direct dependencies and transitive dependencies. This can come in handy if you are assessing an artifact, based on the dependencies it is going to pull into your project's build. When you list dependencies as a list, Nexus is going to perform the same process used by Maven to collapse a tree of dependencies into a list of dependencies using rules to merge and override dependency versions if there are any overlaps or conflicts.

Once you have selected a repository to resolve against and a mode to display an artifact's dependencies, click on *Resolve* as shown in Figure 5.11. Clicking on this button will start the process of resolving dependencies, depending on the number of artifacts already cached by Nexus, this process can take anywhere from a few seconds to a minute. Once the resolution process is finished, you should see the artifact's dependencies, as shown in Figure 5.11.

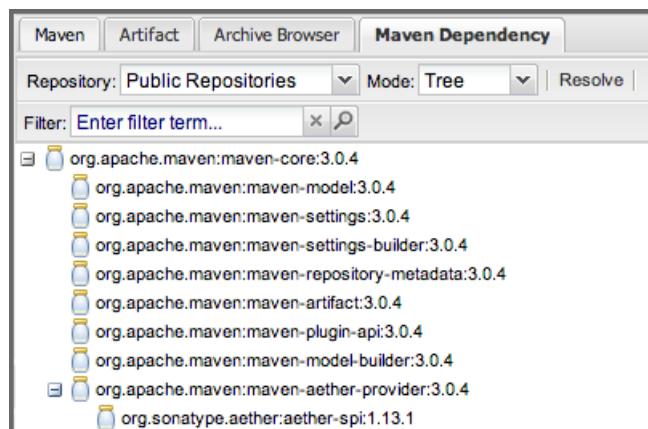


Figure 5.11: View an Artifact's Dependencies

Once you have resolved an artifact's dependencies, you can use the Filter text input to search for particular artifact dependencies. If you double-click on a row in the tree or list of dependencies, you can navigate to other artifacts within the Nexus interface.

## 5.8 Viewing Component Security and License Information

One of the added features of Nexus Professional is the usage of data from Sonatype CLM. This data contains security and license information about artifacts and is accessible for a whole repository in the Repository Health Check feature described in Chapter 12. Details about the vulnerability and security issue ratings and others can be found there as well.

The *Component Info* tab displays the security and licence information available for a specific artifact. It is available in browsing or search results, once you have selected an artifact in the search results list or repository tree view. An example search for Jetty, with the *Component Info* tab visible, is displayed in Figure 5.12. It displays the results from the *License Analysis* and any found *Security Issues*.

The *License Analysis* reveals a medium threat triggered by the fact that Non-Standard license headers were found in the source code as visible in the *Observed License(s) in Source* column. The license found in the pom.xml file associated to the project only documented Apache-2.0 or EPL-1.0 as the *Declared License(s)*.

The screenshot shows the Nexus interface with the 'Component Info' tab selected. In the main search area, 'org.mortbay.jetty' is selected as the group, and 'jetty' is the artifact. The search results table shows three versions of Jetty:

Group	Artifact	Version	Age	Popularity	Security Issues	License Threat	Download
org.mortbay.jetty	jetty	6.1.26	3.4 yrs	<div style="width: 80%;">80%</div>	2	Non-Standard	pom, jar, tests.jar, sources.jar, javadoc.jar
org.mortbay.jetty	jetty	6.1.26RC0	3.5 yrs	<div style="width: 70%;">70%</div>	2	Non-Standard	pom, jar, tests.jar, sources.jar, javadoc.jar
org.mortbay.jetty	jetty	6.1.25	3.7 yrs	<div style="width: 60%;">60%</div>	1	Non-Standard	pom, jar, tests.jar, sources.jar, javadoc.jar

Below the table, the 'License Analysis' section shows:

Threat Level	Declared License(s)	Observed License(s) in Source
Non-Standard	Apache-2.0 or EPL-1.0	Apache-2.0, EPL-1.0, Non-Standard

The 'Security Issues' section shows two entries:

Threat Level	Problem Code	Summary
5	CVE-2011-4461	Jetty 8.1.0.RC2 and earlier computes hash values for form parameters without restricting the ability to trigger hash collisions predictably, which allows remote attackers to cause a denial of service (CPU consumption) by sending many crafted parameters.
	osvdb-78117	Jetty Hash Collision Form Parameter Parsing Remote DoS

On the left, a repository tree view shows the structure of the Jetty 6.1.x releases.

Figure 5.12: Component Info Displaying Security Vulnerabilities for an Old Version of Jetty

The *Security Issues* section displays two issues with *Threat Level* values 5. The *Summary* column contains a small summary description of the security issue. The *Problem Code* column contains the codes, which link to the respective entries in the Common Vulnerabilities and Exposures CVE list as well as the Open Source Vulnerability DataBase OSVDB displayed in Figure 5.13 and Figure 5.14.

**CVE-2011-4461 (under review)**

**Description**

Jetty 8.1.0.RC2 and earlier computes hash values for form parameters without restricting the ability to trigger hash collisions predictably, which allows remote attackers to cause a denial of service (CPU consumption) by sending many crafted parameters.

**References**

- MISC:[http://www.nrunts.com/\\_downloads/advisory28122011.pdf](http://www.nrunts.com/_downloads/advisory28122011.pdf)
- MISC:<http://www.oocrt.org/advisories/oocrt-2011-003.html>
- UBUNTU:USN-1429-1

Figure 5.13: Common Vulnerabilities and Exposures CVE Entry for a Jetty Security Issue

Views This Week	Views All Time	Added to OSVDB	Last Modified	Modified (since 2008)	Percent Complete
15	597	10 months ago	About 1 month ago	3 times	70%

**Timeline**

Vendor Informed Date	Disclosure Date
2011-10-18	2011-12-28

**Description**

Jetty contains a flaw that may allow a remote denial of service. The issue is triggered when an attacker sends multiple crafted parameters which trigger hash collisions, and will result in loss of availability for the program via CPU consumption.

**Classification**

Location: Remote - Network Access  
Attack Type: Denial of Service  
Impact: Loss of Availability  
Solution: Solution Unknown  
Severity: Critical  
Disclosure: Uncoordinated Disclosure  
OSVDB: Web Related

**Solution**

OSVDB is not aware of a solution for this vulnerability.

**Products**

Unknown or Incomplete

Figure 5.14: Open Source Vulnerability DataBase OSVDB Entry for a Jetty Security Issue

**Understanding the Difference, Nexus Professional - CLM Edition** In this section, we've talked about the various ways CLM data is being used, at least at an introductory level. However, understanding the

differences between the Sonatype CLM usage in Nexus Professional and Nexus Professional CLM may still be a little unclear. Rather you are likely asking, "What do I get with Nexus Professional - Sonatype CLM Edition.

Great question. With Sonatype CLM, Nexus Professional is expanded in the two key areas.

## Policy Management

Your organization likely has a process for determining which components can be included in your applications. This could be as simple as limiting the age of the component, or more complex, like prohibiting components with a certain type of licenses or security issue.

Whatever the case, the process is supported by rules. Sonatype CLM Policy management is a way to create those rules, and then track and evaluate your application. Any time a rule is broken, that's considered a policy violation. Violations can then warn, or even prevent a release.

Here's an example of the Sonatype CLM features for Nexus Staging.

The screenshot shows the Nexus Staging Repository interface. At the top, there is a toolbar with icons for Refresh, Close, Promote, Release, and Drop. A dropdown menu labeled 'Filter by profile' is visible. Below the toolbar is a table listing repositories:

Repository	Profile	Status	Updated	Description
catchall-009	CatchAll	open	2013-Apr-10 23:27:04	Implicitly ..
nxs301-008	NXS301	closed	2013-Apr-10 23:12:13	test

Below the table, a repository named 'catchall-009' is selected. The 'Activity' tab is active, showing a tree view of recent events:

- Activities
  - open
  - close
    - Evaluating rules: clm
    - Evaluating rule: clm
    - Failed: clm**
    - 1 rule failed: clm
  - Evaluating rules: Default Always Run
  - Evaluating rule: Repository Writable
  - Passed: Repository Writable
  - All rules passed: Default Always Run
  - Close failed

To the right of the activity tree, there is a summary of the evaluation results:

**Event: Failed: clm**  
Wednesday, April 10, 2013 23:27:04 PDT (GMT-0700)

**Validation failed.**

Found 28 violations in 28 components

**7 CRITICAL**   **21 SEVERE**

**View Full Report**

Figure 5.15: Staging Repository Activity with a CLM Evaluation Failure and Details

## Component Information Panel

The Component Information Panel, or CIP, provides everything you need to know about a component. Looking at the image below, you'll notice two sections. On the left, details about the specific component are provided. On the right, the graph provides a wide variety of information including popularity, license, or security issues. You can even click on each individual version in the graph, which will then display on the left.

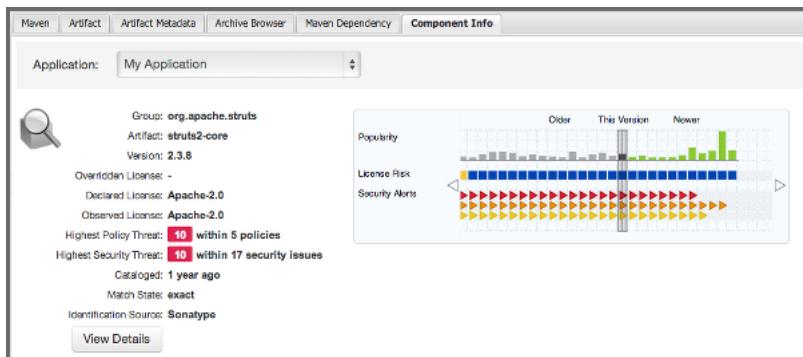


Figure 5.16: Component Information Panel Example

---

### Note

The CIP is then expanded with the View Details button which shows exactly what security or license issues were encountered, as well as any policy violations.

---

If you would like more information about these features, check out our [Sonatype CLM Repository Manager Guide](#).

## 5.9 Browsing Groups

Nexus contains ordered groups of repositories that allow you to expose a series of repositories through a single URL. More often than not, an organization is going to point Maven at the default Nexus groups *Public Repositories*. Most endusers of Nexus are not going to know what artifacts are being served from what specific repository, and they are going to want to be able to browse the public repository group.

To support this use case, Nexus allows you to browse the contents of a repository group as if it were a single merged repository with a tree structure. Figure 5.17, shows the browsing storage interface for

---

a repository group. There is no difference to the user experience of browsing a repository group vs. browsing a repository.

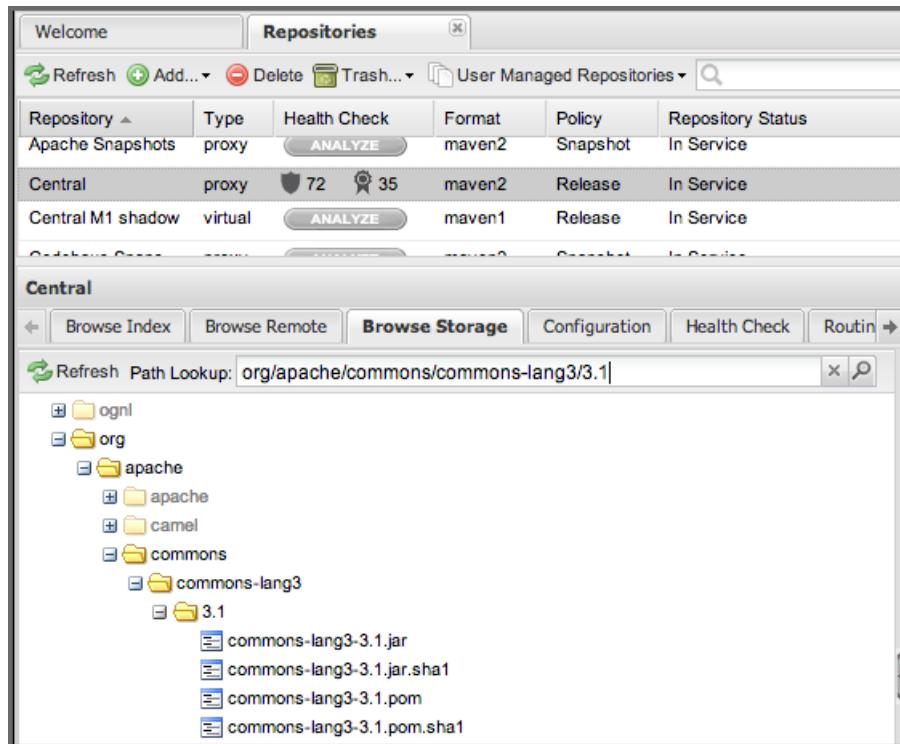


Figure 5.17: Browsing a Nexus Group

When browsing a repository group's storage, you are browsing the underlying storage for all of the repositories in a group. If a repository group contains proxy repositories, the *Browse Storage* tab will show all of the artifacts in the group that have been downloaded from the remote repositories. To browse and search all artifacts available in a group, click on the *Browse Index* tab to load the interface shown in Figure 5.18.

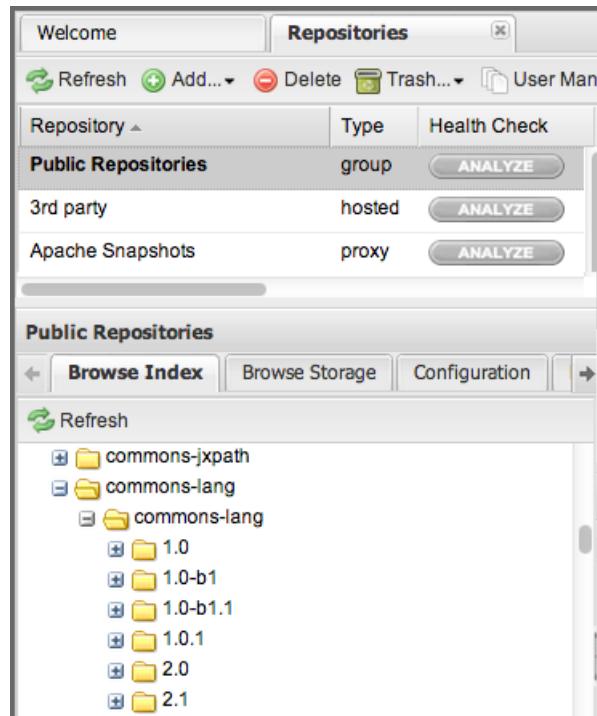


Figure 5.18: Browsing a Nexus Group Index

## 5.10 Searching for Components

### 5.10.1 Search Overview

In the left-hand navigation area, there is an *Artifact Search* text field next to a magnifying glass. To search for an artifact by groupId or artifactId, type in some text and click the magnifying glass. Typing in the search term `junit` and clicking the magnifying glass should yield a search result similar to Figure 5.19.

The screenshot shows the Nexus Repository Manager interface. At the top, there is a search bar with the query "junit". Below the search bar is a table with columns: Group, Artifact, Version, Most Popular Version, and Download. The table contains three rows of results:

Group	Artifact	Version	Most Popular Version	Download
junit	junit-dep	Show All Versions	4.10	pom
junit	junit	Show All Versions	4.10	pom, sources.jar, javadoc.jar, jar
com.googlecode.guice-junit4	guice-junit4-spring-transacti...	Show All Versions	0.2	pom, jar, tests.jar, test-sources.jar, test-javadoc.jar,

Below the table, it says "Displaying Top 40 records" and "Clear Results". On the left, there is a tree view under "Central" showing the structure of the "junit" group. On the right, there is a detailed view for the "junit" artifact, showing its Group, Artifact, Version, Extension, and XML dependency definition.

Figure 5.19: Results of an Artifact Search for "junit"

The groupId in the *Group* column and the artifactId in the *Artifact* column identify each row in the search results table. Each row represents an aggregation of all artifacts in this *Group* and *Artifact* coordinate.

The *Version* column displays the lastest version number available as well as a link to *Show All Versions*.

The *Most Popular Version* column displays the version that has the most downloads by all users accessing the Central Repository. This data can help with the selection of an appropriate version to use for a particular artifact.

The *Download* column displays direct links to all the artifacts available for the latest version. A typical list of downloadable artifacts would include the Java archive *jar*, the Maven pom.xml file *pom*, a Javadoc archive *javadoc.jar* and a Sourcecode archive *sources.jar*, but other download options are also added if more artifacts are available. Click on the link to download an artifact.

Each of the columns in the search results table can be used to sort the table in *Ascending* or *Descending* order. In addition, you can choose to add and remove columns with the sort and column drop-down options visible in Figure 5.20.

The screenshot shows a search results table with the following columns: Version, Most Popular Version, and Download. The Version column has a dropdown menu with 'Show All Versions' and 'Sort Ascending' (A-Z) / 'Sort Descending' (Z-A). The Most Popular Version column has a dropdown menu with 'Sort Ascending' (A-Z) / 'Sort Descending' (Z-A). The Download column has a dropdown menu with 'pom', 'jar, sources.jar, javadoc.jar', and 'pom, jar, tests.jar, test-jar'. A 'Columns' button is also present. Below the table, there are tabs for 'Version', 'Artifact', 'Archive Browser', and 'Maven Dependencies'. The 'Artifact' tab is selected, showing results for 'junit'. The 'Maven Dependencies' tab shows a dependency entry: <dependency><groupId>junit</groupId>

Figure 5.20: Sort and Column Options in the Search Results Table

The repository browser interface below the search results table will displays the artifact selected in the list in the repository structure with the same information panels available documented in Section 5.2. An artifact could be present in more than one repository. If this is the case, click on the value next to *Viewing Repository* to switch between multiple matching repositories.

---

**Warning**

Let me guess? You installed Nexus, ran to the search box, typed in the name of a group or an artifact, pressed search, and saw absolutely nothing. No results. Nexus isn't going to retrieve the remote repository indexes by default. You need to activate downloading of remote indexes for the three proxy repositories that Nexus are shipped with Nexus. Without these indexes, Nexus has nothing to search. Find instructions for activating index downloads in Section 6.2.

---

### 5.10.2 Advanced Search

Clicking on the (Show All Versions) link in the Version column visible in Figure 5.19 will initiate an *Advanced Search* by the groupId and artifactId of the row and result in a view similar to Figure 5.21.

The screenshot shows the Nexus search interface with the 'Search' tab selected. The search parameters are set to 'GAV Search' with 'Group: junit', 'Artifact: junit', and 'Version:'. The results table displays five versions of the junit artifact. The columns include Group, Artifact, Version, Age, Popularity, Security Issues, License Threat, and Download. The popularity bar for version 4.10 is long, while others are shorter. The security issues and license threat columns show no threats for all versions. The download links for each version are identical, pointing to 'pom, jar, sources.jar, javadoc.jar'.

Group	Artifact	Version	Age	Popularity	Security Issues	License Threat	Download
junit	junit	4.10	2.5 yrs	<div style="width: 80%;">██████████</div>	<div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div>	CPL-1.0	pom, jar, sources.jar, javadoc.jar
junit	junit	3.8.2	6.9 yrs	<div style="width: 40%;">██████</div>	<div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div>	CPL-1.0	pom, sources.jar, javadoc.jar, jar
junit	junit	3.8.1	6.9 yrs	<div style="width: 20%;">███</div>	<div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div>	CPL-1.0	pom, sources.jar, jar
junit	junit	4.11	1.4 yrs	<div style="width: 10%;">█</div>	<div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div>	CPL-1.0	pom, jar, sources.jar, javadoc.jar
junit	junit	4.8.2	3.5 yrs	<div style="width: 5%;">█</div>	<div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div> <div style="background-color: #cccccc; width: 10px; height: 10px;"></div>	CPL-1.0	pom, jar, sources.jar, javadoc.jar

Displaying Top 20 records [Clear Results](#)

Figure 5.21: Advanced Search Results for a GAV Search Activated by the Show All Versions Link

The header for the *Advanced Search* contains a selector for the type of search and one or more text input fields to define a search and a button to run a new search with the specified parameters.

The search results table contains one row per *Group* (groupId), *Artifact* (artifactId), and *Version*(version).

In addition, the *Age* column displays the age of the artifacts being available on the Central Repository. Since most artifacts are published to the Central Repository when released, this age gives you a good indication of the actual time since the release of the artifact.

The *Popularity* column shows a relative popularity as compared to the other results in the search table. This can give you a good idea on the adoption rate of a new release. For example if a newer version has a high age value, but a low popularity compared to an older version, you might want to check the upstream project and see if there is any issues stopping other users from upgrading that might affect you as well. Another reason could be that the new version does not provide significant improvements to warrant an upgrade for most users.

The *Security Issues* column shows the number of known security issues for the specific artifact. The *License Threat* column shows a colored square with blue indicating no license threat and yellow, orange and red indicating increased license threats. More information about both indicators can be seen in the *Component Info* panel below the list of artifacts for the specific artifact.

The *Download* column provides download links for all the available artifacts.

The following advanced searches are available:

### Keyword Search

Identical to the *Artifact Search* in the left-hand navigation, this search will look for the specified strings in the groupId and artifactId.

### Classname Search

Rather than looking at the coordinates of an artifact in the repository, the *Classname Search* will look at the contents of the artifacts and look for Java classes with the specified name. For example, try a search for a classname of `Pair` to see how many library authors saw a need to implement such a class, saving you from potentially implementing yet another version.

### GAV Search

The GAV search allows a search using the Maven coordinates of an artifact. These are *Group* (groupId), *Artifact* (artifactId), *Version* (version), *Packaging* (packaging), and *Classifier* (classifier). At a minimum you need to specify a group, artifact, or version in your search. An example search would be with an artifact `guice` and a classifier `no_aop` or a group of `org.glassfish.main.admin` and a packaging `war`. The default packaging is `jar`, with other values as used in the Maven packaging like `ear`, `war`, `maven-plugin`, `pom`, `ejb` and many others being possible choices.

### Checksum Search

Sometimes it is necessary to determine the version of a jar artifact in order to migrate to a qualified version. When attempting this and neither the filename nor the contents of the manifest file in the jar contain any useful information about the exact version of the jar, you can use *Checksum Search* to identify the artifact. Create a sha1 checksum, e.g., with the `shasum` command available on Linux or `fciv` on Windows, and use the created string in a checksum search. This will return one result, which will provide you with the GAV coordinates to replace the jar file with a dependency declaration.

### Metadata Search

Search for artifacts with specific metadata properties is documented in Section [5.10.3](#).

---

#### Tip

The checksum search can be a huge timesaver when migrating a legacy build system, where the used libraries are checked into the version control system as binary artifacts with no version information available.

---

## 5.10.3 Searching Artifact Metadata

To search for artifacts with specific metadata, click on the *Advanced Search* link directly below the search field in the *Artifact Search* submenu of the Nexus menu. This opens the *Search* panel and allows you to select *Metadata Search* in the search type drop-down as shown in Figure [5.22](#).

---

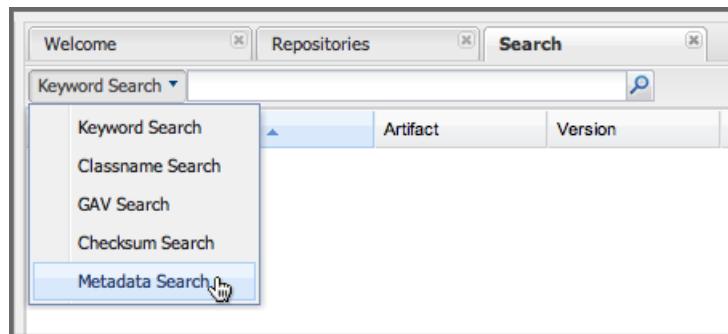


Figure 5.22: Searching Artifact Metadata

Once you select the metadata search you will see two search fields and an operator drop-down. The two search fields are the key and value of the metadata for which you are searching. The operator drop-down can be set to *Equals*, *Matches*, *Key Defined*, or *Not Equal*. *Equals* and *Not Equals* compare the value for a specific key. *Matches* allows the usage of \* to allow any characters. E.g., looking for tr\* would match true but also match tree. The *Key Defined* operator will ignore any value provided and return all artifacts with the supplied key.

A screenshot of the Nexus search results for custom metadata. The search criteria are 'Key: approved' and 'Value: 1'. The results table shows two entries: 'Maven Central (hosted) abbot abbot 0.12.3 jar' and 'Maven Central (hosted) org.apache.activemq activemq-core 5.3.0 jar'. Below the table, it says 'Displaying 2 records' and 'Clear Results'. At the bottom, there are tabs for 'Artifact Information' and 'Artifact Metadata'. The 'Artifact Information' tab is active, showing fields for Group (org.apache.act), Artifact (activemq-core), Version (5.3.0), and Download (pom, artifact). The 'Artifact Metadata' tab is shown as a separate panel on the right containing XML code: &lt;dependency&gt; &lt;groupId&gt;org.apache.activemq&lt;/groupId&gt; &lt;artifactId&gt;activemq-core&lt;/artifactId&gt; &lt;version&gt;5.3.0&lt;/version&gt; &lt;/dependency&gt;.

Figure 5.23: Metadata Search Results for Custom Metadata

Once you locate a matching artifact in the results list, click on the artifact and then select the Artifact Metadata to examine an artifacts metadata as shown in Figure 5.24.

The screenshot shows the Nexus search interface. At the top, there is a search bar with the key 'approvedBy' and value 'to\*'. Below the search bar, a table displays search results for Maven Central, showing one record: org.apache.activemq.activemq-core version 5.3.0 with packaging jar. The 'Artifact Metadata' tab is selected, showing a table of metadata entries:

Key	Value	Namespace
approved	1	urn:nexus:user#
approvedBy	tobrien	urn:nexus/user#
artifactId	activemq-core	urn:maven#
baseVersion	5.3.0	urn:maven#
extension	jar	urn:maven#
groupId	org.apache.activemq	urn:maven#
path	/org/apache/activemq/activemq-core/5....	urn:maven#
repositoryId	central	urn:maven#
type	urn:maven#artifact	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
version	5.3.0	urn:maven#

At the bottom of the interface are 'Save' and 'Reset' buttons.

Figure 5.24: Metadata Search Results for Custom Metadata

## 5.11 Search Example: Analyzing a Security Vulnerability

The following example details how you can analyze security issues of an artifact and determine a solution with the help of information available in Nexus.

You noticed the artifact with the *Group* org.springframework, the *Artifact* spring-beans and *Version* 2.5.4.

Upon further inspection of your software build and the components used, you can confirm that this artifact is indeed part of your shipping software. You might have discovered the need to investigate this initially by performing a repository health check as documented in the prior sections of Chapter 12 or an external resource such as a security mailing list.

**Tip**

Sonatype CLM for CI can help you with the detection of license and security issues during continuous integration builds. Sonatype App Health Check allows you to analyze already assembled application archives.

A GAV search for the artifact in Nexus as documented in Section 5.10 allows you to inspect the *Component Info* tab for the artifact displayed in Figure 5.25.

The screenshot shows the Nexus interface. On the left, the GAV search results for `org.springframework` are displayed, with `spring-beans` selected. The results table shows two entries: `spring-beans` version 2.5.4 and 2.5.3. Both entries have a popularity of 1, 2 security issues, and an Apache-2.0 license threat level. The packaging is listed as `pom, jar, sources.jar`. Below the table, it says "Displaying Top 68 records" and "Clear Results". On the right, the "Component Info" tab is selected. It displays the "License Analysis" section, which shows a Threat Level of Apache-2.0, Declared License(s) of Apache-2.0, and Observed License(s) in Source of Apache-2.0. Below that is the "Security Issues" section, which lists two issues: CVE-2010-1622 (Problem Code) and osvdb-65661 (Problem Code). The first issue has a Threat Level of 5 and a summary about arbitrary code execution via an HTTP request. The second issue has a Threat Level of 1 and a summary about crafted JAR file HTTP Request Arbitrary Code Execution. A sidebar on the left shows a tree view of the `spring-beans` artifact versions, with `2.5.4` expanded to show its sub-components: `spring-beans` and `spring-beans`.

Figure 5.25: GAV Search Results for `org.springframework:spring-beans` and Component Info Tab for Version 2.5.4

For example, after reading the summary and inspecting the entries for the security issues in the security databases linked in the *Problem Code* column, you decide that these issues affect your software and a

fix is required. In order to determine your next steps you search for all versions of the `spring-beans` artifact. As a result you receive the list of all versions available partially displayed in Figure 5.26. The *Security* column in the search results list displays the count of two security issues for the version 2.5.4 of the library.

Group	Artifact	Version	Age	Popularity	Security Issues	License Threat	Download sources.jar
org.springframework...	spring-beans	3.0.5.RELEASE	3.5 yrs	<div style="width: 80%;">80%</div>	<div style="width: 0%; background-color: #cccccc;"></div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	3.0.4.RELEASE	3.6 yrs	<div style="width: 70%;">70%</div>	<div style="width: 0%; background-color: #cccccc;"></div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	3.0.3.RELEASE	3.8 yrs	<div style="width: 60%;">60%</div>	<div style="width: 0%; background-color: #cccccc;"></div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	3.0.2.RELEASE	4.0 yrs	<div style="width: 50%;">50%</div>	<div style="width: 20%; background-color: #ff8c00;">2</div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	3.0.1.RELEASE	4.1 yrs	<div style="width: 40%;">40%</div>	<div style="width: 20%; background-color: #ff8c00;">2</div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	3.0.0.RELEASE	4.3 yrs	<div style="width: 30%;">30%</div>	<div style="width: 20%; background-color: #ff8c00;">2</div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	2.5.6.SEC03	2.6 yrs	<div style="width: 20%;">20%</div>	<div style="width: 0%; background-color: #cccccc;"></div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	2.5.6.SEC02	3.8 yrs	<div style="width: 10%;">10%</div>	<div style="width: 0%; background-color: #cccccc;"></div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	2.5.6.SEC01	5.0 yrs	<div style="width: 5%;">5%</div>	<div style="width: 20%; background-color: #ff8c00;">2</div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	2.5.6	5.4 yrs	<div style="width: 10%;">10%</div>	<div style="width: 20%; background-color: #ff8c00;">2</div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	2.5.5	5.8 yrs	<div style="width: 15%;">15%</div>	<div style="width: 20%; background-color: #ff8c00;">2</div>	Apache-2.0	pom, jar, sources.jar
org.springframework...	spring-beans	2.5.4	6.0 yrs	<div style="width: 5%;">5%</div>	<div style="width: 20%; background-color: #ff8c00;">2</div>	Apache-2.0	pom, jar, sources.jar

Figure 5.26: Viewing Multiple Versions of `org.springframework:spring-beans:x`

Looking at the *Security Issues* column in the results allows you to determine that with the upgrade of the library to version 2.5.6.SEC02 the count of security issues drops to zero. The same applies to version 2.5.6.SEC03, which appears to be the latest version of the 2.x version of the artifact. In addition, the table shows that early versions of the 3.x releases were affected by security issues as well.

With these results, you decide that an immediate update to version 2.5.6.SEC03 will be required as your next step. In the longer term an update to a newer version of the 3.x or even 4.x releases will follow.

The necessary steps to upgrade depend on your usage of the `spring-beans` library. A direct usage of the library will allow you to upgrade it directly. In most cases, this will require an upgrade of other SpringFramework libraries. If you are indirectly using `spring-beans` as a transitive dependency, you will need to figure out how to upgrade either the dependency causing the inclusion or override the version used.

The necessary steps will depend on the build system used, but in all cases you now have the information

at your hands detailing why you should upgrade and what to what version to upgrade to. This allows you to carry out your component lifecycle management effectively. Sonatype CLM offers tools for these migration efforts as well as various ways to monitor your development for security, license, and other issues.

## 5.12 Search Example: Resolving a License Issue

The following example details how you can analyze a license issue of an artifact found in your repository health check and determine a solution with the help of information available in Nexus. The same need for investigation might have been triggered by external means such as a need to do a legal review of all components as part of your release components and the requirement to manage a comprehensive bill of materials.

Your repository health check detail report indicated that Hibernate 3.2.7.ga might have issues due to its *Threat Level* declared as *Non-Standard*. Looking at your software artifacts you found that you are indeed using this version of Hibernate. Searching for the artifact in Nexus provides you with the search results list and the Component Info tab for the specific version displayed in Figure 5.27.

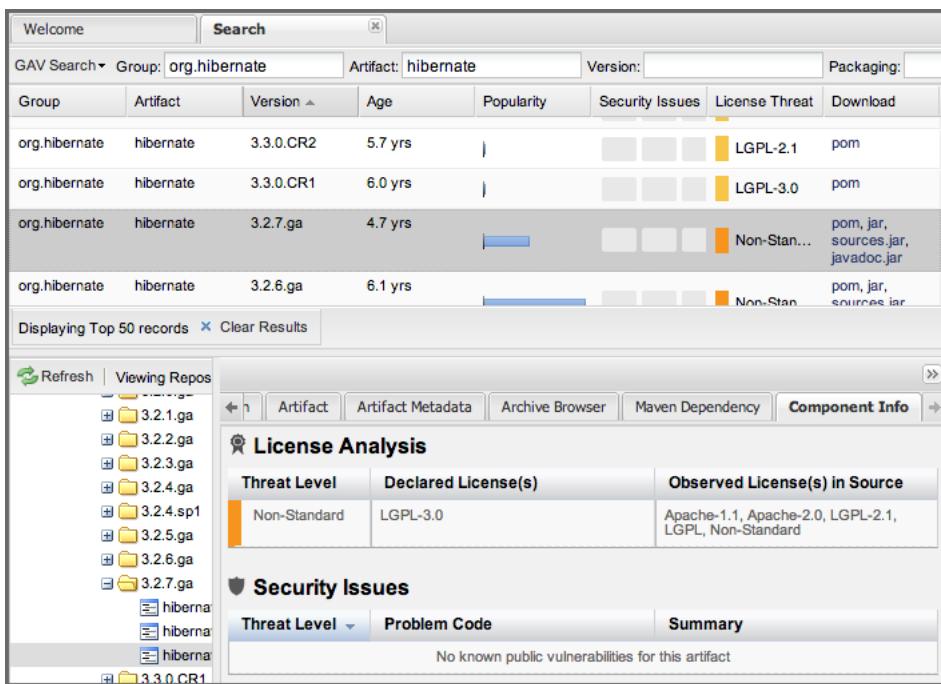


Figure 5.27: Viewing License Analysis Results for Hibernate

The *Component Info* tab displays the declared license of Hibernate is the LGPL-3.0 license. Contrary to that, the licenses observed in the source code include Apache-1.1, Apache-2.0, LGPL-2.1, LGPL and Non-Standard.

Looking at newer versions of Hibernate you find that the observed license in the source code changed to *Not-Provided*. Given this change you can conclude that the license headers in the individual source code files were removed or otherwise altered and the declared license was modified to LGPL-2.1.

With this information in hand you determine that you will need to contact your lawyers to figure out if you are okay to upgrade to a newer version of Hibernate to remedy the uncertainty of the license. In addition, you will need to decide if the LGPL-2.0 is compatible with the distribution mechanism of your software and approved by your lawyers.

In the above steps you learned how Nexus provides a lot of information allowing you to effectively carry out our component lifecycle management with a minimum amount of effort.

## 5.13 Uploading Artifacts

When your build makes use of proprietary or custom dependencies that are not available from public repositories, you will often need to find a way to make them available to developers in a custom Maven repository. Nexus ships with a preconfigured third-party repository that was designed to hold third-party dependencies that are used in your builds. To upload artifacts to a repository, select a hosted repository in the *Repositories* panel and then click on the *Artifact Upload* tab. Clicking on the *Artifact Upload* tab will display the tab shown in Figure 5.28.

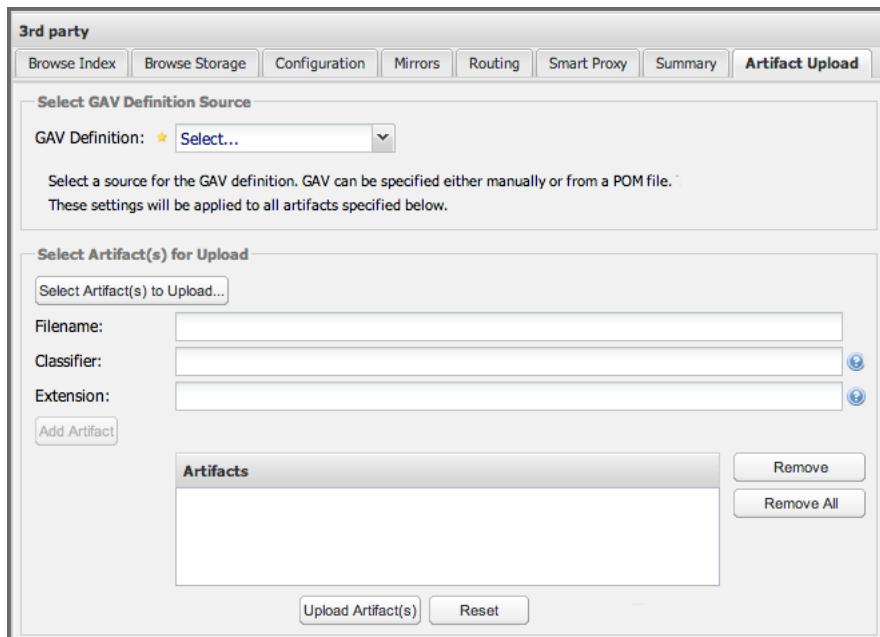


Figure 5.28: Artifact Upload Tab

To upload an artifact, click on *Select Artifact(s) to Upload...*, and select one or more artifacts from the filesystem to upload. Once you have selected an artifact, you can modify the classifier and the extension before clicking on the *Add Artifact* button. Once you have clicked on the *Add Artifact* button, you can then configure the source of the *Group, Artifact, Version* (GAV) parameters.

If the artifact you are uploading is a jar file that was created by Maven it will already have POM information embedded in it. If you are uploading a jar from a vendor you will likely need to set the group identifier, artifact identifier, and version manually. To do this, select *GAV Parameters* from the *GAV Def-*

*inition* drop-down at the top of this form. This will expose a set of form fields which will let you set the *Group*, *Artifact*, *Version*, and *Packaging* of the artifacts being uploaded. Packaging can be selected from the list or provided by typing the value into the input box.

If you would prefer to set the group, artifact, and version from a POM file associated with the uploaded artifact, select *From POM* in the *GAV Definition* drop-down. This will expose a button labeled *Select POM to Upload*. Once a POM file has been selected for upload, the name of the POM file will be displayed in the form field below this button.

---

**Tip**

Uploading a POM file allows you to add further details like dependencies to the file, which improves the quality of the upload by enabling transitive dependency management.

---

The *Artifact Upload* panel supports multiple artifacts with the same group, artifact, and version identifiers. For example, if you need to upload multiple artifacts with different classifiers, you may do so by clicking on *Select Artifact(s) for Upload* and *Add Artifact* multiple times. A common use case for this upload is to upload the pom and jar file as well as the javadoc and sources jar files file for an artifact.

## 5.14 Browsing System Feeds

Nexus provides feeds that expose system events. You can browse these feeds by clicking on *System Feeds* under the *Views/Repositories* menu. Clicking on *System Feeds* will show the panel in Figure 5.29. You can use this simple interface to browse the most recent reports of artifact deployments, cached artifacts, broken artifacts, storage changes and other events that have occurred in Nexus.

The screenshot shows the 'System Feeds' interface in Nexus. At the top, there are 'Refresh' and 'Subscribe' buttons. Below is a table with two sections: 'Feed' and 'New artifacts in all Nexus repositories (cached or deployed)'. The 'Feed' section lists several feeds with their URLs:

Feed	URL
Authentication and Authorization Events	<a href="http://localhost:8081/nexus/service/local/feeds/authenticationAndAuthorizationEvents">http://localhost:8081/nexus/service/local/feeds/authenticationAndAuthorizationEvents</a>
Broken artifacts in all Nexus repositories (checksum error...)	<a href="http://localhost:8081/nexus/service/local/feeds/brokenArtifacts">http://localhost:8081/nexus/service/local/feeds/brokenArtifacts</a>
Broken files in all Nexus repositories (checksum errors, ...)	<a href="http://localhost:8081/nexus/service/local/feeds/brokenFiles">http://localhost:8081/nexus/service/local/feeds/brokenFiles</a>
Error and Warning events	<a href="http://localhost:8081/nexus/service/local/feeds/errorAndWarningEvents">http://localhost:8081/nexus/service/local/feeds/errorAndWarningEvents</a>
New artifacts in all Nexus repositories (cached or deployed)	<a href="http://localhost:8081/nexus/service/local/feeds/recentlyDeployedArtifacts">http://localhost:8081/nexus/service/local/feeds/recentlyDeployedArtifacts</a>
New cached artifacts in all Nexus repositories (cached)	<a href="http://localhost:8081/nexus/service/local/feeds/recentlyChangedFiles">http://localhost:8081/nexus/service/local/feeds/recentlyChangedFiles</a>

The 'New artifacts in all Nexus repositories (cached or deployed)' section displays a list of recent deployments:

Title	Date
<b>com.simpligility.maven:stagingexample:1.6.1</b>	1/25 1:00 pm
The artifact 'com.simpligility.maven:stagingexample:1.6.1' in repository 'Releases' was deployed.	
<b>com.simpligility.maven:stagingexample:1.6</b>	1/25 12:52 pm
The artifact 'com.simpligility.maven:stagingexample:1.6' in repository 'Releases' was deployed. Action was initiated by user "admin". Request originated from IP address 127.0.0.1.	
<b>com.simpligility.maven:stagingexample:1.6-20120125.205105-2</b>	1/25 12:51 pm
The artifact 'com.simpligility.maven:stagingexample:1.6-20120125.205105-2' in repository 'Snapshots' was deployed. Action was initiated by user "admin". Request originated from IP address 127.0.0.1.	
<b>com.simpligility.maven:stagingexample:1.6-20120125.204958-1</b>	1/25 12:49 pm
The artifact 'com.simpligility.maven:stagingexample:1.6-20120125.204958-1' in repository 'Snapshots' was deployed. Action was initiated by user "admin". Request originated from IP address 127.0.0.1.	

Figure 5.29: Browsing Nexus System Feeds

These feeds can come in handy if you are working at a large organization with multiple development teams deploying to the same instance of Nexus. In such an arrangement, all developers in an organization can subscribe to the RSS feeds for New Deployed Artifacts as a way to ensure that everyone is aware when a new release has been pushed to Nexus. Exposing these system events as RSS feeds also opens the door to other, more creative uses of this information, such as connecting Nexus to external, automated testing systems. To access the RSS feeds for a specific feed, select the feed in the System Feeds view panel and then click on the Subscribe button. Nexus will then load the RSS feed in your browser and you can subscribe to the feed in your favorite RSS

There are a number of system feeds available in the System Feeds view, and each has a URL that resembles the following URL:

```
http://localhost:8081/nexus/service/local/feeds/recentlyChangedFiles
```

The URLs can be appended with the parameters `from` and `count` to specify the dataset viewed. E.g.

```
http://localhost:8081/nexus/service/local/feeds/recentlyDeployedArtifacts? ←  
count=100
```

Where `recentChanges` would be replaced with the identifier of the feed you were attempting to read. Available system feeds include:

- Authentication and Authorization Events
- Broken artifacts in all Nexus repositories
- Broken files in all Nexus repositories
- Error and Warning events
- New artifacts in all Nexus repositories
- New cached artifacts in all Nexus repositories
- New cached files in all Nexus repositories
- New cached release artifacts in all Nexus repositories
- New deployed artifacts in all Nexus repositories
- New deployed files in all Nexus repositories
- New deployed release artifacts in all Nexus repositories
- New files in all Nexus repositories
- New release artifacts in all Nexus repositories
- Recent artifact storage changes in all Nexus repositories
- Recent file storage changes in all Nexus repositories
- Recent release artifact storage changes in all Nexus repositories
- Repository Status Changes in Nexus
- System changes in Nexus

## 5.15 Support Tools

*Support Tools* provides a collection of useful information for monitoring and analyzing your Nexus installation. You can access the *Support Tools* in the *Administration* submenu of the Nexus menu.

---

### 5.15.1 System Information

The *System Information* tab displays a large number of configuration details related to

#### Nexus

details about the versions of Nexus and the installed plugins, Nexus install and work directory location, application host and port and a number of other properties.

#### Java Virtual Machine

all system properties like `java.runtime.name`, `os.name` and many more as known by the JVM running Nexus

#### Operating System

including environment variables like `JAVA_HOME` or `PATH` as well as details about the runtime in terms of processor, memory and threads, network connectors and storage file stores.

You can copy a subsection of the text from the panel, use the *Download* button to get a text file or use the *Print* button to produce a document.

### 5.15.2 Support Zip

The *Support ZIP* tab allows you to create a zip archive file that you can submit to Sonatype support via email or a support ticket. The checkboxes in for *Contents* and *Options* allow you to control the content of the archive.

You can include *System Information* as available in the *System Information* tab, a *Thread Dump* of the JVM currently running Nexus, your Nexus general *Configuration* as well as your *Security Configuration*, the Nexus *Log* and a *Metrics* file with network and request-related information.

The options allow you to limit the size of the included files as well as the overall file size. Pressing the *Create* button will gather all files and create the archive in `sonatype-work/nexus/support` and open a dialog to download the file to your workstation.

## 5.16 Working with Your User Profile

As a logged-in user, you can click on your user name in the top right-hand corner of the Nexus user interface to expose a drop-down with an option to *Logout* as well as to access your user *Profile* displayed in Figure 5.30.

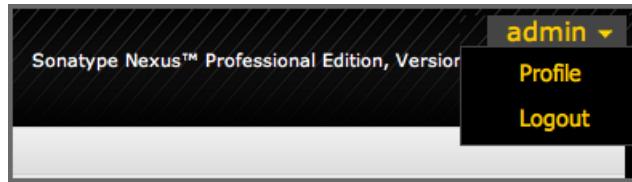


Figure 5.30: Drop Down on User Name with Profile and Logout Options

Once you have selected to display your profile, you will get access to the *Summary* section of the *Profile* tab as displayed in Figure 5.31.

A screenshot of the "Profile" dialog box. The "Summary" tab is selected. The form contains fields for "User ID" (admin), "First Name" (Administrator), "Last Name" (empty), "Email" (changeme@yourcompany.com), and "Password" (Change Password). Each field has a small blue question mark icon to its right. At the bottom are "Save" and "Reset" buttons.

Figure 5.31: Summary Section of the Profile Tab

The *Summary* section allows you to edit your *First Name*, *Last Name*, and *Email* directly in the form.

### 5.16.1 Changing Your Password

In addition to changing your name and email, the user profile allows you to change your password by clicking on the Change Password text. The dialog displayed in Figure 5.32 will be displayed and allow

you to supply your current password, and choose a new password. When you click on Change Password, your Nexus password will be changed.



Figure 5.32: Changing Your Nexus Password

The password change feature only works with the Nexus built-in XML Realm security realm. If you are using a different security realm like LDAP or Crowd, this option will not be visible.

### 5.16.2 Additional User Profile Tabs

The Profile tab can be used by other plugins and features to change or access user specific data and functionality. One such use case is the User Token access documented in Section [6.17](#).

# Chapter 6

## Configuring Nexus

Many of the configuration screens shown in this section are only available to administrative users. Nexus allows the admin user to customize the list of repositories, create repository groups, customize server settings, and create routes or "rules" that Maven will use to include or exclude artifacts from a repository.

### 6.1 Customizing Server Configuration

You can access global Nexus configuration by clicking on *Server* under *Administration* in the left-hand Nexus menu. The server configuration screens' subsections are documented in the following sections..

#### 6.1.1 SMTP Settings

Nexus sends email to users who need to recover user names and passwords, notifications for staging and a number of other uses. In order for these notifications to work, configure the SMTP server settings in this dialog.

You can configure the *Hostname* and *Port* of the SMTP server to use as well as *Username* and *Password*. The *Connection* configuration allows you to configure Nexus to use plain or secure SMTP to connect to the server or to use STARTTLS for the connection, which would upgrade the initially established, plain

---

connection to be encrypted. In all cases you will need to ensure that the correct port is used.

The *System Email* parameter defines the email address used in the `From:` header of an email sent by Nexus. Typically, this would be configured as a "Do-Not-Reply" email address or a mailbox or mailing list monitored by the administrators of the Nexus server.

Once you have configured the parameters you can use the *Test SMTP settings* button to confirm the configured parameters and the successful connection to the server. You will be asked to provide an email address that should receive a test email message. Successful sending will be confirmed in another pop up message.

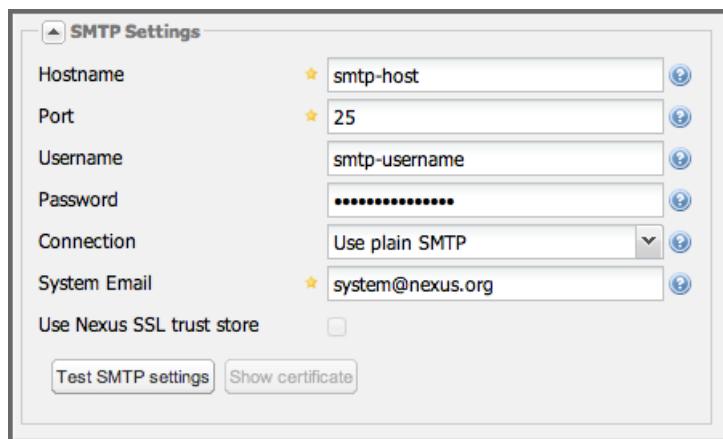


Figure 6.1: Administration SMTP Settings

### 6.1.2 HTTP Request Settings

The HTTP Request Settings allow you to configure the identifier that Nexus uses when it is making an HTTP request. You may want to change this if Nexus needs to use an HTTP Proxy, and the Proxy will only work if the User Agent is set to a specific value.

You can also add extra parameters to place on a GET request to a remote repository. You could use this to add identifying information to requests.

The amount of time Nexus will wait for a request to succeed when interacting with an external, remote repository can be configured with the Request Timeout and Request Retry Attempts settings.

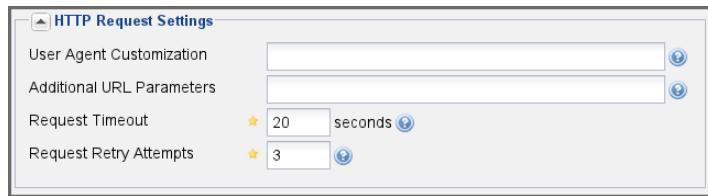


Figure 6.2: Administration HTTP Request Settings

### 6.1.3 Security Settings

The security settings displayed in Figure 6.3 allow you to activate and prioritize security realms by adding them to the *Selected Realms* list on the left and placing them higher or lower on the list.

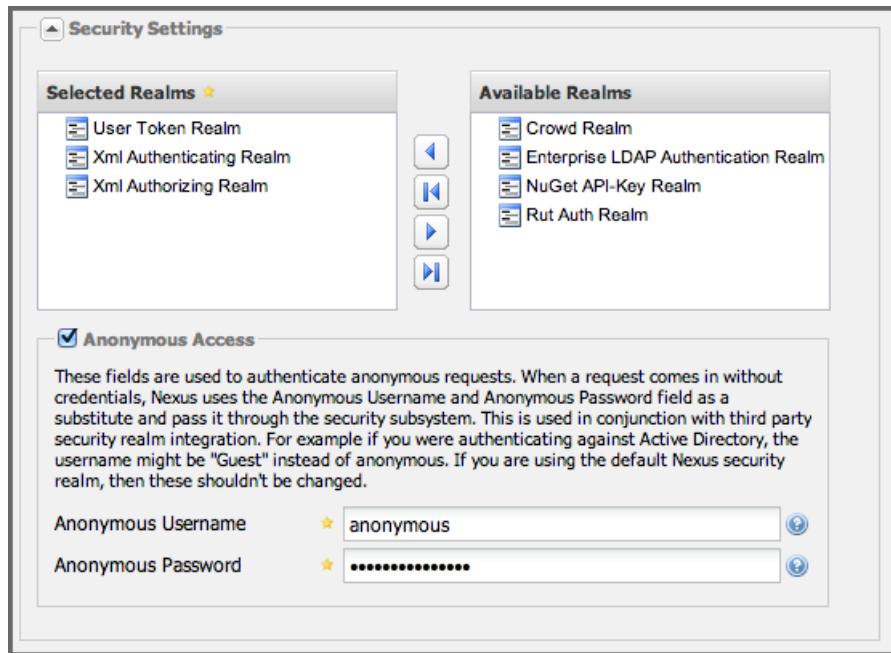


Figure 6.3: Administration Security Settings

Effectively, this configuration determines what authentication realm is used to grant a user access and the

order the realms are used.

### **Xml Authenticating and Xml Authorizing Realm**

These identify the Nexus internal storage uses XML files for storing the security details.

### **(Enterprise) LDAP Authentication Realm**

This realm identifies external storage in an LDAP system with details documented in Chapter 8.

### **Crowd Realm**

This realm identifies external storage in an Atlassian Crowd system with details documented in Chapter 9.

### **Rut Auth Realm**

This realm is external authentication in any system with the user authorization passed to Nexus in a HTTP header field with details documented in Section 6.18.

The *User Token Realm* is required for user token support documented in Section 6.17 and the *NuGet API-Key Realm* is needed for NuGet support documented in Chapter 16.

In addition, you can enable or disable anonymous access and set the username and password for anonymous access. The anonymous username and password are used to integrate with other realms that may need a special username for anonymous access. In other words, the username and password here are what we attempt to authorize when someone makes an anonymous request. You would change the anonymous username to `guest` if you wanted to integrate Nexus with Microsoft's Active Directory.

## **6.1.4 Application Server Settings**

You can change the *Base URL* for your Nexus installation, which is used when generating links in emails and RSS feeds. For example, the Nexus instance for Sonatype development is at <https://repository.sonatype.org>, and it makes use of this *Base URL* field to ensure that links in emails and RSS feeds point back to the correct public URL. Internally Nexus is running on a different port and context than the public port 80 and root context.

If you are hosting Nexus behind a proxy server and you want to make sure that Nexus always uses the specified Base URL, check the *Force Base URL* checkbox. If the Force Base URL is not checked, Nexus will craft URLs in HTTP responses based on the request URL, but it will use the Base URL when it is generating emails.

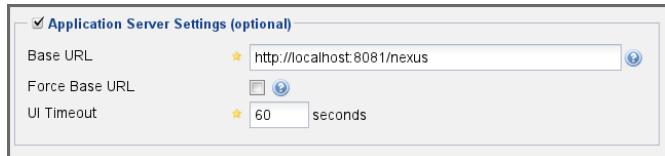


Figure 6.4: Administration Application Server Settings

---

**Tip**

These settings are especially important if Nexus is proxied by an external proxy server using a different protocol like HTTPS rather than plain HTTP known to Nexus or a different hostname like repository.somecompany.com instead of an IP number only.

---

### 6.1.5 Default HTTP and HTTPS Proxy Settings

If your Nexus instance needs to reach public repositories like the Central Repository via a proxy server, you can configure the connection to a proxy server for HTTP and a potentially a different for HTTPS connection. If you do not configure a proxy for HTTPS, the HTTP proxy server settings will be used.

You can specify *Proxy Host* and *Proxy Port* and, optionally, the authentication details for username, password, NT LAN Host and NT LAN Manager Domain. In addition, you can configure a number of hosts that can be reached directly and do not need to go through the proxy in the *Non Proxy Host* setting. Figure 6.5 shows the *Default HTTP Proxy Settings* administration interface. The HTTPS configuration interface looks the same and is found below the HTTP configuration.

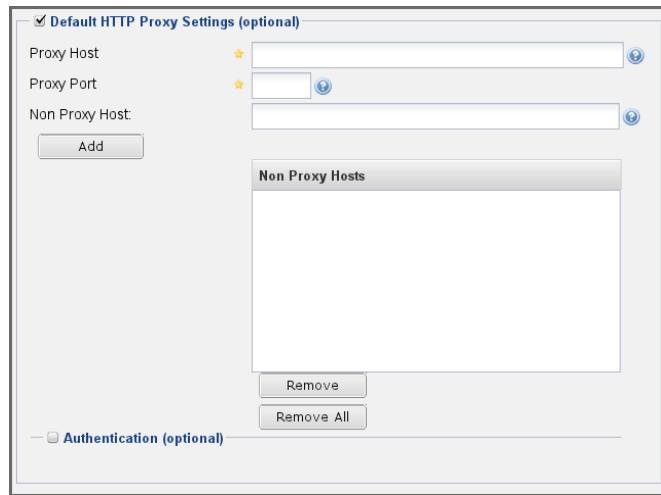


Figure 6.5: Administration Default HTTP Proxy Settings

---

**Tip**

This is a critical initial step for many Enterprise deployments of Nexus deployment, since these environments are typically secured via a HTTP/HTTPS proxy server for all outgoing internet traffic.

---

### 6.1.6 System Notification Settings

When you proxy remote repositories that are not available all the time, Nexus will automatically block and unblock them during downtimes. The *System Notification Settings* allows you define *Email Addresses* and roles for Nexus users that should receive notifications messages for these blocking and unblocking events.

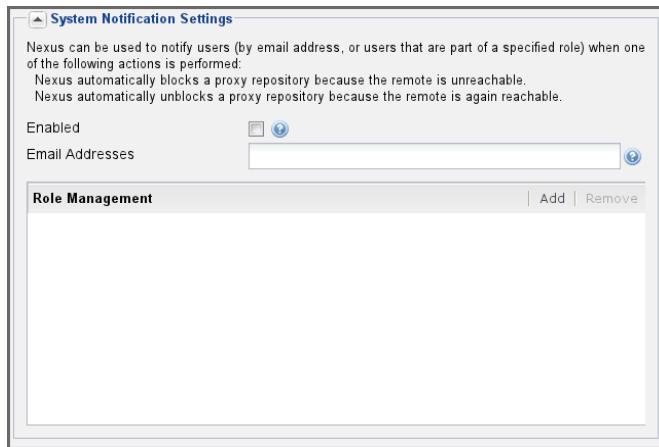


Figure 6.6: Administration System Notification Settings

### 6.1.7 PGP Key Server Information

Nexus Professional uses a PGP Key Server to retrieve PGP keys when validating artifact signatures. To add a new key server, enter the URL in the *Key Server URL* field and click on the *Add* button. To remove a key server, click on the URL you wish to remove from the list and click on the *Remove* button. Key servers are consulted in the order that they are listed in the *Key Server URLs* list. To reorder your key servers, click and drag a URL in the *Key Server URLs* list.



Figure 6.7: Administration PGP Key Server Information

### 6.1.8 New Version Availability

Nexus can notify you of new versions of Nexus via the Nexus interface. To enable this feature, check the Enable checkbox in the *New Version Availability* section of the Nexus server settings as shown in Figure 6.8.



Figure 6.8: Administration New Version Availability

## 6.2 Managing Repositories

To manage Nexus repositories, log in as the administrative user and click on *Repositories* in the *Views/Repositories* menu in the left-hand Nexus menu.

Nexus provides for three different kinds of repositories: *Proxy* Repositories, *Hosted* repositories, and *Virtual* repositories.

### 6.2.1 Proxy Repository

 *Proxy Repository* is a proxy of a remote repository. By default, Nexus ships with the following configured proxy repositories:

#### Apache Snapshots

This repository contains snapshot releases from the Apache Software Foundation.

#### Codehaus Snapshots

This repository contains snapshot releases from Codehaus.

#### Central

This is the *Central Repository* containing release artifacts. Formerly known as *Maven Central*, it is the default built-in repository for Apache Maven and directly supported in other build tools like Gradle, SBT or Ant/Ivy. For Nexus OSS, the URL <http://repo1.maven.org/maven2/> is used, while

Nexus Professional has the SSL secured version <https://secure.central.sonatype.com/maven2/> pre-configured. Nexus OSS users and users of other repository managers can purchase usage of the secured version for a nominal fee.

### 6.2.2 Hosted Repository

A *Hosted Repository* is a repository that is hosted by Nexus. Nexus ships with the following configured hosted repositories:

#### 3rd Party

This hosted repository should be used for third-party dependencies not available in the public Maven repositories. Examples of these dependencies could be commercial, proprietary libraries such as an Oracle JDBC driver that may be referenced by your organization.

#### Releases

This hosted repository is where your organization will publish internal releases.

#### Snapshots

This hosted repository is where your organization will publish internal snapshots.

### 6.2.3 Virtual Repository

A *Virtual Repository* serves as an adaptor to and from different types of repositories. Currently, Nexus supports conversion to and from Maven 1 repositories and Maven 2 repositories. In addition, you can expose any repository format as a NuGet or OBR repository. For example, a Maven 2 repository can contain OSGi Bundles, which can be exposed as a OSGi Bundle repository with the virtual repository Provider set to OBR.

By default it ships with a Central M1 shadow repository that exposes the Central repository in Maven 1 format.

### 6.2.4 Configuring Repositories

The *Repositories* window displayed in Figure 6.9 allows you to create, update and delete different repositories with the *Add*, *Delete* and *Trash* button. Use the *Refresh* button to update the displayed list of

---

repositories and repository groups. The *Trash* button allows you to empty the trash folder into which deleted components are copied, when any delete operations are performed from the Nexus user interface.

By default, the list of repositories displays the repositories configured and managed by the administrator. The drop down on the right of the *Trash* button allows you to switch the list of repositories and view the repositories managed by Nexus. There are staging repositories as documented in Chapter 11 or procurement repositories as documented in Chapter 10.

The screenshot shows the Nexus Repository Management interface. At the top, there's a navigation bar with 'Welcome' and 'Repositories'. Below it is a toolbar with 'Refresh', 'Add...', 'Delete', 'Trash...', and a dropdown for 'User Managed Repositories'. A search bar is also present. The main area is a table titled 'Repositories' with columns: Repository, Type, Health Check, Format, Policy, and Repository Status. It lists four repositories: Central (proxy, maven2, Release, In Service), Codehaus Snapshots (proxy, ANALYZE, maven2, Snapshot, In Service), MyProxyRepo (proxy, ANALYZE, maven2, Release, In Service), and NuGet Gallery (proxy, ANALYZE, nuget, Mixed, In Service). Below the table, a section titled 'Central' is expanded, showing configuration details for the 'Central' repository. These details include: Repository ID (central), Repository Name (Central), Repository Type (proxy), Provider (Maven2), Format (maven2), Repository Policy (Release), Default Local Storage Location (file:///Users/.../Nexus/sonatype-work/nexus/storage/central/), and Override Local Storage Location (empty). Under 'Remote Repository Access', the Remote Storage Location is set to https://secure.central.sonatype.com/maven2/. There are also settings for Download Remote Indexes (True), Auto Blocking Enabled (True), and File Content Validation (True). At the bottom of this section are 'Save' and 'Reset' buttons.

Figure 6.9: Repository Configuration Screen for a Proxy Repository

The list of repositories visible in Figure 6.9 allows you to access more details for each repository by selecting a specific row which displays some information for each repository in the following columns:

**Repository**

the name of the repository with repository groups displayed in bold

**Type**

the type of the repository with values of proxy, hosted or virtual for repositories or group for a repository group

**Health Check**

the result counts for a repository health check as documented in Chapter [12](#)

**Format**

the format used for the storage in the repository with values such as maven2, nuget, site or others

**Policy**

the deployment policy that applies to this repository. A policy applies only to Maven 1 and Maven 2 formatted repositories and allows usage of a *Snapshot* or a *Release* policy.

**Repository Status**

the status of the repository as well as further information about the status. For example, information about SSL certification problems or the status of the remote repository even for a currently disabled proxy repository

**Repository Path**

the direct URL path that exposes the repository via HTTP access and potentially allows access and directory browsing outside of the Nexus interface

Clicking on a column header allows you to sort the list in ascending or descending order based on the column data.

If you right-click on a row, you can trigger a number of actions on the current repository, depending on the repository type. Actions include:

**Expire Cache**

expire the cache of hosted or a proxy repository or a repository group

**Rebuild Metadata**

rebuild the metadata of a hosted Maven 2 repository

**Block Proxy / Allow Proxy**

toggle between allowing or blocking the remote repository configured in a proxy repository

**Put Out Of Service / Put in Service**

enable or disable the repository service to allow changing the availability of all components in it

---

## Repair Index / Update Index

repair or update the index of a hosted or proxy repository or a repository group

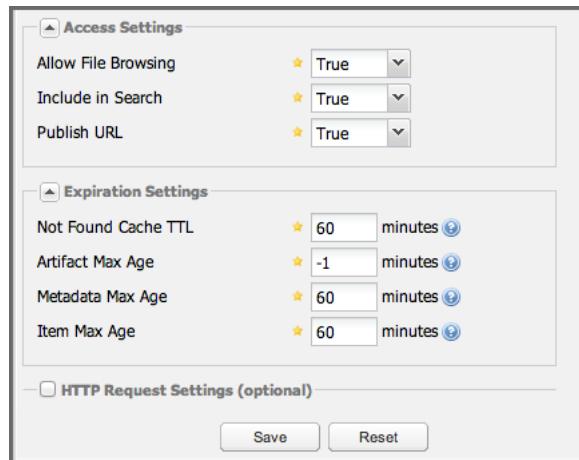


Figure 6.10: Repository Configuration Screen for a Proxy Repository

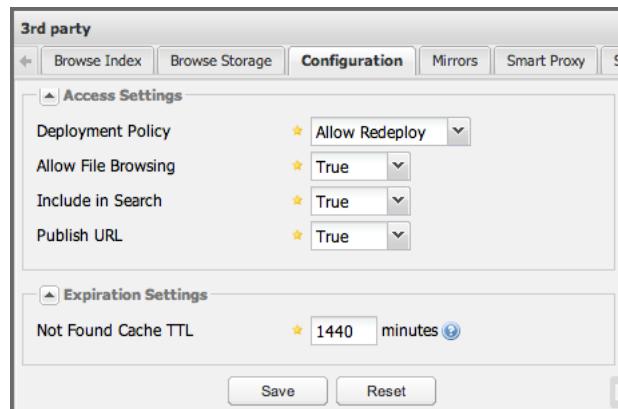


Figure 6.11: Repository Configuration Access Settings for a Hosted Repository

Figure 6.9 and Figure 6.10 show the repository configuration screen for a proxy repository in Nexus. From this screen, you can manage the settings for proxying an external repository:

## Repository ID

The repository ID is the identifier that will be used in the Nexus URL. For example, the proxy repository for the Central Repository has an ID of `central`, this means that Maven and other tools can access the repository directly at `http://localhost:8081/nexus/content/repositories/central`. The *Repository ID* must be unique in a given Nexus installation and is required.

### Repository Name

The display name for a repository is required.

### Repository Type

The type of repository (proxy, hosted, or virtual). You can't change the type of a repository as it is selected when you create a repository.

### Provider and Format

*Provider* and *Format* define in what format Nexus exposes the repository to external tools. Supported formats depend on the installed plugins. Nexus Open Source includes support for Maven 1, Maven 2 and Site repositories. Nexus Professional adds support for NuGet and OBR and additional plugins can add support for P2 and P2 Update Site and other formats.

### Repository Policy

If a proxy repository has a policy of release, then it will only access released versions from the remote repository. If a proxy repository has a policy of snapshot, it will download snapshots from the remote repository.

### Default Storage Location

Not editable, shown for reference. This is the default storage location for the local cached contents of the repository.

### Override Storage Location

You can choose to override the storage location for a specific repository. You would do this if you were concerned about storage and wanted to put the contents of a specific repository (such as `central`) in a different location.

### Remote Repository Access

This section configures proxy repositories and how Nexus interacts with the remote repository, that is being proxied.

#### Remote Storage Location

The *Remote Storage Location* needs to be configured with the URL of the remote repository, that needs to be proxied. When selecting the URL to proxy it is beneficial to avoid proxying remote repository groups. Proxying repository groups prevents some performance optimization in terms of accessing and retrieving the content of the remote repository. If you require components from the group that are found in different hosted repositories on the remote repository server it is better to create multiple proxy repositories that proxy the different hosted repositories from the remote server on your Nexus server instead of simply proxying the group.

#### Download Remote Indexes

Download the index of a remote repository can be configured with this setting. If enabled, Nexus will download the index, if it exists, and use that for its searches as well as serve that up

---

to any clients that ask for the index (like m2eclipse). The default for new proxy repositories is enabled, but all of the default repositories included in Nexus have this option disabled. To change this setting for one of the proxy repositories that ship with Nexus, change the option, save the repository, and then re-index the repository. Once this is done, artifact search will return every artifact available on the Maven Central repository.

### Auto Blocking Enabled

If Auto blocking active is set to true, Nexus will automatically block a proxy repository if the remote repository becomes unavailable. While a proxy repository is blocked, artifacts will still be served to clients from a local cache, but Nexus will not attempt to locate an artifact in a remote repository. Nexus will periodically retest the remote repository and unblock the repository once it becomes available.

### File Content Validation

If set to true, Nexus will perform a lightweight check on the content of downloaded files. This will prevent invalid content to be stored and proxied by Nexus that otherwise can happen in cases where the remote repository (or some proxy between Nexus and the remote repository) returns a HTML page instead of the requested file.

### Checksum Policy

Sets the checksum policy for a remote repository. This option is set to *Warn* by default. The possible values of this setting are:

- *Ignore* - Ignore the checksums entirely
- *Warn* - Print a warning in the log if a checksum is not correct
- *StrictIfExists* - Refuse to cache an artifact if the calculated checksum is inconsistent with a checksum in the repository. Only perform this check if the checksum file is present.
- *Strict* - Refuse to cache an artifact if the calculated checksum is inconsistent or if there is no checksum for an artifact.

### Authentication

This section allows you to set a Username, Password, NT LAN Host, and NT Lan Manager Domain for a remote repository.

### Access Settings

This section allows for the detailed configuration of access to a repository.

### Deployment Policy

This setting controls how a Hosted repository allows or disallows artifact deployment. If this policy is set to *Read Only*, no deployment is allowed. If this policy is set to *Disable Redeploy*, a client can only deploy a particular artifact once and any attempt to redeploy an artifact will result in an error. If this policy is set to *Allow Redeploy*, clients can deploy artifacts to this repository and overwrite the same artifact in subsequent deployments. This option is visible for hosted repositories as shown in Figure 6.11.

### Allow File Browsing

When set to true, users can browse the contents of the repository with a web browser.

### Include in Search

When set to true, this repository is included when you perform a search in Nexus. If this setting is false, the contents of the repository are excluded from a search.

---

### Publish URL

If this property is set to false, the repository will not be published on a URL, and you will not be able to access this repository remotely. You would set this configuration property to false if you want to prevent clients for connecting to this repository directly.

### Expiration Settings

Nexus maintains a local cache of artifacts and metadata, you can configure expiration parameters for a proxy repository. The expiration settings are:

#### Not Found Cache TTL

If Nexus fails to locate an artifact, it will cache this result for a given number of minutes. In other words, if Nexus can't find an artifact in a remote repository, it will not perform repeated attempts to resolve this artifact until the *Not Found Cache TTL* time has been exceeded. The default for this setting is 1440 minutes (or 24 hours).

#### Artifact Max Age

Tells Nexus what that maximum age of an artifact is, before it retrieves a new version from the remote repository. The default for this setting is -1 for a repository with a release policy and 1440 for a repository with snapshot policy.

#### Metadata Max Age

Nexus retrieves metadata from the remote repository. It will only retrieve updates to metadata after the *Metadata Max Age* has been exceeded. The default value for this setting is 1440 minutes (or 24 hours).

#### Item Max Age

Some items in a repository may be neither an artifact identified by the Maven GAV coordinates or metadata for such artifacts. This cache value determines the maximum age for these items before updates are retrieved.

### HTTP Request Settings

In the *HTTP Request Settings* you can change the properties of the HTTP request to the remote repository. You can also configure the *User Agent* of the request, add parameters to a request, and set the timeout and retry behavior. The HTTP request configured is the request made from Nexus to the remote repository being proxied.

Beyond these configurations in the user interface Nexus supports the usage of cookies for remote repositories authentication. Together with the feature to enable circular redirects, this enables proxying repositories like the Oracle Maven repository. The following configuration can be added to `nexus.properties` and allows a functioning proxy repository to the URL `https://maven.oracle.com`.

```
# Comma separated list of hostnames that needs to accept circular ←
# redirections
nexus.remoteStorage.enableCircularRedirectsForHosts=maven.oracle.com
# Comma separated list of hostnames that benefit from using cookies
nexus.remoteStorage.useCookiesForHosts=maven.oracle.com
```

### 6.2.5 Viewing the Summary Panel for a Repository

The *Summary* panel can be loaded by selecting a hosted, proxy, or virtual repository and then clicking on the *Summary* tab. The *Summary* tab of a hosted repository, as shown in Figure 6.12, displays the distributionManagement settings that can be used to configure Maven to publish artifacts to the hosted repository.

The screenshot shows the Nexus Repository Management interface. In the top navigation bar, the 'Repositories' tab is selected. Below it, a table lists three repositories: 'NuGet Releases', 'Releases', and 'Snapshots'. The 'Releases' repository is currently selected. The main content area is titled 'Releases' and contains tabs for 'Configuration', 'Mirrors', 'Routing', 'Smart Proxy', 'Summary', and 'Artif.'. The 'Summary' tab is active. It displays the following information:

- Repository ID: releases
- Repository Name: Releases
- Repository Type: hosted
- Repository Policy: Release
- Repository Format: maven2
- Contained in groups:
  - Public Repositories

Below this, there is a code block showing the XML configuration for distribution management:

```
<distributionManagement>
<repository>
<id>releases</id>
<url>http://localhost:8081/nexus/content/repositories/releases</url>
</repository>
</distributionManagement>
```

Figure 6.12: Repository Summary Panel for a Hosted Repository

The *Summary* panel for a proxy repository, as shown in Figure 6.13, contains all of the repository identifiers and configuration as well as a list of groups in which the repository is contained.

The screenshot shows the Nexus Repository Management interface. At the top, there's a toolbar with icons for Refresh, Add, Delete, Trash, and User Managed Repositories, along with a search bar. Below the toolbar is a table titled 'Repositories' with columns: Repository, Type, Health Check, Format, and Policy. Three repositories are listed: 'Central' (proxy, healthy, maven2, Release), 'Central M1 shadow' (virtual, healthy, maven1, Release), and 'Codehaus Snapshots' (proxy, healthy, maven2, Snapshot). A large central panel is titled 'Central' and contains tabs for Configuration, Health Check, Routing, SSL, Smart Proxy, and Summary. The 'Summary' tab is selected and displays the following information:

```
Repository ID: central
Repository Name: Central
Repository Type: proxy
Repository Policy: Release
Repository Format: maven2
Contained in groups:
  Public Repositories
Remote URL: https://secure.central.sonatype.com/maven2/
```

Figure 6.13: Repository Summary Panel for a Proxy Repository

The *Summary* panel for a virtual repository, as shown in Figure 6.14, displays repository identifiers and configuration as well as the groups in which the repository is contained.

