

Repository Management with Nexus

Ed. 4.0

Contents

1	Concepts	1
1.1	Introduction	1
1.2	The Basics - Components, Repositories and Repository Formats	1
1.3	An Example - Maven Repository Format	4
1.4	Repository Management	7
1.5	Software Supply Chain Automation	9
2	Installation and Running	11
2.1	Introduction	11
2.2	Downloading	11
2.3	Java Runtime Environment	12
2.4	Installing and Running with the Distribution Archive	13
2.5	Installing with Docker	14

2.6	Upgrading	15
2.7	Configuring as a Service	15
2.7.1	Setting up as a Service on Linux	16
2.7.2	Running as a Service on Windows	18
2.7.3	Running as a Service on Mac OS X	18
2.8	Running Behind a Reverse Proxy	19
2.8.1	Example: Reverse Proxy on Restricted Ports	20
2.8.2	Example: Reverse Proxy Virtual Host at Custom Context Path	22
2.8.3	Example: Reverse Proxy SSL Termination at Base Path	23
2.9	Accessing the User Interface	25
2.10	Directories	26
2.10.1	Installation Directory	26
2.10.2	Data Directory	27
2.11	Configuring the Runtime Environment	27
2.11.1	Updating Memory Allocation and other JVM Paramaters	28
2.11.2	Changing the HTTP Port	28
2.11.3	Changing the Context Path	29
2.11.4	Configuring the Data Directory Location	29

2.12 Uninstalling	30
3 Using the User Interface	31
3.1 Introduction	31
3.2 User Interface Overview	31
3.3 Searching for Components	35
3.3.1 Search Criteria and Component Attributes	36
3.3.2 Search Results	39
3.3.3 Preconfigured Searches	39
3.3.4 Example Use Case - SHA-1 Search	40
3.4 Browsing Repositories and Repository Groups	41
3.5 Viewing Component Information	42
3.6 Viewing Asset Information	43
3.7 Working with Your User Profile	45
3.7.1 Changing Your Password	46
4 Configuration	47
4.1 Introduction	47
4.2 System Configuration	48
4.2.1 Bundles	48

4.2.2	Accessing and Configuring Capabilities	49
4.2.3	Email Server	51
4.2.4	Base URL Creation	53
4.2.5	HTTP and HTTPS Request and Proxy Settings	54
4.2.6	Configuring and Executing Tasks	57
4.3	Repository Management	61
4.3.1	Blob Stores	61
4.3.2	Proxy Repository	63
4.3.3	Hosted Repository	64
4.3.4	Repository Group	65
4.3.5	Managing Repositories and Repository Groups	65
4.3.6	Repository Management Example	74
4.4	Support Features	76
4.4.1	Analytics	77
4.4.2	Logging and Log Viewer	79
4.4.3	Metrics	81
4.4.4	Support ZIP	82
4.4.5	System Information	83

5	Security	84
5.1	Introduction	84
5.2	Realms	86
5.3	Privileges	87
5.4	Roles	91
5.5	Users	95
5.6	Anonymous Access	97
5.7	LDAP	98
5.7.1	Introduction	98
5.7.2	Enabling the LDAP Authentication Realm	99
5.7.3	LDAP Connection and Authentication	99
5.7.4	User and Group Mapping	102
5.8	Authentication via Remote User Token	107
5.9	Configuring SSL	108
5.9.1	Outbound SSL - Trusting SSL Certificates of Remote Repositories	108
5.9.2	Outbound SSL - Trusting SSL Certificates Globally	110
5.9.3	Outbound SSL - Trusting SSL Certificates Using Keytool	112
5.9.4	Inbound SSL - Configuring to Serve Content via HTTPS	114

6	Maven Repositories with Apache Maven and Other Tools	116
6.1	Introduction	116
6.2	Maven Repository Format	117
6.3	Proxying Maven Repositories	118
6.4	Hosting Maven Repositories	119
6.5	Grouping Maven Repositories	119
6.6	Browsing and Searching Maven Repositories	120
6.7	Configuring Apache Maven	120
6.8	Configuring Apache Ant and Apache Ivy	122
6.9	Configuring Apache Ant and Eclipse Aether	123
6.10	Configuring Gradle	124
6.11	SBT	125
6.12	Leiningen	126
7	.NET Package Repositories with NuGet	128
7.1	Introduction	128
7.2	NuGet Repository Format	129
7.3	NuGet Proxy Repositories	129
7.4	NuGet Hosted Repositories	131

7.5	NuGet Repository Groups	132
7.6	Accessing Packages in Repositories and Groups	132
7.7	Deploying Packages to NuGet Hosted Repositories	133
7.7.1	Accessing your NuGet API Key	133
7.7.2	Creating a Package for Deployment	134
7.7.3	Command line based Deployment to a NuGet Hosted Repository	134
7.8	Integration with Visual Studio	134
8	Private Registry for Docker	136
8.1	Introduction	136
8.2	SSL and Repository Connector Configuration	137
8.2.1	Tips for SSL Certificate Usage	138
8.3	Support for Docker Registry API	139
8.4	Proxy Repository for Docker	139
8.5	Hosted Repository for Docker (Private Registry for Docker)	141
8.6	Repository Groups for Docker	141
8.7	Authentication	142
8.8	Accessing Repositories	143
8.9	Searching	144

8.10 Pulling Images	145
8.11 Pushing Images	145
9 Node Packaged Modules and npm Registries	147
9.1 Introduction	147
9.2 Proxying npm Registries	148
9.3 Private npm Registries	148
9.4 Grouping npm Registries	149
9.5 Browsing npm Registries and Searching Modules	150
9.6 Configuring npm	150
9.7 npm Security	151
9.8 Publishing npm Packages	151
9.9 Deprecating npm Packages	152
10 Bower Repositories	154
10.1 Introduction	154
10.2 Proxying Bower Repositories	155
10.3 Hosting Bower Repositories	155
10.4 Bower Repository Groups	156
10.5 Installing Bower	156

10.6	Configuring Bower Package Download	157
10.7	Browsing Bower Repositories and Searching Packages	158
10.8	Registering Bower Packages	159
11	PyPI Repositories	160
11.1	Introduction	160
11.2	Proxying PyPI Repositories	161
11.3	Hosting PyPI Repositories	161
11.4	PyPI Repository Groups	162
11.5	Installing PyPI Client Tools	162
11.6	Configuring PyPI Client Tools	163
11.7	SSL Usage for PyPI Repositories	164
11.8	Browsing PyPI Repositories and Searching Packages	165
11.9	Uploading PyPI Packages	165
12	Ruby, RubyGems and Gem Repositories	166
12.1	Introduction	166
12.2	Proxying Gem Repositories	167
12.3	Private Hosted Gem Repositories	168
12.4	Grouping Gem Repositories	169

12.5 Using Gem Repositories	169
12.6 Pushing Gems	171
13 Raw Repositories, Maven Sites and More	173
13.1 Introduction	173
13.2 Creating a Hosted Raw Repository	173
13.3 Creating and Deploying a Maven Site	174
13.3.1 Creating a New Maven Project	174
13.3.2 Configuring Maven for Site Deployment	174
13.3.3 Adding Credentials to Your Maven Settings	176
13.3.4 Publishing a Maven Site	176
13.4 Proxying and Grouping Raw Repositories	178
13.5 Uploading Files to Hosted Raw Repositories	178
14 REST and Integration API	180
14.1 Introduction	180
14.2 Writing Scripts	180
14.3 Managing and Running Scripts	182
14.4 Examples	184

15 Bundle Development	188
15.1 Introduction	188
15.2 Installing Bundles	189
15.3 Bundle Development Overview	190
15.4 Support for a New Repository Format	191
15.4.1 Format, Recipe and Facet	193
15.4.2 Storage	193
15.4.3 User Interface	194
15.4.4 Tasks	194
15.5 Contributing Bundles	194
A Contributing	196
B Copyright	198
C Creative Commons License	200
C.1 Creative Commons BY-NC-ND 3.0 US License	201
C.2 Creative Commons Notice	205

Preface

Introduction

This book covers the concepts of repository management, software supply chain management and component management in general and specifically the usage of Nexus Repository Manager Pro and Nexus Repository Manager OSS, in general also referred to as Nexus Repository Manager. It details all aspects of set-up and running the repository manager. Specifically this documentation covers version 3.0.2.

This documentation was last updated and published on 2017-08-07.

How to Use this Documentation

Formats This Nexus Repository Manager documentation is available online in HTML format, so you can read it in web browser, while using the repository manager or referring other users to specific content.

The navigation bar on the left-hand side of the documentation allows you to access documentation for different versions of Nexus Repository Manager as well as a number of other resources.

The top right-hand side of the documentation features a search input box that accesses all content to provide you with the relevant information for your search term.

In addition to this online version the documentation can be downloaded in Portable Document Format (PDF) and Electronic Publication (EPUB) format for offline access.

Nexus Repository Manager Editions The documentation covers all editions of the repository manager - Nexus Repository Manager OSS and Nexus Repository Manager Pro. Each chapter or smaller section title in the documentation is followed by a text, showing in which edition the specific features are available e.g.:

Available in Nexus Repository OSS and Nexus Repository Pro

If you can not find any specific mentions of a specific edition, it is safe to assume that the feature can be found in all editions. Keep in mind that Nexus Repository Manager Pro is an extension of Nexus Repository Manager OSS.

Conventions Used in the Documentation A number of conventions are used through out the documentation to simplify following the instructions and content.

A user interface label or button is highlighted. E.g. use the *Applications* button to access the list of applications.

A code segment or command line like `mvn clean install` input in a paragraph uses monospace fonts just like they would be used in a command line window.

Larger code segments use the same monospace fonts and are encapsulated in a block:

```
$ mvn --version
Apache Maven 3.3.3 (79941207757...; 2015-04-22T06:57:37-05:00)
Maven home: /opt/apache-maven-3.3.3
Java version: 1.8.0_60, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home ↩
/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.10.5", arch: "x86_64", family: "mac"
```

The documentation uses specific blocks for notes, tips and important messages:

Note

You can download Java from the Oracle website.

Tip

The repository manager should be set up to run as a service.

**Important**

Nexus Repository Manager requires Java 8.

In addition there are blocks for warnings and cautioning alerts:

**Warning**

Mounting the storage directory via NFS can have negative performance and stability effects.

**Caution**

Be sure to perform a complete backup before upgrading the repository manager.

Application screenshots and other images are typically included as numbered figures and referenced from the flowing text.

Next Steps With all this in mind you can learn more about the [concepts](#) used or learn more about [installing](#), [configuring](#) or [using](#) the repository manager or choose to read a section about your specific use case or question.

Chapter 1

Concepts

1.1 Introduction

Using the Nexus Repository Manager as well as the tools for software supply chain automation using the Nexus IQ Server and associated tools of the Nexus platform requires an understanding of a few concepts and terms like *Component*, *Repository*, *Repository Format* and others. This chapter provides you with all the necessary background and knowledge as well as an idea of a progression in your usage of the tools from the Nexus platform.

1.2 The Basics - Components, Repositories and Repository Formats

The Nexus platform with Nexus Repository Manager Pro, Nexus Repository Manager OSS and Nexus IQ Server is all about working with components and repositories.

So what are components? A component is a resource like a library or a framework that is used as part of your software application at run-time, integration or unit test execution time or required as part of your build process. It could be an entire application or a static resource like an image.

Typically these components are archives of a large variety of files including

- Java byte code in class files
- C object files
- text files e.g. properties files, XML files, JavaScript code, HTML, CSS
- binary files such as images, PDF files, sound and music files
- and many others

The archives are using numerous formats such as

- Java JAR, WAR, EAR formats
- plain ZIP or .tar.gz files
- Other package formats such as NuGet packages, Ruby gems, NPM packages
- Executable formats such as .exe or .sh files, Android APK files, various installer formats, ...

Components can be composed of multiple, nested components themselves. E.g., consider a Java web application packaged as a WAR component. It contains a number of JAR components and a number of JavaScript libraries. All of these are standalone components in other contexts and happen to be included as part of the WAR component.

Components provide all the building blocks and features that allow a development team to create powerful applications by assembling them and adding their own business related components to create a full-fledged, powerful application.

In different tool-chains components are called *artifact*, *package*, *bundle*, *archive* and other terms. The concept and idea remains the same and *component* is used as the independent, generic term.

Components are identified by a set of specific values - the *coordinates*. A generic set of these coordinates is the usage of *group*, *name* and *version*. The names and the usage of these coordinates changes with the tool-chains used. Components can also be the anchor for further metadata.

Assets Assets are the material addition to all this metadata. The actual archive file is an asset associated with the component. Many formats have a one-to-one mapping for component to asset.

More complex formats however have numerous assets associated with a component. For example a typical JAR component in a Maven repository is defined at least by the POM and the JAR files - both of which constitute separate assets belonging to the same components. Additional files such as JavaDoc or Sources JAR files are assets that belong to the same component.

The Docker format, on the other hand, gives assets unique identifiers and calls them Docker layers. These assets can be reused for different components - the Docker images. A Docker layer, for example, could be a specific operating system referenced by multiple Docker images.

Components in Repositories A wide variety of components exists and more are continuously created by the open source community as well as proprietary vendors. There are libraries and frameworks written in various languages on different platforms that are used for application development every day. It has become a default pattern to build applications by combining the features of multiple components with your own custom components containing your application code to create an application for a specific domain.

In order to ease the consumption and usage of components, they are aggregated into collections of components. These are called a *repository* and are typically available on the internet as a service. On different platforms terms such as *registry* and others are used for the same concept.

Example for such repositories are

- the Central Repository, also known as Maven Central
- the NuGet Gallery
- RubyGems.org
- npmjs.org

and a number of others. Components in these repositories are accessed by numerous tools including

- package managers like npm, nuget or gem,
- build tools such as Maven, Gradle, rake, grunt...
- IDE's such as Eclipse, IntelliJ...

and many, many others.

Repositories have Formats The different repositories use different technologies to store and expose the components in them to client tools. This defines a *repository format* and as such is closely related to the tools interacting with the repository.

E.g. the Maven repository format relies on a specific directory structure defined by the identifiers of the components and a number of XML formatted files for metadata. Component interaction is performed via plain HTTP commands and some additional custom interaction with the XML files.

Other repository formats use databases for storage and REST API interactions, or different directory structures with format specific files for the metadata.

1.3 An Example - Maven Repository Format

Maven developers are familiar with the concept of a repository, since repositories are used by default. The primary type of a binary component in a Maven format repository is a JAR file containing Java byte-code. This is due to the Java background of Maven and the fact that the default component type is a JAR. Practically however, there is no limit to what type of component can be stored in a Maven repository. For example, you can easily deploy WAR or EAR files, source archives, Flash libraries and applications, Android archives or applications or Ruby libraries to a Maven repository.

Every software component 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 components, which a given component depends upon for successful compilation or execution.

When Maven downloads a component like a dependency or a plugin from a repository, it also downloads that component's POM. Given a component's POM, Maven can then download any other components that are required by that component.

Maven and other tools, such as Ivy or Gradle, which interact with a Maven repository to search for binary software components, model the projects they manage and retrieve software components on-demand from a repository.

The Central Repository When you download and install Maven without any customization, it retrieves components from the Central Repository. It serves millions of Maven users every single day. It is the default, built-in repository using the Maven repository format and is managed by Sonatype. Statistics about the size of the Central Repository are available at <http://search.maven.org/#stats>.

The Central Repository is the largest repository for Java-based components. It can be easily used from

other build tools as well. 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:

Component Metadata

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

Release Stability

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

Component Security

The Central Repository contains cryptographic hashes and PGP signatures that can be used to verify the authenticity and integrity of software components served and supports connections in a secure manner via HTTPS.

Performance

The Central Repository is exposed to the users globally via a high performance content delivery network of servers.

In addition to the Central Repository, there are a number of major organizations, such as Red Hat, Oracle or the Apache Software foundation, which maintain separate, additional repositories. Best practice to facilitate these available repositories is to install Nexus Repository Manager OSS or Nexus Repository Manager Pro and use it to proxy and cache the contents on your own network.

Component Coordinates and the Repository Format Component coordinates create a unique identifier for a component. Maven coordinates use the following values: *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 components 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 *artifactId* is an identifier for a software component and should be a descriptive name. The combination of *groupId* and *artifactId* must be unique for a specific project.

version

The version of a project ideally follows the established convention of **semantic versioning**. For example, if your simple-library component 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 component it is managing. Packaging can be anything that describes any binary software format including *zip*, *nar*, *war*, *ear*, *sar*, *aar* and others.

Tools designed to interact Maven repositories translate component 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 also downloads 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 components, which are then retrieved from the same repository using the same URL patterns.

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

While it is possible to create a repository which serves both release and snapshot components, repositories are usually segmented into release or snapshot repositories serving different consumers and maintaining different standards and procedures for deploying components. 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 components can be deployed and changed frequently without regard for stability and repeatability concerns.

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

Release

A release component is a component which was created by a specific, versioned release. For example, consider the 1.2.0 release of the `commons-lang` library stored in the Central Repository. This release component, `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 Central Repository. Released components 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 component is associated with a PGP signature, an MD5 and SHA check-sum which can be used to verify both the authenticity and integrity of the binary software component.

Snapshot

Snapshot components are components generated during the development of a software project. A Snapshot component has both a version number such as 1.3.0 or 1.3 and a time-stamp in its name. For example, a snapshot component 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 components from a repository also know how to interrogate the metadata associated with a Snapshot component to retrieve the latest version of a Snapshot dependency from a repository.

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

Looking at the Maven repository format and associated concepts and ideas allowed you grasp some of the details and intricacies involved with different tools and repository formats, that will help you appreciate the need for [repository management](#).