The screenshot shows the Nexus Repository Management interface. At the top, there's a toolbar with 'Welcome', 'Repositories' (selected), and other options like 'Add...', 'Delete', 'Trash...', and 'User Managed Repositories'. Below the toolbar is a table with columns: 'Repository', 'Type', 'Health Check', 'Format', and 'Policy'. Three repositories are listed: 'Central' (proxy, healthy, maven2, Release), 'Central M1 shadow' (virtual, healthy, maven1, Release, ANALYZE button), and 'Codehaus Snapshots' (proxy, healthy, maven2, Snapshot, ANALYZE button). A modal window titled 'Central M1 shadow' is open, showing its summary details: Repository ID: central-m1, Repository Name: Central M1 shadow, Repository Type: virtual, Repository Policy: Release, Repository Format: maven1, and Contained in groups: (empty).

Figure 6.14: Repository Summary Panel for a Virtual Repository

### 6.2.6 Auto Block/Unblock of Remote Repositories

What happens when Nexus is unable to reach a remote repository? If you've defined a proxy repository and the remote repository is unavailable, Nexus will now automatically block the remote repository. Once a repository has been auto-blocked, Nexus will then periodically retest the remote repository and unblock the repository once it becomes available. You can control this behavior by changing the *Auto Blocking Enabled* setting under the *Remote Repository Access* section of the proxy repository configuration as shown in the following figure to *True*:

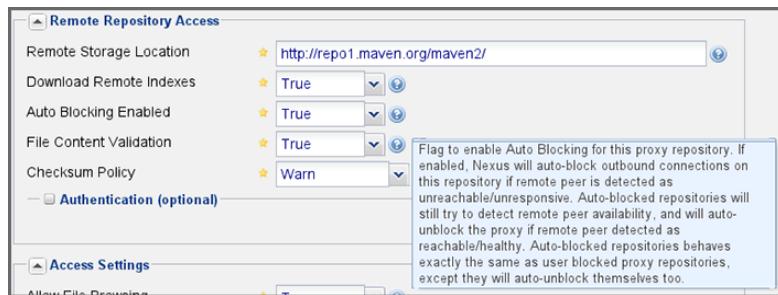


Figure 6.15: Configuring Remote Repository Auto Block/Unblock

## 6.3 Managing Repository Groups

Repository groups are a powerful feature of Nexus. They allow you to combine multiple repositories and other repository groups of the same repository format in a single repository group. This single group and the associated URL can then be used as a single access point to all components in a specific format sourced from a number of repositories.

This eases the configuration for the users and at the same time allows the Nexus administrators to add more repositories and therefore components without requiring changes on the client computers.

Use the left-hand panel *Repositories* menu item in the *Views/Repositories* menu to access the repositories and groups management interface.

To create a new repository group, press the *Add* button above the repository list and select *Repository Group*. In the configuration tab provide a *Group ID* and *Group Name*. The *Group ID* will be part of the URL to the repository group and should therefore use a limited set of characters and not contain spaces. Ideally use only lowercase letters and numbers and characters like -.

The selection of the *Provider* determines the repository *Format* and therefore the list of *Available Repositories* automatically. To add repositories to the repository group, drag them to the *Ordered Group Repositories* or use the arrows between the two lists.

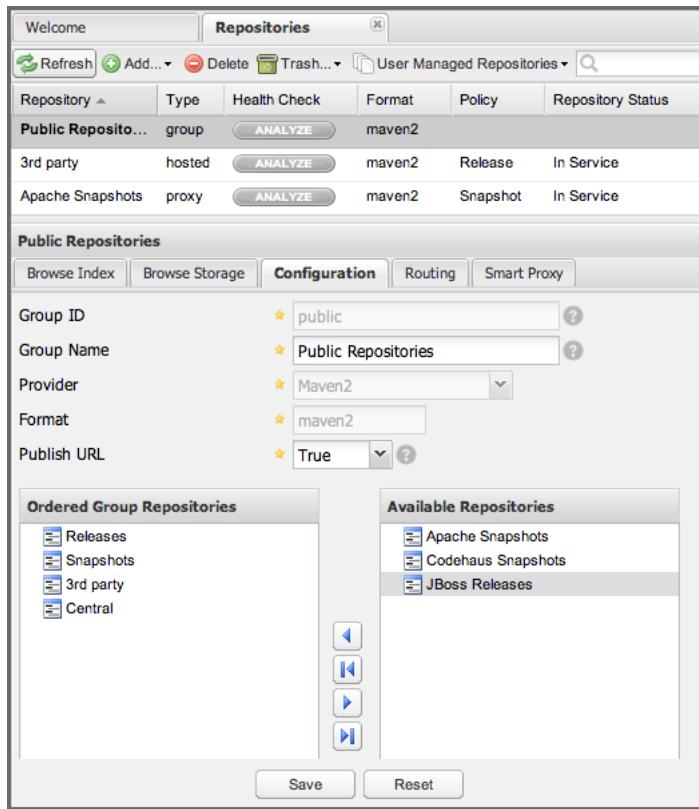


Figure 6.16: Group Configuration Screen in Nexus

Note that the order of the repositories listed in *Ordered Group Repositories* is important. When Nexus searches for a component in a group, it will return the first match. To reorder a repository in this list, click and drag the repositories and groups in the *Ordered Group Repositories* selection list.

The order of repositories or other groups in a group can be used to influence the effective metadata that will be retrieved from a repository group. We recommend placing hosted repositories higher in the list than proxy repositories within the list. For proxy repositories Nexus needs to periodically check the remote for updates, which will incur more overhead than a hosted repository lookup.

We also recommend placing repositories with a higher probability of matching the majority of components higher in this list. If most of your components are going to be retrieved from the Central Repository, putting *Central* higher in this list than a smaller, more focused repository is going to be better for performance, as Nexus is not going to interrogate the smaller remote repository for as many missing artifacts.

Once a repository group is configured it can be used from the client as discussed in e.g. Section 4.2, Section 17.5, Section 18.5 or Section 16.6 and further repositories can be added easily.

Nexus ships with one group: `public`. The Public Repositories group uses the Maven 2 repository format and combines the important external Central Repository with the hosted repositories: 3rd Party, Releases, and Snapshots.

In Section 4.2 we configured Maven via the `settings.xml` to look for artifacts in the public group managed by Nexus. Figure 6.16 shows the group configuration screen in Nexus. In this figure you can see the contents of the *Public Repositories* group.

## 6.4 Managing Routing

Routing can be considered the internal activities Nexus performs in order to determine where to look for a specific component in a repository. The routing information has an impact on the performance of component retrieval as well as determining the availability of components.

A large portion of the performance gains achievable with correct and optimized routing information is configured by Nexus itself with automatic routing, documented in Section 6.4.1. Fine grained control and further customizations in terms of access provision can be achieved with some manual routing configuration documented in Section 6.4.2.

### 6.4.1 Automatic Routing

Automatic routing is handled by Nexus on a per repository basis. You can access the configuration and further details in the Routing tab after selecting a repository in the list accessible via the *Repositories* item in the *Views/Repositories* left-hand menu.

The routing information consists of the top two levels of the directory structure of the repository and is stored in a `prefixes.txt` file. It allows Nexus to automatically route only component requests with the corresponding `groupId` values to a repository, as found in the text file. This, in turns, avoids unnecessary index or even remote repository access and therefore greatly improves performance.

Nexus generates the `prefixes.txt` file for a hosted repository and makes it available for remote downloads. Each deployment of a new component will trigger an update of the file for the hosted repository as well as the prefix files for any repository groups that contain the hosted repository. You can access it in the

---

*Routing* tab of a hosted repository as displayed in Figure 6.17 by clicking on the *Show prefix file* link on the right. In addition, the *Publishing* section shows the *Status* of the routing information, a *Message* with further details, and the date and time of the last update in the *Published On* field.

The screenshot shows the Nexus Repository Management interface. At the top, there's a table with two rows: 'Releases' (hosted, maven2, Release, In Service) and 'Snapshots' (hosted, maven2, Snapshot, In Service). Below this is a navigation bar with tabs: Index, Browse Storage, Configuration, Mirrors, Routing (which is selected and highlighted in blue), Smart Proxy, Summary, and Artifact UI. Under the Routing tab, there's a 'Publishing' section. It contains three entries: 'Status:' followed by 'Published.', 'Message:' followed by 'Prefix file published successfully.', and 'Published on:' followed by 'Mon Mar 24 2014 10:50:40 GMT-0700 (PDT)' and a link 'Show prefix file'.

Figure 6.17: Automatic Routing for a Hosted Repository

The *Routing* tab for a proxy repository displayed in Figure 6.18 contains the Discovery section. It displays the *Status* and a more detailed *Message* about the prefix file access. The *Last run* field displays the date and time of the last execution of the prefix file discovery. Such an execution can be triggered by pressing the *Update now* button. Otherwise, the *Update Interval* allows you to trigger a new discovery every one, two, three, six, nine or twelve hours or as a daily or weekly execution.

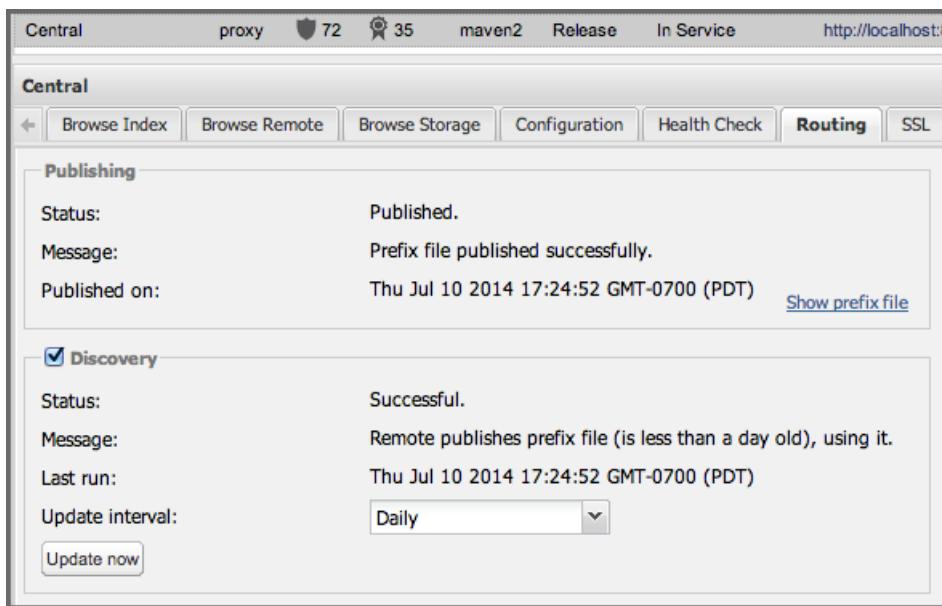


Figure 6.18: Automatic Routing for a Proxy Repository

For a proxy repository, the prefix file is either downloaded from the remote repository or a generation is attempted by scraping the remote repository. This generation is not attempted for remote Nexus repository groups, since they are too dynamic in nature and should not be proxied directly. Scraping of hosted or proxy repositories as well as Subversion-based repositories is supported.

The generation of the prefix file in all the Nexus deployments proxying each other greatly improves performance for all Nexus instances. It lowers network traffic and load on the servers, since failing requests and serving the respective HTTP error pages for a component that is not found is avoided for each component. Instead, the regularly light weight download of the prefix file establishes a good high-level knowledge of components available.

Automatic Routing is configured by Nexus automatically brings significant performance benefits to all Nexus instances proxying each other in a network and on the wider internet. It does not need to be changed apart from tweaking the update interval. To exercise even finer control than provided by Automatic Routing use Routing as documented in Section 6.4.2.

## 6.4.2 Manual Routing Configuration

Nexus routes are like filters you can apply to groups in terms of security access and general component retrieval, and can reduce the number of repositories within a group accessed in order to retrieve an artifact. The administration interface for routes can be accessed via the *Routing* menu item in the *View/Repositories* menu in the left-hand navigation panel.

Routes allow you to configure Nexus to include or exclude specific repository content paths from a particular artifact search when Nexus is trying to locate an artifact in a repository group. There are a number of different scenarios in which you might configure a route.

The most commonly configured scenario is when you want to make sure that you are retrieving artifacts in a particular group ID from a particular repository. This is especially useful when you want your own organization's artifacts from the hosted Release and Snapshot repositories only.

Routes are applicable when you are trying to resolve an artifact from a repository group. Using routes allows you to modify the repositories Nexus will consult when it tries to resolve an artifact from a group of repositories.

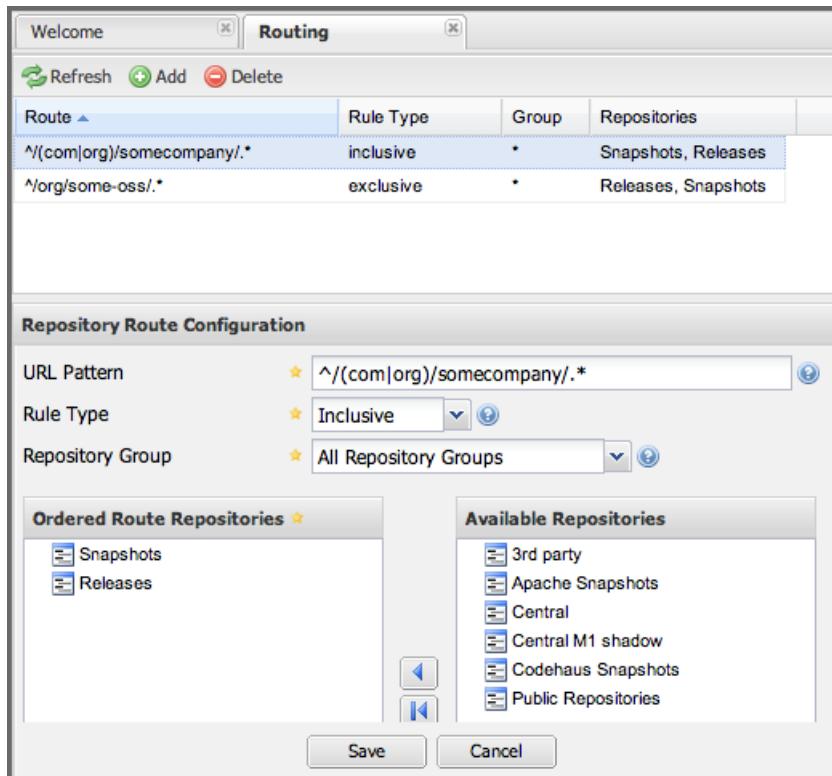


Figure 6.19: Routing Configuration Screen in Nexus

Figure 6.19 shows the *Routing* configuration screen. Clicking on a route will bring up a screen that will allow you to configure the properties of a route. The configuration options available for a route are:

### URL Pattern

Nexus uses the *URL Pattern* will use to match a request to Nexus. If the regular expression in this pattern is matched, Nexus will either include or exclude the listed repositories from a particular artifact query. In Figure 6.19 the two patterns are:

**^/ (com|org) /somecompany / .\***

This pattern would match all paths that start with either /com/somecompany/ or /org/somecompany/. The expression in the parenthesis matches either com or org, and the . \* matches zero or more characters. You would use a route like this to match your own organization's artifacts and map these requests to the hosted Releases and Snapshots repositories.

**^/org/some-oss/.\***

This pattern is used in an exclusive route. It matches every path that starts with /org/some-oss/.

This particular exclusive route excludes the local hosted Releases and Snapshots directory for all artifacts that match this path. When Nexus tries to resolve artifacts that match this path, it will exclude the Releases and Snapshots repositories.

`(?!/org/some-oss/.*) .*`

Using this pattern in an exclusive route allows you to exclude everything, except the "org/some-oss" project(s). It uses a special negative matching regular expression.

### Rule Type

Rule Type can be either *inclusive*, *exclusive* or *blocking*. An inclusive rule type defines the set of repositories that should be searched for artifacts when the URL pattern has been matched. An exclusive rule type defines repositories which should not be searched for a particular artifact. A blocking rule will completely remove accessibility to the components under the specific pattern in a specified repository group.

### Ordered Route Repositories

Nexus searches an ordered list of repositories to locate a particular artifact. This order only affects the order of routes used and not the order of the repositories searched. That order is set by the order of the repositories in the group repository's configuration.

In Figure 6.19 you can see the two dummy routes that Nexus has configured as default routes. The first route is an inclusive route, and it is provided as an example of a custom route an organization might use to make sure that internally generated artifacts are resolved from the Releases and Snapshots repositories only. If your organization's group IDs all start with `com.somecompany`, and if you deploy internally generated artifacts to the Releases and Snapshots repositories, this Route will make sure that Nexus doesn't waste time trying to resolve these artifacts from public repositories like the Central Repository or the Apache Snapshots repository.

The second dummy route is an exclusive route. This route excludes the Releases and Snapshots repositories when the request path contains `/org/some-oss`. This example might make more sense if we replaced `some-oss` with `apache` or `codehaus`. If the pattern was `/org/apache`, this rule is telling Nexus to exclude the internal Releases and Snapshots repositories when it is trying to resolve these dependencies. In other words, don't bother looking for an Apache dependency in your organization's internal repositories.

---

#### Tip

Exclusive rules will positively impact performance, since the number of repositories that qualify for locating the artifact, and therefore the search effort is reduced.

---

What if there is a conflict between two routes? Nexus will process inclusive routes before it will process the exclusive routes. Remember that routes only affect Nexus' resolution of artifacts when it is searching

---

a Group. When Nexus starts to resolve an artifact from a repository group it will start with the list of repositories in a group. If there are matching inclusive routes, Nexus will then take the intersection of the repositories in the group and the repositories in the inclusive route. The order as defined in the group will not be affected by the inclusive route. Nexus will then take the result of applying the inclusive route and apply the exclusive route to that list of repositories. The resulting list is then searched for a matching artifact.

One straightforward use of routes is to create a route that excludes the Central Repository from all searches for your own organization's hosted artifacts. If you are deploying your own artifacts to Nexus under a groupId of `org.mycompany`, and if you are not deploying these artifacts to a public repository, you can create a rule that tells Nexus not to interrogate Central for your own organization's artifacts. This will improve performance because Nexus will not need to communicate with a remote repository when it serves your own organization's artifacts. In addition to the performance benefits, excluding the Central Repository from searches for your own artifacts will reduce needless queries to the public repositories.

---

**Tip**

This practice of defining an inclusive route for your internal artifacts to only hit internal repositories is a crucial best practice of implementing a secure component lifecycle management in your organization and a recommended step for initial Nexus configuration. Without this configuration, requests for internal artifacts will be broadcasted to all configured external proxy repositories. This could lead to an information leak, where e.g., your internet traffic reveals that your organization works on a component with the artifact coordinates of `com.yourcompany.website:new-super-secret-feature:1.0-SNAPSHOT`.

---

In addition to defining inclusive and exclusive routes, you can define blocking routes. A blocking route can be created by creating a route with no repositories in the ordered list of repositories. It allows you to completely block access to artifacts with the specified pattern(s) from the group. As such, blocking routes are a simplified, coarse-grained access control.

---

**Tip**

Check out Chapter 10 for fine-grained control of artifact availability and use blocking routes sparingly.

---

To summarize, there are creative possibilities with routes that the designers of Nexus may not have anticipated, but we advise you to proceed with caution if you start relying on conflicting or overlapping routes. Use routes sparingly, and use coarse URL patterns. Remember that routes are only applied to groups and are not used when an artifact is requested from a specific repository.

---

## 6.5 Managing Scheduled Tasks

Nexus allows you to schedule tasks that will be applied to all repositories or to specific repositories on a configurable schedule. Use the *Scheduled Tasks* menu item in the *Administration* menu to access the screen, shown in Figure 6.20, that allows you to manage your Scheduled Tasks.

Enabled	Name	Type	Status	Schedule	Next Run	Last Run	Last ...
true	BackupConfig	Backup all Nexus Co...	Waiting	daily	Tue Aug...	n/a	n/a
true	EmptyTrashByHand	Empty Trash	Waiting	manual	n/a	n/a	n/a
true	GetThemAll	Download Indexes	Waiting	weekly	Mon Au...	n/a	n/a
true	Health Check: central	Check for new report...	Waiting	hourly	Tue Aug...	Mon Au...	Ok [1s]
true	Health Check: jboss...	Check for new report...	Waiting	hourly	Tue Aug...	Mon Au...	Ok [1s]

**Scheduled Task Configuration**

**Task Settings**

Backups to Keep: 5

Alert Email: (empty)

Recurrence: Daily

**Schedule Settings**

Start Date: 08/16/2012

Recurring Time: 11:08 GMT-0700 (PDT)

Save Cancel

Figure 6.20: Managing Nexus Scheduled Tasks

The list interface allows you to *Add* new tasks and *Run*, *Cancel*, and *Delete* existing tasks as well as *Refresh* the list with respective buttons above the list.

When creating or updating a scheduled task, you can configure the following properties:

### Enabled

Enable or disable a specific task.

**Name**

Provide a name to identify the task in the user interface and log files.

**Task Type**

Specify the type of action the scheduled task executes. The list of available task types is documented in more detail below.

**Task Settings**

Configure the task settings specific to the selected task type. Tasks affecting a repository have a setting called *Repository/Group* that allows you to let the task affect all repositories and groups or only a specific one.

**Alert Email**

Configure a notification email for task execution failures. If a scheduled task fails a notification email containing the task identifier and name as well as the stack trace of the failure will be sent to the configured email recipient.

**Recurrence**

Configure the schedule for the task executions. Available choices are Manual, Once, Hourly, Daily, Weekly, Monthly and Advanced. All choices provide a custom user interface for scheduling the specific recurrence. Weekly scheduling requires at least one day of the week to be selected. The Advanced setting allows you to provide a CRON expression to configure more complex schedules.

The following kinds of scheduled task types are available:

**Backup All Nexus Configuration Files**

This scheduled task will archive the contents of the `sonatype-work/nexus/conf` directory.

Once a backup has been run, the contents of the backup will be available in `sonatype-work/nexus/backup` in a series of ZIP archives that use a timestamp in the filename. This task is a feature of Nexus Professional.

**Backup npm metadata database**

A backup archive of the npm metadata database is created in the `sonatype-work/nexus/backup/npm` with a date and time stamp in the filename. This backup is intended to be used for disaster recovery in case the npm metadata database got corrupted.

**Download Indexes**

This scheduled task will cause Nexus to download indexes from remote repositories for proxied repositories. The Download Remote Indexes configuration also needs to be enabled on the proxy repository.

**Download NuGet Feed**

This task allows you to download the feed for a NuGet proxy repository. For one-time invocation, you can enable the *Clear feed cache?* setting, which will delete the cache completely and re-fetch all data. The setting *Fetch all versions?* will trigger the download of all versions of an artifact in contrast to the default behavior of getting only the latest version.

## Drop Inactive Staging Repositories

Staging repositories can be dropped by user interaction or automated systems using the Nexus Staging Maven Plugin or Ant Task or a REST API call. Heavy users of the Nexus staging features observe that some staging and build promotion repositories are inadvertently left behind. This scheduled task can be used to drop all these repositories. You can configure the duration of inactivity to include the days after the repositories are dropped as well as the status of the repositories. Any change of the staging repository like a state change from open to closed to promoted or released as well other changes to the repository meta data like a description update are counted as an activity. You can configure to *Scan open repositories*, *Scan closed repositories*, *Scan promoted repositories* and *Scan released repositories* for inactivity and therefore potentially drop them with this task. This will allow you to avoid accumulating a large number of stale staging repositories.

## Empty Trash

The Evict and Purge actions do not delete data from the Nexus working directory. They simply move data to be cleared or evicted to a trash directory under the Nexus work directory. This task deletes the data in this trash directory older than the number of days specified in the task setting *Purge Items older than (days)*.

## Evict Unused Proxied Items From Repository Caches

This scheduled task tells Nexus to delete all proxied items that haven't been "used" (referenced or retrieved by a client) in a number of days as specified in Evict Items older than (days). This can be a good job to run if you are trying to conserve storage space and do not need all of the artifacts in the future e.g., to reproduce old builds without renewed retrieval. This is particularly useful for a personal Nexus deployment with a large change rate of artifacts combined with limited diskspace.

## Expire Repository Caches

Repositories have several caches to improve performance. This task expires the caches causing Nexus to recheck the remote repository for a proxy repository or the file system for a hosted repository. You can configure the repository or group to be affected with the task setting Repository-/Group. Alternatively you can provide a Repository Path to configure the content that should be expired.

## Mirror Eclipse Update Site

The P2 plugin allows you to mirror Eclipse update sites. This task can be used to force updates of repositories that went out of sync.

## Optimize Repository Index

To speed up searches in Nexus, this task tells the internal search engine to optimize its index files. This has no affect on the indexes published by Nexus. Typically, this task does not have to run more than once a week.

## Publish Indexes

Just as Maven downloads an index from a remote repository, Nexus can publish an index in the same format. This will make it easier for people using m2eclipse or Nexus to interact with your repositories.

## Purge Broken Proxied Rubygems Metadata

This task allows you to delete the broken metadata of a proxy gem repository.

---

## Purge Nexus Timeline

Nexus maintains a lot of data that relates to the interaction between itself, proxied remote repositories, and clients on Nexus. While this information can be important for purposes of auditing, it can also take up storage space. Using this scheduled task you can tell Nexus to periodically purge this information. The setting "Purge Items older than (days)" controls the age of the data to be deleted.

## Purge Orphaned API Keys

This scheduled tasks will delete old, unused API keys generated and used by various plugins. For example, it should be scheduled when using the User Token feature or NuGet repositories. It will purge orphaned API keys e.g., after users reset their token and should be scheduled to run regularly, specifically when internal security policies for password resets and you are using an external security provider like LDAP with this requirement for resets to access Nexus.

## Rebuild Hosted Rubygems Metadata Files

This task can be used to get the metadata file for a hosted gem repository recreated based on the actual components found in the repository.

## Rebuild Maven Metadata Files

This task will rebuild the maven-metadata.xml files with the correct information and will also validate the checksums (.mh5/.sha1) for all files in the specified Repository/Group. Typically this task is run manually to repair a corrupted repository.

## Rebuild NuGet Feed

If you are using NuGet, pushing your artifacts into a NuGet hosted repository and are proxying that repository to other users, this task can be used to rebuild the feed.

## Rebuild P2 metadata and Rebuild P2 repository

These tasks can be used to rebuild the metadata or the full repository with a P2 format. You can specify a Repository/Group or a Repository Path to determine which content to affect.

## Rebuild hosted npm metadata

The npm metadata for a hosted repository can be rebuilt based on the components found in the storage of a hosted repository. The task can serve as a recovery tool in cases where the npm metadata database got corrupted or the component storage was created manually or via some external process like e.g. an rsync copying

## Remove Releases From Repository

In many use cases of a repository manager, it is necessary to keep release components for long periods of time or forever. This can be necessary for reproducibility reasons, in order to ensure users have access to old versions or even just for audit or legal reasons. However, in other use cases, there is no value in keeping old release components. One example would be a when using a continuous delivery approach onto a single deployment platform with no roll back support. In other cases, it could also be impractical due to the mere number and size of the release components.

This scheduled task allows you to trigger the deletion of release components, supporting these use cases taking care of meta data updates, and removing the need to manually delete the components or use an external system to trigger the deletion.

To configure the task, you specify the repository where release components are to be deleted as well as the number of component versions to keep for a specific groupId and artifactId coordinate. The task generates a list of all versions of a component for each groupId and artifactId coordinate combination and sorts it according to the version number. The ordering is derived by parsing the version string and supports [semantic versioning](#) with additional semantics for specific classifiers. Further details can be found in the documentation for the implementing class [GenericVersionScheme](#).

Optionally, the *Repository Target* parameter can be used to narrow down the content of the repository that is analyzed, to determine if any deletion should occur. Choosing All (Maven2) is suitable to cause all Maven 2-formatted repositories to be analysed. If you want to only target a specific groupId and artifactId combination or a number of them you can create a suitable repository target as documented in Section [6.14](#) and use it in the configuration of the scheduled task.

## Remove Snapshots from Repository

Often, you will want to remove snapshots from a snapshot repository to preserve storage space. This task supports this deletion for time stamped snapshots as created by Maven 3.x in a deployment repository. Note that configuring and running this job is not enough to reclaim disk space. You will also need to configure a scheduled job to empty the trash folder. Files are not deleted by the *Remove Snapshots* job. They are only moved into the trash folder. When you create a scheduled task to remove snapshots, you can specify the *Repository/Group* to affect as well as:

### ***Minimum Snapshot Count***

This configuration option allows you to specify a minimum number of snapshots to preserve per artifact. For example, if you configured this option with a value of 2, Nexus will always preserve at least two snapshot artifacts. A value of -1 indicates that all snapshots should be preserved.

### ***Snapshot Retention (days)***

This configuration option allows you to specify the number of days to retain snapshot artifacts. For example, if you want to make sure that you are always keeping the last three day's worth of snapshot artifacts, configure this option with a value of 3. The minimum count overrides this setting.

### ***Remove if released***

If enabled and a released artifact with the same GAV coordinates is detected all snapshots will be removed.

### ***Grace period after release (days)***

The configuration *Remove if released* causes snapshots to be deleted as soon as the scheduled task is executed. This can lead to builds that still reference the snapshot dependency to fail. This grace period parameter allows you to specify a number of days to delay the deletion, giving the respective projects referencing the snapshot dependency time to upgrade to the release component or the next snapshot version.

### ***Delete immediately***

If you want to have artifacts deleted directly rather than moved to the trash, you can enable this setting.

When doing regular deployments to a snapshot repository via a CI server, this task should be configured to run regularly.

## Repair Repositories Index

In certain cases it might be required to remove the internal index as well as the published ones of a repository. This task does that and then rebuilds the internal index by first trying to download remote indexes (if a proxy repository), then scanning the local storage and updating the internal index accordingly. Lastly, the index is published for the repository as well. There should be no need to schedule this task. But when upgrading Nexus, the upgrade instructions may sometimes include a manual step of executing this task.

## Synchronize Shadow Repository

This service synchronizes a shadow (or virtual) repository with its master repository. This task is only needed when external changes affected a source repository of a virtual repository you are using.

## Update Repositories Index

If files are deployed directly to a repository's local storage (not deployed through Nexus), you will need to instruct Nexus to update its index. When executing this task, Nexus will update its index by first downloading remote indexes (if a proxy repository) and then scan the local storage to index the new files. Lastly, the index is published for the repository as well. Normally, there should be no need to schedule this task. One possible exception would be if files are deployed directly to the local storage regularly.

## Yum: Generate Metadata

The metadata for a yum repository is created and maintained by the `createrepo` tool. This scheduled task allows you to run it for a specific repository and optionally configure the output directory.

Beyond these tasks any plugin can provide additional scheduled tasks, which will appear in the drop-down once you have installed the plugin.

The Evict and Purge actions do not delete data from the Nexus working directory. They simply move data to be cleared or evicted to a trash directory under the Nexus work directory. If you want to reclaim disk space, you need to clear the Trash on the Browse Repositories screen. If something goes wrong with a evict or clear service, you can move the data back to the appropriate storage location from the trash. You can also schedule the Empty Trash service to clear this directory on a periodic basis.

---

### Tip

In order to keep the heap usage in check it is recommended that you schedule an "optimize indexes" task to run weekly. A number of other maintenance tasks should also be scheduled for production deployments.

---

Setting up scheduled tasks adapted to your usage of Nexus is an important first step when setting up a Nexus instance. Go through the list of task types and consider your usage patterns of Nexus. Also

---

update your scheduled tasks when changing your usage. E.g., if you start to regularly deploy snapshots by introducing continuous integration server builds with deployment.

## 6.6 Accessing and Configuring Capabilities

Capabilities are features of Nexus and Nexus plugins that can be configured by a user in the generic administration view accessible in the left-hand navigation menu *Administration* under *Capabilities*.



### Warning

In many cases you will not need to configure anything in *Capabilities* unless explicitly instructed to do so by the Sonatype support team. Execute any capability changes with caution, potentially backing up your configuration before proceeding.

---

Nexus Professional ships with a number of capabilities preinstalled and allows you to enable/disable them. An example capability is *Outreach Management* displayed in Figure 6.21. The capabilities management interface supports adding new capabilities by pressing the *New* button, copying a selected capability from the list by pressing the *Duplicate* button and deleting a selected capability with the *Delete* button. Pressing the *Refresh* button updates the list of capabilities. The list of capabilities can be filtered with the search input box in the header of the list and sorted by the different columns by pressing a column header. The list uses the following columns:

#### Status

The status column does not have a title. Enabled capabilities have a green checkmark added on top of a blue icon. If an enabled capability is not fully operational the icon displays a warning sign on top of the blue icon and the entire row is surrounded with a red border; you can find out further information in a warning message below the list of the capabilities and above the individual tabs. Disabled capabilities use a greyed out icon.

#### Type

The *Type* column provides the specific type of a capability in the list.

#### Category

The *Category* is optional and details the wider context the capability belongs to.

#### Repository

The *Repository* value is optional and references the repository for which the specific capability is configured.

---

## Description

The *Description* column contains further descriptive information about the capability.

## Notes

The *Notes* column can contain user created notes about the capability.

The screenshot shows the Nexus Capabilities Management interface. At the top, there's a toolbar with 'Welcome' and 'Capabilities' tabs, and standard file operations like Refresh, New, Duplicate, and Delete. Below the toolbar is a table with columns: Type, Category, Repository, Description, and Notes. The table lists several capabilities:

Type	Category	Repository	Description	Notes
Branding			conf/branding.png	
<b>Outreach : Management</b>				
SSL: Repository	Security	Central	Central	
SSL: Repository	Security	JBoss Rel...	JBoss Releases	
Secure Central	Security		Ready: true, Tracked: 1	Automatically added o...
Smart Proxy: Identity	Smart Proxy		21EB6831-CE52390C-147BF1CD-...	

Below the table, there's a section titled 'Outreach : Management' with tabs for Summary, Settings, Status, and About. The Settings tab is active, showing configuration options:

- Enabled: A checkbox with a question mark icon.
- Base URL: A field containing <http://links.sonatype.com/products/nexus/outreach> with a star icon and a question mark icon.
- Disable Caching: A checkbox with a question mark icon.
- Override URL: An empty field with a question mark icon.

At the bottom of the settings panel are 'Save' and 'Discard' buttons.

Figure 6.21: Capabilities Management Interface with the Outreach Management Details Visible

Every capability can be inspected and configured by selecting it in the list and using the tabs underneath the list.

The *Summary* tab displays the *Type* of the capability as well as optionally the *Description*, the *Category* and the *Repository*. The *Notes* field can be used to provide a descriptive text about the capability or any other notes related to it and can be persisted by pressing the *Save* button. The *Discard* link can be used to reset any changes in the tab.

The *Settings* tab allows you to activate or deactivate the capability with the *Enabled* checkbox. Below this checkbox, each capability type has specific additional configuration parameters available. Mousing over

the help icon beside the input field or checkbox reveals further information about the specific parameter. Once you have completed the configuration, press the *Save* button. The *Discard* link can be used to reset any changes in the tab.

The *Status* tab displays a text message that details the status of the capability and any potential problems with the configuration. Depending on the capability, the reasons can vary widely. For example, the *Secure Central* capability requires Nexus to run on a JVM with specific security features. If the JVM is not suitable, an error message with further details is displayed in the *Status* column.

The *About* tab displays a descriptive text about the purpose of the capability.

Creating a new capability by pressing the *New* button will display a new form allowing you to configure the capability in a dialog. The *Type* drop-down allows you to decide what capability to create, and a selection changes the rest of the available information and configuration in the dialog. You can configure if the capability should be enabled with the *Enabled* checkbox. Once you have completed the configuration, press *Add* and the capability will be saved and appear in the list.

Many of the built-in capabilities and plugins can be configured in the *Capabilities* administration section but also in other more user friendly, targeted user interface sections, e.g., the user token feature administrated by using the interface available via the *User Token* menu item in the *Security* left-hand menu as well as by editing the user token capability. Other capabilities are internal to Nexus functionality and sometimes managed automatically by the responsible plugin. Some optional configuration like the branding plugin is only done in the capabilities administration. The branding plugin allows the customization of the icon in the top left-hand corner of the user interface header and is described in Section 6.7.

## 6.7 Customizing the Nexus Application with Branding

The branding plugin is part of Nexus Professional and allows you to customize your Nexus instance by replacing the default Sonatype Nexus logo in the top left-hand corner of the header with an image of your choice.

You can configure it by adding the *Branding* capability as documented in Section 6.6 and enabling it. By default, the branding plugin will look for the new logo in a file called `branding.png` in your Nexus data directory's `conf` folder. By default, the location is therefore `sonatype-work/nexus/conf/branding.png`. The new logo needs to be a PNG image. To blend in well in the UI, it is recommended that it is of 60 pixels height and has a transparent background.

If it fails to find a new logo, the plugin will fall back to using the default Sonatype Nexus logo.

---

Prior to Nexus 2.7, the branding plugin was an optional plugin of Nexus Professional and needed to be installed following the documentation in Section 22.1. In this case you needed to add a branding.image.path property to the *nexus.properties* file in *\$NEXUS\_HOME/conf/*:

```
branding.image.path=/data/images/nexus_logo.png
```

## 6.8 Configuring Outreach Content in Welcome Tab

The Nexus Outreach Plugin is installed and enabled by default in Nexus Open Source and Nexus Professional. It allocates space underneath the search feature on the *Welcome* tab for linking to further documentation and support resources. This data is retrieved from Sonatype servers.

In a case where this outgoing traffic from your Nexus instance or the resulting documentation and links are not desired, the plugin can be disabled. The plugin can be disabled in the settings for the *Outreach:Management* capability as documented in Section 6.6.

You can safely remove the plugin as well without any other negative side effects. To do so, simply remove the *nexus-outreach-plugin-X.Y.Z* folder in *\$NEXUS\_HOME/nexus/WEB-INF/plugin-repository/* and restart your Nexus instance.

## 6.9 Network Configuration

By default, Nexus listens on port 8081. You can change this port, by changing the value in the *\$NEXUS\_HOME/conf/nexus.properties* file shown in [Contents of conf/nexus.properties](#). To change the port, stop Nexus, change the value of applicationPort in this file, and then restart Nexus. Once you do this, you should see a log statement in *\$NEXUS\_HOME/logs/wrapper.log* telling you that Nexus is listening on the altered port.

### Contents of conf/nexus.properties

```
# Sonatype Nexus
# =====
# This is the most basic configuration of Nexus.

# Jetty section
application-port=8081
application-host=0.0.0.0
nexus-webapp=${bundleBasedir}/nexus
```

```
nexus-webapp-context-path=/nexus  
  
# Nexus section  
nexus-work=${bundleBasedir}/../sonatype-work/nexus  
runtime=${bundleBasedir}/nexus/WEB-INF
```

## 6.10 Logging

You can configure the level of logging for Nexus and all plugins as well as inspect the current log using the Nexus user interface. Access the *Logging* panel by clicking on the *Logging* menu item in the *Administration* submenu in the *Nexus* menu. Clicking on this link will display the panel shown in Figure 6.22.

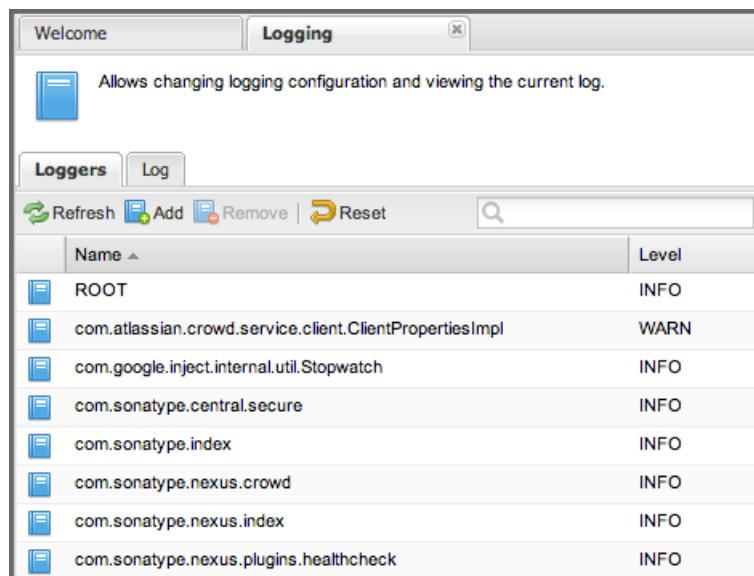


Figure 6.22: The Logging Panel with the Loggers Configuration

The *Loggers* tab in the panel allows you to configure the preconfigured loggers as well as add and remove loggers. You can modify the log level for a configured logger by clicking on the *Level* value e.g., `INFO`. It will change into a drop-down of the valid levels including `OFF`, `DEFAULT`, `INFO` and others.

If you select a row in the list of loggers, you can delete the highlighted logger by pressing the *Remove*

button above the list. The *Add* button beside it can be used to create new loggers in a dialog. You will need to know the logger you want to configure. Depending on your needs you can inspect the source of Nexus OSS and the plugins as well as the source of your own plugins to determine the related loggers or contact Sonatype support for detailed help. In addition, it is important to keep in mind that some loggers will change between Nexus and plugin versions used.

The *Reset* button allows you to remove all your custom loggers and get back to the setup shipped with Nexus.

The loggers configured in the user interface are persisted into `sonatype-work/nexus/conf/logback-overrides.xml` and override any logging levels configured in the main Nexus log file `logback-nexus.xml` as well as the other `logback-*` files. If you need to edit a logging level in those files, we suggest to edit the overrides file. This will give you access to edit the configuration in the user interface at a later stage and also ensure that the values you configure take precedence.

The *ROOT* logger level controls how verbose the Nexus logging is in general. If set to `DEBUG`, Nexus will be very verbose printing all log messages including debugging statements. If set to `ERROR`, Nexus will be far less verbose, only printing out a log statement if Nexus encounters an error. `INFO` represents an intermediate amount of logging.

---

**Tip**

When configuring logging, keep in mind that heavy logging can have a significant performance impact on an application and any changes in the user interface trigger the change to the logging immediately.

---

In Nexus releases prior to 2.7, logging configuration needed to be done by editing the `logback-nexus.xml` file found in `sonatype-work/nexus/conf`.

Once logging is configured as desired, you can inspect the impact of your configuration on the *Log* tab. It allows you to copy the log from the server to your machine by pressing the *Download* button. The *Mark* button allows you to add a custom text string into the log, so that you can create a reference point in the log file for an analysis of the file. It will insert the text you entered surrounded by \* symbols as visible in Figure 6.23.

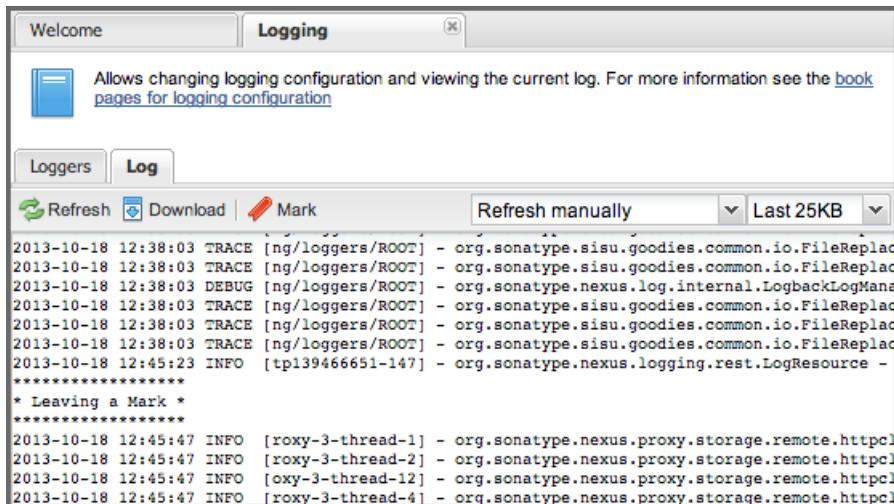


Figure 6.23: Viewing the Nexus Log with a Mark

The *Refresh* button on the left triggers an immediate update of the log. The refresh drop-down on the right can be used to trigger updates of the log in regular time intervals or manually. The size drop-down beside it allows you to control the size of the log snippet displayed in the user interface.

## 6.11 Nexus Plugins and the REST API

As documented in Section 22.1, Nexus is built as a collection of plugins supported by a core architecture and additional plugins can be installed.

You can use the Nexus Plugin Console to list all installed Nexus plugins and browse REST services made available by the installed plugins. To open the Nexus Plugin Console, click on the *Plugin Console* link in the *Administration* menu in the left-hand Nexus menu.

Once you open the Plugin Console, you will see a list of plugins installed in your Nexus installation. Clicking on a plugin in this list will display information about the plugin including name, version, status, a description, SCM information about the plugin, and the URL of the plugin's project web site and links to the plugin documentation.

The screenshot shows the Nexus Plugin Console interface. At the top, there's a header bar with tabs for 'Welcome' and 'Plugin Console'. Below the header is a toolbar with a refresh button. The main area is a table with columns for Name, Version, Description, and Status. The table lists several plugins, including the 'Nexus Core API (Restlet 1.x Plugin)' which is highlighted. A detailed view of this plugin is shown below the table, listing its name, version, status, description, SCM version, timestamp, site URL, and documentation link.

Name	Version	Description	Status
Nexus Archive Browser Plugin	2.6.2-01	Allows browsing and streaming arti...	Activated
Nexus CLM Plugin	2.6.2-01	Provides integration with Sonatype...	Activated
Nexus Capabilities Plugin	2.6.2-01	Allows Nexus to define and manag...	Activated
Nexus Configuration Backup Plugin	2.6.2-01	Adds a task that can periodically b...	Activated
<b>Nexus Core API (Restlet 1.x Plugin)</b>	2.6.2-01	Provides Nexus Core REST API.	Activated
Nexus Crypto Provider Plugin	2.6.2-01	Provides support for cryptography...	Activated
Nexus Dependency Report Plugin	2.6.2-01	Display a list of dependencies for ...	Activated
Nexus Enterprise LDAP Plugin	2.6.2-01	Adds an LDAP realm to Nexus.	Activated
Nexus ExtJS3 UI	2.6.2-01	Provides a Nexus user interface h...	Activated

**Nexus Core API (Restlet 1.x Plugin)**

Name	Nexus Core API (Restlet 1.x Plugin)
<b>Version</b>	2.6.2-01
<b>Status</b>	Activated
<b>Description</b>	Provides Nexus Core REST API.
<b>SCM Version</b>	4f08cf0b9b1dfd59cb181ae79650b6f2998e49aa
<b>SCM Timestamp</b>	N/A
<b>Site</b>	<a href="http://links.sonatype.com/products/nexus/oss/home">http://links.sonatype.com/products/nexus/oss/home</a>
<b>Documentation</b>	<a href="#">Nexus Core API (Restlet 1.x)</a>

Figure 6.24: Plugin Console

All the functionality in the Nexus user interface is accessing the REST API's provided by the different plugins. An example for the plugin documentation is the main documentation for the core Nexus API linked off the Nexus Restlet 1.x Plugin from Figure 6.24 and displayed in Figure 6.25

The screenshot shows the documentation page for the Nexus Core API (Restlet 1.x Plugin) REST API. The title is 'NEXUS CORE API (RESTLET 1.X PLUGIN) REST API'. Below the title, there are links for 'REST' and 'Data Model'. The main content is titled 'REST Resources'. It explains that the API supports the REST model and lists the following resources:

- [/all/repositories](#)
- [/artifact/maven](#)
- [/artifact/maven/content](#)
- [/artifact/maven/redirect](#)
- [/artifact/maven/resolve](#)

Figure 6.25: Documentation Website for the Core REST API

You can use the Nexus REST API to integrate Nexus in your external systems.

If your external integration uses Java, or is otherwise JVM based, then you can use the Nexus client using the dependency from [Nexus Client Core Dependency for Maven Projects](#) with the version corresponding to your Nexus server version.

### Nexus Client Core Dependency for Maven Projects

```
<dependency>
    <groupId>org.sonatype.nexus</groupId>
    <artifactId>nexus-client-core</artifactId>
    <version>2.11.2.-06</version>
</dependency>
```

Examples of using the client library can be found in the [Nexus Maven Plugins](#) or the [Nexus Ant Tasks](#).

The REST API can be invoked from many other programming and scripting languages. A simple example of using the `curl` command in a shell script is displayed in [A curl Invocation Loading the List of Users from Nexus](#).

### A curl Invocation Loading the List of Users from Nexus

```
curl -X GET -u admin:admin123 http://localhost:8081/nexus/service/local/ ←
      users
```

## 6.12 Managing Security

Nexus has role-based access control (RBAC) that gives administrators very fine-grained control over who can read from a repository (or a subset of repositories), who can administer the server, and who can deploy to repositories. The security model in Nexus is also so flexible as to allow you to specify that only certain users or roles can deploy and manage artifacts in a specific repository under a specific groupId or asset class. The default configuration of Nexus ships with four roles and four users with a standard set of permissions that will make sense for most users. As your security requirements evolve, you'll likely need to customize security settings to create protected repositories for multiple departments or development groups. Nexus provides a security model which can adapt to any scenario. The security configuration is done via menu items in the *Security* submenu in the left-hand Nexus menu.

Nexus' RBAC system is designed around the following four security concepts:

### Privileges

Privileges are rights to read, update, create, or manage resources and perform operations. Nexus ships with a set of core privileges that cannot be modified, and you can create new privileges to allow for fine-grained targeting of role and user permissions for specific repositories.

### Targets

Privileges are usually associated with resources or targets. In the case of Nexus, a target can be a specific repository or a set of repositories grouped in something called a repository target. A target can also be a subset of a repository or a specific set of assets within a repository e.g. all javadoc archives only. Using a target you can for example also apply a specific privilege to a single groupId and all components using it.

### Roles

Collections of privileges can be grouped into roles to make it easier to define collections of privileges common to certain classes of users. For example, deployment users will all have similar sets of permissions. Instead of assigning individual privileges to individual users, you use roles to make it easier to manage users with similar sets of privileges. A role has one or more privileges and/or one or more roles.

### Users

Users can be assigned roles and will model the individuals who will be logging into Nexus and reading, deploying, or managing repositories.

## 6.13 Managing Privileges

You can access the configuration of privileges via the *Privileges* menu item in the *Security* submenu in the left-hand Nexus menu.

Nexus has three types of privileges:

- application privileges - covers actions a user can execute in Nexus,
- repository target privileges - governs the level of access a user has to a particular repository or repository target, and
- repository view privileges - controls whether a user can view a repository

Behind the scenes, a privilege is related to a single REST operation and method like create, update, delete, read.

---

The screenshot shows the 'Privileges' section of the Nexus web interface. At the top, there's a search bar and a refresh button. Below is a table with columns: Name, User ..., Type, Target, Repository, and Method. The table contains several rows:

Name	User ...	Type	Target	Repository	Method
3rd party - (view)	false	Repository View		3rd party	
Administrator privilege (ALL)	false	Application		*	
All M1 Repositories - (create)	false	Repository Target	All (Maven1)	All Repositories	create,read
All M1 Repositories - (delete)	false	Repository Target	All (Maven1)	All Repositories	delete,read
All M1 Repositories - (read)	false	Repository Target	All (Maven1)	All Repositories	read

Below the table, a specific privilege is selected: 'Administrator privilege (ALL)'. A detailed configuration form is displayed with the following fields:

Name	Administrator privilege (ALL)
Description	Give permission to everything available in nexus.
Type	Application
Method	*
Permission	nexus:*

Figure 6.26: Managing Security Privileges

To create a new privilege, click on the *Add...* button in the *Privileges* panel and choose *Repository Target Privilege*. Creating a privilege will load the New Repository Target Privilege form shown in Figure 6.27. This form takes a privilege name, a privilege description, the repository to target, and a repository target.

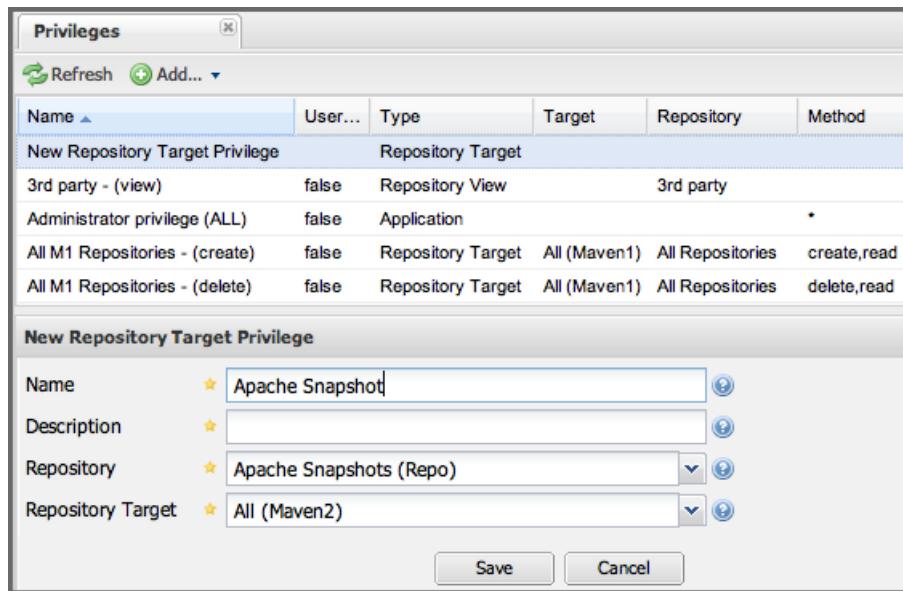


Figure 6.27: Creating a New Repository Target Privilege

Once you create a new privilege, it will create four underlying privileges: create, delete, read, and update. The four privileges created by the form in Figure 6.27 are shown in Figure 6.28.

The screenshot shows the 'Privileges' management screen in the Nexus interface. At the top, there's a toolbar with a refresh button, an 'Add...' button, and a dropdown menu. Below the toolbar is a table with the following columns: Name, User..., Type, Target, Repository, and Method. The table contains four rows of privilege entries:

Name	User...	Type	Target	Repository	Method
Apache Snapshot - (create)	true	Repository Target	All (Maven2)	Apache Snaps...	create,read
Apache Snapshot - (delete)	true	Repository Target	All (Maven2)	Apache Snaps...	delete,read
Apache Snapshot - (read)	true	Repository Target	All (Maven2)	Apache Snaps...	read

Below the table is a section titled 'API-Key Access' with the following fields:

Name	API-Key Access
Description	Give permission to use an API-Key to access the server
Type	Application
Method	*
Permission	apikey:access

Figure 6.28: Create, Delete, Read, and Update Privileges Created

## 6.14 Managing Repository Targets

A *Repository Target* is a set of regular expressions to match on the path of artifacts in a repository (in the same way as the routing rules work). Nexus is preconfigured with a number of repository targets and allows you to create additional ones. Access the management interface visible in Figure 6.29 via the *Repository Targets* menu item in the left-hand *Views.Repositories* sub menu.

Name	Repository Type	Patterns
All (Maven1)	maven1	.*
All (Maven2)	maven2	.*
All (nuget)	nuget	.*
All (obr)	obr	.*
All (p2)	p2	.*
All (site)	site	.*
All but sources (Maven2)	maven2	(?!.*-sources.*).*
All Metadata (Maven2)	maven2	.*maven-metadata.xml.*
nexus-eval-guide	maven2	.*/org/sonatype/nexus/examples/.*

**Repository Target Configuration**

Name: All (Maven2)

Repository Type: Maven2

Pattern Expression:

Patterns:

- .\*

Buttons: Add, Remove, Remove All, Save, Cancel

Figure 6.29: Managing Repository Targets

Repository targets allow you to define, for example, a target called Apache Maven with a pattern of `^/org/apache/maven/.*`. This would match all artifacts with a groupId of `org.apache.maven` and any artifacts within nested groupIds like `org.apache.maven.plugins`.

A pattern that would capture more artifacts like all artifacts with any part of the path containing `maven` could be `.*maven.*`.

The regular expressions can also be used to exclude artifacts as visible with the pattern `(?!.*-sources.*).*` in Figure 6.30 where artifacts with the qualifier `-sources` are excluded. The syntax used for the expressions is the **Java syntax**, that is similar but not identical to the Perl syntax.

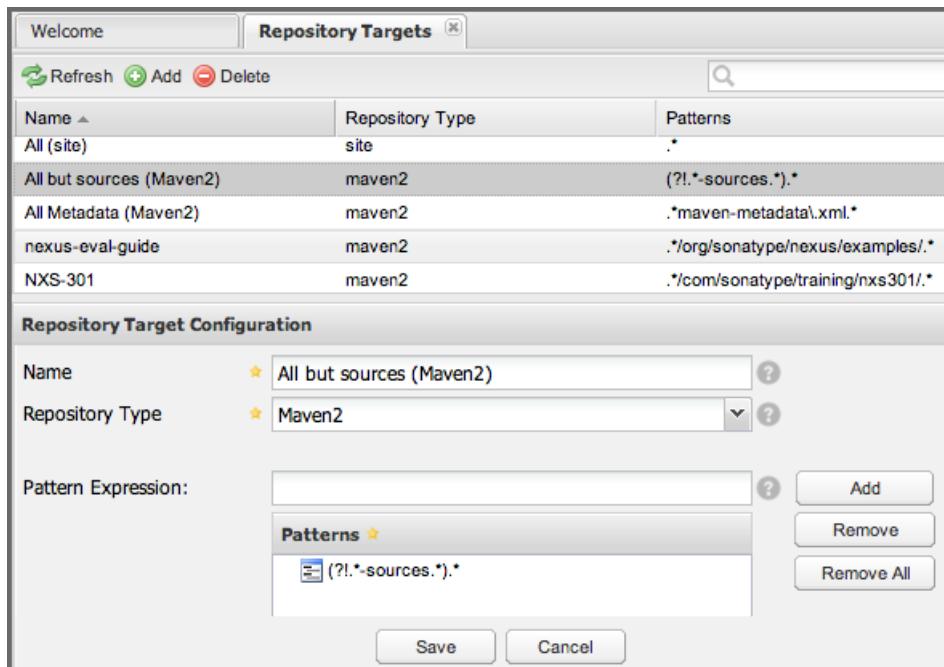


Figure 6.30: Excluding Source Artifacts from a Repository Targets

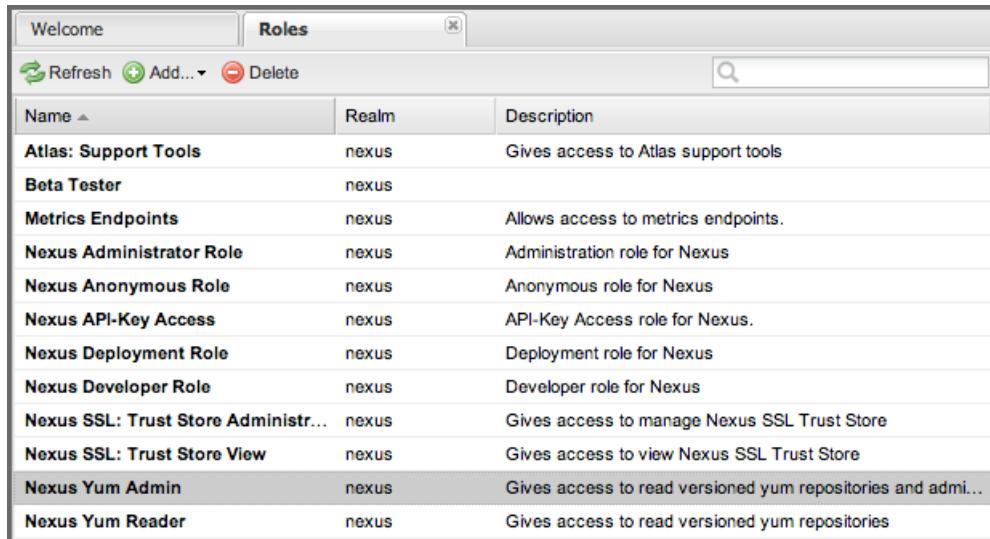
By combining multiple patterns in a repository target, you can establish a fine-grained control of artifacts included and excluded.

Once you have created a repository target, you can utilize it as part of your security setup. You can add a new privilege that relates to the target and controls the CRUD (Create, Read, Update and Delete) operations for artifacts matching that path. The privilege can even span multiple repositories. With this setup you can delegate all control of artifacts in *org.apache.maven* to a "Maven" team. In this way, you don't need to create separate repositories for each logical division of your artifacts.

Repository targets are also used for matching artifacts for implicit capture in the Staging Suite as documented in Chapter 11.

## 6.15 Managing Roles

Nexus ships with a large number of roles predefined including *Nexus Administrator Role*, *Nexus Anonymous Role*, *Nexus Developer Role*, and *Nexus Deployment Role*. Click on the *Roles* menu item under *Security* in the *Nexus* menu to show the list of roles shown in Figure 6.31.



The screenshot shows a table titled "Roles" with columns "Name", "Realm", and "Description". The table lists various predefined roles:

Name	Realm	Description
Atlas: Support Tools	nexus	Gives access to Atlas support tools
Beta Tester	nexus	
Metrics Endpoints	nexus	Allows access to metrics endpoints.
Nexus Administrator Role	nexus	Administration role for Nexus
Nexus Anonymous Role	nexus	Anonymous role for Nexus
Nexus API-Key Access	nexus	API-Key Access role for Nexus.
Nexus Deployment Role	nexus	Deployment role for Nexus
Nexus Developer Role	nexus	Developer role for Nexus
Nexus SSL: Trust Store Administr...	nexus	Gives access to manage Nexus SSL Trust Store
Nexus SSL: Trust Store View	nexus	Gives access to view Nexus SSL Trust Store
Nexus Yum Admin	nexus	Gives access to read versioned yum repositories and adm...
Nexus Yum Reader	nexus	Gives access to read versioned yum repositories

Figure 6.31: Viewing the List of Defined Roles

To create a new role, click on the *Add...* button, select *Nexus Role* and fill out the New Nexus Role form shown in Figure 6.32.

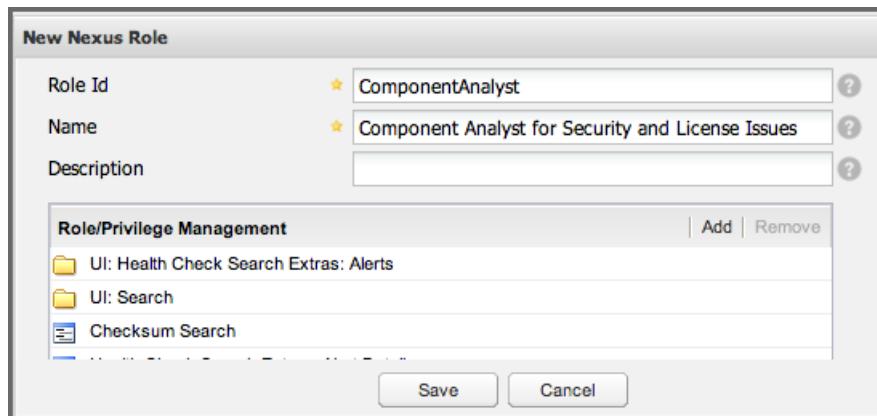


Figure 6.32: Creating a New Nexus Role

When creating a new role, you will need to supply a *Role ID*, a *Name* and a *Description*. Roles are comprised of other roles and individual privileges. To assign a role or privilege to a role, click on *Add* button under *Role/Privilege Management* to access the *Add Roles and Privileges* dialog displayed in Figure 6.33. It allows you to filter the paged displayed of all the available roles and privileges with a filter text as well as narrowing the search to roles or privileges only. Using the filter and the paging you will be able to find the desired role or privilege quickly.

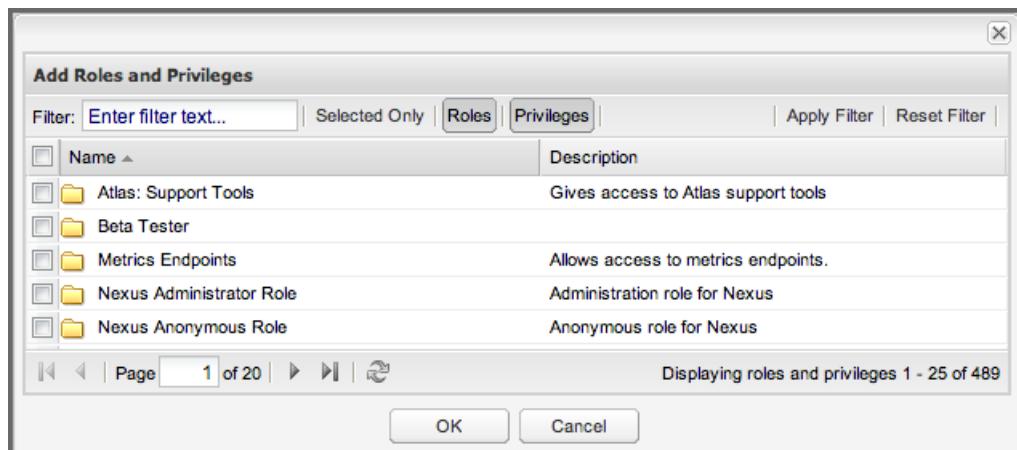


Figure 6.33: The Dialog to Add Roles and Privileges

The built-in roles are managed by Nexus and cannot be edited or deleted. The role configuration section below the list is visible but disabled for these roles.

A Nexus role is comprised of other Nexus roles and individual Nexus privileges. To view the component parts of a Nexus Role, select the role in the Roles list and then choose the *Role Tree* tab as shown in Figure 6.34.

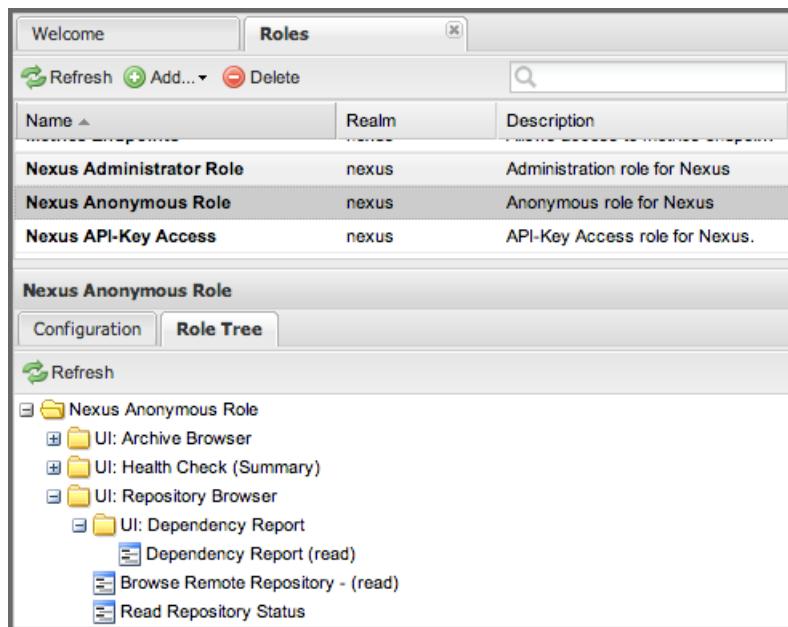


Figure 6.34: Viewing a Role Tree

---

**Tip**

With the Repository Targets, you have fine-grained control over every action in the system. For example, you could make a target that includes everything except sources (`. * (? !-sources) \ . *`) and assign that to one role while giving yet another role access to everything. Using these different access roles e.g., you can host your public and private artifacts in a single repository without giving up control of your private artifacts.

---

## 6.16 Managing Users

Nexus ships with three users: *admin*, *anonymous*, and *deployment*. The admin user has all privileges, the anonymous user has read-only privileges, and the deployment user can both read and deploy to repositories. If you need to create users with a more focused set of permissions, you can click on *Users* under *Security* in the left-hand *Nexus* menu. Once you see the list of users, you can click on a user to edit that specific user's *First Name*, *Last Name* and *Email*. Editing a users *Status* allows you to activate or disable a user altogether. You can also assign or revoke specific roles for a particular user.

The screenshot shows the Nexus User Management interface. At the top, there are tabs for 'Welcome', 'Roles', and 'Users'. The 'Users' tab is selected. Below the tabs is a search bar with placeholder text 'Select a filter to enable username search'. Underneath is a table listing four users:

User ID	Realm	First Name	Last Name	Email	Status	Roles
deployment	default	Deployment	User	changeme1@yourcompany.com	Active	Repo: All Repositories (Full C...)
admin	default	GENERIC	ADMIN	nexus-admins@example.com	Active	Repo: All Repositories (Full C...)
manfred	default	Manfred	Moser	manfred@sonatype.com	Active	Repo: All Repositories (Full C...)
anonymous	default	Nexus	Anonymous User	changeme2@yourcompany.com	Active	Nexus Anonymous Role, Rep...

Below the table, the 'admin' user is selected. The 'Config' tab is active, showing the following configuration details:

User ID	admin
First Name	GENERIC
Last Name	ADMIN
Email	nexus-admins@example.com
Status	Active

Under the configuration section is a 'Role Management' panel. It contains a list of assigned roles: 'Nexus Administrator Role' and 'Repo: All Repositories (Full Control)'. There are 'Add' and 'Remove' buttons next to the list. At the bottom of the panel are 'Save' and 'Reset' buttons.

Figure 6.35: Managing Users

Clicking the *Add* button in the *Role Management* section will bring up the list of available roles in a pop-up window visible in Figure 6.36. It allows you filter and search for roles and add one or multiple roles to the user.

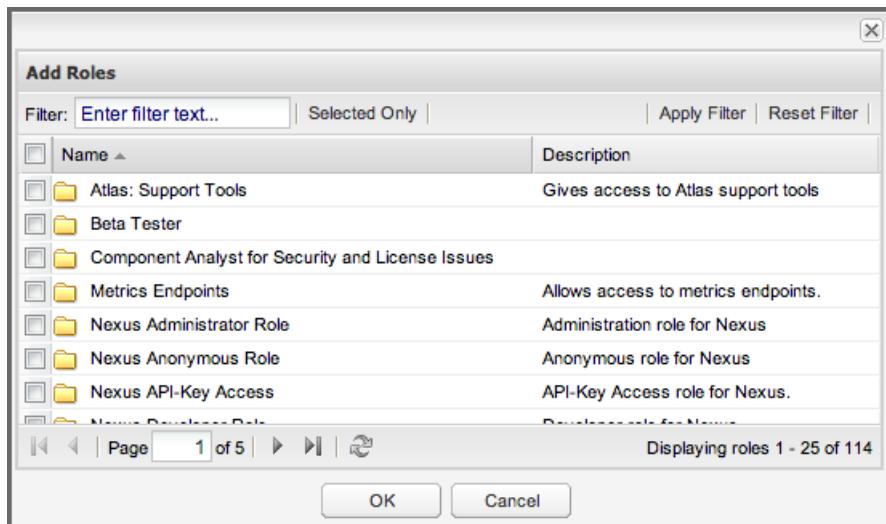


Figure 6.36: Adding Roles to a User

A user can be assigned one or more roles that in turn can include references to other Nexus roles or to individual Nexus privileges. To view a tree of assigned Nexus roles and privileges, select the *Role Tree* for a particular user as shown in Figure 6.37.

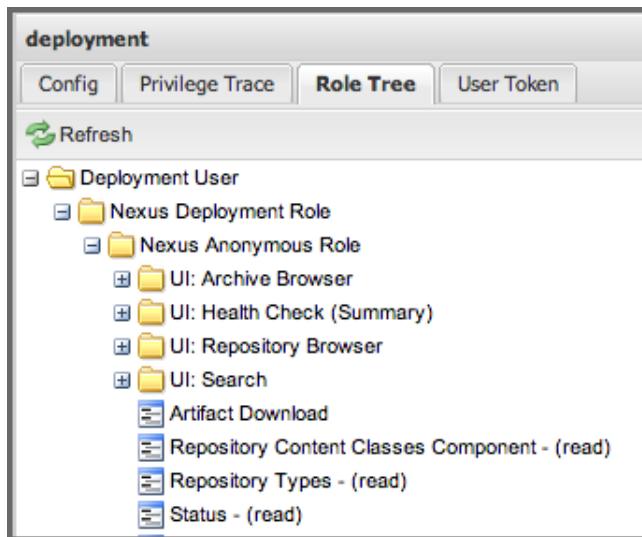


Figure 6.37: Nexus User Role Tree

If you need to find out exactly how a particular user has been granted a particular privilege, you can use the *Privilege Trace* panel as shown in Figure 6.38. The *Privilege Trace* panel lists all of the privileges that have been granted to a particular user in the *Privileges* section. Clicking on a privilege loads a tree of roles that grant that particular privilege to a user. If a user has been assigned a specific privilege by more than one Role or Privilege assignment, you will be able to see this reflected in the *Role Containment* list.

The screenshot shows the 'deployment' section of the Nexus interface. At the top, there are tabs: 'Config', 'Privilege Trace' (which is selected), 'Role Tree', and 'User Token'. Below the tabs is a 'Refresh' button. A message says: 'Select a privilege to view the role(s) in the user that grant the privilege.' On the left, under 'Privileges', a list includes: 'All Repositories - (delete)', 'All Repositories - (read)', 'All Repositories - (update)', 'All Repositories - (view)', 'Artifact Download', 'Browse Archive Files (read)' (which is selected), and 'Browse Remote Repository - (read)'. On the right, under 'Role Containment', a tree structure shows: 'UI: Archive Browser' containing 'Nexus Anonymous Role' which contains 'Nexus Deployment Role'.

Figure 6.38: Nexus User Privilege Trace

Additional plugins can contribute further panels for the security configuration of a user. An example of an additional panel is the *User Token* panel, added by the User Token feature of Nexus Professional as documented in Section 6.17.

## 6.17 Security Setup with User Tokens

### 6.17.1 Introduction

When using Apache Maven with Nexus, the user credentials for accessing Nexus have to be stored in clear text in the user's settings.xml file. Maven has the ability to encrypt passwords in setting.xml, but the need for it to be reversible in order to be used, limits its security. In addition, the general setup and use is cumbersome, and the potential need for regular changes due to strong security requirements e.g., with regular, required password changes triggers the need for a simpler and more secure solution.

Other build systems use similar approaches and can benefit from the usage of User Token as well.

The User Token feature of Nexus fills that need for Apache Maven as well as other build systems and users. It introduces a two-part token for the user, replacing the username and password with a user code and a pass code that allows no way of recovering the username and password from the user code and pass

code values; yet can be used for authentication with Nexus from the command line via Maven as well as in the UI.

This is especially useful for scenarios where single sign-on solutions like LDAP are used for authentication against Nexus and other systems and the plain text username and password cannot be stored in the `settings.xml` following security policies. In this scenario the generated user tokens can be used instead.

User token usage is integrated in the Maven settings template feature of Nexus documented in Chapter 13 to further simplify its use.

### 6.17.2 Enabling and Resetting User Tokens

The user token-based authentication can be activated by a Nexus administrator or user with the role `usertoken-admin` or `usertoken-all` by accessing the *User Token* item in the *Security* submenu on the left-hand *Nexus* menu.

Once user token is *Enabled* by activating the checkbox in the administration tab displayed in Figure 6.39 and pressing *Save*, the feature is activated and the additional section to Reset All User Tokens is available as well.

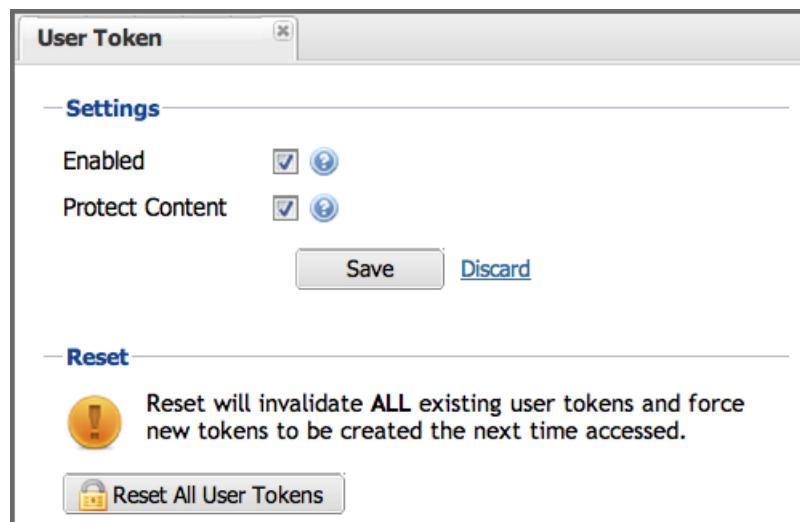


Figure 6.39: User Token Administration Tab Panel

Selecting the *Protect Content* feature configures Nexus to require a user token for any access to the content urls of Nexus that includes all repositories and groups. This affects read access as well as write access e.g., for deployments from a build execution or a manual upload.

Activating *User Token* as a feature automatically adds the *User Token Realm* as a *Selected Realm* in the *Security Settings* section as displayed in Figure 6.40 and available in the *Server* section of the left-hand *Administration* menu. If desired, you can reorder the security realms used, although the default settings with the *User Token Realm* as a first realm is probably the desired setup. This realm is not removed when the *User Token* feature is disabled; however, it will cleanly pass through to the next realm and with the realm remaining any order changes stay persisted in case the feature is reactivated at a later stage.

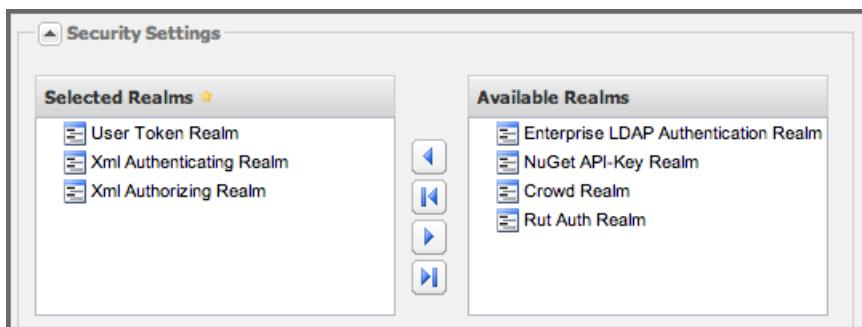


Figure 6.40: Selected Realms Server Security Settings with User Token Realm activated

Besides resetting all user tokens, an administrator can reset the token of an individual user by selecting the *User Token* tab in the *Users* administration from the *Security* menu in the left-hand navigation displayed in Figure 6.41. The password requested for this action to proceed is the password for the currently logged in administrator resetting the token(s).

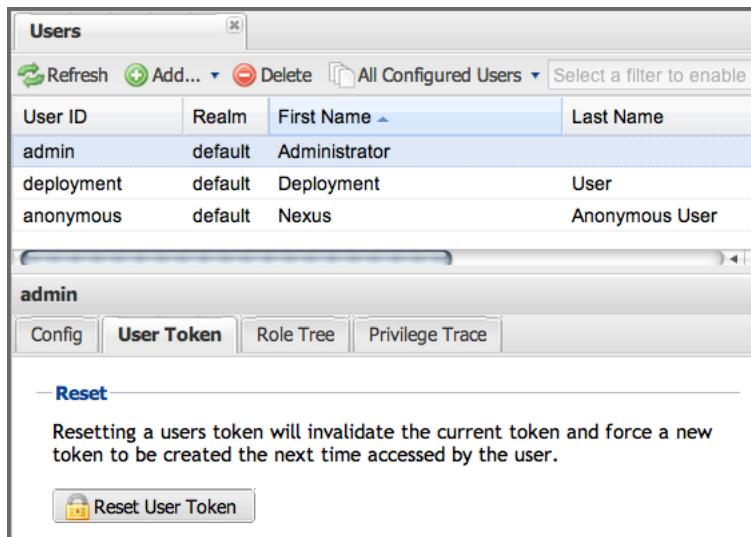


Figure 6.41: User Token Reset for Specific User in Security Users Administration

**Warning**

Resetting user tokens forces the users to update the `settings.xml` with the newly created tokens and potentially breaks any command line builds using the tokens until this change is carried out. This specifically also applies to continuous integration servers using user tokens or any other automated build executions.

### 6.17.3 Accessing and Using Your User Tokens

With user token enabled, any user can access his/her individual tokens via their *Profile* panel. To access the panel, select *Profile* when clicking on the user name in the top right-hand corner of the Nexus user interface. Then select *User Token* in the drop-down to get access to the *User Token* screen in the *Profile panel* displayed in Figure 6.42.

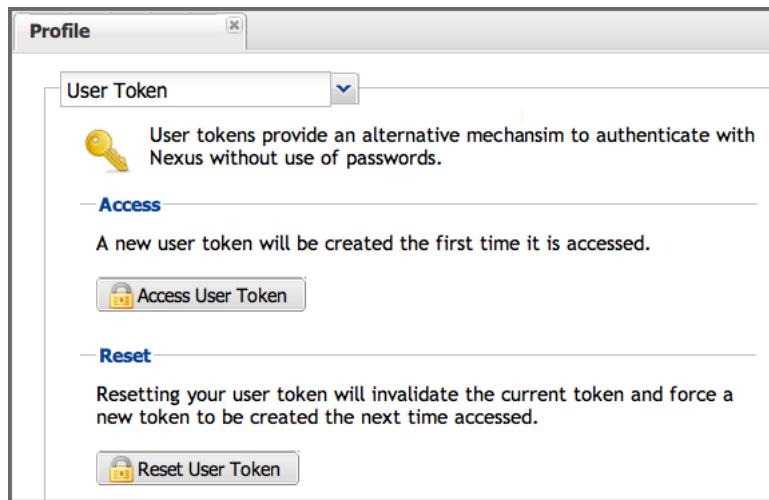


Figure 6.42: User Token Panel for the Logged in Users in the Profile Section

In order to be able to see this *User Token* panel the user has to have the `usertoken-basic` role or the `usertoken-user` privilege. To access or reset the token you have to press the respective button in the panel and then provide your username and password in the dialog.

Resetting the token will show and automatically hide a dialog with a success message and accessing the token will show the dialog displayed in Figure 6.43.

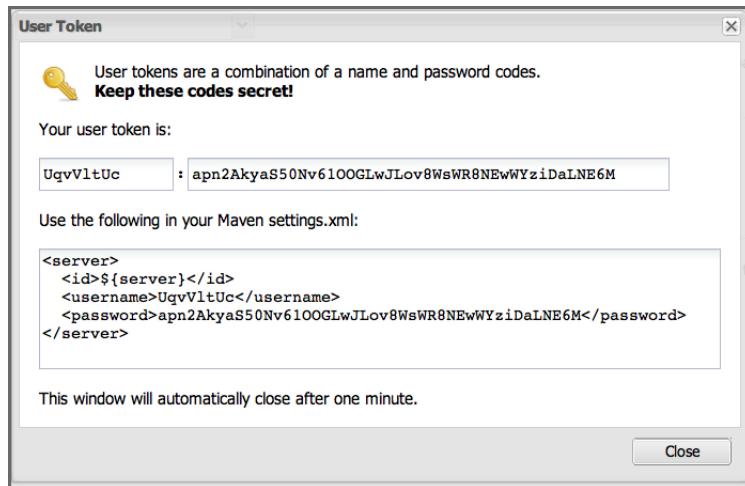


Figure 6.43: Accessing the User Token Information

The User Token dialog displays the user code and pass code tokens in separate fields in the top level section as well as a server section ready to be used in a Maven settings.xml file. When using the server section you simply have to replace the \${server} placeholder with the repository id that references your Nexus server you want to authenticate against with the user token. The dialog will close automatically after one minute or can be closed with the Close button.

The user code and pass code values can be used as replacements for username and password in the login dialog for Nexus. It is also possible to use the original username and the pass code to log in to Nexus.

With content protection enabled, command line access to Nexus will require the tokens to be supplied. Access to e.g., the releases repository via

```
curl -v --user admin:admin http://localhost:9081/content/repositories/ →
    releases/
```

has to be replaced with the usage of user code and pass code separated by colon in the curl command line like this

```
curl -v --user HdeHuL4x:Y7ZH6ixZFdOVwNpRhaOV+phBISmipsfwVxPRUH1gkV09 http →
    ://localhost:9081/content/repositories/releases/
```

User token values can be accessed as part of the Maven settings template feature automating updates as documented in Chapter 13.

**Note**

The user tokens are created at first access whether that is by using the Nexus user interface or the Nexus Maven Plugin.

---

#### 6.17.4 Configuring User Token behavior

The user token feature is preconfigured with built-in parameters and no external configuration file is created by default. It is however possible to customize some behavior by creating a file *sonatype-work/nexus/conf/usertoken.properties*'.

The following properties can be configured:

**usertoken.userTokenServiceImpl.allowLookupByUserName**

This parameter controls if username lookup is allowed when using a pass code. The default is set to true. If set to false, user code and pass code have to be used to authenticate, otherwise username and pass code is also possible. This would be the more secure setting.

**usertoken.userTokenServiceImpl.restrictByUserAgent**

With this value set to true (the default), any access to the Nexus content with content protection enabled will only be allowed to browser-based access even without credentials. Other tools like curl or wget or other command-line tools will be blocked. With the more secure setting of false, any access without correct codes will be disallowed.

The *usertoken.* prefix is optional when the properties are loaded from the *usertoken.properties* file.

## 6.18 Authentication via Remote User Token

Nexus allows integration with external security systems that can pass along authentication of a user via the `Remote_User` HTTP header field - Remote User Token *Rut* authentication. There are either web-based container or server-level authentication systems like [Shibboleth](#). In many cases, this is achieved via a server like [Apache HTTPD](#) or [nginx](#) proxying Nexus. These servers can in turn defer to other authentication storage systems e.g., via the [Kerberos](#) network authentication protocol. These systems and setups can be described as Central Authentication Systems CAS or Single Sign On SSO.

From the users perspective, he/she is required to login into the environment in a central login page that

---

then propagates the login status via HTTP headers. Nexus simply receives the fact that a specific user is logged in by receiving the username in a HTTP header field.

The HTTP header integration can be activated by adding and enabling the *Rut Auth* capability as documented in Section 6.6 and setting the *HTTP Header name* to the header populated by your security system. Typically, this value is REMOTE\_USER, but any arbitrary value can be set. An enabled capability automatically causes the *Rut Auth Realm* to be added to the *Selected Realms* in the *Security Settings* described in Section 6.1.3.

When an external system passes a value through the header, authentication will be granted and the value will be used as the user name for configured authorization scheme. For example, on a default Nexus installation with the Xml authorization scheme enabled, a value of *deployment* would grant the user the access rights in the user interface as the *deployment* user.

A seamless integration can be set up for users if the external security system is exposed via LDAP and configured in Nexus as LDAP authorization realm combined with external role mappings and in parallel the sign-on is integrated with the operating system sign-on for the user.

# Chapter 7

## Nexus Smart Proxy

### 7.1 Introduction

#### **Default is Polling**

Typically an organization runs a single Nexus instance to proxy external components as well as host internally produced components. When a build is running against this Nexus instance, it will look for any new components in the proxied remote repositories. This adds additional network traffic that in many cases will just be a response from the remote server indicating that there are no changes.

This polling approach is fine for smaller deployments. It will not result in immediately updated components as soon as they become available upstream. In distributed teams with multiple Nexus instances, this delay can result in build failures and delays. The only way you are going to achieve that everything is up to date is by setting your expiration times to zero and constantly polling.

#### **Smart Proxy Introduces Publish-Subscribe**

Increasingly, Nexus is used in globally distributed teams or used by projects that span multiple organizations. In many cases, it is advisable for each physical location to host its own Nexus instance. This local instance hosts its own components and proxies the other servers.

An example deployment scenario is displayed in Figure 7.6. Using the traditional polling approach,

---

specifically when used with snapshot repositories, can result in significant traffic and a performance hit for all involved servers.

The *Smart Proxy* feature replaces this constant polling approach with a *Publish/Subscribe*-based messaging approach between Nexus instances sharing a mutual trust. Once a component is published to a repository a message is sent to all subscribing in the smart proxy message queue that details the availability of new component. The subscribers are therefore immediately aware of any new deployment and can provide these components without having to poll the publishing server.

The result is a significantly improved performance due to nearly immediate availability of upstream component information directly in the downstream Nexus instances.

## 7.2 Enabling Smart Proxy Publishing

In order to enable the smart proxy feature on your Nexus instance, you need to navigate to the global *Smart Proxy* configuration screen. It is available in the left-hand navigation in the *Enterprise* section. Selecting *Smart Proxy* will show you the configuration screen displayed in Figure 7.1.

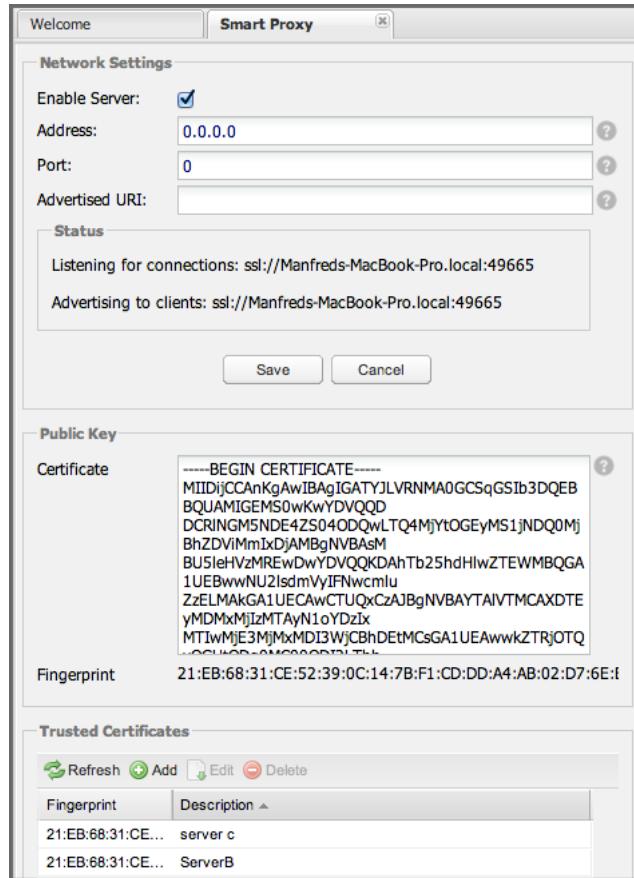


Figure 7.1: Global Configuration for Smart Proxy

The *Network Settings* section allows you to enable the smart proxy server with a checkbox. This will need to be enabled on all servers that publish events in the smart proxy network, while servers that act only as subscribers can leave this option unchecked.

In addition, you can configure the address and port where the publishing server will be available. The default address of 0.0.0.0 will cause the proxy to listen on all addresses. The default port number of 0 will trigger usage of a random available port number for connection listening. If a random port is used, it will be chosen when the server (re)starts.

With the *Advertised URI* field it is possible to configure a specific address to be broadcasted by the proxy to the subscribing smart proxy clients enabling, e.g., usage of a publicly available fully qualified

hostname, including the domain or also just the usage of an externally reachable IP number.

**Important**

It is important to configure the ports in Nexus and any firewall between the servers to allow the direct socket connection between the servers and to avoid using random ports.

---

The *Status* field below the form will show the current status of the smart proxy including the full address and port.

The *Public Key* field displays the key identifying this server instance. It is automatically populated with the certificate associated with the public/private key pair that is generated when the server is first run.

**Tip**

The key is stored in `sonatype-work/nexus/conf/keystore/private.ks` and identifies this server. If you copy the sonatype work folder from one server to another as part of a migration or a move from testing to staging or production you will need to ensure that keys are not identical between multiple servers. To get a new key generated, simply remove the keystore file and restart Nexus.

---

## 7.3 Establishing Trust

The servers publishing as well as subscribing to events identify themselves with their public key. This key has to be registered with the other servers in the *Trusted Certificates* section of the *Smart Proxy* configuration screen.

To configure two Nexus repository servers as trusted smart proxies, you copy the public key from the certificate of the other server in the *Trusted Certificates* configuration section by adding a new trusted certificate with a meaningful description as displayed in Figure 7.2 and Figure 7.3.

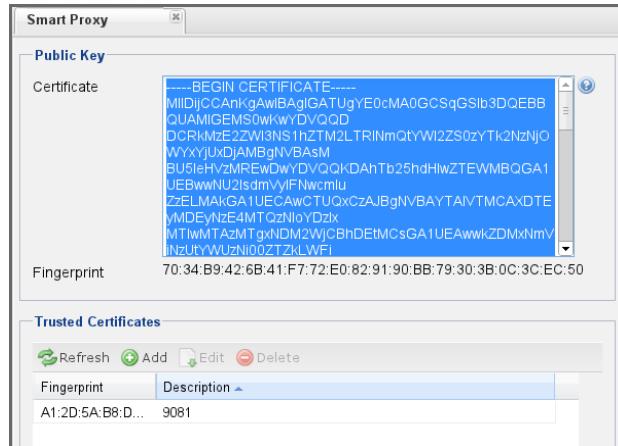


Figure 7.2: Copying a Certificate



Figure 7.3: Adding a Trusted Certificate

All of the key generation and certificates related to the trust management is handled by Nexus, itself and no external configuration or usage of external keys is necessary.

## 7.4 Repository Specific Smart Proxy Configuration

Once Smart Proxy has been configured and enabled as described above, you have to configure which repositories contents should be proxied more efficiently between the servers. This is done in the *Repositories* administration interface in a separate configuration tab titled *Smart Proxy*, which allows you to configure repository-specific details as compared to server wide details described above.

On the publishing Nexus server you have to enable smart proxy on the desired hosted, virtual or proxy repositories in the repository configuration. This is accomplished by selecting the *Publish Updates* checkbox in the *Publish* section of the *Smart Proxy* configuration for a specific repository as displayed in Figure 7.4 and pressing save.

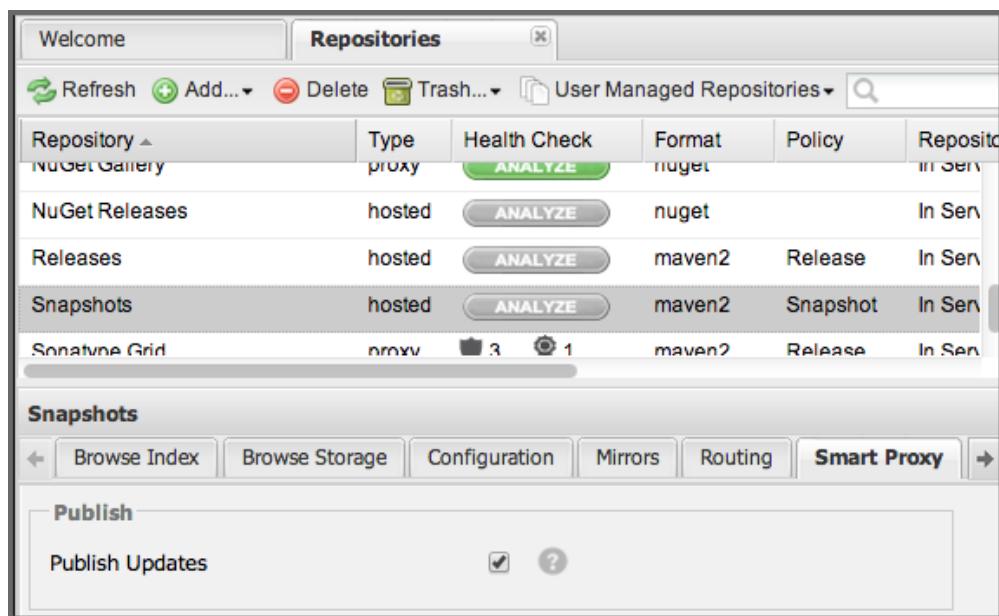


Figure 7.4: Smart Proxy Settings for a Hosted Repository

On the Nexus instance subscribing to the publishing server you have to create a new proxy repository to expose the proxied components. The smart proxy configuration for this repository displayed in Figure 7.5 allows you to activate the *Receive Updates* checkbox in the *Subscribe* configuration section. With a working trust established between the publishing and subscribing Nexus servers the Smart Proxy configuration of the proxy repository on the subscribing Nexus will display connection status.

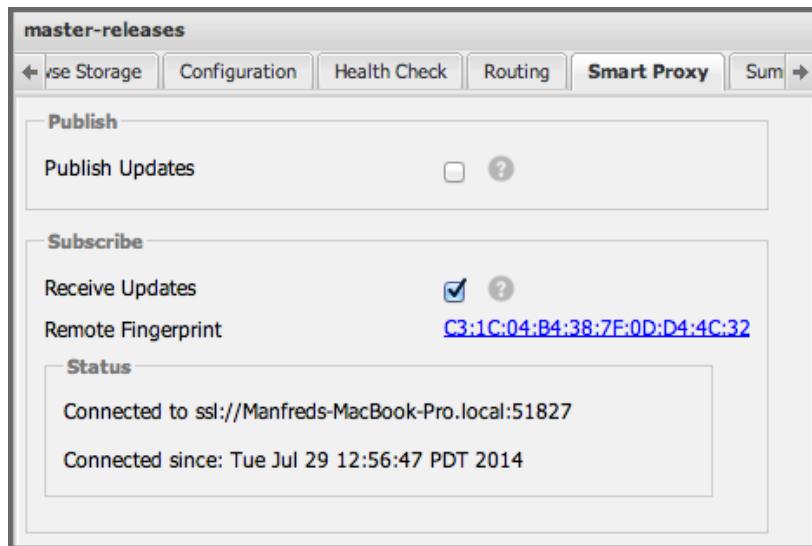


Figure 7.5: Smart Proxy Settings for a Proxy Repository

## 7.5 Smart Proxy Security and Messages

Smart Proxy messages are started with an initial handshake via HTTP. This handshake allows the two server to exchange their keys and confirm that they are configured with a valid trust relationship to communicate. After a successful handshake, messages are sent in the middleware layer and can be configured to be sent via SSL encrypted messages.

The following events are broadcasted via Smart Proxy.

- a new component has been deployed
- a component has been deleted
- a component has been changed
- repository cache or a part of it has been cleared
- Smart Proxy publishing has been disabled

On the recipient side this will cause the changes to be applied, mimicking what happened on the publisher. If Smart Proxy is disabled the subscription will be stopped.

## 7.6 Example Setup

The deployment scenario displayed in Figure 7.6 is a typical use case for Smart Proxy. Component development is spread out across four distributed teams located in New York, London, Bangalore and San Jose. Each of the teams has a Nexus instance deployed in their local network to provide the best performance for each developer team and any locally running continuous integration server and other integrations

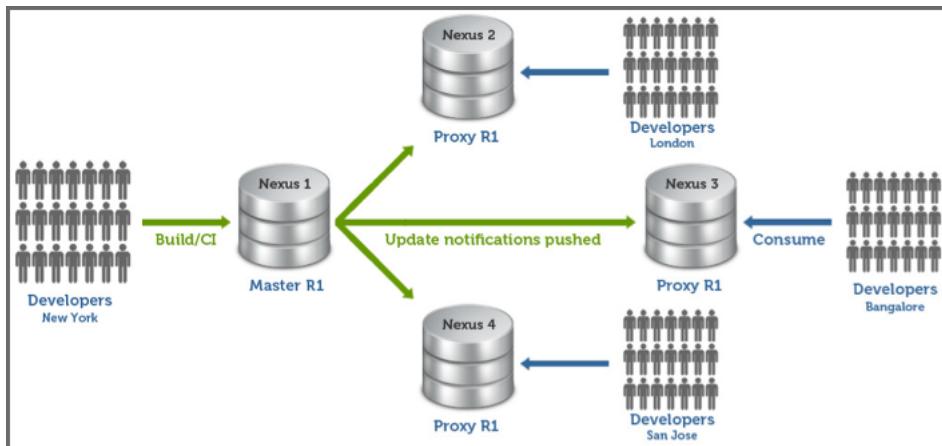


Figure 7.6: Deployment Scenario for a Smart Proxy Use Case

When the development team in New York does a commit to their component build, a continuous integration server deploys a new component snapshot version to the Nexus 1 instance.

With smart proxy enabled, this deployment is immediately followed by notifications, sent to the trusted smart proxy subscribers in Nexus 2, Nexus 3 and Nexus 4. These are collocated with the developers in London, Bangalore, and San Jose and can be configured to immediately fetch the new components available. At a minimum they will know about the availability of new component versions without the need to poll Nexus 1 repeatedly, therefore, keeping performance high for everyone.

When a user of Nexus 2, 3 or 4 build a component that depends on a snapshot version of the component

from Nexus 1, smart proxy guarantees that the latest version published to Nexus 1 is used.

To configure smart proxy between these servers for the snapshots repository you have to

1. add the public key of Nexus 1 as trusted certificate to Nexus 2, 3 and 4
2. add the public keys of Nexus 2, 3 and 4 as trusted certificate to Nexus 1
3. enable smart proxy publishing on the snapshot repository on Nexus 1
4. set up new proxy repositories to proxy the Nexus 1 snapshot repository on Nexus 2, 3 and 4
5. enable smart proxy subscription on the new proxy repositories
6. optionally enable prefetching of components
7. add the new proxy repositories to the public group on Nexus 2, 3 and 4

With this setup, any snapshot deployment from the New York team on Nexus 1 is immediately available to the development team in London, Bangalore, and San Jose.

## 7.7 Advanced Configuration

Typically smart proxy is configured in the dedicated user interfaces provided and described earlier in this chapter. More fine grained and advanced configuration is exposed in the capabilities administration of Nexus documented in Section [6.6](#).

Specifically the following capabilities for the core smart proxy features are automatically created and maintained.

### Smart Proxy: Identity

Provides the unique identity for the Nexus server.

### Smart Proxy: Messaging

Provides the core messaging facilities for smart proxy.

### Smart Proxy: Trust

Configures a trust relationship with a remote node.

### Smart Proxy: Secure Connector

Secures the connection using identity and trust.

---

In addition you can find one smart proxy capabilities for all repositories configured to be publish or subscribe updates with smart proxy.

### **Smart Proxy: Publish**

Configures publishing updates to a specific repository via smart proxy.

### **Smart Proxy: Subscribe**

Configures subscribing to updates for a specific proxy repository. This capability exposes the additional setting *Delete* in the *Settings* tab. If deletion is enabled, any component deletions in the publishing repository is also carried out in the subscribing repositories. The *Preemptive Fetch* flag allows you to enable a download of components to the susbscribing proxy repository prior to any component requests received by it. The default behaviour with preemptive fetch disabled only publishes the fact that new components are available from the publishing repository.

## Chapter 8

# Nexus LDAP Integration

### 8.1 Introduction

Nexus Open Source has a Lightweight Directory Access Protocol (LDAP) Authentication realm which provides Nexus with the capability to authenticate users against an LDAP server. In addition to handling authentication, Nexus can be configured to map Nexus roles to LDAP user groups. If a user is a member of a group that matches the ID of a Nexus role, Nexus will grant that user the matching Nexus role. In addition to this highly configurable user and group mapping capability, Nexus can augment LDAP group membership with Nexus-specific user-role mapping.

In addition to the basic LDAP support from Nexus Open Source, Nexus Professional offers LDAP support features for enterprise LDAP deployments. These include the ability to cache authentication information, support for multiple LDAP servers and backup mirrors, the ability to test user logins, support for common user/group mapping templates, and the ability to support more than one schema across multiple servers.

### 8.2 Enabling the LDAP Authentication Realm

In order to use LDAP authentication in Nexus, you will need to add the Nexus LDAP Authentication Realm to the Selected Realms in the Security section of the Server configuration panel. To load the *Server* configuration panel, click on the *Server* link under *Administration* in the *Nexus* menu. Once you

---

have the *Server* configuration panel loaded, select *Enterprise LDAP Authentication Realm* (or *OSS LDAP Authentication Realm*) in the *Available Realms* list under the *Security Settings* section and click the *Add* button (or *Left Arrow*) as shown in Figure 8.1 and ensure that the LDAP realm is located below the XML realms in the list.

This is necessary, so that Nexus can be used by *anonymous*, *admin* and other users configured in the XML realms even with LDAP authentication offline or unavailable. Any user account not found in the XML realms, will be passed through to LDAP authentication.

Next, click on the *Save* button at the bottom of the Server configuration panel to have the change applied.

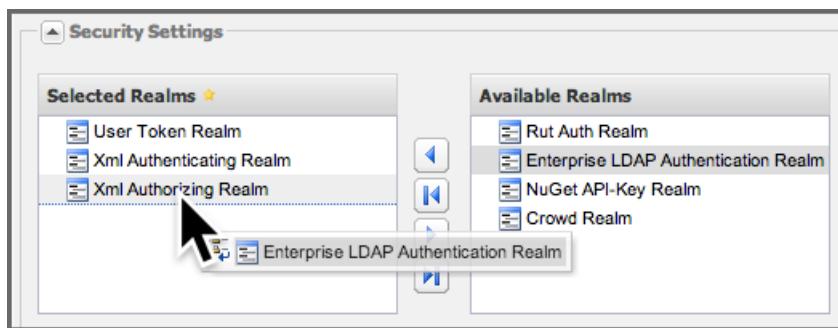


Figure 8.1: Adding the LDAP Authentication Realm to Available Realms

## 8.3 Configuring Nexus LDAP Integration

To configure LDAP integration, click on the *Enterprise LDAP* menu item in Nexus Professional or the *LDAP Configuration* menu item in Nexus Open Source in the *Security* menu in the left-hand Nexus menu.

Clicking on the Enterprise LDAP/LDAP Configuration menu item will load the LDAP Configuration panel. The following sections outline the configuration options available in the LDAP Configuration Panel.

## 8.4 Connection and Authentication

Figure 8.2 shows a simplified LDAP configuration for Nexus configured to connect to an LDAP server running on localhost port 10389 using the search base of `ou=system`. On a more standard installation, you would likely not want to use Simple Authentication as it sends the password in clear text over the network, and you would also use a search base that corresponds to your organization's top-level domain components such as `dc=sonatype, dc=com`.

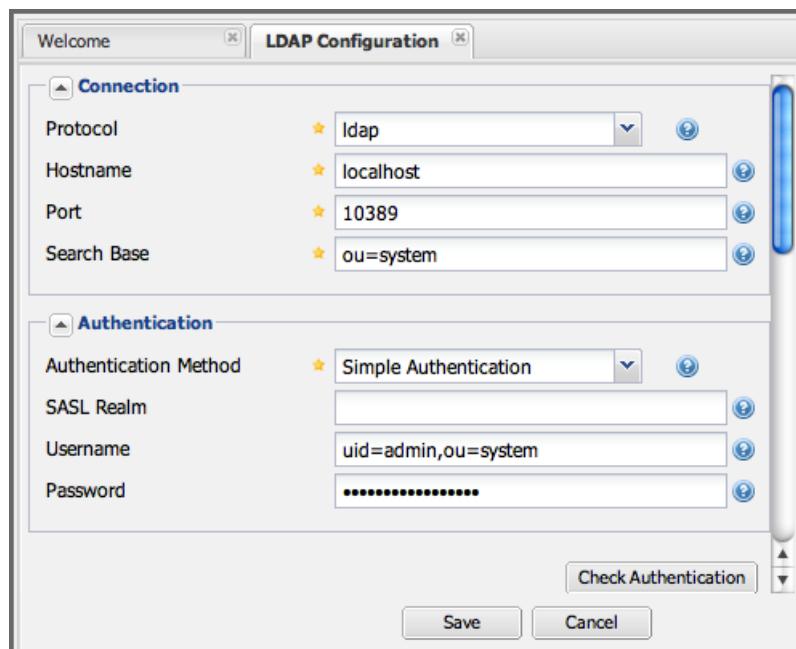


Figure 8.2: A Simple LDAP Connection and Authentication Setup

The following parameters can be configured in the *Connection* and *Authentiation* sections of the *LDAP Configuration* panel.

### Protocol

Valid values in this drop-down are `ldap` and `ldaps` that correspond to the Lightweight Directory Access Protocol and the Lightweight Directory Access Protocol over SSL.

### Hostname

The hostname or IP address of the LDAP.

**Port**

The port on which the LDAP server is listening. Port 389 is the default port for the ldap protocol, and port 636 is the default port for the ldaps.

**Search Base**

The search base is the Distinguished Name (DN) to be appended to the LDAP query. The search base usually corresponds to the domain name of an organization. For example, the search base on the Sonatype LDAP server could be `dc=sonatype, dc=com`.

**Authentication Method**

Nexus provides four distinct authentication methods to be used when connecting to the LDAP Server:

**Simple Authentication**

Simple authentication is not recommended for production deployments not using the secure ldaps protocol as it sends a clear-text password over the network.

**Anonymous Authentication**

Used when Nexus only needs read-only access to non protected entries and attributes when binding to the LDAP.

**Digest-MD5**

This is an improvement on the CRAM-MD5 authentication method. For more information, see <http://www.ietf.org/rfc/rfc2831.txt>.

**CRAM-MD5**

The Challenge-Response Authentication Method (CRAM) is based on the HMAC-MD5 MAC algorithm. In this authentication method, the server sends a challenge string to the client. The client responds with a username followed by a Hex digest that the server compares to an expected value. For more information, see RFC 2195.

For a full discussion of LDAP authentication approaches, see <http://www.ietf.org/rfc/rfc2829.txt> and <http://www.ietf.org/rfc/rfc2251.txt>.

**SASL Realm**

The Simple Authentication and Security Layer (SASL) realm used to connect. It is only available if the authentication method is Digest-MD5 or CRAM-MD5.

**Username**

Username of an LDAP user with which to connect (or bind). This is a *Distinguished Name* of a user who has read access to all users and groups.

**Password**

Password for an administrative LDAP user.

## 8.5 User and Group Mapping

The *LDAP Configuration* panel in Nexus Open Source contains sections to manage *User Element Mapping* and *Group Element Mapping* in the *User and Group Settings* tab. These configuration sections are located in a separate panel called *User and Group Settings* in Nexus Professional. This panel provided a *User & Group Templates* drop-down displayed in Figure 8.3 that will adjust the rest of the user interface based on your template selection.

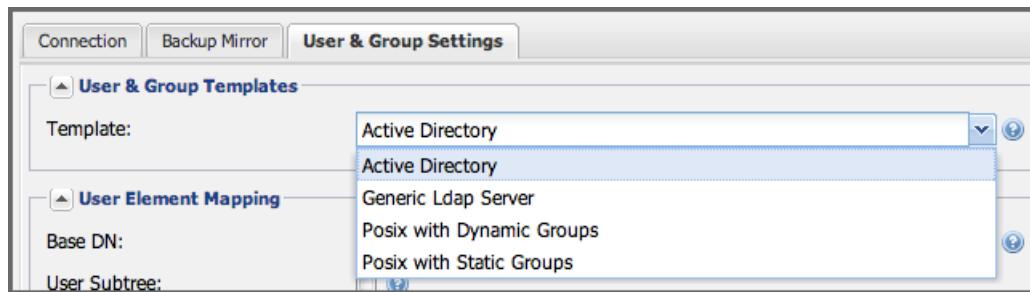


Figure 8.3: User and Group Templates Selection Drop Down

The User Element Mapping displayed in Figure 8.4 has been prepopulated by the Active Directory selection in the template drop-down and needs to be configured as required by your LDAP server. The available fields are:

### Base DN

Corresponds to the *Base DN* containing user entries. This DN is going to be relative to the *Search Base*, specified in Figure 8.2. For example, if your users are all contained in `ou=users,dc=sonatype,dc=com` and you specified a Search Base of `dc=sonatype,dc=com`, you would use a value of `ou=users`.

### User Subtree

Values are *True* if there is a tree below the Base DN that can contain user entries and *False* if all users are contained within the specified Base DN. For example, if all users are in `ou=users,dc=sonatype,dc=com` this field should be *False*. If users can appear in organizational units within organizational units such as `ou=development,ou=users,dc=sonatype,dc=com`, this field should be *True*.

### Object Class

This value defaults to `inetOrgPerson` which is a standard object class defined in [RFC 2798](#). This Object Class (`inetOrgPerson`) contains standard fields such as `mail`, `uid`. Other possible values are `posixAccount` or a custom class.

### User ID Attribute

This is the attribute of the Object class that supplies the User ID. Nexus will use this attribute as the Nexus User ID.

### Real Name Attribute

This is the attribute of the Object class that supplies the real name of the user. Nexus will use this attribute when it needs to display the real name of a user.

### E-Mail Attribute

This is the attribute of the Object class that supplies the email address of the user. Nexus will use this attribute when it needs to send an email to a user.

### Password Attribute

This control is only available in Nexus Open Source and replaced by the *Use Password Attribute* section from [?informalfigure] in Nexus Professional. It can be used to configure the Object class, which supplies the password ("userPassword").

The screenshot shows the 'User Element Mapping' configuration window. It includes fields for 'Base DN' (cn=users), 'Object Class' (user), 'User ID Attribute' (sAMAccountName), 'Real Name Attribute' (cn), and 'E-Mail Attribute' (mail). Each field has a help icon (blue question mark) and a checkbox for configuration.

Figure 8.4: User Element Mapping

Once the checkbox for *Use Password Attribute* has been selected, the interface from [?informalfigure] allows you to configure the optional attribute. When not configured authentication will occur as a bind to the LDAP server. Otherwise this is the attribute of the Object class that supplies the password of the user. Nexus will use this attribute when it is authenticating a user against an LDAP server.

The screenshot shows the 'Use Password Attribute' configuration window, which contains a single field for 'Password Attribute' with a help icon (blue question mark).

The *Group Type* drop-down displayed in Figure 8.5 and Figure 8.6 determines which fields are available in the user interface. Groups are generally one of two types in LDAP systems - static or dynamic. A static group contains a list of users. A dynamic group is a list of groups to which user belongs. In LDAP a static group would be captured in an entry with an Object class *groupOfUniqueNames* that contains one or more *uniqueMember* attributes. In a dynamic group configuration, each user entry in LDAP contains an attribute that lists group membership.

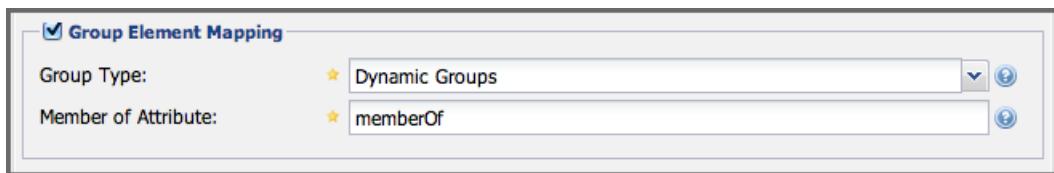


Figure 8.5: Dynamic Group Element Mapping

Dynamic groups are configured via the *Member of Attribute* parameter. Nexus will inspect this attribute of the user entry to get a list of groups of which the user is a member. In this configuration, a user entry would have an attribute that would contain the name of a group, such as *memberOf*.

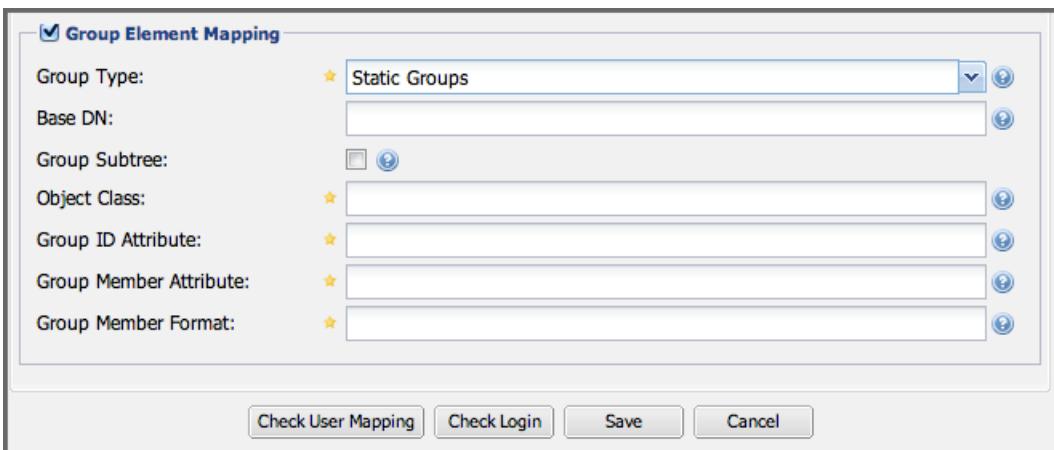


Figure 8.6: Static Group Element Mapping

Static groups are configured with the following parameters:

#### Base DN

This field is similar to the Base DN field described for *User Element Mapping*. If your groups were defined under `ou=groups,dc=sonatype,dc=com`, this field would have a value of `ou=groups`.

### Group Subtree

This field is similar to the *User Subtree* field described for *User Element Mapping*. If all groups are defined under the entry defined in *Base DN*, this field should be false. If a group can be defined in a tree of organizational units under the Base DN, then the field should be *true*.

### Object Class

This value defaults to `groupOfUniqueNames` which is a standard object class defined in [RFC 4519](#). This default (`groupOfUniqueNames`) is simply a collection of references to unique entries in an LDAP directory and can be used to associate user entries with a group. Other possible values are `posixGroup` or a custom class.

### Group ID Attribute

Specifies the attribute of the Object class that specifies the *Group ID*. If the value of this field corresponds to the ID of a Nexus role, members of this group will have the corresponding Nexus privileges. Defaults to `cn`.

### Group Member Attribute

Specifies the attribute of the Object class which specifies a member of a group. A `groupOfUniqueNames` has multiple `uniqueMember` attributes for each member of a group. Defaults to `uniqueMember`.

### Group Member Format

This field captures the format of the *Group Member Attribute*, and is used by Nexus to extract a user-name from this attribute. For example, if the *Group Member Attribute* has the format `uid=brian,ou=users,dc=sonatype,dc=com`, then the *Group Member Format* would be `uid=$username,ou=users,dc=sonatype,dc=com`. If the *Group Member Attribute* had the format `brian`, then the *Group Member Format* would be `$username`.

If your installation does not use Static Groups, you can configure Nexus LDAP Integration to refer to an attribute on the User entry to derive group membership. To do this, select Dynamic Groups in the Group Type field in Group Element Mapping.

Once you have configured the *User & Group Settings* you can check the correctness of your user mapping by pressing the *Check User Mapping* button visible in Figure 8.6.

Nexus Professional offers a button *Check Login* to check an individual users login and can be used as documented in Section 8.11.5.

Press the *Save* button after successful configuration.

---

## 8.6 Mapping Users and Groups with Active Directory

When mapping users and groups to an Active Directory installation, try the common configuration values listed in Table 8.2 and Table 8.3.

Table 8.1: Connection and Authentication Configuration for Active Directory

Configuration Element	Configuration Value
Protocol	ldap
Hostname	Hostname of Active Directory Server
Port	389 (or port of AD server)
Search Base	DC=yourcompany,DC=com (customize for your organization)
Authentication	Simple Authentication
Username	CN=Administrator,CN=Users,DC=yourcompany,DC=com

Table 8.2: User Element Mapping Configuration for Active Directory

Configuration Element	Configuration Value
Base DN	cn=users
User Subtree	false
Object Class	user
User ID Attribute	sAMAccountName
Real Name Attribute	cn
E-Mail Attribute	mail
Password Attribute	(Not Used)

Table 8.3: Group Element Mapping Configuration for Active Directory

Configuration Element	Configuration Value
Group Type	Dynamic Groups
Member Of Attribute	memberOf

**Warning**

 You should connect to the Active Directory through port 3268 if you have a multi domain, distributed Active Directory forest. Connecting directly to port 389 might lead to errors. Port 3268 exposes Global Catalog Server that exposes the distributed data. The SSL equivalent connection port is 3269.

## 8.7 Mapping Users and Groups with posixAccount

When mapping users and groups to LDAP entries of type posixAccount, try the common configuration values listed in Table 8.4 and Table 8.5.

Table 8.4: User Element Mapping Configuration for posixAccount

Configuration Element	Configuration Value
Base DN	(Not Standard)
User Subtree	false
Object Class	posixAccount
User ID Attribute	sAMAccountName
Real Name Attribute	uid
E-Mail Attribute	mail
Password Attribute	(Not Used)

Table 8.5: Group Element Mapping Configuration for posixGroup

Configuration Element	Configuration Value
Group Type	Static Groups
Base DN	(Not Standard)
Group Subtree	false
Object Class	posixGroup
Group ID Attribute	cn
Group Member Attribute	memberUid
Group Member Format	

## 8.8 Mapping Roles to LDAP Users

Once *User and Group Mapping* has been configured, you can start verifying how LDAP users and groups are mapped to Nexus roles. If a user is a member of an LDAP group that has a *Group ID* corresponding to the ID of a Nexus role, that user is granted the appropriate permissions in Nexus. For example, if the LDAP user entry in `uid=brian,ou=users,dc=sonatype,dc=com` is a member of a *groupOfUniqueNames* attribute value of `admin`, when this user logs into Nexus, he/she will be granted the Nexus administrator role if the *Group Element Mapping* is configured properly. To verify the *User Element Mapping* and *Group Element Mapping*, click on *Check User Mapping* in the *LDAP Configuration* panel directly below the *Group Element Mapping* section, Figure 8.7 shows the results of this check.

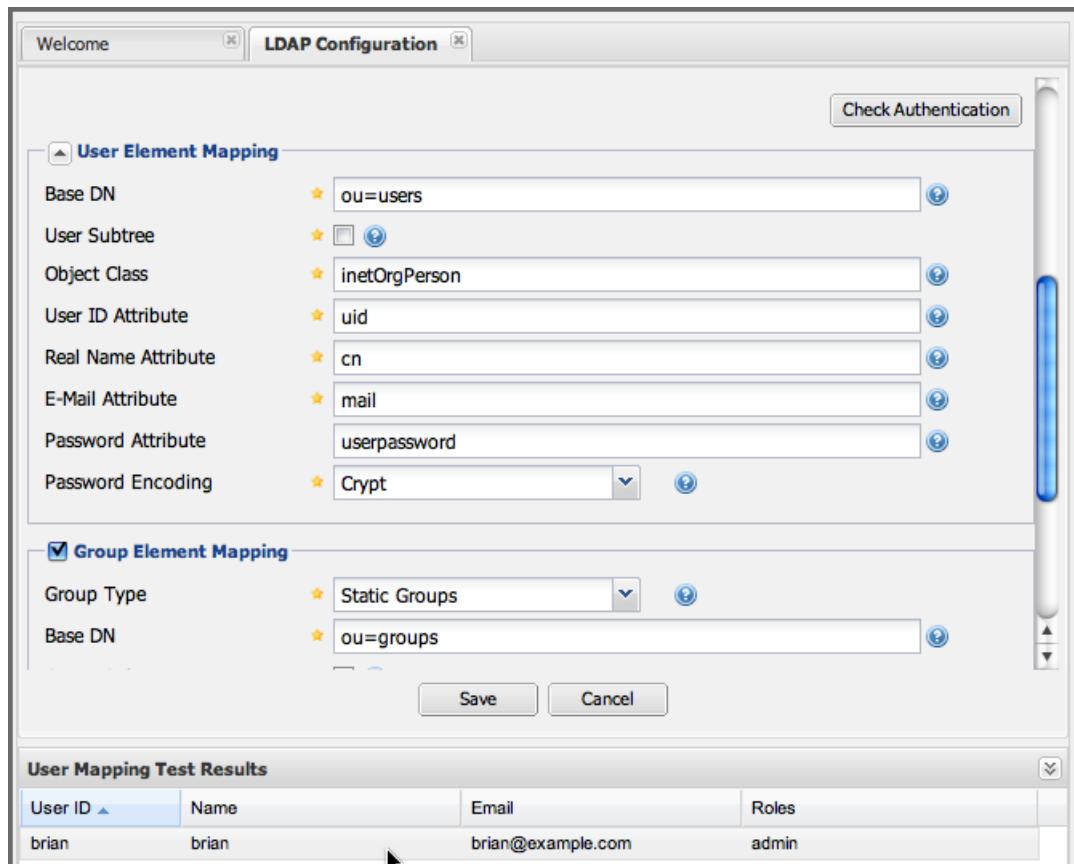


Figure 8.7: Checking the User and Group Mapping in LDAP Configuration

In Figure 8.7, Nexus LDAP Integration locates a user with a User ID of "brian" who is a member of the "admin" group. When brian logs in, he will have all of the rights that the admin Nexus Role has.

## 8.9 Mapping Nexus Roles for External Users

If you are unable to map all of the Nexus roles to LDAP groups, you can always augment the role information by adding a specific user-role mapping for an external LDAP user in Nexus. In other words, if you need to make sure that a specific user in LDAP gets a specific Nexus role and you don't want to model this as a group membership, you can add a role mapping for an external user in Nexus.

Nexus will keep track of this association independent of your LDAP server. Nexus continues to delegate authentication to the LDAP server for this user. Nexus will continue to map the user to Nexus roles based on the group element mapping you have configured, but Nexus will also add any roles specified in the User panel. You are augmenting the role information that Nexus gathers from the group element mapping.

Once the user and group mapping has been configured, click on the *Users* link under *Security* in the Nexus menu. The *Users* tab is going to contain all of the configured users for this Nexus instance as shown in Figure 8.8. A configured user is a user in a Nexus-managed realm or an *External User* that has an explicit mapping to a Nexus role. In Figure 8.8, you can see the three default users in the Nexus-managed default realm plus the *brian* user from LDAP. The *brian* user appears because this user has been mapped to a Nexus role.

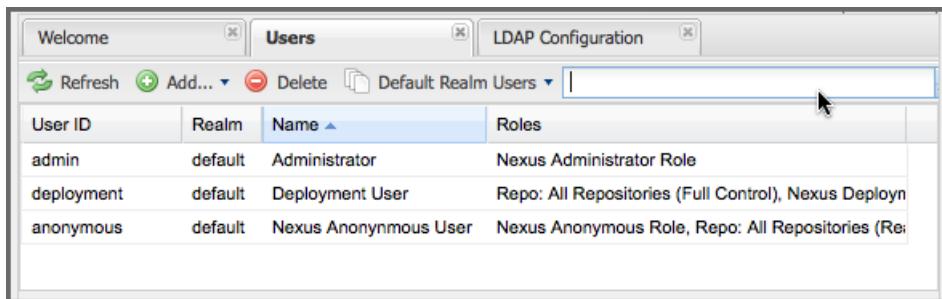
The screenshot shows the 'Users' management interface in Nexus. At the top, there's a toolbar with 'Refresh', 'Add...', 'Delete', and a dropdown for 'All Configured Users'. Below the toolbar is a search bar. The main area is a table with columns: 'User ID', 'Realm', 'Name', 'Email', and 'Roles'. The table contains the following data:

User ID	Realm	Name	Email	Roles
admin	default	Administrator	changeme@yourcompany.com	Nexus Administrator Role
brian	LDAP	Brian Fox		Nexus Deployment Role
deployment	default	Deployment User	changeme1@yourcompany.com	Repo: All Repositories (F)
anonymous	default	Nexus Anonymous User	changeme2@yourcompany.com	Nexus Anonymous Role

Below the table, a 'Deployment User' dialog is open for the 'deployment' user. It includes fields for 'User ID' (deployment), 'Name' (Deployment User), 'Email' (changeme1@yourcompany.com), and 'Status' (Active). There are two panels: 'Selected Roles' containing 'Nexus Deployment Role' and 'Available Roles' containing 'Nexus Administrator Role'. At the bottom are 'Save' and 'Reset' buttons.

Figure 8.8: Viewing All Configured Users

The list of users in Figure 8.8 is a combination of all of the users in the Nexus default realm and all of the *External Users* with role mappings. To explore these two sets of users, click on the *All Configured Users* drop-down and choose *Default Realm Users*. Once you select this, click in the search field and press Enter. Searching with a blank string in the *Users* panel will return all of the users of the selected type. In Figure 8.9 you see a dialog containing all three default users from the Nexus default realm.



User ID	Realm	Name	Roles
admin	default	Administrator	Nexus Administrator Role
deployment	default	Deployment User	Repo: All Repositories (Full Control), Nexus Deploy
anonymous	default	Nexus Anonymous User	Nexus Anonymous Role, Repo: All Repositories (Re

Figure 8.9: All Default Realm Users

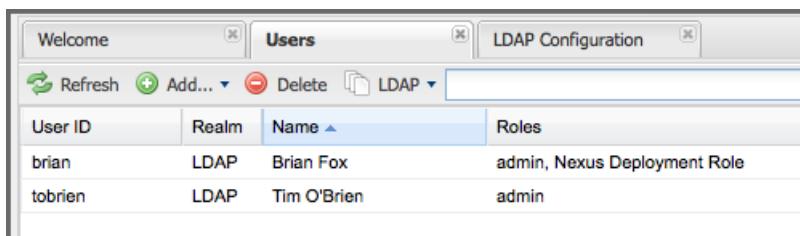
If you wanted to see a list of all LDAP users, select *LDAP* from the *All Configured Users* drop-down shown in Figure 8.8 and click on the search button (magnifying glass) with an empty search field. Clicking search with an empty search field will return all of the LDAP users as shown in Figure 8.10.

---

**Note**

Note that the user `tobrien` does not show up in the *All Configured Users* list. This is by design. Nexus is only going to show you information about users with external role mappings. If an organization has an LDAP directory with thousands of developers, Nexus doesn't need to retain any configuration information for users that don't have custom Nexus role mappings.

---



User ID	Realm	Name	Roles
brian	LDAP	Brian Fox	admin, Nexus Deployment Role
tobrien	LDAP	Tim O'Brien	admin

Figure 8.10: All LDAP Users

To add a mapping for an external LDAP user, you would click on the *All Configured Users* drop-down and select *LDAP*. Once you've selected LDAP, type in the user ID you are searching for and click the search button (magnifying glass icon to right of the search field). In Figure 8.11, a search for "brian" yields one user from the LDAP server.

---

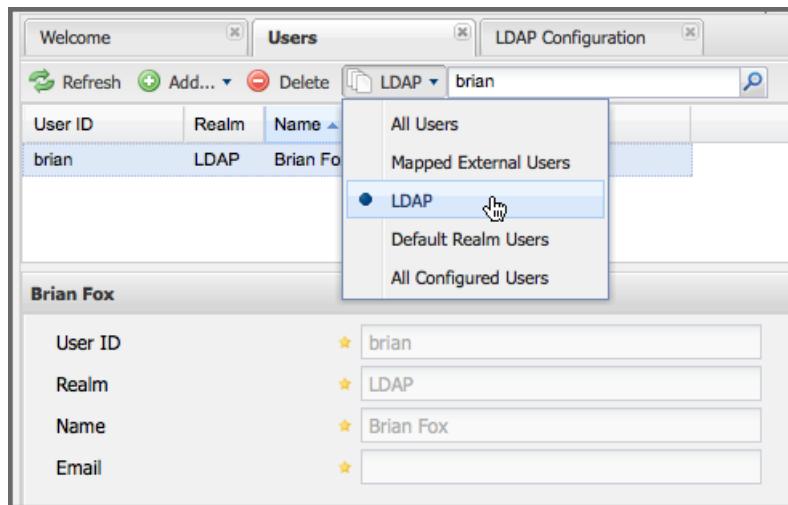


Figure 8.11: Search LDAP Users

To add a Nexus role mapping for the external user `brian` shown in Figure 8.11, click on the user in the results table and drag a role from *Available Roles* to *Selected Roles* as shown in Figure 8.12. In this case, the user "brian" is mapped to the Administrative group by virtue of his membership in an "admin" group in the LDAP server. In this use case, a Nexus administrator would like to grant Brian the Deployment Role without having to create a LDAP group for this role and modifying his group memberships in LDAP

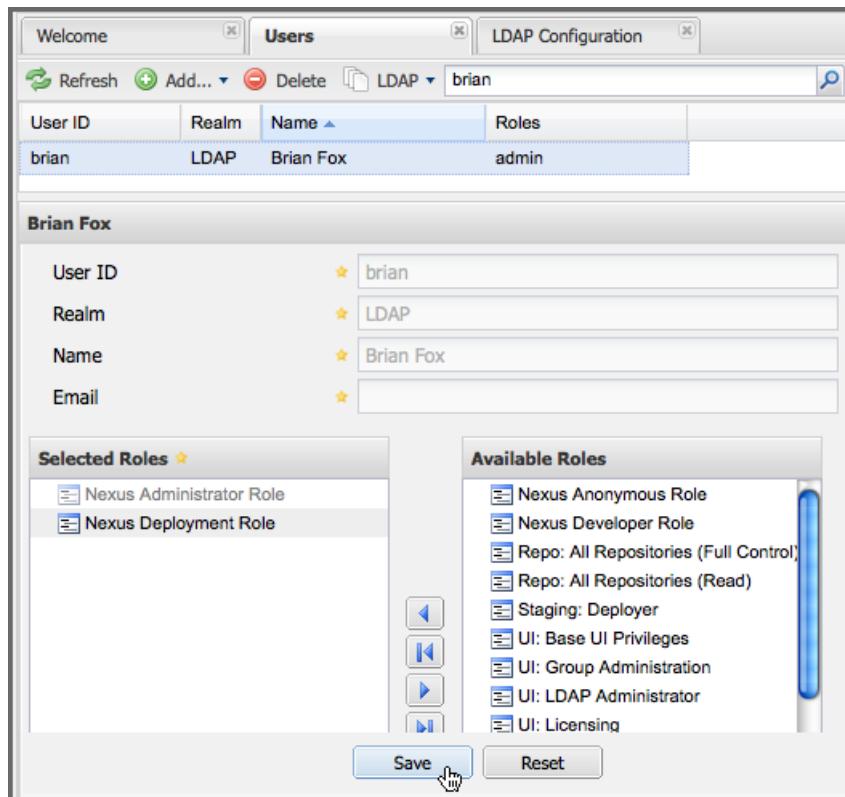


Figure 8.12: Mapping the Deployment Role to an External User

The end result of this operation is to augment the Group-Role mapping that is provided by the LDAP integration. You can use LDAP groups to manage coarse-grained permissions to grant people administrative privileges and developer roles, and if you need to perform more targeted privilege assignments in Nexus you can Map LDAP users to Nexus roles with the techniques shown in this section.

## 8.10 Mapping External Roles to Nexus Roles

Nexus makes it very straightforward to map an external role to an internal Nexus role. This is something you would do, if you want to grant every member of an externally managed group (such as an LDAP group) a certain privilege in Nexus. For example, assume that you have a group in LDAP named `svn` and you want to make sure that everyone in the `svn` group has Nexus administrative privileges. To do this,

you would click on the *Add..* drop-down in the *Roles* panel as shown in Figure 8.13. This drop-down can be found in the roles management panel which is opened by clicking on *Roles* in the *Security* menu.

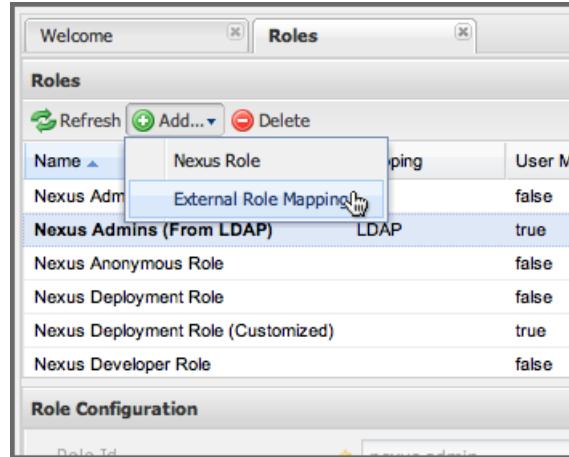


Figure 8.13: Selecting External Role Mapping in the Role Management Panel

Selecting *External Role Mapping* under *Add...* will show you a dialog containing a drop-down of *External Realms*. Selecting an external realm such as LDAP will then bring up a list of roles managed by that external realm. The dialog shown in Figure 8.14 shows the external realm LDAP selected and the role "svn" being selected to map to a Nexus role.

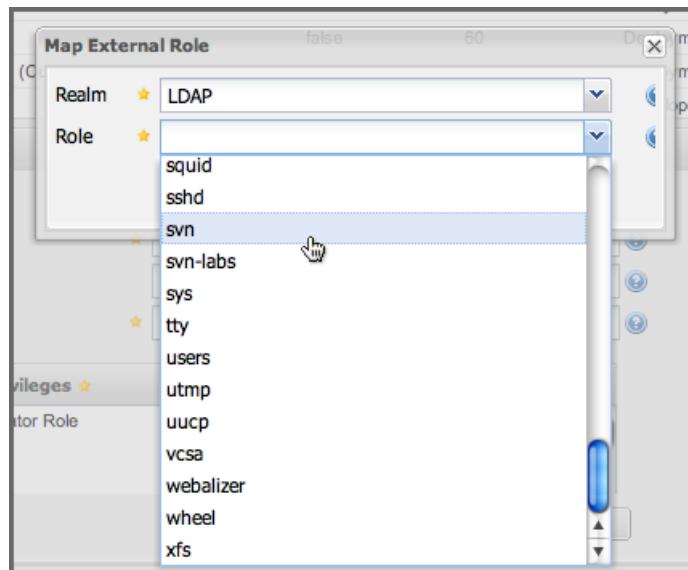


Figure 8.14: Selecting an Externally Managed Role to Map to a Nexus Role

Once the external role has been selected, Nexus will create a corresponding Nexus Role. You can then assign other roles to this new externally mapped role. Figure 8.15 shows that the SVN role from LDAP is being assigned the Nexus Administrator Role. This means that any user that is authenticated against the external LDAP Realm who is a member of the svn LDAP group will be assigned a Nexus role that maps to the Nexus Administrator Role.

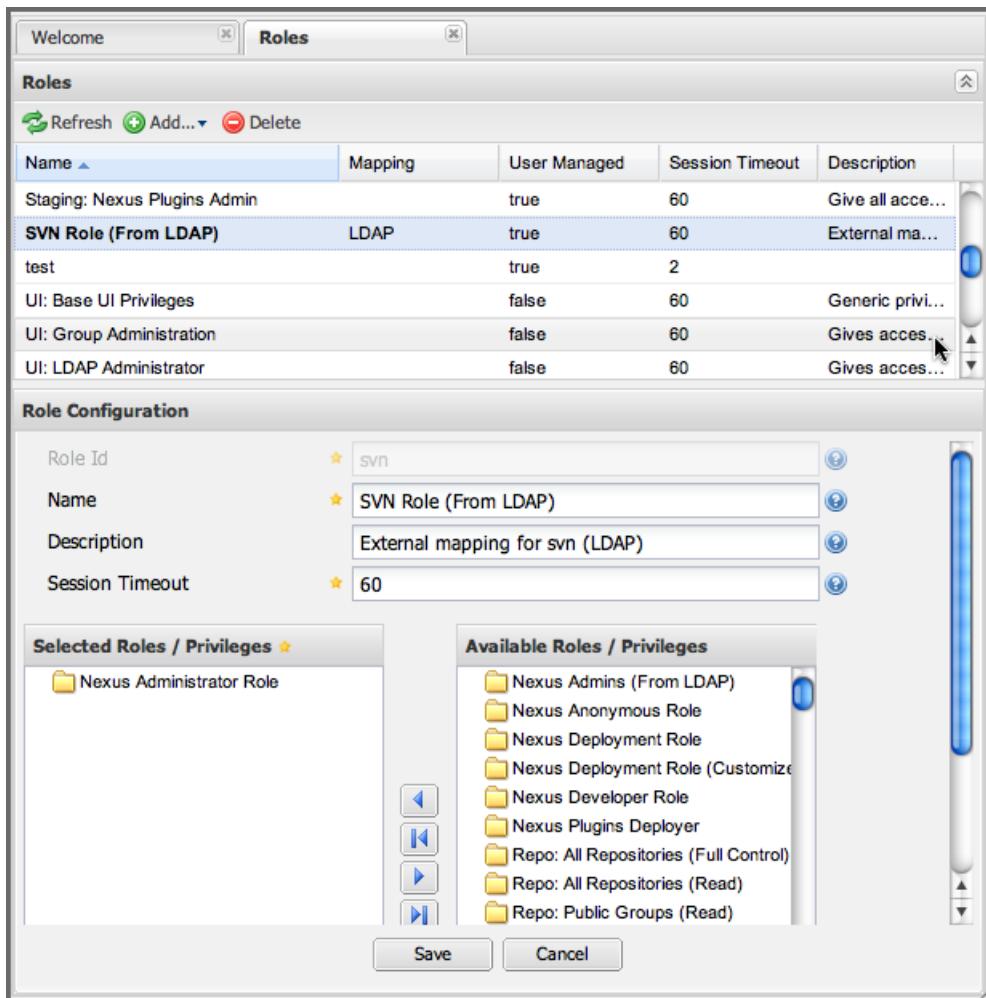


Figure 8.15: Mapping an External Role to a Nexus Role

## 8.11 Enterprise LDAP Support

The following sections outline Enterprise LDAP features available in Nexus Professional.

### 8.11.1 Enterprise LDAP Fail-over Support

When an LDAP server fails, the applications authenticating against it can also become unavailable. Because a central LDAP server is such a critical resource, many large software enterprises will install a series of primary and secondary LDAP servers to make sure that the organization can continue to operate in the case of an unforeseen failure. Nexus Professional's Enterprise LDAP plugin now provides you with the ability to define multiple LDAP servers for authentication. To configure multiple LDAP servers, click on Enterprise LDAP under Security in the Nexus application menu. You should see the Enterprise LDAP panel shown in the following figure.

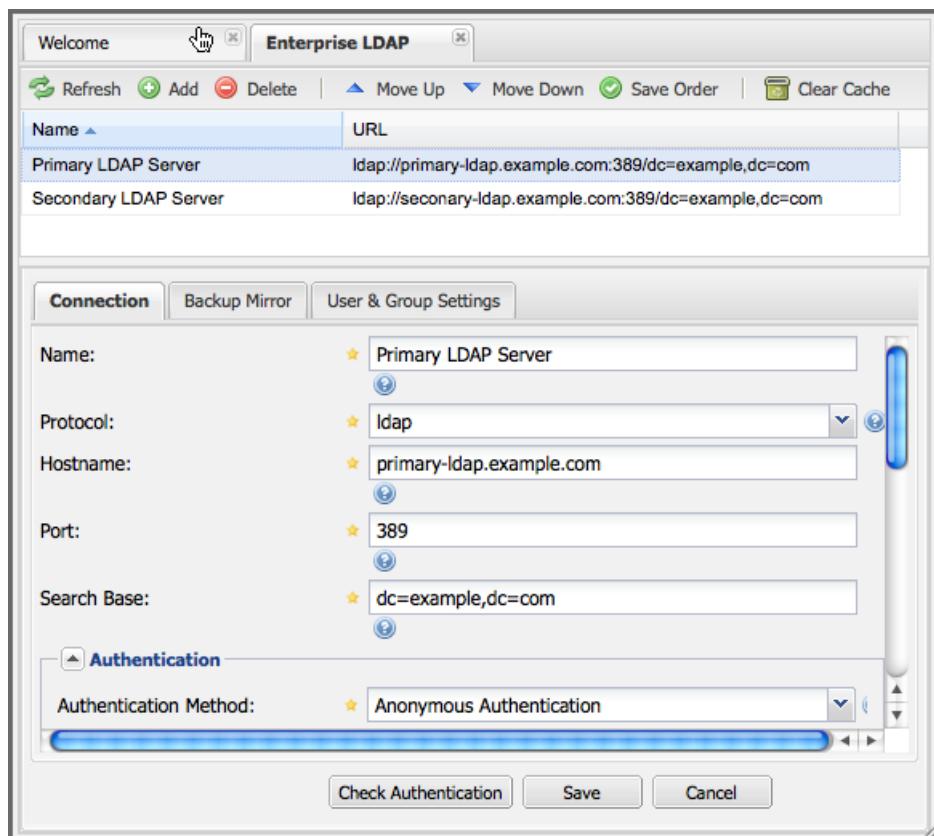


Figure 8.16: Defining Multiple LDAP Servers in Nexus Professional

You can use the *Backup Mirror* setting for an LDAP repository. This backup mirror is another LDAP server that will be consulted if the original LDAP server cannot be reached. Nexus Professional assumes

that the backup mirror is a carbon copy of the original LDAP server, and it will use the same user and group mapping configuration as the original LDAP server. Instead of using the backup mirror settings, you could also define multiple LDAP backup mirrors in the list of configured LDAP servers shown in the previous figure. When you configure more than one LDAP server, Nexus Professional will consult the servers in the order they are listed in this panel. If Nexus can't authenticate against the first LDAP server, Nexus Professional will move on to the next LDAP server until it either reaches the end of the list or finds an LDAP server to authenticate against.

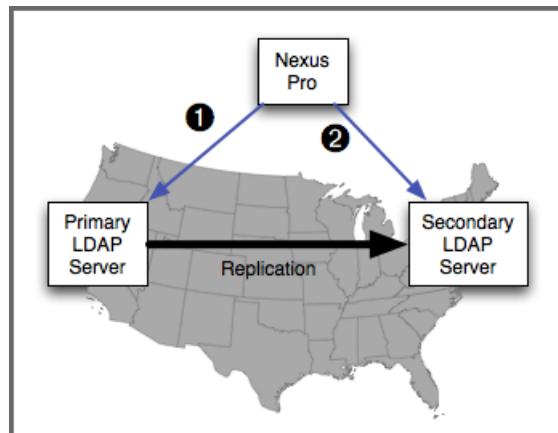


Figure 8.17: Use Multiple LDAP Servers in a Fail-over Scenario

The feature just described is one way to increase the reliability of your Nexus instance. In the previous case, both servers would have the same user and group information. The secondary would be a mirror of the primary. But, what if you wanted to connect to two LDAP servers that contained different data? Nexus Professional also provides...

If you want to connect to two LDAP servers that contain different data, Nexus Professional also provides support for multiple servers and LDAP schemas as described in Section 8.11.2.

### 8.11.2 Support for Multiple Servers and LDAP Schemas

The same ability to list more than one LDAP server also allows you to support multiple LDAP servers that may or may not contain the same user authentication information. Assume that you had an LDAP server for the larger organization containing all of the user information across all of the departments. Now assume that your own department maintains a separate LDAP server that you use to supplement this larger LDAP installation. Maybe your department needs to create new users that are not a part of the

larger organization, or maybe you have to support the integration of two separate LDAP servers that use different schema on each server.

A third possibility is that you need to support authentication against different schema within the same LDAP server. This is a common scenario for companies that have merged and whose infrastructures have not yet been merged. To support multiple servers with different user/group mappings or to support a single server with multiple user/group mappings, you can configure these servers in the Enterprise LDAP panel shown above. Nexus will iterate through each LDAP server until it can successfully authenticate a user against an LDAP server.

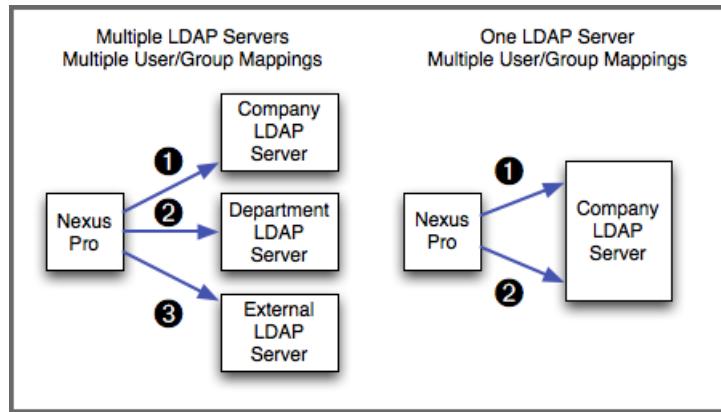


Figure 8.18: Supporting Multiple LDAP Schemas with Nexus Professional

### 8.11.3 Enterprise LDAP Performance Caching and Timeout

If you are constantly authenticating against a large LDAP server, you may start to notice a significant performance degradation. With Nexus Professional you can cache authentication information from LDAP. To configure caching, create a new server in the Enterprise LDAP panel, and scroll to the bottom of the Connect tab. You should see the following input field which contains the number of seconds to cache the results of LDAP queries.

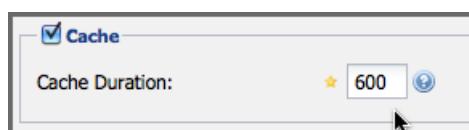


Figure 8.19: Setting the LDAP Query Cache Duration (in Seconds)

You will also see options to alter the connection timeout and retry interval for an LDAP server. If you are configuring a number of different LDAP servers with different user and group mappings, you will want to make sure that you've configured low timeouts for LDAP servers at the beginning of your Enterprise LDAP server list. If you do this properly, it will take Nexus next to no time to iterate through the list of configured LDAP servers.

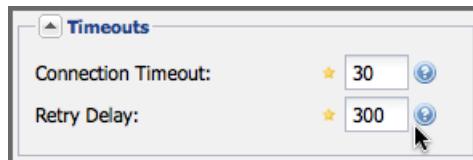


Figure 8.20: Setting the LDAP Connection Timeout (in Seconds)

We improved the overall caching in this release. The cache duration is configurable and applies to authentication and authorization, which translates into pure speed! Once you've configured LDAP caching in Nexus Professional, authentication and other operations that involve permissions and credentials once retrieved from an external server will run in no time.