1.4 Repository Management

The proliferation of different repository formats and tools accessing them as well as the emergence of more publicly available repositories has triggered the need to manage access and usage of these repositories and the components they contain.

In addition, hosting your own private repositories for internal components has proven to be a very efficient methodology to exchange components during all phases of the software development life cycle. It is considered a best practice at this stage.

The task of managing all the repositories your development teams interact with can be supported by the use of a dedicated server application - a repository manager.

Put simply, a repository manager provides two core features:

- the ability to proxy a remote repository and cache components saving both bandwidth and time required to retrieve a software component from a remote repository repeatedly, and
- the ability to host a repository providing an organization with a deployment target for internal software components.

Just as Source Code Management (SCM) tools are designed to manage source code, repository managers have been designed to manage and track external dependencies and components generated by your build.

Repository managers 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, by facilitating the exchange and usage of binary components.

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 a public 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 the Central Repository into your organization. You can use it to proxy the Central Repositories and other repositories, and host your own repositories for internal and external use.

Capabilities of a Repository Manager In addition to these two core features, a repository manager can support the following use cases:

- allows you to manage binary software components through the software development life-cycle,
 - search and catalogue software components,
 - control component releases with rules and add automated notifications
 - integrate with external security systems, such as LDAP or Atlassian Crowd
 - manage component metadata
 - host external components, not available in external repositories
 - control access to components and repositories
 - display component dependencies
-

- browse component archive contents

Advantages of Using a Repository Manager Using a repository manager provides a number of benefits including:

- improved software build performance due to faster component download off the local repository manager
- reduced bandwidth usage due to component caching
- higher predictability and scalability due to limited dependency on external repositories
- increased understanding of component usage due to centralized storage of all used components
- simplified developer configuration due to central access configuration to remote repositories and components on the repository manager
- unified method to provide components to consumers reducing complexity overheads
- improved collaboration due the simplified exchange of binary components

1.5 Software Supply Chain Automation

Once you adopting a repository manager as a central point of of storage and exchange for all component usage, the next step is expand its use in your efforts to automate and manage the software supply chain throughout your software development life-cycle.

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 work-flow system just to create a website with a publishing work-flow would be extremely inefficient.

Development starts with the selection of suitable components for your projects based on comprehensive information about the components and their characteristics e.g., in terms of licenses used or known security vulnerabilities available in Nexus Repository Manager Pro. Besides focusing on being a 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.

Software supply chain automation progresses through your daily development efforts, your continuous integration builds and your release processes all the way to your applications deployed in production environments at your clients or your own infrastructure.

Nexus IQ Server provides a number of tools to improve your component usage in your software supply chain allowing you to automate your processes to ensure high quality output, while increasing your development speed towards continuous deployment procedures. These include:

- integration with common development environments like the Eclipse IDE
- plugins for continuous integration servers such as Jenkins, Hudson or Eclipse
- visualizations in quality assurance tools like SonarQube
- command line tools for custom integrations
- notifications to monitor component flows

Nexus IQ Server enables you to ensure the integrity of the modern software supply chain, amplifying the benefits of modern development facilitating component usage, while reducing associated risks.

Chapter 2

Installation and Running

2.1 Introduction

Nexus Repository Manager is a Java application that requires a Java Runtime Environment. When you run Nexus Repository Manager, you are running a server application with a web-based user interface. The application itself runs with the Eclipse Jetty servlet container and Apache Karaf OSGi-container.

Installation is a simple process. This chapter provides further details to get started and keep the repository manager running successfully in production deployments.

2.2 Downloading

Nexus Repository Manager can be downloaded from [Sonatype](#). Distributions are available for the 64-bit versions for Apple OSX, Microsoft Windows and Unix/Linux. They contain all necessary resources to install and run the repository manager. You can download the plain archive file for your operating system of a specific release version.

The plain archive files are Gzip TAR (TGZ) or ZIP files and are suitable for installation without a graphical user interface purely using command line-based interaction. The file names include operating system qualifiers and are similar to similar to:

```
nexus-3.0.2-02-mac.tgz
nexus-3.0.2-02-unix.tar.gz
nexus-3.0.2-02-win64.zip
```

Next steps, after a successful download, depend on the operating system you are using and are documented in Section 2.4.

2.3 Java Runtime Environment

Nexus Repository Manager requires a Java 8 Runtime Environment (JRE) from Oracle. The distributions for OSX and Windows include suitable runtime environments for the specific operating system. The distributions for Unix do not include the runtime environment.

If you prefer to use an external runtime or use a Unix operating system, it is recommended to use the latest version of Java 8 available from the [Oracle website](#). You can choose to install the full JDK or the JRE only.

You can confirm the installed Java version with the `java` command:

```
$ java -version
java version "1.8.0_60"
Java(TM) SE Runtime Environment (build 1.8.0_60-b27)
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
```

When multiple JDK or JRE versions are installed, you need to ensure the correct version is configured by running the above command as the operating system user, that is used to run the repository manager. This allows you to have a dedicated runtime environment for the repository manager installed that is not on the path and not used by other installed applications. It allows you to separate upgrades of the Java runtime used by the repository manager from upgrades of the runtime used by other applications.

Tip

OpenJDK or other Java distributions or older Java versions are not supported.

Potentially you need to update the configuration to specify a specific JDK or JRE installation path:

To set the path for a specific Java location open the `bin/nexus` script and locate the line `INSTALL4J_JAVA_HOME_OVERRIDE`. Remove the hash and specify the location of your JDK/JRE:

```
INSTALL4J_JAVA_HOME_OVERRIDE=/usr/lib/jvm/java-8-oracle
```

The startup script verifies the runtime environment by checking for the existence of the nested `bin/java` command as well as major and minor version of the runtime to be the required `1.8`. If the configured runtime is not suitable, it will proceed with a best effort to locate a suitable runtime configured on the path or via the `JAVA_HOME` environment variable. If successful, it will start up the repository manager with this JVM.

2.4 Installing and Running with the Distribution Archive

The distribution archives combine the application and all required resources in an archive file. Installing and running Nexus Repository Manager is straightforward. Simply unpack the archive in a directory, to which you have full access.

If you are installing the repository manager on a local workstation to give it a test run, you can install it in your home directory or wherever you like. Nexus Repository Manager doesn't have any hard-coded directories and will run from any directory.

You can extract the archive ZIP for Windows with any archiving tool like **7zip** or on the command line with e.g.

```
$ 7za.exe e nexus-3.0.2-02-win64.zip
```

On Windows you should install the repository manager outside Program Files to avoid problems with Windows file registry virtualization. If you plan to run the repository manager as a specific user you can install it into the `AppData\Local` directory of that users home directory. Otherwise simply use e.g., `C:\nexus` or something similar, ensuring that the user running the application has full access.

On OSX or Linux the downloaded GZip'd TAR archive can be extracted with:

```
$ tar xvzf nexus-3.0.2-02-mac.tgz
$ tar xvzf nexus-3.0.2-02-unix.tar.gz
```

You install the repository manager in a directory other than your users home directory. On a Unix machine common practice is to use `/opt`.

The extraction process creates a directory with a number of directories inside. This directory is referred to as the *installation directory*. It contains the application and related resources as well as the *data directory* for component and repository storage. Further details about these folders can be found in [Section 2.10](#)

The `bin` folder contains the generic startup scripts for Unix-like platforms called `nexus`. The Windows platform equivalent is called `nexus.exe`. To start the repository manager from the `bin` folder on a Unix-like platform like Linux use

```
./nexus run
```

The equivalent invocation on Windows requires a `/` in front of the `run` and any other commands.

```
nexus.exe /run
```

Starting the repository manager with the `run` command will leave it running in the current shell and display the log output.

The repository manager is fully started once you see a message similar to the following in the log:

```
Started Nexus Repository Manager 3.0.2-02
```

In order to shut down the repository manager running via the `run` command, you have to press `CTRL-C`.

The `nexus` script can be used to manage the repository manager as a background application on OSX and Unix with the `start`, `stop`, `restart`, `force-reload` and `status` commands. The Windows `nexus.exe` command supports similar commands with a prefix of `/` e.g., `/run`.

Once the repository manager is started you can access the user interface as details in [Section 2.9](#).

2.5 Installing with Docker

Available in Nexus Repository OSS and Nexus Repository Pro

Docker automates the deployment of applications inside virtualized Linux containers. You can create a container that supports the installation of Nexus Repository Manager Pro and Nexus Repository Manager OSS.

To install the repository manager with a Docker image, follow the steps at [Docker Hub](#).

2.6 Upgrading

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

To keep the upgrade simple schedule downtime to preserve important directories during the process. Follow the steps in the support [knowledge base article](#).

Note

Upgrading to Nexus Repository Manager OSS 3.0.0 can only be performed by users who run the milestone 7 release of the repository manager. Be sure to manually back up the milestone 7 data directory to another location. It is a crucial step to properly upgrade the application.

2.7 Configuring as a Service

Available in Nexus Repository OSS and Nexus Repository Pro

When installing Nexus Repository Manager for production usage it has to be configured it to run as a service, so it restarts after the server reboots. It is good practice to run that service or daemon as a specific user that has only the required access rights.

Installation from the [distribution archive](#) does not include the configuration of a service. The following sections provide instructions for configuring the service manually. Independent of the operating system the steps are

- Create operating sytem user with limited access rights dedicated to run the repository manager as a service
 - Ensure suitable Java runtime environment is installed - see Section [2.3](#)
 - Configure the service and ensure it starts as part of the operating sytem boot process
-

**Warning**

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

2.7.1 Setting up as a Service on Linux

You can configure the repository manager to run as a service with `init.d` or `systemd`. Both are startup frameworks used in Linux-based systems such as Ubuntu and CentOS. They are, essentially, initscripts that load commands to manage the repository manager daemon.

Before running the service configure an absolute path for your repository manager files. Then create a `nexus` user with sufficient access rights to run the service.

Change `NEXUS_HOME` to the absolute folder location in your `.bashrc` file, then save.

```
NEXUS_HOME="/opt/nexus"
```

In `bin/nexus.rc` assign the user between the quotes in the line below.

```
run_as_user="nexus"
```

If you use `init.d` instead of `systemd`, symlink `$NEXUS_HOME/bin/nexus` to `/etc/init.d/nexus`:

```
sudo ln -s $NEXUS_HOME/bin/nexus /etc/init.d/nexus
```

2.7.1.1 Running the Service

chkconfig.

This example uses `chkconfig`, a tool that targets the initscripts in `init.d` to run the `nexus` service. Run these commands to activate the service:

```
cd /etc/init.d
sudo chkconfig --add nexus
sudo chkconfig --levels 345 nexus on
sudo service nexus start
```

The second command adds nexus as a service to be started and stopped with the 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. Then the service command starts the repository manager.

update-rc.d.

This example uses `update-rc.d`, a tool similar to the `chkconfig`.

```
cd /etc/init.d
sudo update-rc.d nexus defaults
sudo service nexus start
```

In the second line you will run a default priority to add the `nexus` service before starting it.

systemd.

This example is a script that uses `systemd` to run the repository manager service. Create a file called `nexus.service`. Add the following contents, then save the file in the `/etc/systemd/system/` directory.

```
[Unit]
Description=nexus service
After=network.target

[Service]
Type=forking
ExecStart=/opt/nexus/bin/nexus start
ExecStop=/opt/nexus/bin/nexus stop
User=nexus
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

Activate the service with the following commands:

```
sudo systemctl daemon-reload
sudo systemctl enable nexus.service
sudo systemctl start nexus.service
```

After starting the service for any Linux-based operating systems, verify that the service started successfully.

```
tail -f /opt/nexus/data/log/nexus.log
```

The tail command verifies that the service has been started successfully. If successful, you should see a message notifying you that it is listening for HTTP.

2.7.2 Running as a Service on Windows

The startup script that runs Nexus Repository Manager Pro and Nexus Repository Manager OSS on Windows platforms is `bin/nexus.exe`. The script includes standard commands for starting and stopping the service. It also contains commands `install` and `uninstall` to create and delete the configuration for the service.

You can create the service configuration with:

```
nexus.exe /install
```

The created service is available named *nexus* in common console application to manage services such as Windows Services. You can start, stop and restart the service there as well as configure it to start as part of a operating system startup.

2.7.3 Running as a Service on Mac OS X

The standard way to run a service on Mac OS X is to use `launchd`, a program that starts, stops and manages daemons and scripts in Apple OS X environments. To run the service you need to create an XML document called with the file extension `.plist` to define its properties. An example plist file for the repository manager 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 the repository manager and adapt permissions and the `RUN_AS_USER` setting in the `nexus` startup script.

With this setup the repository managers, starts 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
```

2.8 Running Behind a Reverse Proxy

Available in Nexus Repository OSS and Nexus Repository Pro

Nexus Repository Manager is a sophisticated server application with a web-application user interface, answering HTTP requests using the high-performance servlet container **Eclipse Jetty**.

Organizations are sometimes required to run applications like Nexus Repository Manager behind a **reverse proxy**. Reasons may include:

- security and auditing concerns
- network administrator familiarity
- organizational policy
- disparate application consolidation
- virtual hosting
- exposing applications on restricted ports
- SSL termination

This section provides some general guidance on how to configure common reverse proxy servers to work with Nexus Repository Manager. Always consult your reverse proxy administrator to ensure you configuration is secure.

The default webapp context path for the repository manager user interface is `/`. In the instance where the repository manager needs to be proxied at a different base path you must change the default path by editing a property value. In Section 4.2.4 follow the steps to change or update the base URL if you want an alternate server name.

In the following examples, review the sections on changing the **HTTP port** and **context path** to properly reverse-proxy the repository manager.

2.8.1 Example: Reverse Proxy on Restricted Ports

Scenario: You need to expose the repository manager on restricted port 80. The repository manager should not be run with the root user. Instead run your reverse proxy on the restricted port 80 and the repository manager on the default port 8081. End users will access the repository manager using the virtual host URL **`http://www.example.com/nexus`** instead of **`http://localhost:8081/nexus`**.

Ensure your external hostname (`www.example.com`) routes to your reverse proxy server. In this example use the default content path (`/`)

Apache httpd.

```
ProxyRequests Off
ProxyPreserveHost On

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

nginx.

```
http {

    proxy_send_timeout 120;
    proxy_read_timeout 300;
    proxy_buffering off;
    keepalive_timeout 5 5;
    tcp_nodelay on;

    server {
        listen *:80;
        server_name www.example.com;

        # allow large uploads of files - refer to nginx ↵
        # documentation
        client_max_body_size 1G;

        # optimize downloading files larger than 1G - refer to ↵
        # nginx doc before adjusting
        #proxy_max_temp_file_size 2G;

        location /nexus {
            proxy_pass http://localhost:8081/nexus;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For ↵
            $proxy_add_x_forwarded_for;
        }
    }
}
```

2.8.2 Example: Reverse Proxy Virtual Host at Custom Context Path

Scenario: You need to expose the repository manager using a custom host name `repo.example.com` on a restricted port at a base path of `/nexus`.

Ensure your external hostname (`repo.example.com`) routes to your reverse proxy server and edit the webapp path a slash at end (/).

Apache httpd.

```
ProxyRequests Off
ProxyPreserveHost On

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

nginx.

```
http {
    proxy_send_timeout 120;
    proxy_read_timeout 300;
    proxy_buffering off;
    keepalive_timeout 5 5;
    tcp_nodelay on;

    server {
        listen *:80;
        server_name repo.example.com;

        # allow large uploads of files - refer to nginx ↔
        # documentation
        client_max_body_size 1G;

        # optimize downloading files larger than 1G - refer to ↔
        # nginx doc before adjusting
        # proxy_max_temp_file_size 2G;

        location / {
            proxy_pass http://localhost:8081/nexus;
```

```
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For ←
            $proxy_add_x_forwarded_for;
    }
}
)
```

2.8.3 Example: Reverse Proxy SSL Termination at Base Path

Scenario: Your organization has standardized a reverse proxy to handle SSL certificates and termination. The reverse proxy virtual host will accept HTTPS requests on the standard port 443 and serve content from the repository manager running on the default non-restricted HTTP port 8081 transparently to end users.

Ensure your external host name (`repo.example.com`) routes to your reverse proxy server and edit the webapp path to be slash (/).

To test your configuration, review the steps to [generate a self-signed SSL certificate](#) for reverse proxy servers.

Apache httpd. Ensure Apache httpd is loading `mod_ssl`.

```
Listen 443

ProxyRequests Off
ProxyPreserveHost On

<VirtualHost *:443>
    SSLEngine on

    SSLCertificateFile "example.pem"
    SSLCertificateKeyFile "example.key"

    ServerName repo.example.com
    ServerAdmin admin@example.com
    ProxyPass / http://localhost:8081/
    ProxyPassReverse / http://localhost:8081/
    RequestHeader set X-Forwarded-Proto "https"

    ErrorLog logs/repo.example.com/nexus/error.log
    CustomLog logs/repo.example.com/nexus/access.log common
```

```
</VirtualHost>
```

nginx. Make sure nginx is compiled using the `--with-http_ssl_module` option.

```
http {  
  
    proxy_send_timeout 120;  
    proxy_read_timeout 300;  
    proxy_buffering     off;  
    keepalive_timeout   5 5;  
    tcp_nodelay         on;  
  
    server {  
        listen        *:443;  
        server_name    repo.example.com;  
  
        # allow large uploads of files - refer to nginx ↵  
        documentation  
        client_max_body_size 1G;  
  
        # optimize downloading files larger than 1G - refer to ↵  
        nginx doc before adjusting  
        #proxy_max_temp_file_size 2G;  
  
        ssl on;  
        ssl_certificate      example.pem;  
        ssl_certificate_key  example.key;  
  
        location / {  
            proxy_pass http://localhost: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;  
            proxy_set_header X-Forwarded-Proto "https";  
        }  
    }  
}
```

Note

Consult your reverse proxy product documentation for details: [Apache httpd \(mod_proxy, mod_ssl\)](#), [nginx \(ngx_http_proxy_module, ssl compatibility\)](#)

2.9 Accessing the User Interface

Once the repository manager is started, the application is listening on the configured IP address range and port. By default any IP address and port 8081 are used. To access the web application user interface, fire up a web browser and type in the URL `http://serveripaddress:port` e.g. `http://localhost:8081/`. If the repository manager started up successfully and network settings allow you to connect to the server, the user interface looks similar to Figure 2.1.