#### 8.11.4 User and Group Templates

If you are configuring your Nexus Professional instance to connect to an LDAP server there is a very good chance that your server follows one of several, well-established standards. Nexus Professional's LDAP server configuration includes these widely used user and group mapping templates that great simplify the setup and configuration of a new LDAP server. To configure user and group mapping using a template, select a LDAP server from the Enterprise LDAP panel, and choose the User and Group Settings. You will see a User & Group Templates section as shown in the following figure.

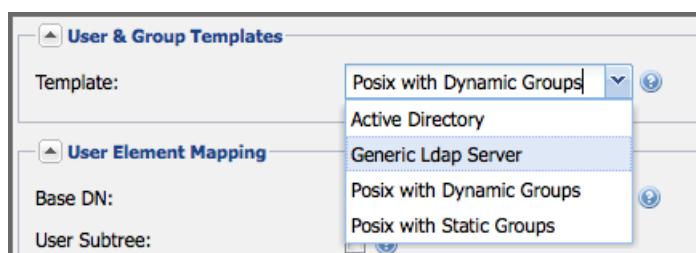


Figure 8.21: Using User and Group Mapping Templates

### 8.11.5 Testing a User Login

Nexus Professional provides you with the ability to test a user login directly. To test a user login, go to the User and Group Settings tab for a server listed in the Enterprise LDAP panel. Scroll to the bottom of the form, and you should see a button named "Check Login".

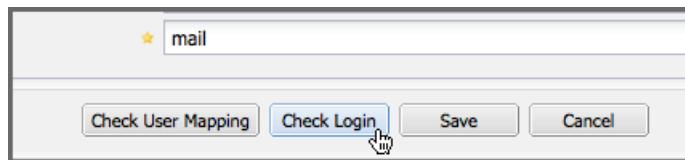


Figure 8.22: Testing a User Login

If you click on Check Login, you will then be presented with the login credentials dialog shown below. You can use this dialog to login as an LDAP user and test the user and group mapping configuration for a particular server. This feature allows you to test user and group mapping configuration directly and to quickly diagnose and address difficult authentication and access control issues via the administrative interface.

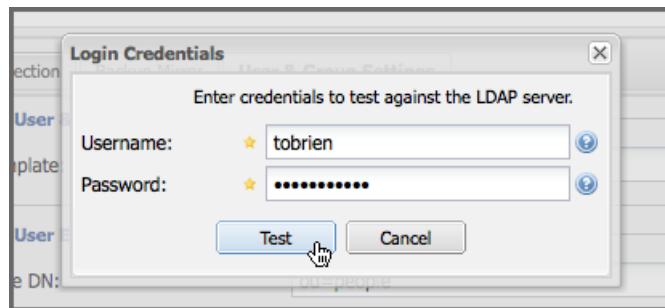


Figure 8.23: Supply a User's Login Credentials

## Chapter 9

# Atlassian Crowd Support

**Atlassian Crowd** is a single sign-on and identity management product that many organizations use to consolidate user accounts and control which users and groups have access to which applications. Nexus Pro contains a security realm that allows you to configure Nexus to authenticate against an Atlassian Crowd instance.

The following steps are necessary to configure Nexus with Crowd-based authentication:

1. [Prepare Nexus](#)
2. [Prepare Atlassian Crowd](#)
3. [Configure the Nexus Crowd Connection](#)
4. [Configure Nexus Crowd Security](#)
5. [Activate the Nexus Crowd Realm](#)

---

**Note**

Atlassian Crowd support is a Nexus Professional feature.

---

## 9.1 Prepare Nexus for Atlassian Crowd

Atlassian Crowd support is preinstalled and ready to configure in Nexus Professional 2.7+.

In older Nexus versions, Nexus Crowd support is implemented as an optional plugin that comes as part of any Nexus Professional download. The directory containing the plugin code is called either enterprise-crowd-plugin-X.Y.Z or nexus-crowd-plugin-X.Y.Z. Install the plugin following the instructions in Section [22.1](#).



### Warning

**Using LDAP and Crowd Realms together in Nexus may work, but this is not supported.**

If you already use Nexus LDAP support, we recommend adding your LDAP server as a Crowd directory accessible to the Crowd *nexus* application instead of using both LDAP and Crowd realms in Nexus.

---

## 9.2 Prepare Atlassian Crowd

### 9.2.1 Compatibility

Always use the latest version of Crowd available at the time your version of Nexus was released. When upgrading to a newer Crowd server, carefully review the Crowd server release notes for REST API backwards compatibility issues.

Crowd support in Nexus 2.7 and greater will only work in Crowd versions (2.1+) that support the Crowd REST API. Older versions use a deprecated SOAP-based API and are less reliable and performant.

Crowd support is actively tested with the highest available version of Crowd at the time Nexus is released.

## 9.2.2 Configure a Nexus Application in the Atlassian Crowd Server

---

**Note**

These instructions are a general guide to adding an application to Crowd. For current detailed instructions, visit the [official Crowd documentation](#).

---

To connect Nexus to Atlassian's Crowd, you will need to configure Nexus as an application in Crowd.

1. Login to Crowd as a user with administrative rights.
2. Click on the *Applications* tab.
3. Click *Add Application* to display the form shown in Figure 9.1, and create a new application with the following values in the *Details* tab of the Add Application form:
  - Application Type: Generic Application
  - Name: nexus
  - Description: Sonatype Nexus Professional
4. Choose a password for this application. Nexus will use this password to authenticate with the Crowd server. Click on the Next button.

The screenshot shows the 'Add Application' dialog box. The 'Details' tab is active. The 'Application Type' dropdown is set to 'Generic Application'. The 'Name' field contains 'nexus'. The 'Description' field contains 'Sonatype Nexus Professional'. The 'Password' and 'Confirm Password' fields both contain masked text. At the bottom are 'Next >' and 'Cancel' buttons.

Figure 9.1: Creating a Nexus Crowd Application

Clicking on *Next* will advance the form to the *Connection* tab shown in Figure 9.2. In this tab you need to supply the URL of your Nexus application instance and the remote IP address for Nexus. Figure 9.2, shows the Connection form configured for a local instance of Nexus. If you were configuring Crowd and Nexus in a production environment, you would supply the URL that users would use to load Nexus in a web browser and you would supply the IP address that Nexus will be connecting from. Once you have completed the *Connection* form, click on *Next* to advance to the *Directories* form shown in Figure 9.3.

The screenshot shows the 'Add Application - nexus' configuration interface. It has five tabs at the top: 1. Details, 2. Connection, 3. Directories, 4. Authorisation, and 5. Confirmation. The '2. Connection' tab is active. There are two main input fields: 'URL:' containing 'http://localhost:8081/nexus' with a 'Resolve IP Address' button next to it, and 'Remote IP Address:' containing '127.0.0.1'. Below these fields is explanatory text: 'The URL where this application resides, e.g. http://jira.atlassian.com' and 'The IP address for the application, e.g. 127.0.0.1'. At the bottom are 'Next >' and 'Cancel' buttons.

Figure 9.2: Creating a Nexus Crowd Application Connection

The *Directories* form allows you to select the user directory used for Nexus authentication. In this example, the default *User Management* directory will be used.

The screenshot shows the 'Add Application - nexus' configuration interface. The '3. Directories' tab is active. A single input field 'User Management:' contains a checked checkbox. Next to the checkbox is the text 'Crowd Internal Directory – This is a user management directory'. At the bottom are 'Next >' and 'Cancel' buttons.

Figure 9.3: Choosing Atlassian Crowd Application Directories

Clicking on the *Next* button in the *Directories* form advances to the *Authorisation* form shown in Figure 9.4. If any of the directories selected in the previous form contain groups, each group is displayed on this form next to a checkbox. You can select *Allow all users* for a directory or you can select specific groups that are allowed to authenticate to Nexus via Crowd. This option would be used if you wanted

to limit Nexus access to specific subgroups within a larger Crowd directory. If your entire organization is stored in a single Crowd directory, you may want to limit Nexus access to a group that contains only developers and administrators.

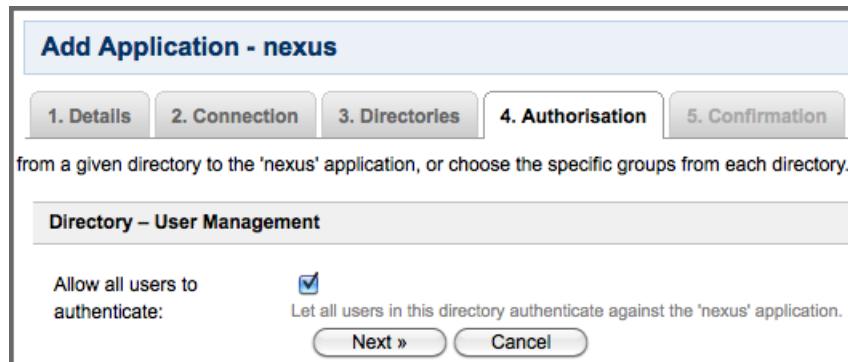


Figure 9.4: Creating a Nexus Crowd Application Authorization

## 9.3 Configure Nexus Crowd Integration

### 9.3.1 Configure Nexus to Trust Crowd's Secure URL (Optional)

Although optional, we advise the connection from Nexus to your Crowd server to use the HTTPS protocol.

If the Crowd Server certificate is not signed by a public certificate authority, you may have to explicitly trust the server certificate using [Nexus SSL support](#). A common symptom observed are peer not authenticated messages, when trying to connect to the Crowd server.

Steps to explicitly trust the Crowd Server URL certificate in Nexus are:

1. [Enable the SSL: Crowd capability](#)
2. [Add the Crowd server certificate to Nexus truststore](#)
3. [Configure Crowd Connection URL using the HTTPS url](#)

**Note**

The *SSL: Crowd* capability is only available in Nexus 2.7+. Older versions must manually configure trust using an explicit truststore specified with JRE system properties.

### 9.3.1.1 Enabling the SSL: Crowd Capability

1. Login to Nexus as an Administrator.
2. In the sidebar menu, click *Administration* → *Capabilities* to open the *Capabilities* panel.
3. Click the *Add* button in the panel toolbar. Select *SSL: Crowd* in the *Type* field. Make sure the *Enabled* checkbox is checked, and click the *Save* button.

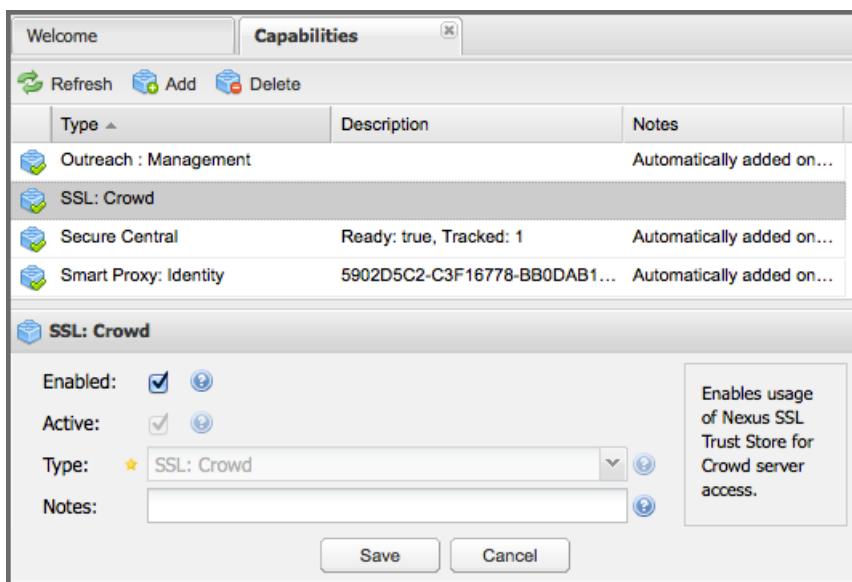


Figure 9.5: SSL: Crowd Capability

### 9.3.1.2 Adding the Crowd Server Certificate to the Nexus Truststore

In order to add the server certificate of your Crowd server to the Nexus truststore, locate the HTTPS *Crowd Server URL* and follow the *Load from server* instructions in Section 24.2.1.

### 9.3.2 Configure Nexus Crowd Connection

The Crowd Configuration screen displayed in Figure 9.6 can be accessed by users with administrative privileges in Nexus by selecting *Crowd* in the *Security* section of the Nexus menu.

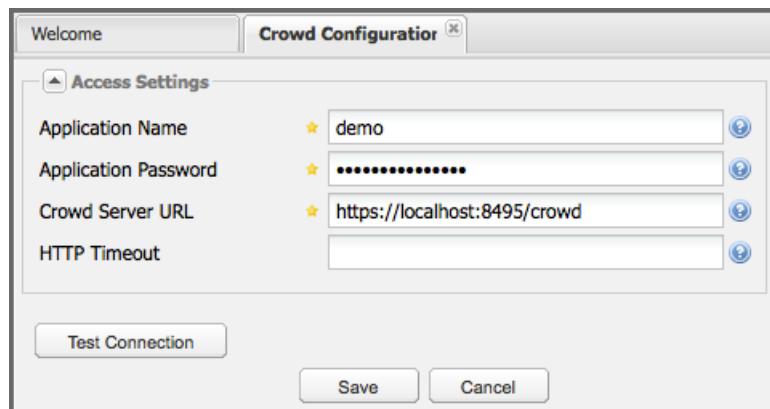


Figure 9.6: Crowd Configuration Panel

This panel contains the following fields:

#### Application Name

This field contains the application name of a Crowd application. This value should match the value in the Name field of the form shown in Figure 9.1.

#### Application Password

This field contains the application password of a Crowd application. This value should match the value in the Password field of the form shown in Figure 9.1.

#### Crowd Server URL

This is the URL used to connect to the Crowd Server. Both *http://* and *https://* URLs are accepted. You may need to [trust the crowd server certificate](#) if a *https://* URL is used.

#### HTTP Timeout

The HTTP Timeout specifies the number of milliseconds Nexus will wait for a response from Crowd. A value of zero indicates that there is no timeout limit. Leave the field blank to use the Nexus server default HTTP timeout.

You can use the *Test Connection* button to validate if your connection to Crowd is working. Once you

have a working connection, do not forget to *Save* your configuration. Use *Cancel* to abort saving any changes.

## 9.4 Configure Nexus Crowd Security

There are two approaches available to manage what privileges a Crowd user has when they login to Nexus.

1. [Mapping Crowd Groups to Nexus Roles](#)
2. [Mapping Crowd Users to Nexus Roles](#)

---

**Note**

Mapping Crowd Groups to Nexus Roles is preferable because there is less configuration is involved overall in Nexus and assigning users to Crowd groups can be centrally managed inside of Crowd by your security team after the initial Nexus setup

---

### 9.4.1 Mapping a Crowd Group to Nexus Role

When mapping a Crowd group to a Nexus role, you are specifying the permissions ( via roles ) that users within the Crowd group will have after they authenticate to Nexus.

To map a Crowd group to a Nexus role, open the *Roles* panel by clicking on the *Roles* link under the *Security* section of the Nexus sidebar menu. Click on the *Add...* button and select *External Role Mapping* as shown in Figure 9.7 and the [Map External Role](#) dialog.

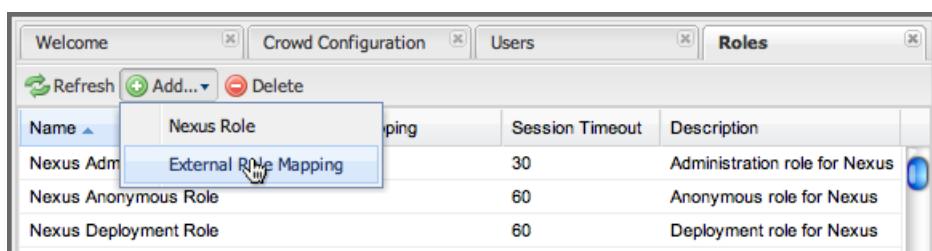


Figure 9.7: Adding an External Role Mapping

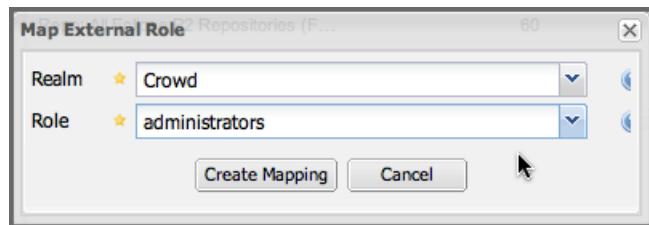


Figure 9.8: Mapping an External Crowd Group to a Nexus Role

After choosing the *Crowd* realm, the *Role* drop-down should list all the Crowd groups the *nexus* crowd application has access to. Select the group to would like to map in the *Role* field and click *Create Mapping*.

---

**Note**

If you have two or more groups in Crowd accessible to the *nexus* application with the same name but in different directories, Nexus will only list the first one that Crowd finds. Therefore, Crowd administrators should avoid identically named groups in Crowd directories.

---

Before saving the group-to-role mapping, **you must add at least one Nexus role to the mapped group**. After you have added the Nexus roles using the *Add* button, click the *Save* button.

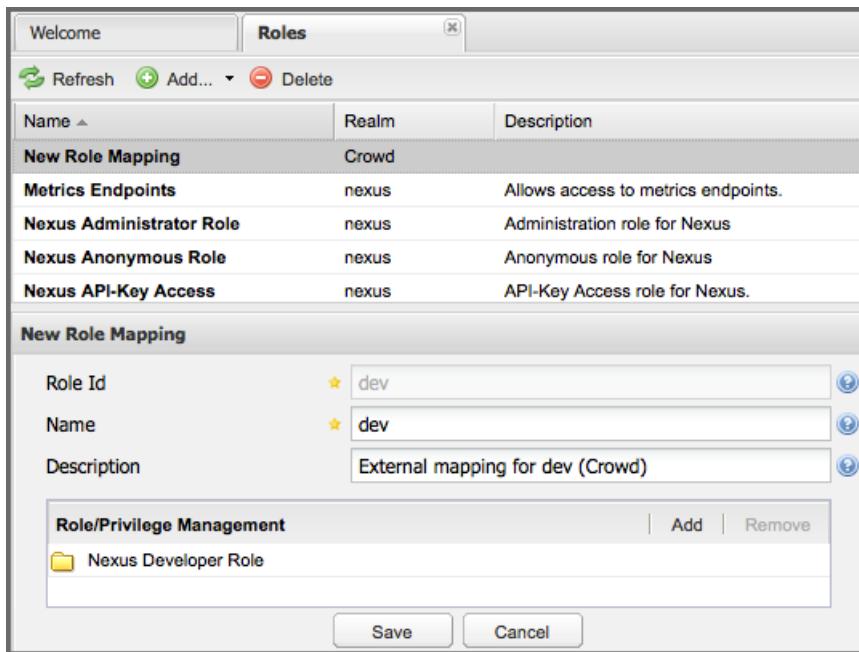


Figure 9.9: Unsaved Mapping of External Crowd *dev* Group to Nexus Developers Role

Saved mappings will appear in the list of Nexus Roles with a mapping value of *Crowd*, as shown in Figure 9.10.

The screenshot shows the Nexus Roles configuration interface. At the top, there is a navigation bar with tabs for 'Welcome' and 'Roles'. Below the navigation bar is a toolbar with 'Refresh', 'Add...', and 'Delete' buttons. The main area displays a table of roles:

Name	Realm	Description
dev	Crowd	External mapping for dev (Crowd)
Metrics Endpoints	nexus	Allows access to metrics endpoints.
Nexus Administrator Role	nexus	Administration role for Nexus
Nexus Anonymous Role	nexus	Anonymous role for Nexus
Nexus API-Key Access	nexus	API-Key Access role for Nexus.

Below the table, a section titled 'dev' contains configuration details for the 'dev' role:

Role Id	Value
Name	dev
Description	External mapping for dev (Crowd)

Under the 'Configuration' tab, there is a 'Role/Privilege Management' section containing a tree view with 'Nexus Developer Role' selected. At the bottom of the page are 'Save' and 'Reset' buttons.

Figure 9.10: Mapped External Crowd *dev* Group to Nexus Developers Role

#### 9.4.2 Mapping a Crowd User to Nexus Role

To illustrate this feature, consider the Crowd server user with an id of `brian`. As visible in the Crowd administrative interface in Figure 9.11, the user is a member of the `dev` group.

The screenshot shows the Crowd interface with the 'Users' tab selected. On the left, there's a sidebar with options like 'Search Users', 'Add User', 'Import Users', 'Reset Password', and 'Remove User'. The main area is titled 'View User – brian'. Below the title, there are tabs for 'Details', 'Attributes', 'Groups', and 'Applications', with 'Groups' being the active tab. It displays a table of groups the user is a member of:

Group	Description	Active
<a href="#">dev</a>	developers	true

At the bottom are buttons for 'Add Groups' and 'Remove Groups'.

Figure 9.11: Crowd Groups for User "brian"

To add an *External User Role Mapping*, open the *Users* panel in Nexus by clicking *Users* in the *Security* section of the Nexus sidebar menu.

Click on the *Add...* button and select *External User Role Mapping* from the drop-down as shown in Figure 9.12.

The screenshot shows the Nexus interface with the 'Users' tab selected. At the top, there are tabs for 'Welcome', 'Crowd Configuration', 'Users', and 'Roles'. Below the tabs is a toolbar with 'Refresh', 'Add...', 'Delete', and a search bar. The main table lists users with columns for 'User ID', 'Name', 'Email', and 'Roles'. One row for 'admin' is selected, and a dropdown arrow next to the 'Add...' button is pointing to the 'External User Role Mapping' option.

User ID	Name	Email	Roles
admin	Nexus User	changeme@yourcompany.com	Nexus Admin...
deployment	External User Role Mapping	changeme1@yourcompany.com	Repo: All O...
anonymous	Nexus Anonymous User	changeme2@yourcompany.com	Repo: All O...

Figure 9.12: Adding an External User Role Mapping

Selecting *External User Role Mapping* will show a mapping panel where you can [locate a user by Crowd user id](#).

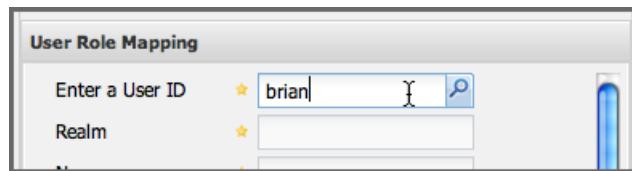


Figure 9.13: Locate a Crowd User by User ID

Typing the Crowd user id - for example `brian` - in the *Enter a User ID* field and clicking the magnifying glass icon, will cause Nexus to search for a user ID `brian` in all known realms, including Crowd.

Once you locate the Crowd user, use *Add* button to add Nexus roles to the Crowd User. **You must map at least one Nexus role to the Crowd managed user** in order to *Save*. Figure 9.14 displays the `brian` Crowd realm user as a member of the `dev` Crowd group and the mapped Nexus role called *Nexus Administrator Role*. External groups like `dev` are bolded in the *Role Management* list.

The screenshot shows the Nexus interface for managing users. At the top, there's a navigation bar with tabs for 'Welcome', 'Roles', 'nexus', and 'Users'. Below the navigation is a toolbar with 'Refresh', 'Add...', 'Delete', and a search field for 'All Configured Users'. A note says 'Select a filter to enable username search'.

The main area displays a table of users:

User ID	Realm	Email	Roles
admin	default	changeme@yourcompany.com	Nexus Administrator Role
deployment	default	changeme1@yourcompany.com	Repo: All Repositories (Full Control), Nexus Deplo...
anonymous	default	changeme2@yourcompany.com	Nexus Anonymous Role, Repo: All Repositories (...)
brian	Crowd	noop@sonatype.com	dev, Nexus Administrator Role

Below the table, a specific user is selected: 'brian'. The interface shows configuration details for this user:

**Config** tab (selected):

User ID:	★ brian
Realm:	★ Crowd
First Name:	★ Fake
Last Name:	★ User
Email:	★ noop@sonatype.com

**Role Management** section:

- dev (selected)
- Nexus Administrator Role

At the bottom are 'Save' and 'Reset' buttons.

Figure 9.14: Mapped External Crowd User Example

## 9.5 Activate Nexus Crowd Realm

The final step to allow Crowd users to authenticate against Nexus is to activate the Crowd authorization realm in the *Security Settings* displayed in Figure 9.15.

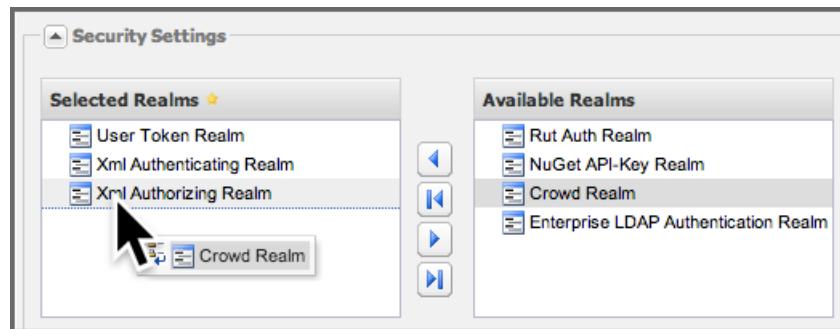


Figure 9.15: Activating the Crowd Realm

1. Select *Administration* → *Server* from the Nexus sidebar menu.
2. Scroll down to the *Security Settings* section.
3. Drag *Crowd Realm* from the list of *Available Realms* to the end of the *Selected Realms* list.
4. Save the server settings.

## Chapter 10

# Nexus Procurement Suite

### 10.1 Introduction

Nexus Procurement Suite provides an organization with control over what components are allowed into a repository from an external, proxied repository such as the Central Repository. Such control can be a prerequisite for organizations unwilling or unable to trust the entire contents of an external public repository. If an organization is developing mission critical code, they will likely want to subject every third party dependency to intense scrutiny and testing before making the component available to build a release or support a team of developers. In most Enterprise development environments, a developer can't just decide to add a new dependency to Hibernate or to the Spring Framework on a whim; the decision to add dependencies to third-party libraries will need to be funnelled through an oversight process that relies on an architect or an administrator to promote components to a certified release repository.

Another more common experience is an organization that needs to proxy like the Central Repository or any other public repository, but wants to limit access to specific versions of components or prevent dependencies on all components contained under a specific group. Some organizations are more amenable to trusting the contents of a remote, proxied repository like the Central Repository, but they also need the ability to block certain dependencies. Maybe you work on a team that needs to limit access to dependencies with a certain license, or maybe you just want to make sure no one uses a problematic version of Hibernate with a known bug? The procurement suite is the tool that provides for both coarse and fine-grained control of the artifacts that can appear in a repository.

---

## 10.2 The Stages of Procurement

A procured repository is a hosted Repository that procures components from a Proxy Repository while procurement is enabled. For example, one could create a hosted repository named "Approved From Central" and then configure this hosted repository to procure components from the "Central" repository. Once the hosted repository has been created and the source of procurement has been configured, the repository will obtain components from the proxy repository as long as procurement is activated. If you start procurement for a hosted repository, the hosted repository will fetch artifacts from the proxy repository specified in the procurement settings. If you stop procurement for a hosted repository, no additional components will be retrieved from the proxy repository specified in the procurement settings. Without procurement active it is a hosted repository and therefore completely static.

The ability to enable or disable procurement for a hosted repository comes in very handy when you want to "certify" a hosted repository as containing all of the components (no more and no less) required for a production build. You can start procurement, run a build that triggers artifact procurement, and then stop procurement, knowing that the procured repository now contains all of the components required for building a specific project. Stopping procurement assures you that the contents of the repository will not change if the third-party, external proxied repository does. This is an extra level of assurance that your release components depend on a set of components under your complete control.

## 10.3 Two Approaches to Procurement

There are two main use cases for the Procurement Suite. In the first use case, the *Procured Release Repository*, the procurement features are used to create a procured release repository to make sure that the organization has full control over the components that are making it into a production release. The other use case, the *Procured Development Repository*, is for organizations that need more up-front control over which artifacts are allowed during the development of a project. The following sections describe these two uses cases in more detail.

### 10.3.1 Procured Release Repository

The Procurement Suite can be used in two different ways. In the "Procured Release" mode, developers work with a proxied third-party repository exactly as they would without the Procurement Suite. When a developer needs to add a dependency on a new artifact, Nexus will retrieve the artifact from the third-party repository (like Central or Apache Snapshots) and this artifact will be served to Maven via a proxied Nexus repository. When a QA or Release engineer needs to build a release or staging artifact, the Release

---

or QA build would be configured to execute against a procured repository or repository group with only approved and procured repositories. A procured repository is one that only serves the components that have been explicitly approved using the Procurement Suite.

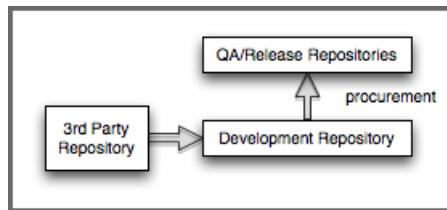


Figure 10.1: Procurement to a Certified Release Repository

In this model, developers can add as many third-party dependencies as they want, and it is the responsibility of the QA and release engineers to approve (or procure) components from the development Repository to the QA/Release repository. Developers can move forward, adding dependencies freely from a third-party, proxied repository, but once it is time to populate a release repository, a Nexus administrator can audit the required components, create a hosted repository, turn on procurement, populate the repository, and then deactivate procurement. This has the effect of "locking down" the components that are involved in a production release.

### 10.3.2 Procured Development Repository

There are some development environments that require even more control over which components can be used and referenced by developers. In these situations, it might make sense to only allow developers to work with a procured repository. In this mode, a developer must ask a Nexus administrator for permission to add a dependency on a particular third-party artifact. A procurement manager would then have to approve the component or group of components so that they would be made available to the developers. This is the "ask-first" model for organizations that want to control which components make it into the development cycle.

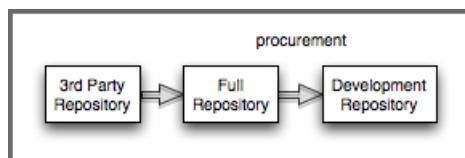


Figure 10.2: Procurement to a Certified Development Repository

This is a model common in industries that have strict oversight requirements. More often than not, banks, hospitals, and government agencies have fairly strict regulations on the software that can be used by large development teams. With the Procured Development Repository approach, an architecture group can have full control over what components can be referenced by a large development team.

### 10.3.3 Providing Access with a Repository Group

In a typical usage a software build relies on approved components that have successfully passed procurement and additional components that have been authored internally in the organization and are available on Nexus as well.

In order to use a combination of such components together with the procured component, you should set up a repository group that contains all repositories with preapproved components as well as the procurement repository. For example, the release and snapshot repositories could be added to the group, based on the assumption that any internally authored components deployed there are automatically approved. In addition, you could add the third-party repository, if all uploads to it are done with prior approval of the specific components.

Once this repository group is set up, you can reference it from any tool just like the public group, e.g., in a separate settings.xml used by builds that can only have access to the approved components.

---

**Tip**

When running builds you need to make sure that you run to run clean builds. No components from other builds, accessing non-procured repositories, should be in the local repository of the build. This ensures that only approved components are used in the build. The easiest way to achieve this is to clear the local repository before a build or to run the build against a project specific local repository.

---

## 10.4 Setting up a Procured Repository

If you installed Nexus Professional, the Nexus Procurement Suite is already installed and available via the *Artifact Procurement* option in the *Enterprise* menu of the Nexus interface.

This section will walk through the process of creating and configuring a hosted repository named *Approved From Central* which will be procured from the *Central* proxy repository. Setting up a procured repository consists of the following steps:

---

- Enable the remote index downloads for the proxy repository, that will act as the source of the procured components.
- Create a hosted repository, which will be the target of the procurement.
- Configure procurement for the hosted repository.
- Configure the procurement rules.

Before configuring a procured repository, you need to make sure that you have enabled Remote Index downloading for the proxied repository that will serve as the source for your procured repository.

---

**Note**

If you are attempting to procure components from a remote repository that does not have a repository index, you can still use the procurement suite. Without a remote repository index, you will need to configure procurement rules manually without the benefit of the already populated repository tree shown in Section [10.5](#).

---

#### 10.4.1 Enable Remote Index Downloads

When you configure procurement rules for a hosted repository, the administrative interface displays the repository as a tree view using the Maven repository format of groups and components using populated from remote repository's index. Nexus ships with a set of proxy repositories, but remote index downloading is disabled by default.

To use procurement effectively, you will need to tell Nexus to download the remote indexes for a proxy repository. Click on  under *Views/Repositories* in the *Nexus* menu, then click on the *Central Repository* in the list of repositories. Click on the *Configuration* tab, locate *Download Remote Indexes*, and switch this option to *True* as shown in Figure [10.3](#).

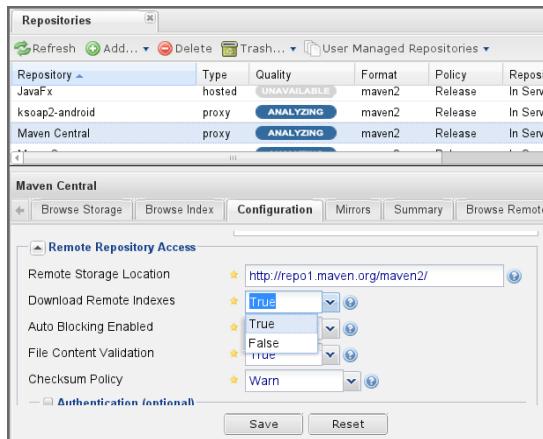


Figure 10.3: Enabling Remote Index Downloads for a Proxy Repository

Click on the *Save* button in the dialog shown in Figure 10.3. Right-click on the repository row in the Repositories list and select *Update Index*. Nexus will then download the remote repository index and recreate the index for any repository groups that contain this proxied repository.

Nexus may take a few minutes to download the remote index for a large repository. Depending on your connection to the Internet, this process can take anywhere from under a minute to a few minutes. The size of the remote index for the Central Repository currently exceeds 50MB and is growing in parallel to the size of the repository itself.

To check on the status of the remote index download, click on *System Feeds* under *Views/Repositories* in the *Nexus* menu. Click on the last feed to see a list of *System Changes in Nexus*. If you see a log entry like the one highlighted in Figure 10.4, Nexus has successfully completed the download of the remote index from the Central Repository.

The screenshot shows the 'System Feeds' section of the Nexus interface. It displays a list of feeds with their URLs:

- Broken artifacts in all Nexus repositories (checksum errors, v) <http://localhost:8081/nexus/service/local/feeds/brokenArtifacts>
- New artifacts in all Nexus repositories (cached or deployed). <http://localhost:8081/nexus/service/local/feeds/recentCacheOrDeployments>
- New cached artifacts in all Nexus repositories (cached). <http://localhost:8081/nexus/service/local/feeds/recentlyCached>
- New deployed artifacts in all Nexus repositories (deployed). <http://localhost:8081/nexus/service/local/feeds/recentlyDeployed>
- Recent storage changes in all Nexus repositories (caches, d) <http://localhost:8081/nexus/service/local/feeds/recentChanges>

Below this, the 'System changes in Nexus.' section shows two reindexing tasks:

Title	Date
Reindexing	Today 1:58 pm
Reindexing repository with ID=central : Process started on Sun Dec 21 13:56:05 CST 2008, finished successfully on Sun Dec 21 13:58:33 CST 2008.	
Reindexing	Today 1:56 pm
Reindexing repository with ID=central : Process started on Sun Dec 21 13:56:05 CST 2008, not yet finished..	
procurement.scan	Fri 10:41 pm

Figure 10.4: Verification that the Remote Index has been Downloaded

#### 10.4.2 Create a Hosted Repository

When you configure procurement you are establishing a relationship between a proxy repository and a hosted repository. The hosted repository will be the static container for the components, while the proxy repository acts as the component source. To create a hosted repository, select *Repositories* from the *Views.Repositories* section of the *Nexus* menu, and click on the *Add* button selecting *Hosted Repository* as shown in Figure 10.5.

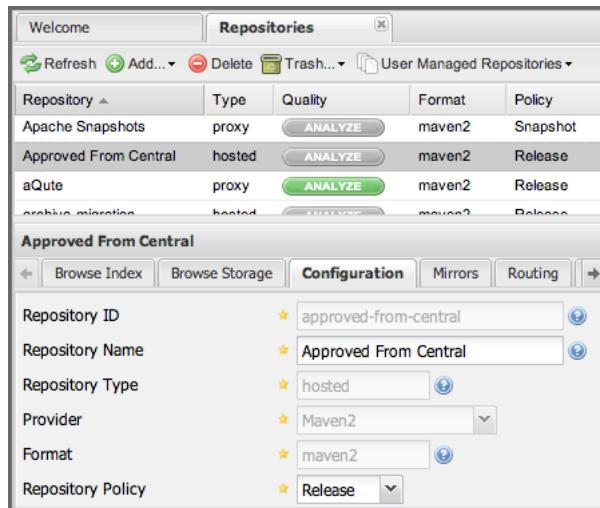


Figure 10.5: Adding the "Approved From Central" Hosted Repository

Selecting *Hosted Repository* will then load the configuration form. Create a repository with a *Repository ID* of `approved-from-central` and a name of Approved From Central. Make the release policy Release. Click the *Save* button to create the new hosted repository.

### 10.4.3 Configuring Procurement for Hosted Repository

At this point, the list of Repositories will have a new Hosted repository named +Approved From Central=. The next step is to start procurement for the new repository. When you do this, you are establishing a relationship between the new hosted repository and another repository as source of components. Typically, this source is a proxy repository. In this case, we're configuring procurement for the repository and we're telling the Procurement Suite to procure artifacts from the *Central* proxy repository. To configure this relationship and to start procurement, click on *Artifact Procurement* under the *Enterprise* menu. In the *Procurement* panel, click on *Add Procured Repository* as shown in Figure 10.6.

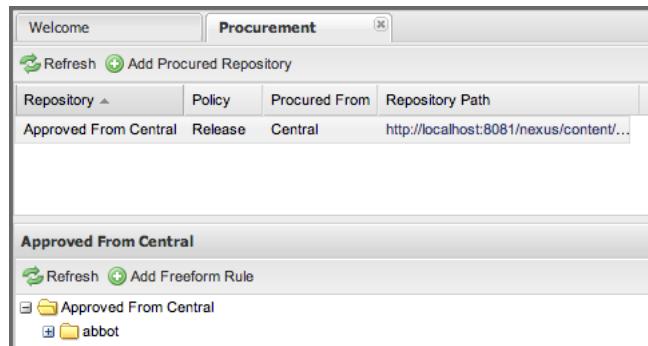


Figure 10.6: Adding a Procured Repository

You will then be presented with the Start Procurement dialog as shown in Figure 10.7. Select the "Central" proxy repository from the list of available Source repositories.

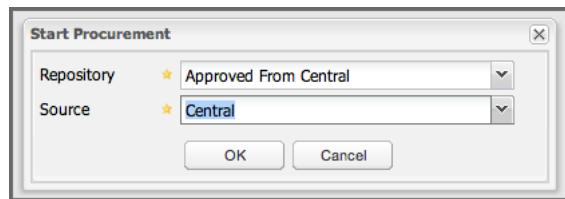


Figure 10.7: Configuring Procurement for a Hosted Repository

Procurement is now configured and started. If you are using an instance of Nexus installed on localhost port 8081, you can configure your clients to reference the new repository at <http://localhost:8081/nexus/content/>.

By default, all artifacts are denied and without further customization of the procurement rules no components will be available in the new repository.

One interesting thing to note about the procured repository is that the repository type changed once procurement was started. When procurement is activated for a hosted repository, the repository will not show up in the repositories list as a *User Managed Repository*. Instead it will show up as a proxy repository in the list of *Nexus Managed Repositories*. Use the drop-down for *User Managed/Nexus Managed Repositories* in the Repositories list. Click Refresh in the Repositories list, and look at the *Approved From Central* repository in the list of Nexus Managed Repositories. You will see that the repository type column contains *proxy* as shown in Figure 10.8. When procurement is started for a hosted repository,

it is effectively a proxy repository, and when it is stopped it will revert back to being a normal hosted repository.

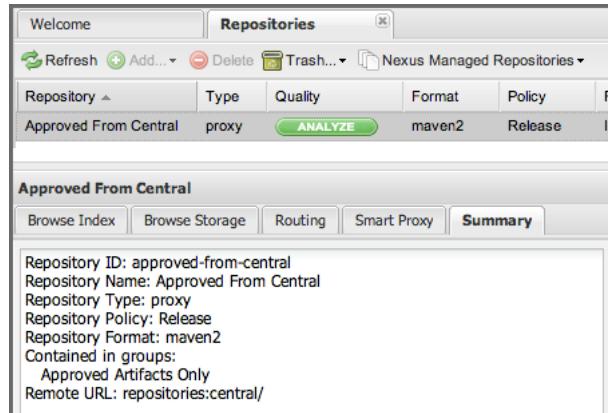


Figure 10.8: Hosted Repository is a Nexus Managed Proxy Repository while Procurement is Active

#### 10.4.4 Procured Repository Administration

Once you've defined the relationship between a hosted repository and a proxy repository and you have started procurement, you can start defining the rules that will control which components are allowed in a procured repository and which components are denied. You can also start and stop procurement. This section details some of the administration panels and features that are available for a procured repository.

A procurement rule is a rule to allow or deny the procurement of a group, artifact, or a collection of groups or artifacts. You load the Artifact Procurement interface by selecting Artifact Procurement in the Enterprise menu of the Nexus left-hand navigation. Clicking on this link will load a list of procured repositories. Clicking on the repository will display the proxied source repository and the current content of the procured repository in a tree as shown in Figure 10.9.

This section will illustrate the steps required for blocking access to a specific component and then selectively allowing access to a particular version of that same component. This is a common use case in organizations that want to standardize specific versions of a particular dependency.

**Note**

If you are attempting to procure components from a remote repository that does not have a repository index, you can still use the procurement suite. Without a remote repository index, you will need to configure procurement rules manually without the benefit of the already populated repository tree shown in this section.

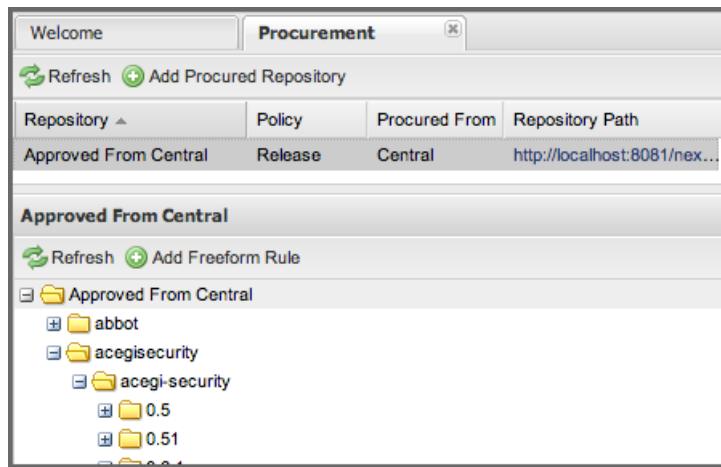


Figure 10.9: Viewing a Repository in the Artifact Procurement Interface

The directory tree in Figure 10.9 is the index of the proxy repository from which artifacts are being procured.

## 10.5 Configuring Procurement

To configure a procurement rule, right-click on a folder in the tree. Figure 10.10 displays the procurement interface after right-clicking on the `org/eclipse/aether` component folder.

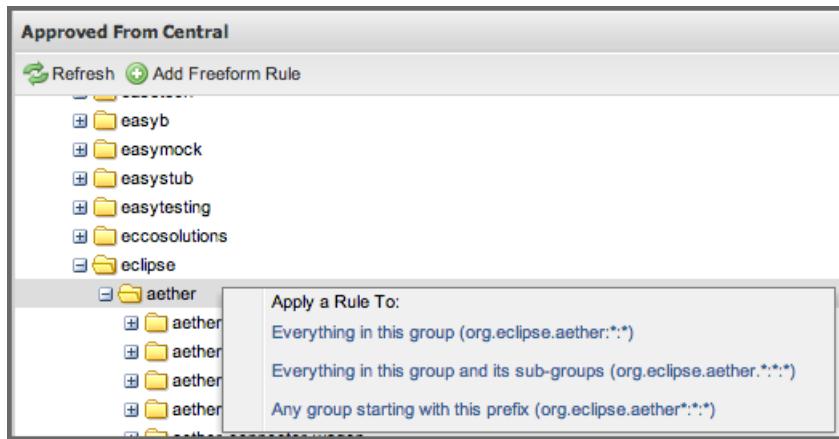


Figure 10.10: Applying a Rule to a Component Folder for `org/eclipse/aether`

In this dialog, we are deciding to configure a rule for everything within the group and its sub groups that display the rule configuration dialog displayed in Figure 10.11. The dialog to add rules allows you to select the available rule, e.g., a Forced Approve/Deny Rule, and configure the rule properties. The displayed dialog approves all components Eclipse Aether components.



Figure 10.11: Approving `org.eclipse.aether` Components

By right-clicking on the top level folder of the repository, as displayed in Figure 10.12, you can configure rules for the complete repository as well as access all configured rules via the *Applied Rules* option.



Figure 10.12: Accessing the Global Repository Configuration

This allows you to set up a global rule, like blocking all components from the repository. Once you have configured this you can then selectively allow specific versions of a component. Figure 10.13 displays the options available for configuring rules for a specific component version of the Apache Commons Collections component.

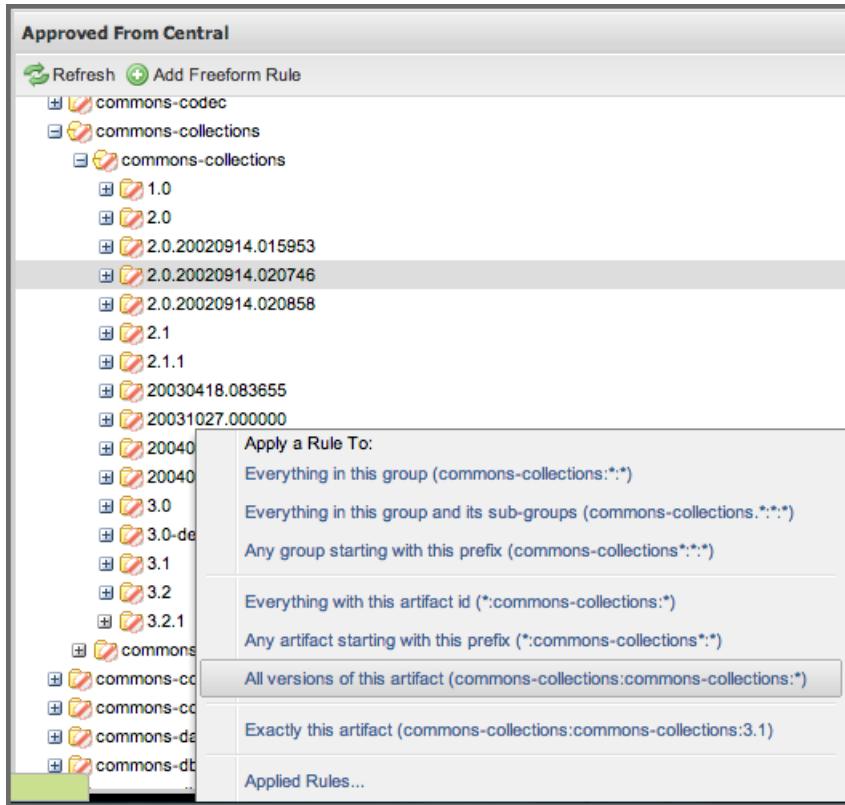


Figure 10.13: Procurement Configurations Options for a Specific Component Version

Once you approve a specific version, the tree view will change the icons for the component displaying green checkmarks for approved components and red cross lines for denied components as visible in Figure 10.14. The icons are updated for signature validation rule violations, if applicable, showing a yellow icon.

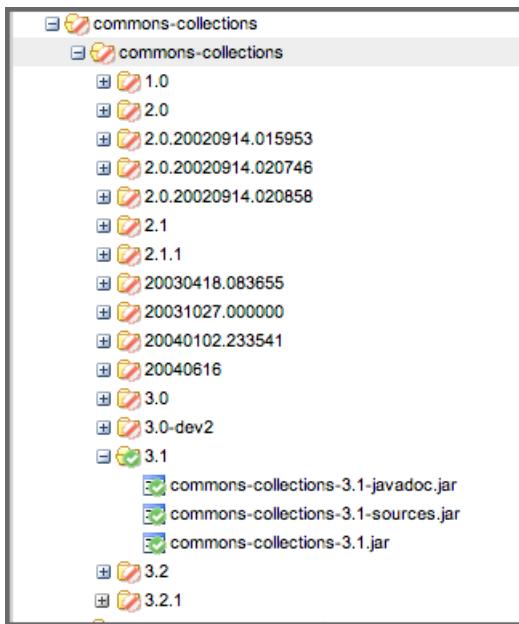


Figure 10.14: Procurement Repository Tree View with Rule Visualization

An example dialog of Applied Rules for the complete repository, as configured by `::*`, is visible in Figure 10.15. This repository currently denies access to all components, only approving components within `org/apache/maven` and `org/eclipse/aether`.

This dialog gives the procurement administrator a fine-grained view into the rules that apply to the complete repository. A view of all Applied Rules for a specific repository folder can be accessed by right-clicking on the folder and selecting Applied Rules. The dialog allows you to remove specific rules or all rules as well.

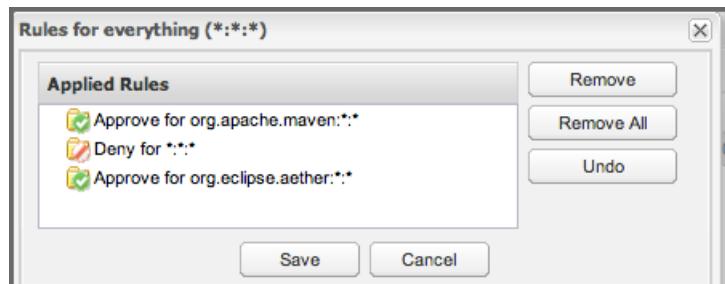


Figure 10.15: Applied Rules for the Complete Procurement Repository

The *Refresh* button above the tree view of a repository tree view allows you to update the tree view and to see all of the applied rules. The *Add Freeform Rule* button allows you to display the dialog to manually configure a procurement rule displayed in Figure 10.16. This is especially useful if the tree view is not complete due to a missing repository index or if you have detailed knowledge of the component to which you want to apply a rule. The format for entering a specific component in the *Enter GAV* input field is the short form for a Maven component coordinate using the groupId, artifactId and version separated by `:`. The `*` character can be used as a wildcard for a complete coordinate.

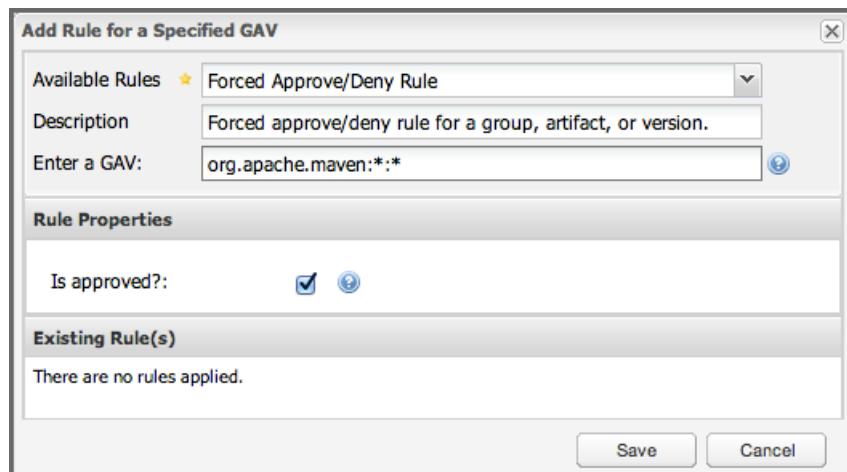


Figure 10.16: Adding a Freeform Rule

Examples for freeform rule coordinates are:

**\* : \* : \***

matches any component in the complete repository

**org.apache.ant : \* : \***

matches any component with the groupId `org.apache.ant` located in `org/apache/ant`

**org.apache.ant.\* : \* : \***

matches any component with the groupId `org.apache.ant` located in `org/apache/ant` as well as any sub-groups e.g., `org.apache.ant.ant`

These coordinates are displayed in the Maven build output log when retrieving a component fails. You can see them as part of the error message with the addition of the packaging type. It is therefore possible to cut and paste the respective coordinates from the build output and insert them into a freeform rule. Once you have done that you can kick off the build again, potentially forcing downloads with the option `-U` and continue procurement configuration for further components.

## 10.6 Stopping Procurement

Some organizations may want to lock down the components that a release build can depend upon. It is also a good idea to make sure that your build isn't going to be affected by changes to a repository not under your control. A procurement administrator can configure a procured repository, start procurement, and run an enterprise build against the repository to populate the procured, hosted repository with all of the necessary components. After this process, the procurement administrator can stop procurement and continue to run the same release build against the hosted repository that now contains all of the procured components while being a completely static repository.

To stop procurement, go to the procurement management interface by clicking on *Artifact Procurement* under the *Enterprise* section of the *Nexus* menu. Right-click on the repository and choose *Stop Procurement* as shown in Figure 10.17.

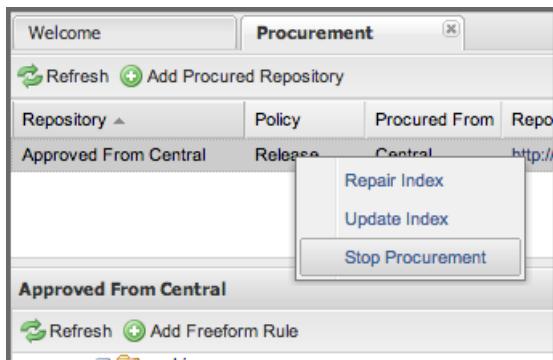


Figure 10.17: Stopping Procurement for a Procured Repository

After choosing *Stop Procurement*, you will then see a dialog confirming your decision to stop procurement. Once procurement is stopped, the procured repository will revert back to being a hosted repository.

In order to add further components, you create a procurement repository off the hosted repository as you did initially. If the repository contains components already, activating procurement will automatically generate rules that allow all components already within the repository.

# Chapter 11

## Improved Releases with Nexus Staging

### 11.1 Introduction

If you release software, you will often need to test a release before deploying it to a production system or an externally accessible repository. For example, if you are developing a large, enterprise web application, you may want to stage a release candidate to a staging system and perform a series of rigorous tests before a release manager makes a decision to either return a system to development or deploy a system to production.

The staging suite in Nexus Professional allows an organization to create a temporary staging repository and to manage the promotion of artifacts from a staging repository to a release repository. This ability to create an isolated, release candidate repository that can be discarded or promoted makes it possible to support the decisions that go into certifying a release, while the certification process is done on the same binaries that will ultimately be released.

#### 11.1.1 Releasing Software without a Staging Repository

Without the staging suite, when a developer deploys an artifact to a hosted repository such as the release repository, this artifact is published and immediately made available, having no oversight, no process and no certification process. There is no chance to test the artifact before writing the artifact to a hosted repository. If there is a mistake in the release, often the only option available is to republish the artifacts

---

to the release repository or deploy a new version of the artifacts.

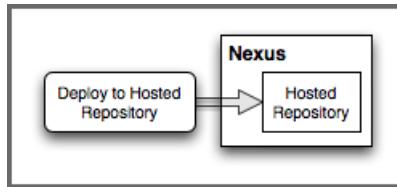


Figure 11.1: Release Deployment Without the Nexus Staging Suite

While this is acceptable for some users, organizations and enterprises with a QA cycle often need a temporary storage for potential release candidates: a staging repository. With the Nexus staging suite, an organization can automatically stage releases to a temporary repository that can then be used to test and certify a set of artifacts, before they are published to a final release repository. This temporary repository can then be promoted as a whole or dropped, depending on the results of testing. When used, the binary artifacts being tested and certified are the identical artifacts that will ultimately be released. You will not have a clean fresh build kicked off after the certification finished, as is often the case without a staging suite being used.

### 11.1.2 How the Staging Suite Works

Here's how staging works in Nexus Professional:

1. A developer deploys an artifact (or a set of artifacts) to Nexus Professional.
2. The staging suite intercepts this deployment and determines if the deployment matches for a staging profile.
3. If a match is found, a temporary staging repository is created and the artifacts are deployed to this repository.
4. Once the developer has deployed a set of artifacts to Nexus, they will then "Close" the staging repository.
5. The Staging Suite will then add this temporary staging repository to one or more Target Repository Groups.

Once the staging repository is closed and has been added to a target repository group, the artifacts in the staging repository are available to Nexus users for testing and certification via a repository group. Tests

can be performed on the artifacts as if they were already published in a hosted repository. At this point different actions can be performed with the staging repository:

### Release

A Nexus user can "release" a staging repository and select a hosted repository to which to publish artifacts. Releasing the contents of a repository publishes all artifacts from the staging repository to a hosted repository and deletes the temporary staging repository.

### Drop

A Nexus user can "drop" a staging repository. Dropping a staging repository will remove it from any groups and delete the temporary staging repository.

### Promote

If your Nexus installation contains Build Promotion profiles, you will also see an option to "promote" a staging repository to a Build Promotion Group. When you promote a staging repository you expose the contents of that staging repository via additional groups. Build Promotion profiles are explained in detail in the next section.

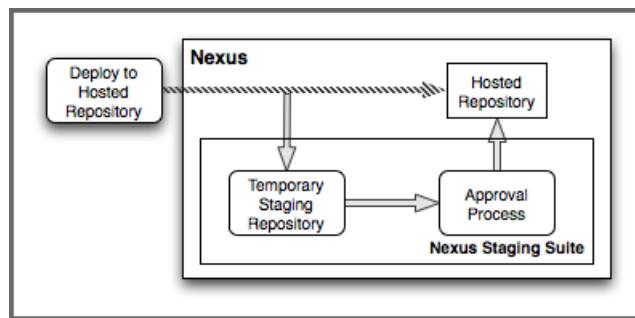


Figure 11.2: Release Deployment with the Nexus Staging Suite

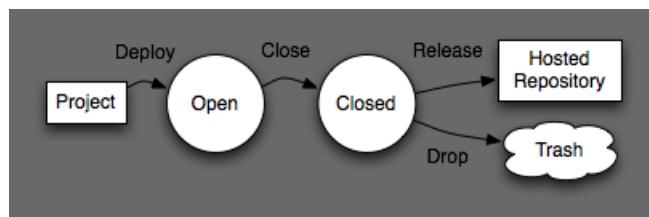


Figure 11.3: The Stages of a Staging Repository Starting with Deployment and Ending with a Release or a Drop of the Repository

## 11.2 Configuring the Nexus Staging Suite

### 11.2.1 Overview

The Staging Suite is part of the default Nexus Professional install and is accessible with the menu items *Staging Profiles*, *Staging Repositories*, *Staging Ruleset*, and *Staging Upload* options in the left-hand navigation menu of the Nexus interface called *Build Promotion*.

*Staging Profiles* define the rules by which artifact deployments from your project are intercepted by Nexus and staged in *Staging Repositories*.

*Staging Repositories* are dynamically created by Nexus as they are needed. They are temporary holding repositories for your artifacts that are used for the different staging related steps. Using them in the Nexus user interface, users can promote the contents of the staging repository to a hosted repository, discard them, and perform other tasks.

*Staging Rulesets* allow you to define rules that the artifacts being deployed have to follow in order to allow successful deployment.

*Staging Upload* allows you to manually upload artifacts to Nexus via the user interface rather than by using your build system.

### 11.2.2 Configuring a Staging Profile

*Staging profiles* control the process by which artifacts are selected for staging. When you define a Staging profile, you are defining a set of rules which will control the way in that Nexus intercepts an artifact deployment and what actions to perform during and after staging the artifacts. When you click on *Staging Profiles* in the *Nexus* menu, you will see a list of configured staging profiles. This list allows you to *Add...* and *Delete* staging profiles. Click on an existing staging profile in the list and the panel below the list will display the configuration of the profile.

The list of staging profiles displayed also determines the order in which the profiles are examined when a component is deployed to staging. Going down the list each profile is checked for a match of the deployed component characteristics to the configuration of the profile. If a match is found a staging repository for this profile with the deployed components is created. Otherwise the next profile in the list is examined. Specifically with implicit matching criteria being used for your deployments as explained in more detail below, this order becomes important and can be controlled by selecting a staging profile and using the

*Move Up* and *Move Down* buttons on the top of the list. Once you have created the desired order, press the *Save Order* button and confirm the order in the dialog.

Clicking on *Add...* will display the drop-down menu shown in Figure 11.4.

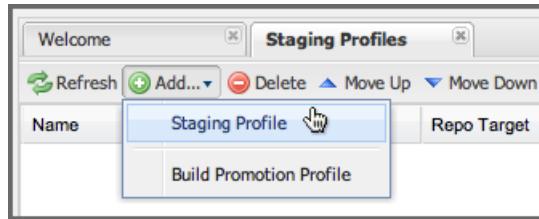


Figure 11.4: Adding a Staging Profile

Selecting *Staging Profile* will create a new staging profile and display the form shown in Figure 11.5.

Figure 11.5 defines a staging profile named `Test`. It is configured to only intercept explicit deployments in the *Profile Selection Strategy* using the *Profile ID* and the Nexus Staging Maven Plugin. It uses the template *Maven2 (hosted, release)* for newly created temporary staging repositories, and it will automatically add closed staging repositories to the *Public Repositories* group. In addition, it is configured to verify the deployment against the rules defined in Sonatype CLM for the *CLM Application Id* `bom1-12345678`.

Name	Mode	Repo Target	Release Re...	Target Groups
TrainingRelease	Deploy and UI Upload	Simpligility	Releases	Public Repositories
CLMTest	Deploy	All (Maven2)	Releases	Approved Artifacts Only, Public Repositories
Test	Deploy and UI Upload	All (Maven2)	Releases	Public Repositories

**Test**

Profile ID	12f3a105e1212751
Deploy URL	http://localhost:8081/nexus/service/local/staging/deploy/maven2
Profile Name	Test
Profile Selection Strategy	Explicit Only - only selected by ID
Searchable Repositories	<input checked="" type="checkbox"/>
Staging Mode	Deploy and UI Upload
Template	Maven2 (hosted, release)
Repository Target	All (Maven2)
Release Repository	Releases
CLM Application Id	bom1-12345678
Content Type	maven2

**Target Groups**

- Public Repositories

**Available Groups**

- Approved Artifacts Only

Figure 11.5: Creating a New Staging Profile

The form allows you to configure a profile with the following fields:

### Profile ID and Deploy URL

These two fields are displayed as "read only" once a profile has been created. The Profile ID displays the unique identifier that can be used for staging to this repository using the Nexus Staging Maven Plugin. The Deploy URL displays the generic staging URL that can be used with the default Maven Deploy Plugin together with the Repository Target configuration to intercept the deployment and move the artifacts into the Staging Suite instead.

### Profile Name

The name of the staging profile. This can be an arbitrary value. It is simply a convenience for the

Nexus Administrator, and it is also used to create Nexus roles that are used to grant permissions to view and manipulate staging repositories created by this profile.

### Profile Selection Strategy

Select the strategy used by Nexus to select this staging profile. `Explicit` or `Implicit` is the default behavior and causes Nexus to select the profile by the provided staging profile identifier and to fall back to an automatic determination, if none is provided. It is necessary to be used with the Maven deploy plugin and the correct staging profile is determined using repository targets together with the generic deploy URL of Nexus.

When using the Nexus Staging Maven Plugin for deployments, and therefore an explicitly defined staging profile in the project POM, the setting should be changed to `Explicit Only`. This will prevent the profile from implicitly capturing a deployment in this repository due to the matching defined and allow Nexus to ensure that the deployment reaches the staging profile with the configured staging profile ID, even if the default matching and staging profile order could potentially cause a deployment to end up in a different profile.

### Searchable Repositories

The default value of enabling this feature will cause any new artifacts in this staging profile to be added to the indexes and therefore be available in search queries. Disable this feature to "hide" artifacts in staging.

### Staging Mode

This field contains the options `Deploy`, `UI Upload`, and `Deploy and UI Upload`. This controls how artifacts can be staged to this staging profile. If `Deploy` is selected, artifacts can only be deployed using Maven to upload build artifacts. If `UI Upload` is selected, users can upload artifacts to Nexus using the Nexus user interface.

### Template

Defines the template for the format of the temporary staging repositories created by this staging profile. The current version of Nexus Professional provides the option `Maven2 (hosted, release)` only. Additional templates can be supplied by plugins that enable staging for other repository types. An example for such a plugin is the [Nexus Yum Plugin](#).

### Repository Target

When a developer deploys an artifact to the generic Deploy URL, the Staging Suite will check to see if the artifact matches the patterns defined in this Repository Target. The repository target defines the "trigger" for the creation of a staging repository from this staging profile and is only needed for implicit deployments with the Deploy URL and not for explicit deployments using the Profile ID.

### Release Repository

Staged artifacts are stored in a temporary staging repository that is made available via Target Groups. Once a staged deployment has been successfully tested, artifacts contained in the temporary staging repository are promoted to a hosted repository as their final storage place. The Release Repository setting configures this target release repository for this staging profile.

### CLM Application Id

Configure the application identifier defined in the Sonatype CLM server to allow to use of the rules defined there for staging within Nexus. More details can be found in Section [11.6](#).

### Content Type

Nexus can create staging repositories for repositories of type Maven2. This value is automatically selected based on the chosen template.

### Target Groups

When a Staging Repository is *closed* and is made available to users and developers involved in the testing process, the temporary Staging Repository is added to one or more Repository Groups. This field defines those groups. It is a best practice to create a separate group, different from the group typically used for development like the default *Public Repositories* group for staging. This prevents the staged artifacts from leaking to all users and allows you to control access to the them via security settings for the separate repository group. In many cases multiple target groups can be useful for different user groups to have access.

### Close Repository Notification Settings

After a developer has deployed a set of related release artifacts, a staging repository is *closed*. This means that no further artifacts can be deployed to the same staging repository. A repository would be closed when a developer is satisfied that a collection of staged artifacts is ready to be certified by a manager or a quality assurance resource. In this setting, it is possible to define email addresses and roles that should be notified of a staging repository being closed. A notification email will be sent to all specified email addresses, as well as all Nexus users in the specified roles, informing them that a staging repository has been closed. It is also possible to select that the creator of the staging repository receives this notification.

### Promote Repository Notification Settings

Once a closed staging repository has been certified by whomever is responsible for testing and checking a staged release, it can then be promoted (published) or dropped (discarded). In this setting, it is possible to define the email addresses and Nexus security roles that should be notified of a staging repository being promoted. A notification email will be sent to all specified email addresses, as well as all Nexus users in the specified roles, informing them that a staging repository has been promoted. It is also possible to select that the creator of the staging repository receives this notification.

### Drop Repository Notification Settings

In this setting, it is possible to define email addresses and roles notified when a staging repository is being dropped. A notification email will be sent to all specified email addresses, as well as all Nexus users in the specified roles, informing them that a staging repository has been dropped. It is also possible to select that the creator of the staging repository receives this notification.

### Close Repository Staging Rulesets

This defines the rulesets applied to a staging repository before it can be closed. If the staging repository does not pass the rules defined in the specified rulesets, you will be unable to close it. For more information about rulesets, see Section [11.5](#).

---

## Promote Repository Staging Rulesets

This defines the rulesets applied to a staging repository on promotion. If the staging repository does not pass the rules defined in the specified rulesets, the promotion will fail with an error message supplied by the failing rule. For more information about rulesets, see Section 11.5.

### 11.2.3 Configuring Build Promotion Profiles

A build promotion profile is used when you need to add an additional step between initial staging and final release. To add a new *Build Promotion* profile, open the *Staging Profiles* link from the *Nexus* menu and click on *Add...* to display the drop-down menu shown in Figure 11.6. Select *Build Promotion Profile* from this drop-down to create a new build promotion profile.

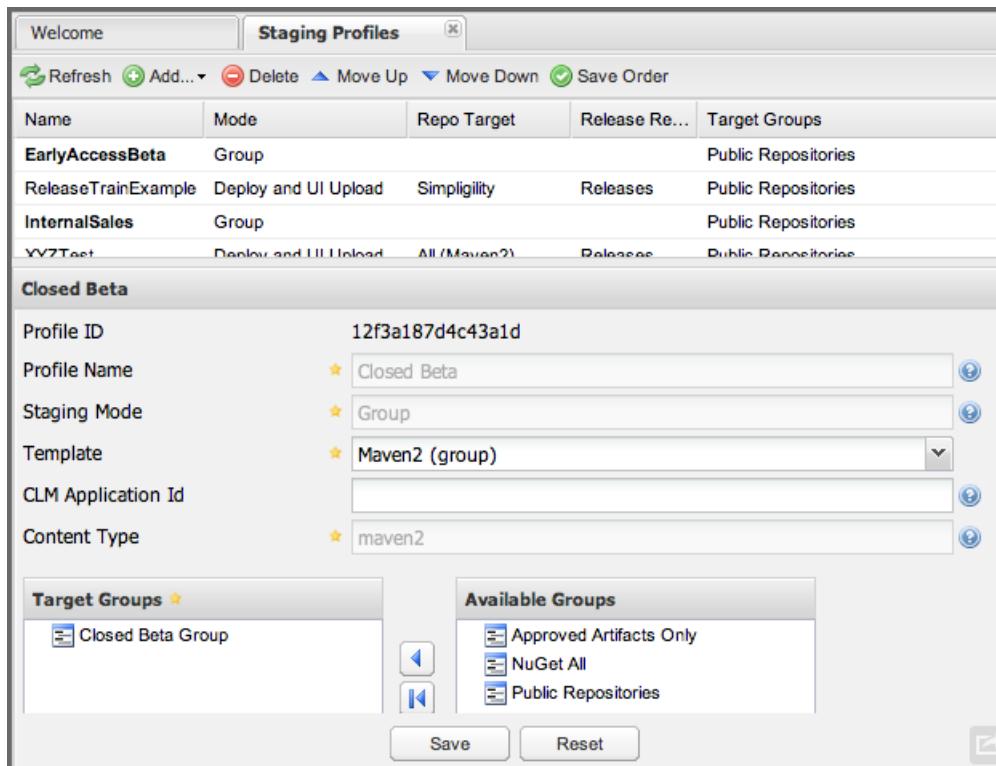


Figure 11.6: Multilevel Staging and Build Promotion

After creating a new build promotion profile, you will see the form shown in Figure 11.7. This form

contains the following configuration fields:

### Profile Name

The name for the build promotion profile displayed in the promotion dialog and associated with repositories created from this promotion profile.

### Template

The template for repositories generated by this build promotion profile. The default value for this field is Maven2 (group).

### Target Groups

The *Target Groups* field is the most important configuration field for a build promotion profile, as it controls the group through which promoted artifacts are made available. Artifacts can be made available through one or more groups.

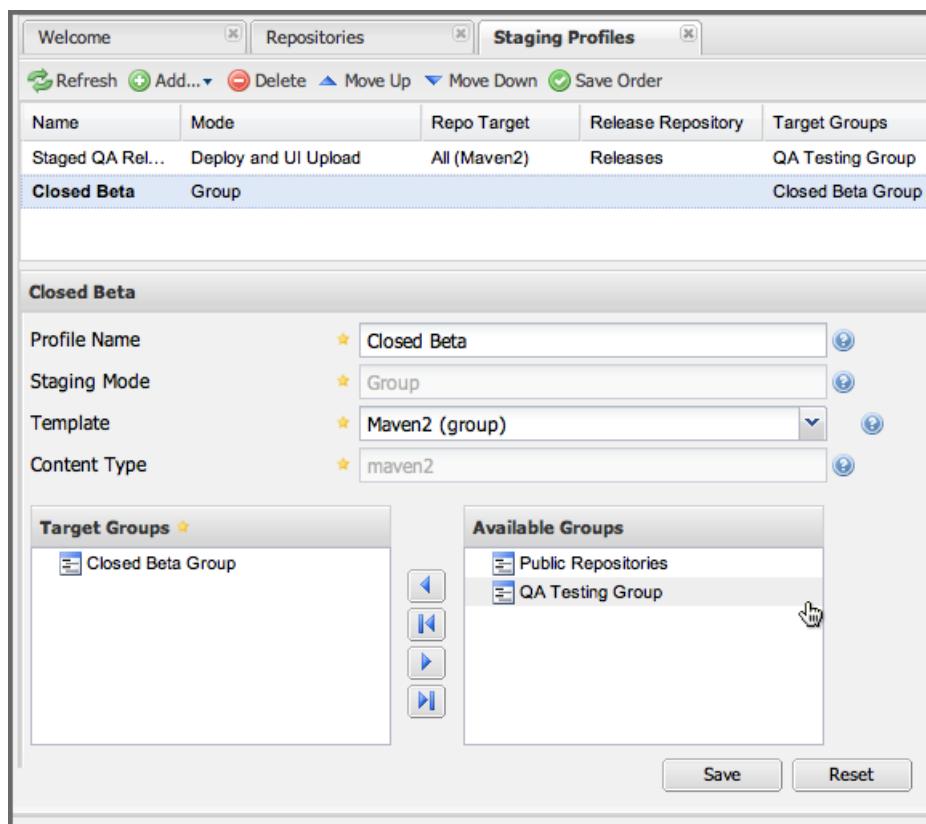
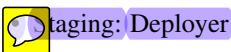


Figure 11.7: Configuring a Build Promotion Profile

### 11.2.4 Staging Related Security Setup

Staging Suite is controlled by three roles:



- Staging: Promoter
- Staging: Repositories

These roles are available as general *admin* roles that apply to all staging profiles with the respective access. When you create a new staging profile, Nexus will create new roles that grant permissions specific to that staging profile. If you created the staging profile named `Test`, Nexus created the three new and profile-specific roles:

#### Staging: Repositories (Test)

This role grants a user read and view access to the staging repositories created by the *Test* staging profile.

#### Staging: Deployer (Test)

This role grants all of the privileges from the Staging: Repositories role and, in addition, grants the user permission to deploy artifacts, close and drop any staging repository created by the *Test* staging profile.

#### Staging: Promoter (Test)

This role grants the user the right to promote staging repositories created by the *Test* staging profile.

To perform a staged deployment, the user deploying the artifact must have the *Staging: Deployer (admin)* role or the *Staging: Deployer* role for a specific staging profile.

To configure the deployment user with the appropriate staging role, click on Users under the *Security* menu in the *Nexus* menu. Once you see the *Users* panel, click on the deployment user to edit this user's roles. Click on the *Add* button in the *Role Management* section of the *Config* tab visible in Figure 11.8 for the user to be able to add new roles to the user.