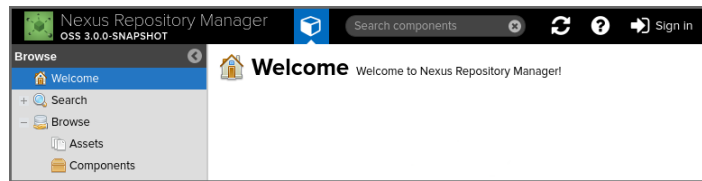


Figure 2.1: Initial User Interface

While the documentation uses `localhost` throughout, 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 the repository manager.

The user interface shows the features available to an anonymous user. The repository manager installation includes an administrative user with full access. Its username is `admin` and the password is `admin123`. You can sign in with the button on the top right corner of the user interface.

Next steps after successfully accessing the user interface are detailed in Chapter 3, Chapter 4 and following chapters about various repository formats and tools such as:

- Chapter 6
- Chapter 7
- Chapter 8
- Chapter 9
- Chapter 10
- Chapter 13

More information about security related topics can be found in Chapter 5.

2.10 Directories

There are two main directories created and used by the repository manager.

Installation directory

This directory contains the Nexus Repository Manager application and all the required additional components such as Java libraries and configuration files. The name of the directory by default uses `nexus-` and is appended with the version name. In this documentation it is referred to as `$install-dir` in any code segments.

Data Directory

This directory contains all the repositories, components and other data that is being stored and managed by the repository manager. It is located within the installation directory by default for archive-based installs and called `data`. In this documentation it is referred to as `$data-dir` in any code segments.

2.10.1 Installation Directory

The installation directory includes a number of nested directories:

```
$ ls -l nexus-3.0.2-02
LICENSE.txt
NOTICE.txt
bin
data
deploy
etc
lib
public
system
```

LICENSE.txt and NOTICE.txt

contain legal details about the license and copyright notices.

bin

contains the `nexus` startup script itself as well as startup-related configuration files.

data

This *data directory* contains all of the repository and configuration data. By default, from a distribution archive install, this directory is nested within the installation directory. More details can be found Section [2.10.2](#).

etc

contains configuration files.

lib

contains binary libraries related to Apache Karaf.

public

contains public resources of the application.

system

contains all components and plugins that constitute the application.

2.10.2 Data Directory

The data directory contains subdirectories such as `blobs`, `db`, `elasticsearch` and others. These contain all the components, repository, configuration and other data presented by the repository manager.

2.11 Configuring the Runtime Environment

Configuring the specifics of the repository manager runtime involves configuration for all components in various configuration files and startup scripts. This section details these and provides recipes for specific tasks.

The startup of the JVM running the repository manager is managed via files in the `$install-dir/bin` directory within the installation. The application startup is performed with the JVM configuration in the file `$install-dir/bin/nexus.vmoptions`:

```
-Xms1200M
-Xmx1200M
-XX:+UnlockDiagnosticVMOptions
-XX:+UnsyncloadClass
-Djava.net.preferIPv4Stack=true
-Dkaraf.home=.
-Dkaraf.base=.
-Dkaraf.etc=etc
-Djava.util.logging.config.file=etc/java.util.logging.properties
-Dkaraf.data=data
-Djava.io.tmpdir=data/tmp
-Dkaraf.startLocalConsole=false
```

The main location for configuration files is the `etc` directory. The directory includes:

config.properties

The main configuration for the Apache Karaf runtime. This file should *not* be modified.

custom.properties

Customizable configuration used by Apache Karaf. This file can be used to pass additional parameters to the Apache Karaf container.

jetty-*.xml

A number of configuration files for Eclipse Jetty

org.apache.* and org.ops4j.*

Various Karaf and OSGi related configuration files.

org.sonatype.nexus.cfg

Main configuration file for the application allowing you to configure aspects such as ports used for HTTP and HTTPS access, location of the data and configuration storage as well as the context path and host.

system.properties

Configuration parameters used for the JVM and application start up.

2.11.1 Updating Memory Allocation and other JVM Parameters

The default and maximum heap sizes for the repository manager are a value of 1200M, suitable for most usage patterns. As a Java application running on the JVM the repository manager is using JVM configuration parameters for numerous settings as part of the startup parameters for the JVM. These values are defined in the configuration file `$install-dir/bin/nexus.vmoptions`. Increased memory configuration can be set with e.g. :

```
-Xms1500M  
-Xmx2G
```

Other JVM parameters such as GC algorithm can be configured in the same location.

2.11.2 Changing the HTTP Port

The default value for the HTTP port used to access the repository manager user interface and resources is 8081. Therefore the user interface would be available at `http://localhost:8081/`. To change or

update the port locate the line `application-port=8081` in `$install-dir/etc/org.sonatype.nexus.cfg`, then edit the number. Here is an example where you would change the port to 9081:

```
application-port=9081
```

Therefore, the exposed URL will be `http://localhost:9081/`.

2.11.3 Changing the Context Path

To change or update the context path in the instance you want point to a specific webapp or component, locate the `nexus-context-path=/` line in `$install-dir/etc/org.sonatype.nexus.cfg`. Here is an example where you expose the user interface to a `components` directory.

```
nexus-context-path=/components/
```

Therefore, if the port is set to 9081, the exposed URL will be `http://localhost:9081/components/`.

2.11.4 Configuring the Data Directory Location

[Distribution archive installation](#) of the repository manager configures the location of the [data directory](#) to be nested inside the application directory.

The configuration of this folder is located in `$install-dir/bin/nexus.vmoptions`. For example, if you want to use the absolute path `/opt/repository/storage/`, you have to change to:

```
-Dkaraf.data=/opt/repository/storage  
-Djava.io.tmpdir=/opt/repository/storage/tmp
```

Note

Find out more about upgrading this version of Nexus Repository Manager to 3.1.0 in the dedicated [support article](#).

2.12 Uninstalling

To uninstall the repository manager from an archive installation, remove the service configuration and delete the entire directory.

Chapter 3

Using the User Interface

3.1 Introduction

This chapter covers the basic aspects of the user interface of Nexus Repository Manager Pro and Nexus Repository Manager OSS applicable for read-only access including an overview of the user interface features, searching components and browsing repositories and other features that are, by default, available to anonymous users and similar read-only roles.

Administrative tasks like configuring repositories, task, security and many other aspects are documented in [Chapter 4](#).

3.2 User Interface Overview

Available in Nexus Repository OSS and Nexus Repository Pro

The user interface is used with a web browser and works best with modern browsers. Older versions such as Microsoft Internet Explorer 8 or earlier are not supported and actively blocked from using the repository manager user interface to avoid an unsatisfactory user experience.

The repository manager provides anonymous access for users who only need to search for components, browse the repositories and access components via client tools such as Maven or NuGet. This anonymous access level is a configurable, read-only mode that includes the main user interface elements as shown in Figure 3.1.

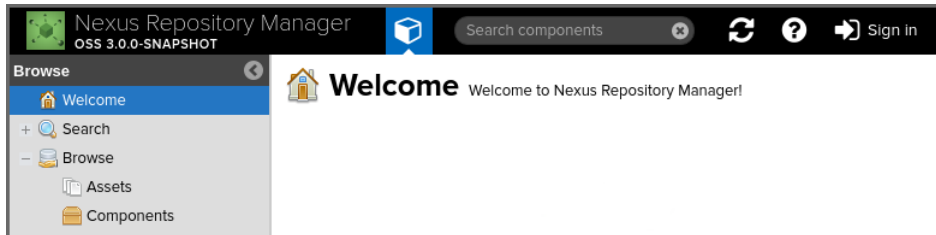


Figure 3.1: User Interface for Anonymous Users

Once a user is logged in further features become available depending on the user's privileges. An example for the *admin* user including the *Administration* menu icon is visible in Figure 3.2.

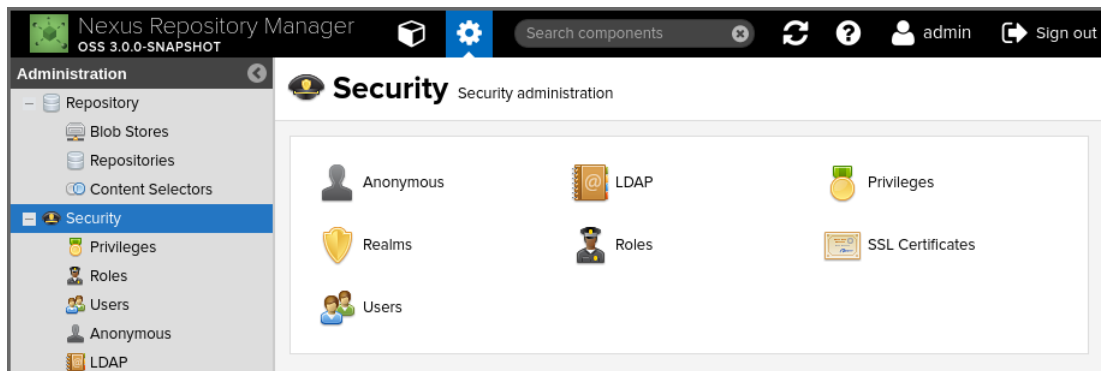


Figure 3.2: User Interface for Logged In *admin* User

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

Main toolbar

The top of the page contains the header with a number of elements starting on the left with the logo:

Logo and version label

The logo and the version label differ for Nexus Repository Manager OSS and Nexus Repos-

itory Manager Pro and allows you to know what version of the repository manager you are accessing at a glance.

Browse button

The browse button allows you to switch to the *Browse* menu items in the main menu section on the left of the user interface. The contents of the menu will depend on your assigned user privileges.

Administration button

The administration button allows to switch to the *Administration* menu items in the main menu section on the left of the user interface as visible in Figure 3.2. The contents of the menu will depend on your assigned user privileges.

Search input box


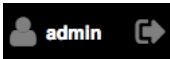
The search input box can be used to start a keyword search. The results are displayed in the feature view panel.

Refresh button

The refresh button is a global refresh button that affects all views in the user interface including the feature view panel. E.g., it refreshes the search results view, the user list or the staging repository list, if they are currently the active feature view.

Help button

Clicking the help button opens up the help menu. It contains a link to specific help about the currently active feature view. The *About* item displays a dialog with details about the version as well as license and copyright information. The *Documentation*, *Knowledge base*, *Community*, *Issue tracker* and *Support* items link to the respective pages on the Sonatype websites.

Sign In  and user account/signout buttons ; The *Sign In* button allows you to sign in to the user interface as a specific user. Doing so gives you access to the privileges assigned to the user, changes the *Sign in* button to a *Sign out* button and adds a button displaying the user's name. The user's name button functions to access the *Account* feature view as part of the *User* menu in the main menu on the left with any other user features the account can access.

Main Menu

The main menu on the left contains either the *Browse*, the *Administration* or the *User* menu items. The exact list of available menu items depends on the current user's assigned privileges. E.g., the *Administration* menu as visible in Figure 3.2 includes the *Security* section, which is not available to anonymous users by default. 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. Selecting a menu item triggers the display of the respective feature view in the feature view panel.

Feature View Panel

The feature view panel in the center of the user interface right of the main menu initially displays the *Welcome* feature view. It changes display based on your selected item in the main menu.

Figure 3.3 shows a typical user interface appearance of the repository manager with the *Users* feature view in the feature view panel. It shows a list of users.

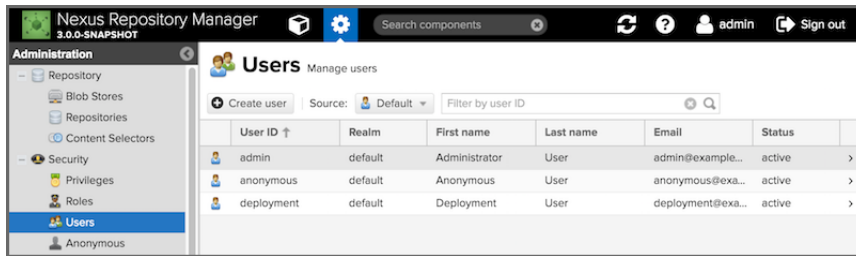


Figure 3.3: Typical Example Interface with a List

Clicking on a row in the list, switches the feature view to a specific display for the item in the row as visible in Figure 3.4. The top level navigation allows you get back to the list by clicking on the *Users* label. The form below has a number of sections that can be accessed via buttons as well as specific functionality like deletion and their associated buttons.

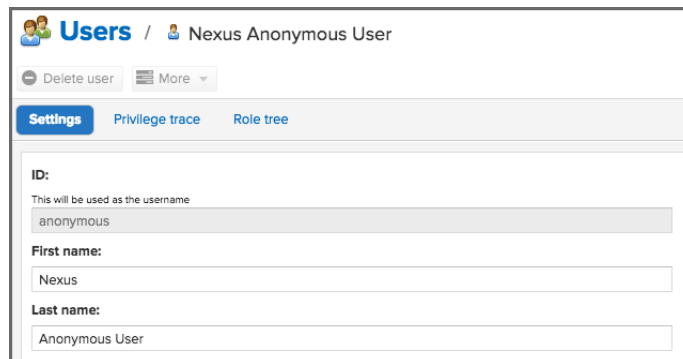
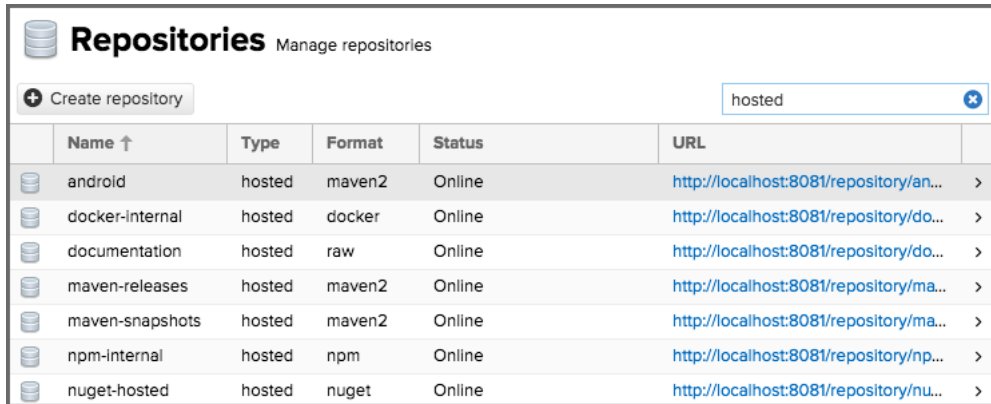


Figure 3.4: Typical Example Interface for Editing and Viewing

The list header features buttons for various operations that differ per list as well as an input box that allows you to filter the list by any terms used in any column. Figure 3.5 shows an example use case where

a user typed "Hosted" in the filter box and the list of repositories only shows hosted repositories. This filtering works for all columns in a list and can be used in most list displays in the repository manager. 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 screenshot shows the 'Repositories' management page in Nexus. At the top, there's a 'Create repository' button and a search filter box containing the text 'hosted'. Below this is a table listing the repositories. The table has columns for Name, Type, Format, Status, and URL. All listed repositories are of type 'hosted' and status 'Online'.

	Name ↑	Type	Format	Status	URL	
	android	hosted	maven2	Online	http://localhost:8081/repository/an...	>
	docker-internal	hosted	docker	Online	http://localhost:8081/repository/do...	>
	documentation	hosted	raw	Online	http://localhost:8081/repository/do...	>
	maven-releases	hosted	maven2	Online	http://localhost:8081/repository/ma...	>
	maven-snapshots	hosted	maven2	Online	http://localhost:8081/repository/ma...	>
	npm-internal	hosted	npm	Online	http://localhost:8081/repository/np...	>
	nuget-hosted	hosted	nuget	Online	http://localhost:8081/repository/nu...	>

Figure 3.5: Filtering the Repository List to Display Only Hosted Repositories

The column headers in most lists can be clicked to invoke a sorting of the list by the respective column as well as activate and deactivate specific columns.

3.3 Searching for Components

Available in Nexus Repository OSS and Nexus Repository Pro

Searching components in the repository manager is an important use case for being able to access information about specific components including different versions that are available, security and license data and other information as well as for build tool migrations, download of deployment packages and other component related development, QA and operations activities.

The different search modes can be accessed with the *Browse* button in the main toolbar and selecting *Search* or one of the nested options like *Custom*, *Maven* and others. The common feature view with the criteria drop-down selector for the search without results is displayed in Figure 3.6.

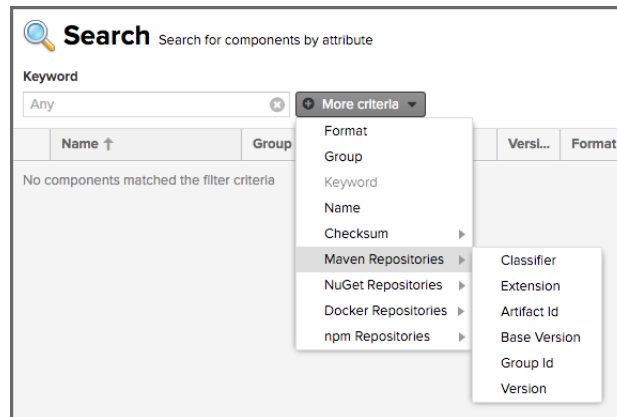


Figure 3.6: Keyword Search with *Format* and *More criteria* Input

Beneath the search title is the search criteria input area that displays the current criteria input e.g., *Keyword*. Beside the current criteria is a *More Criteria* button that allows you to add further criteria to your search. Each criteria can be removed by clicking on the minus/dash icon within the criteria input box. The cross/x in the input box resets the value. In Figure 3.6 you can see the *Format* criteria added to the search.