The screenshot shows the Nexus User Management interface. At the top, there is a header bar with a 'Users' tab, a refresh button, an 'Add...' button, a 'Delete' button, a dropdown menu for 'All Configured Users', and a search bar. Below the header is a table with columns: User ID, Realm, First Name, Last Name, Email, and Roles. Three users are listed: 'admin' (First Name: Administrator), 'deployment' (First Name: Deployment), and 'anonymous' (First Name: Nexus). The 'deployment' user is selected, indicated by a blue border around its row.

Below the table, the user's configuration details are shown in a form:

User ID	deployment
First Name	Deployment
Last Name	User
Email	changeme1@yourcompany.com
Status	Active

Underneath the form is a 'Role Management' section with a tree view:

- Nexus Deployment Role
- Repo: All Repositories (Full Control)
- Staging: Deployer (admin)

At the bottom of the configuration panel are 'Save' and 'Reset' buttons.

Figure 11.8: Adding a Role to a User

Use the *Filter* section with the keyword `Staging` and press the *Apply Filter* button to see all available staging-related roles as displayed in Figure 11.8.

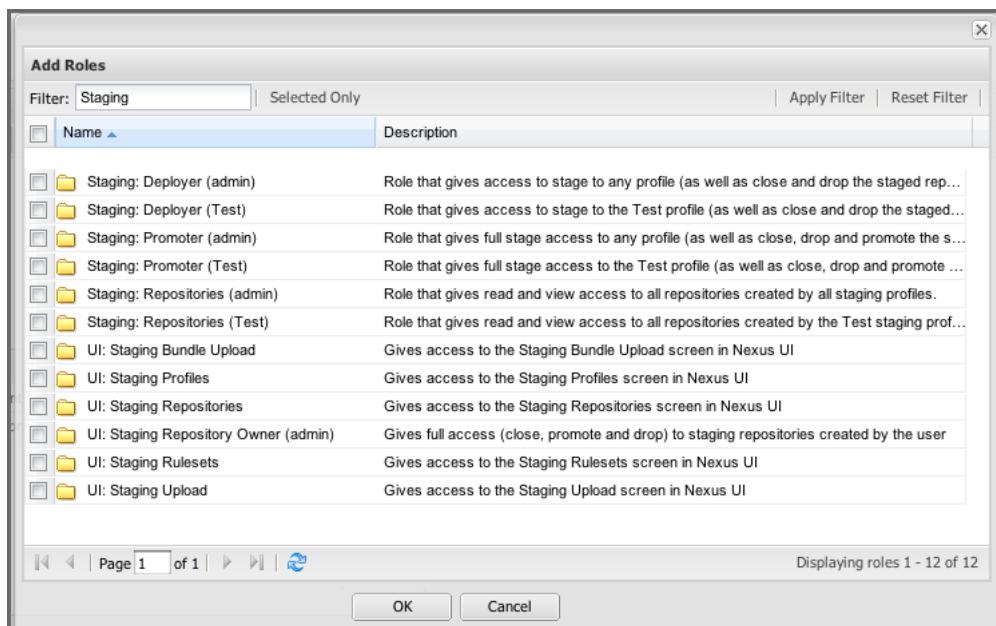


Figure 11.9: Available Roles for Staging with a Test Staging Profile

You should see the "Staging: Deployer (admin)" role listed as well as the *Test* staging profile-specific role, the promoter and repositories ones for *admin* and *Test* and a few staging user interface related roles. These roles are required if interaction with the staging suite in the Nexus user interface is desired and allow you to control the details about this access. If you need to add a specific permission to activate a single Staging Profile, you would select that specific role.

Once the deployment user has the "Staging: Deployer (admin)" role, you can then use this user to deploy to the staging URL and trigger any staging profile. Without this permission, the deployment user would not be able to publish a staged artifact.

In a similar fashion, you can assign the promoter role to users.

In addition to the roles created a number of specific privileges is available to further customize the access to the staging suite:

### **Staging Profiles**

Allows control of create, read, delete and update operations on staging profiles.

**Staging Repository: test-001**

There are separate privileges for each staging repository allowing create, read, update and delete operations are generated automatically.

**Staging: All Profiles, Owner All Profiles and Profile xyz**

These staging profile specific-privileges can be granted for drop, promote, read and finish operations.

**Staging: Rule Set and Staging: Rule Types**

Control access to staging rules and rule types

**Staging: Upload**

controls access to the manual staging upload user interface

**Staging: Repositories, Promote Repository, Profile Ordering, Close Staging and others**

A number of application user interface-specific privileges allow fine-grained control over access in the user interface.

### 11.2.5 Using Repository Targets for Staging

The Staging Suite intercepts deployments to Nexus using Repository Targets as documented in Section [6.14](#) when using implicit matching as a profile selection strategy, based on the artifacts path in the repository.

For example, if you wanted to intercept all deployments to the com.sonatype.sample groupId, you would create a repository target with a pattern with a regular expression of `^/com/sonatype/sample/.*` and use that repository target in your Staging Profile configuration.

## 11.3 Configuring Your Project for Deployment

Once Nexus is configured to receive artifacts in the staging suite as documented in Section [11.2](#), you will have to update your project build configuration to deploy to the staging suite.

The preferred way to do this is to take advantage of the features provided by the Nexus staging Maven plugin or the Nexus staging Ant tasks as documented in Section [11.3.1](#) and Section [11.3.2](#).

If you need to continue to use the Maven deploy plugin, you can read about using it with the Nexus staging suite in Section [11.3.3](#).

With all tools you can use the manual upload of your artifacts documented in Section [11.3.5](#).

### 11.3.1 Deployment with the Nexus Staging Maven Plugin

The Nexus staging Maven plugin is a Nexus specific and more powerful replacement for the Maven deploy Plugin with a number of features specifically geared towards usage with the Nexus staging suite. The simplest usage can be configured by adding it to the project build plugins section as an extension:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.sonatype.plugins</groupId>
      <artifactId>nexus-staging-maven-plugin</artifactId>
      <version>1.6.3</version>
      <extensions>true</extensions>
      <configuration>
        <serverId>nexus</serverId>
        <nexusUrl>http://localhost:8081/nexus/</nexusUrl>
      </configuration>
    </plugin>
```

---

#### Note

It is important to use a version of the plugin that is compatible with your Nexus server. Version 1.2 is compatible with Nexus 2.3, Version 1.4.4 is compatible with Nexus 2.4, Version 1.4.8 is compatible with Nexus 2.5 and 2.6. 1.5 and 1.6.x can be used for Nexus 2.7 to 2.10. The latest version of the plugin available is always compatible with the latest available version of Nexus. Try to use the newest possible plugin version to take advantage of any available improvements.

---

Following Maven best practices, the version should be pulled out into a `pluginManagement` section in a company POM or parent POM.

This configuration works only in Maven 3 and automatically replaces the `deploy` goal invocation of the Maven deploy plugin in the `deploy` Maven lifecycle phase with the `deploy` goal invocation of the Nexus staging Maven plugin.

The minimal required configuration parameters for the Nexus staging Maven plugin are:

---

#### serverId

The `id` of the `server` element in `settings.xml` from which the user credentials for accessing Nexus should be retrieved.

#### **nexusUrl**

The base URL at which the Nexus server to be used for staging is available.

With this configuration the Nexus staging Maven plugin will stage the artifacts locally and connect to Nexus. Nexus will try to determine the appropriate staging profile by matching the artifact path with any repository targets configured with staging profiles with an activated implicit profile selection strategy. If an appropriate staging profile is found, a staging repository is created on the fly and the artifacts are deployed into it. If no profile is found, the upload will fail.

To successfully deploy to your Nexus instance, you will need to update your Maven Settings with the credentials for the deployment user. These credentials are stored in the Maven Settings file in `~/.m2/settings.xml`.

To add these credentials, add the following element to the servers element in your `~/.m2/settings.xml` file as shown in [Listing deployment credentials in Maven Settings](#).

#### **Listing deployment credentials in Maven Settings**

```
<settings>
  ...
  <servers>
    ...
    <server>
      <id>nexus</id>
      <username>deployment</username>
      <password>deployment123</password>
    </server>
  </servers>
  ...
</settings>
```

Note that the server identifier listed in [Listing deployment credentials in Maven Settings](#) should match the `serverId` parameter you are passing to the Nexus Staging Maven Plugin and in the example contains the default password for the Nexus deployment user - `deployment123`. You should change this password to match the deployment password for your Nexus installation.

If more control is desired over when the plugins deploy goal is activated or if Maven 2 is used, you have to explicitly deactivate the Maven Deploy Plugin and replace the Maven Deploy Plugin invocation with the Nexus Staging Maven Plugin like visible in in [Usage of Nexus Staging Maven Plugin for Maven 2](#).

## Usage of Nexus Staging Maven Plugin for Maven 2

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-deploy-plugin</artifactId>
      <configuration>
        <skip>true</skip>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.sonatype.plugins</groupId>
      <artifactId>nexus-staging-maven-plugin</artifactId>
      <executions>
        <execution>
          <id>default-deploy</id>
          <phase>deploy</phase>
          <goals>
            <goal>deploy</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <serverId>nexus</serverId>
        <nexusUrl>http://localhost:8081/nexus/</nexusUrl>
        <!-- explicit matching using the staging profile id -->
        <stagingProfileId>129341e09f2ee275</stagingProfileId>
      </configuration>
    </plugin>
  ...

```

The implicit matching relies on the setup of repository targets as well as the correct order of staging profiles and is therefore an error prone approach when many staging profiles are in use.

The preferred way to work in this scenario is to change the profile selection strategy on all staging profiles to explicit only and pass the staging profile ID to the Nexus staging Maven plugin using the `stagingProfileId` configuration parameter as documented above. A full example `pom.xml` for deployment of snapshot as well as release builds with the Nexus staging Maven plugin using explicit matching for the staging profile and locally staged builds and atomic uploads is available in [Full example pom.xml for Nexus Staging Maven Plugin usage](#).

### Full example `pom.xml` for Nexus Staging Maven Plugin usage

```
<project>
  <modelVersion>4.0.0</modelVersion>
```

```
<groupId>com.sonatype.training.nxs301</groupId>
<artifactId>explicit-staging-example</artifactId>
<version>1.0.0</version>

<distributionManagement>
    <snapshotRepository>
        <id>nexus-snapshots</id>
        <url>http://localhost:8081/nexus/content/repositories/snapshots</url>
    </snapshotRepository>
</distributionManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.sonatype.plugins</groupId>
            <artifactId>nexus-staging-maven-plugin</artifactId>
            <version>1.6.3</version>
            <extensions>true</extensions>
            <configuration>
                <serverId>nexus-releases</serverId>
                <nexusUrl>http://localhost:8081/nexus/</nexusUrl>
                <!-- update this to the correct id! -->
                <stagingProfileId>1296f79efe04a4d0</stagingProfileId>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

In order to deploy project artifacts to Nexus with the above setup you would invoke a build with `mvn clean deploy`.

The build will locally stage the artifacts for deployment in `target/nexus-staging` on the console and create a closed staging repository in Nexus holding the build artifacts. This execution of the `deploy` goal of the Nexus staging Maven plugin performs the following actions:

- Artifacts are staged locally.
- A staging profile is selected either implicitly or explicitly.
- A staging repository is either created on the fly, if needed, or just selected.
- An atomic upload to the staging repository is performed.
- The staging repository is closed (or dropped if upload fails).

The log of a successful deployment would look similar to this:

```
[INFO] --- nexus-staging-maven-plugin:1.1.1:deploy (injected-nexus-deploy) ←
@ staging-example ---
[INFO] Using server credentials with ID="nexus-releases" from Maven ←
settings.
[INFO] Preparing staging against Nexus on URL http://localhost:8081/nexus/
[INFO] * Remote Nexus reported itself as version 2.2.1 and edition " ←
Professional"
[INFO] * Using staging profile ID "12a1656609231352" (matched by Nexus).
[INFO] Staging locally (stagingDirectory=
"/Users/manfred/dev/explicit-staging-example/target/nexus-staging/12 ←
a1656609231352")...
Uploading: file: ... explicit-staging-example-1.0.0.jar
Uploaded: file: ... explicit-staging-example-1.0.0.jar (4 KB at 1051.1 KB/ ←
sec)
Uploading: file: ... explicit-staging-example-1.0.0.pom
Uploaded: file: ... explicit-staging-example-1.0.0.pom (4 KB at 656.2 KB/ ←
sec)
Downloading: file: ...maven-metadata.xml
Uploading: file: ...maven-metadata.xml
Uploaded: file: ... maven-metadata.xml (322 B at 157.2 KB/sec)
[INFO] Staging remotely...
[INFO] Uploading locally staged directory: 12a1656609231352
[INFO] Performing staging against Nexus on URL http://localhost:8081/nexus ←
/
[INFO] * Remote Nexus reported itself as version 2.2.1 and edition " ←
Professional"
[INFO] * Created staging repository with ID "test-002",
applied tags: {javaVersion=1.6.0_37, localUsername=manfred}
[INFO] * Uploading locally staged artifacts to:
http://localhost:8081/nexus/service/local/staging/deployByRepositoryId/ ←
test-002
[INFO] * Upload of locally staged artifacts done.
[INFO] * Closing staging repository with ID "test-002".
[INFO] Finished staging against Nexus with success.
```

Failures are accompanied by error reports that reveal further details:

```
[ERROR] Error while trying to close staging repository with ID "test-003".
[ERROR]
[ERROR] Nexus Staging Rules Failure Report
[ERROR] =====
[ERROR]
[ERROR] Repository "Test-003 (u:admin, a:127.0.0.1)" (id=n/a) failures
[ERROR]   Rule "RepositoryWritePolicy" failures
[ERROR]     * Artifact updating: Repository ='releases:Releases' does
not allow updating
```

```
artifact='/com/sonatype/training/nexus/explicit-staging-example/t1.0.0/ ←
    staging-example-1.0.0.jar'
[ERROR]      * Artifact updating: Repository ='releases:Releases' does
not allow updating
artifact='/com/sonatype/training/nexus/explicit-staging-example/1.0.0/ ←
    staging-example-1.0.0.pom'
[ERROR]
[ERROR]
```

If the configuration parameter `skipStagingRepositoryClose` set to `true` is passed to the plugin execution, the remote staging repository will not be closed.

Instead of Nexus creating a staging repository based on the implicit or explicit staging profile selection, you can explicitly configure the staging repository to use by providing the staging repository name as value of the `stagingRepositoryId` configuration property via the plugin configuration or command line invocation.

The identifier of a staging repository can be determined by looking at the name column in the list of staging repositories. The name column used the capitalized ID and adds the username and address the staging was deployed from in brackets. For example a name could be `Test-003` (`u: admin, a: 127.0.0.1`). The ID of this staging repository is `test-003`.

Together with skipping the closing of the repository using `skipStagingRepositoryClose`, it is possible to get multiple builds to deploy to the same staging repository and, therefore, have a number of artifacts go through the staging workflow together. An alternative to this approach would be to create an aggregating project that assembles all artifacts together, e.g., in an assembly and then use this project for staging.

Finally to override all staging, you can define the full repository URL to deploy to with the `deployUrl` configuration parameter. For example, see below:

```
http://localhost:8081/nexus/content/repositories/releases/
```

This would cause any staging to be skipped and a straight upload of the artifacts to the repository to occur.

As part of the configuration section for the plugin you can define tags with arbitrary key and value names. For example, you could create a tag with key `localUsername` and a value of the current user picked up from the `USER` environment variable:

```
...
<configuration>
...
...
```

```
<tags>
  <localUsername>${env.USER}</localUsername>
  <javaVersion>${java.version}</javaVersion>
</tags>
...
```

Once artifacts are released these tags are transformed into attributes stored along the artifacts in the release repository and can be accessed via the REST interface and, therefore, any plugin and user interface integration.

In addition to the above documented configuration options that determine the behaviour of the Nexus Staging Maven Plugin, further configuration can be provided with the following parameters:

#### **altStagingDirectory**

Defaulting to target/nexus-staging you can set the property to set a different folder for the local staging.

#### **autoReleaseAfterClose**

If you set this flag to `true`, the staging repository will be closed and, following a successful validation of all staging rules including potential Sonatype CLM based validation, released. By default this property is set to `false`. Changing it to `true` can be a useful setup for continuous integration server based releases.

#### **description**

Allows you to provide a description for the staging repository action (like close or drop) carried out as part of the plugin execution. The description will then be used in any notification just like a description provided in the user interface.

#### **keepStagingRepositoryOnFailure**

Setting this flag to `true` will cause the plugin to skip any clean up operations like dropping a staging repository for failed uploads, by default these clean up operations occur.

#### **keepStagingRepositoryOnCloseRuleFailure**

With the default setting of `false`, the Nexus staging Maven plugin will drop the created staging repository if any staging rule violation occurs. If this flag is set to `true`, it will not drop the staging repository. This allows you to inspect the deployed components in order to figure out why a rule failed causing the staging failure.

#### **skipStagingRepositoryClose**

Set this to `true` to turn off the automatic closing of a staging repository after deployment.

#### **skipNexusStagingDeployMojo**

Set to `false` by default, this flag will cause to skip any execution of the `deploy` goal of the plugin when set to `true` similar to `maven.deploy.skip`

---

**skipStaging**

Set to `false` by default this flag will cause to skip any execution of the plugin when set to true.

**skipRemoteStaging**

If this flag is set to `true` any step related to remote staging will be skipped and only local staging will be performed. The default setting is `false`.

**skipLocalStaging**

When `true`, bypass all staging specific features. Remote deploys happen inline at deploy phase of each module, not at build end. The deployment repository is "sourced" from pom.xml `<distributionManagement>`. Which distribution repository is used depends on the project having a release or snapshot version. Essentially this option makes the staging plugin execution act like the default `maven-deploy-plugin`. The default setting is `false`.

**stagingProgressTimeoutMinutes**

Defaulting to 5 minutes, this configuration allows you to set the timeout for staging operations. Changes are most often required for complex staging operations involving custom staging rules or Sonatype CLM integration.

**stagingProgressPauseDurationSeconds**

The default of 3 seconds can be changed if larger pauses between progress polls for staging operations are desired.

With `skipRemoteStaging` set to `true`, only the local staging happens. This local staging can then be picked up for the remote staging and closing by running the `deploy-staged` goal of the plugin explicitly like this

```
mvn nexus-staging:deploy-staged
```

Besides the default `deploy` goal the Nexus staging Maven plugin supports a number of additional goals. By configuring executions of the goals as part of your POM or manually invoking them further automation of a staged release process can be achieved.

**deploy-staged**

Perform full staging deployment workflow for a locally staged project, e.g., with the artifacts in `target/nexus-staging`.

**deploy-staged-repository**

Perform an upload of a repository from the local filesystem to a staging repository.

**close**

Close the staging repository for current context.

**drop**

Drop the staging repository for current context.

---

**release**

Release the staging repository for current context.

**promote**

Promote the staging repository for the current context.

Closing, dropping, and releasing the staging repository using the goals relies on content of a local staging folder .

Promoting additionally needs the build promotion profile name passed in via the `buildPromotionProfileId` configuration parameter.

The `deploy-staged-repository` goal can be used to stage a repository. Typically, a local repository is created with an invocation of the deploy similar to

```
mvn deploy -DaltDeploymentRepository=local::default::file://path
```

To deploy this file system repository with the goal, you have to provide the path to this repository with the `repositoryDirectory` parameter as well as `nexusUrl`, `serverId` and `stagingProfileId`. Optionally you can configure the repository to stage into with `stagingRepositoryId`. This aggregated command is then be run outside any specific Maven project.

While the above goals need the context of a project with configuration for the Nexus Staging Plugin in the POM file, it is possible to execute staging repository-related tasks without a project as well. The Nexus Staging Maven Plugin offers remote-control goals to control staging in Nexus:

**rc-close**

Close a specified staging repository.

**rc-drop**

Drop a specified staging repository.

**rc-release**

Release a specified staging repository.

**rc-promote**

Promote a specified staging repository.

**rc-list**

List all staging repositories.

When invoking these goals outside a project context, you need to have the Nexus staging Maven plugin `groupId` specified as a `pluginGroup` in your `settings.xml`:

---

```
<pluginGroups>
  <pluginGroup>org.sonatype.plugins</pluginGroup>
</pluginGroups>
```

In addition, you need to specify all parameters on the command line as properties passed in via `-Dkey=value`.

At a minimum the required parameters `serverId` and `nexusUrl` have to be specified:

```
mvn nexus-staging:rc-close -DserverId=nexus -DnexusUrl=http://localhost ↵
:8081/nexus
```

Depending on the goal you will have to configure the staging repositories you want to close, drop or release with

```
-DstagingRepositoryId=repo-001,repo-002
```

and you can also supply a description like this

```
-Ddescription="Dropping since QA of issue 123 failed"
```

For promoting, you need to add the required parameter that specifies the build promotion profile identifier:

```
-DbuildPromotionProfileId=12a25eabf8c8b3f2
```

A successful remote control drop would be logged in the command line similar to this

```
-- nexus-staging-maven-plugin:1.2:rc-drop (default-cli) @ standalone-pom --
[INFO] Connecting to Nexus...
[INFO] Using server credentials with ID="nexus-releases" from Maven   ↵
      settings.
[INFO] RC-Dropping staging repository with IDs=[test-003]
[INFO]   ↵
-----
[INFO] BUILD SUCCESS
[INFO]   ↵
----- ↵
```

An example usage of the `rc-list` goal with output is

---

```
$mvn nexus-staging:rc-list -DnexusUrl=http://localhost:8081/nexus  
-DserverId=nexus  
...  
[INFO] --- nexus-staging-maven-plugin:1.5.1:rc-list (default-cli) @ ←  
standalone-pom ---  
[INFO] Connecting to Nexus...  
[INFO] Using server credentials with ID="nexus" from Maven settings.  
[INFO] Getting list of available staging repositories...  
[INFO]  
[INFO] ID State Description  
[INFO] example_release_profile-1000 OPEN Implicitly created (auto  
staging).  
...
```

**Warning**

The Nexus Maven Plugin in versions earlier than 2.1.0 had goals to work with staging repositories. These goals have been deprecated in favour of the remote control goals of the Nexus Staging Maven Plugin.

### 11.3.2 Deployment with the Nexus Staging Ant Tasks

The Nexus staging Ant tasks provide equivalent features to the Nexus staging Maven plugin for Apache Ant users covering all use cases for interacting with the Nexus staging suite.

Historically Ant builds typically have components that are required for the build, statically managed in the version control system or even outside the project workspace altogether. More modern Ant builds use Apache Ivy or Eclipse Aether for resolving dependencies dynamically as well as deployment build outputs to a repository manager. Examples projects setups using Ivy as well as Aether can be found in the [Nexus book examples project](#). This project includes examples for integration with the Nexus staging Ant tasks.

To use the Ant tasks in your Ant build file, download the complete JAR with the included dependencies from the Central Repository. Simply search for *nexus-staging-ant-tasks* and download the JAR file with the *uber* classifier e.g., *nexus-staging-ant-tasks-1.6-2-uber.jar*.

After downloading, put the JAR file somewhere in your project or in your system so you can add it to the classpath in your build file with a task definition. In the following example, the JAR file is placed in a *tasks* folder within the project.

```
<taskdef uri="antlib:org.sonatype.nexus.ant.staging"
         resource="org/sonatype/nexus/ant/staging/antlib.xml">
  <classpath>
    <fileset dir="tasks" includes="nexus-staging-ant-tasks-*uber.jar" />
  </classpath>
</taskdef>
```

To enable the tasks in your build file using a shortcut for the namespace, e.g., `staging`, you have to add it to the project node:

```
<project xmlns:staging="antlib:org.sonatype.nexus.ant.staging" ...>
```

The deployment-related information for your project is captured in a `nexusStagingInfo` section in your build file that contains all the necessary configuration.

```
<staging:nexusStagingInfo id="target-nexus"
  stagingDirectory="target/local-staging">
  <staging:projectInfo groupId="org.sonatype.nexus.ant"
    artifactId="nexus-staging-ant-tasks"
    version="1.0" />
  <staging:connectionInfo
    baseUrl="http://localhost:8081/nexus">
    <staging:authentication
      username="deployment"
      password="deployment123" />
  </staging:connectionInfo>
</staging:nexusStagingInfo>
```

### **nexusStagingInfo:id**

The identifier that allows you to reference the staging information in the Ant build file.

### **stagingInfo:stagingDirectory**

The local staging directory, a place where local staging will happen. Ensure that this directory is cleaned up by a `clean` task or alike, if any.

### **projectInfo**

The project information targetting a staging profile. This can be done explicitly with the `stagingProfileId` or implicitly with `groupId`, `artifactId` and `version`. `stagingRepositoryId` can also be part of `projectInfo` identifying a staging repository for interaction.

### **connectionInfo:baseUrl**

The base URL of the Nexus server you want to deploy to and interact with.

If necessary the `connectionInfo` can have a nested `proxy` section

```
<staging:proxy  
    host="proxy.mycorp.com"  
    port="8080">  
    <staging:authentication  
        username="proxyUser"  
        password="proxySecret" />  
</staging:proxy>
```

With the above setup you are ready to add a `deploy` target to your build file that stages the artifacts locally as well as remotely and closes the staging repository.

```
<target name="deploy" description="Deploy: Local and Remote Staging">  
    <staging:stageLocally>  
        <staging:nexusStagingInfo  
            refid="target-nexus" />  
        <fileset dir="target/local-repo"  
            includes="**/*.*" />  
    </staging:stageLocally>  
  
    <staging:stageRemotely>  
        <staging:nexusStagingInfo  
            refid="target-nexus" />  
    </staging:stageRemotely>  
</target>
```

The folder `target/local-repo` has to contain the components in a directory structure resembling the Maven repository format using the `groupId`, `artifactId` and `version` coordinates of the component mapped to directory names. It will be merged into the target release repository, when the staging repository is released. An example on how to create such a structure in Ant can be found in the staging example for Apache Ivy and Eclipse Aether in the [Nexus book examples project](#).

Similarly, you can create a target that releases the staged artifacts by adding the `releaseStagingRepository` task to the end of the target:

```
<staging:releaseStagingRepository>  
    <staging:nexusStagingInfo  
        refid="target-nexus" />  
</staging:releaseStagingRepository>
```

The `stageLocally` task takes a `fileset` as configuration. The `stageRemotely` task has additional configuration options.

**keepStagingRepositoryOnFailure**

Set to `true` this causes the remote staging repository to be kept rather than deleted in case of a failed upload. Default setting is `false`

**skipStagingRepositoryClose**

By default a staging repository is automatically closed, setting this parameter to `true` will cause the staging repository to remain open.

In addition to the tasks for local and remote staging, the Nexus staging Ant tasks include tasks for closing, dropping, releasing and promoting a staging repository:

- `closeStagingRepository`
- `dropStagingRepository`
- `releaseStagingRepository`
- `promoteStagingRepository`

All these tasks take the context information from the local staging directory or from the optional parameter `stagingRepositoryId`. The task to promote a repository has the additional, mandatory attribute `buildPromotionProfileId` to specify the build promotion profile to promote.

The timing of the task operation can be affected by the following configuration parameters:

**stagingProgressTimeoutMinutes**

Defaulting to 5 minutes, this configuration allows you to set the timeout for staging operations. Changes are most often required for complex staging operations involving custom staging rules or Sonatype CLM integration.

**stagingProgressPauseDurationSeconds**

The default of 3 seconds can be changed if larger pauses between progress polls for staging operations are desired.

### 11.3.3 Deployment with the Maven Deploy Plugin

When using the Maven deploy plugin with the Nexus staging suite, you rely on implicit matching of the artifacts against a staging profile based on a repository target definition.

---

To deploy a staged release, a developer needs to deploy to the staging URL. To configure a project to deploy to the staging URL, add the `distributionManagement` element to your project's POM.

### Listing the Staging URL in `distributionManagement`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
...
<distributionManagement>
  <repository>
    <id>nexus</id>
    <name>Nexus Staging Repo</name>
    <url>http://localhost:8081/nexus/service/local/staging/deploy/maven2 ←
      /</url>
  </repository>
</distributionManagement>
...
</project>
```

This configuration element, `distributionManagement`, defines the repository to which our deployment will be made. It references the staging suite's URL: <http://localhost:8081/nexus/service/local/-staging/deploy/maven2>

This URL acts as a virtual repository to be published to. If an artifact being published matches one of the repository targets in a staging profile, that staging profile is *activated* and a temporary staging repository is created.

Once the sample project's `distributionManagement` has been set to point at the Nexus staging URL and your deployment credentials are updated in your `~/.m2/settings.xml` file, you can deploy to the staging URL. To do this, run `mvn deploy`:

```
$ mvn deploy
[INFO] Scanning for projects...
[INFO]   ←
-----
[INFO] Building staging-test
[INFO]   task-segment: [deploy]
[INFO]   ←
-----
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
```

```
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory: /private/tmp/staging-test/target/ ↵
    surefire-reports

...
[INFO] [jar:jar]
[INFO] [install:install]
[INFO] Installing /private/tmp/staging-test/target/staging-test-1.0.jar to ↵
    \
~/.m2/repository/com/sonatype/sample/staging-test/1.0/staging-test-1.0.jar
[INFO] [deploy:deploy]
altDeploymentRepository = null
Uploading: http://localhost:8081/nexus/service/local/staging/deploy/maven2 ↵
    /
com/sonatype/sample/staging-test/1.0/staging-test-1.0.jar
2K uploaded
[INFO] Uploading project information for staging-test 1.0
[INFO] Retrieving previous metadata from nexus
[INFO] repository metadata for: 'artifact com.sonatype.sample:staging-test ↵
    ,
could not be found on repository: nexus, so will be created
[INFO] Uploading repository metadata for: 'artifact com.sonatype.sample: ↵
    staging-test'
[INFO] ←
----- ←

[INFO] BUILD SUCCESSFUL
```

If the staging suite is configured correctly, any deployment to the staging URL matching in a repository target configured for a staging profile should be intercepted by the staging suite and placed in a temporary staging repository. Deployment with the Maven deploy plugin will not automatically close the staging repository. Closing the staging repository has to be done with the Nexus user interface or the Nexus staging Maven plugin. Once this repository has been closed, it will be made available in the target group you selected when you configured the staging profile.

#### 11.3.4 Deployment and Staging with Gradle

The Gradle build system can be used to deploy components to Nexus with the Gradle Maven plugin. The Nexus Staging Ant Tasks can be used in Gradle allowing full integration of the staging suite features in a Gradle build.

An example project showcasing this integration is available in the [Nexus book examples project](#).

### 11.3.5 Manually Uploading a Staged Deployment in Nexus

You can also upload a staged deployment via the Nexus interface. To upload a staged deployment, select *Staging Upload* from the *Nexus* menu. Clicking *Staging Upload* will show the panel shown in Figure 11.10.

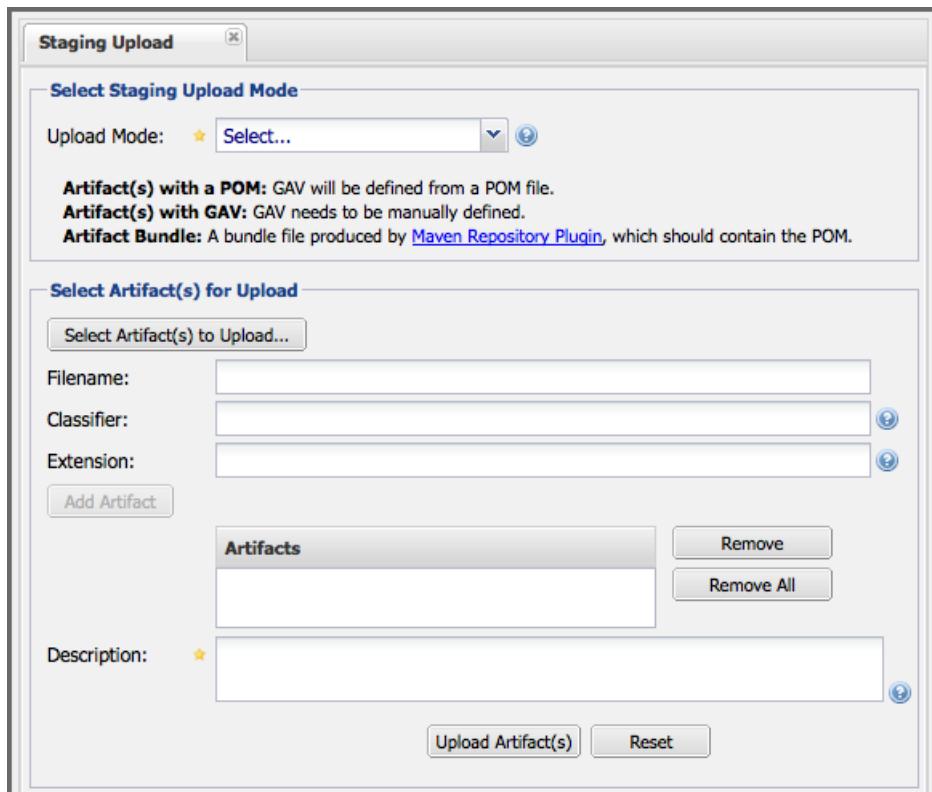


Figure 11.10: Uploading a Staged Deployment in Nexus

To upload an artifact, click on *Select Artifact(s) for Upload...* and select an artifacts from the filesystem to upload. Once you have selected an artifact, you can modify the classifier and the extension before clicking on the *Add Artifact* button. Repeat this process to upload multiple artifacts for the same *Group*, *Artifact* and *Version* (GAV) coordinates like a JAR, the POM and maybe a sources and javadoc JAR in

addition. Once you have added all the artifacts, you can then configure the source of the Group, Artifact, Version (GAV) parameters.

If the artifact you are uploading is a JAR file that was created by Maven, it will already have POM information embedded in it, but if you are uploading a JAR from a vendor you will likely need to set the Group Identifier, Artifact Identifier, and Version manually. To do this, select GAV Parameters from the GAV Definition drop-down at the top of this form. Selecting *GAV Parameters* will expose a set of form fields that will let you set the *Group*, *Artifact*, *Version*, and *Packaging* of the artifacts being uploaded. If you would prefer to set the Group, Artifact, and Version from a POM file that was associated with the uploaded artifact, select From POM in the GAV Definition drop-down. Selecting From POM in this drop-down will expose a button labeled *Select POM to Upload*. Once a POM file has been selected for upload, the name of the POM file will be displayed in the form field below this button.

The *Staging Upload* panel supports multiple artifacts with the same *Group*, *Artifact*, and *Version* identifiers. For example, if you need to upload multiple artifacts with different classifiers, you may do so by clicking on *Select Artifact(s) for Upload* and *Add Artifact* multiple times. This interface also accepts an *Artifact Bundle* which is a JAR that contains more than one artifact, which is documented in more detail in Section 11.7.

Once a staging artifact upload has been completely configured, click on *Upload Artifact(s)* button to begin the upload process. Nexus will upload the artifacts to the Staging URL which will trigger any staging profiles that are activated by the upload by explicitly matching using the repository targets configured with the staging profiles. If a staging profile is activated, a new staging repository will be created and can be managed using the procedures outlined in Section 11.4.

## 11.4 Managing Staging Repositories in Nexus

With a staging profile configured and a deployment completed as outlined in Section 11.2 and Section 11.3, you will have an automatically generated staging repository. A list of all staging repositories can be accessed by selecting the *Staging Repositories* item in the *Build Promotion* menu and is displayed in Figure 11.11.

The screenshot shows a software interface titled "Staging Repositories". At the top, there is a toolbar with icons for Refresh, Close, Promote, Release, Drop, and a "Filter by profile" dropdown. Below the toolbar is a table with five columns: Repository, Profile, Status, Updated, and Description. A single row is visible, representing a repository named "nxs301-001" with profile "NXS301", status "closed", updated on "2013-Mar-27 10:10:14", and description "Implicitly created (aut)". A checkbox is checked next to the repository name.

	Repository	Profile	Status	Updated	Description
<input checked="" type="checkbox"/>	nxs301-001	NXS301	closed	2013-Mar-27 10:10:14	Implicitly created (aut)

Figure 11.11: Staging Repositories List Panel

Actions for the selected staging repository/ies in the list include options to *Close*, *Promote*, *Release* or *Drop*. The *Refresh* button can be used to force a reload of the list of repositories. The *Filter by profile* drop-down allows you to select one or multiple staging profiles from which the repositories in the list were created. The list of repositories itself displays a number of columns with details for each repository. Further columns can be added by pressing on the drop-down triangle beside the currently selected column. Sorting by a single column in *Ascending* or *Descending* order can be set from the same drop-down as the column addition.

---

#### Note

When triggering a transition for a staging repository from e.g., the open state to a the closed state, a background task performs all the necessary operations. Since these are potentially longer running tasks, the user interface is not immediately updated. You are required to press *Refresh* to get the latest state of all repositories.

---

By default the following columns are displayed:

#### Checkbox

A checkbox to allow operations on multiple repositories.

#### Status Icon

An icon symbolizing the status of the staging repository.

#### Repository

The name of the staging repository.

#### Profile

The name of the staging profile, that was used to create the staging repository.

#### Status

Status of the repository.

---

**Updated**

Date and time of the last update.

**Description**

Textual description of the repository.

Additional columns are:

**Release To**

Target repository for the components in the staging repository after release.

**Promoted To**

The build promotion profile, to which a staging repository was optionally promoted to.

**Owner**

The username of the creator of the staging repository.

**Created**

Date and time of the creation of the staging repository.

**User Agent**

User agent string sent by the tool used for the deployment, e.g., Apache-Maven/3.0.5.

---

**Tip**

You can also access staging repositories in the list of repositories available in the *Repositories* panel available via the *Views/Repositories* as a Nexus managed repository.

---

In the following sections, you will walk through the process of managing staging repositories. Once you have deployed a set of related components, you must close the repository moving it from an *Open* to a *Closed* state unless the deployment tool automatically closed the staging repository.

A repository in the *Closed* state is added to a Repository Group and is made available for testing purposes or other inspection and can no longer receive additional components in it.

When the component examination is complete, you can either *Promote*, *Release*, or *Drop* the closed repository.

If the repository is dropped, the repository is discarded and removed from the Repository Group and the components are moved to the Trash.

---

If the repository is promoted, it is assigned to a build promotion profile for further staging activities.

If the repository is released, its components are moved to the target repository configured in the staging profile.

---

**Note**

A scheduled task documented in Section 6.5 can be used to clean up inactive staging repositories automatically.

---

Selecting a staging repository in the list displays further details about the repository in the *Summary*, *Activity*, and *Content* tabs below the list. An example for an open repository is displayed in Figure 11.12.

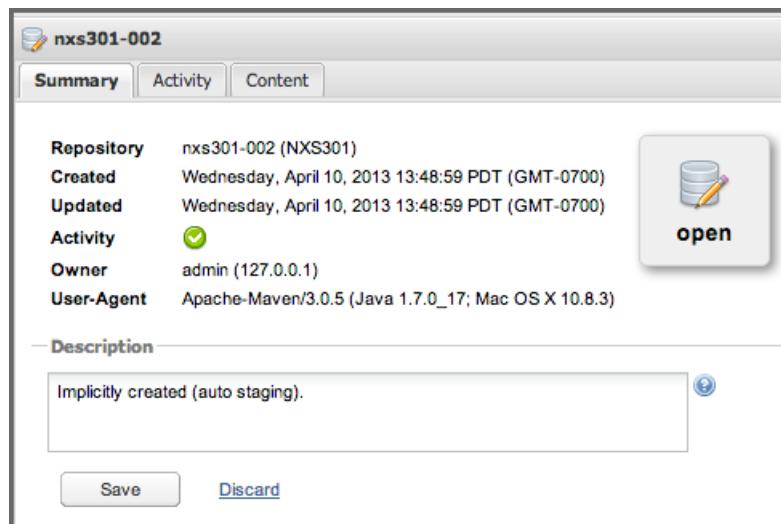


Figure 11.12: List of Activities Performed on a Promoted Staging Repository

The *Summary* tab displays a number of properties of the staging repository and allows you to edit the *Description*. The properties include the name of the repository, created date/time and updated date/time, activity indicator, owner and originating IP number of the deployment as well as the user agent string sent by the deployment. All staging operations have a default description that is used if the input field is left blank.

The *Activity* tab shows all the activities that occurred on a specific staging repository. An example for a

promoted repository is displayed in Figure 11.13. The activities are separated per activity and list all events that occurred in an activity. Selecting an event displays further details about the event on the right side of the tab.

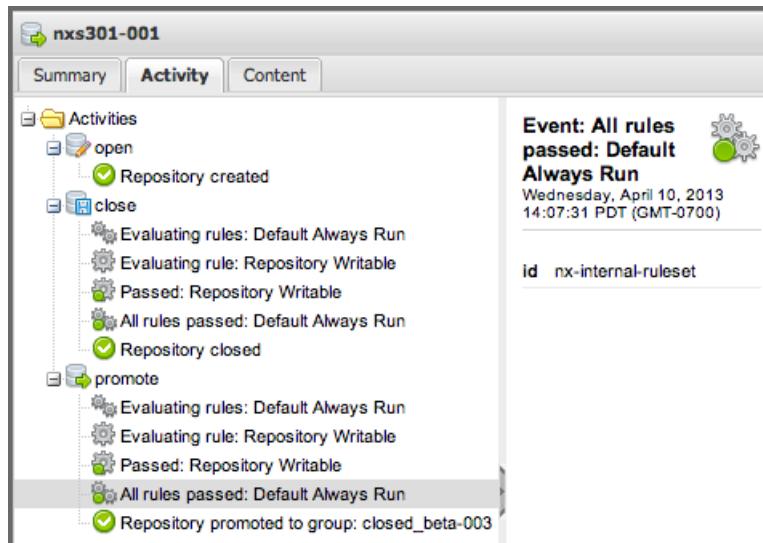


Figure 11.13: Details of an Open Staging Repository as Displayed under the List of Staging Repositories

The *Content* tab displays a repository browser view of the staging repository content and allows you to filter and display the components in the tree view. Selecting a specific component triggers the display of further panels with further information about the component, in the same manner as other repository browser views. The tabs include Maven and Artifact information and others.

For build promotion profile an additional *Members* tab is shown. It displays the source repositories and build promotion profiles from which this current build promotion profile was created.

#### 11.4.1 Closing an Open Repository

Once you deploy a component that triggers a staging profile, Nexus staging Suite will create a repository that contains the components you deployed. A separate staging repository is created for every combination of User ID, IP Address, and User Agent. This means that you can perform more than one deployment to a single staging repository, as long as you perform the deployment from the same IP with the same deployment user and the same installation of Maven.

You can perform multiple deployments to an open staging repository. Depending on the deployment tool and your configuration, the staging repository might be automatically closed during deployment or left open until manually closed.

Once you are ready to start testing the staging repository content, you will need to transition the repository from the open state to the closed state. This will close the staging repository to more deployments.

To close a repository, select the open staging repository in the list and by clicking the checkbox in the list or anywhere else in the row. For an open repository, the *Close* and the *Drop* buttons above the table will be activated. Pressing the *Close* button will bring up the dialog for a staging deployer to describe the contents of the staging repository and confirm . This description field can be used to pass essential information to the person who needs to test a deployment.

In Figure 11.14, the description field is used to describe the release for the user who needs to certify and promote a release.

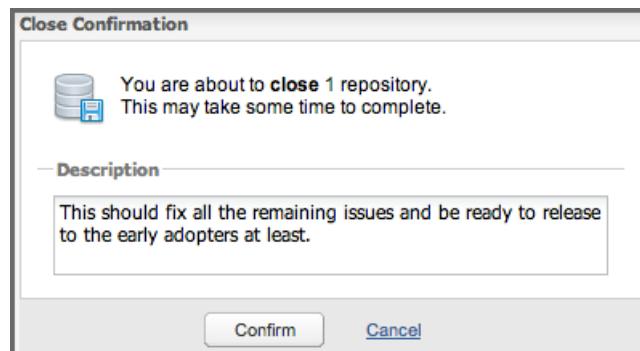


Figure 11.14: Confirmation and Description Dialog for Closing a Staging Repository

Confirming this state transition will close the repository and add the repository to the repository groups configured in the staging profile. The updated status will be visible in the list of staging repositories after a *Refresh*, since the transition could take longer depending on the configured staging rules and potential validation against Sonatype CLM.

### 11.4.2 Using the Staging Repository

Once the staging repository has been closed, it will automatically be added to the repository group(s) that are specified as target groups in the staging profile configuration.

This has the effect of making the staged artifacts available to everyone who is referencing this group. Developers who are referencing this repository group can now test and interact with the staged artifacts as if they were published to a Hosted repository.

While the artifacts are made available in a repository group, the fact that they are held in a temporary staging directory gives the staging user the option of promoting this set of artifacts to a hosted repository. Alternatively, the user can drop this temporary staging repository, if there are problems discovered during the testing and certification process for a release.

Once a staging repository is closed, you can also browse and search the repository in the staging repositories list.

To view all staging repositories, click on the *Repositories* item in the *Views/Repositories* menu and then select *Nexus Managed Repositories* as shown in Figure 11.15.

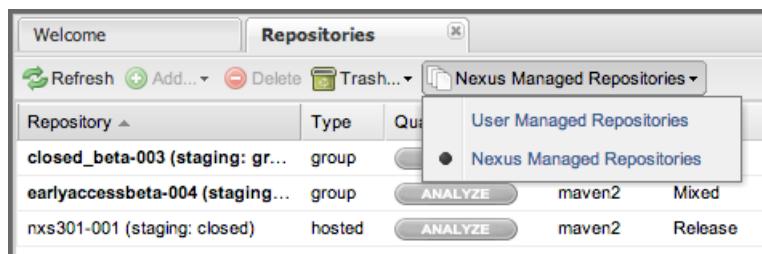


Figure 11.15: Viewing Nexus Managed Repositories

This list allows you to access all *Nexus Managed Repositories*, just like the *User Managed Repositories*, including browsing the content and accessing detailed information about the components in the repository. In addition to staging repositories, the list included procured repositories as documented in Chapter 10.

### 11.4.3 Releasing a Staging Repository

When you are finished testing or certifying the contents of a staging repository, you are ready to either release, promote, or drop the staging repository. Dropping the staging repository will delete the temporary it from Nexus and remove any reference to this repository from the groups with which it was associated. Releasing the staging repository allows you to publish the contents of this temporary repository to a hosted repository. Promoting the repository will move it to a build promotion profile.

You can release a staging repository by pressing *Release*, after selecting a closed staging repository from the staging repositories list. The *Release Confirmation* dialog displayed in Figure 11.16 will allow you to supply a description and configure if the staging repository should be automatically dropped after the components have been released to the hosted repository.

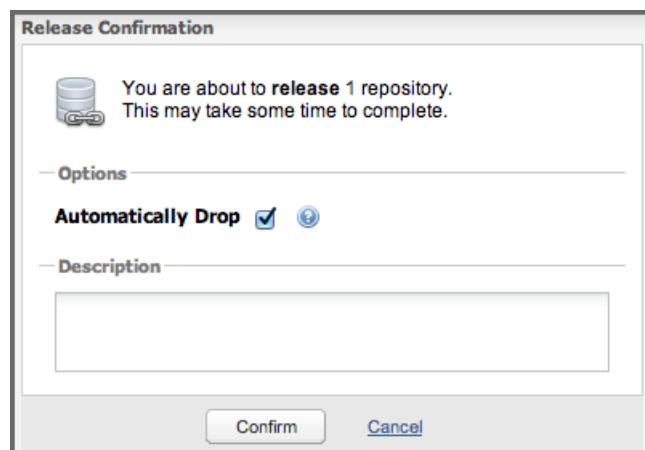


Figure 11.16: Confirmation Dialog for Releasing a Staging Repository

### 11.4.4 Promoting a Staging Repository

If you have a closed staging repository that you want to promote to a Build Promotion Profile, open the list of Staging Repositories and click the *Promote* button to bring up the *Promote Confirmation* dialog displayed in Figure 11.16. It allows you to select the build promotion profile to which you want to stage the repository to as well as provide a description.

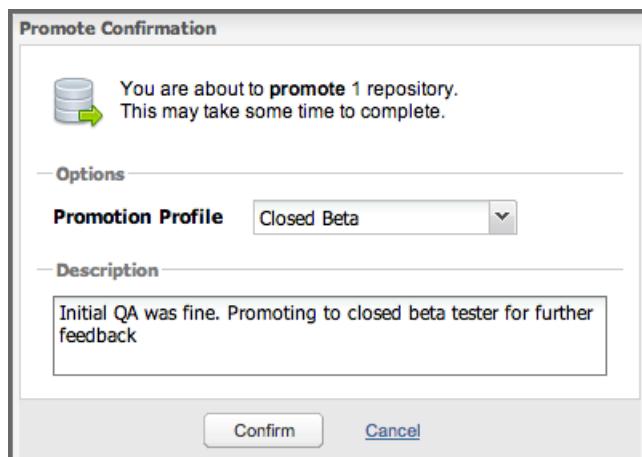


Figure 11.17: Confirmation Dialog for Promoting a Staging Repository

Clicking on the *Promote* button in the dialog will promote the staging repository to a build promotion repository and expose the contents of the selected staging repository through the target group(s) associated with the build promotion profile.

The build promotion repository is accessible in the staging repository list as displayed in Figure 11.18. If you add the column *Promoted To* to the list you will observe that Nexus keeps track of the promotion source. The *Members* tab for a build promotion repository displays the path of a build promotion repository back to a staging repository. One or more staging repositories can be promoted to a single build promotion profile.

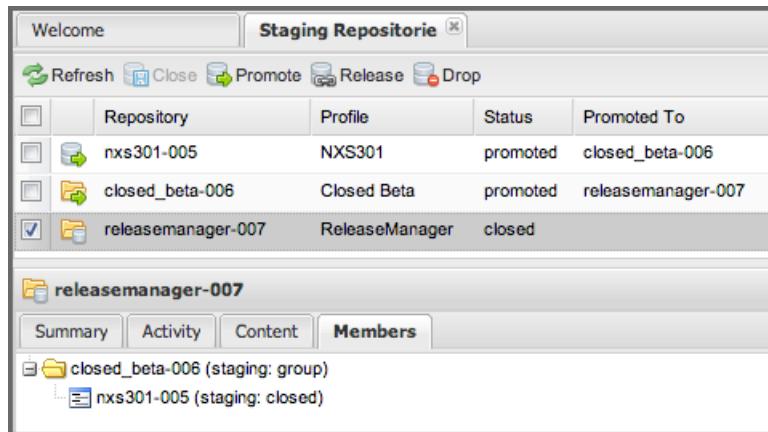


Figure 11.18: A Build Promotion Repository and its Members Panel

#### 11.4.5 Releasing, Promoting, and Dropping Build Promotion Profiles

When you configure a build promotion profile and promote staging repositories to promotion profiles, each build promotion profile creates a repository that contains one or more staging repositories. Just like you can promote the contents of a staging repository to a build promotion profile, you can also promote the contents of a build promotion profile to another build promotion profile. When you do this you can create hierarchies of staging repositories and build promotion profiles that can then be dropped or released together.

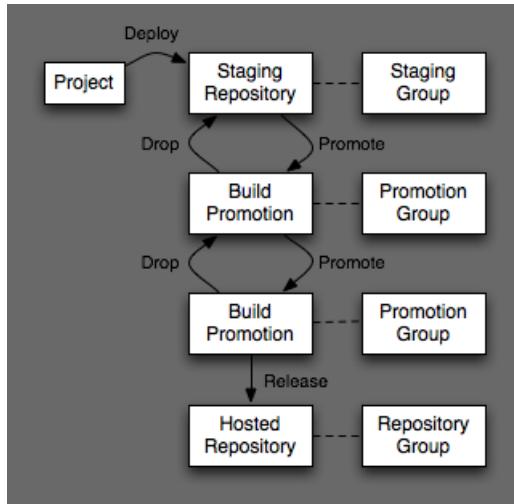


Figure 11.19: Releasing, Promoting, and Dropping Build Promotion Profiles

When you promote a staging repository to a build promotion profile, you make the contents of a staging repository available via a repository group associated with a build promotion profile.

For example, if you staged a few artifacts to a *QA* staging repository and then subsequently promoted that repository to a *Closed Beta* build promotion group, the contents of the *QA* staging repository would initially be made available via a *QA* repository group. After a build promotion, these artifacts would also be available via a *Closed Beta* repository group.

You can take it one step further and promote the contents of the *Closed Beta* build promotion profile to yet another build promotion profile. In this way you can have an arbitrary number of intermediate steps between the initial staging deployment and the final release.

If you drop the contents of a build promotion profile, you roll back to the previous state. For example, if you decided to drop the contents of the *Closed Beta* build promotion group, Nexus will revert the status of the staging repository from promoted to closed and make the artifacts available via the *QA* staging repository. The effects of promoting, dropping, and releasing artifacts through a series of staging profiles and build promotion profiles is shown in Figure 11.19.

When you perform a release on a build promotion profile, it rolls up to release all its members, ultimately reaching a staging repository. Each staging repository releases its components to the release repository configured in Figure 11.5. Because a build repository can contain one or more promoted staging repositories, this means that releasing a build promotion profile can cause components to be published to more

than one release repository.

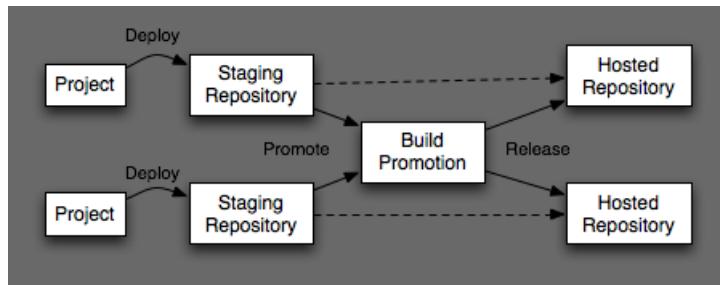


Figure 11.20: Promoting Multiple Repositories to the Same Build Promotion Profile

Build promotion profiles are not directly related to release repositories, only staging profiles are directly associated with target release repositories. Figure 11.20 illustrates this behavior with two independent staging repositories, each configured with a separate release repository. Releasing the build promotion profile causes Nexus to publish each staging repository to a separate hosted repository.

#### 11.4.6 Multilevel Staging and Build Promotion

Nexus also supports multilevel staging and build promotion. With multilevel staging, a staging repository can be tested and then promoted to multiple separate build promotion profiles consecutively and exposed through different repository groups to allow for additional testing and qualification before a final frelease. Figure 11.21 illustrates a potential use for multilevel staging:

##### Stage

A developer publishes components to a QA staging profile that exposes the staged components in a QA repository group used by an internal quality assurance team for testing.

##### Promote to Beta

Once the QA team has successfully completed testing, they promote the temporary staging repository to a build promotion profile that exposes the staged components to a limited set of customers who have agreed to act as beta testers for a new feature.

##### Release

Once this *Closed Beta* testing period is finished, the staged repository is then released and the artifacts it contains are published to a hosted release repository and exposed via the public repository group.

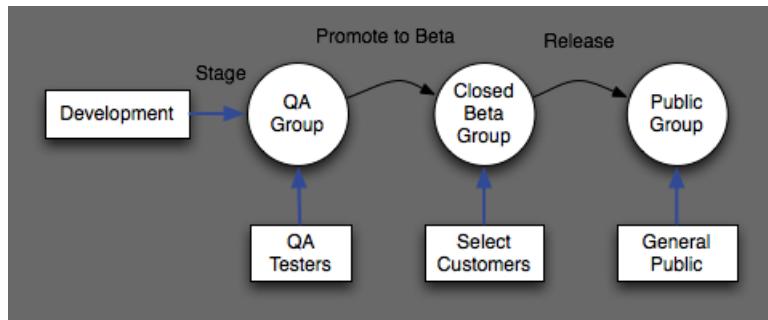


Figure 11.21: Multilevel Staging and Build Promotion

To support this multilevel staging feature, you can configure Build Promotion profiles as detailed in Section 11.2.3. Once you have promoted a Staging Repository to a Build Promotion profile, you can drop, promote, or release the artifacts it contains as detailed in Section 11.2.

## 11.5 Enforcing Standards for Deployment and Promotion with Rule-sets

Nexus has the ability to define staging rules that must be satisfied to allow successful deployment or before a staging repository can be promoted.

### 11.5.1 Managing Staging Rulesets

Staging rulesets are customizable groups of rules that are validated against the components in a staging repository when the repository is closed or promoted. If any rules cannot be validated, closing or promoting the repository will fail.

A staging repository associated with a staging ruleset configured in the staging profile cannot be closed or promoted until all of the rules associated with the rulesets have been satisfied. This allows you to set standards for your own hosted repositories, and it is the mechanism that is used to guarantee the consistency of components stored in the Central Repository.

To create a Staging Ruleset, click on the *Staging Ruleset* item in the *Build Promotion* menu. This will

load the interface shown in Figure 11.22. The Staging Ruleset panel is used to define sets of rules that can be applied to staging profiles.

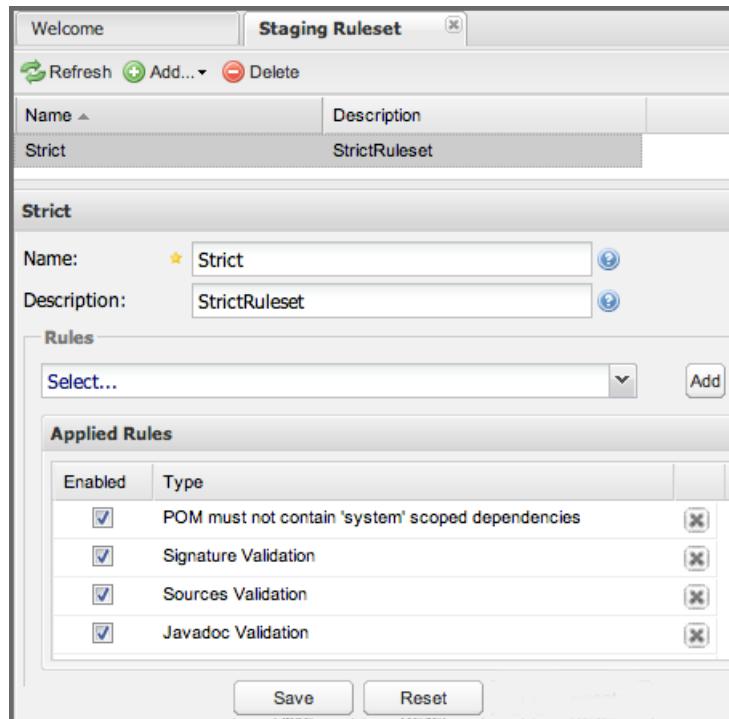


Figure 11.22: Creating a Staging Ruleset

Nexus contains the following rules:

#### Artifact Uniqueness Validation

This rule checks to see that the component being released, promoted, or staged is unique in a particular Nexus instance.

#### Checksum Validation

This rule validates that file checksum files are present and correct for the published components.

#### Javadoc Validation

The Javadoc Validation rule will verify that every project has a component with the javadoc classifier. If you attempt to promote a staging repository that contains components not accompanied by "-javadoc.jar" artifacts, this validation rule will fail.

### POM Validation

The Staging POM Validation rule will verify Project URL - project/url, Project Licenses - project/licenses and Project SCM Information - project/scm. Any of these POM elements cannot be missing or empty.

### POM must not contain *system* scoped dependencies

Ensures that no dependency is using the scope system. This allows for a path definition ultimately making the component rely on a specific relative path and using it is considered bad practice and violates the idea of having all necessary components available in repositories.

### POM must not contain release repository

This rule can ensure that no repository element is defined in the POM. This is important since it potentially would circumvent the usage of the repository manager and could point to other repositories that are not actually available to a user of the component

### Signature Validation

The Signature Validation rule verifies that every item in the repository has a valid PGP signature. If you attempt to promote a staging repository that contains artifacts not accompanied by valid PGP signature, this validation will fail.

### Sources Validation

The Sources Validation rule will verify that every project has an artifact with the sources classifier. If you attempt to promote a staging repository that contains artifacts not accompanied by "-sources.jar" artifacts, this validation rule will fail.

## 11.5.2 Defining Rulesets for Promotion

To define a ruleset to be used for closing or promotion, edit the staging profile by selecting it in the staging profile list. Scroll down to the sections *Close Repository Staging Rulesets* and *Promote Repository Staging Rulesets* as shown in Figure 11.23 and add the desired available rulesets to the left-hand list of activated rulesets for the current staging profile.

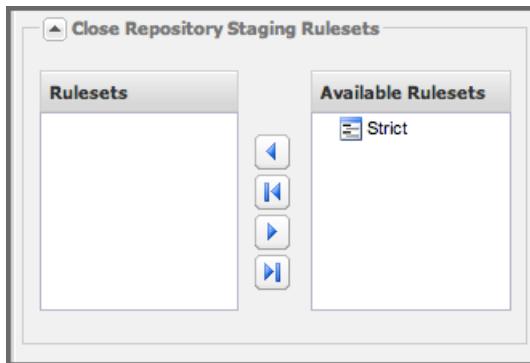


Figure 11.23: Associating a Staging Ruleset with a Staging Profile

The next time you attempt to close or promote a staging repository that was created with this profile, Nexus Professional will check that all of the rules in the associated rulesets are being followed.

## 11.6 Policy Enforcement with Sonatype CLM

As discussed in Chapter 2, Component Lifecycle Management (CLM) and Repository Management are closely related activities. The Sonatype CLM suite of tools provides a server application for administering your component usage policies and other features that integrate with other tools of the suite. It has access to extensive security vulnerability and license information data from the Sonatype CLM backend that can be used as input for your policies. For example you could establish a policy that is logged as violated, if any component in your software has a known security vulnerability or uses a license that is incompatible with your business model.

Nexus Professional - CLM Edition is an important component that can take advantage of the CLM server. The Sonatype CLM server can be integrated to validate policies as part of your usage of the staging suite of Nexus.

Detailed instructions on how to install and configure the Sonatype CLM server as well as the integration in Nexus can be found in the Sonatype CLM [documentation](#).

## 11.7 Artifact Bundles

### 11.7.1 Introduction

Artifact bundles are groups of related artifacts that are all related by the same groupId, artifactId, and version (GAV) coordinate. They are used by projects that wish to upload artifacts to the Central Repository.

Bundles must contain the following POM elements:

- modelVersion
- groupId
- artifactId
- packaging
- name
- version
- description
- url
- licenses
- scm
  - url
  - connection

### 11.7.2 Creating an Artifact Bundle from a Maven Project

Artifact bundles are created with the Maven Repository Plugin. For more information about the Maven Repository plugin, see <http://maven.apache.org/plugins/maven-repository-plugin/>.

[Sample POM Containing all Required Bundle Elements](#) lists a project's POM that satisfies all of the constraints that are checked by the Maven Repository plugin. The following POM contains a description

---

and a URL, SCM information, and a reference to a license. All of this information is required before an artifact bundle can be published to the Maven Central repository.

### Sample POM Containing all Required Bundle Elements

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.sonatype.sample</groupId>
  <artifactId>sample-project</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>sample-project</name>
  <description>A Sample Project for the Nexus Book</description>
  <url>http://books.sonatype.com</url>
  <licenses>
    <license>
      <name>The Apache Software License, Version 2.0</name>
      <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
      <distribution>repo</distribution>
    </license>
  </licenses>
  <scm>
    <connection>
      scm:git:git://github.com/sonatype/sample-project.git
    </connection>
    <url>http://github.com/sonatype/sample-project.git</url>
    <developerConnection>
      scm:git:git://github.com/sonatype-sample-project.git
    </developerConnection>
  </scm>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

---

To create a bundle from a Maven project, run the `repository:bundle-create` goal. This goal will check the POM to see if it complies with the standards for publishing a bundle to a public repository. It will then bundle all of the artifacts generated by a particular build. To build a bundle that only contains the standard, unclassified artifact from a project, run `mvn repository:bundle-create`. To generate a bundle that contains

```
more than one artifact, run mvn javadoc:jar source:jar repository:bundle-create:  
~/examples/sample-project$ mvn javadoc:jar source:jar repository:bundle- ↵  
    create  
[INFO] Scanning for projects...  
[INFO] Searching repository for plugin with prefix: 'javadoc'.  
[INFO]  ↵  
----- ↵  
[INFO] Building sample-project  
[INFO]   task-segment: [javadoc:jar, source:jar, repository:bundle-create ↵  
    ]  
[INFO]  ↵  
----- ↵  
[INFO] [javadoc:jar {execution: default-cli}]  
Loading source files for package com.sonatype.sample...  
Constructing Javadoc information...  
Standard Doclet version 1.6.0_15  
Building tree for all the packages and classes...  
...  
[INFO] Preparing source:jar  
[INFO] No goals needed for project - skipping  
[INFO] [source:jar {execution: default-cli}]  
...  
----- ↵
```

## TESTS

```
Running com.sonatype.sample.AppTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.03 sec
```

```
Results :
```

```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

```
[INFO] [jar:jar {execution: default-jar}]  
[INFO] Building jar: ~/temp/sample-project/target/sample-project-1.0.jar  
[INFO] [repository:bundle-create {execution: default-cli}]  
[INFO] The following files are marked for inclusion in the repository ↵  
  bundle:  
  
0.) Done  
1.) sample-project-1.0.jar  
2.) sample-project-1.0-javadoc.jar  
3.) sample-project-1.0-sources.jar
```

```
Please select the number(s) for any files you wish to exclude, or '0' when ↵  
  \  
you're done. Separate the numbers for multiple files with a comma (',').  
----- ↵
```

```
Selection:  
0  
[INFO] Building jar: ~/temp/sample-project/target/sample-project-1.0- ↵  
  bundle.jar  
[INFO]  ↵  
----- ↵  
[INFO] BUILD SUCCESSFUL  
[INFO]  ↵  
----- ↵  
[INFO] Total time: 11 seconds  
[INFO] Finished at: Sat Oct 10 21:24:23 CDT 2009  
[INFO] Final Memory: 36M/110M  
[INFO]  ↵  
----- ↵
```

Once the bundle has been created, there will be a bundle JAR in the target directory. As shown in the following command output, the bundle JAR contains a POM, the project's unclassified artifact, the javadoc artifact, and the sources artifact.