Each criteria can be used with a search term and supports the * character (star, asterisk) for pattern matching. E.g., you could search with the *Group* search criteria and search for `org.sonatype.nexus.*`. This would return components with the group of `org.sonatype.nexus`, but also `org.sonatype.nexus.plugins` and many others.

3.3.1 Search Criteria and Component Attributes

A number of criteria can be used with any repository format and returns results from all components in all repositories:

Keyword

A keyword is a string used for a search, where matches in *Format*, *Group*, *Name*, *Version* and all other component metadata values are returned.

Format

The format of the repository in which to look for a component. E.g. Nexus Repository Manager OSS supports `maven2`, `docker`, `nuget` and `raw`.

Group

An identifier that groups components in some way, such as by organization. It can also be used to simply to create a specific namespace for a project. Not all repository formats use the notion of a group. Some tools simply use a different name for the concept e.g., `org` for Apache Ivy or `groupId` for Apache Maven and the *maven2* repository format. In the case of a maven2 repository, `group` is a required attribute. Other formats, like the *nuget* repository format, do not use `group` at all.

Name

The name of a component constitutes its main identifier. Different repository formats use a different name for the concept such as `artifactId` for Apache Maven and the *maven2* repository format.

Version

The version of a component allows you to have different points in time of a component released. Various tools such as Maven or NuGet use the term version. Other build systems call this differently e.g. `rev`, short for revision, in the case of Apache Ivy. In most repository formats version numbers are not enforced to follow a specific standard and are simply a string. This affects the sort order and can produce unexpected results.

Checksum - MD5, SHA-1, SHA-256 or SHA-512

A checksum value of a component file generated by an MD5, SHA-1, SHA-256 or SHA-512 algorithm.

In addition there are criteria that can be used to search for components in repositories with specific formats only:

Maven Repositories**Group Id**

The Maven `groupId` for a component. Other build systems supporting the Maven repository format call this differently e.g. `org` for Apache Ivy and `group` for Gradle and Groovy Grape. *Group Id* is equivalent to *Group*.

Artifact Id

The Maven `artifactId` for a component. Other build systems call this differently e.g. `name` for Apache Ivy and Gradle, and `module` for Groovy Grape. *Artifact Id* is equivalent to *Name*.

Classifier

The Maven *classifier* for a component. Common values are `javadoc`, `sources` or `tests`.

Packaging

The Maven `packaging` for a component, which is `jar` by default. Other values as used in Maven and other build tools are `ear`, `war`, `maven-plugin`, `pom`, `ejb`, `zip`, `tar.gz`, `aar` and many others.

Base Version

The base version of the component/asset. Typically this is the same value as the version for release components. `SNAPSHOT` development components use a time-stamped version but the base version uses the `SNAPSHOT` version e.g. version of `1.0.0-20151001.193253-1` and base version of `1.0.0-SNAPSHOT`.

Extension

The extension used for a specific asset of a component.

npm Repositories

Additional criteria for component searches in *npm Repositories* are:

Group

The *npm Group* is equivalent to the component group, also sometimes referred to as scope by npm users.

Name

The *npm Name* is equivalent to the component version.

Version

The *npm Version* is equivalent to the component version.

NuGet Repositories**ID**

The NuGet component identifier is known as `Package ID` to NuGet users.

Tags

Additional information about a component formatted as space-delimited keywords, chosen by the package author.

Docker Repositories**Image Name**

The name for the Docker image. It is equivalent to the *Name* of the component in the repository manager that represents the Docker image.

Image Tag

The tag for the Docker image. It is equivalent to the *Version* of the component in the repository manager that represents the Docker image.

Layer Id

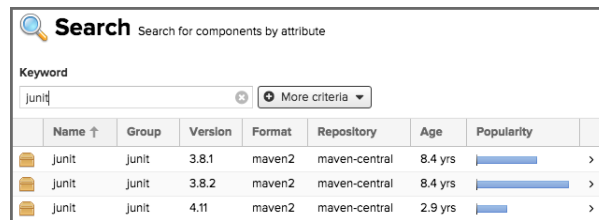
The unique identifier for a Docker image layer. It is equivalent to the *layerId* attribute of the component in the repository manager that represents the Docker image.

Raw Repositories

Searches in *Raw Repositories* can be narrowed down with the *Path* criteria. It allows you to specify a file path to the components in the raw repository. The search can return all components or files with the respective path pattern.

3.3.2 Search Results

Once you have provided your search terms in one or multiple criteria input fields, like the *Keywords* criteria in the *Search* feature view, the results become visible in the component list, with an example displayed in Figure 3.7. The components are listed with their *Name*, *Group*, *Version*, *Format*, *Repository*, *Age* and *Popularity* information and are sorted alphabetically by *Name*. Columns and sort order can be adjusted like in all other lists.



The screenshot shows a search interface with a search bar containing 'junit'. Below the search bar is a table with 7 columns: Name, Group, Version, Format, Repository, Age, and Popularity. The table contains three rows of results for 'junit'.

	Name ↑	Group	Version	Format	Repository	Age	Popularity
	junit	junit	3.8.1	maven2	maven-central	8.4 yrs	<div><div></div></div>
	junit	junit	3.8.2	maven2	maven-central	8.4 yrs	<div><div></div></div>
	junit	junit	4.11	maven2	maven-central	2.9 yrs	<div><div></div></div>

Figure 3.7: Results of a Component Search for `junit`

The *Age* column displays the age of the component. The age of a component is typically calculated from the initial release to a repository — typically a public repository such as the Central Repository. Since most Java components are published to the Central Repository when released, this age gives you a good indication of the actual time since the release of the component. For other repository formats and related upstream public repositories the availability of data may differ.

The *Popularity* column shows a relative popularity as compared to the other component versions. 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.

Selecting a component in the list changes to a display of the component information documented in Section 3.5.

3.3.3 Preconfigured Searches

Keyword Search

The main toolbar includes a *Search components* text input field. Type your search term and press *enter* and the repository manager performs a search by *Keyword*.

The same search can be accessed by selecting the *Search* item in the *Browse* main menu. The search term can be provided in the *Keyword* input field in the *Search* feature view.

Custom Search

A configurable search using the criteria you select is available via the *Custom* menu item in the *Search* section of the *Browse* main menu. Initially it has no criteria and it allows you to create a search with criteria you add with the *More Criteria* button.

Bower Search

The *Bower* search is a predefined search available via the *Bower* menu item in the *Search* section of the *Browse* menu. It defaults to inputs for *Name* and *Version* and supports adding further criteria. The format is configured to *bower*.

Docker Search

The *Docker* search is a predefined search available via the *Docker* menu item in the *Search* section of the *Browse* main menu. It defaults to inputs for *Image Name*, *Image Tag* and *Layer Id* and supports adding further criteria. The format is configured to *docker*.

Maven Search

The *Maven* search is a predefined search available via the *Maven* menu item in the *Search* section of the *Browse* main menu. It defaults to inputs for *Group Id*, *Artifact Id*, *Version*, *Base Version*, *Classifier* and *Extension* and supports adding further criteria. The format is configured to *maven2*.

NuGet Search

The *NuGet* search is a predefined search available via the *NuGet* menu item in the *Search* section of the *Browse* main menu. It defaults to inputs for *ID* and *Tags* and supports adding further criteria. The format is configured to *nuget*.

npm Search

The *npm* search is a predefined search available via the *npm* menu item in the *Search* section of the *Browse* main menu. It defaults to inputs for *Scope*, *Name* and *Version* and supports adding further criteria.

Raw Search

The *Raw* search is a predefined search available via the *Raw* menu item in the *Search* section of the *Browse* main menu. It defaults to an input for *Path* and supports adding further criteria. The format is configured to *raw*.

3.3.4 Example Use Case - SHA-1 Search

Sometimes it is necessary to determine the version of a component, where you only have access to the binary file without any detailed component information. When attempting this identification and neither

the filename nor the contents of the file contain any useful information about the exact version of the component, you can use *SHA-1* search to identify the component.

Create a sha1 checksum, e.g., with the `shasum` command available on Linux or OSX or `fciv` on Windows, and use the created string in a *Custom* search by adding the *SHA-1* criteria from the *Checksum* section of the *More criteria* control.

The search will return a result, which will provide you with the detailed information about the file allowing you to replace the file with a dependency declaration. E.g. you can derive the Maven coordinates of a jar file and use them in a dependency declaration.


Tip

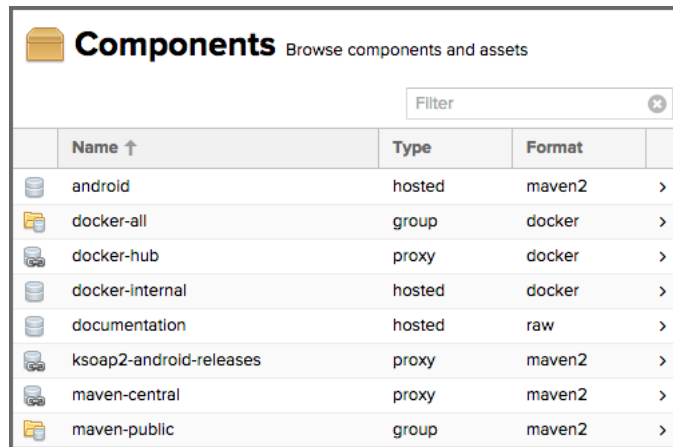
A SHA-1 or similar checksum search can be a huge timesaver when migrating from a legacy build system, where the used libraries are checked into the version control system as binary components with no version information available.

3.4 Browsing Repositories and Repository Groups

Available in Nexus Repository OSS and Nexus Repository Pro

One of the most straightforward uses of the repository manager is to browse the contents of a repository or a repository group. Browsing allows you to inspect the contents of any repository or repository group for all the supported repository formats.

Click on the Browse button  in the main toolbar to access the *Browse* menu and the *Components* and *Assets* menu items. The *Component* as well as the *Assets* feature views allowing you to select a repository or repository group to browse from the list of all repositories as displayed in Figure 3.8.



The screenshot shows the 'Components' page in Nexus, titled 'Browse components and assets'. It features a search bar labeled 'Filter' with a star icon. Below the search bar is a table with four columns: 'Name' (with an upward arrow), 'Type', 'Format', and an empty column. The table lists eight repositories, each with a small icon to its left and a right-pointing arrow to its right. The repositories are: 'android' (hosted, maven2), 'docker-all' (group, docker), 'docker-hub' (proxy, docker), 'docker-internal' (hosted, docker), 'documentation' (hosted, raw), 'ksoap2-android-releases' (proxy, maven2), 'maven-central' (proxy, maven2), and 'maven-public' (group, maven2).

	Name ↑	Type	Format	
	android	hosted	maven2	>
	docker-all	group	docker	>
	docker-hub	proxy	docker	>
	docker-internal	hosted	docker	>
	documentation	hosted	raw	>
	ksoap2-android-releases	proxy	maven2	>
	maven-central	proxy	maven2	>
	maven-public	group	maven2	>

Figure 3.8: List of Repositories to Access for Component Browsing

Once you clicked on the row for a specific repository a list of components in the repository is displayed. It uses the same columns as the search results displayed in Figure 3.7. You can filter the list content, change the rows and select ordering.

3.5 Viewing Component Information

Available in Nexus Repository OSS and Nexus Repository Pro

Once you located a component by browsing a repository or via a search and selected it in the list, you see the component information and a list of associated assets. An example is displayed in Figure 3.9.

The information displayed includes the name and format of the repository that contains the component as well as the component identifiers *Group*, *Name* and *Version*. *Most popular version* contains the version number of the same component that is most popular in its usage within a specific group and name. *Popularity* shows a relative percentage of popularity between the displayed component against all other versions of this component. A value of 100% signals this version to be the most popular. 50% means that the specific version is half as popular as the most popular version. Popularity data is provided by the Sonatype Data Services based on requests from the Central Repository and other data and not available for all components. *Age* shows the age of the component.

None of the popularity or age data is viewable without Repository Health Check enabled.

A list of one or more assets associated with the component is shown below the component information. Click on the row with the *Name* of the asset you want to inspect to view the asset information documented in Section 3.6.

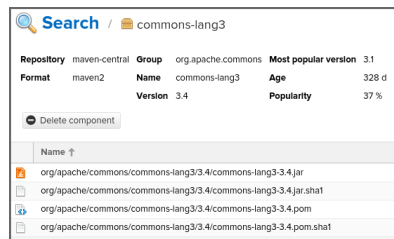


Figure 3.9: Example for Component Information and List of Associated Assets

To delete a component press the *Delete component* button as shown in Figure 3.9. A modal will pop up to confirm the deletion. You can only delete components from hosted and proxy repositories. A deletion of a components triggers the deletion of all its associated assets, in most repository formats.

Note

In some repository formats assets are shared across component and these remain after deletion a component deletion. For example, while a Docker image is a component and can be deleted, the layers that make it up remain after its deletion as these assets are potentially shared with other Docker images.

3.6 Viewing Asset Information

Available in Nexus Repository OSS and Nexus Repository Pro

Asset information can be accessed by browsing assets directly or from a component information view. The *Delete* button allows you to remove an asset. The information itself is broken up into two sections, accessible by tabs below the *Delete* button.

The *Info* section contains a number of attributes about the specific asset. An example is displayed in

Figure 3.10.

Path

the path to the asset in the repository

Content type

the MIME type of the asset

File size

the size of the file in KB

Last updated

the date and time when the asset was last updated

Last accessed

the date and time when the asset was last accessed

Locally cached

set to *true* if the asset can be found in the repository manager storage, *false* indicates that the metadata about the asset is available, while the asset itself has not been downloaded

Blob reference

a unique identifier pointing at the the binary blob representing the asset in the repository manager storage

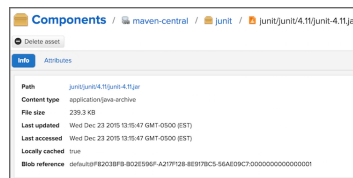
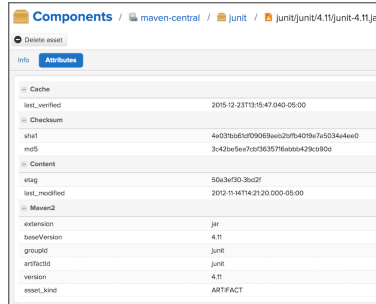


Figure 3.10: Asset Info Example

The *Attributes* section contains further metadata about the asset related to *Cache*, *Checksum* and *Content attributes*. An example is displayed in Figure 3.11.

Assets can include format specific attributes displayed in additional sections. For example an asset in a Maven2 repository has a *Maven2* section with attributes for *extension*, *baseVersion*, *groupId*, *artifactId*, *version* and *asset_kind*.



The screenshot shows the 'Components' page in Nexus, specifically the 'Attributes' tab for the artifact 'junit:junit:4.11:junit-4.11.jar'. The page includes a 'Delete asset' button and a table of attributes.

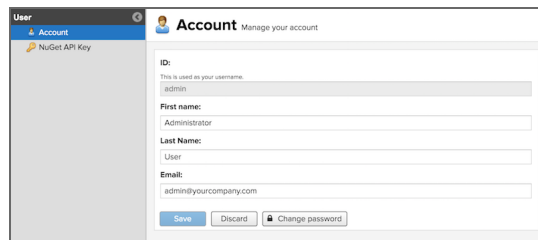
Attribute	Value
Cache	
last_verified	2015-12-23T13:15:47.040-05:00
Checksum	
sha1	4a0396a65f00089a6020f64096c7d5034a4ee0
md5	3c42be5ee7c0f35376e00a429c180d
Content	
etag	50a3ef30-30x2f
last_modified	2012-11-14T14:21:20.000-05:00
Maven2	
extension	jar
baseVersion	4.11
groupId	junit
artifactId	junit
version	4.11
asset_kind	ARTIFACT

Figure 3.11: Asset Attributes Example

3.7 Working with Your User Profile

Available in Nexus Repository OSS and Nexus Repository Pro

As a logged-in user, you can click on your user name on the right-hand side of the main toolbar to switch the main menu to contain the *User* menu. Pressing on the *Account* menu item displays the *Account* feature in the main feature panel as displayed in Figure 3.12.



The screenshot shows the 'Account' feature panel in Nexus, titled 'Manage your account'. It contains a form for editing user details. The 'ID' field is pre-filled with 'admin'. The 'First name' field is pre-filled with 'Administrator'. The 'Last Name' field is pre-filled with 'User'. The 'Email' field is pre-filled with 'admin@yourcompany.com'. There are 'Save', 'Discard', and 'Change password' buttons at the bottom.

Field	Value
ID	admin
First name	Administrator
Last Name	User
Email	admin@yourcompany.com

Figure 3.12: Editing User Details in the Account Feature Panel

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

3.7.1 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* button. You will be prompted to authenticate with your current password and subsequently supply your new password in pop up dialogs.

Tip


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

Chapter 4

Configuration

Available in Nexus Repository OSS and Nexus Repository Pro

4.1 Introduction

This chapter covers all aspects of configuring Nexus Repository Manager Pro and Nexus Repository Manager OSS. Specifically the sections and menu items of the *Administration* main menu are covered. It can be accessed by authorized users by pressing the *Administration* button  in the main toolbar.

Tip

The default user for accessing these features has the username *admin* and the password *admin123*. More fine-grained access can be configured as detailed in [Chapter 5](#).

The *Administration* menu contains the following sections:

Repository

The *Repository* section allows you to manage all *Repositories* and related configurations such as *Routing* and *Targets*.

Security

This section provides access to all the configuration features related to authentication and authorization of users including *Privileges*, *Roles*, *Users*, but also *LDAP*, *Atlassian Crowd*, *SSL Certificates* and *User Token*.

Support

Access a number of features that allow you to administer and monitor your repository manager successfully like *Logging* and *System Information*.

System

The [general configuration](#) for getting started and running the repository manager with e.g., [HTTP](#) or [Email Server](#) settings, but also [Capabilities](#) and [Tasks](#) to run regularly and other configurations.

4.2 System Configuration

The *System* section of the *Administration* menu gives you access to a number of configuration features that you typically need to configure, after successful installation. The following sections detail:

- Access information about used [Bundles](#)
- Advanced configuration with [Capabilities](#)
- [Email/SMTP server configuration](#)
- [HTTP/HTTPS proxy server configuration](#)
- Setting a [Base URL](#) for the repository manager
- Configuration and management of automated maintenance [Tasks](#)

4.2.1 Bundles

Available in Nexus Repository OSS and Nexus Repository Pro

The Nexus Repository Manager application runs on the OSGi container [Apache Felix](#). All features and plugins are managed by the container and are implemented as OSGi bundles.

The *Bundles* feature view is available in the *System* section of the *Administration* main menu. It allows you to inspect a list of all the OSGi bundles that are deployed as part of the application and access detailed information about each bundle.

Tip

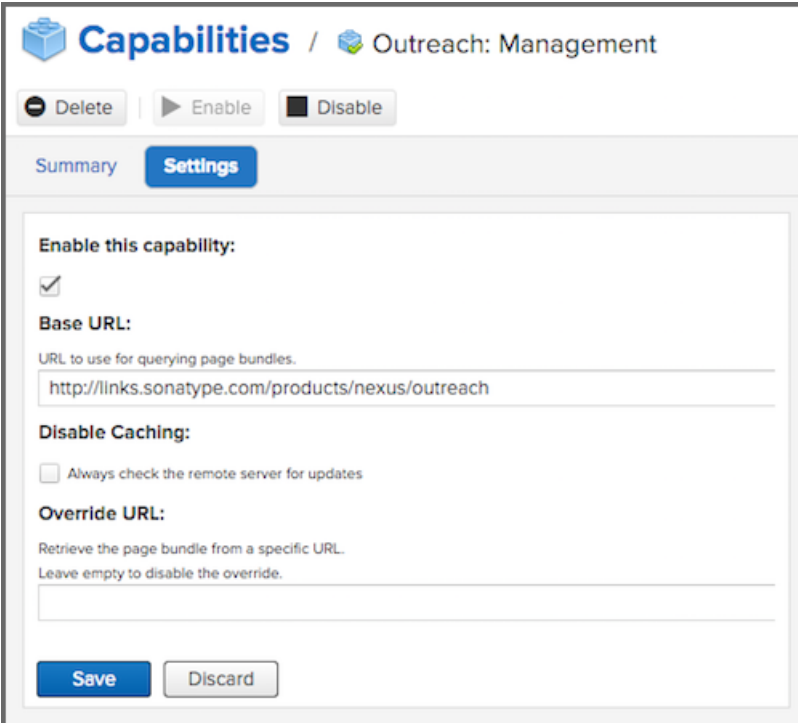
Find out more about OSGi and OSGi bundles on the [website of the OSGi Alliance](#).

4.2.2 Accessing and Configuring Capabilities

Available in Nexus Repository OSS and Nexus Repository Pro

Capabilities are features of the repository manager and plugins that can be configured by a user in a generic administration feature view accessible in the *System* section of the *Administration* main menu via *Capabilities*.

The repository manager ships with a number of capabilities preinstalled and allows you to enable/disable them. An example capability is *Outreach: Management* displayed in Figure 4.1.



The screenshot shows the 'Capabilities' management interface for the 'Outreach: Management' capability. At the top, there are three buttons: 'Delete' (with a trash icon), 'Enable' (with a play icon), and 'Disable' (with a square icon). Below these is a tabbed interface with 'Summary' and 'Settings' tabs; the 'Settings' tab is currently selected. The 'Settings' section contains the following options:

- Enable this capability:** A checkbox that is checked.
- Base URL:** A text field with the value 'http://links.sonatype.com/products/nexus/outreach'. Above the field is the text 'URL to use for querying page bundles.'
- Disable Caching:** A checkbox that is unchecked, with the text 'Always check the remote server for updates' below it.
- Override URL:** A text field that is empty. Above the field is the text 'Retrieve the page bundle from a specific URL. Leave empty to disable the override.'

At the bottom of the settings section are two buttons: 'Save' and 'Discard'.

Figure 4.1: Outreach: Management Capability Settings

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:

State

The state column does not have a title. Enabled capabilities have a green checkmark added on top of a blue icon. 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.

Description

The *Description* column contains further descriptive information about the capability.

Notes

The *Notes* column can contain user created text about the capability.

Every capability can be inspected and configured by selecting it in the list. The resulting view display the *Summary* view of the specific capability. This view includes the display of *Type*, *State* and optionally *Category* and *Description* in the *Summary* section as well as further information in the *Status*, *About* and *Notes* sections. The *Status* section 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. The *About* section displays a descriptive text about the purpose of the capability. 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 *Delete* button below the title allows you to remove a capability and it's configuration entirely. The *Enable* and *Disable* buttons on the other hand can be used to switch the state of the capability.

The *Settings* view allows you to activate or deactivate the capability with the *Enable this capability* checkbox. Below this checkbox, each capability type has specific additional configuration parameters available. Once you have completed the configuration, press the *Save* button.

The capabilities management feature view supports adding new capabilities by pressing the *Create capability* button above the list and selecting the desired capability *Type* from the list. The next view allows you to perform any capability-specific configuration and finish the process by pressing the *Create capability* button below the parameters.

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 of Nexus

Repository Manager Pro can be administrated by using the interface available via the *User Token* menu item in the *Security* section of the *Administration* menu as well as by editing the user token capability. Other capabilities are internal functionality and sometimes managed automatically by the responsible plugin. Some optional configuration like the branding plugin or advanced features of the smart proxy configuration are only done in the capabilities administration.

Usage of specific capabilities to achieve a variety of tasks is detailed in parts of the documentation.

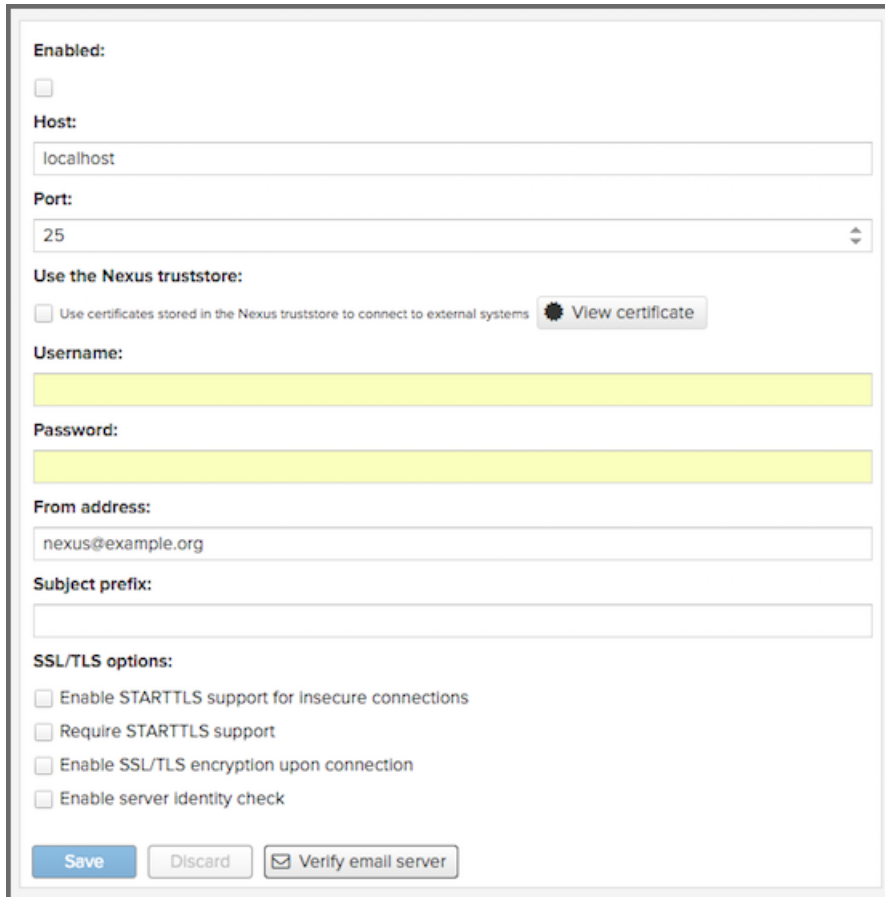
**Warning**

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

4.2.3 Email Server

Available in Nexus Repository OSS and Nexus Repository Pro

The repository manager may send out email messages for a number of reasons. In order for these messages to be delivered, you need to configure the connection to the SMTP server under the *Email Server* menu item in the *System* section of the *Administration* menu as displayed in [Figure 4.2](#).



The image shows a configuration window for an email server. It contains several sections: 'Enabled' with a checkbox; 'Host' with a text field containing 'localhost'; 'Port' with a spinner box set to '25'; 'Use the Nexus truststore' with a checkbox and a 'View certificate' button; 'Username' and 'Password' with yellowed-out text fields; 'From address' with a text field containing 'nexus@example.org'; 'Subject prefix' with an empty text field; and 'SSL/TLS options' with four unchecked checkboxes. At the bottom are 'Save', 'Discard', and 'Verify email server' buttons.

Enabled:

☐

Host:

localhost

Port:

25

Use the Nexus truststore:

☐ Use certificates stored in the Nexus truststore to connect to external systems [View certificate](#)

Username:

Password:

From address:

nexus@example.org

Subject prefix:

SSL/TLS options:

☐ Enable STARTTLS support for insecure connections

☐ Require STARTTLS support

☐ Enable SSL/TLS encryption upon connection

☐ Enable server identity check

[Save](#) [Discard](#) [Verify email server](#)

Figure 4.2: Email Server Configuration

The following configuration options are available:

Enabled

Determines whether email sending is activated or not, independent of a server being configured.

Host and Port

The name of the host and the port to use to connect to the SMTP server.

Use the Nexus truststore

This checkbox allows you to configure the repository manager to use its certificate truststore. You can also view and import the certificate from the email server. Further details are documented in

Section [5.9](#).

Username and Password

The credentials of the user of the SMTP server to use for authentication.

From address

This parameter defines the email address used in the `From:` header of any email sent by the repository manager. Typically, this is configured as a "Do-Not-Reply" email address or a mailbox or mailing list monitored by the administrators of the repository manager.

Subject prefix

This parameter allows you to define a prefix used in the subject line of all emails sent by the repository manager. This allows the recipients to set up automatic filtering and sorting easily. An example is `[Nexus Notification]`.

SSL/TLS options

These options can be used to configure usage of Transport Layer Security (TLS) and Secure Sockets Layer (SSL) when emails are sent by the repository manager using SMTP. These options include the ability to use STARTTLS, which upgrades the initially established, plain connection to be encrypted when sending emails. The following options are available to you:

Enable STARTTLS support for insecure connections

This checkbox allows you to enable support for upgrading insecure connections using STARTTLS.

Require STARTTLS support

This checkbox requires that insecure connections are upgraded using STARTTLS. If this is not supported by the SMTP server, no emails are sent.

Enable SSL/TLS encryption upon connection

This checkbox enables SSL/TLS encryption for the transport on connection using SMTPS/POPS.

Enable server identity check

This option verifies the server certificate when using TLS or SSL, following the RFC 2595 specification.

Once you have configured the parameters you can use the *Verify email server* button to confirm the configured parameters and the successful connection to the server. You are asked to provide an email address that should receive a test email message. Successful sending is confirmed in a message.

4.2.4 Base URL Creation

The *Base URL* is the address end users apply when navigating to the user interface. The repository manager only uses this value to construct absolute URLs to your user interface inside of email notifications.

The most common reason why the address would be different is if you have a [reverse proxy](#) in front of the Nexus Repository Manager, terminating HTTP requests at an address different from where the repository manager is running.

To set the Base URL:

- Go to the *System* section of the *Administration* menu and select *Capabilities*.
- Search for an existing *Base URL* capability and select it if it exists or click the *Create capability* button and choose *Base URL* from the *Select Capability Type* list.
- Enter a new URL value.
- Press the *Create capability* to add the *Base URL*.

4.2.5 HTTP and HTTPS Request and Proxy Settings

Available in Nexus Repository OSS and Nexus Repository Pro

The repository manager uses HTTP requests to fetch content from remote servers. In some cases a customization of these requests is required. Many organizations use proxy servers for any outbound HTTP network traffic. The connection to these proxy servers from the repository manager needs to be configured to allow it to reach remote repositories. All this can be configured in the *HTTP* configuration available via the *System* section of the *Administration* menu and displayed in Figure [4.3](#).

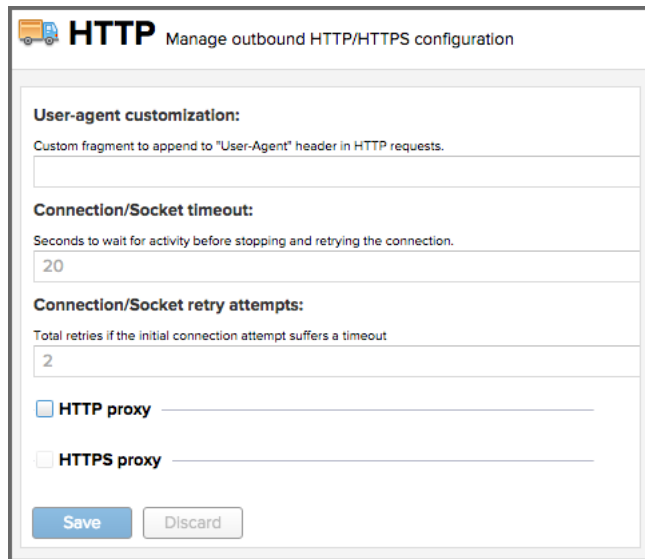


Figure 4.3: Configuring HTTP Request Settings

User-agent customization

The HTTP configuration in *User-agent customization* allows you to append a string to the User-Agent HTTP header field. This can be a required customization by your proxy servers.

Connection/Socket timeout and attempts

The amount of time in seconds the repository manager waits for a request to succeed when interacting with an external, remote repository as well as the number of retry attempts to make when requests fail can be configured with these settings.

If your repository manager instance needs to reach public repositories like the Central Repository via a proxy server, you can configure the connection to a proxy server. Typically such an internal proxy server proxies HTTP as well as HTTPS connections to external repositories. In this case you configure a HTTP proxy. Select the checkbox beside *HTTP Proxy* and configure the parameters in the sections displayed in Figure 4.4. If your organization uses a separate, additional proxy server for HTTPS connections, you have to configure it in the *HTTPS Proxy* section.

Tip

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

☒ HTTP proxy

HTTP proxy host:
No http:// required (e.g. "proxy-host" or "192.168.1.101")

1 This field is required

HTTP proxy port:

1 This field is required

☒ Authentication

Username:

1 This field is required

Password:

Windows NTLM hostname:

Windows NTLM domain:

☐ HTTPS proxy

Hosts to exclude from HTTP/HTTPS proxy:
Accepts Java "http.nonProxyHosts" wildcard patterns (one per line, no \\' hostname delimiters)

+ x

Figure 4.4: Configuring HTTP Proxy Settings

You can specify the *HTTP proxy host* and the *HTTP proxy port* of the HTTP or HTTPS proxy server and, optionally, the *Authentication* details for *Username* and *Password*. If a Windows NT LAN Manager is used to authenticate with the proxy server you can configure the needed connection details in *Windows NTLM hostname* and *Windows NTLM domain*.

In addition, you can configure a number of hosts that the repository manager reaches directly, ignoring the proxy settings. Requests to them should not go through the configured HTTP/HTTPS proxy. These hosts can be configured in the *Hosts to exclude from HTTP/HTTPS proxy* setting. You can add a hostname in the input box and add it with the + button. The * character can be used for wildcard matching for numerous host names allowing a setting such as *.example.com. Entries can be removed with the x button.

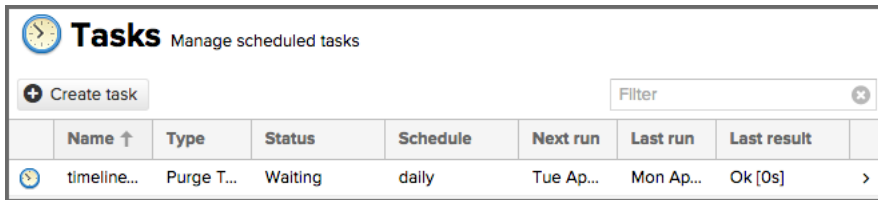
Figure 4.4 shows the *HTTP Proxy* administration interface. The HTTPS configuration interface looks the

same and is found below the HTTP configuration.

4.2.6 Configuring and Executing Tasks

Available in Nexus Repository OSS and Nexus Repository Pro

The repository manager allows you to schedule the execution of maintenance tasks. The tasks can carry out regular maintenance steps that will be applied to all repositories or to specific repositories on a configurable schedule or simply perform other system maintenance. Use the *Tasks* menu item in the *System* section of the *Administration* menu to access the feature view, shown in Figure 4.5, that allows you to manage your *Tasks*.