```
~/examples/sample-project$ cd target  
~/examples/sample-project/target$ jar tvf sample-project-1.0-bundle.jar  
0 Sat Oct 10 21:24:24 CDT 2009 META-INF/  
98 Sat Oct 10 21:24:22 CDT 2009 META-INF/MANIFEST.MF  
1206 Sat Oct 10 21:23:46 CDT 2009 pom.xml  
2544 Sat Oct 10 21:24:22 CDT 2009 sample-project-1.0.jar  
20779 Sat Oct 10 21:24:18 CDT 2009 sample-project-1.0-javadoc.jar  
891 Sat Oct 10 21:24:18 CDT 2009 sample-project-1.0-sources.jar
```

### 11.7.3 Uploading an Artifact Bundle to Nexus

To upload an artifact bundle to Nexus Professional, you have to have a repository target for the project configured as described in Section 6.14.

Once that is done, select *Staging Upload* from the *Build Promotion* section of the *Nexus* menu. This will load the *Staging Upload* tab. Choose *Artifact Bundle* from the *Upload Mode* drop-down. The *Staging Upload* panel will switch to the form shown in Figure 11.24. Click on *Select Bundle to Upload...* and then select the JAR that was created with the Maven repository plugin used in the previous sections. Once a bundle is selected, click on *Upload Bundle*.

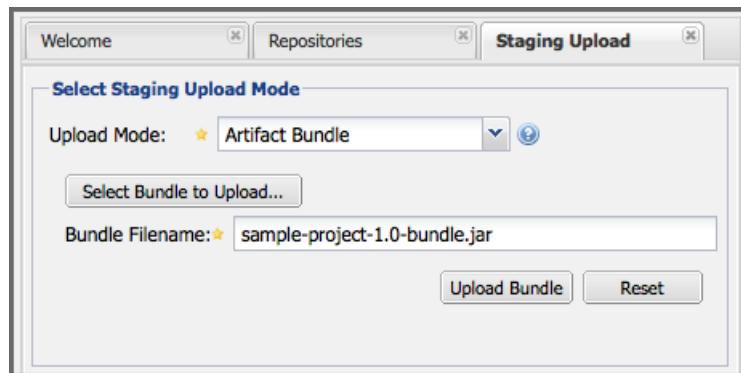


Figure 11.24: Uploading an Artifact Bundle

After a successful upload, a dialog displays the name of the created staging repository in a URL that links to the content of the repository. To view the staging repository, click on the *Staging Repositories* link in the *Build Promotion* section of the Nexus menu. You should see that the *Staging Artifact Upload* created and closed a new staging repository as shown in Figure 11.25. This repository contains all of the artifacts contained in the uploaded bundle. It allows you to promote or drop the artifacts contained in a bundle as a single unit.

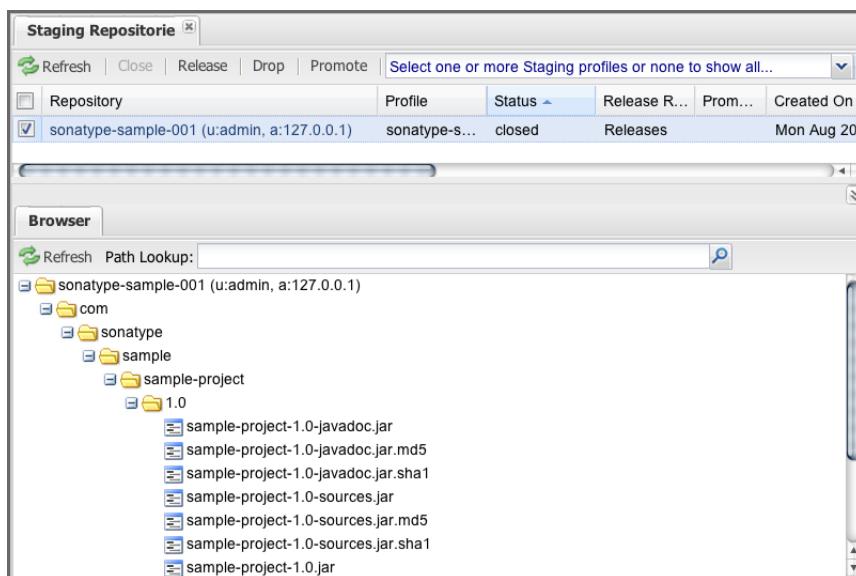


Figure 11.25: Staging Repository Created from Artifact Bundle Upload

Once the staging repository is closed, you can promote it to a Build Promotion Profile or release it to the target repository of the staging profile as documented in Section [11.4](#).

## Chapter 12

# Repository Health Check

Repository Health Check is a feature of Nexus that integrates data from [Sonatype CLM](#) and the related Hosted Data Services (HDS) run by Sonatype. Sonatype CLM is a suite of separate products that consists of tools to monitor and manage license, quality, and security data about artifacts used in your software development life cycle for your Component Lifecycle Management (CLM) efforts.

Repository health check provides access to a limited subset of the available data in Sonatype CLM and HDS right in your Nexus server. HDS exposes data about the artifacts in the [Central Repository](#) and other public repositories, including license information, security vulnerability data, and other statistics like relative usage popularity and age. Repository health check allows you to examine the available security and license data about components in a repository.

Repository health check analyzes all artifacts found in a proxy repository of any format. Maven 2 format repositories need to have a release policy configured.

---

### Note

At this time, while NuGet proxy repositories do support Repository Health Check, only identification of components is performed. No license or security data is provided at this time.

---

## 12.1 Configuring Repository Health Check

### 12.1.1 Configuration Per Repository

Repository health check can be setup for any repository, as long as the following apply:

- The Repository *Type* is *Proxy*.
- The Repository *Policy* is **not** *Snapshot*.
- The Repository is *In Service*.

Repository health check for a single repository can be enabled in one of two ways. The quickest way is to simply click the *Analyze* button. After pressing this button, you will be prompted to either analyze all or only the selected repository.

Alternatively, you can select the repository in the list of repositories and then set the *Enabled* configuration in the *Health Check* tab to *true* as displayed in Figure 12.1. Administrator privileges are required to perform this configuration.

The screenshot shows the Nexus Repository Management interface. At the top, there's a toolbar with 'Refresh', 'Add...', 'Delete', 'Trash...', and 'User Managed Repositories'. Below the toolbar is a table titled 'Repositories' with columns: Repository, Type, Health Check, Format, Policy, and Repository Status. The table lists several repositories:

Repository	Type	Health Check	Format	Policy	Repository Status	
Public Repositories	group	<button>ANALYZE</button>	maven2			
3rd party	hosted	<button>ANALYZE</button>	maven2	Release	In Service	
Apache Snapshots	proxy	<button>ANALYZE</button>	maven2	Snapshot	In Service	
Central	proxy	<span>shield 100</span>	<span>107</span>	maven2	Release	In Service
Central M1 shadow	virtual	<button>ANALYZE</button>	maven1	Release	In Service	
Codehaus Snapshots	proxy	<button>ANALYZE</button>	maven2	Snapshot	In Service	
MyProxyRepo	proxy	<button>ANALYZE</button>	maven2	Release	In Service	
Releases	hosted	<button>ANALYZE</button>	maven2	Release	In Service	
Snapshots	hosted	<button>ANALYZE</button>	maven2	Snapshot	In Service	

Below the table, a section titled 'MyProxyRepo' contains tabs: Browse Index, Browse Storage, Configuration, **Health Check**, Routing, SSL, Smart Proxy, and Summary. Under the 'Health Check' tab, there's a form titled 'Configure Repository Health Check' with a dropdown menu set to 'false'. At the bottom of this section are 'Save' and 'Reset' buttons.

Figure 12.1: Enabling Repository Health Check

---

**Note**

After enabling Repository Health Check for the first time you will be presented with an acceptance of the Terms of Service.

---

Once enabled, a scheduled task that performs the initial analysis is created and started. This task uses the identifier of the repository and the prefix *Health Check*: as a name and is configured to run regularly. New component data is supplied by the CLM data service to Nexus daily. The recurrence frequency can be changed in the scheduled task administration described in Section 6.5. Disabling health check for a specific repository removes this scheduled task automatically.

After a successful analysis, the *Health Check* column in the list of repositories will display security and license issue counts for the repository. An example is displayed in Figure 12.2.

Repository	Type	Health Check	Format	Policy	Repository Status
Public Repositories	group	<a href="#">ANALYZE</a>	maven2		
3rd party	hosted	<a href="#">ANALYZE</a>	maven2	Release	In Service
Apache Snapshots	proxy	<a href="#">ANALYZE</a>	maven2	Snapshot	In Service
Central	proxy	100  107	maven2	Release	In Service
Central M1 shadow	virtual	<a href="#">ANALYZE</a>	maven1	Release	In Service
Codehaus Snapshots	proxy	<a href="#">ANALYZE</a>	maven2	Snapshot	In Service
MyProxyRepo	proxy	<a href="#">ANALYZE</a>	maven2	Release	In Service
Releases	hosted	<a href="#">ANALYZE</a>	maven2	Release	In Service
Snapshots	hosted	<a href="#">ANALYZE</a>	maven2	Snapshot	In Service

Figure 12.2: The Repositories List with Health Check Result Counts

Hovering your mouse pointer over that value will display the *Repository Health Check* summary data in a pop-up window. A sample window is displayed in Figure 12.3.

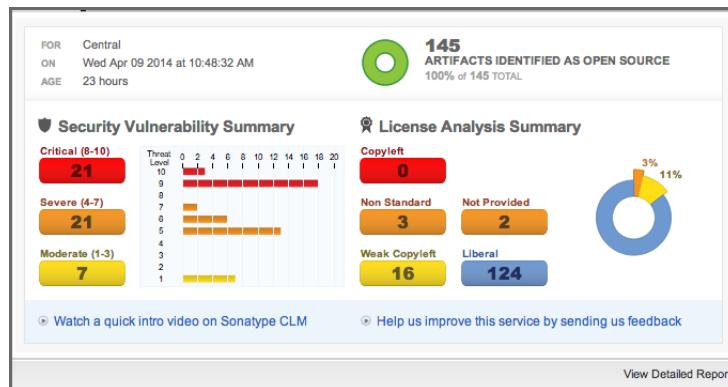


Figure 12.3: A Result Summary for a Repository Health Check

At the bottom of the pop-up window, you find the button *View Detailed Report* to access the detailed report. It will show up in another tab in the main area of the Nexus user interface.

### 12.1.2 Global Configuration

Alternatively to enabling and disabling health check for each repository, you can enable health check globally. This can be achieved by creating and configuring a new capability called *Health Check: Configuration*. Details about managing capabilities can be found in Section 6.6.

The health check configuration capability allows you to enable and disable it with the *Enabled* checkbox and set up health check for all proxy repositories by enabling *Configure for all proxy repositories*. With this configuration, health check will be enabled for all existing proxy repositories. Any newly created proxy repository will automatically have health check enabled as well.

---

**Note**

When disabling the global configuration option, if you also have the Repositories tab open, be sure to refresh Nexus to avoid viewing older data.

---

## 12.2 Accessing the Detailed Repository Health Check Report

The detailed report contains the same overview data and charts for security and license information at the top displayed in Figure 12.4 .

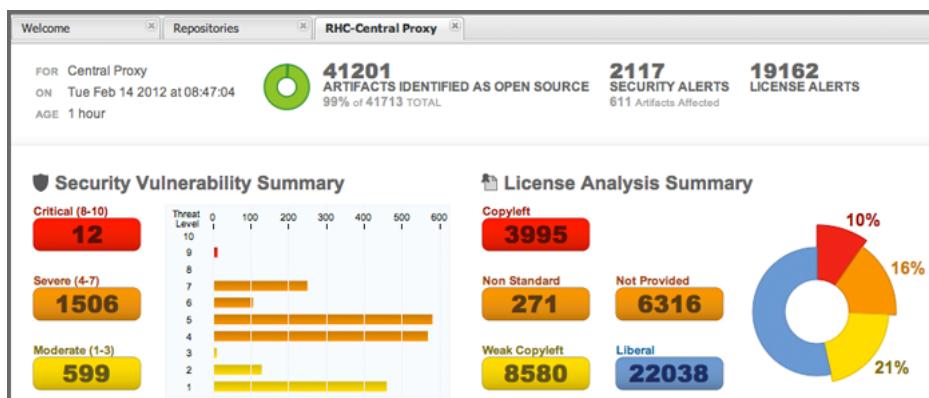
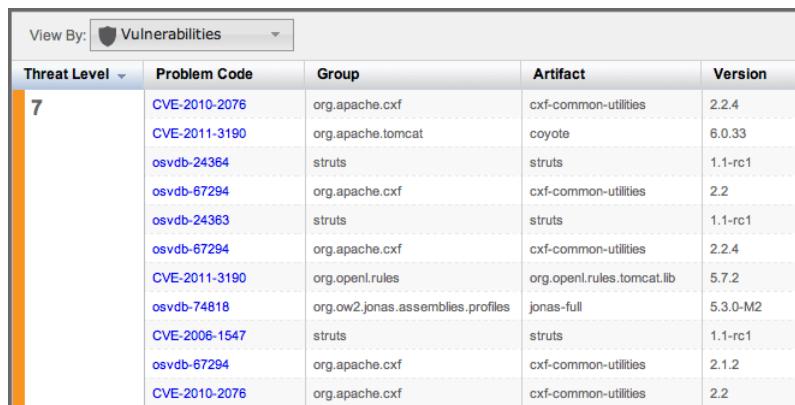


Figure 12.4: Summary of the Detailed Repository Health Check Panel

Below this overview, as visible in Figure 12.4, a drop-down for security and license information allows you to toggle between two lists displaying further details. Select to *View By: Vulnerabilities* to inspect the security issues and *View By: Artifacts* to review the license information. Both lists have a filter for each column at the bottom of the list that allows you to narrow down the number of rows in the table and find specific entries easily.

The security list as visible in Figure 12.5 contains columns for *Threat Level*, *Problem Code* and the GAV parameters identifying the affected artifact. The *Problem Code* column is a link to the security warning referenced and commonly links a specific entry in the [Open Source Vulnerability Database](#) or the [Common Vulnerabilities and Exposures](#) list. Both of these databases have a descriptive text for the vulnerability and further information and reference links.



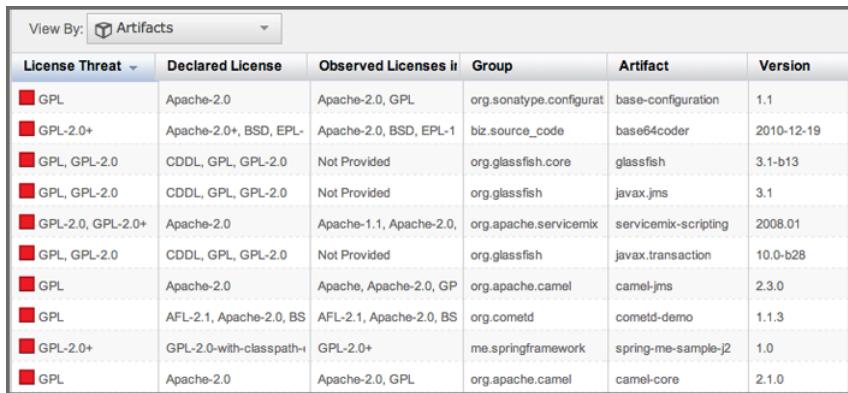
A screenshot of a web-based application interface titled "Detailed Repository Health Check Report". At the top, there is a dropdown menu labeled "View By:" with an orange shield icon, currently set to "Vulnerabilities". Below the header, there is a table with the following columns: Threat Level, Problem Code, Group, Artifact, and Version. The table contains 11 rows of data. The "Threat Level" column uses a color-coded scale: orange for values 4-7, red for values 8-10, and grey for values 1-3. The "Problem Code" column contains links to specific security issues. The "Group" column lists the affected library components. The "Artifact" column shows the specific version of the artifact. The "Version" column lists the version numbers. The data is as follows:

Threat Level	Problem Code	Group	Artifact	Version
7	<a href="#">CVE-2010-2076</a>	org.apache.cxf	cxfr-common-utilities	2.2.4
7	<a href="#">CVE-2011-3190</a>	org.apache.tomcat	coyote	6.0.33
7	<a href="#">osvdb-24364</a>	struts	struts	1.1-rc1
7	<a href="#">osvdb-67294</a>	org.apache.cxf	cxfr-common-utilities	2.2
7	<a href="#">osvdb-24363</a>	struts	struts	1.1-rc1
7	<a href="#">osvdb-67294</a>	org.apache.cxf	cxfr-common-utilities	2.2.4
7	<a href="#">CVE-2011-3190</a>	org.openl.rules	org.openl.rules.tomcat.lib	5.7.2
7	<a href="#">osvdb-74818</a>	org.ow2.jonas.assemblies.profiles	jonas-full	5.3.0-M2
7	<a href="#">CVE-2006-1547</a>	struts	struts	1.1-rc1
7	<a href="#">osvdb-67294</a>	org.apache.cxf	cxfr-common-utilities	2.1.2
7	<a href="#">CVE-2010-2076</a>	org.apache.cxf	cxfr-common-utilities	2.2

Figure 12.5: The Security Data in the Detailed Repository Health Check Report

The *Threat Level* is rated in values used by the vulnerability databases and ranges from 0 for a low threat to 10 for the highest threat. *Critical* values (noted in red) range from 8 to 10. *Severe* values (noted in orange) range from 4-7, and *Moderate* values (noted in yellow) range from 1 to 3.

The license list as visible in Figure 12.6 shows a derived threat in the *Effective License Threat* column. The *Declared License* column details the license information found in POM file. The *Observed Licenses in Source* columns lists all the licenses found in the actual source code of the library in the form of file headers and license files. The next columns for the GAV parameters allow you to identify the artifact. The last column *Security Issues* displays an indicator for potentially existing security issue for the same artifact.



The screenshot shows a table titled 'View By: Artifacts' with a dropdown menu. The table has columns: License Threat, Declared License, Observed Licenses in, Group, Artifact, and Version. The data is as follows:

License Threat	Declared License	Observed Licenses in	Group	Artifact	Version
GPL	Apache-2.0	Apache-2.0, GPL	org.sonatype.configurat	base-configuration	1.1
GPL-2.0+	Apache-2.0+, BSD, EPL-	Apache-2.0, BSD, EPL-1	biz.sourceforge	base64coder	2010-12-19
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	org.glassfish.core	glassfish	3.1-b13
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	org.glassfish	jaxws-jms	3.1
GPL-2.0, GPL-2.0+	Apache-2.0	Apache-1.1, Apache-2.0,	org.apache.servicemix	servicemix-scripting	2008.01
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	org.glassfish	jaxws-transaction	10.0-b28
GPL	Apache-2.0	Apache, Apache-2.0, GP	org.apache.camel	camel-jms	2.3.0
GPL	AFL-2.1, Apache-2.0, BS	AFL-2.1, Apache-2.0, BS	org.cometd	cometd-demo	1.1.3
GPL-2.0+	GPL-2.0-with-classpath+I	GPL-2.0+	me.springframework	spring-me-sample-j2	1.0
GPL	Apache-2.0	Apache-2.0, GPL	org.apache.camel	camel-core	2.1.0

Figure 12.6: The License Data in the Detailed Repository Health Check Report

Licenses such as GPL-2.0 or GPL-3.0 are classified as the highest *License Threat* and labeled as *Copyleft* and use red as signaling color.

A *Non Standard* or *Not Provided* license is classified as a moderate threat and uses orange. *Non Standard* as a classification is triggered by the usage of atypical licenses for open source software such as [CharityWare license](#), [BeerWare](#), [NCSA Open Source License](#) and many others. *Not Provided* is triggered as classification if no license information was found anywhere.

Licenses such as CDDL-1.0, EPL-1.0 or GPL-2.0-CPE receive a *Weak Copyleft* classification and yellow as notification color.

*Liberal* licenses that are generally friendly to inclusion in commercial products use blue and include licenses such as Apache-2.0, MIT or BSD.

A general description about the implications of the different licenses is available when hovering over the specific category in the *License Analysis Summary*. Further information about the different licenses can be obtained from the [Open Source Initiative](#). Mixed license scenarios like a mixture of licenses such as Apache-1.1, Apache-2.0, LGPL and LGPL-2.1 can be complicated to assess in its impact and might be legally invalid depending on the combination of licenses observed. Detailed implications to your business and software are best discussed with your lawyers.

Nexus will report all artifacts in the local storage of the respective repository in the detail panel. This means that at some stage a build running against your Nexus instance required these artifacts and caused Nexus to download them to local storage.

To determine which project and build caused this download to be able to fix the offending dependency by upgrading to a newer version or removing it with an alternative solution with a more suitable license, you will have to investigate all your projects.

Sonatype CLM itself helps with these tasks by enabling monitoring of builds and products, analyzing release artifacts and creating bill of material and other reports.

## Chapter 13

# Managing Maven Settings

### 13.1 Introduction

When you move an organization to a repository manager such as Nexus, one of the constant challenges is keeping everyone's Maven settings synchronized to ensure the Nexus server is used and any further configuration in the settings file is consistent. In addition, different users or use cases require different settings files. You can find out more about the Maven settings file in Chapter 4. Nexus Professional allows you to define templates for Maven settings stored on the server and provide them to users via the user interface or automated download.

If a Nexus administrator makes a change that requires every developer to modify his or her `~/.m2/settings.xml` file, this feature can be used to manage the distribution of Maven settings changes to the entire organization. Once you have defined a Maven settings template in Nexus Professional, developers can then use the Nexus M2Settings Maven Plugin to retrieve the new Maven settings file directly from Nexus Professional.

### 13.2 Manage Maven Settings Templates

To manage Maven settings templates, click on *Maven Settings* in the *Enterprise* section of the Nexus menu on the left side of the Nexus UI. This will load the panel shown in Figure 13.1.

---

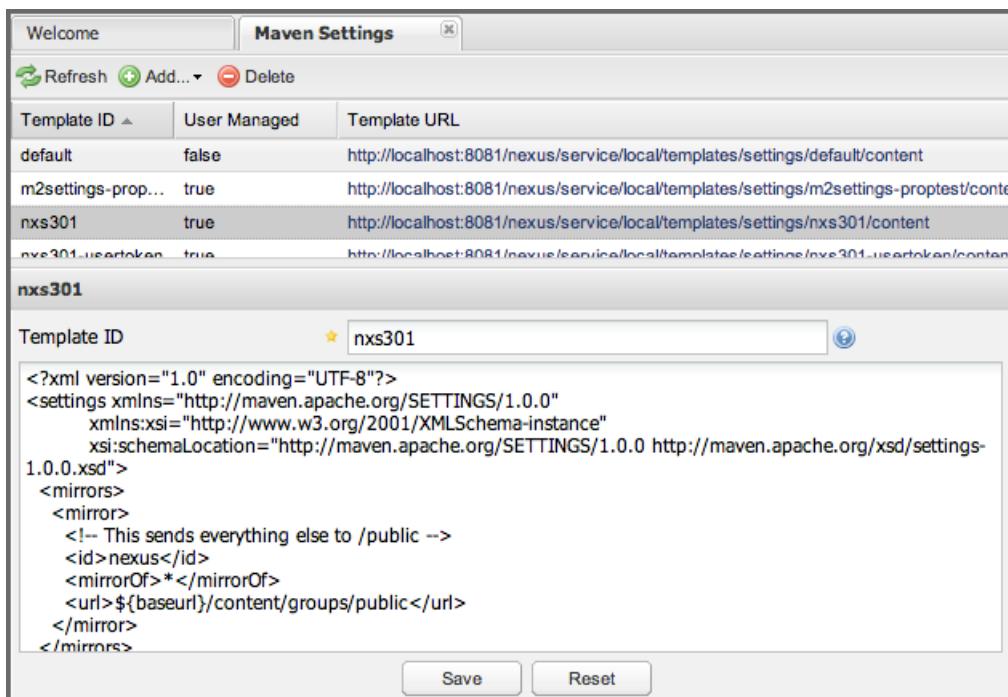


Figure 13.1: The Maven Settings Panel

The Maven Settings panel allows you to add, delete, and edit Maven Settings templates. The default template has an ID of `default` and can not be changed. It contains the recommended settings for a standard Nexus installation. To create a new Maven settings template, click on the *Add...* button and select *Settings Template*. Once the new template is created, assign a name to the template in the *Template ID* text input and click the *Save* button.

To edit a template, click on a template that has a *User Managed* value of `true` in the list and edit the template in the tab below the list. Once you are finished editing the template, click *Save* to save the template. When editing the template you can insert some property references that will be replaced on the server with their values at request time:

### **baseurl**

The base URL of the Nexus installation.

### **userId**

The user id of the user that is generating a Maven Settings file from this template.

Server side interpolation takes effect even when the download of the settings template is done with tools like curl. These properties can be referenced in the settings file using the the syntax \${property}:

```
<settings>
  <mirrors>
    <mirror>
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>${baseurl}/content/groups/public</url>
    </mirror>
  ...

```

To preview a Maven settings template, click on the *Template URL* in the list. Clicking on this URL loads a dialog window that contains the Maven Settings file generated from this template. This rendered view of the Maven Settings template has all variable references replaced using the current context of the user. This is the result of running the property replacement on the Nexus server.

The Nexus M2Settings Maven Plugin supports the more powerful and feature-rich, client-side replacement of properties using a \${[property]} syntax.

Client-side properties supported by the Nexus M2Settings Maven Plugin are

**baseurl**

The base URL of the Nexus installation.

**userId or username**

The username of the user that is requesting a Maven Settings file from this template.

**password**

The password of the user.

**userToken**

The formatted user token composed of name code, : and pass code.

**userToken.nameCode**

The name code part of the user token.

**userToken.passCode**

The pass code part of the user token.

**userToken.passCode.encrypted**

The encrypted pass code part of the user token.

Client side interpolation allows you to fully populate a <server> section with the required properties either with the plain text username and password:

---

```
<server>
  <id>nexus</id>
  <username>$[username]</username>
  <password>$[password]</password>
</server>
```

You can also use the the usertoken equivalent:

```
<server>
  <id>nexus</id>
  <!-- User-token: ${userToken} -->
  <username>$[userToken.nameCode]</username>
  <password>$[userToken.passCode]</password>
</server>
```

Alternatively you can use [Maven master-password encryption](#) with the master keyword in `settings-security.xml`

```
<server>
  <id>nexus-client-side-interp-encrypted</id>
  <!-- Maven master password encrypted user token password -->
  <username>$[userToken.nameCode]</username>
  <password>$[userToken.passCode.encrypted]</password>
</server>
```

The usage of the `.encrypted` key results in values similar to the following snippet:

```
<server>
  <id>nexus-client-side-interp-encrypted</id>
  <!-- master password encrypted user token password -->
  <username>KOYC8Q76</username>
  <password>{fsx2f...}</password>
</server>
```



### Warning

`userToken.*` properties are only expanded to values if the User Token feature as documented in Section [6.17](#) is enabled and configured.

## 13.3 Nexus M2Settings Maven Plugin

Once you have defined a set of Maven templates, you can use the Nexus M2Settings Maven Plugin to distribute changes to the settings file to the entire organization.

### 13.3.1 Running the Nexus M2Settings Maven Plugin

To invoke a goal of the Nexus M2Settings Maven Plugin, you will initially have to use a fully qualified groupId and artifactId in addition to the goal. An example invocation of the `download` goal is:

```
mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:download
```

In order to be able to use an invocation with the simple plugin prefix like this

```
mvn nexus-m2settings:download
```

you have to have the appropriate plugin group `org.sonatype.plugins` configured in your Maven Settings file:

```
<settings>
  ...
  <pluginGroups>
    <pluginGroup>org.sonatype.plugins</pluginGroup>
  </pluginGroups>
  ...
```

An initial invocation of the `download` goal will update your settings file with a template from Nexus Professional. The default template in Nexus Professional adds the `org.sonatype.plugins` group to the `pluginGroups`, so you will not have to do this manually. It is essential that you make sure that any new, custom templates also include this plugin group definition. Otherwise, there is a chance that a developer could update his or her Maven Settings and lose the ability to use the Nexus Maven plugin with the short identifier.

---

#### Tip

This practice of adding `pluginGroups` to the settings file is useful for your own Maven plugins or other plugins that do not use the default values of `org.apache.maven.plugins` or `org.codehaus.mojo` as well, since it allows the short prefix of a plugin to be used for an invocation outside a Maven project using the plugin.

---

The download goal of the Nexus M2Settings Maven Plugin downloads a Maven Settings file from Nexus Professional and stores it locally. The default file name for the settings file is the Maven default for the current user of `~/.m2/settings.xml`. If you are replacing a Maven Settings file, this goal can be configured to make a backup of an existing Maven Settings file.

---

**Note**

The download with the Nexus Maven Plugin is deprecated and has been replaced with the Nexus M2Settings Maven Plugin.

---

### 13.3.2 Configuring Nexus M2Settings Maven Plugin

The download goal of the Nexus M2Settings Maven plugin prompts the user for all required parameters, which include the Nexus server URL, the username and password, and the template identifier.

---

**Note**

For security reasons, the settings download requires an HTTPS connection to your Nexus instance. If you are running Nexus via plain HTTP you will have to set the `secure` parameter to `false`.

---

The required configuration parameters can either be supplied as invocation parameters or when prompted by the plugin and are:

**nexusUrl**

Points to the Nexus server installation's base URL. If you have installed Nexus on your local machine, this would be <http://localhost:8081/nexus/>. Access via HTTP only works with the `secure` configuration parameter set to `false`.

**username**

The username to use for authenticating to Nexus. Default value is the Java System property `user.name`.

**password**

The password to use for authenticating to Nexus.f

**templateId**

The Template ID for the settings template as defined in the Nexus user interface.

---

Additional general configuration parameters are related to the security of the transfer and the output file:

**secure**

By default set to `true`, this parameter forces a Nexus URL access with HTTPS. Overriding this parameter and setting it to `false` allows you to download a settings file via HTTP. When using this override it is important to keep in mind that the username and password transferred via HTTP can be intercepted.

**outputFile**

Defines the filename and location of the downloaded file and defaults to the standard `~/.m2/settings.xml`.

**backup**

If true and there is a pre-existing `settings.xml` file in the way of this download, back up the file to a date-stamped filename, where the specific format of the timestamp is given by the `backupTimestampFormat` parameter. Default value is `true`.

**backup.timestampFormat**

When backing up an existing `settings.xml` file, use this date format in conjunction with `SimpleDateFormat` to construct a new filename of the form: `settings.xml-(format)`. Date stamps are used for backup copies of the `settings.xml` to avoid overwriting previously backed up `settings` files. This protects against the case where the download goal is used multiple times with incorrect settings, where using a single static backup file name would destroy the original, preexisting settings. Default value is: `yyyyMMddHHmmss`.

**encoding**

Use this optional parameter to define a non-default encoding for the settings file.

As a Maven plugin, the Nexus M2Settings Maven Plugin relies on Apache Maven execution and on the fact that the Central Repository can be contacted for downloading the required plugins and dependencies. If this access is only available via a proxy server you can configure the proxy related parameters `proxy`, `proxy.protocol`, `proxy.host`, `proxy.port`, `proxy.username` and `proxy.password`.

### 13.3.3 Downloading Maven Settings

You can download the Maven Settings from Nexus Professional with a simple invocation, and rely on the plugin to prompt you for the required parameters:

```
$ mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:download
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
```

```
[INFO] -----
[INFO]
[INFO] --- nexus-m2settings-maven-plugin:1.6.2:download (default-cli) @ ←
      standalone-pom ---
Nexus URL: https://localhost:8081/nexus
Username [manfred]: admin
Password: *****
[INFO] Connecting to: https://localhost:8081/nexus (as admin)
[WARNING] Insecure protocol: https://localhost:8081/nexus/
[INFO] Connected: Sonatype Nexus Professional 2.4.0-07
Available Templates:
  0) default
  1) example
Select Template: 0
[INFO] Fetching content for templateId: default
[INFO] Backing up: /Users/manfred/.m2/settings.xml to: /Users/manfred/.m2/ ←
      settings.xml-20130404120146
[INFO] Saving content to: /Users/manfred/.m2/settings.xml
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 29.169s
[INFO] Finished at: Thu Apr 04 12:01:46 PDT 2013
[INFO] Final Memory: 12M/153M
[INFO] -----
```

If your Nexus server is hosted internally and does not use HTTPS you can download a settings file with

```
$ mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:download -Dsecure ←
  =false
```

As displayed, the plugin will query for all parameters and display a list of the available templates. Alternatively, you can specify the username, password, Nexus URL, and template identifier on the command line.

```
$ mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:download \
-DnexusUrl=https://localhost:8443/nexus \
-Dusername=admin \
-Dpassword=admin123 \
-DtemplateId=default
```

Enabling proxy access with `-Dproxy=true` will trigger the plugin to query the necessary configuration:

```
[INFO] Connecting to: https://localhost:8443/nexus (as admin)
Proxy Protocol:
  0) http
```

```
1) https
Choose: 1
Proxy Host: myproxy.example.com
Proxy Port: 9000
Proxy Authentication:
  0) yes
  1) no
Choose: 0
Proxy Username [manfred]: proxy
Proxy Password: *****
[INFO] Proxy enabled: proxy@https:myproxy.example.com:9000
```

In some scenarios you have to get an initial settings file installed on a computer that does not have internet access and, therefore, cannot use the Maven plugin. For this first initial configuration that connects the computer to Nexus for following Maven invocations, a simple HTTP GET command to retrieve an unmodified settings file can be used:

```
curl -u admin:admin123 -X GET "http://localhost:8081/nexus/service/local/←
  templates/settings/default/content" > ~/.m2/settings.xml
```

Modify the commandline above by changing the username:password supplied after -u and adapting the url to Template URL visible in the Nexus user interface. This invocation will however not replace parameters on the client side, so you will have to manually change any username or password configuration, if applicable.

## 13.4 Summary

Overall the Maven Settings integration in Nexus allows you to maintain multiple settings template files on the central Nexus server. You can configure settings files for different use cases like e.g.,

- referencing a repository group containing only approved components in the mirror section for your release or QA builds,
- providing an open public group mirror reference to all of your developers for experimentation with other components.

By using the Nexus M2Settings Maven Plugin you can completely automate initial provisioning and updates of these settings files to your users.

---

## Chapter 14

# OSGi Bundle Repositories

### 14.1 Introduction

Nexus supports the OSGi Bundle Repository format. The OSGi Bundle format is defined by the [OSGi RFC 112 "Bundle Repository."](#) It is a format for the distribution of OSGi *bundles* which includes any components that are described by the OSGi standards set forth in RFC 112. An OBR repository has a single XML file that completely describes the contents of the entire repository. Nexus Professional can read this OBR repository XML and create proxy repositories that can download OSGi bundles from remote OBR repositories. Nexus Professional can also act as a hosting platform for OSGi bundles. You can configure your builds to publish OSGi bundles to Nexus Professional, and then expose these bundle repositories to internal or external developers using Nexus Professional as a publishing and distribution platform.

Nexus Professional can also act as a bridge between Maven repositories and OSGi bundle repositories. When you configure a virtual OBR repository that uses a Maven 2 repository as a source repository, Nexus Professional will expose artifacts with the appropriate metadata from the Maven repository as OSGi bundles. In this way, you can unify your OSGi and non-OSGi development efforts and publish artifacts with the appropriate OSGi metadata to Nexus Professional. Non-OSGi clients can retrieve software artifacts from a Maven repository, and OSGi-aware clients can retrieve OSGi bundles from a virtual OBR repository.

The following sections detail the procedures for creating and managing OBR repositories.

---

Nexus Professional has OBR support installed by default. Prior to any usage in Nexus OSS the Nexus OBR Plugin needs to be installed. You can download the `-bundle.zip` file for your specific Nexus version from the Central Repository:

- [Nexus OBR Plugin](#)

Extract the file into `sonatype-work/nexus/plugin-repository` and restart Nexus. Ensure to repeat the step for any Nexus upgrades.

## 14.2 Proxy OSGi Bundle Repositories

Nexus can proxy an OSGi Bundle Repository using the OBR repository XML as the remote storage location. To create a new proxy OBR repository access the *Repositories* view from the *Views.Repositories* submenu and click the *Add..* button above the list of repositories and choose *Proxy Repository* from the drop-down of repository types.

In the *New Proxy Repository* configuration tab, supply a *Repository ID* and a *Repository Name* and select *OBR* as the *Provider*.

Then enter the URL to the remote repository OBR XML as the *Remote Storage Location* and click *Save*.

Figure 14.1 provides a sample configuration used to create a proxy of the Apache Felix OBR repository.

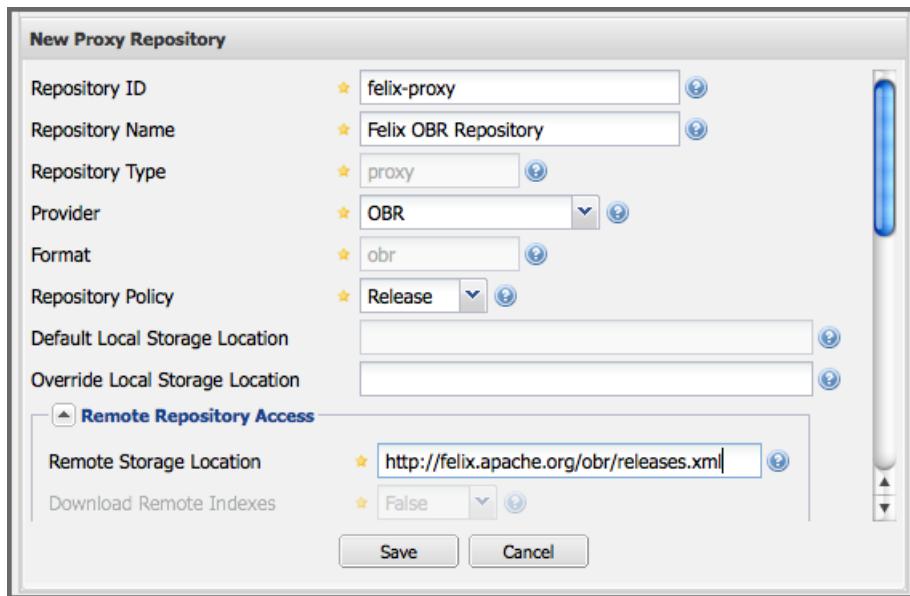


Figure 14.1: Creating an OSGi Bundle Proxy Repository

To verify that the OBR proxy repository has been properly configured, you can then load the OBR XML from Nexus Professional. If Nexus Professional is properly configured, you will be able to load the `obr.xml` by navigating to the `.meta` directory:

```
$curl http://localhost:8081/nexus/content/repositories/felix-proxy/.meta/ ←
  obr.xml
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type='text/xsl' href='http://www2.osgi.org/www/obr2html. ←
  xsl'?>
<repository name='Felix OBR Repository' lastmodified='1247493075615'>
  <resource id='org.apache.felix(javax.servlet)/1.0.0'
    presentationname='Servlet 2.1 API'
    symbolicname='org.apache.felix(javax.servlet)'
    uri='../../bundles/org.apache.felix(javax.servlet)-1.0.0.jar'
    version='1.0.0'>
    <description>
      Servlet 2.1 API
    </description>
    <documentation>
      http://www.apache.org/
    </documentation>
    <license>
      http://www.apache.org/licenses/LICENSE-2.0.txt
    </license>
```

```
</license>  
...
```

## 14.3 Hosted OSGi Bundle Repositories

Nexus can host an OSGi Bundle Repository, providing you with a way to publish your own OBR bundles. To create a hosted OBR repository access the *Repositories* view from the *Views/Repositories* submenu and click the *Add..* button above the list of repositories and choose *Hosted Repository* from the drop-down of repository types.

In the *New Hosted Repository* configuration tab, supply a *Repository ID* and a *Repository Name* and select *OBR* as the *Provider*.

Then enter the URL to the remote repository OBR XML as the *Remote Storage Location* and click *Save*.

Figure 14.2 provides some sample configuration used to create a hosted OBR repository.

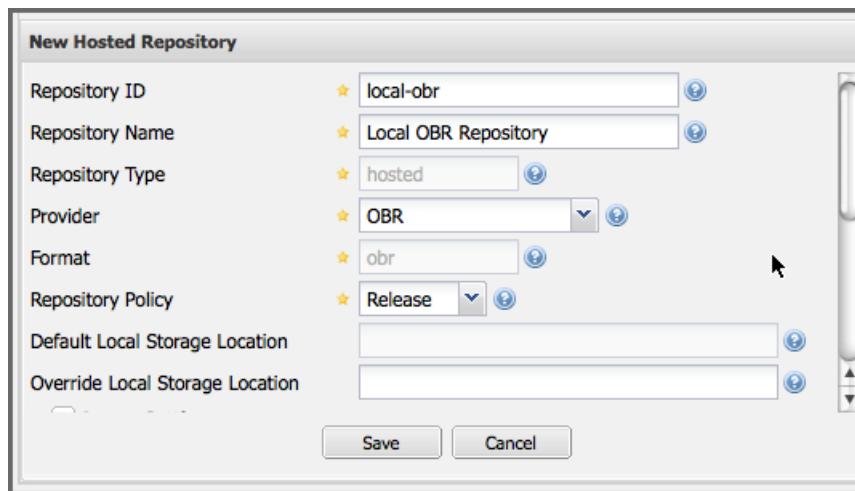


Figure 14.2: Creating a Hosted OSGi Bundle Repository

## 14.4 Virtual OSGi Bundle Repositories

Nexus Professional can be configured to convert a traditional Maven repository into an OSGi Bundle repository using a virtual OBR repository. To configure a virtual OBR repository, create a new *Virtual Repository* in the *Repositories* administration area providing a *Repository ID* and *Repository Name* as well as the *Source Nexus Repository ID* setting the repository you want to expose as OBR. Finally set the *Provider to OBR* and click *Save*.

Figure 14.3 provides a sample configuration used to create a virtual OBR repository that transforms the proxy repository for *Maven Central* into an OBR repository.

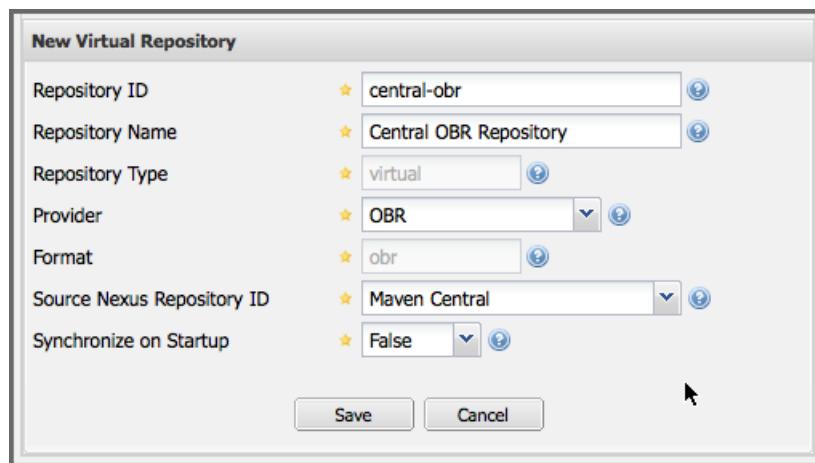


Figure 14.3: Creating a Virtual OSGi Bundle Repository from a Maven Repository

## 14.5 Grouping OSGi Bundle Repositories

Just like Nexus can group Maven repositories, Eclipse update sites, and P2 repositories, Nexus can also be configured to group OSGi Bundle Repositories. To group OSGi bundle repositories, create a new *Repository Group* and set the *Provider* to *OBR* and select the repositories you want to group after providing a *Group ID* and a *Group Name*.

Figure 14.4 shows an example of the a new repository group that contains a hosted OSGi Bundle repository, a virtual OSGi Bundle repository, and a OSGi Bundle proxy repository.

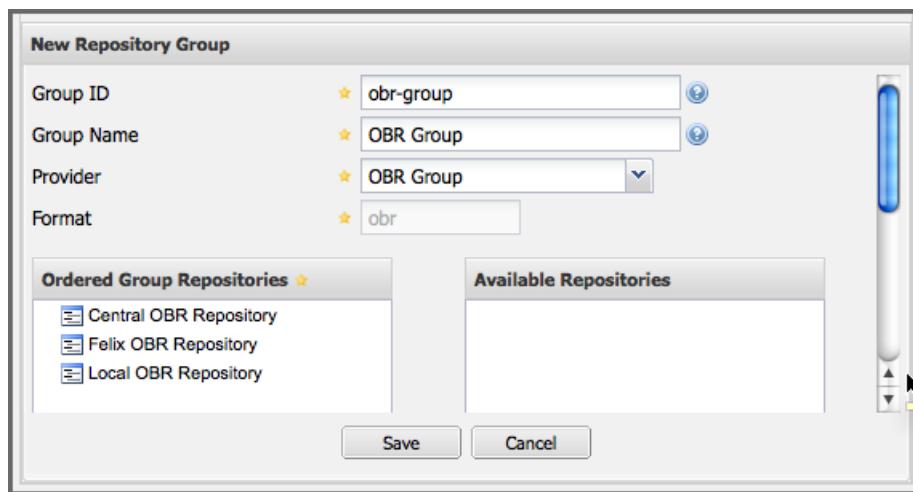


Figure 14.4: Creating a New OSGi Bundle Repository Group

## Chapter 15

# P2 Repositories

### 15.1 Introduction

Nexus supports the P2 Repository format. The P2 repository format is a provisioning platform for Eclipse components. For more information about the P2 repository format, see the [Equinox P2 documentation](#) on the Eclipse Wiki.

The following sections detail the procedures for creating and managing P2 repositories.

Nexus Professional has P2 support installed by default. Prior to any usage in Nexus OSS the Nexus P2 Bridge Plugin and the Nexus P2 Repository Plugin need to be installed. You can download the bundle.zip files for your specific Nexus version from the Central Repository:

- [Nexus P2 Repository Plugin](#)
- [Nexus P2 Bridge Plugin](#)

Extract the two files into `sonatype-work/nexus/plugin-repository` and restart Nexus.

---

## 15.2 Proxy P2 Repositories

Nexus can proxy a P2 Repository. To create a new proxy P2 repository:

1. Click *Repositories* in the *Views/Repositories* menu.
2. Click the *Add..* button above the list of repositories, and choose *Proxy Repository* from the drop-down of repository types.
3. In the New Proxy Repository window,
  - a. Select *P2* as the *Provider*.
  - b. Supply a *Repository ID* and a *Repository Name*.
  - c. Enter the URL to the remote P2 repository as the *Remote Storage Location*.
  - d. Click *Save*.

Figure 15.1 provides a sample configuration used to create a proxy of the Indigo Simultaneous Release P2 repository.

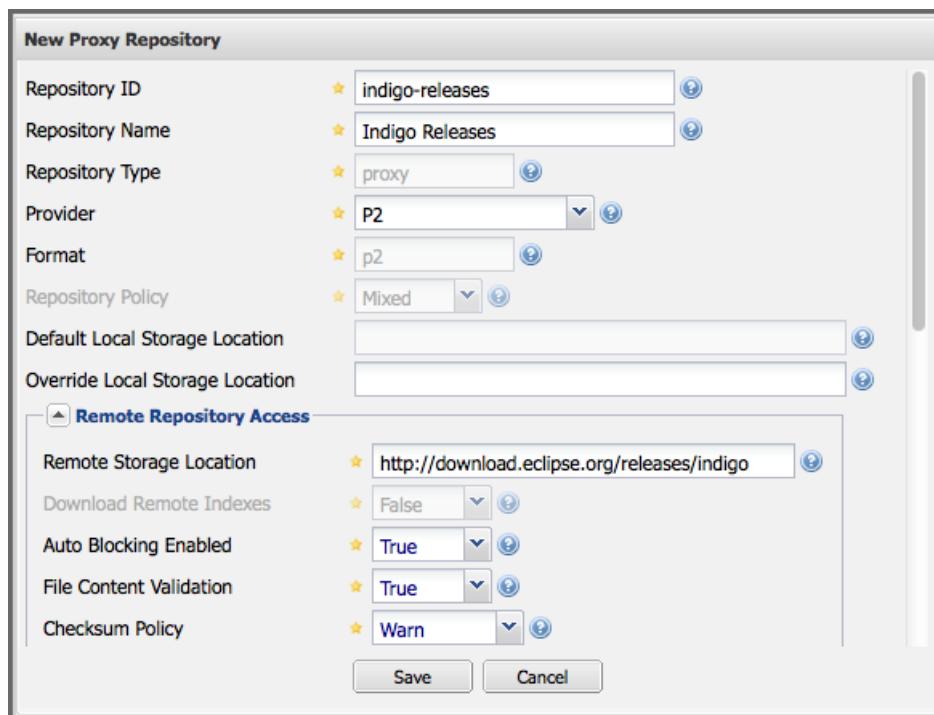


Figure 15.1: Creating a P2 Proxy Repository

### 15.3 Grouping P2 Repositories

Just like Nexus can group Maven repositories and OBR repositories, Nexus can also be configured to group P2 Repositories. To group P2 repositories:

1. Click *Repositories* in the *Views/Repositories* menu.
2. Click the *Add..* button above the list of repositories, and choose *Repository Group* from the drop-down of repository types.
3. In the New Repository Group window,
  - a. Select *P2* as the *Provider*.
  - b. Drag and drop one or more P2 repositories into the new group.

- c. Supply a *Group ID* and a *Group Name*.
- d. Click *Save*.

Figure 15.2 shows an example of a repository group that contains two P2 proxy repositories.

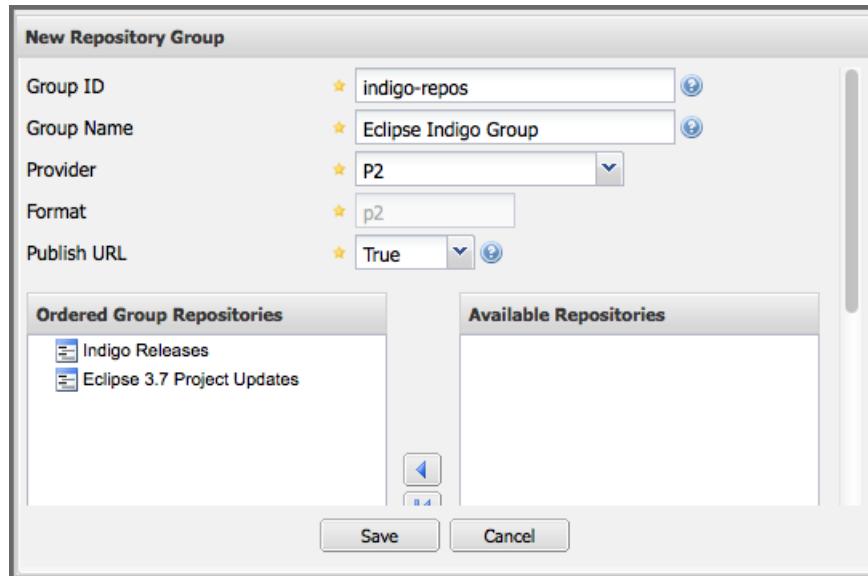


Figure 15.2: Creating a New P2 Repository Group

## Chapter 16

# .NET Package Repositories with NuGet

### 16.1 Introduction

With the creation of the [NuGet](#) project, a package management solution for .NET developers has become available. Similar to Apache Maven dependency management for Java developers, NuGet makes it easy to add, remove, and update libraries and tools in Visual Studio projects that use the .NET Framework.

The project websites at [www.nuget.org](http://www.nuget.org) and [nuget.codeplex.com](http://nuget.codeplex.com) host tool downloads and detailed documentation as well as links to further resources and provide a repository and features to upload your open source NuGet packages. With the NuGet Gallery a repository of open source libraries and tools is available and the need for repository management arises.



#### Important

With the release of Nexus 2.9, NuGet support is available in Nexus Professional and Nexus Open Source.

---

Nexus supports the NuGet repository format for hosted and proxy repositories. Nexus also supports aggregation of NuGet repositories and conversion of other repositories containing .nupkg components to the NuGet format. This allows you to improve collaboration and control, while speeding up .NET development, facilitating open source libraries and sharing of internal component across teams. When

---

you standardize on a single repository for all your development and use it for internal components as well, you will get all the benefits of Nexus when working in the .NET architecture.

To share a library or tool with NuGet, you create a NuGet package and store it in the Nexus-based NuGet repository. Similarly, you can use packages others have created and made available in their NuGet repositories by proxying them or downloading the packages and installing them in your own hosted repository for third party packages.

The NuGet Visual Studio extension allows you to download the package from the repository and install it in your Visual Studio project or solution. NuGet copies everything and makes any required changes to your project setup and configuration files. Removing a package will clean up any changes as required.

---

**Tip**

Using NuGet repositories benefits from a larger memory size available to Nexus. This memory allocation can be configured in `wrapper.conf` as documented in Section [3.5](#).

---

## 16.2 NuGet Proxy Repositories

The NuGet Gallery is the central repository used by all package authors and consumers. To reduce duplicate downloads and improve download speeds for your developers and CI servers, you should proxy the NuGet Gallery with Nexus. If you use other external repositories, you should also proxy these.

To proxy an external NuGet repository, you simply create a new *Proxy Repository* as documented in Section [6.2](#). The *Provider* has to be set to *NuGet*. The *Remote Storage Location* has to be set to the URL of the remote repository you want to proxy. The URL for the main NuGet Gallery repository is

```
https://www.nuget.org/api/v2/
```

A complete configuration for proxying the NuGet Gallery is visible in Figure [16.1](#).

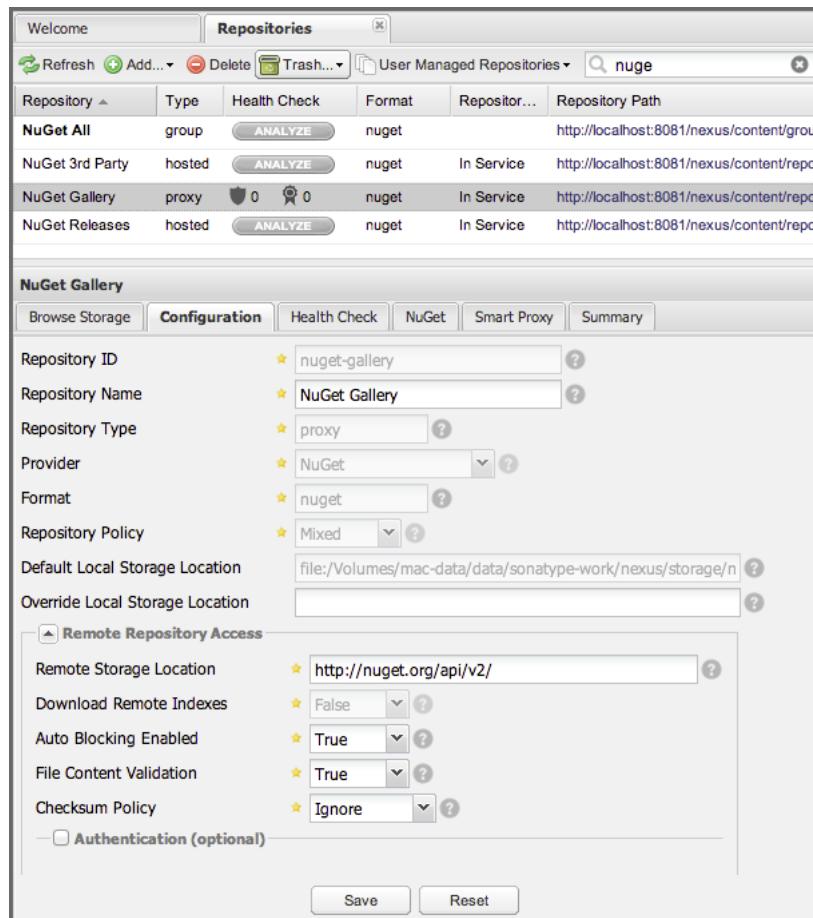


Figure 16.1: NuGet Proxy Repository Configuration for the NuGet Gallery

The repository configuration for a NuGet proxy repository has an additional tab titled *NuGet* as visible in Figure 16.2. It displays the *Package Source URL* that is URL where the repository is available as a NuGet repository.

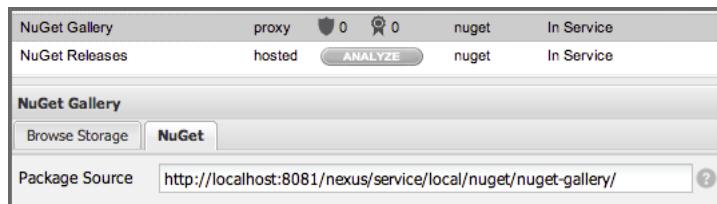


Figure 16.2: NuGet Gallery with Package Source URL

By default, searches in NuGet repositories in Nexus will be passed through to the remote repositories, and the search results are merged with internal search results and included in an internally managed index. This merging has to make some assumptions to generate component counts.

The accuracy of searches for components available in a remote repository can benefit from an index download. In most use cases, this will not be necessary. If you need a completely accurate view of available components and counts, you can create a scheduled tasks to download the remote index. Create a new task with the task type *Download NuGet Feed* in the *Scheduled Tasks* administration section.

By default the task incrementally downloads information about the latest version of published packages. If you want details about all versions, you can select the *Fetch all versions?* checkbox.

The NuGet Gallery is, at this stage, over 1GB large and downloading the feed will therefore take a little while. In addition, the index will be rather large and having the complete index in your Nexus NuGet database negatively impacts search performance.



#### Warning

Prior to Nexus 2.9 the scheduled task to download the feed was created automatically, since it was required then. This is no longer necessary and most users should remove the task and carefully consider the implications of running the task.

## 16.3 NuGet Hosted Repositories

A hosted repository for NuGet can be used to upload your own packages as well as third-party packages. It is good practice to create two separate hosted repositories for these purposes.

To create a NuGet hosted repository, simply create a new *Hosted Repository* and set the *Provider* to NuGet. A sample configuration for an internal releases NuGet hosted repository is displayed in Figure 16.3.

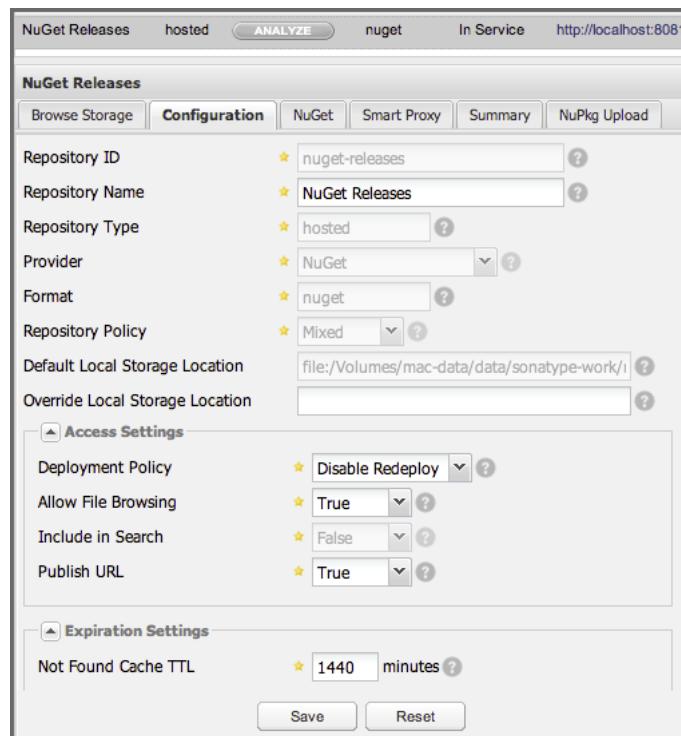


Figure 16.3: Example Configuration for a NuGet Hosted Repository for Release Packages

Besides the *NuGet* tab, the configuration for the repository has a *NuPkg Upload* tab as displayed in Figure 16.4 that allows you to manually upload one or multiple packages.

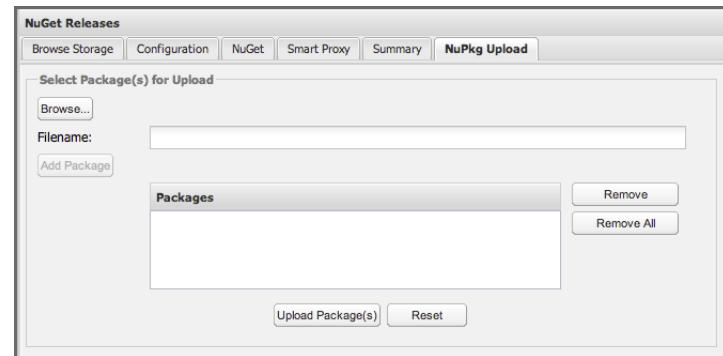


Figure 16.4: The NuPkg Upload Panel for a Hosted NuGet Repository

The NuGet feed is immediately updated as packages are deployed or deleted from the host repository. To rebuild the feed for a hosted NuGet repository you can manually schedule a *Rebuild NuGet Feed* task.

## 16.4 NuGet Virtual Repositories

If you have deployed NuGet packages to a Maven repository in the past, you can expose them to Visual Studio by creating a virtual repository as documented in Section 6.2 and setting the *Format* to *NuGet Shadow Repository*. The setup displayed in Figure 16.5 shows a virtual repository configured to expose the content of the regular Maven *Releases* repository as a NuGet repository, so that NuGet can access any NuGet packages deployed to the releases repository.

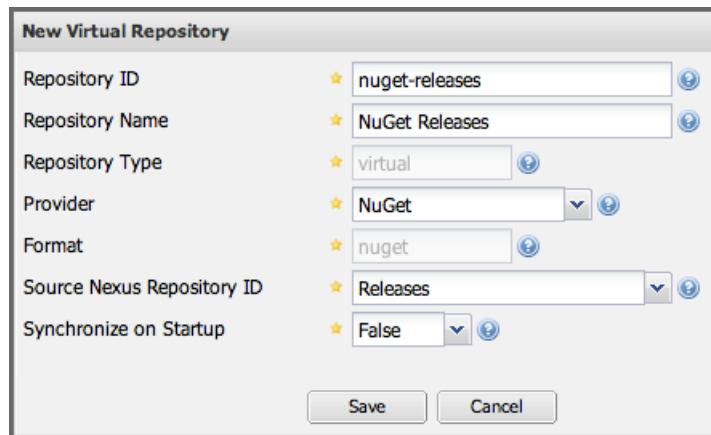


Figure 16.5: A Virtual NuGet Repository for the Releases Repository

The NuGet feed is immediately updated as packages are deployed or deleted from the shadowed repository. To rebuild the feed for a virtual NuGet repository, you can manually schedule a *Synchronize Shadow Repository* task.

## 16.5 NuGet Group Repositories

A repository group is the recommended way to expose all your NuGet repositories from Nexus to your users, without needing any further client side configuration. A repository group allows you to expose the aggregated content of multiple proxy and hosted repositories with one URL to your tools. This is possible for NuGet repositories by creating a new *Repository Group* with the *Format* set to NuGet as documented in Section 6.3.

A typical, useful example would be to group the proxy repository that proxies the NuGet Gallery, a NuGet, hosted repository with internal software packages and another NuGet, hosted repository with third-party packages.. The configuration for such a setup is displayed in Figure 16.6.

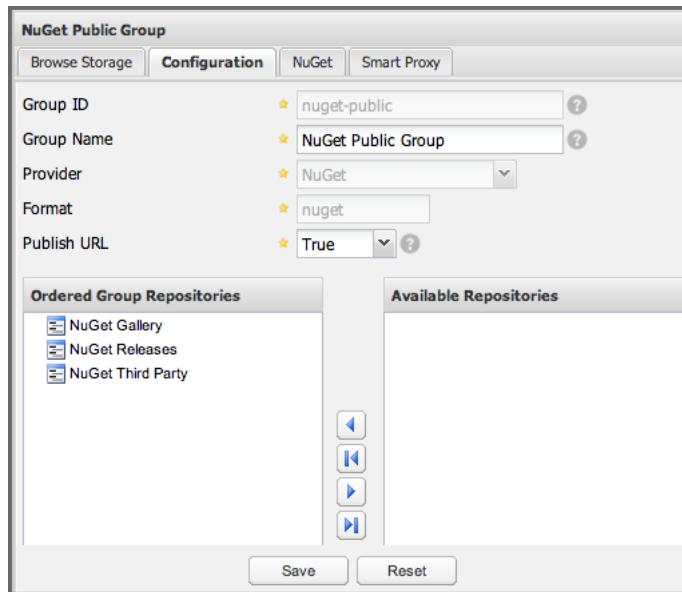


Figure 16.6: A Public NuGet Group Combining a Proxy and Two Hosted Repositories

Using the *Repository Path* of the repository group as your NuGet repository URL in your client tool will give you access to the packages in all three repositories with one URL. Any new packages added as well as any new repositories added to the group will automatically be available.

## 16.6 Accessing Packages in Repositories and Groups

Once you have set up your hosted and proxy repositories for NuGet packages, and potentially created a repository group, you can access them with the `nuget` tool on the command line. Copy the *Package Source URL* from the *NuGet* tab of the repository/group configuration you want to access and add it to `nuget` on the command line with e.g.:

```
nuget sources add -name NuGetNexus -source http://localhost:8081/nexus/ ↳  
    service/local/nuget/nuget-public
```

Replace `localhost` with the public hostname or URL of your Nexus server and `nuget-public` with the name of the repository you want to proxy. Ideally, this will be your NuGet group.

After this source was added, you can list the available packages with the command `nuget list`.

Access to the packages is not restricted by default. If access restrictions are desired, you can [configure Nexus security](#) directly or via [LDAP/Active Directory external role mappings](#) combined with [repository targets](#) for fine grained control. Authentication from NuGet is then handled via NuGet API keys as documented in Section [16.7](#).

## 16.7 Deploying Packages to NuGet Hosted Repositories

In order to authenticate a client against a NuGet repository, NuGet uses an API key for deployment requests. These keys are generated separately on request from a user account on the NuGet gallery and can be regenerated at any time. At regeneration, all previous keys generated for that user are invalid.

### 16.7.1 Creating a NuGet API-Key

For usage with Nexus, NuGet API keys are only needed when packages are going to be deployed; therefore, API key generation is by default not exposed in the user interface to normal users. Only users with the *Deployer* role have access to the API keys.

Other users that should be able to access and create an API key have to be given the *Nexus API-Key Access* role in the *Users* security administration.

In addition, the *NuGet API-Key Realm* has to be activated. To do this, simply add the realm to the selected realms in the *Security Settings* section of the *Server* configuration available in the *Administration* submenu of the left-hand navigation *Nexus* panel.

Once this is set up, you can view as well as reset the current *Personal API Key* in the *NuGet* tab of any NuGet proxy or hosted repository as visible in Figure [16.7](#)

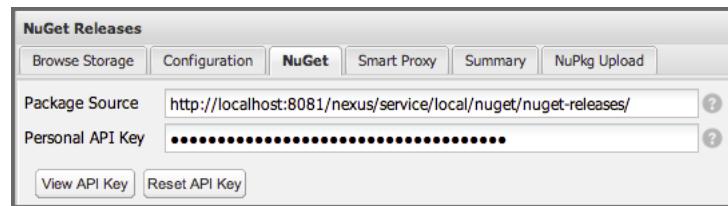


Figure 16.7: Viewing and Resetting the NuGet API Key in the NuGet Configuration Tab

### 16.7.2 Creating a Package for Deployment

Creating a package for deployment can be done with the `pack` command of the `nuget` command line tool or within Visual Studio. Detailed documentation can be found on the [NuGet website](#).

### 16.7.3 Deployment with the NuPkg Upload User Interface

Manual upload of one or multiple packages is done on the *NuPkg Upload* tab of the repository displayed in Figure 16.4. Press the *Browse* button to access the package you want to upload on the file system and press *Add Package*. Repeat this process for all packages you want upload, and press *Upload Package(s)* to complete the upload.

### 16.7.4 Command line based Deployment to a Nexus NuGet Hosted Repository

Alternatively to manual uploads, the `nuget` command line tool allows you to deploy packages to a repository with the `push` command. The command requires you to use the *API Key* and the *Package Source* path. Both of them are available in the NuGet tab of the hosted NuGet repository to where you want to deploy. Using the `delete` command of `nuget` allows you to remove packages in a similar fashion.

Further information about the command line tool is available in the [on-line help](#).

## 16.8 Integration of Nexus NuGet Repositories in Visual Studio

In order to access a Nexus NuGet repository or preferably all Nexus NuGet repositories exposed in a group, you provide the *Repository Path* in the Visual Studio configuration for the *Package Sources* of the *Package Manager* as displayed in Figure 16.8.

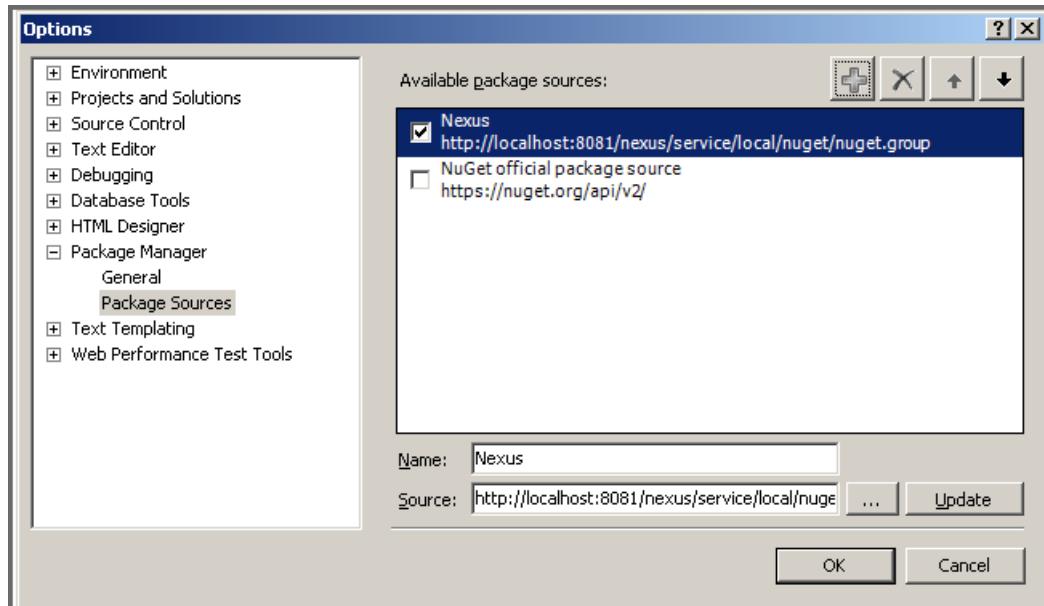


Figure 16.8: Package Source Configuration for the Package Manager in Visual Studio

With this configuration in place, all packages available in your Nexus NuGet repository will be available in the *Package Manager* in Visual Studio.

## Chapter 17

# Node Packaged Modules and npm Registries

### 17.1 Introduction

The command line tool `npm` is a package management solution for Javascript-based development. It is used to create and use *node packaged modules* and is built into the popular Javascript platform `Node.js`, which is mostly used for server-side application development.

The npm registry at <https://www.npmjs.org/> is the default package registry, from which components can be retrieved. It contains a large number of open source packages for Node.js based server-side application development, build tools like `bower` or `grunt` and many other packages for a variety of use cases.

Nexus supports the npm registry format for proxy repositories. This allows you to take advantage of the packages in the npm registry and other public registries without incurring repeated downloads of packages, since they will be proxied in Nexus.

In addition, Nexus supports running your own private registry - also known as a hosted repository using the npm format. You can share internally developed, proprietary packages within your organization via these private registries allowing you to collaborate efficiently across development teams with a central package exchange and storage location.

---

To simplify configuration Nexus supports aggregation of npm registries. This allows you to expose all the external package from the npm registry and other public registries as well as the private registries as one registry, which greatly simplifies client configuration.

To share a package or tool with npm, you create a npm package and store it in the Nexus-based npm registry. Similarly, you can use packages others have created and made available in their NPM repositories by proxying them or downloading the packages and installing them in your own private registry for third party packages.

**Important**

npm support is a feature of Nexus 2.10 and higher and is available in Nexus Professional and Nexus Open Source and requires npm version 1.4 and above.

---

## 17.2 Proxying npm Registries

To reduce duplicate downloads and improve download speeds for your developers and CI servers, you should proxy the registry hosted at <https://registry.npmjs.org>. It is accessed directly by npm out-of-the-box. You can also proxy any other registries you require.

To proxy an external npm registry, you simply create a new *Proxy Repository* as documented in Section 6.2. The *Provider* has to be set to NPM. The *Remote Storage Location* has to be set to the URL of the remote repository you want to proxy. The official URL for the main npm registry is

```
https://registry.npmjs.org
```

A complete configuration for proxying the default npm registry is visible in Figure 17.1.

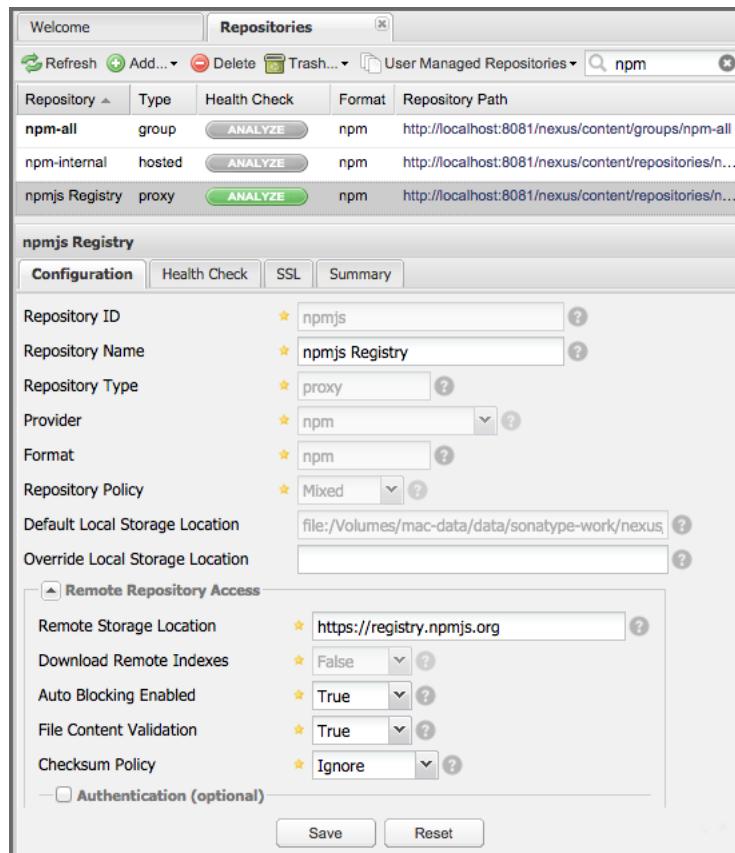


Figure 17.1: Proxy Repository Configuration for the npm Registry



### Warning

Browsing the registry storage or the remote registry and searching for packages in Nexus is not supported

## 17.3 Private npm Registries

A private npm registry can be used to upload your own packages as well as third-party packages. You can create a private npm registry by setting up a hosted repository with the npm format in Nexus. It is good practice to create two separate hosted repositories for these purposes.

To create a hosted repository with npm format, simply create a new *Hosted Repository* and set the *Provider* to npm as documented in Section 6.2. A sample configuration for an internal releases npm hosted repository is displayed in Figure 17.2.

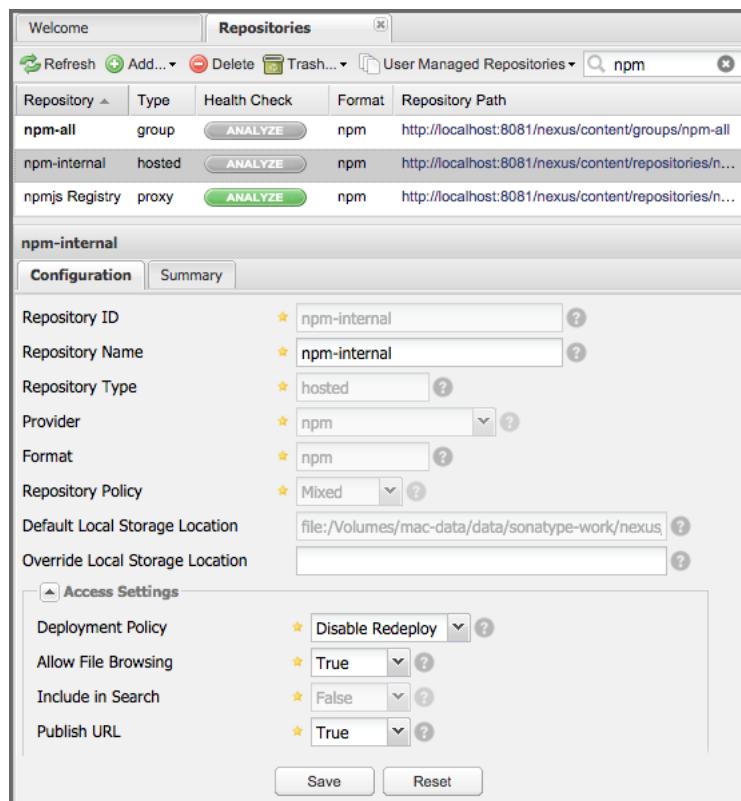


Figure 17.2: Example Configuration for a Private npm Registry

The npm registry information is immediately updated as packages are deployed or deleted from the repository.

**Warning**

Browsing the registry storage or searching for packages in Nexus not yet supported.

---

The scheduled tasks to recreate the npm metadata database based on the components in a hosted repository and to back up the database are documented in Section [6.5](#).

## 17.4 Grouping npm Registries

A repository group is the recommended way to expose all your npm registries repositories from Nexus to your users, without needing any further client side configuration. A repository group allows you to expose the aggregated content of multiple proxy and hosted repositories with one URL to npm and other tools. This is possible for npm repositories by creating a new *Repository Group* with the *Provider* set to npm as documented in Section [6.3](#).

A typical, useful example would be to group the proxy repository that: proxies the npm registry, a npm, hosted repository with internal software packages and another npm, hosted repository with third-party packages. The configuration for such a setup is displayed in Figure [17.3](#).

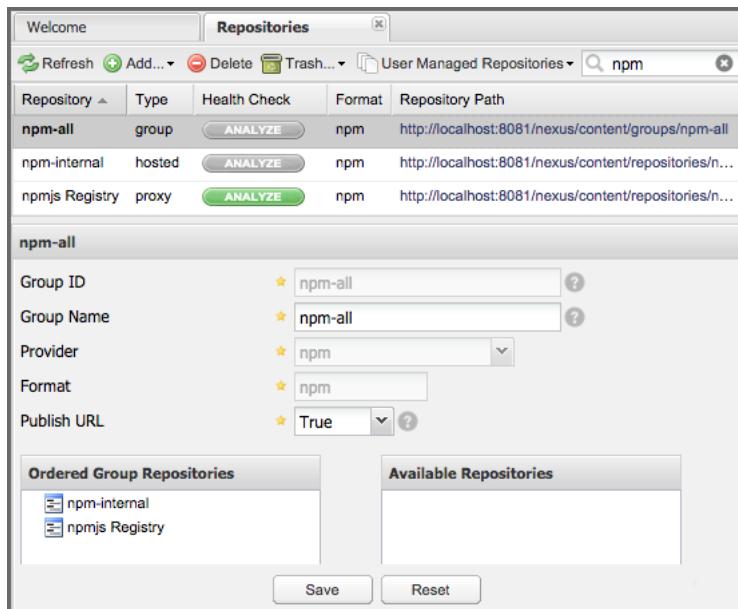


Figure 17.3: An npm Group Combining the npm Registry and Two Private Registries

Using the *Repository Path* of the repository group as your npm repository URL in your client tool will give you access to the packages in all three repositories with one URL. Any new packages added as well as any new repositories added to the group will automatically be available.

## 17.5 Configuring npm

Once you have set up your hosted and proxy repositories for npm packages, and created a repository group to merge them, you can access them with the `npm` tool on the command line as one registry.

You can configure the registry used by npm in your `.npmrc` file located in your user's home directory. If the file does not exist simply create it and add the registry configuration with the URL of your npm repository group. You can see the public URL of your group repository in the repository list in the *Repository Path* column.

### Registry configuration in `.npmrc`

```
registry = http://localhost:8081/nexus/content/groups/npm-all/
```

With this configuration any npm commands will use the new registry from Nexus. The command line output will reference the URLs. E.g.