Name ↑	Type	Status	Schedule	Next run	Last run	Last result	
timeline...	Purge T...	Waiting	daily	Tue Ap...	Mon Ap...	Ok [0s]	>

Figure 4.5: Managing Tasks

The list interface allows you to add new tasks with the *Create task* button as well as inspect and work with the configured tasks. The list shows the following columns:

Name

A user-defined name for the task to identify it in the user interface and log files.

Type

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

Status

Tasks can either be *Waiting* for their next run, currently *Running* or *Disabled*.

Schedule

The *Schedule* column shows the *Task frequency* e.g., *Daily*, *Monthly*, *Manual* and others.

Next run

This column displays date and time of the next execution of the task based on the configured schedule.

Last run and Last result

These columns display the date and time as well as the result and duration of the last execution of the specific task.

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

Task enabled

Enable or disable a specific task with the checkbox.

Notification 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.

Task frequency

Selecting the task frequency allows you to configure the schedule for the task executions. Available choices are *Manual*, *Once*, *Hourly*, *Daily*, *Weekly*, *Monthly* and *Advanced (provide a CRON expression)*. Apart from *Manual*, all choices trigger display of 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 *Start date* and *Start time* allow you to configure a specific date and time from when the schedule should be activated. The *Time to run this task* settings is used to configure the actual time of the task execution.

Task-type specific configuration is displayed below the notification email input and differs for each scheduled task.

The following task types are available to perform specific maintenance:

Compact blob store

Content deleted from a [blob store](#) is not physically deleted from the storage device. Instead it is only internally marked for deletion. This task performs the actual deletion of the relevant files, and therefore frees up the storage space.

Execute script

Scripts can be provided in the *Source* field and have to be written using the Groovy programming language. These scripts can use the APIs of the repository manager to perform maintenance and other modification tasks. Please consult the Javadoc and source for further information.

Publish Maven indexes

Maven indexes can be used to download an index of available components to a client including a developer's IDE, for example. The task publishes the index for all or a specific Maven repository.

Purged incomplete docker uploads

Docker uploads to a repository can be hundreds of MB in size. It is possible to have incomplete uploads or orphaned files remain in temporary storage as a result of incomplete or interrupted uploads. The temporary storage consumed by these incomplete or orphaned uploads can be cleaned up with this task. You can configure the minimum age of incomplete uploads to be purged and have them deleted by the task execution. In addition, any incomplete uploads from docker that have been orphaned by a repository manager restart will be cleaned up whenever the task executes.

Purge orphaned API keys

This scheduled tasks deletes old, unused API keys. These keys are generated, for example, when using the User Token feature or publishing to NuGet repositories. A key becomes unused, when the user account is deleted. The task purges these orphaned API keys and should be scheduled to run regularly to remove these redundant keys.

Purge unused components and assets

This task can be used to remove components and assets in proxy repositories. Any component that has not been requested in the configured number of days will be purged.

Purge unused Maven snapshot versions

This task can be used to remove unused snapshots from Maven repositories. Any snapshot that has not been requested in the configured number of days will be purged.

Rebuild Maven repository metadata

This task rebuilds the `maven-metadata.xml` files with the correct information and will also validate the checksums (`.md5/sha1`) for all files in the specified maven2 hosted repository. The *Group Id*, *Artifact Id* and *Base Version* parameters allow you to narrow down the section of the repository that will be repaired. Typically this task is run manually to repair a corrupted repository.

Rebuild repository index

With support for hosted and proxy repositories, this task can rebuild the index. It inspects actual components and assets found in the repository and rebuilds the index to reflect the true content for supporting search and browse actions.

Remove Maven indexes

This task is the counter-part to the task *Publish Maven indexes* and can remove the index.

Remove snapshots from Maven repository

This task can be scheduled to remove SNAPSHOT-versioned components from a Maven repository. Typically this is useful to preserve storage space, as old SNAPSHOT versions are not accessed after deployment of a new snapshot and no longer added value. The tasks removes all metadata about the components and assets affected, while it does not reclaim disk space used by the binary assets. This can be achieved by running a *Compact blob store* task afterwards. 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 component SNAPSHOT version. For example, if you configured this option with a value of 2, the repository manager will always preserve at least two time-stamped SNAPSHOT versions. 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 component SNAPSHOT versions. For example, if you want to make sure that you are always keeping the last three day's worth of snapshots, configure this option with a value of 3. The *Minimum snapshot count* configuration overrides this setting.

Remove if released

If checked, all SNAPSHOT versions that match any released component found with the same groupId and artifactId coordinates will be removed. For example, if a release version of *com.example:hello-world:1.0.0* is found, all *com.example:hello-world:1.0.0-SNAPSHOT* assets are deleted.

Grace period after release (days)

This parameter allows you to specify a number of days before released snapshots are purged. If a release associated to a snapshot has an updated timestamp and falls within the set grace period, it will not be purged. This setting will give the respective project that references the snapshot dependency time to upgrade to the release component or the next snapshot version.

Caution

The deletion of Maven snapshots when *Remove if released* is checked takes precedence over the number you select in the *Minimum snapshot count* field. Also, it is possible to configure the task in such a way that the results may be unexpected. For example, if configured to keep 0 minimum snapshots older than 0 days, all snapshots everywhere will be deleted, despite whether or not a grace period is configured for releases.

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

Setting up tasks execution adapted to your usage of the repository manager is an important first step when setting up a Nexus Repository Manager instance. Go through the list of task types and consider your usage patterns. In addition update your tasks when changing your usage. E.g., if you start to regularly deploy snapshots by introducing continuous integration server builds with deployment.

4.3 Repository Management

Available in Nexus Repository OSS and Nexus Repository Pro

Repositories are the containers for the components provided to your users as explained in more detail in Chapter 1. Creating and managing repositories is an essential part of your Nexus Repository Manager configuration, since it allows you to expose more components to your users.

It supports proxy repositories, hosted repositories and repository groups using a number of different repository formats.

The binary parts of a repository are stored in blob stores, which can be configured by selecting *Blob stores* from the *Repository* sub menu of the *Administration* menu.

To manage repositories select the *Repositories* item in the *Repository* sub menu of the *Administration* menu.

4.3.1 Blob Stores

A blob store is a storage mechanism for the binary parts of the components and their assets. Each blob store can be used by one or multiple repositories and repository groups. A *default* blob store that is based on a file system storage within the *data* directory configured during the installation is automatically configured.

The *Blob stores* feature view available via the *Blob stores* item in the *Repository* sub menu of the *Administration* menu displays a list of all configured blob stores. The columns provide some detail about each blob store:

Name

the name of the blob store as displayed in the repository administration

Type

the type of the blob store backend, currently only *File* is available representing a file system-based storage

Blob count

the number of blobs currently stored

Total size

the size of the blob store

Available space

the overall space available for the blob store

Click on a specific row to inspect further details of the selected blob store. The details view displays *Type* and *Name* and the absolute *Path* to the file system storage.

The *Create blob store* button allows you to add further blob stores. You can configure the *Type* and *Name* for the blob store. The *Path* parameter should be an absolute path to the desired file system location. It has to be fully accessible by the operating system user account running the Nexus repository manager.

Once a blob store has been created it can no longer be modified and any blob store used by a repository or repository group can not be deleted.

Blobs deleted in a repository are only marked for deletion. The *Compact blob store* [task](#) can be used to permanently delete these *soft-deleted* blobs and therefore free up the used storage space.

4.3.1.1 Choosing the Number of Blob Stores

You will need to choose how many blob stores to create, and how you allocate repositories to these blob stores. This decision should be based on:

- the size of your repositories
- the rate at which you expect them to grow over time
- the storage space available to your Nexus Repository Manager
- the options you have available for adding storage space

For the time being, once a repository is allocated to a blob store, it is there permanently. Blob stores can be moved from one storage device to another (e.g. to a larger storage device) using a manual process, but blob stores cannot be split, nor can repositories span multiple blob stores. For these reasons, your approach to using blob stores should be chosen carefully.

The simplest approach is to create a single blob store per storage device and divide your repositories among them. This is suitable if either:

- Your repositories are growing slowly enough that you won't exceed your available storage within a year
- If you exceed available storage, you will be able to move blob stores to larger storage devices.

You should separate repositories into two or more blob stores per storage device if both:

- You expect to exceed your currently available storage within a year
- You cannot move blob stores to larger storage devices, so you must add capacity to Nexus Repository Manager by adding additional storage devices.

The most flexible approach is to create a separate blob store for each repository, although this is not recommended except in extreme cases of unpredictable capacity because of the administrative complexity.

The current repository-to-blob store limitations will be removed in an upcoming release of Nexus Repository Manager, which will make it possible to revise your repository/blob store approach over time.

4.3.1.2 Estimating Blob Store Size

Blob stores contain two files for each binary component stored in Nexus Repository Manager:

- The binary component, stored as a .bytes file (whose size is the same as the component)
- A properties file that stores a small amount of metadata for disaster recovery purposes (<1k)

The total storage size of a blob store is therefore approximately the total size of all of your components, plus an allowance for the properties files and the block size of your storage device. (On average, this will be $1.5 * \# \text{ of components} * \text{block size}$.)

4.3.2 Proxy Repository

A repository with the type *proxy*, also known as a proxy repository, is a repository that is linked to a remote repository. Any request for a component is verified against the local content of the proxy repository. If no local component is found, the request is forwarded to the remote repository. The component is then

retrieved and stored locally in the repository manager, which acts as a cache. Subsequent requests for the same component are then fulfilled from the local storage, therefore eliminating the network bandwidth and time overhead of retrieving the component from the remote repository again.

By default, the repository manager ships with the following configured proxy repositories:

maven-central

This proxy repository accesses the [Central Repository](#), formerly known as Maven Central. It is the default component repository built into Apache Maven and is well-supported by other build tools like Gradle, SBT or Ant/Ivy.

nuget.org-proxy

This proxy repository accesses the [NuGet Gallery](#). It is the default component repository used by the `nuget` package management tool used for .Net development.

4.3.3 Hosted Repository

A repository with the type *hosted*, also known as a hosted repository, is a repository that stores components in the repository manager as the authoritative location for these components.

By default, the repository manager ships with the following configured hosted repositories:

maven-releases

This hosted repository uses the *maven2* repository format with a release version policy. It is intended to be the repository where your organization publishes internal releases. You can also use this repository for third-party components that are not available in external repositories and can therefore not be retrieved via a configured proxy repository. Examples of these components could be commercial, proprietary libraries such as an Oracle JDBC driver that may be referenced by your organization.

maven-snapshots

This hosted repository uses the *maven2* repository format with a snapshot version policy. It is intended to be the repository where your organization publishes internal development versions, also known as snapshots.

nuget-hosted

This hosted repository is where your organization can publish internal releases in repository using the NuGet repository format. You can also use this repository for third-party components that are not available in external repositories, that could potentially be proxied to gain access to the components.

4.3.4 Repository Group

A repository with the type *group*, also known as repository group, represents a powerful feature of Nexus Repository Manager. They allow you to combine multiple repositories and other repository groups in a single repository. This in turn means that your users can rely on a single URL for their configuration needs, while the administrators can add more repositories and therefore components to the repository group.

The repository manager ships with the following groups:

maven-public

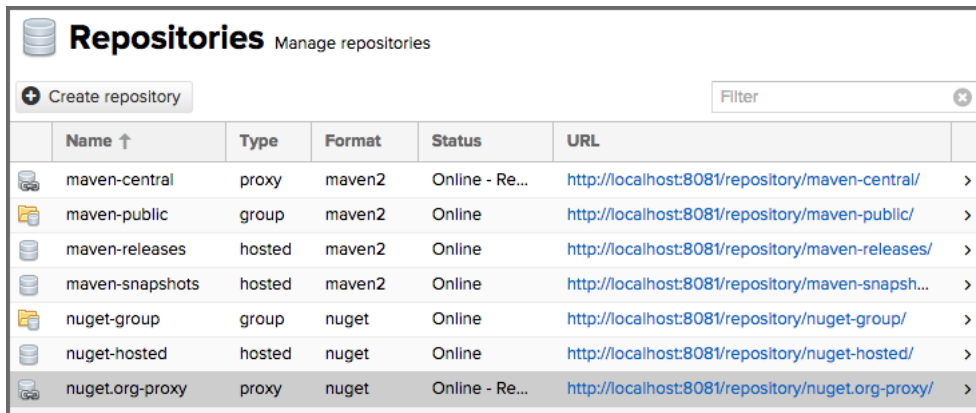
The *maven-public* group is a repository group of *maven2* formatted repositories and combines the important external proxy repository for the Central Repository with the hosted repositories *maven-releases* and *maven-snapshots*. This allows you to expose the components of the Central Repository as well as your internal components in one single, simple-to-use repository and therefore URL.

nuget-group

This group combines the nuget formatted repositories *nuget-hosted* and *nuget.org-proxy* into a single repository for your .Net development with NuGet.

4.3.5 Managing Repositories and Repository Groups

The administration user interface for repositories and repository groups is available via the *Repositories* item in the *Repository* sub menu of the *Administration* menu. It allows you to create and configure repositories as well as delete them and perform various maintenance operations. The initial view displayed in Figure 4.6 features a list of all configured repositories and repository groups.



The screenshot shows the 'Repositories' management interface in Nexus. It features a header with a database icon, the title 'Repositories', and the subtitle 'Manage repositories'. Below the header is a 'Create repository' button and a 'Filter' input field. The main content is a table with columns: Name (with an upward arrow), Type, Format, Status, and URL. Each row represents a repository and includes a small icon on the left and a chevron on the right. The repositories listed are: maven-central (proxy, maven2, Online - Re...), maven-public (group, maven2, Online), maven-releases (hosted, maven2, Online), maven-snapshots (hosted, maven2, Online), nuget-group (group, nuget, Online), nuget-hosted (hosted, nuget, Online), and nuget.org-proxy (proxy, nuget, Online - Re...).








	Name ↑	Type	Format	Status	URL	
	maven-central	proxy	maven2	Online - Re...	http://localhost:8081/repository/maven-central/	>
	maven-public	group	maven2	Online	http://localhost:8081/repository/maven-public/	>
	maven-releases	hosted	maven2	Online	http://localhost:8081/repository/maven-releases/	>
	maven-snapshots	hosted	maven2	Online	http://localhost:8081/repository/maven-snapsh...	>
	nuget-group	group	nuget	Online	http://localhost:8081/repository/nuget-group/	>
	nuget-hosted	hosted	nuget	Online	http://localhost:8081/repository/nuget-hosted/	>
	nuget.org-proxy	proxy	nuget	Online - Re...	http://localhost:8081/repository/nuget.org-proxy/	>

Figure 4.6: List of Repositories

The list of repositories displays some information for each repository in the following columns

Name

the unique name of the repository or repository group

Type

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

Format

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

Status

the status of the repository as well as further information about the status. A functioning repository would show the status to be *Online*. Additional information can e.g., be about SSL certification problems or the status of the remote repository for a currently disabled proxy repository

URL

the direct URL path that exposes the repository via HTTP access and potentially, depending on the repository format, allows access and directory browsing

Health Check

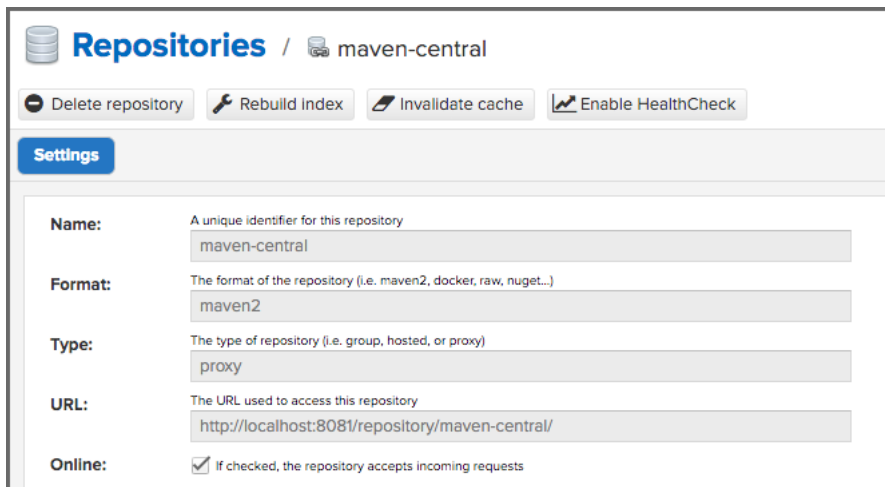
displays the repository health statistics from a previously run Repository Health Check, or a button to start the analysis

The *Create repository* button above the repository list triggers a dialog to select the *Recipe* for the new repository. The recipe combines the format and the type of repository into a single selection. Depending on your repository manager version and installed plugins, the list of available choices differs.

For example to create another release repository in *maven2* format, you would click on the row with the recipe *maven2 (hosted)* in the dialog. If you wanted to proxy a *maven2* repository, choose *maven 2 (proxy)*. On the other hand if you want to proxy a nuget repository, choose *nuget (proxy)*. With *maven2 (group)* you can create a repository group for *maven2* repositories.

After this selection, you are presented with the configuration view, that allows you to fill in the required parameters and some further configuration. The exact details on the view depend on the selected repository provider and are identical to the administration for updating the configuration of a repository documented in the following sections.

Once you have created a repository or repository group, it is available in the list for further configuration and management. Clicking on a specific row allows you to navigate to this repository specific administration section. An example for the *maven-central* repository is partially displayed in Figure 4.7.



The screenshot shows the Nexus 'Repositories' administration page for a repository named 'maven-central'. At the top, there are four action buttons: 'Delete repository', 'Rebuild index', 'Invalidate cache', and 'Enable HealthCheck'. Below these is a 'Settings' tab. The configuration form includes the following fields:

- Name:** A unique identifier for this repository. Value: maven-central
- Format:** The format of the repository (i.e. maven2, docker, raw, nuget...). Value: maven2
- Type:** The type of repository (i.e. group, hosted, or proxy). Value: proxy
- URL:** The URL used to access this repository. Value: http://localhost:8081/repository/maven-central/
- Online:** A checkbox labeled 'If checked, the repository accepts incoming requests'. It is currently checked.