```
$ npm install bower
npm install bower
npm http GET http://localhost:8081/nexus/content/groups/npm-all/bower
npm http 200 http://localhost:8081/nexus/content/groups/npm-all/bower
npm http GET http://localhost:8081/nexus/content/groups/npm-all/bower- ↵
  config
npm http GET http://localhost:8081/nexus/content/groups/npm-all/bower- ↵
  endpoint-parser
...
...
```

By default any anonymous user has read access to the repositories and repository groups. If desired, Nexus can be configured to require authentication by adding always-auth=true to the `.npmrc` file and adding the encoded authentication details as documented in Section [17.6](#).

## 17.6 Publishing npm Packages

Publishing your own packages to a npm hosted repository allows you to share packages across your organization or with external partners.

The `npm publish` command uses a `registry` configuration value to know where to publish your package. There are several ways to change the `registry` value to point at your Nexus hosted npm repository.

Since the `.npmrc` file usually contains a `registry` value intended only for getting new packages, a simple way to override this value is to provide a `registry` to the `publish` command:

```
npm publish --registry http://localhost:8081/nexus/content/repositories/ ↵
  npm-internal/
```

Alternately, you can edit your `package.json` file and add a `publishConfig` section:

```
"publishConfig" : {
  "registry" : "http://localhost:8081/nexus/content/repositories/npm- ↵
    internal/"
},
```

Publishing requires authentication. It can be configured by adding an `_auth` value to `.npmrc`. The value has to be generated by base64-encoding the string of `username:password`. You can create this encoded string with the command line call `openssl` e.g.: for the default `admin` user of Nexus:

```
echo -n 'admin:admin123' | openssl base64
```

Other tools for the encoding are `uuencode` or, for Windows users, `certutil`. To use `certutil` on Windows you need to put the credentials to be encoded into a file:

```
admin:admin123
```

Then run:

```
c:\certutil /encode in.txt out.txt
```

After this the base64 encoded credentials can be found in between the begin and end certificate lines in the output file:

```
-----BEGIN CERTIFICATE-----  
YWRtaW46YWRtaW4xMjM=  
-----END CERTIFICATE-----
```

Once you have the encoded credentials the value as well as author information can then be added to the `.npmrc` file:

```
init.author.name = Jane Doe  
init.author.email = jane@example.com  
init.author.url = http://blog.example.com  
# an email is required to publish npm packages  
email=jane@example.com  
_auth=YWRtaW46YWRtaW4xMjM=
```

---

**Tip**

Whatever tool you use to generate the encoded username and password string, try to encode the string `admin:admin123`, which should result in `YWRtaW46YWRtaW4xMjM=`. Another example for a valid setup is `jane:testpassword123` resulting in `amFuZTp0ZXN0cGFzc3dvcmQxMjM=`.

---

With this configuration you can run `npm publish` for your package. More information about package creation can be found on the [npm website](#).

---

Once a package is published to the private registry in Nexus, any other developers or build servers, that access Nexus via the repository group have instant access to the packages.

## Chapter 18

# Ruby, RubyGems and Gem Repositories

### 18.1 Introduction

For developers using the [Ruby](#) programming language, the `gem` tool serves as their package management solution. In fact, since version 1.9 of Ruby, it has been included as part of the default Ruby library. Packages are called *gems* and, just like all package managers, this allows for ease of use when distributing programs or libraries.

Of course, package management really only goes as far as improving distribution. A great feat certainly, but to really find success, a development community needs to exist. At the heart of every development community, especially those like Ruby, where open source projects are one of the most critical elements, the community needs a place to host and share their projects.

Enter [RubyGems hosted at rubygems.org](#) - the most popular and leading gem hosting service supporting the Ruby community. Here, a large variety of open source Ruby projects supply their gems for download to all users.

Ruby has been a successful platform for developers for a long time now. The popularity of Ruby and therefore the usage of gems and Gem repositories means that lots of teams are downloading and exchanging lots of components on a regular basis. Obviously, this can (and does) become a crunch on resources, not to mention a pain to manage.

---

Luckily Nexus supports Gem repositories. A user can connect to Nexus to download gems from RubyGems, create proxies to other repositories, and host their own or third-party gems. Any gem downloaded via Nexus needs to be downloaded from the remote repository, like RubyGems, only once and is then available internally via Nexus. Gems pushed to Nexus automatically become available to everyone else in your organization. Using Nexus as a proxy avoids the overhead of teams and individual developers having to repeatedly download components or share components in a haphazard and disorganized manner.

**Important**

Gem repository support is a feature of Nexus 2.11 and higher, and is available in all Nexus editions.

---

The following features are included as part of the Nexus Gem repository support:

- Proxy repository for connecting to remote Gem repositories and caching gems on Nexus to avoid duplicate downloads and wasted bandwidth and time
- Hosted repository for hosting gems package and providing them to your users
- Repository groups for merging multiple hosted and proxy Gem repositories and easily exposing them as one URL to all users

**Tip**

None of this functionality requires Ruby (or any extra tooling) to be installed on the operating system running Nexus. Ruby specific details are implemented using [JRuby](#) bundled with Nexus.

---

## 18.2 Proxying Gem Repositories

To reduce duplicate downloads and improve download speeds for your developers, continuous integration servers and other systems using `gem`, you should proxy the RubyGems repository and any other repositories you require.

To proxy an external Gem repository, like RubyGems, simply create a new *Proxy Repository* as documented in Section [6.2](#). The *Provider* has to be set to `Rubygems`. The *Remote Storage Location* has to be set to the URL of the remote repository you want to proxy. The official URL for Rubygems.org is

---

<https://rubygems.org>

This main configuration for proxying RubyGems is visible in Figure 18.1. Further configuration details are available in Section 6.2.

The screenshot shows the Nexus Repository Management interface. In the top navigation bar, the 'Repositories' tab is selected. Below it, a table lists several repositories:

Repository	Type	Format	Repository Path
Releases	hosted	maven2	http://localhost:8081/nexus/content/repositories/releases
RubyGems.org	proxy	rubygems	http://localhost:8081/nexus/content/repositories/rubygems-org
Site Internal	hosted	site	http://localhost:8081/nexus/content/sites/site-internal
Snapshots	hosted	maven2	http://localhost:8081/nexus/content/repositories/snapshots

Below the table, the configuration for the 'RubyGems.org' repository is displayed. The configuration tabs include Configuration, Health Check, SSL, Smart Proxy, and Summary. The 'Configuration' tab is active. The configuration fields are as follows:

- Repository ID: rubygems-org
- Repository Name: RubyGems.org
- Repository Type: proxy
- Provider: Rubygems
- Format: rubygems
- Repository Policy: Mixed
- Default Local Storage Location: file:/Volumes/mac-data/data/sonatype-work/nexus/storage/ri
- Override Local Storage Location: (empty)
- Remote Repository Access:
  - Remote Storage Location: https://rubygems.org

At the bottom of the configuration panel are 'Save' and 'Reset' buttons.

Figure 18.1: Proxy Gem Repository Configuration for RubyGems

If you are using Nexus Professional and are proxying a repository via HTTPS, you can get the certificate added to the Nexus truststore to simplify management using the the SSL tab of the repository configuration.

A scheduled task can be used to purge broken metadata of a proxy gem repository and can be configured as documented in Section 6.5.

## 18.3 Private Hosted Gem Repositories

A private Gem repository on Nexus can be used as target to push your own gems as well as third-party gems and subsequently provide them to your users. It is good practice to create two separate hosted Gem repositories for internal and third-party gems.

To create a hosted Gem repository, simply create a new *Hosted Repository* and set the *Provider* to Rubygems as documented in Section 6.2. A sample configuration for an internal hosted Gem repository is displayed in Figure 18.2.

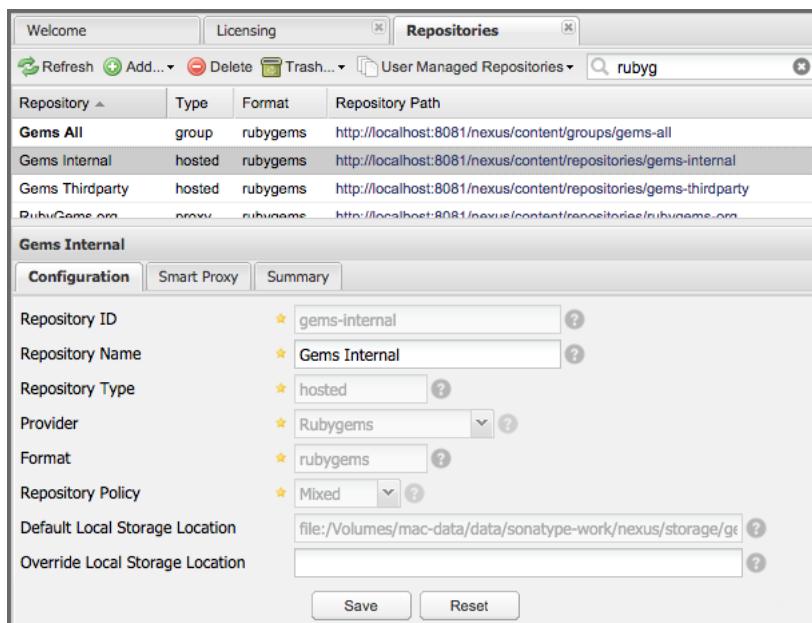


Figure 18.2: Example Configuration for a Private Gem Repository

The Gem repository information is immediately updated as gems are pushed to the repository or deleted from it.

A scheduled task can be used to rebuild the metadata of a hosted gem repository and can be configured as documented in Section 6.5.

## 18.4 Grouping Gem Repositories

A repository group is the recommended way to expose all your Gem repositories from Nexus to your users, without needing any further client side configuration after initial setup. A repository group allows you to expose the aggregated content of multiple proxy and hosted Gem repositories with one URL to gem and other tools. This is possible for Gem repositories by creating a new *Repository Group* with the *Provider* set to Rubygems as documented in Section 6.3.

A typical, useful example would be to group the proxy repository that proxies the RubyGems repository, a hosted Gem repository with internal software gems, and another hosted Gem repository with third-party gems. The configuration for such a setup is displayed in Figure 18.3.

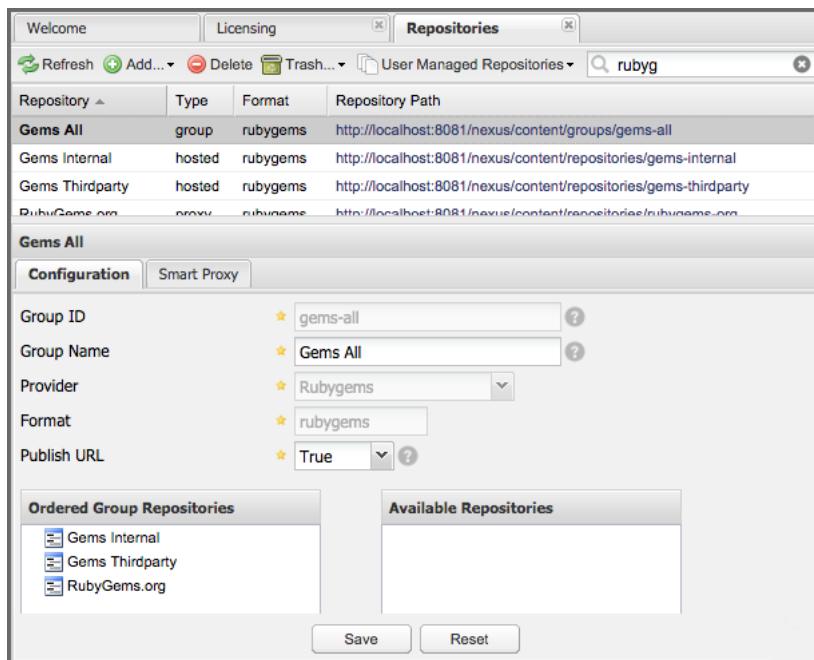


Figure 18.3: A Gem Repository Group Combining the RubyGems Proxy Repository and Two Private Gem Repositories

Using the *Repository Path* of the repository group as your Gem repository URL in your client tool gives you access to the gems in all three repositories with one URL.

Any new gem added to the remote proxy Gem repositories or the hosted Gem repositories becomes immediately available to all users of the Gem repository group. Adding a new proxy Gem repository to the group makes all gems immediately available to the users as well.

## 18.5 Using Gem Repositories

Once you have configured Nexus with the Gem repository group, you can add it to your configuration for the `gem` command line tool.

You can add the URL of a Nexus gems repository or better the Gem repository group using the *Repository Path* from the repository list with a command like

```
gem sources --add http://localhost:8081/nexus/content/groups/gems-all/
```

In order to take full advantage of Nexus and the proxying of gems, you should remove any other sources. By default `https://rubygems.org/` is configured and this can be removed with

```
$ gem sources --remove https://rubygems.org/  
https://rubygems.org/ removed from sources
```

Subsequently you should clear the local cache with

```
$ gem sources -c  
*** Removed specs cache ***
```

To check a successful configuration you can run

```
$ gem sources  
*** CURRENT SOURCES ***  
  
http://localhost:8081/nexus/content/groups/gems-all/
```

With this setup completed any installation of new gems with `gem install GEMNAME` e.g., `gem install rake` will download from Nexus.

By default read access to Nexus is available to anonymous access and no further configuration is necessary. If your Nexus server requires authentication, you have to add the *Basic Auth* authentication details to the sources configuration:

```
$ gem sources --add  
http://myuser:mypassword@localhost:8081/nexus/content/repositories/gems- ↵  
all/
```

If you are using the popular **Bundler** tool for tracking and installing gems, you need to install it with `gem`:

```
$ gem install bundle  
Fetching: bundler-1.7.7.gem (100%)  
Successfully installed bundler-1.7.7  
Fetching: bundle-0.0.1.gem (100%)  
Successfully installed bundle-0.0.1  
Parsing documentation for bundle-0.0.1  
Installing ri documentation for bundle-0.0.1  
Parsing documentation for bundler-1.7.7  
Installing ri documentation for bundler-1.7.7  
Done installing documentation for bundle, bundler after 4 seconds  
2 gems installed
```

To use Nexus with Bundler, you have to configure the Gem repository group as a mirror:

```
$ bundle config mirror.http://rubygems.org  
http://localhost:8081/nexus/content/repositories/gems-all
```

You can confirm the configuration succeeded by checking the configuration:

```
$ bundle config  
Settings are listed in order of priority.  
The top value will be used.  
mirror.http://rubygems.org  
Set for the current user (/Users/manfred/.bundle/config):  
"http://localhost:8081/nexus/content/repositories/gems-all"
```

With this configuration completed, you can create a `Gemfile` and run `bundle install` as usual and any downloads of gem files will be using the Nexus Gem repository group configured as a mirror.

## 18.6 Pushing Gems

At this point you have set up the various Gem repositories in Nexus (proxy, hosted, and group), and are successfully using them for installing new gems on your systems. A next step can be to push gems to

---

hosted Gem repositories to provide them to other users. All this can be achieved on the command line with the features of the `nexus` gem.

The `nexus` gem is available at RubyGems and provides features to interact with Nexus including pushing gems to a hosted Gem repository including the necessary authentication.

You can install the `nexus` gem with

```
$ gem install nexus
Fetching: nexus-1.2.1.gem (100%)
...
Successfully installed nexus-1.2.1
Parsing documentation for nexus-1.2.1
Installing ri documentation for nexus-1.2.1
Done installing
```

After successful installation, you will get a confirmation with some documentation and a success message and you can push your gem to a desired repository. The initial invocation will request the URL for the GEM repository and the Nexus credentials needed for deployment. Subsequent pushes will use the cached information.

```
$gem nexus example-1.0.0.gem
Enter the URL of the rubygems repository on a Nexus server
URL: http://localhost:8081/nexus/content/repositories/gems-internal
The Nexus URL has been stored in ~/.gem/nexus
Enter your Nexus credentials
Username: admin
Password:
Your Nexus credentials have been stored in /Users/manfred/.gem/nexus
Uploading gem to Nexus...
Created
```

By default pushing an identical version to the repository, as known as redeployment, is not allowed in a hosted Gem repository. If desired this configuration can be changed, although we suggest to change the version for each new deployment instead.

The `nexus` gem provides a number of additional features and parameters. You can access all the documentation with

```
$ gem help nexus
```

E.g. you can access a list of all configured repositories with

---

```
$gem nexus --all-repos  
  
DEFAULT:  
http://localhost:8081/nexus/content/repositories/gems-internal
```

## Chapter 19

# RPM Packages and YUM Repositories

### 19.1 Introduction

RPM packages and the RPM package manager solution yum are used as the default application package manager on Linux based operating systems such as Red Hat, CentOS, Fedora, Oracle Linux, SUSE, openSUSE, Scientific Linux and others.

The yum repository support of Nexus allows you to expose RPM packages hosted in a Maven repository in the yum repository format. It generates the yum metadata, so that systems with yum support can use Nexus as a software package repository.

This enables a build and deployment pipeline for Java or other JVM-based applications via Maven repositories to Linux computers. E.g., a Java Enterprise Archive (EAR) or Web Archive (WAR) or some other application is deployed to a Maven repository. The deployment is performed by a CI server build using Maven or other build systems or as a manually run deployment. Once Nexus hosts the application RPM package, it can be retrieved via yum for installation and updates on testing and production systems. The metadata of the RPM package can additionally trigger installation of other required packages including e.g. a Java runtime or an application server.

---

## 19.2 Installation and Requirements

Yum support is bundled with all Nexus editions and no further installation steps are required. It relies on the commands `createrepo` and `mergerepo` to be installed on the operating system running the Nexus server and to be available on the path. Documentation about these commands can be found on the [createrepo website](#). Typically `createrepo` is installed on RPM-based Linux distributions and as such they are suitable to run Nexus with yum support. If desired the path to the commands can be configured in the Nexus user interface.

If your RPM-based system does not have this command you can install it by running

```
yum install createrepo
```

with a sufficiently privileged user.

## 19.3 Configuration

Yum related configuration is done with the *Capabilites* management of Nexus as documented in Section [6.6](#).

The capability *Yum: Configuration* allows you to enable or disable yum support. It can only be enabled successfully, if the `createrepo` and the `mergerepo` commands can be found by Nexus. By default Nexus will look for them on the path. The configuration settings *Path of "createrepo"* and *Path of "mergerepo"* allow you to alternatively configure a specific absolute path.

The parameter *Max number of parallel threads* defaults to ten and defines how many threads can be used to manage the yum repositories with the `createrepo` and the `mergerepo` commands.

You need to ensure that this capability is enabled, before proceeding with your repository specific configuration. The *Status* tab of the capability displays the detected versions for `createrepo` and `mergerepo` and details any problems as applicable.

### 19.3.1 Configure Hosted Yum Repositories

To expose a Maven repository like *Releases* via yum, press the *New* button in the capabilities configuration tab and select *Yum: Generate Metadata* from the *Type* drop down in the dialog displayed in Figure 19.1.

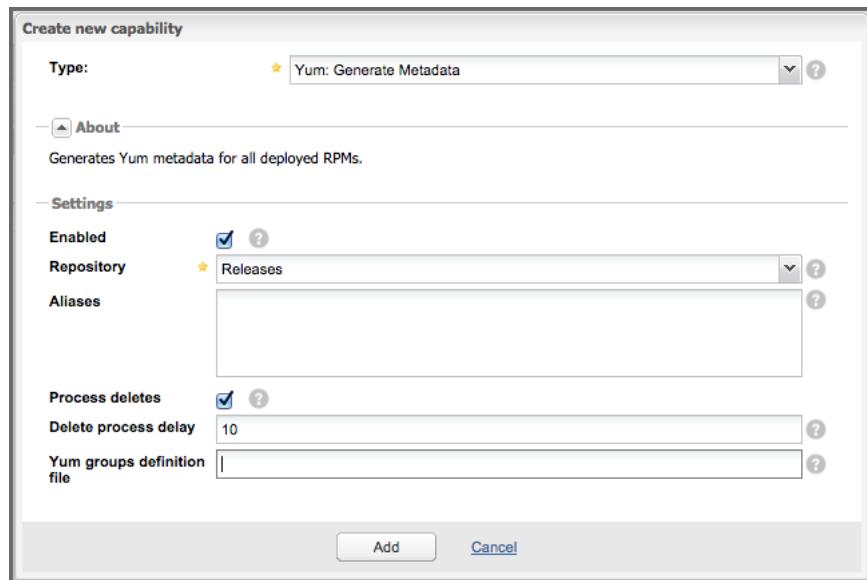


Figure 19.1: Yum Configuration for the Hosted *Releases* Repository

The *Repository* drop down allows you to select the hosted Maven repository. Release as well as snapshot policy repositories can be configured. Once configured, any RPM package added to the hosted Maven repository is available via yum. The same URL of the repository used for Maven based access e.g., `http://localhost:8081/nexus/content/repositories/releases` and displayed in the repository administration area list, can be used as the URL for a yum repository in the yum configuration.

The Nexus yum integration supports versioned views on a repository. The URL `http://localhost:8081/nexus/` exposes a yum repository with all packages with version 1.2.3 in the `releases` repository. A custom repodata folder is available at the context.

The *Aliases* field can be used to define alternative access paths to specific versions. For example, you can configure alias values of

```
production=1.2,testing=2.0
```

These values would in turn expose the version 1.2 under a URL like `http://localhost:8081/nexus/service/local/yum/repos/releases/test/1.2` and the version 2.0 at `http://localhost:8081/nexus/service/local/yum/repos/releases/test/2.0`. Using these URLs in the yum configuration on the target servers as a static URL enables upgrades to new versions by simply changing the alias e.g. to `production=1.3` and running a yum update command on the target server.

Besides maintaining the aliases in the capability administration, it is possible to create or update an alias in the command line:

```
curl -d "1.0" --header "Content-Type: text/plain" http://localhost:8081/nexus/service/local/yum/alias/releases/development/
```

Usage of the alias-based URL is done via the normal yum configuration e.g. with a file `/etc/yum.repos.d/nexus-production.repo` and the following content:

```
[nexus-production]
name=Nexus Production Repository
baseurl=http://localhost:8081/nexus/service/local/yum/repos/releases/
production/
enabled=1
protect=0
gpgcheck=0
metadata_expire=30s
autorefresh=1
type=rpm-md
Promote RPM through Stages
```

By deploying new versions and switching alias associations to the versions, a controlled roll out of new versions of RPM archives to target servers can be achieved.

The configuration options *Process deletes* and *Delete process delay* can be used to enable updates to the yum metadata, following delete operations of rpm packages in the Maven repository.

The *Yum groups definition file* configuration allows you to configure a path to a package groups configuration file. This file is typically named `comps.xml` and can be used to define a group of RPM packages. The groups can then be managed with commands such as `yum grouplist`, `yum groupinstall` and `yum groupremove`.

Once the capability is saved, the *Status* tab displays an example yum configuration for accessing the repository. Each RPM deployed to the repository causes Nexus to update the yum metadata immediately.

The metadata used by yum is available in the `repodata` context e.g., at `.../nexus/content/repositories/repo/repodata/`

---

in the following files. Apart from the `repomd.xml` file, the files are prepended with a unique hash value as part of the name to avoid caching issues:

**repomd.xml**

This XML file contains information about the other metadata files.

**hash-primary.xml.gz**

This zipped XML file describes the primary metadata of each RPM archive in the repository.

**hash-filelists.xml.gz**

This zipped XML file describes all the files contained within each RPM archive.

**hash-other.xml.gz**

This zipped XML file contains further, miscellaneous information regarding each RPM archive.

### 19.3.2 Proxying Repositories

The Nexus yum integration is able to proxy yum-enabled Maven repositories from remote Nexus servers. The metadata in these repositories contains absolute URLs, which will cause yum to use these URLs. The capability *Yum: Proxy Metadata* can be configured on such a proxy repository. It will cause the URLs in the metadata to be rewritten and corrected for the current Nexus server.

This allows the proxy repositories to be part of a repository group and expose the correct yum metadata via the merged metadata creation on the group.

### 19.3.3 Configure Repository Group for yum

To expose a Maven repository group to yum, simply add a new capability with the type *Yum: Merge Metadata* and select the repository group in the *Group* drop down. Figure 19.2 shows the *Settings* tab for the *Public Repositories* configured for yum.



Figure 19.2: Yum Configuration for the Hosted *Releases* Repository

This configuration causes Nexus to merge the yum metadata of all repositories in the repository group. Metadata generation has to be configured for the individual repositories desired to be exposed as part of the group. The URL of the repository group, can now be used as the URL for a yum repository in the yum configuration, since the same metadata files are being maintained and exposed via the `repodata` context like in a hosted repository.

#### 19.3.4 Scheduled Tasks

The yum support in Nexus includes a [scheduled task](#) called *Yum: Generate Metadata* that can be run to generate yum metadata with `createrepo` for a specific repository.

Typically this task does not need to be run, however it can be useful when RPM files already exist in a repository or are deployed in some external mode that requires a manually triggered update of the metadata.

The *Optional Output Directory* parameter can be used to get the metadata created in a different folder from the default `repo-data` in repository root.

The parameter *Single RPM per directory* is activated by default and causes the task to take only one RPM file per directory in the Maven repository into account when creating the yum metadata.

The *Full Rebuild* parameter can be activated to force Nexus to traverse all directories in the repository in order to find the RPM files that need to be taken into account for the metadata creation. This option is off by default and causes Nexus to take the existing metadata cache as a basis for the update.

## 19.4 Example Usages

The Nexus artifact upload to a hosted repository allows you to publish any RPM file to a Maven repository and subsequently expose it via the yum integration. This is a basic use case, that can be used to e.g., exposed third-party supplied RPM archives. The more advanced setup involves a Maven project that creates the RPM as detailed in this section.

The [RPM Maven Plugin](#) can be used to create an RPM package of a Java application and attach it as a secondary built artifact with the `attached-rpm` goal. An example plugin configuration for a `war` project can be found in [?simpara].

If your project includes a `distributionManagement` for the `releases` repository, a build with `mvn clean deploy`, causes the `war` as well as the `rpm` file to be uploaded to Nexus. With yum configured for the `releases` repository in Nexus, the RPM package can be consumed by any server configured to access the repository with yum.

Maven pom.xml snippet for configuring and attaching an RPM

```
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>rpm-maven-plugin</artifactId>
      <version>2.1</version>
      <executions>
        <execution>
          <id>build-rpm</id>
          <goals>
            <goal>attached-rpm</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <group>Applications/Internet</group>
        <copyright>EPL</copyright>
        <requires>
          <require>tomcat8</require>
        </requires>
        <mappings>
          <mapping>
            <directory>/var/lib/tomcat8/webapps/${project.build.finalName} <!--
              }</directory>
            <sources>
              <source>
                <location>${project.build.directory}/${project.build. -->
```

```
        finalName}>/location>
    </source>
</sources>
</mapping>
</mappings>
</configuration>
</plugin>
...

```

Now that Nexus hosts a RPM package with your Java web application in a yum repository, you can configure yum on the target server to retrieve it for installation. You have to configure yum to include the Nexus repository as a package source. Depending on your specific Linux distribution, file paths and tools for this configuration will differ. A typical example would be to create a new file e.g. `nexus.repo` in `/etc/yum.repos.d`. A sample configuration for the `public` group can be found in [Example yum source repository configuration](#).

### Example yum source repository configuration

```
[nexus-public]
name=Nexus Releases Repository
baseurl=http://yournexusservhost/nexus/content/groups/public
enabled=1
protect=0
gpgcheck=0
metadata_expire=30s
autorefresh=1
type=rpm-md
```

Once the configuration is added you can install or update any RPM packages from Nexus as usual with `yum install <packagename>` or `yum update <packagename>`. This includes any required dependencies like a servlet container or a Java runtime as declared in the RPM Maven Plugin configuration and therefore the RPM/yum metadata.

## 19.5 Staging with RPMs

The [Nexus Staging Suite](#) of Nexus Professional can be used with yum repositories allowing you to optimize the release process for your RPM packages.

The capability *Yum: Staging Generate Metadata* allows you to configure yum for a *Staging Profile*. Any staging repository created from a deployment via the staging profile is then automatically configured

---

as a yum repository. The *Aliases* configuration allows for the same mechanism as the capability *Yum: Generate Metadata* documented earlier.

The capability *Yum: Staging Merge Metadata* can be used to configure yum metadata creation for a build promotion profile and the attached repository groups.

If a staging repository or build promotion repository is configured for yum metadata generation and exposed via a repository group that is configured for yum metadata merging, the metadata from staging will be merged appropriately.

## Chapter 20

# Deploying Sites to Nexus

### 20.1 Introduction

Nexus includes a repository provider for hosting static websites - the *Site* format. Hosted repositories with this format can be used to hold a Maven-generated website. This chapter details the process of configuring a site repository and configuring a simple Maven project to publish a Maven-generated project site to an instance of Nexus.

### 20.2 Creating a New Maven Project

In this chapter, you will be creating a simple Maven project with a simple website that will be published to a Nexus *Site* repository. To create a new Maven project, use the archetype plugin's `archetype:generate` goal on the command line, and supply the following identifiers:

- groupId: org.sonatype.books.nexus
  - artifactId: sample-site
  - version: 1.0-SNAPSHOT
  - package: org.sonatype.books.nexus
-

```
~/examples$ mvn archetype:generate
[INFO] [archetype:generate {execution: default-cli}]
[INFO] Generating project in Interactive mode
Choose archetype:
1: internal -> appfuse-basic-jsf
...
13: internal -> maven-archetype-portlet (A simple portlet application)
14: internal -> maven-archetype-profiles ()
15: internal -> maven-archetype-quickstart ()

...
Choose a number: (...14/15/16...) 15: : 15
Define value for groupId: : org.sonatype.books.nexus
Define value for artifactId: : sample-site
Define value for version: 1.0-SNAPSHOT: : 1.0-SNAPSHOT
Define value for package: org.sonatype.books.nexus: : org.sonatype.books. ↵
    nexus
Confirm properties configuration:
groupId: org.sonatype.books.nexus
artifactId: sample-site
version: 1.0-SNAPSHOT
package: org.sonatype.books.nexus
Y: :
[INFO] Parameter: groupId, Value: org.sonatype.books.nexus
[INFO] Parameter: packageName, Value: org.sonatype.books.nexus
[INFO] Parameter: package, Value: org.sonatype.books.nexus
[INFO] Parameter: artifactId, Value: sample-site
[INFO] Parameter: basedir, Value: /private/tmp
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] OldArchetype created in dir: /private/tmp/sample-site
[INFO] -----
[INFO] BUILD SUCCESSFUL
```

After running the archetype:generate command, you will have a new project in a sample-site/ subdirectory.

## 20.3 Configuring Maven for Site Deployment

To deploy a site to a Nexus Site repository, you will need to configure the project's distribution management settings, add site deployment information, and then update your Maven settings to include the appropriate credentials for Nexus.

---

Add the following section to sample-site/pom.xml before the dependencies element. This section will tell Maven where to publish the Maven-generated project website:

### Distribution Management for Site Deployment to Nexus

```
<distributionManagement>
  <site>
    <id>nexus</id>
    <url>dav:http://localhost:8081/nexus/content/sites/site/</url>
  </site>
</distributionManagement>
```

The URL in the distribution management does not change with the project versions automatically, which means that any redeployment overwrites old content and potentially leaves old stale files behind. To have a new deployment directory for each version, change the URL to a parameterized setup or a hardcoded specific URL for your project version.

If you combine this approach with a redirector or a static page that links to the different copies of your site, you can e.g., maintain separate sites hosting your javadoc and other documentation for different releases of your software.

The dav protocol used by for deployment to Nexus requires that you add the implementing library as a dependency to the Maven site plugin configuration.

### Configuring Version 3.4 of the Maven Site Plugin with DAV support

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.4</version>
      <dependencies>
        <dependency>
          <groupId>org.apache.maven.wagon</groupId>
          <artifactId>wagon-webdav-jackrabbit</artifactId>
          <version>2.6</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

## 20.4 Adding Credentials to Your Maven Settings

When the Maven Site plugin deploys a site to Nexus, it needs to supply the appropriate deployment credentials to Nexus. To configure this, you need to add credentials to your Maven Settings. Open up your `~/.m2/settings.xml` and add the following server configuration to the servers element.

### Configuring Deployment Credentials for Nexus Site Deployment

```
<settings>
  <servers>
    <server>
      <id>nexus</id>
      <username>deployment</username>
      <password>deployment123</password>
    </server>
  </servers>
</settings>
```

---

#### Note

[Configuring Deployment Credentials for Nexus Site Deployment](#) uses the default deployment user and the default deployment user password. You will need to configure the username and password to match the values expected by your Nexus installation.

---

## 20.5 Creating a Site Repository

To create a site repository, log in as a user with Administrative privileges, and click on *Repositories* under *Views/Repositories* in the Nexus menu. Under the Repositories tab, click on the *Add...* drop-down and choose *Hosted Repository* as shown in Figure 20.1.

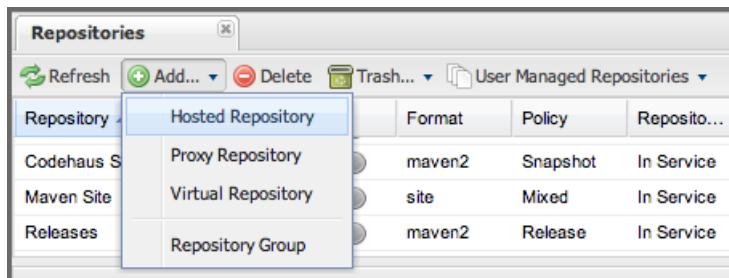


Figure 20.1: Adding a Hosted Repository

In the *New Hosted Repository* form, click on the *Provider* drop-down and chose the *Site* provider as shown in Figure 20.2. Although you can use any arbitrary name and identifier for your own Nexus repository, for the chapter's example, use a Repository ID of `site` and a Repository Name of `Maven Site`.

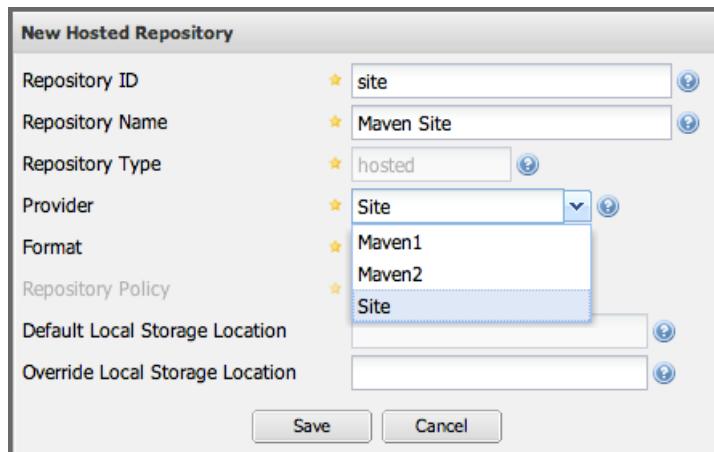


Figure 20.2: Creating a New Maven Site Repository

After creating a new Site repository, it should appear in the list of Nexus repositories as shown in Figure 20.3. Note that the Repository Path shown in Figure 20.3 is the same as the repository path referenced in [Distribution Management for Site Deployment to Nexus](#).

Repository	Type	Health Check	Format	Policy	Repository Sta...	Repository Path
JBoss Releases	proxy	0	0	maven2	Release	In Service
Maven Site	hosted	<b>ANALYZE</b>	site	Mixed	In Service	<a href="http://localhost:8081/nexus/content/sites/site">http://localhost:8081/nexus/content/sites/site</a>
NuGet Relea...	hosted	<b>ANALYZE</b>	nuget		In Service	<a href="http://localhost:8081/nexus/content/reposito...">http://localhost:8081/nexus/content/reposito...</a>

Figure 20.3: Newly Created Site Repository

---

**Tip**

The Site provider support is implemented in the Nexus Site Repository Plugin and is installed by default in Nexus Open Source as well as Nexus Professional.

---

## 20.6 Add the Site Deployment Role

In the Maven Settings shown in [Configuring Deployment Credentials for Nexus Site Deployment](#), you configured your Maven instance to use the default deployment user and password. To successfully deploy a site to Nexus, make sure that the deployment user has the appropriate role and permissions. To add the site deployment role to the deployment user, click on *Users* under the *Security* section of the *Nexus* menu, and click on the *Add* button in the *Role Management* section. This will trigger the display of the *Add Roles* dialog that will allow you to apply a filter value of `site` to locate the applicable roles as shown in Figure 20.4.

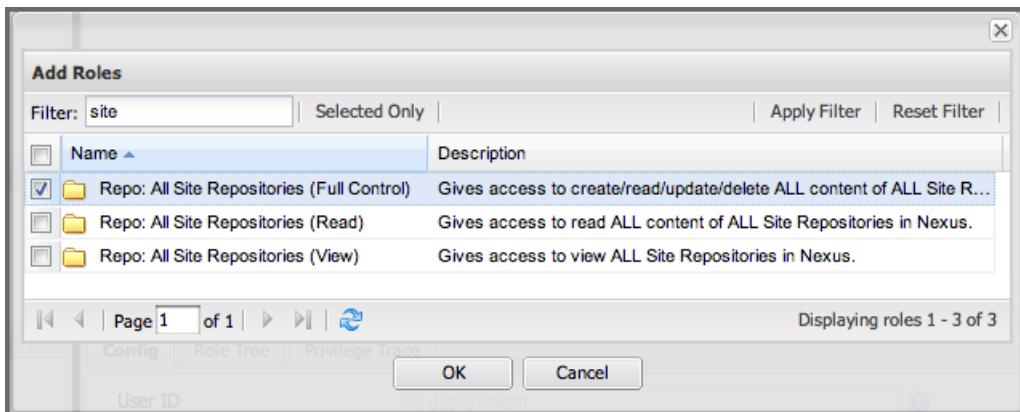


Figure 20.4: Adding the Site Deployment Role to the Deployment User

Check the box beside the "Repo: All Site Repositories (Full Control)" role in the list and press OK in the dialog. After the dialog closes, you should see the new role in the *Role Management* section. Click on the *Save* button to update the roles for the deployment user. The deployment user now has the ability to publish sites to a Maven site repository.

## 20.7 Publishing a Maven Site to Nexus

To publish a site to a Maven Site repository in Nexus, run mvn site-deploy from the sample-site/ project created earlier in this chapter. The Maven Site plugin will deploy this site to Nexus using the credentials stored in your Maven Settings.

```
~/examples/sample-site$ mvn site-deploy
[INFO] Scanning for projects...
[INFO]   ←
-----
[INFO] Building sample-site
...
[INFO] Generating "About" report.
[INFO] Generating "Issue Tracking" report.
[INFO] Generating "Project Team" report.
[INFO] Generating "Dependencies" report.
[INFO] Generating "Project Plugins" report.
[INFO] Generating "Continuous Integration" report.
[INFO] Generating "Source Repository" report.
```

```
[INFO] Generating "Project License" report.
[INFO] Generating "Mailing Lists" report.
[INFO] Generating "Plugin Management" report.
[INFO] Generating "Project Summary" report.
[INFO] [site:deploy {execution: default-cli}]
http://localhost:8081/nexus/content/sites/site/ - Session: Opened
Uploading: ./css/maven-base.css to http://localhost:8081/nexus/content/ ←
sites/site/
#http://localhost:8081/nexus/content/sites/site//./css/maven-base.css \
- Status code: 201

Transfer finished. 2297 bytes copied in 0.052 seconds
Uploading: ./css/maven-theme.css to http://localhost:8081/nexus/content/ ←
sites/site/
#http://localhost:8081/nexus/content/sites/site//./css/maven-theme.css \
- Status code: 201

Transfer finished. 2801 bytes copied in 0.017 seconds

Transfer finished. 5235 bytes copied in 0.012 seconds
http://localhost:8081/nexus/content/sites/site/ - Session: Disconnecting
http://localhost:8081/nexus/content/sites/site/ - Session: Disconnected
[INFO] ←
----- ←

[INFO] BUILD SUCCESSFUL
[INFO] ←
----- ←

[INFO] Total time: 45 seconds
[INFO] Finished at: Sat Oct 03 07:52:35 CDT 2009
[INFO] Final Memory: 35M/80M
[INFO] -----
```

Once the site has been published, you can load the site in a browser by going to <http://localhost:8081/nexus/content/sites/site/>.

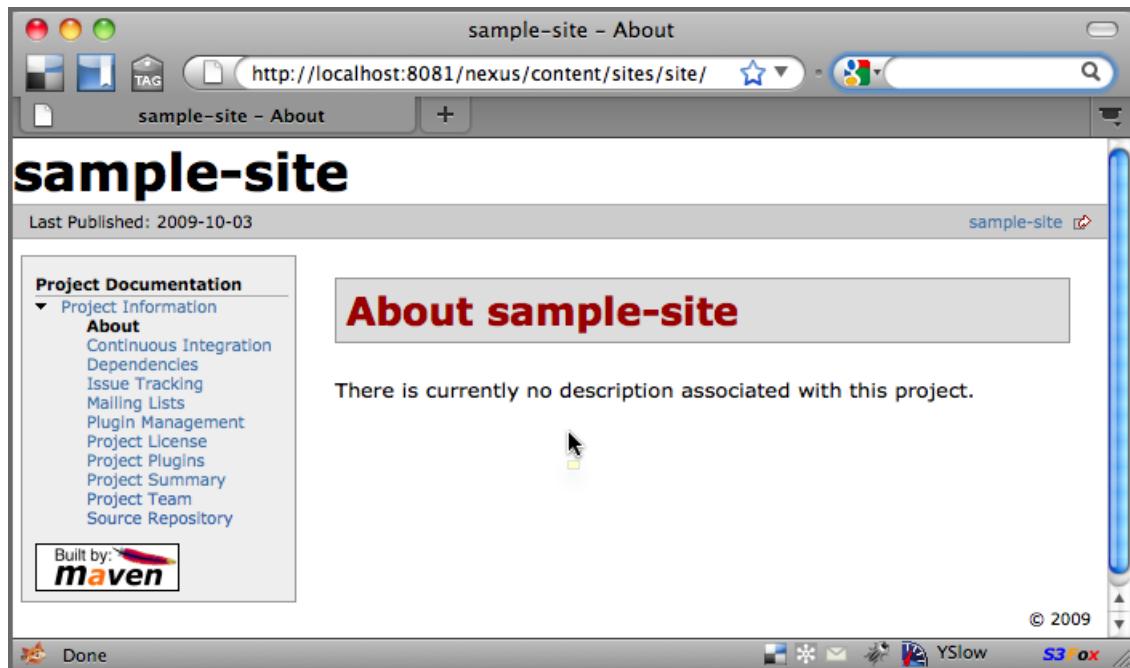


Figure 20.5: Sample Site Maven Project Website

## Chapter 21

# Nexus Best Practises

### 21.1 Introduction

Once you decide to install a Repository Manager, the next decision is how to set up your repositories, particularly if you have multiple teams sharing the same instance. Nexus is very flexible in this area and supports a variety of configurations. I'll first describe the options and then discuss the thought process used to decide what makes sense for your organization.

### 21.2 Repositories Per Project/Team

The first and most obvious way to support multiple teams is to configure a pair of repositories per team (one release, one snapshot). The team is then given the appropriate C.R.U.D. permissions, and they are able to use the system for their artifacts.

The Sonatype Open Source Repository Hosting (OSSRH) Nexus install available at <http://oss.sonatype.org> is for the most part configured in this manner, where each project has their own repositories separate from everyone else.

## 21.3 Partition Shared Repositories

Another option is to have a single pair (or a few pairs) of release and snapshot repositories for your entire organization. In this case, the access is controlled by repository targets.

Simply put, a repository target is a way to manage a set of components based on their paths in a repository. A repository target is simply a list of regular expressions and a name. For example, a repository target pattern for Apache Maven would be `./org/apache/maven/.` or for Nexus OSS it would be `./org/sonatype/nexus/..`

---

### Note

While it is most common to manage artifacts based on the path of their groupId, the *Regular Expression* is matched against the entire path, and so it is also possible, for example, to define *Sources* as `.*-sources.jar`. It is also worth noting that repository targets are not mutually exclusive. It is perfectly valid for a given path to be contained by multiple targets.

---

In this model, you would create a repository target for each project in your system. You are then able to take the repository target and associate it with one or more repositories or repository groups. This creates new C.R.U.D. privileges specific to the repository or group. For example, you could take the Maven repository target, associate it with the release and snapshot repository. You get privileges you can assign to Create, Read, Update, Delete "Maven" (`./org/apache/maven/.`) artifacts in my Release and Snapshot repositories.

This method is used to manage the <http://repository.apache.org> instance, where we have just one Release and Snapshot repository and each project team gets permissions to their artifacts based on the path.

### 21.3.1 Selecting an Approach

First of all, these choices aren't mutually exclusive. In fact, the first option builds upon the default repository target of `.*` which simply gives you access to all artifacts regardless of the path. You still associate the default repository target with specific repositories to create the assignable privileges

In general, it's my opinion that fewer repositories will scale better and are easier to manage. It's also easier to start off with a single pair of repositories with the default target and simply refine the permissions as you scale. Most things that are configured per repository (Cache, Storage location, Snapshot purging,

---

etc.) will generally be applicable for all projects, so this mode avoids the duplication of these tasks. Since everything will be stored together in a single folder on disk, it makes backups easier as well.

The reasons why you would want multiple sets of repositories is essentially the opposite of above: If you need different expiration, Snapshot purging, or storage folders, then a single shared repo won't work. Replication and failover strategies may also make this method easier to support. If you absolutely must maintain total separation between Project teams, i.e. they can't read each other's artifacts, then this solution might be more applicable as well.

In summary, Nexus allows you to control the security of your components based on the repository and/or the path of the components, meaning it is possible to slice and dice the system any way you see fit. The default suggestion is to use as few hosted repositories as possible and control the permissions by using repository targets.

## Chapter 22

# Nexus Plugins

Nexus Open Source and Nexus Professional are built using a plugin architecture, where each version includes a different set of plugins. You can install plugins available from the open source community, other vendors, or created by yourself in addition to the default plugins.

Nexus plugins can provide further functionality for the backend such as support for new repository formats, specific behavior for components, new scheduled tasks, new staging rules, and any other additional functionality as well as new user interface components and modifications. They can also group a number of these features together in one plugin.

### 22.1 Managing Nexus Plugins

All plugins supplied by Sonatype are installed as part of the default configuration and can be found in `$NEXUS_HOME/nexus/WEB-INF/plugin-repository`. Most plugins are enabled by default.

Some plugins expose a capability as documented in Section 6.6 and can be enabled, disabled, and otherwise configured in the capability administration. The branding plugin or the outreach plugin are examples of plugins exposing capabilities.

**Note**

Prior to Nexus 2.7 optional plugins, supplied by Sonatype, can be found in the directory `$NEXUS_HOME/nexus/WEB-INF/optional-plugins`. To install any of these, simply copy the folder containing the desired plugin into `$NEXUS_HOME/nexus/WEB-INF/plugin-repository`. When updating Nexus, redo the install of any optional plugins using the newest version shipping with the download of the new Nexus version. Any configuration of the plugin will be preserved from one version to the other.

---

Plugins supplied by third parties or ones that you authored are installed by copying the folder with the plugin code into `sonatype-work/nexus/plugin-repository` or extracting the plugin bundle zip file in that folder.

After a restart of Nexus, the new plugins will be active and ready to use. Upgrades are done by shutting down Nexus, copying the newer plugin into the folder, removing the older one, and restarting Nexus.

Capability-based plugins can be disabled in the capability administration. Otherwise, plugins can be removed by deleting the respective folder in the `plugin-repository` and restarting Nexus.

## 22.2 Developing Nexus Plugins

Developing Nexus plugins allow you to customize and further enhance Nexus beyond the features and capabilities offered. This section provides you with the information to begin developing your own plugins.

The preferred way to write Nexus plugins is to use Java as the implementation language and Apache Maven as the build system. The [Nexus Example Plugins](#) project demonstrates a number of plugin examples for Nexus Open Source and Nexus Professional. Further examples are the [plugins of Nexus Open Source](#).

The easiest way to create a new Nexus plugin project is to replicate a plugin with a similar functionality from these projects. The existing plugins and codebase should be used as examples for your own functionality. Inspect the source code of plugins with similar functionality, and read the JavaDoc documentation for the involved classes.

---

**Note**

The Maven archetype `nexus-archetype-quickstart` is deprecated.

---

To gain access to all the components needed for your Nexus plugin development, you have to proxy the Sonatype grid repository with the URL below:

```
https://repository.sonatype.org/content/groups/sonatype-public-grid/
```

For some Nexus Professional specific plugins, you might need access to the private grid. We suggest that you work with the support team in this situation.

Set up your project to include inheriting from the parent of all the Nexus Open Source plugins with the version of Nexus you are targeting as displayed in [Inheriting from the nexus-plugins Parent](#).

### Inheriting from the nexus-plugins Parent

```
<parent>
  <groupId>org.sonatype.nexus.plugins</groupId>
  <artifactId>nexus-plugins</artifactId>
  <version>2.11.2.-06</version>
</parent>
```

---

#### Warning

 It is best to use the identical version of the parent as the Nexus instance no which you want to run your plugin. When developing a plugin you are using large parts of Nexus internals, which are subject to change from one version of Nexus to another. This same logic applies to any dependencies as well.

---

A Nexus plugin Maven project creates a custom build output file in the form of a zip file that contains all dependencies, in addition to your class files and resources from your plugin and some metadata. Enable this by changing the packaging and adding the bundle plugin listed in [nexus-plugin Packaging](#).

### nexus-plugin Packaging

```
<project>
...
<groupId>com.myorganization.nexus.plugins</groupId>
<artifactId>example-nexus-plugin</artifactId>
<version>1.0-SNAPSHOT</
<packaging>nexus-plugin</packaging>
...
<build>
  <plugins>
```

```
<plugin>
  <groupId>org.sonatype.nexus</groupId>
  <artifactId>nexus-plugin-bundle-maven-plugin</artifactId>
  <extensions>true</extensions>
</plugin>
</plugins>
</build>
```

Add the dependencies in [Adding the Nexus Plugin API and Testsupport](#) to your Maven project pom.xml file, to access the Nexus Plugin API and test support.

### Adding the Nexus Plugin API and Testsupport

```
<dependencies>
  <dependency>
    <groupId>org.sonatype.nexus</groupId>
    <artifactId>nexus-plugin-api</artifactId>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.sonatype.nexus</groupId>
    <artifactId>nexus-plugin-testsupport</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

These dependencies pull in a large number of transitive dependencies that expose Nexus functionality and other libraries to your project. Depending on the type of plugin and functionality you aim to create, additional dependencies and other details can be added to this minimal project setup. A large number of further classes is available and can be used as part of your plugin development. Some of these classes are contained in other plugins of Nexus. If you want to use these, you have to add a dependency to this plugin to your plugin's pom.xml.

An example is a plugin you create that exposes a REST API for further integrations with tools outside of Nexus similar to how all other Nexus plugins expose a REST API. The dependency to add is displayed in [Adding a Dependency to the Nexus Siesta Plugin](#).

### Adding a Dependency to the Nexus Siesta Plugin

```
<dependency>
  <groupId>com.sonatype.nexus.plugins</groupId>
  <artifactId>nexus-siesta-plugin</artifactId>
  <type>nexus-plugin</type>
  <scope>provided</scope>
```

```
</dependency>
```

Nexus and Nexus plugins use JSR-330 annotations like `@javax.inject.Inject` and the Google Guice dependency injection framework. Typical classes are `@Named` and are often a `@Singleton`. Other components are typically injected via constructor injection as displayed in the example from the `virusscan` example plugin in [Constructor Injection](#).

## Constructor Injection

```
@Inject
public VirusScannerRequestProcessor(final EventBus eventBus,
                                    final List<VirusScanner> scanners)
{
    this.eventBus = Preconditions.checkNotNull(eventBus);
    this.scanners = Preconditions.checkNotNull(scanners);
    ...
}
```

Your Maven project setup should follow the typical standard directory layout conventions. In addition, static resources such as JavaScript files, images, and CSS should be placed in `src/main/resources/static`.

Once you have created your Maven project as described above, you can build the plugin with

```
mvn clean install
```

A successful build includes the creation of a `*-bundle.zip` file in the `target` folder. To install your plugin into Nexus you can extract it into the `plugin-repository` directory as described in Section [22.1](#).

## 22.3 Summary

The Nexus architecture is largely based on plugins including the differentiation of Nexus Open Source and Nexus Professional. By inspecting the example plugins and the Nexus open source project, you can create additional Nexus functionality for yourself as well as potentially share it with the Nexus user community.

## Chapter 23

# Migrating to Nexus

If you have been running another repository manager, such as Artifactory, Archiva, or Proximity, and you want to migrate this repository to Nexus, you can do so by copying the files from a standard Maven 2 repository file layout to Nexus.

Depending on your repository managers, you will have to use different approaches to get access to a repository in Maven 2 format on disk.

Nexus stores its artifacts in standard Maven 2 layout, and they are served directly from disk, and can therefore be easily integrated into an existing Nexus instance as a new hosted repository.

### 23.1 Migrating from Archiva

#### 23.1.1 Introduction

This appendix walks you through the process of migrating an existing [Archiva](#) installation to a new Nexus installation.

---

### 23.1.2 Migrating Archiva Repositories

Archiva uses the file system to store hosted repositories and proxied repositories, making migration from Archiva to Nexus very simple. The following sections outline the process for migrating existing Archiva repositories to a new Nexus instance.

### 23.1.3 Migrating an Archiva Managed Repository

Archiva Managed Repositories are the equivalent of Nexus Hosted repositories. To migrate a Managed Repository from Archiva to Nexus, do the following:

- Create a New Hosted Repository in Nexus.
- Copy the Contents of the Archiva Managed Repository to the Storage Directory of the newly-created Nexus Hosted Repository.
- Rebuild the Index for the New Nexus Hosted Repository.

The following example will walk through the process of migrating the Archiva repository named `internal`, to a new Nexus Hosted repository named "internal". To view your managed repositories in Archiva, login to Archiva as an administrative user and click on the *Repositories* link in the left-hand navigation menu. Clicking on *Repositories* will list all of your Archiva Managed repositories as shown in Figure 23.1.

The screenshot shows the 'Managed Repositories' interface for Archiva. At the top right are 'Add', 'Edit', 'Delete', and a refresh icon. The repository details for 'Archiva Managed Internal Repository' are listed:

Identifier	internal
Name	Archiva Managed Internal Repository
Directory	./data/repositories/internal
WebDAV URL	<a href="http://localhost:8080/archiva/repository/internal/">http://localhost:8080/archiva/repository/internal/</a>
Type	Maven 2.x Repository
Releases Included	
Snapshots Included	
Scanned	
Scanning Cron	0 0 * * *
Actions	<a href="#">Scan Repository Now</a>
Stats	No Statistics Available.
POM Snippet	<a href="#">Show POM Snippet</a>

Figure 23.1: Archiva Managed Repositories

To migrate this Managed repository to a Nexus Hosted repository, find the directory in which Archiva stores all of the repository artifacts. To do this, click on the *Edit* link listed next to the name of the repository you want to migrate as shown in Figure 23.1. Clicking on *Edit* should load the form shown in Figure 23.2.

**Admin: Edit Managed Repository**

*ID:* internal

**Name\***: Archiva Managed Internal Repository

**Directory\***: ./data/repositories/internal

**Index Directory:** [ ]

**Type:** Maven 2.x Repository

**Cron\***: 0 0 \* \* \* ?

**Repository Purge By Days Older Than:** 30

**Repository Purge By Retention Count:** 2

Releases Included

Snapshots Included

Scannable

Delete Released Snapshots

**Update Repository**

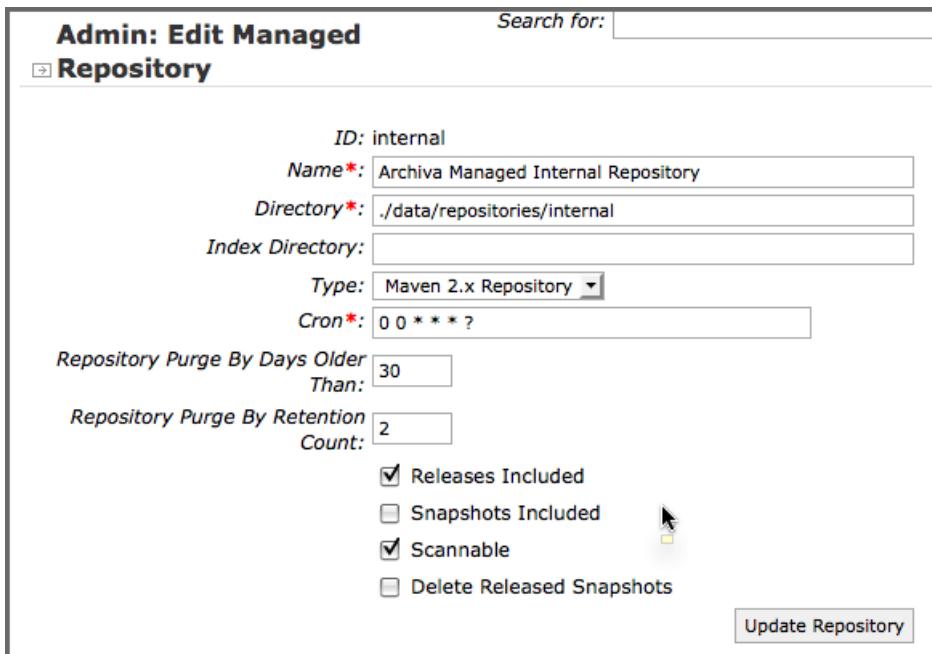


Figure 23.2: Editing an Archiva Managed Repository

Take note of the file path for Directory. The file path shown in Figure 23.2 is ./data/repositories/internal. If Archiva is installed in /usr/local/archiva-1.2.1, it should correspond to the directory /usr/local/archiva-1.2.1/data/repositories/internal. You will use this path later in this section to copy the contents of your old Archiva Managed Repository to your new Nexus Hosted Repository.

Next, create a new Nexus repository with the same identifier and Name as the old Archiva Managed Repository. To do this, log into Nexus as an administrative user, click on Repositories in the left-hand Nexus navigation menu, and then click on the Add drop-down as shown in Figure 23.3. Select "Hosted Repository" and then fill out the Repository ID and Repository Name to match the name of the old Archiva repository. If you are migrating a Snapshot repository, select a Repository Policy of Snapshot, and if you are migrating a Release repository select a Snapshot Policy of Release.

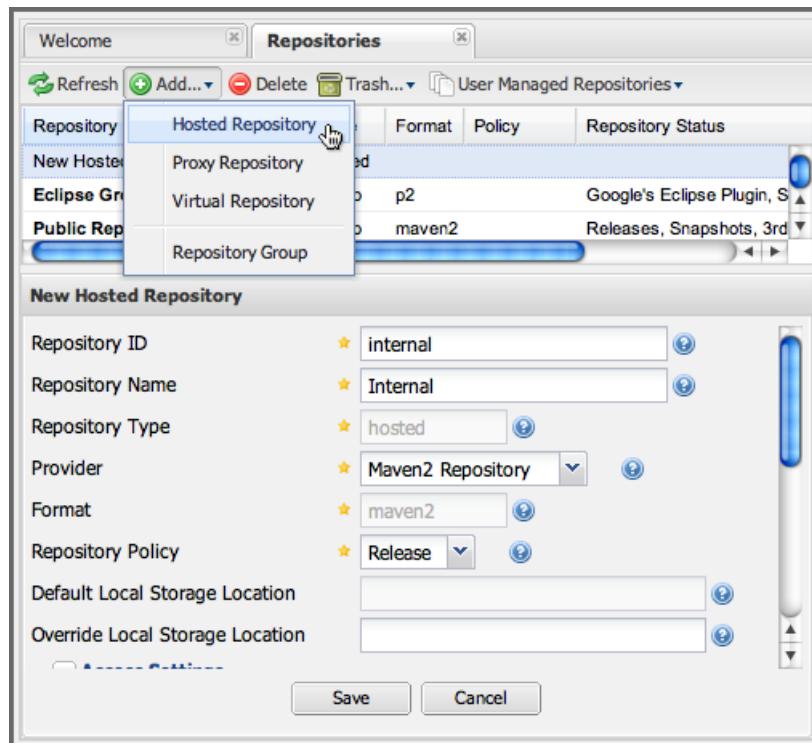


Figure 23.3: Creating a Nexus Hosted Repository

Now, you'll need to copy the Archiva repository to the Nexus repository. You can do this by copying the contents of the Archiva repository directory to the Nexus repository storage directory. If we assume that Archiva is installed in /usr/local/archiva-1.2.1, Nexus is installed in /usr/local/nexus, and the Sonatype Work directory is /usr/local/sonatype-work. You can copy the contents of the Archiva managed repository to the new Nexus hosted repository by executing the following command:

```
$ cp -r /usr/local/archiva-1.2.1/data/repositories/internal/* \
/usr/local/sonatype-work/nexus/storage/internal/
```

If you are migrating to a Nexus instance on a different server, you can simply create an archive of the /usr/local/archiva-1.2.1/data/repositories/internal directory, copy it to the new server, and then decompress your repository archive in the appropriate directory.

**Warning**

 Archiva stores artifacts from proxied remote repositories in the same directory as artifacts in a managed repository. If you have been proxying a remote repository, you might want to remove artifacts that have been proxied from a remote repository. For example, if your organization uses a groupId of org.company for internal project, you can make sure to only copy the artifacts under the corresponding org/company/.

Once the contents of the repository have been copied to the Nexus Hosted repository, you must rebuild the repository index as shown in Figure 23.4. Right-clicking on the repository in the list of Nexus repositories will display the context menu shown in the following figure.

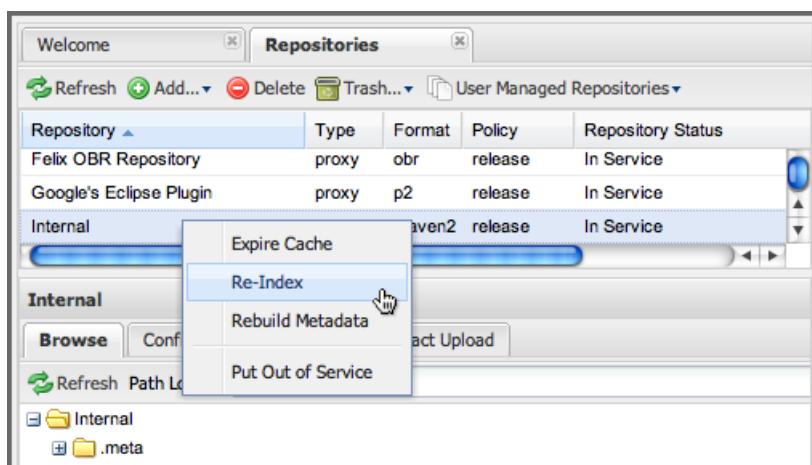


Figure 23.4: Rebuilding the Index of a Nexus Hosted Repository

Once the migration is complete, you will be able to search and browse the contents of your newly migrated Nexus Hosted repository.

### 23.1.4 Migrating an Archiva Proxy Connector

Archiva allows you to define remote repositories and repository connectors to proxy remote repositories and cache remote artifacts from remote repositories in Archiva Managed Repositories. While Nexus also provides Proxy repositories, there is one major difference between Nexus and Archiva. Where Nexus maintains a separate local storage directory for each proxy repository, Archiva combines cached remote

artifacts into a single file system with the contents of a managed repository. In other words, there is no good way to transfer an existing local cache of artifacts between Archiva and Nexus without manually manipulating the contents of Archiva's Managed Repository directory.

To recreate an Archiva repository connector in Nexus as a Proxy repository and to preserve the local cache of artifacts from this repository. You'll need to create a Proxy repository in Nexus, copy the contents of the existing proxy repository to the Nexus storage location for your new Proxy repository, and then rebuild the metadata of your new Nexus Proxy repository.

First step is to take a look at the Remote Repositories in your Archiva installation. Log in as an administrative user and then click on *Repositories* under the *Administration* menu in the left-hand Archiva navigation menu. Once you've clicked this link and loaded the list of repositories, scroll to the bottom of the page to see the list of remote repositories as shown in Figure 23.5.

Identifier	Name	URL	Type
central	Central Repository	http://repo1.maven.org/maven2	Maven 2.x Repository
maven2-repository.dev.java.net	Java.net Repository for Maven 2	http://download.java.net/maven/2/	Maven 2.x Repository

Figure 23.5: Browsing Archiva Remote Repositories

Defining a proxy repository in Archiva involves associating one of the remote repositories defined in Figure 23.5 with one of the Managed Repositories defined in Figure 23.1. Once you do this, requests for artifacts from the managed repository will also query the remote repository. If an artifact is found in the remote repository, it will be retrieved and stored in the managed repository's storage directory. To see a list of proxy connectors and the managed repositories with which they are associated, click on *Proxy Connectors* in the left-hand Archiva menu and you will see a list similar to that shown in Figure 23.6.

The screenshot shows the 'Repository Proxy Connectors' section of the Archiva interface. At the top right is a green '+' icon labeled 'Add'. Below it, there's a header 'internal' with a small orange icon, followed by the text 'Archiva Managed Internal Repository'. Underneath this, there are two proxy connector entries:

- Proxy Connector**: 'central'. It features a globe icon, the name 'central', the description 'Central Repository', and the URL '<http://repo1.maven.org/maven2>'. To the right of the entry are edit (pencil), move up (green triangle), move down (red triangle), and delete (red X) icons.
- Proxy Connector**: 'maven2-repository.dev.java.net'. It features a globe icon, the name 'maven2-repository.dev.java.net', the description 'Java.net Repository for Maven 2', and the URL '<http://download.java.net/maven/2/>'. To the right of the entry are edit (pencil), move up (green triangle), move down (red triangle), and delete (red X) icons.

Below each entry is a 'Settings' link.

Figure 23.6: Archiva Proxy Connectors

Click on the edit icon (or pencil) next to second Proxy Connector listed in Figure 23.6, to load the settings form for this proxy connector shown in Figure 23.7. You should use the settings for this proxy connect to configure your new Nexus Proxy repository.

The screenshot shows the configuration interface for a Proxy Connector in Archiva. At the top, there are three dropdown menus: 'Network Proxy\*' set to '(direct connection)', 'Managed Repository\*' set to 'internal', and 'Remote Repository\*' set to 'maven2-repository.dev.java.net'. Below these are several policy settings:

Policies:	Return error when:	On remote error:	Releases:	Snapshots:	Checksum:	Cache failures:
	always	stop	once	never	fix	yes

Under 'Properties', there is a text input field followed by an 'Add Property' button. A message below states 'No properties have been set.' In the 'Black List' section, there is a text input field followed by an 'Add Pattern' button. A message below states 'No black list patterns have been set.' In the 'White List' section, there is a text input field followed by an 'Add Pattern' button. Three entries are listed in the white list: "'javaxax/\*\*'" with a red X, "'org/jvnet/\*\*'" with a red X, and "'com/sun/\*\*'" with a red X. At the bottom right is a 'Save Proxy Connector' button.

Figure 23.7: Archiva Proxy Connector Settings

To create a Proxy repository that will correspond to the Proxy Connector in Archiva, log into Nexus as an administrative user, and click on Repositories in the left-hand Nexus menu. Once you can see a list of Nexus repositories, click on Add... and select Proxy Repository from the drop-down of repository types. In the New Proxy Repository form (shown in Figure 23.8) populate the repository ID, repository Name, and use the remote URL that was displayed in Figure 23.5. You will need to create a remote repository for every proxy connector that was defined in Archiva.

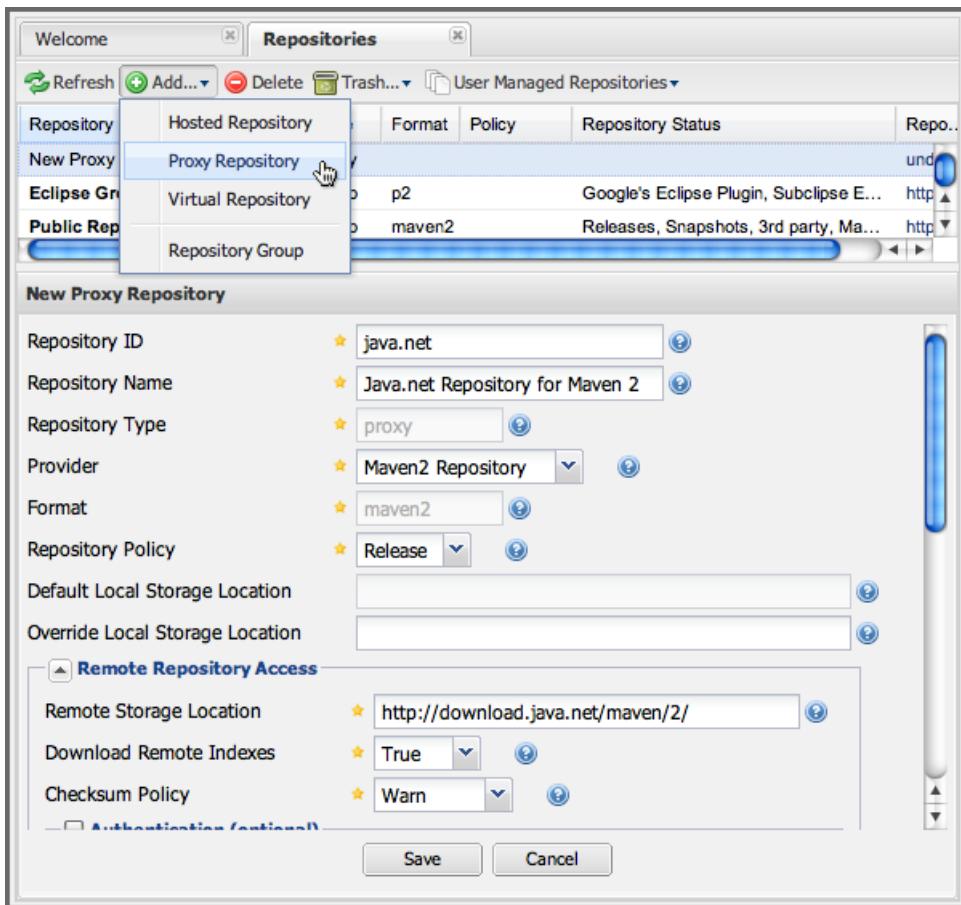


Figure 23.8: Creating a Nexus Proxy Repository

To expose this new Proxy repository in a Repository Group, create a new Nexus Repository group or select an existing group by clicking on Repositories in the left-hand Nexus menu. Click on a repository group and then select the Configuration tab to display the form shown in Figure 23.9. In the Configuration tab you will see a list of Order Group Repositories and Available Repositories. Click and drag your new Nexus Proxy repository to the list of Ordered Group Repositories, and click Save.

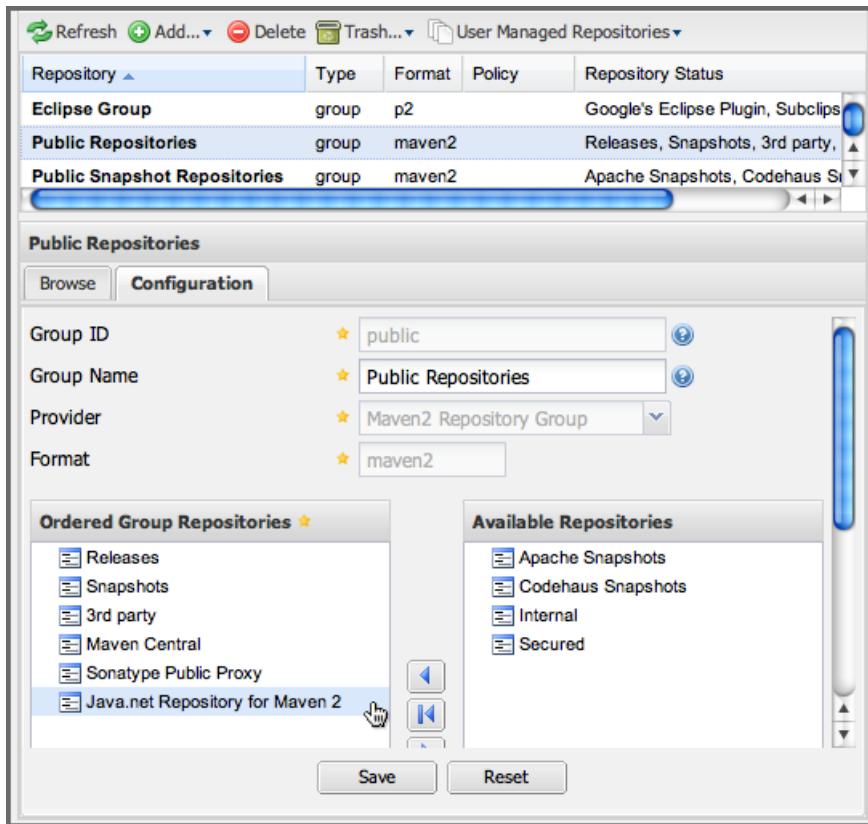


Figure 23.9: Adding a Proxy Repository to a Repository Group

Next, you will need to define repository groups that will tell Nexus to only locate certain artifacts in the newly created proxy repository. In , Archiva defined three patterns that were used to filter artifacts available from the proxy connector. These three patterns were "javax/", "com/sun/", and "org/jvnet/\*\*". To recreate this behavior in Nexus, define three Routes which will be applied to the group you configured in Figure 23.9. To create a route, log in as an administrative user, and click on Routes under the Administration menu in the left-hand Nexus menu. Click on Add.. and add three inclusive routes that will apply to the repository group you configured in Figure 23.9.

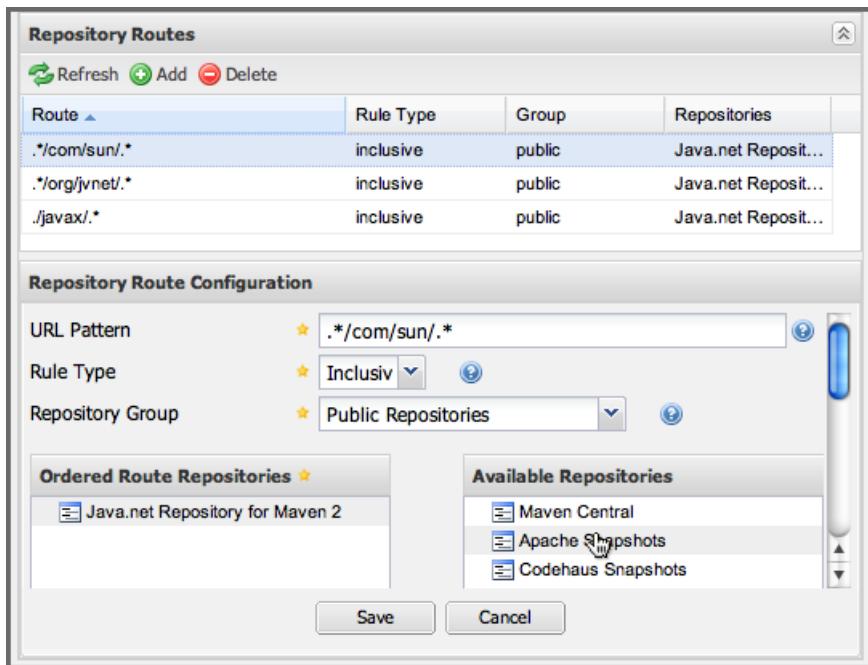


Figure 23.10: Defining Nexus Routes

## 23.2 Migrating from Artifactory

This appendix provides a guideline for migrating a Maven repository from Artifactory to Nexus.

Typically migrating from Artifactory revolves around migrating hosted repositories only, since any proxy repositories configured in Artifactory can just be set up with the same configuration in Nexus, and all data will be retrieved from the upstream repositories again.

Hosted repositories on the other hand have to be migrated. The best practice for migration is to use the import/export feature of Artifactory and migrate one hosted repository after another. Please consult the Artifactory documentation for step-by-step instructions on how to export a repository.

After the export, you have to create a hosted repository in Nexus e.g., with the name `old-releases` as documented in Section 4.4. This will create a folder in `sonatype-work/nexus/storage/old-releases`.

Now you are ready to take the exported repository and copy it into the newly created storage folder.

Going back to the Nexus user interface, navigate to the repository administration and select the *Browse Storage* panel. Right-click on the root folder of the repository and select *Rebuild Metadata* first. and as a second step select *Update Index*. Once these tasks are completed, the migrated repository is ready to be used.

After these task are completed, you will probably want to add the migrated repository to the Public Repositories group or any other group in which you want the migrated repository content to be available.

If you want to ensure that the repository does not get any further content added, you can set the *Deployment Policy* to *Read Only* in the *Access Settings* of the repository *Configuration* panel.

## Chapter 24

# Configuring Nexus for SSL

### 24.1 Introduction

Using Secure Socket Layer SSL to secure protocols like HTTP, LDAP and SMTP is a critical step of securing your Nexus setup. Since Nexus is serving content as well as connecting to external sources, there are two aspects of SSL configuration related to Nexus:

- Configuring SSL certificate usage when connecting to external systems including
  - Proxying a remote repository available via HTTPS
  - Connecting to a SSL secured SMTP server
  - Connecting to an LDAP server via LDAPS
- Exposing the Nexus user interface and content via HTTPS

Securing all connections to external systems with SSL as well as exposing Nexus via SSL are both recommended best practices for any deployment.

Especially when you set up a repository manager for a team of developers spread out over a variety of locations both internal and external to a corporate network, you will likely want to secure your repository using SSL.

---

We generally recommend to secure your repository using SSL, especially when creating a repository for a team of developers in different geographical locations both internal and external to a corporate network.

## 24.2 SSL Client Certificates

### 24.2.1 SSL Certificate Management

Nexus allows you to manage all SSL certificates directly in the user interface. The administration interface for SSL certificates as visible in Figure 24.1 and can be accessed by selecting *SSL Certificates* in the left-hand *Administration* menu. The list of certificates displayed shows the certificate for the SSL-secured Central Repository preconfigured in Nexus Professional and a self-signed certificate registered in Nexus.

---

**Note**

The SSL Certificate Management is a Nexus Professional feature.

---

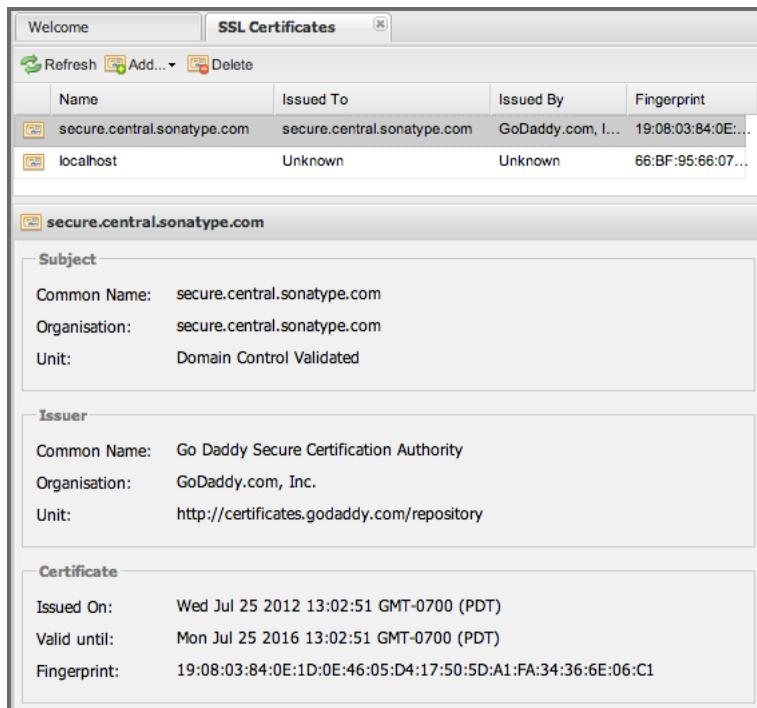


Figure 24.1: SSL Certificates Administration

The actual list of SSL certificates can be reloaded by clicking the *Refresh* button above the list. In addition, certificates can be added and deleted with the *Add* and *Delete* buttons.

Pressing the add button provides a choice to load a certificate from a server with the *Load from server* option or to insert a certificate in PEM format with the *Paste PEM*.

The dialog to load a certificate from a server allows you to provide a host name, a `hostname:port` string or a full URL. When providing a host name a connection via `http://` using the default SSL port 443 will be attempted. Using a full URL on the other hand gives the most control.

As an example, you could retrieve the certificate for the secured Central Repository using the URL

```
https://repo1.maven.org
```

Besides retrieving certificates for servers running HTTPS, you can retrieve and register the certificate for email and directory servers. An LDAP directory server certificate can be loaded with a URL using the LDAPS protocol and the desired host name and port similar to `ldaps://localhost:10636`. A SMTP server can be queried with a similar pattern using `smtps://localhost:465`. After successful retrieval, the details of the certificate as displayed in a dialog. Figure 24.2 shows the result from querying a certificate from `smtps://smtp.gmail.com:465`. Pressing the *Add Certificate* button will save the certificate within Nexus and allow you to connect to the associated services.

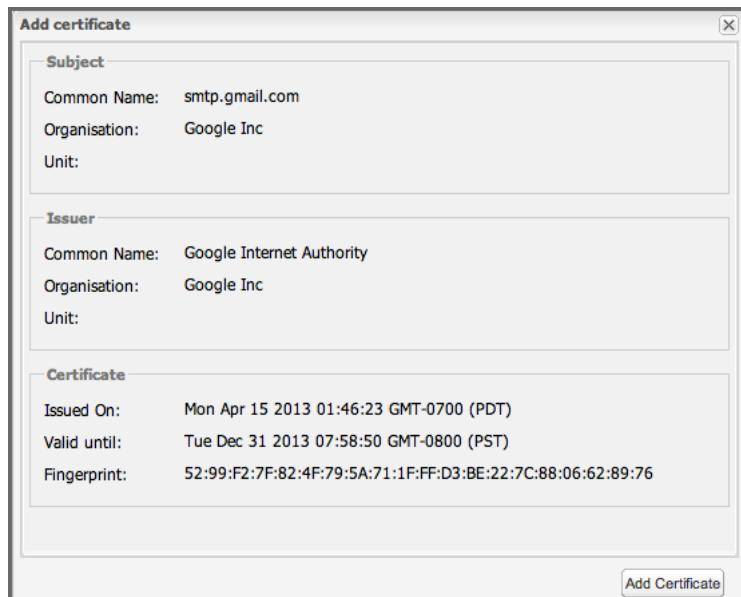


Figure 24.2: Certificate Details Displayed after Successful Retrieval

The dialog displays details about the certificate owner in the *Subject* section, the certificate issuer in the *Issuer* section and the certificate itself in the *Certificate* section. The same data is displayed below the list of certificates, when you select a specific certificate in the list.

The alternate method of registering a certificate with Nexus uses the PEM format of the [X.509 certificate](#) as used by SSL. An example of inserting such a certificate in the dialog is shown in Figure 24.3.

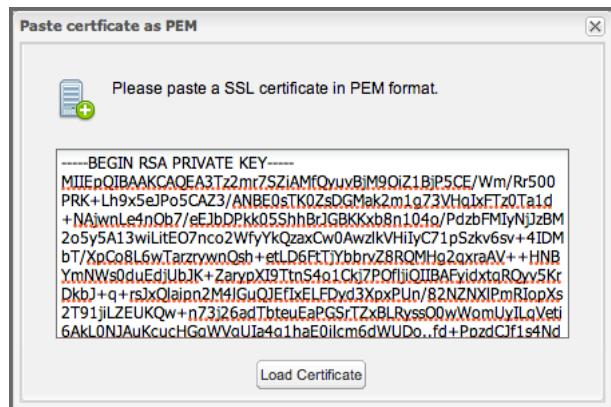


Figure 24.3: Providing a Certificate in PEM Format

Once a certificate for an LDAP server or SMTP server has been registered in Nexus, you can configure connections to these servers in the LDAP and Server/SMTP Settings administration user interfaces.

### 24.2.2 Proxying SSL Secured Repositories

When setting up a proxy repository with a remote storage location secured with HTTPS the repository administration will display an *SSL* configuration tab under the list of repositories if the proxy repository is selected. For a repository using a self-signed certificate, the repository status will initially be set to be in service, but the remote will be automatically blocked and set to be unavailable, since the certificate of the remote server is not trusted. Remote repositories that use a certificate authority(CA)-signed certificate will be automatically trusted.

The *SSL* tab displays as visible in Figure 24.4 the details of the certificate and allows you to add the certificate to the trust store or to remove it from it with the button on the top right-hand corner named *Add to trust store* and *Remove from trust store* respectively.

In addition, the checkbox on the top left corner allows you to store the certificate in the Nexus internal SSL trust store. Otherwise the certificate is installed into the trust store of the Java Virtual Machine (JVM) running Nexus. Using the Nexus internal trust store is recommended. It will work fine, even when migrating Nexus from one machine to another or when switching the Java runtime and JVM between restarts for example during upgrades. At runtime the JVM and Nexus trust stores are merged and both used so you can use a combination, if your organization e.g., maintains a default trust store for all JVM installations.

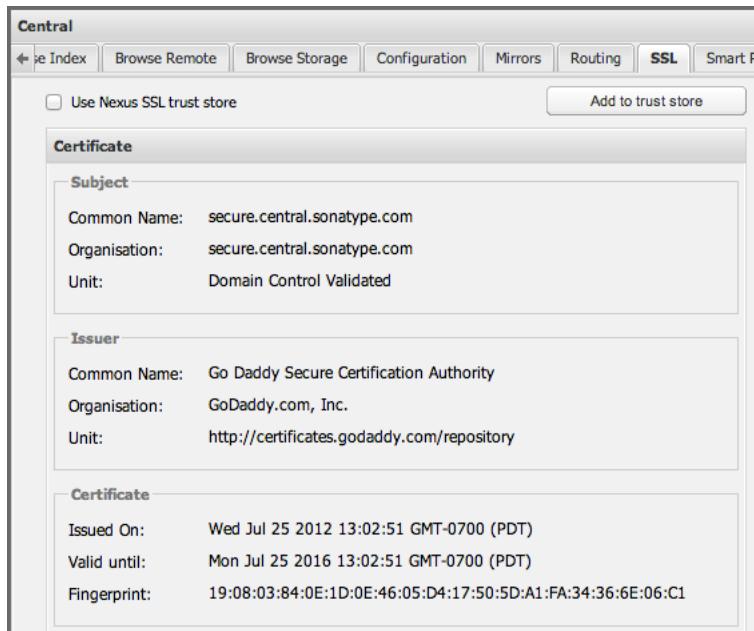


Figure 24.4: SSL Tab for a Proxy Repository with Remote Server Using HTTPS

When removing a certificate from the trust store, a Nexus restart is required.

### 24.2.3 Manually Configuring Trust Stores

The Nexus user interface should be sufficient to work with the trust stores and certificates. In older versions of Nexus as well as some use cases, you need to manually configure the trust store.

Sonatype provides an import-ssl tool that can be downloaded from <http://download.sonatype.com/nexus-import-ssl.jar>. It allows you to import a client certificate in two steps:

- importing the server's SSL chain and
- importing the client SSL key/certificate pair.

The Java Virtual Machine running Nexus uses the [Java Secure Socket Extension \(JSSE\)](#) to enable secure

Internet communication. It uses two certificate stores - `truststore` and `keystore`.

A truststore contains certificates from servers run by other parties with whom you expect to communicate, or from Certificate Authorities that you trust to identify other parties. This truststore ships with a number of CA's out-of-the-box, trusted root certificates.

A keystore contains private keys and the certificates with their corresponding public keys. Typically, they are stored in separate files stored in the default location of  `${JRE_HOME}/lib/security/cacerts`.

Some notes about the location of the keystore and default keystore passwords:

- If you are using the default JSSE keystore locations on either a Linux or OS X platform, you must run the commands below as the root user. You can do this either by changing to the root user (`su -`) or by using the `sudo` command: `sudo [command]`.
- The default password used by Java for the built-in keystores is *changeit*. If your key-store uses a different password, you'll need to specify that password as the last parameter on the command lines above.
- If you want to specify your own keystore/truststore file, provide that in place of `<keystore_dir>` in the examples below.
- If you're using a password other than *changeit* for your keystore, you should supply it immediately following the keystore path in the commands below.
- If you specify a keystore location that doesn't exist, the `import-ssl` utility will create it on-demand.

Before you begin the process of importing a Server SSL Chain and a client certificate you will need the following:

- Network access to the SSL server you are connecting to,
- An SSL client certificate,
- and a certificate password.

For server certificates you should either import directly into  `${JRE_HOME}/lib/security/cacerts`, or make a copy of the file and import into that.

---

**Warning**

If you replace the existing truststore rather than adding to it or if you override the truststore location, you will lose all of the trusted CA root certificates of the JRE and no SSL sites will be accessible.

---

#### 24.2.3.1 Import the Server SSL Chain

The first command imports the entire self-signed SSL certificate chain for central.sonatype.com into your JSSE keystore:

```
$ java -jar import-ssl.jar server repol.maven.org <keystore>
```

Substitute the server name used in the previous listing with the server name to which you are attempting to connect. This particular command will connect to `https://repol.maven.org`, retrieve, and import the server's SSL certificate chain.

#### 24.2.3.2 Import the Client SSL Key/Certificate Pair

The second command imports your client-side SSL certificate into the JSSE keystore, so Nexus can send it along to the server for authentication:

```
$ java -jar import-ssl.jar client <your-certificate.p12> \  
<your-certificate-password> keystore
```

When the client command completes, you should see a line containing the keystore path. Please note this, as you will use it in your next configuration step.

```
...  
Writing keystore: /System/Library/Frameworks/JavaVM.framework/\  
Versions/1.6.0/Home/lib/security/jsssecacerts
```

If you want to make a new keystore into which to import your keys, use the keytool that ships with your Java installation to create an empty keystore:

```
keytool -genkey -alias foo -keystore keystore  
keytool -delete -alias foo -keystore keystore
```

**Tip**

Make sure to use the keytool commands for your Java version used to run Nexus. The documentation for keytool is available online for [Java 6](#) as well as [Java 7](#).

---

#### 24.2.3.3 Configuring Nexus Startup

Once both sets of SSL certificates are imported to your keystore and/or truststore, you can modify the *wrapper.conf* file located in `$NEXUS_HOME/bin/jsw/conf/` to inject the JSSE system properties necessary to use these certificates, as seen below adapting the iterator number (10, 11..) to start at the last used value, which depends on the rest of your configuration.

```
warpper.java.additional.10=-Djavax.net.ssl.keyStore=<keystore>
warpper.java.additional.11=-Djavax.net.ssl.keyStorePassword=< ↵
    keystore_password>
warpper.java.additional.12=-Djavax.net.ssl.trustStore=<truststore>
warpper.java.additional.13=-Djavax.net.ssl.trustStorePassword=< ↵
    truststore_password>
```

Once you have configured the Nexus startup option shown above, restart Nexus and attempt to proxy a remote repository which requires an SSL client certificate. Nexus will use the keystore location and keystore password to configure the SSL interaction to accept the server's SSL certificate and send the appropriate client SSL certificate using the manual configuration you have completed with the import-ssl tool.

## 24.3 Configuring Nexus to Serve via SSL

Providing access to the Nexus user interface and content via HTTPS only is a recommended best practice for any deployment.

The recommended approach to implementation is to proxy Nexus behind a server that is configured to serve content via SSL and leave Nexus configured for http. The advantage of this approach is that Nexus can easily be upgraded and there is no need to work with the JVM truststore. In addition, you can use the expertise of your system administrators and the preferred server for achieving the proxying, which in most cases will already be in place for other systems.

Common choices are servers like Apache httpd, nginx, Eclipse Jetty or even dedicated hardware appli-

---

ances. All of them can easily be configured to serve SSL content, and there is a large amount of reference material available for configuring these servers to serve secure content. For example, Apache httpd would be configured to use mod\_ssl.

Alternatively the Jetty instance that is part of the default Nexus install can be configured to serve SSL content directly, and if you would like to avoid the extra work of putting a web server like Apache httpd in front of Nexus, this section shows you how to do that.

---

**Tip**

Keep in mind that you will have to redo some of these configurations each time you upgrade Nexus, since they are modifications to the embedded Jetty instance located in `$NEXUS_HOME`.

---

To configure Nexus to serve SSL directly to clients, you need to perform the following steps:

As a first step you have to add the file `jetty-https.xml` to the Jetty startup configuration in `wrapper.conf` as detailed in Section [3.10.2](#).

Next, the HTTP port you want to use for the HTTPS connection has to be defined by setting the `application-port-ssl` property in `nexus.properties`.

```
application-port-ssl=8443
```

Now you are ready to create a keystore file. Instructions are available on the [Eclipse Jetty documentation site](#) or directly on the documentation site for the `keytool`. As a result of this procedure you will have a keystore file and the password values for `keyStorePassword`, `keyManagerPassword` and `trustStorePassword`.

Insert the values in the `jetty-https.xml` file in `NEXUS_HOME/conf`. The default configuration in that file suggests to create a subdirectory `NEXUS_HOME/conf/ssl` and copy the keystore file in there and rename it to `keystore.jks`. You can either do that or choose a different location or filename for your keystore file and update the paths for the `keystore` and `truststore` in the `jetty-https.xml` file.

Once this is all in place you can start up Nexus and access the user interface at e.g., `https://localhost:8443/nex`. If you have just created a self-signed certificate, modern web browsers will warn you about the certificate and you will have to acknowledge the fact that the certificate is self-signed. To avoid this behavior, you have to get a certificate signed by a signing authority or reconfigure the web browser.

---

Nexus is now available via HTTPS. If desired you can configure automatic redirection from HTTP to HTTPS by adding usage of `jetty-http-redirect-to-https.xml` as additional app parameters in `wrapper.conf` as well as update the `Base URL` in your Nexus server configuration.

If you are setting up this redirection and therefore aim to expose your Nexus server only via HTTPS you should also configure Nexus to mark session cookies as secure, so that they are only sent to HTTPS browser access. This prevents any session hijacking. You can configure this in `nexus.properties` with

```
shiro.secureSessionCookies=true
```

# Chapter 25

## Evaluating Nexus Step by Step

### 25.1 Prerequisites and Preparation

The following guide for evaluating Sonatype Nexus is based on an assumption of installing Nexus itself as well as the various technologies used in the specific evaluation example all on one computer. A more extended evaluation of Nexus in a team environment should follow the instructions for a full installation as documented in the book [Repository Management with Nexus](#). Consult the book for further in-depth documentation about all features of Nexus.

Besides the installation of Nexus itself, various evaluations will need different prerequisites installed on the machine you use for your evaluation. The installation instructions of these technologies follow below. Only follow the instructions referenced from the examples in which you are interested. For example you will only need to install Visual Studio and NuGet if you want to evaluate the .Net Integration of Nexus.

#### 25.1.1 A Note about the Operating System

Some of the tasks described are referencing command line calls. Where that is the case, this guide will use Unix typical commands and syntax as used on a bash shell. This is the most common environment on Linux and Mac OSX computers. On Windows machines, a bash shell can be installed as well, using the cygwin system. However the typical usage would be to use the Windows command prompt with slightly different calls. Table 25.1 displays a number of examples for typical tasks carried out in the evaluations

---

with their bash as well as Windows shell commands.

Table 25.1: Commandline Invocation Examples

Task	Bash Shell	Window Shell
Delete a file	rm filename	del filename
Delete a directory	rm -rf directoryname	rmdir directoryname
Delete a directory in users home directory	rm -rf ~/.m2/repository	rmdir /S %HOMEPATH%\ .m2\repository
Change to the users home directory	cd ~	cd %HOMEPATH%
Script invocation	./build	build.bat
Gradle Wrapper script invocation	./gradlew	gradlew.bat

## 25.1.2 Java Runtime

Nexus itself as well as some of the technologies used in the evaluation require a Java runtime or development kit, which is available for most operating systems. We recommend to install the latest Oracle Java 7 JDK available from the [download web page](#) and following the installation instructions on the same site.

After a successful installation, you can verify it by running the command `java -version`, which should result in an output similar to

```
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)
```



### Warning

Nexus requires Java 7.

### 25.1.3 Apache Maven

Apache Maven can be retrieved from the [download page](#) and installed following the instructions available there. We recommend the usage of the latest available Maven 3 version.

After a successful installation you can verify it with running the command `mvn --version`, which should result in an output similar to

```
Apache Maven 3.2.3 (33f...; 2014-08-11T13:58:10-07:00)
Maven home: /opt/tools/apache-maven-3.2.3
Java version: 1.7.0_65, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.7.0_65.jdk/Contents/Home ←
/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.8.5", arch: "x86_64", family: "mac"
```

### 25.1.4 Gradle

The examples in this guide use the so-called Gradle wrapper script. It allows you to get Gradle installed automatically by the wrapper and invoke all Gradle commands via it. To use it you simple invoke all `gradle` commands with `./gradlew` on Unix based systems and `gradlew.bat` on Windows instead of `gradle`.

Alternatively Gradle can be retrieved from the [download page](#) and installed following the instructions available in the [User Guide](#). We recommend the usage of the latest available Gradle version.

After a successful installation, you can verify it with running the command `gradle -v`, which should result in an output similar to

```
Gradle 2.0

Build time: 2014-07-01 07:45:34 UTC
Build number: none
Revision: b6ead6fa452dfdadec484059191eb641d817226c

Groovy: 2.3.3
Ant: Apache Ant(TM) version 1.9.3 compiled on December 23 2013
JVM: 1.7.0_65 (Oracle Corporation 24.65-b04)
OS: Mac OS X 10.8.5 x86_64
```

### 25.1.5 Apache Ant and Apache Ivy

Apache Ant can be retrieved from the [download page](#) and installed following the instructions available in the [manual](#). We recommend the usage of the latest available Ant version.

After a successful completion, you can verify your Ant installation by running the command `ant -version`, which should result in an output similar to

```
Apache Ant(TM) version 1.9.2 compiled on July 8 2013
```

The example projects used in this guide contain ant targets in their build files that will automatically install Apache Ivy as part of the build. Alternatively you can retrieve Apache Ivy from the [download page](#) and install it following the [instructions](#).

### 25.1.6 Microsoft Visual Studio and NuGet

Microsoft Visual Studio and NuGet are needed to evaluate the .Net support of Nexus Professional. There are a number of different Visual Studio distributions. Some of these distributions may have NuGet already installed, while others do not. Even if your Visual Studio installation is bundled with NuGet, you will want to make sure that you have upgraded to the latest version of the tool.

NuGet is a fast-paced project, and you'll find that new packages available on NuGet Gallery may not be compatible with older versions of the NuGet package manager.

For detailed instructions on installing NuGet in Visual Studio, please go to the [NuGet project's documentation site](#) and refer to the [Installing NuGet](#) instructions.

## 25.2 Getting Started

This guide is based on the usage of Nexus Professional. A lot of the core features are available in Nexus Open Source as well and some examples are suitable to assess the open source version as well.

- **Step 1:** [Download the Nexus Professional Trial Installer](#) for your operating system.
  - **Step 2:** Run the Nexus Professional Installer.
-

- **Step 3:** Start Nexus from the Nexus Professional Installer.

When Nexus has started just click the URL in the wizard or go to <http://localhost:8081/nexus> in a browser window.

---

**Note**

This guide and the examples reference the Nexus URL <http://localhost:8081/nexus>. If you have chosen to use a different port during the installation of the trial simply change the URLs.

---

Below are several directories to know:

Nexus Installation Directory: This is where the Nexus application files are installed on your system. We refer to this as `<nexus_install>`.

Nexus Work Directory: This directory contains your specific Nexus instance configuration files. We refer to this as `<nexus_work>`.

Nexus Eval Guide Directory: This directory contains supporting sample project files and this document. We refer to this as `<nexus_eval>`.

---

**Note**

You can locate these directories by viewing the Nexus Control Panel.

---

In case something goes wrong and Nexus seems to be unavailable, you can examine the following two log files to diagnose problems.

```
<nexus_work>/logs/nexus-launcher.log  
<nexus_work>/logs/nexus.log
```

Nexus tries to listen on port 8081. If you have another application listening on this port, Nexus will not be able to start. You can change the port Nexus listens on. Open this file

```
<nexus_install>/conf/nexus.properties
```

Edit the line that looks like this:

---

```
application-port=8081
```

For example, to access Nexus on port 9090 instead, change the line to

```
application-port=9090
```

Save the file and restart Nexus.

### 25.2.1 Activating Your Nexus Trial

Once Nexus is started and you are accessing the user interface the first time, you will see the trial activation form. Provide your full name, email address, organization, and location and click on *Submit Activation Request*.

You will immediately receive an email from Sonatype with the subject “Your Nexus Professional Trial License,” which contains your trial license key. Paste this license key into the license field in the Nexus Professional user interface. Click *Activate* to activate your 14-day Nexus Professional trial. Once your trial is activated, you will be presented with the Nexus user interface.

### 25.2.2 Logging Into Nexus as an Administrator

After activating your Nexus install, you can log into Nexus as an administrator. Go to <http://localhost:8081/nexus/> and click on the Login button in the upper right-hand corner of the interface.

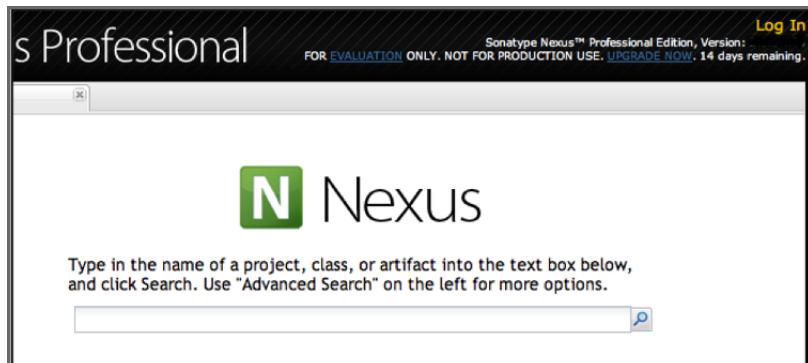


Figure 25.1: Nexus User Interface with Login

The default administrator username is `admin` and password is `admin123`.

The Nexus Professional Trial evaluation guide assumes that you are logged in as an administrator.

### 25.2.3 Getting Started with Your Nexus Professional Evaluation

To make it easier to evaluate Nexus, we've created a set of projects to demonstrate the features of Nexus Open Source and Nexus Professional. These example projects are bundled with the trial installer for your convenience.

In addition, they are available as the `nexus-book-examples` project on GitHub at <https://github.com/sonatype/nexus-book-examples> for you to download and inspect separately, if desired. The latest version of all the examples is available as a zip archive at <https://github.com/sonatype/nexus-book-examples/archive/master.zip>.

When you downloaded the trial distribution of Nexus Professional, your server is also preconfigured to demonstrate important features.

The Nexus trial distribution contains the following customizations:

- Nexus has been preconfigured to download the search index from the Central Repository.
  - A Staging profile has been configured to demonstrate release management.
-

- Nexus proxies NuGet Gallery so that you can quickly evaluate support for .NET development.

## 25.3 The Basics: Proxying and Publishing

After a few weeks the importance of having a repository manager is so obvious no one on my team can believe we used to develop software without one.

— Build Engineer *Financial Industry*

If you are new to repository management, the first step is to evaluate the two basic benefits of running a repository manager: proxying and publishing.

You can reap these benefits with any Java/JVM build system that includes declarative dependency management and understands the Maven repository format. In the following we are going to cover the details for Apache Maven, Gradle and Apache Ant/Apache Ivy based builds. Build tools like SBT, Leiningen, Gant/Grails and others can be configured to do the same and get access to the same benefits.

### 25.3.1 Proxying Components

If you use a dependency in your software, your build downloads components from a remote repository, such as the [Central Repository](#) and others. Your systems depend on these components. If one of these critical remote repositories becomes unavailable, your productivity can grind to a halt.

This is where Nexus can help. Nexus is preconfigured to proxy the Central Repository, and other remote repositories can be easily added. Once set up, Nexus maintains a local cache of the needed components from the remote repositories for you. Your build is more reliable when all the components you require are cached by Nexus. It is providing you with dramatic efficiency and speed improvements across your entire development effort.

#### In this example, you will...

- Configure your build to download components from Nexus.
  - Pre-cache dependencies and build components with an initial build.
  - Note organization-wide improvements in build reliability.
-

**Let's get started using the provided scripts:**

The eval bundle includes an installation of Apache Maven as well scripts that isolate your evaluation from the rest of your system and make it extremely easy for you to follow. The Gradle examples use a wrapper script to allow you to simply follow the example. To follow the Ant/Ivy examples you will have to install Apache Ant as explained in Section [25.1.5](#).

1. Go to the Nexus evaluation guide directory you configured during the Nexus Professional install, which is named evalguide by default and can be found in your users home directory, and run the command:

```
$ cd maven  
$ ./build -f simple-project/pom.xml clean install
```

To use Apache Maven or if you want to try Gradle use

```
$ cd gradle/simple-project  
$ ./gradlew build
```

With Apache Ant and Ivy you can run

```
$ cd ant-ivy/simple-project  
$ ant jar
```

2. As the project builds, you will notice that all components are downloaded from your local Nexus instance installed with requests from Apache Maven:

```
Downloading: http://localhost:8081/nexus/content/groups/public/org  
    /apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5. ←  
        pom  
Downloaded: http://localhost:8081/nexus/content/groups/public/org  
    /apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5. ←  
        pom  
(4 KB at 1.3 KB/sec)  
...
```

Here are examples from Gradle:

```
Download http://localhost:8081/nexus/content/groups/public/org/  
    codehaus/jackson/jackson-core-asl/1.8.0/jackson-core-asl-1.8.0.jar  
Download http://localhost:8081/nexus/content/groups/public/org/  
    codehaus/jackson/jackson-mapper-asl/1.8.0/jackson-mapper-asl-1.8.0. ←  
        jar  
Download http://localhost:8081/nexus/content/groups/public/com/  
    google/sitebricks/sitebricks-converter/0.8.5/sitebricks-converter ←  
        -0.8.5.jar  
...
```

Here are examples from Apache Ivy:

```
[ivy:retrieve] downloading http://localhost:8081/nexus/content/
  groups/public/asm/asm-commons/3.2/asm-commons-3.2.jar ...
[ivy:retrieve] .. (32kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] asm#asm-commons;3.2!asm-commons.jar (313 ←
  ms)
...
...
```

3. After the build has successfully completed, delete the local Maven repository cache in the eval guide directory and rerun the build as before

```
$ cd maven
$ rm -rf repository
```

Delete the Gradle cache with

```
$ rm -rf ~/.gradle
```

Delete the Ivy cache with

```
$ ant clean-cache clean
```

4. Notice how the downloads are occurring much faster. The components are no longer retrieved from the remote repositories before being served by Nexus, but they are supplied straight from the proxy repository cache in Nexus.
5. To verify that components are being cached in Nexus, open the Repositories panel by clicking on *Repositories* in the left-hand Nexus menu. Once the list of repositories is displayed, select Central. Click on the *Browse Storage* tab and observe the tree of components downloaded and successfully cached in Nexus.

### Alternatively using your own Apache Maven setup:

1. Ensure that Apache Maven is installed as a prerequisite as documented in Section [25.1.3](#).
2. Go to the Nexus evaluation guide directory you configured during the Nexus Professional install and configure Maven to access Nexus with the provided *settings.xml*. Ensure to back up any existing settings file and adapt the port in the mirror url, if you have chosen to use a different port than 8081 in the Nexus trial installer.

```
$ cp maven/settings/settings.xml ~/.m2/
```

3. Optionally, if you do not want to use the default local repository location of Maven in `~/.m2/repository`, change the `localRepository` settings in the *settings.xml* file to an absolute path.

4. Build the simple-project, and observe the downloads from the Nexus repository.

```
$ cd maven/simple-project/  
$ mvn clean install
```

5. After the build has successfully completed, delete the local Maven repository cache and rerun the build. Notice the improved build performance and the cached components in Nexus.

```
$ rm -rf ~/.m2/repository
```

## Conclusion

Your builds will be faster and more reliable now that you are caching components in Nexus and retrieving them from there. Once Nexus has cached a component locally, there is no need to make another roundtrip to the remote repository server. The caching benefits all tools configured to access Nexus.

### 25.3.2 Publishing Components

Nexus makes it easier to share components internally. How do you distribute and deploy your own applications? Without Nexus, internal code is often distributed and deployed using an SCM, a shared file system, or some other inefficient method for sharing binary components.

With Nexus you create hosted repositories, giving you a place to upload your own components to Nexus. You can then feed your components back into the same repositories referenced by all developers in your organization.

**In this example, you will...**

- Publish a component to Nexus.
- Watch another project download this component as a dependency from Nexus.

**Let's get started using the provided scripts:**

1. Follow the proxying evaluation example from Section [25.3.1](#).

2. Go to the Nexus evaluation guide directory and publish the simple-project to Nexus with the Maven wrapper script.

```
$ cd maven  
$ ./build -f simple-project/pom.xml clean deploy
```

With your own Maven installation you can use

```
$ cd maven/simple-project/  
$ mvn clean deploy
```

To deploy the project with Gradle, you can run the commands

```
$ cd gradle/simple-project  
$ ./gradlew upload
```

The equivalent Ant invocation is

```
$ cd ant-ivy/simple-project  
$ ant deploy
```

3. The simple-project has been preconfigured to publish its build output in the form of a JAR component to your local instance of Nexus Professional.
4. Observe how the build tools log the deployment to Nexus, e.g., Maven

```
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/  
org/sonatype/nexus/examples/simple-project/1.0.0-SNAPSHOT/  
simple-project-1.0.0-20130311.231302-1.jar  
Uploaded: http://localhost:8081/nexus/content/repositories/snapshots/  
org/sonatype/nexus/examples/simple-project/1.0.0-SNAPSHOT/  
simple-project-1.0.0-20130311.231302-1.jar (3 KB at 38.2 KB/sec)
```

Gradle

```
Uploading:  
org/sonatype/nexus/examples/simple-project/1.0-SNAPSHOT/  
simple-project-1.0-20130306.173412-1.jar  
to repository remote at  
http://localhost:8081/nexus/content/repositories/snapshots
```

or Ivy

```
[ivy:publish] :: publishing :: org.sonatype.nexus.examples#simple- ←  
project  
[ivy:publish] published simple-project to http://localhost:8081  
/nexus/content/repositories/snapshots/org/sonatype/nexus/examples/  
simple-project/1.0-SNAPSHOT/simple-project-1.0-SNAPSHOT.jar
```

5. To verify that the simple-project component was deployed to Nexus, click on Repositories and then select the Snapshots repository. Select the Browse Storage tab as shown in this illustration.

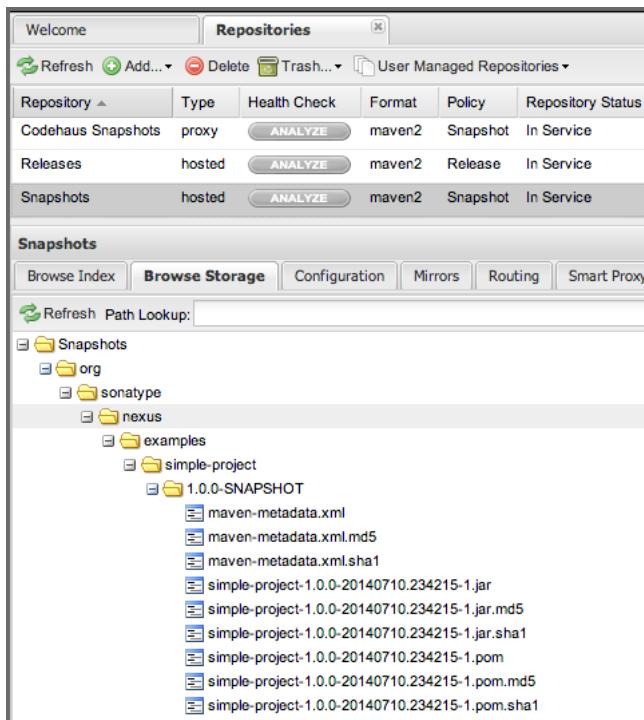


Figure 25.2: Successfully Deployed Components in the Snapshots Repository

6. Once this component has been published, return to the evaluation sample projects directory and run a build of another-project:

```
$ cd maven  
$ build -f another-project/pom.xml clean install
```

With your own Maven installation you can use

```
$ cd maven/another-project  
$ mvn clean install
```

To build the second project with Gradle, simply use

```
$ cd gradle/another-project  
$ ./gradlew build
```

Perform the same action with Ant using

```
$ cd ant-ivy/another-project  
$ ant jar
```

7. This second project has a dependency on the first project declared in the Maven pom.xml with

```
<dependency>  
    <groupId>org.sonatype.nexus.examples</groupId>  
    <artifactId>simple-project</artifactId>  
    <version>1.0.0-SNAPSHOT</version>  
</dependency>
```

and in the Gradle build.gradle file as

```
dependencies {  
    compile "org.sonatype.nexus.examples:simple-project:1.0.0-SNAPSHOT" ←  
    "  
}
```

Ivy declares the dependency in ivy.xml and it looks like this

```
<dependencies>  
    <dependency org="org.sonatype.nexus.examples" name="simple-project"  
        rev="1.0.0-SNAPSHOT"/>  
</dependencies>
```

During the build, it is relying on Nexus when it attempts to retrieve the component from simple-project.

Now that you are sharing components of your projects internally, you do not need to build each other's software projects anymore. You can focus on writing the code for your own components and the integration of all components to create a larger software component. In fact, it does not even matter which build tool created the component, since the Maven repository format is understood by all of them.

## Conclusion

Sonatype Nexus Open Source and Professional can serve as an important tool for collaboration between different developers and different development groups. It removes the need to store binaries in source control or shared filesystems and makes collaboration more efficient.

## 25.4 Governance

### 25.4.1 Identify Insecure OSS Components In Nexus

The Repository Health Check in Nexus Professional turns your repository manager into the first line of defence against security vulnerabilities. Nexus Professional scans components and finds cached components with known vulnerabilities from the Common Vulnerabilities and Exposures (CVE) database. You can get an immediate view of your exposure from the Repository Health Check summary report with vulnerabilities grouped by severity according to the Common Vulnerability Scoring System (CVSS).

As your developers download components, they may be unwittingly downloading components with critical security vulnerabilities that might expose your applications to known exploits. According to a joint study by Aspect Security and Sonatype released in 2012, Global 500 corporations downloaded 2.8 million flawed components in one year. Nexus becomes an effective way to discover flawed components in your repositories allowing you to avoid falling victim to known exploits.

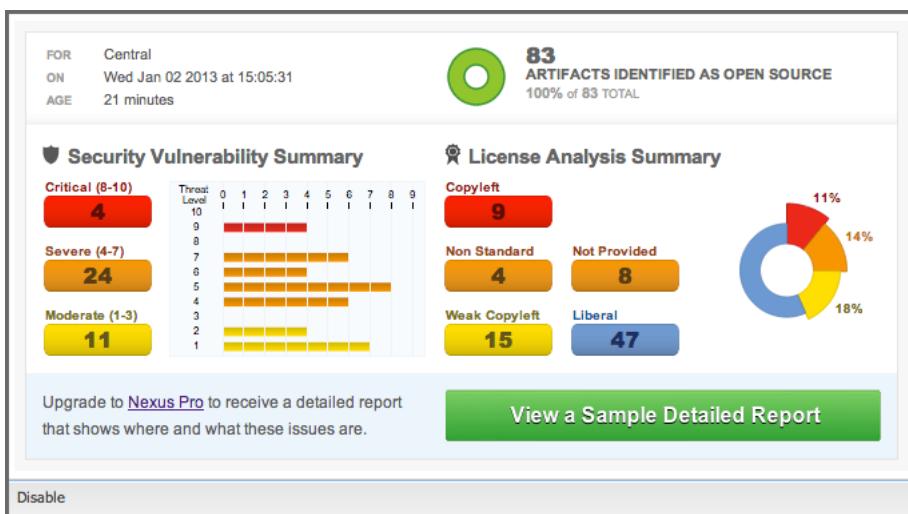


Figure 25.3: Repository Heath Check Summary

#### In this example, you will...

- Start an analysis of all components proxied from the Central Repository.

- Inspect the number of security vulnerabilities found.

## Let's get started

1. Follow the proxying examples in Section 25.3 to seed the Central proxy repository of your Nexus instance. These examples include several components with security vulnerabilities and license issues as dependencies.
2. Once your Nexus instance has cached the components, open the Nexus interface, log in as administrator and click on the green Analyze button next to your Central proxy repository.
3. After the completion of the analysis, the button will change into an indicator of the number of security and license issues found.
4. Hover your mouse over the indicator and Nexus will show you a summary report detailing the number and type of security vulnerabilities present in your repository.
5. Optionally, build some of your own applications to get further components proxied and see if additional security issues appear.

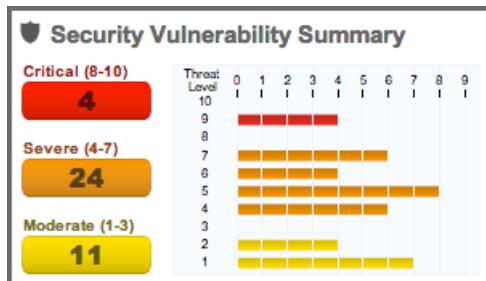


Figure 25.4: Security Vulnerability Summary Display from Repository Health Check

Nexus Professional users gain access to further details about all the components with security vulnerabilities, including their repository coordinates to uniquely identify the component as well as links to the vulnerability database records for further details.

**Conclusion**

The Repository Heath Check of Nexus allows you to get an understanding of all the security vulnerabilities affecting the components you have proxied into your environment and which might potentially be part of the software you are creating, distributing, and deploying in production environments.

### 25.4.2 Track Your Exposure To OSS Licenses

With Open Source Software (OSS) component usage as the de facto standard for enterprise application development, the importance of tracking and identifying your exposure to OSS licenses is an essential part of the software development lifecycle. Organizations need tools that let them govern, track, and manage the adoption of open source projects and the evaluation of the licenses and obligations that are part of OSS development and OSS component usage.

With Nexus Professional's Repository Health Check, your repository becomes more than just a place to store binary components. It becomes a tool to implement policies and govern the open source licenses used in development to create your applications.

**In this example, you will...**

- Start an analysis of all components proxied from the Central Repository
- Inspect the number of license issues found.

**Let's get started**

1. Follow the proxying examples in Section [25.3](#) to seed the Central proxy repository of your Nexus instance. These examples include several components with security vulnerabilities and license issues as dependencies.
2. Once your Nexus instance has cached the components, log in to the Nexus interface as administrator and click on the green Analyze button next to your Central proxy repository in the *Repositories* list.
3. After the completion of the analysis, the button will change into an indicator of the number of security and license issues found.
4. Hover your mouse over the indicator and Nexus will show you a summary report detailing the number and type of license issues of components present in you repository.

5. Optionally, build some of your own applications to get further components proxied and see if additional license issues appear.

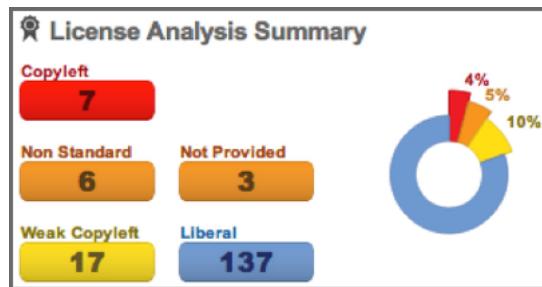


Figure 25.5: License Analysis Summary Display from Repository Health Check

Nexus Open Source and the trial version show the summary information found by the analysis.

Nexus Professional customers can access a detailed report to identify specific components with known security vulnerabilities or unacceptable licenses. The component lists can be sorted by OSS license or security vulnerabilities, and Nexus Professional provides specific information about licenses and security vulnerabilities. A detailed walkthrough of this report is available on the [Sonatype website](#).

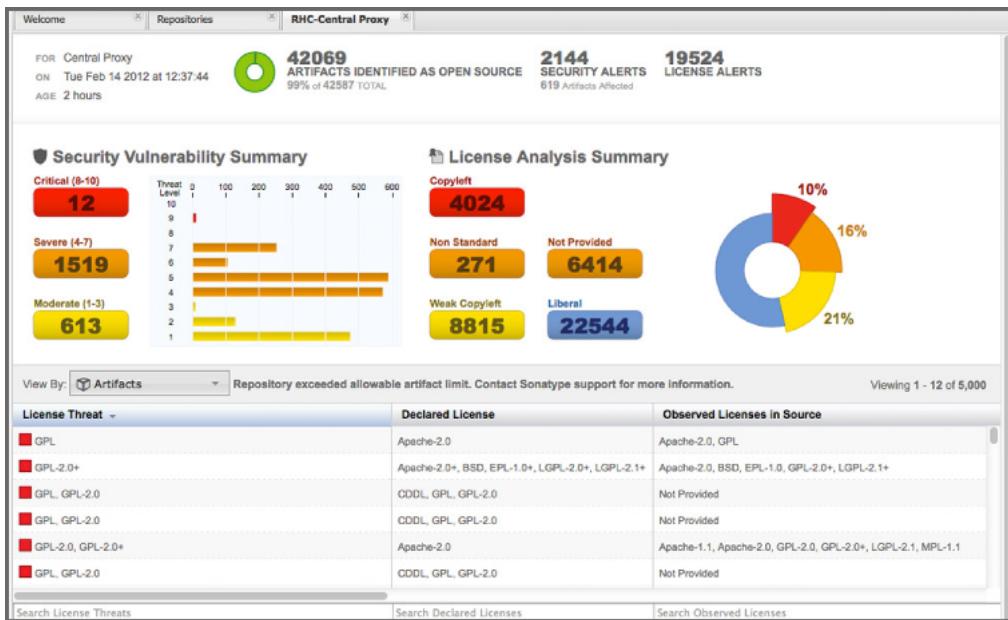


Figure 25.6: Repository Health Check Details with License Issues List

## Conclusion

OSS License compliance and security assessments are not something you do when you have the time. It is something that should be a part of your everyday development cycle, as it is with Nexus Professional's Repository Health Check.

## 25.5 Process Improvements

### 25.5.1 Grouping Repositories

Once you have established Nexus and set up your build, provisioning system, and other tools to connect to Nexus, you can take advantage of Nexus repository groups. The best practice to expose Nexus is to get users to connect to the Public Repositories group as configured in the settings.xml as documented in Section 25.3.1.

When all clients are connecting to Nexus via a group, you can easily provide additional repository content to all users by adding new repositories to the group.

For example, imagine a group in your organization is starting to use components provided by the JBoss release repository available at <https://repository.jboss.org/nexus/content/repositories/releases/>. The developers are already accessing Nexus via the public group. All you have to do is to create a new proxy repository for the JBoss release repository and add it to the public group and all developers, continuous integration (CI) servers and other tools will have access to the additional components.

To add the Grails repositories, proxy them and add them to the group. The same approach applies to proxy Clojars or other repository of a business partner or supplier who is protected by user credentials.

Another advantage of groups is that you can mix release and snapshot repositories and easily expose all the components via one easy access point.

Besides using the default public group, you can create additional groups that expose other contexts. An example would be to create a group for all staged releases allowing a limited number of users access to your release artifacts as part of the release process.

### Conclusion

Using groups allows you to expose multiple repositories, mix snapshot and release components and easily administrate it all on the Nexus server. This allows you to provide further components to your developers or other users, without requiring a change on these client system, tremendously simplifying the administration effort.

### 25.5.2 Staging a Release with Nexus

When was the last time you did a software release to a production system? Did it involve a QA sign-off? What was the process you used to redeploy, if QA found a problem at the last minute? Developers often find themselves limited by the amount of time it takes to respond and create incremental builds during a release.

The Nexus Staging Suite changes this by providing workflow support for binary software components. If you need to create a release component and deploy it to a hosted repository, you can use the Staging Suite to post a release, which can be tested, promoted, or discarded, before it is committed to a release repository.

**In this example, you will...**

- Configure a project to publish its build output component to Nexus.
- Deploy a release and view the deployed component in a temporary staging repository.
- Promote or discard the contents of this temporary staging repository.

**Let's get started using the provided scripts:**

1. This example assumes that you have successfully deployed the simple-project as documented in Section [25.3.1](#).
2. Inspect the preconfigured *Example Release Profile* staging profile by selecting it from the list available after selecting *Staging Profiles* in the the *Build Promotion* menu in the left-hand navigation.
3. Notice that the version of the simple-project in the pom.xml ends with -SNAPSHOT. This means that it is in development.
4. Change the version of the simple project to release version by removing the -SNAPSHOT in a text editor or run the command

```
$ ./build -f simple-project/pom.xml versions:set -DnewVersion=1.0.0
```

5. Publish the release to the Nexus Staging suite with

```
$ ./build -f simple-project/pom.xml clean deploy
```

6. To view the staging repository, click on *Staging Repositories* in the *Build Promotion* menu and you should see a single staging repository as shown in this illustration.
7. Click on *Close* to close the repository and make it available via the public group.
8. Experiment with Staging, at this point you can:
  - a. Click on *Drop* to discard the contents of the repository and be able to stage another release.
  - b. Click on *Release* to publish the contents of the repository to the release repository.
9. Once you release the staging repository, you will be able to find the release components in the *Releases* hosted repository.

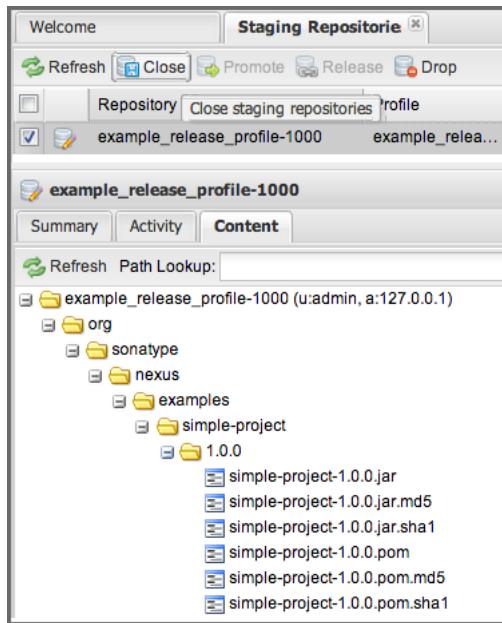


Figure 25.7: Closing a Staging Repository in the Nexus User Interface

The individual transactions triggered by closing, dropping, promoting, or releasing a staging repository can be enriched with email notifications as well as staging rule inspections of the components.

#### Alternatively using your own Apache Maven setup:

1. Follow the steps described above with the modification of setting the new version with

```
$ cd maven/simple-project  
$ mvn versions:set -DnewVersion=1.0.0
```

2. And publishing to the Nexus Staging suite with

```
$ mvn clean deploy
```

**Conclusion**

Staging gives you a standard interface for controlling and managing releases. A collection of related release components can be staged for qualification and testing as a single atomic unit. These staged release repositories can be discarded or released pending testing and evaluation.

### 25.5.3 Hosting Project Web Sites

Nexus Professional and Open Source can be used as a publishing destination for project websites. You don't have to worry about configuring another web server or configuring your builds to distribute the project site using a different protocol. Simply point your Maven project at Nexus and deploy the project site.

With Nexus as a project's site hosting solution, there's no need to ask IT to provision extra web servers just to host project documentation. Keep your development infrastructure consolidated and deploy project sites to the same server that serves your project's components.

You can use this feature internally, but it is even better suited if you are providing an API or components for integration. You can host full project websites with JavaDoc and any other desired documentation right with the components you provide to your partners and customers.

**In this example, you will...**

- Create a Hosted repository with the Maven Site provider.
- Configure your project to publish a website to Nexus Professional.

**Let's get started using the provided scripts:**

1. Create a hosted repository with the *Site* format and a *Repository ID* called *site* → [Read more...](#)
2. Deploy the simple-project component and site to Nexus:  

```
$ ./build -f simple-project/pom.xml clean deploy site-deploy
```
3. Browse the generate site on Nexus at <http://localhost:8081/nexus/content/sites/site/>
4. Optionally, configure your own Maven project to deploy a site to Nexus → [Read more...](#)

5. Publish it to Nexus → [Read more...](#)

#### **Alternatively using your own Apache Maven setup:**

1. Follow the steps described above with the modification of deploying the site with

```
$ cd maven/simple-project  
$ mvn clean deploy site-deploy
```

#### **Conclusion**

If your projects need to publish HTML reports or a project web site, Nexus provides a consolidated target for publishing project-related content.

### **25.5.4 Process and Security Improvements with Maven Settings Management and User Token**

The Maven `settings.xml` file plays a key role for retrieving as well as deploying components to Nexus. It contains `<server>` sections that typically contain the username and password for accessing Nexus in clear text. Especially with single sign-on (SSO) solutions used for Nexus authentication, this is not desirable. In addition, security policies often mean that the file regularly needs to be updated.

The User Token feature of Nexus Professional allows you to replace the SSO username and password with Nexus-specific tokens that are autogenerated and managed by Nexus.

Furthermore, the Nexus Maven Settings Management allows you to manage Maven Settings. Once you have developed a Maven Settings template, developers can connect to Nexus Professional using the Nexus M2Settings Maven plugin that will take responsibility for downloading a Maven Settings file from Nexus and replacing the existing Maven Settings on a local workstation. It can be configured to automatically place your user tokens in the `settings.xml` file.

#### **In this example, you will...**

- Explore the configuration of a Maven Settings template in Nexus Professional.
-

- Activate and access your user token.

## Let's get started

1. Log into Nexus as administrator and access the *Maven Settings* administration via the item in the *Enterprise* menu.
2. Press the *Add* button, provide a name and edit the new settings file.
3. Add the server section:

```
<servers>
  <server>
    <id>nexus</id>
    <!-- User-token: ${userToken} -->
    <username>${userToken.nameCode}</username>
    <password>${userToken.passCode}</password>
  </server>
</servers>
```

4. Read more about potential configuration and usage in [Manage Maven Settings Templates](#)
5. Downloading the settings template requires Nexus running via HTTPS and can then be performed with the command below and following the prompts:

```
mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:1.6.2:download ←
-Dsecure=false
```

6. Note that the *secure* option is set to *false* for your evaluation. The plugin would otherwise abort for using the insecure HTTP protocol once you provide your evaluation Nexus url of `http://localhost:8081/nexus`. For a production usage, we recommend using the secure HTTPS protocol for your Nexus deployments.
7. Find out more about the usage in Download Settings from Nexus → [Read more...](#)
8. Activate User Token in the configuration in the *Security* menu *User Token* administration by checking the *Enabled* box and pressing the *Save* button.
9. Access your *User Profile* in the drop-down of your user name in the top right-hand corner of the Nexus user interface.
10. Use the drop-down in the *Profile* panel to access *User Token*.
11. In the *User Token* screen, press *Access User Token*, provide your username and password again, and inspect the tokens in the pop-up dialog.

**Conclusion**

The distribution of `settings.xml` is a crucial part of the rollout of Nexus usage. With the help of the the Nexus M2Settings Maven Plugin and the server side settings template, it is possible to automate initial distribution as well as updates to the used `settings.xml` files. The User Token feature allows you to avoid having SSO credentials exposed in your file system at all.

## 25.6 .NET Integration

### 25.6.1 Consume .NET Components from NuGet Gallery

The NuGet project provides a package and dependency management solution for .NET developers. It is integrated directly into Visual Studio and makes it easy to add, remove and update libraries and tools in Visual Studio and on the command line for projects that use the .NET Framework. Nexus can act as a proxy between your developer's Visual Studio instances and the public NuGet Gallery.

When you configure Nexus Professional to act as a proxy for NuGet Gallery you gain a more reliable build that depends on locally cached copies of the components on which you depend. If NuGet Gallery has availability problems, your developers can continue to be productive. Caching components locally will also result in a faster response for developers downloading .NET dependencies.

**In this example, you will...**

- Configure your Visual Studio instance to download NuGet packages from your local Nexus server.
- Consume components from NuGet Gallery via Nexus.

**Let's get started**

Your Nexus Professional Trial instance has been preconfigured with the following NuGet repositories:

- A Proxy Repository for NuGet Gallery
- A Hosted Repository for your internal .NET components

- A Group which combines both the NuGet Gallery Proxy and the Hosted NuGet Repository

The screenshot shows the 'Repositories' section of the Nexus Professional interface. At the top, there's a search bar with the text 'nuget'. Below it is a table with columns: Repository, Type, H..., Format, ..., Repository Status, and Repository Path. The table contains four rows:

Repository	Type	H...	Format	...	Repository Status	Repository Path
NuGet Public Group	group		nuget		In Service	<a href="http://localhost:8081/nexus/content/group/">http://localhost:8081/nexus/content/group...</a>
NuGet Gallery	proxy		nuget		In Service	<a href="http://localhost:8081/nexus/content/repo...">http://localhost:8081/nexus/content/repo...</a>
NuGet Releases	hosted		nuget		In Service	<a href="http://localhost:8081/nexus/content/repo...">http://localhost:8081/nexus/content/repo...</a>
NuGet Third Party	hosted		nuget		In Service	<a href="http://localhost:8081/nexus/content/repo...">http://localhost:8081/nexus/content/repo...</a>

Figure 25.8: NuGet Repositories in Repository List Accessed Using the List Filter Feature

To consume .NET components from Nexus you will need to install the NuGet feature in Visual Studio as referenced in Section [25.1.6](#) and configure it appropriately:

1. Open Nexus Professional, click on *Repositories* in the left-hand navigation menu and locate the *NuGet Group* repository group. This is the aggregating group from which Visual Studio should download packages. Click on this repository group in the list of repositories.
2. Select the NuGet tab below the list of repositories with the NuGet Group selected and copy the URL in the *Package Source* field to your clipboard. The value should be `http://localhost:8081/nexus/serv...`
3. Now in Visual Studio, right-click on a Visual Studio project and select *Add Library Reference*.
4. In the *Add Library Package Reference*, click on the *Settings* button in the lower left-hand corner.
5. This will bring up an *Options* button. Remove the initial NuGet repository location and replace it with a reference to your Nexus instance. Clicking *Add* to add the reference to your Nexus instance.
6. Click *OK* to return to the *Add Library Package Reference* dialog.
7. Select the *Online* item in the left-hand side of the dialog. At this point Visual Studio will interrogate your Nexus instance for a list of NuGet packages.
8. You can now locate the package you need and install it.
9. To verify that the NuGet package components are being served from Nexus you can return to the Nexus web interface and browse the local storage of your NuGet proxy repository.

---

**Note**

Watch [this video](#) of the steps being performed in Visual Studio.

---

The above instructions were created using Visual Studio 10 Web Developer Express. Your configuration steps may vary if you are using a different version of Visual Studio.

### Conclusion

When your developers are consuming OSS .NET components through a Nexus proxy of NuGet gallery your builds will become more stable and reliable over time. Every component will be downloaded to Nexus only once, and every following download will enjoy the performance and reliability of a local download from the Nexus cache.

## 25.6.2 Publish and Share .NET Components with NuGet

Nexus Professional can improve collaboration and control, while increasing the speed of .NET development. NuGet defines a packaging standard that organizations can use to share components.

If your organization needs to share .NET components, you can publish these components to a hosted NuGet repository on Nexus Professional. This makes it easy for projects within your organization to start publishing and consuming NuGet packages using Nexus as a central hub for collaboration.

Once NuGet packages are published to your Nexus Professional instance they are automatically added to the NuGet repository group, making your internal packages as easy to consume as packages from NuGet Gallery.

### In this example, you will...

- Publish NuGet packages to a Hosted NuGet repository.
- Distribute custom .NET components using Nexus.

### Let's get started:

1. Follow the example from Section [25.6](#) to set up proxying of NuGet packages from Nexus.
2. Activate the NuGet API Security Realm → [Read more...](#)
3. Create a NuGet Package in Visual Studio. Creating a package for deployment can be done with the pack command of the nuget command line tool or within Visual Studio. Detailed documentation can be found on the [NuGet website](#).

4. Publish a NuGet Package to Nexus Professional → [Read more...](#)

## Conclusion

Once NuGet packages are published to your Nexus Pro instance and are available via a NuGet repository group, your internal packages will be as easy to consume as packages from NuGet Gallery.

This will greatly improve sharing of components and reuse of development efforts across your teams and allow you to modularize your software.

### 25.6.3 Security

#### 25.6.3.1 Integration with Enterprise LDAP Solutions

Organizations with large, distributed development teams often have a variety of authentication mechanisms, from multiple LDAP servers with multiple User and Group mappings, to companies with development teams that have been merged during an acquisition. Nexus Professional's Enterprise LDAP support was designed to meet the most complex security requirements and give Nexus administrators the power and flexibility to adapt to any situation.

Nexus Professional offers LDAP support features for enterprise LDAP deployments including detailed configuration of cache parameters, support for multiple LDAP servers and backup mirrors, the ability to test user logins, support for common user/group mapping templates, and the ability to support more than one schema across multiple servers.

#### Let's get started

Read more about [configuring Enterprise LDAP](#) to learn about the following:

- Configuring LDAP caching and timeout.
- Configuring and testing LDAP failover.
- Using LDAP user and group mapping templates for Active Directory, POSIX with dynamic or static groups or generic LDAP configuration.

With Enterprise LDAP support in Nexus Professional, you can do the following:

- Cache LDAP authentication information.
- Use multiple LDAP servers, each with different User and Group mappings.
- Use LDAP servers with multiple backup instances and test the ability of Nexus to failover in the case of an outage.
- Augment the roles from LDAP with Nexus specific privileges.

### Conclusion

When you need LDAP integration, you will benefit from using Nexus Professional. Nexus Professional can support the largest development efforts, with some of the most complex LDAP configurations, including multiple servers and support for geographic failover and does so in production with many users every day.

#### 25.6.3.2 Integration with Atlassian Crowd

If your organization uses Atlassian Crowd, Nexus Professional can delegate authentication and access control to a Crowd server by mapping Crowd groups to Nexus roles.

#### Let's get started

1. Configure the Crowd Plugin → [Read more...](#)
2. Map Crowd Groups to Nexus Roles → [Read more...](#)
3. Add the Crowd Authentication Realm → [Read more...](#)

### Conclusion

If you've consolidated authentication and access control using Atlassian Crowd, take the time to integrate your repository manager with it as well. Nexus Professional's support for Crowd makes this easy.

## 25.6.4 Enterprise Deployments

### 25.6.4.1 Scaling Nexus Deployments for Distributed Development

Avoid downtime by deploying Nexus in a highly available configuration! With the Nexus Professional feature Smart Proxy, two distributed teams can work with local instances of Nexus that will inform each other of new components as they are published. Smart Proxy is an enhanced proxy setup with push notifications and potential prefetching of components. It allows you to keep proxy repositories on multiple Nexus servers in sync without sacrificing performance.

A team in New York can use a Nexus instance in New York and a team in Sydney can use an instance in Australia. If a component has been deployed, deleted, or changed, the source repository notifies the proxy. Both teams are assured that Nexus will never serve stale content. This simple mechanism makes it possible to build complex distributed networks of Nexus instances relying on this publish/subscribe approach.

#### In this example, you will...

- Setup two instances of Nexus Professional.
- Configure one instance to proxy the hosted instances of the other instance.
- Configure the proxying instance to subscribe to Smart Proxy events.

#### Let's get started

1. Enable Smart Proxy publishing → [Read more...](#)
2. Establish trust between Nexus instances → [Read more...](#)
3. Configure Smart Proxy for specific repositories → [Read more...](#)

#### Conclusion

With Smart Proxy, two or more distributed instances of Nexus can stay up to date with the latest published components. If you have distributed development teams, Smart Proxy will allow both teams to access a high-performance proxy that is guaranteed to be up to date.

## Chapter 26

# Nexus Community

### 26.1 Introduction

Sonatype Nexus Open Source as well Sonatype Nexus Professional are widely used and adapted to the varied circumstances and requirements at open source projects and organizations and small to very large enterprises.

Integrating Nexus and expanding its features is encouraged and enabled by the availability of Nexus Open Source under the Eclipse Public License, [the REST API](#) of Nexus and [the support for plugins](#) as part of Nexus itself, including writing your own plugins.

A number of tools are available to facilitate the community of Nexus users.

#### Mailing Lists

A number of mailing lists are available for the Nexus community:

##### **Nexus Users**

General discussion and support for anyone using Nexus - [Subscribe](#) or [Browse the Archive](#).

##### **Nexus Developers**

General discussion and support for anyone who wants to get involved in the development of Nexus - [Subscribe](#) or [Browse the Archive](#).

##### **Nexus Professional Users**

General discussion and support for professional Nexus users - [Subscribe](#).

### Chat

Sonatype provides a live chat channel to connect to other users and developers as well as Sonatype support and development staff.

### Source Code

The Nexus Open Source codebase is a great reference for your development of custom integrations and plugins. It is available on GitHub at <https://github.com/sonatype/nexus-oss>.

## 26.2 Community Projects Overview

Nexus community projects range from open source efforts run by Sonatype, projects run by Nexus Professional customers or Nexus Open Source users to one man, one-off hacks for some older version of Nexus.

When using any of these projects, ensure you keep the quality of the project and their impacts on your production Nexus instance in mind.

## 26.3 Nexus Plugins

Nexus plugins expand functionality of Nexus itself in various aspects on the user interface and underlying features:

### Nexus Open Source Plugins <https://github.com/sonatype/nexus-oss/tree/master/plugins>

Large number of plugins bundled with Nexus Open Source including YUM support, P2 support and others.

### Example Plugins <https://github.com/sonatype/nexus-example-plugins>

Example plugins from Sonatype.

### APT Plugin <https://github.com/Tangresh/nexus-apt-plugin>

APT/DEB repository support for Nexus.

### Rundeck Plugin <https://github.com/rundeck/nexus-rundeck-plugin>

Nexus integration with Rundeck

### Webhook Plugin <https://github.com/vbehar/nexus-webhook-plugin>

Support for webhook notifications for component deployments to Nexus.

---

**Artifact Usage Plugin** <https://github.com/saleemshafi/nexus-artifact-usage-plugin>

Plugin to display components depending on a specific component.

**Dependency Mgt. Plugin** <https://github.com/Terracotta-OSS/nexus-dependency-management-plugin>

Plugin to display the dependency tree of a component with further detailed information.

**GroupId Mgt. Plugin** <https://github.com/UW-Madison-DoIT/nexus-groupid-management-plugin>

Plugin to help with provisioning security per groupId.

**Repository Cleanup Plugin** <https://github.com/Vlatombe/nexus-repository-cleanup-plugin/>

Scheduled task that can remove components based on age and a regular expression pattern.

**Gitlab Token Auth Plugin** <https://github.com/jdamick/nexus-gitlab-token-auth-plugin>

Nexus authentication support using [Gitlab](#) user token.

**AWS S3 Publish Plugin** <https://github.com/carrot-garden/carrot-nexus>

Nexus plugin to publish components deployed to Nexus also to AWS S3.

**Hipchat for Nexus Plugin** <https://bitbucket.org/tpettersen/hipchat-for-nexus>

Supports notifications in HipChat when components matching a pattern are deployed to Nexus.

## 26.4 Nexus Integrations

**Nexus Maven Plugins** <https://github.com/sonatype/nexus-maven-plugins>

The official Nexus Staging Maven Plugin and the Nexus M2Settings Maven Plugin from Sonatype.

The plugins are using the Nexus REST API client library and can be used as example for your own Maven plugins or other Java based clients.

**Nexus Ant Tasks** <https://github.com/sonatype/nexus-ant-tasks>

The official Nexus Staging Ant Tasks from Sonatype.

**Puppet Module for Nexus** <https://github.com/hubspotdevops/puppet-nexus>

Puppet module to install and configure Nexus, authored by HubSpot

**Puppet Module for Nexus** [https://forge.puppetlabs.com/atlassian/nexus\\_rest](https://forge.puppetlabs.com/atlassian/nexus_rest)

Another Puppet module to manage Nexus, authored by Atlassian

**Nexus Cookbook** <https://github.com/RiotGames/nexus-cookbook>

Chef cookbook to install and configure Nexus.

**Openshift Nexus** <https://github.com/hongun/openshift-nexus>

Scripts to provision Nexus on [OpenShift](#).

---

**Nexus CLI** [https://github.com/RiotGames/nexus\\_cli](https://github.com/RiotGames/nexus_cli)

Set of command line programs to interact with Nexus.

**Nexus RPM Package** <https://github.com/jbraeuer/nexus-oss-rpms>

Nexus as RPM package.

**Nexus DEB Package** <https://github.com/tobrien/nexus-oss-deb>

Nexus as DEB package.

**Puppet Nexus Client** <https://github.com/cescoffier/puppet-nexus>

Puppet module to retrieve components from Nexus.

**Gradle Plugin** <https://github.com/bmuschko/gradle-nexus-plugin>

Gradle plugin to deploy components to Nexus and via OSSRH to the Central Repository.

**Gradle Staging Plugin** <https://github.com/adaptivecomputing/plugins-gradle/tree/master/nexus-workflow>

Gradle plugin to deploy components to Nexus and via OSSRH to the Central Repository with good support for staging automation.

**SBT Plugin** <https://github.com/xerial/sbt-sonatype>

Gradle plugin to deploy components to Nexus and via OSSRH to the Central Repository.

**List Versions Jenkins Plugin** <https://github.com/USGS-CIDA/list-nexus-versions-plugin>

Jenkins plugin to display component versions available in Nexus.

**Nexus Metadata Jenkins Plugin** <https://github.com/marcelbirkner/nexus-metadata-plugin>

Jenkins plugin to add custom metadata with deployments to Nexus Professional.

**Artifact Promotion Jenkins Plugin** <https://github.com/jenkinsci/artifact-promotion-plugin>

Jenkins plugin allowing you to promote components to different repositories in Nexus OSS

**Go Maven Poller** <https://github.com/ThoughtWorksInc/go-maven-poller>

Package material plugin for Go that can poll Nexus for components.

**Nexus Docker Image** <https://registry.hub.docker.com/u/conceptnotfound/sonatype-nexus/>

simple Docker image with Nexus

**Nexus NPM Docker Image** <https://github.com/marcellodesales/nexus-npm-registry-docker-image>

Docker Image of Nexus with NPM support preconfigured

## 26.5 Other Community Projects

**Nexus Performance Testing Library** <https://github.com/sonatype/nexus-perf>

Regression and stress test library for Nexus from Sonatype.

---

**Repository Management With Nexus** <https://github.com/sonatype/nexus-book>

The source code for the book, which is the official documentation for Nexus Open Source and Nexus Professional.

**Nexus Book Examples** <https://github.com/sonatype/nexus-book-examples>

Examples for the Nexus trial guide chapter of the book *Repository Management with Nexus*.

**Nexus Introduction** <https://github.com/sonatype/nexus-introduction-presentation>

Slides and examples to present about Sonatype Nexus at user groups or in similar settings.

## 26.6 Contributing

All of the projects listed in Section 26.2 are community efforts and open to your participation. If you are aware of any other projects or would like to have your project listed here, please contact us at [books@sonatype.com](mailto:books@sonatype.com).

## Appendix A

# Contributing to the Nexus Book

The Nexus Book is an open source project in which you can participate, if you have an idea for documentation. Sonatype's books include open writing efforts, and we see the value of the documentation contributions the same as code contributions. If you are interested in our technology and would like to contribute, please review the basics described in Appendix A.

### Contributor License Agreement (CLA)

In order to contribute to the Nexus book, you will first need to fill out a contributor license agreement. This is a legal agreement between you and Sonatype that ensures that your contributions are not covered by any other legal requirements. Sonatype requires contributors to sign this agreement for all major contributions larger than a single section. If your contribution consists of finding and fixing simple typos or suggesting minor changes to the wording or sequence of a particular section, you can contribute these changes via the Sonatype support site or directly as a pull request on the github project. If your contribution involves direct contribution of a number of sections or chapters you will first need to sign our Contributor License Agreement (CLA).

To download the CLA from the following URL: <http://www.sonatype.org/SonatypeCLA.pdf>

Once you have completed and signed this document, you can email the scan to [books@sonatype.com](mailto:books@sonatype.com).

**How to Contribute** The source code for the book is hosted on GitHub in the [nexus-book](#) project. Instructions on tools used to author content as well as building the book and more can be found there.

---

## Appendix B

# Copyright

Copyright © 2011-2014 Sonatype, Inc. All rights reserved.

Online version published by Sonatype, Inc.

Nexus™, Nexus Professional™, and all Nexus-related logos are trademarks or registered trademarks of Sonatype, Inc. in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, Inc. in the United States and other countries.

IBM® and WebSphere® are trademarks or registered trademarks of International Business Machines, Inc. in the United States and other countries.

Eclipse™ is a trademark of the Eclipse Foundation, Inc. in the United States and other countries.

Apache and the Apache feather logo are trademarks of The Apache Software Foundation.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Sonatype, Inc. was aware of a trademark

---

claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

## Appendix C

# Creative Commons License

This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 United States license. For more information about this license, see <http://creativecommons.org/licenses/by-nc-nd/3.0/us/>. You are free to share, copy, distribute, display, and perform the work under the following conditions:

- You must attribute the work to Sonatype, Inc. with a link to <http://www.sonatype.com>.
- You may not use this work for commercial purposes.
- You may not alter, transform, or build upon this work.

If you redistribute this work on a web page, you must include the following link with the URL in the *about attribute* listed on a single line (remove the backslashes and join all URL parameters):

```
<div xmlns:cc="http://creativecommons.org/ns#"  
about="http://creativecommons.org/license/results-one?q_1=2&q_1=1\"  
&field_commercial=n&field_derivatives=n&field_jurisdiction=us\"  
&field_format=StillImage&field_worktitle=Repository%3A+\Management\"  
&field_attribute_to_name=Sonatype%2C+Inc.\\"  
&field_attribute_to_url=http%3A%2F%2Fwww.sonatype.com\"  
&field_sourceurl=http%3A%2F%2Fwww.sonatype.com%2Fbook\"  
&lang=en_US&language=en_US&n_questions=3">  
<a rel="cc:attributionURL" property="cc:attributionName"  
href="http://www.sonatype.com">Sonatype, Inc.</a> /  
<a rel="license"
```

```
href="http://creativecommons.org/licenses/by-nc-nd/3.0/us/"> >  
CC BY-NC-ND 3.0</a>  
</div>
```

When downloaded or distributed in a jurisdiction other than the United States of America, this work shall be covered by the appropriate ported version of Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 license for the specific jurisdiction. If the Creative Commons Attribution-Noncommercial-No Derivative Works version 3.0 license is not available for a specific jurisdiction, this work shall be covered under the Creative Commons Attribution-Noncommercial-No Derivative Works version 2.5 license for the jurisdiction in which the work was downloaded or distributed. A comprehensive list of jurisdictions for which a Creative Commons license is available can be found on the Creative Commons International web site at <http://creativecommons.org/international>.

If no ported version of the Creative Commons license exists for a particular jurisdiction, this work shall be covered by the generic, unported Creative Commons Attribution-Noncommercial-No Derivative Works version 3.0 license available from <http://creativecommons.org/licenses/by-nc-nd/3.0/>.

## C.1 Creative Commons BY-NC-ND 3.0 US License

### Creative Commons Attribution-NonCommercial-NoDerivs 3.0 United States

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

#### 1. Definitions

- a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with one or more other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

- b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
  - c. "Licensor" means the individual, individuals, entity or entities that offers the Work under the terms of this License.
  - d. "Original Author" means the individual, individuals, entity or entities who created the Work.
  - e. "Work" means the copyrightable work of authorship offered under the terms of this License.
  - f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.
  3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:
    - a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works; and,
    - b. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(d) and 4(e).

1. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
  - a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource

Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of a recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. When You distribute, publicly display, publicly perform, or publicly digitally perform the Work, You may not impose any technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by Section 4(c), as requested.

- b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
- c. If You distribute, publicly display, publicly perform, or publicly digitally perform the Work (as defined in Section 1 above) or Collective Works (as defined in Section 1 above), You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear, if a credit for all contributing authors of the Collective Work appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this clause for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

## 2. Representations, Warranties and Disclaimer

---

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR

OFFERS THE WORK AS-IS AND ONLY TO THE EXTENT OF ANY RIGHTS HELD IN THE LICENSED WORK BY THE LICENSOR. THE LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MARKETABILITY, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

1. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
2. Termination
  - a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works (as defined in Section 1 above) from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
  - b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.
3. Miscellaneous
  - a. Each time You distribute or publicly digitally perform the Work (as defined in Section 1 above) or a Collective Work (as defined in Section 1 above), the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
  - b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
  - c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

- d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

## C.2 Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at <http://creativecommons.org/>.