Figure 4.7: Partial Repository Configuration for a Proxy Repository

The repository administration feature view has buttons to perform various actions on a repository. The buttons displayed depend on the repository format and type. The following buttons can be found:

Delete repository

The *Delete repository* button allows you to delete the repository and all related configuration and components, after confirming the operation in a dialog.

Invalidate cache

The *Invalidate cache* button invalidates the caches for this repository. The exact behavior depends on the repository type:

Proxy repositories

Invalidating the cache on a proxy repository clears the proxy cache such that any items cached as available will be checked again for any changes the next time they are requested. This also clears the negative cache for the proxy repository such that any items that were not found within the defined cache period will be checked again the next time they are requested.

Repository groups

Invalidating the cache of a repository group, clears the group cache such that any items fetched and held in the group cache, such as Maven metadata, will be cleared. This action also invalidates the caches of any proxy and group repositories that are members of this group.

Rebuild Index

The *Rebuild Index* button allows you to drop and recreate the search index for the proxy repository, synchronizing the contents with search index. This button is only available for proxy repositories.

The following properties can be viewed for all repositories and can not be edited after the initial creation of the repository.

Name

The *Name* is the identifier that will be used in the URL for access to the repository. For example, the proxy repository for the Central Repository has a name of `maven-central`. The *Name* must be unique in a given repository manager installation and is required.

Format

Format defines in what format the repository manager exposes the repository to external tools. Supported formats depend on the edition of the repository manager and the installed plugins. Examples are *maven2*, *nuget*, *raw*, *docker*, *npm* and others.

Type

The type of repository - *proxy*, *hosted* or *group*.

URL

It shows the user facing URL this means that Maven and other tools can access the repository directly at e.g., `http://localhost:8081/repository/maven-central`.

Online

The checkbox allows you set whether this repository is available to client side tools or not.

Beyond the generic fields used for any repository, a number of different fields are used and vary depending on the repository format and type. They are grouped under a number of specific headers that include configuration for the related aspects and include:

- Storage
- Hosted
- Proxy
- Negative Cache
- HTTP
- Maven 2
- NuGet
- and others

4.3.5.1 Storage

Every repository needs to have a *Blob store* configured to determine where components are stored. The drop-down allows you to select from all the configured blob stores. Documentation about creating blob stores can be found in [Section 4.3.1](#).

The *Strict Content Type Validation* allows you to activate a validation that checks the MIME type of all files published into a repository to conform to the allowed types for the specific repository format.

4.3.5.2 Hosted

A hosted repository includes configuration of a *Deployment policy* in the *Hosted* configuration section. Its setting controls how a hosted repository allows or disallows component deployment.

If the 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 component once and any attempt to deploy a component again will result in an error. The disabled redeploy is the default value, since most client tools assume components to be immutable and will not check a repository for changed components that have already been retrieved and cached locally.

If the policy is set to *Allow redeploy*, clients can deploy components to this repository and overwrite the same component in subsequent deployments.

4.3.5.3 Proxy

The configuration for proxy repositories in the *Proxy* section also contains the following parameters:

Remote Storage

A proxy repository on the other hand requires the configuration of the *Remote Storage*. It needs to be configured with the URL of the remote repository, that should 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 repository manager instead of simply proxying the group.

Use the Nexus truststore

This checkbox allows you to elect for the repository manager to manage the SSL certificate of the remote repository. It is only displayed - if the remote storage uses a HTTPS URL. The *View certificate* button triggers the display of the SSL certificate details in a dialog. The dialog allows you to add or remove the certificate from the certificate truststore maintained by the repository manager. Further details are documented in Section [5.9.1](#).

Blocked

Setting a repository to blocked causes the repository manager to no longer send outbound requests to the remote repository.

Auto blocking enabled

If *Auto blocking enabled* is set to true, the repository manager automatically blocks a proxy repository if the remote repository becomes unavailable. While a proxy repository is blocked, components will still be served to clients from a local cache, but the repository manager will not attempt to locate an component in a remote repository. The repository manager periodically retests the remote repository and unblocks it once it becomes available.

Maximum component age

When the proxy receives a request for a component, it does not request a new version from the remote repository until the existing component is older than *Maximum component age*.

Maximum metadata age

The repository manager retrieves metadata from the remote repository. It will only retrieve updates

to metadata after the *Maximum metadata age* has been exceeded. If the metadata is component metadata, it uses the longer of this value and *Maximum component age* before rechecking.

4.3.5.4 Negative Cache

Not found cache enabled/Not found cache TTL

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

4.3.5.5 HTTP

The *HTTP* configuration section allows you to configure the necessary details to access the remote repository, even if you have to provide authentication details in order to access it successfully or if you have to connect to it via a proxy server.

Note

This configuration is only necessary, if it is specific to this repository. Global HTTP proxy and authentication is documented in Section [4.2.5](#).

Authentication

This section allows you to select *Username* or *Windows NTLM* as *authentication type*. Subsequently you can provide the required *Username* and *Password* for plain authentication or *Username*, *Password*, *Windows NTLM hostname* and *Windows NTLM domain* for *Windows NTLM*-based authentication.

HTTP request settings

In the *HTTP Request Settings* you can change the properties of the HTTP requests to the remote repository. You can append a string to the user-agent HTTP header in the *User-agent customization* of the request and add parameters to the requests in *URL parameters*. Additionally you can set the timeout value for requests in seconds in *Connection timeout* and configure a number of *Connection retries*. The HTTP requests configured are applied to all requests made from the repository manager to the remote repository being proxied.

Some repository formats include configuration options, such as these formats:

- *Repository Connectors*, *Docker Registry API Support* and *Docker Index* for Docker repositories - Section 8.2, Section 8.3 and Section 8.4
- *Maven 2* for Maven repositories - Section 6.2
- *NuGet* for NuGet proxy repositories - Section 7.2
- *Bower* for Bower proxy repositories - Section 10.2

4.3.5.6 Repository Groups

The creation and configuration for a repository group differs a little from pure repositories. It allows you to manage the member repositories of a repository group. An example for a repository group using the *maven2* format is visible in Figure 4.8. In this figure you can see the contents of the *maven-public* group that is pre-configured in Nexus Repository Manager.

The screenshot shows the 'Settings' page for a repository group named 'maven-public'. At the top, there are buttons for 'Delete repository' and 'Invalidate cache'. The 'Settings' section includes fields for 'Name' (maven-public), 'Format' (maven2), 'Type' (group), and 'URL' (http://localhost:8081/repository/maven-public/). The 'Online' checkbox is checked. Below this is the 'Storage' section with a 'Blob store' dropdown set to 'default' and a checked 'Strict Content Type Validation' option. The 'Group' section shows 'Member repositories' with an 'Available' list (empty) and a 'Members' list containing 'maven-releases', 'maven-snapshots', and 'maven-central'. Navigation buttons (up, down, left, right) are between the lists. At the bottom are 'Save' and 'Discard' buttons.

Repositories / maven-public

Delete repository Invalidate cache

Settings

Name: A unique identifier for this repository
maven-public

Format: The format of the repository (i.e. maven2, docker, raw, nuget...)
maven2

Type: The type of repository (i.e. group, hosted, or proxy)
group

URL: The URL used to access this repository
http://localhost:8081/repository/maven-public/

Online: ☒ If checked, the repository accepts incoming requests

Storage

Blob store:
Blob store used to store asset contents
default

Strict Content Type Validation:
☒ Validate that all content uploaded to this repository is of a MIME type appropriate for the repository format

Group

Member repositories:
Select and order the repositories that are part of this group

Available

Filter

Members

maven-releases
maven-snapshots
maven-central

Save Discard

Figure 4.8: Repository Group Configuration

The *Format* and *Type* are determined by the selection of the provider in the creation dialog e.g., *maven2 (group)* for the *maven-public* as a *maven2* format repository group.

The *Name* is set during the creation and is fixed once the repository group is created.

The *Online* checkbox allows you set whether this repository group is available to client side tools or not.

The *Member repositories* selector allows you to add repositories to the repository group as well as remove them. The *Members* column includes all the repositories that constitute the group. The *Available* column includes all the repositories and repository groups that can potentially be added to the group.

Note that the order of the repositories listed in the *Member* section is important. When the repository manager searches for a component in a repository group, it will return the first match. To reorder a repository in this list, click and the drag the repositories and groups in the *Members* list or use the arrow buttons between the *Available* and *Members* list. These arrows can be used to add and remove repositories as well.

The order of repositories or other groups in a group can be used to influence the effective metadata that will be retrieved by Maven or other tools from a repository group. It is recommended practice to place hosted repositories higher in the list than proxy repositories. For proxy repositories, the repository manager needs to check the remote repository which will incur more overhead than a hosted repository lookup.

It is also recommended to place 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 *maven-central* higher in this list than a smaller, more focused repository is going to be better for performance, as the repository manager is not going to interrogate the smaller remote repository for as many missing components. These best practices are implemented in the default configuration.

4.3.6 Repository Management Example

The following sections detail some common steps of your repository management efforts on the example of a *maven2* repository.

4.3.6.1 Adding Repositories for Missing Dependencies

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

Tip

More details about client tool configuration for Maven repositories can be found in [Chapter 6](#).

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 publicly available repositories in its configuration. When you encounter such a project all you have to do is

- add this repository as a new *maven2* format, proxy repository
- and then add the new proxy repository to the *maven-public* group.

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

4.3.6.2 Adding a New Repository

Once you have established the URL and format of the remote repository you are ready to configure the repository. E.g. the JBoss.org releases repository contains your missing component. Click on the *Create repository* button in the *Repositories* feature view and click on *maven2 (proxy)* from the list in the dialog.

In the configuration dialog:

- Set *Name* to `jboss-releases`
- Set *Remote storage* to `https://repository.jboss.org/nexus/content/repositories/releases/`
- For a *maven2* format repository, confirm that the *Version policy* is set correctly to *Release*.
- Click on the *Create repository* button at the end of the form

The repository manager is now 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 with the same URL setting version policy to *Snapshot*.

4.3.6.3 Adding a Repository to a Group

Next you will need to add the new repository *jboss-releases* to the *maven-public* repository group. To do this, click on the row of the *maven-public* group in the *Repositories* feature view.

To add the new repository to the public group, find the repository in the *Available* list on the left, click on the repository you want to add and drag it to the right to the *Members* list. Once the repository is in that list, you can click and drag the repository within that list to alter the order in which the group will be searched for a matching component. Press the *Save* button to complete this configuration.

In the last few sections, you learned how to add new repositories to a build in order to download components 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 repository manager to add the two repositories to the public group. If all of the developers are configured to point to the public group, 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 component resolving across multiple repositories will be handled by the repository manager and therefore be much faster than client side resolution done by Maven each time.

4.4 Support Features

Nexus Repository Manager provides a number of features that allow you to ensure your server is configured correctly and provides you with tools to investigate details about the configuration. This information can be useful for troubleshooting and support activities.

All support features are available in the *Support* group of the *Administration* menu in the main menu section and include:

- [Analytics](#)
- [Logging and Log Viewer](#)
- [Metrics](#)
- [Support ZIP](#)

- [System Information](#)

4.4.1 Analytics

Available in Nexus Repository OSS and Nexus Repository Pro

The analytics integration allows Sonatype to gather data about of your repository manager usage, since it enables the collection of event data. It collects non-sensitive information about how you are using the repository manager and allows Sonatype to achieve a better understanding of usage overall and therefore drive product innovation following your needs.

The collected information is limited the primary interaction points between your environment and the repository manager. None of the request specific data (e.g., credentials or otherwise sensitive information) is ever captured.

Tip

The data is can be 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.

You can enable the event logging in the *Analytics* feature view available via *Analytics* menu item in the *Support* section of the *Administration* menu. Select the checkbox beside *Collect analytics events* and press the *Save* button.

You can choose to provide this data automatically to Sonatype by selecting the checkbox beside *Enable anonymized analytics submission to Sonatype*. 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* feature view available via the *Analytics* section of the *Administration* menu displayed in [Figure 4.9](#).

	Event type	Timestamp...	Sequence	Duration (...)	User	Attributes
+	Ext.Direct	142963717...	249	4954000	admin	name=rapture_State_get, action=rapture_Sta...
	name	rapture_State_get				
	action	rapture_State				
	type	POLL				
	success	true				
+	Ext.Direct	142963716...	248	5182000	admin	name=rapture_State_get, action=rapture_Sta...
+	Ext.Direct	142963716...	247	4290000	admin	name=rapture_State_get, action=rapture_Sta...
+	Ext.Direct	142963715...	246	5064000	admin	name=rapture_State_get, action=rapture_Sta...

Figure 4.9: List of Analytics Events

The list of events shows the *Event type*, the *Timestamp*, the *Sequence* number and the *Duration* 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 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 repository manager instance.

Tip

More information about capabilities can be found in [Section 4.2.2](#).

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.

**Important**

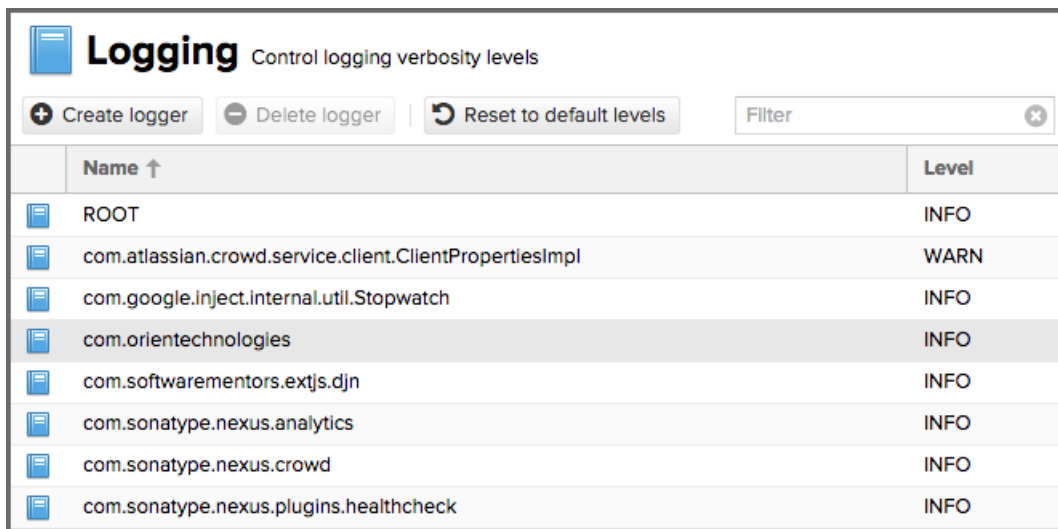
Sonatype values your input greatly and hopes you will activate the analytics feature and the automatic submission to allow us to ensure ongoing development is well aligned with your needs. In addition, Sonatype appreciates any further direct contact and feedback in person and look forward to hearing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

4.4.2 Logging and Log Viewer

Available in Nexus Repository OSS and Nexus Repository Pro

You can configure the level of logging for the repository manager and all plugins as well as inspect the current log using the user interface with the *Logging* and the *Log Viewer* feature views.

Access the *Logging* feature view displayed in Figure 4.10 with the *Logging* menu item in the *Support* section in the *Administration* main menu.



Name ↑	Level
ROOT	INFO
com.atlassian.crowd.service.client.ClientPropertiesImpl	WARN
com.google.inject.internal.util.Stopwatch	INFO
com.orienttechnologies	INFO
com.softwarementors.extjs.djn	INFO
com.sonatype.nexus.analytics	INFO
com.sonatype.nexus.crowd	INFO
com.sonatype.nexus.plugins.healthcheck	INFO

Figure 4.10: The Logging Feature View for Configuring Loggers

The *Logging* feature view 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. Press the *Update* button to apply the change.

The *Create logger* button can be used to create new loggers. You will need to know the *Logger name* you want to configure. Typically this corresponds to the Java package name used in the source code. Depending on your needs you can inspect the source of Nexus Repository Manager 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.

If you select a row in the list of loggers, you can delete the highlighted logger by pressing the *Delete logger* button above the list. This only applies to previously created custom loggers. To disable a default configured logger, set it to `OFF`.

**Important**

When upgrading the repository manager, keep in mind that some loggers change between versions, so if you rely on specific loggers, you might have to reconfigure them.

The *Reset to default levels* button allows you to remove all your custom loggers and get back to the setup shipped with a fresh install of the repository manager.

The loggers configured in the user interface are persisted into `$data-dir/etc/logback-overrides.xml` and override any logging levels configured in the main log file `logback-nexus.xml` as well as the other `logback-*` files. If you need to edit a logging level in those files, 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 logging is in general. If set to `DEBUG`, logging will be very verbose, printing all log messages including debugging statements. If set to `ERROR`, logging will be far less verbose, only printing out a log statement if the repository manager 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 trigger the change to the logging immediately.

Once logging is configured as desired, you can inspect the impact of your configuration in the *Log Viewer*

feature view. It allows you to copy the log from the server to your machine by pressing the *Download* button. The *Create 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 4.11.

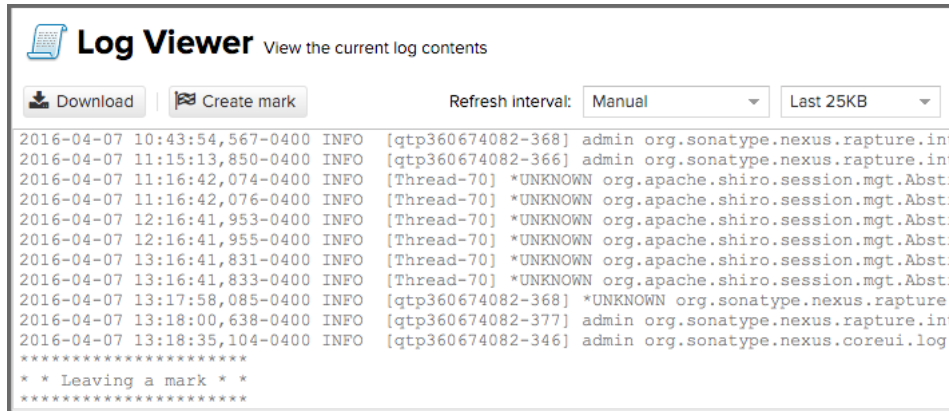


Figure 4.11: Viewing the Log with an Inserted Mark

The *Refresh interval* configuration on the right on the top of the view allows you to configure the timing for the refresh as well as the size of the log displayed. A manual refresh can be triggered with the general refresh button in the main toolbar.

4.4.3 Metrics

Available in Nexus Repository OSS and Nexus Repository Pro

The *Metrics* feature view is available in the *Support* section of the *Administration* main menu. It provides insight to characteristics of the Java virtual machine JVM running the repository manager and is displayed in Figure 4.12.

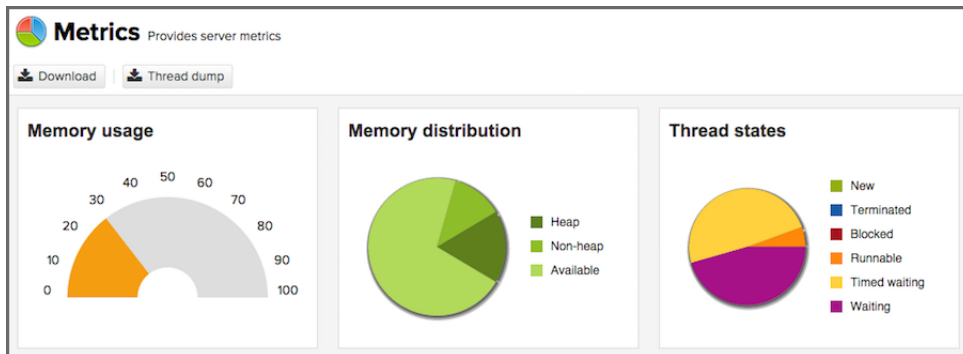


Figure 4.12: JVM Metrics

The *Memory usage*, *Memory distribution* and *Thread states* charts provide some simple visualizations. The *Download* button allows you to retrieve a large number of properties from the JVM and download them in a JSON-formatted text file. Pressing the *Thread dump* button triggers the creation of a thread dump of the JVM and a download of the resulting text file.

4.4.4 Support ZIP

Available in Nexus Repository OSS and Nexus Repository Pro

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

You can include the *System information report* as available in the *System Information* tab, a *JVM thread-dump* of the JVM currently running the repository, your general *Configuration files* as well as your *Security configuration files*, the *Log files* and *System and component metrics* with network and request-related information and *JMX information*.

The *Options* allow you to limit the size of the included files as well as the overall ZIP archive file size. Pressing the *Create support ZIP* button gathers all files, creates the archive in `$data-dir/downloads` and opens a dialog to download the file to your workstation. This dialog shows the *Name*, *Size* and exact *Path* of the support ZIP file.

4.4.5 System Information

Available in Nexus Repository OSS and Nexus Repository Pro

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

Nexus

details about the versions of the repository manager and the installed plugins, 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 the repository manager

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 or use the *Download* button to retrieve a JSON-formatted text file.

Chapter 5

Security

Available in Nexus Repository OSS and Nexus Repository Pro

5.1 Introduction

Nexus Repository Manager Pro and Nexus Repository Manager OSS use role-based access control that gives administrators very fine-grained control over user rights to

- read from a repository or a subset of repositories,
- administer the repository manager or specific parts of the configuration,
- access specific parts of the user interface,
- deploy to repositories or even just specific sections of a repository.

The default configuration ships with roles and users with a standard set of permissions. As your security requirements evolve, you will likely need to customize security settings to create protected repositories for multiple departments or development groups. Nexus Repository Manager provides a security model that can adapt to any scenario.

Tip

The default administrator user give you full control and uses the username *admin* and the password *admin123*.

This chapter covers all aspects of security of the repository manager including

- user account and access right management related to user interface as well as to component access documented in Section 5.3, Section 5.4 and Section 5.5
- selection of security backend systems documented Section 5.2 including the built-in system as well as LDAP and others
- management of SSL certificates from remote repositories, SMTP and LDAP servers documented in Section 5.9

Security-related configuration can be performed with the feature views available via the *Security* section of the *Administration* main menu. Many of the features shown in this section are only available to users with the necessary privileges to access them.

The role-based access control system is backed by different authentication and authorizations systems as documented in Section 5.2 and designed around the following security concepts:

Privileges

[Privileges](#) are rights to read, update, create, or manage resources and perform operations related to the user interface as well as the components managed by the repository manager in the various repositories. The repository manager ships with a set of core privileges that cannot be modified.

Roles

Privileges can be grouped into collections called [roles](#) to make it easier to define privileges common to certain classes of users. For example, administrative 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.

Users

[Users](#) can be assigned one or more roles, and model the individuals who will be logging into the user interface and read, deploy, or manage repositories as well as connect from client tools such as Apache Maven.

5.2 Realms

Available in Nexus Repository OSS and Nexus Repository Pro

The feature view for security realms administration displayed in Figure 5.1 allows you to activate and prioritize security realms used for authentication and authorization by adding them to the *Active* list on the right and placing them higher or lower on the list. It can be accessed via the *Realms* menu item in the *Security* submenu in the *Administration* main menu.

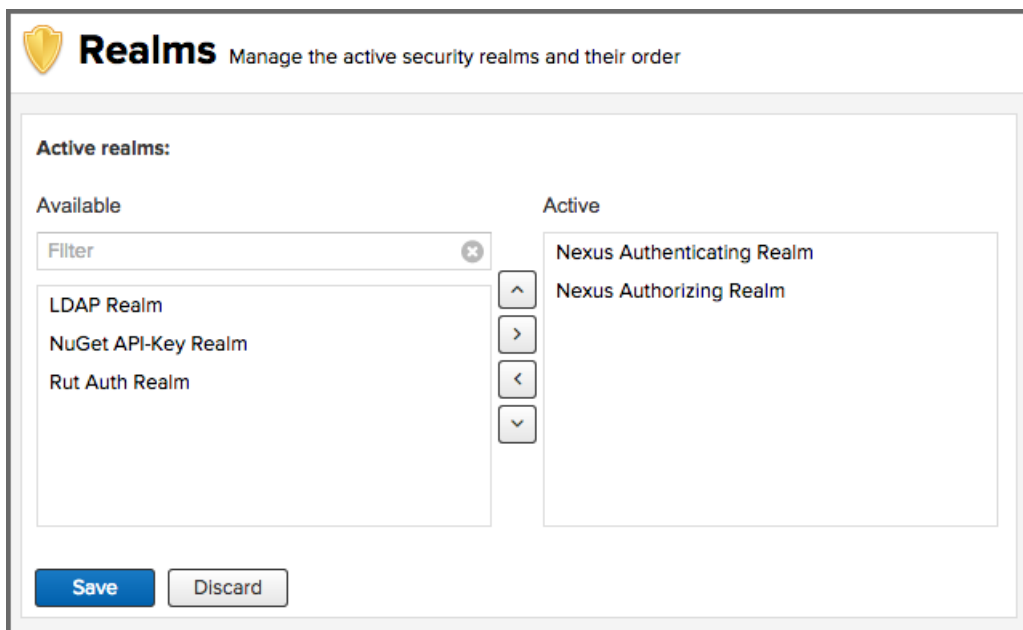


Figure 5.1: Security Realms Administration

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

Nexus Authenticating Realm and Nexus Authorizing Realm

These are the built-in realms used by default. They allow the repository manager to manage security setup without additional external systems.

LDAP Realm

This realm identifies external storage in an LDAP system including e.g., Microsoft ActiveDirectory, ApacheDS, OpenLDAP with details documented in Section [5.7](#).

Rut Auth Realm

This realm uses an external authentication in any system with the user authorization passed to the repository manager in a HTTP header field with details documented in Section [5.8](#).

NuGet API-Key Realm

This realm is required for deployments to nuget-format repositories as documented in Chapter [7](#).



Warning

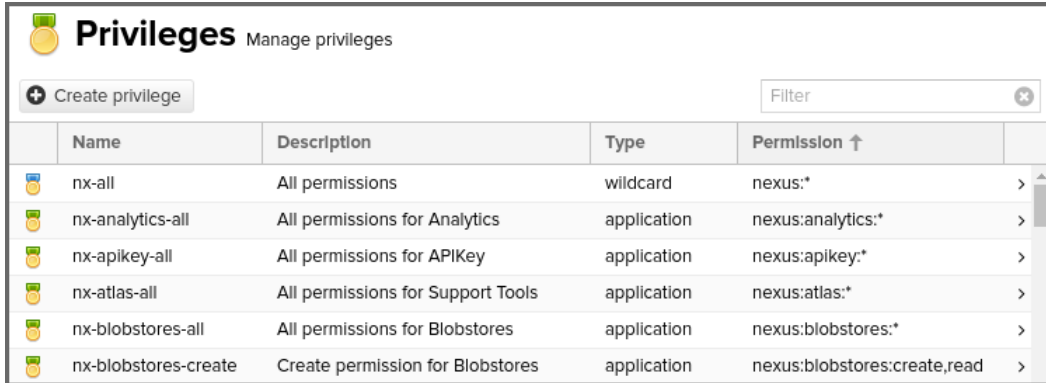
Removing all realms from the *Active* section prevents access to the repository manager for any user including any administrative access and has to be avoided.

5.3 Privileges

Available in Nexus Repository OSS and Nexus Repository Pro

Privileges control access to specific functionality of the repository manager and can be grouped as a role and assigned to a specific users.

To access *Privileges* go to the *Security* section, where it's listed as a sub-section. An extensive list of privileges is already built in the repository manager and is partially depicted in Figure [5.2](#). This feature allows you inspect existing privileges and create custom privileges.









	Name	Description	Type	Permission ↑	
	nx-all	All permissions	wildcard	nexus:*	>
	nx-analytics-all	All permissions for Analytics	application	nexus:analytics:*	>
	nx-apikey-all	All permissions for APIKey	application	nexus:apikey:*	>
	nx-atlas-all	All permissions for Support Tools	application	nexus:atlas:*	>
	nx-blobstores-all	All permissions for Blobstores	application	nexus:blobstores:*	>
	nx-blobstores-create	Create permission for Blobstores	application	nexus:blobstores:create,read	>

Figure 5.2: Partial List of Security Privileges

The list of privileges displays an icon for the privilege Type as the first column, followed by:

Name

the internal identifier for the privilege

Description

a human readable description of the purpose of the privilege

Type

the aspect of the repository manager to which this privilege applies

Permission

the internal permission definition as used by the embedded security framework

Further details are available after pressing on a specific row in the detail view.

Click the *Create privilege* button to view a list of privilege types, as seen in Figure 5.3.

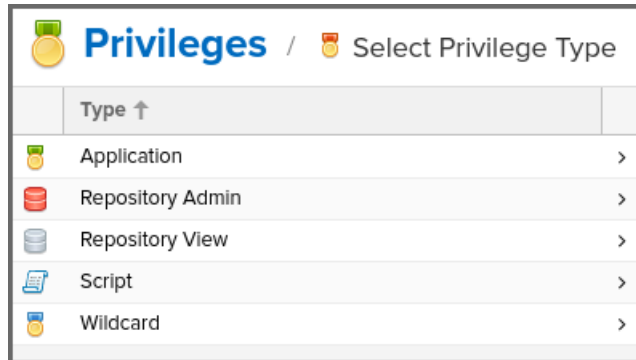


Figure 5.3: Choosing Privilege Types

Select the privilege type corresponding to the area of the repository manager you wish to grant permissions. The privilege types are as follows:

Application

These are privileges related to a defined aspect of the repository manager.

Repository Admin

These are privileges related to the administration and configuration of a specific repository.

Repository View

These are privileges controlling access to the content of a specific repository.

Script

These are privileges related to the execution and management of scripts as documented in Chapter 14.

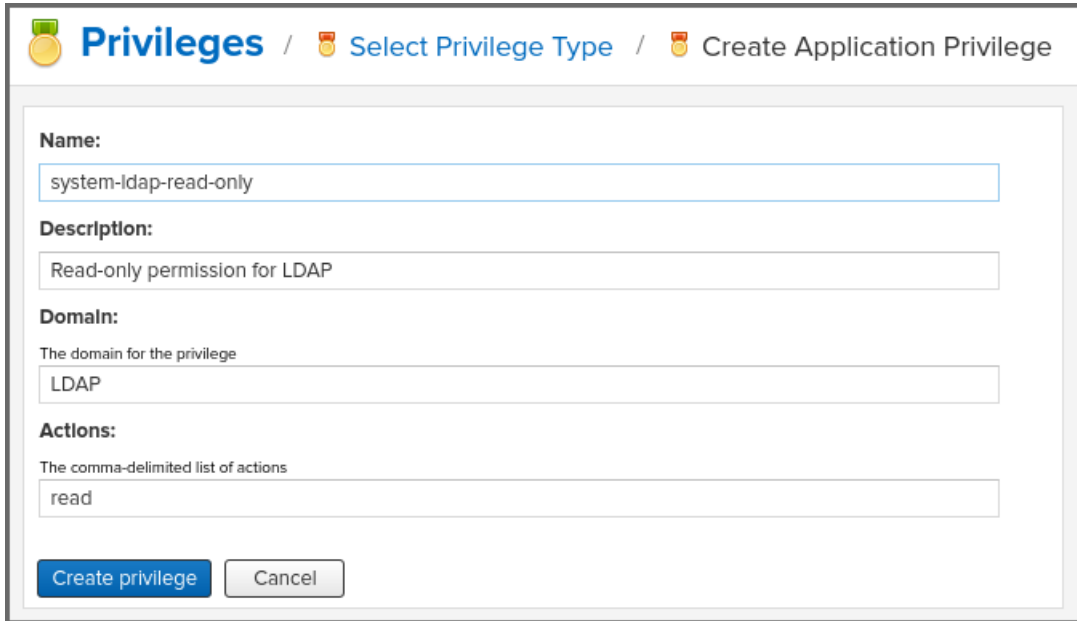
Wildcard

These are privileges that use patterns to group other privileges.

In all *Privilege Types*, above, the variables assigned to a role are defined as *Actions*. *Actions* can either be exclusive or a combination of add, browse, create, delete, edit, read and * (all) storage functions.

To save a new custom privilege click the *Create privilege* button. The privilege can be found listed among the default privileges on the main *Privileges* screen. You can use the *Filter* input box to find a specific privilege.

In the following example, an *Application* privilege type is created.



The screenshot shows a web interface for creating a privilege. The breadcrumb trail at the top is: **Privileges** / **Select Privilege Type** / **Create Application Privilege**. The form contains the following fields:

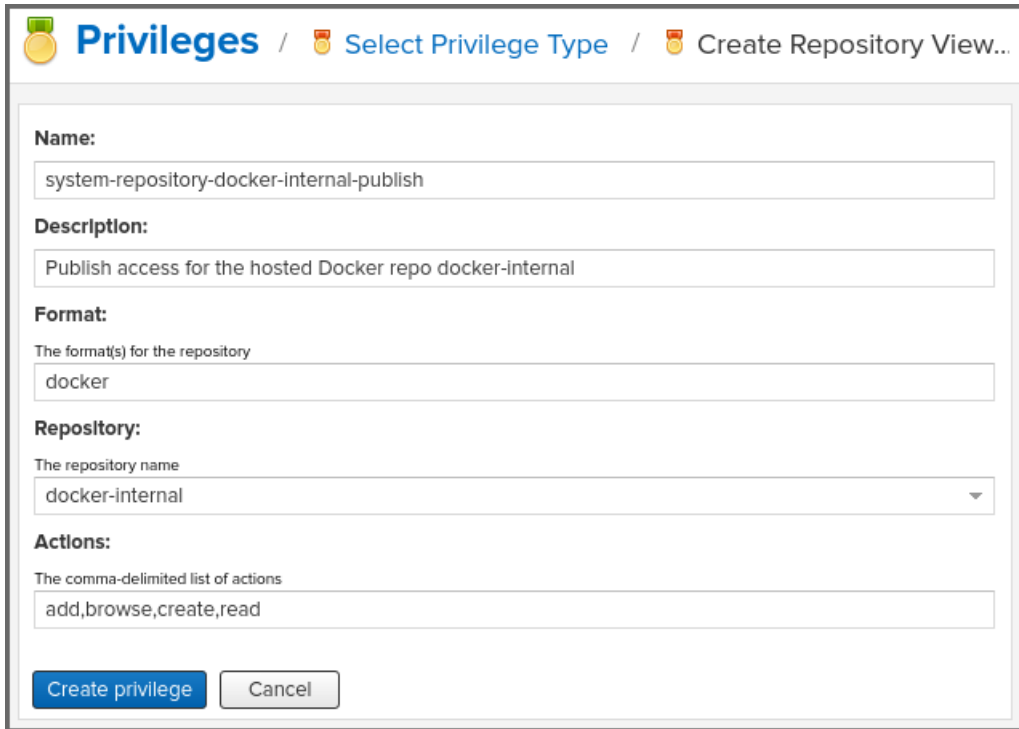
- Name:** A text input field containing "system-ldap-read-only".
- Description:** A text input field containing "Read-only permission for LDAP".
- Domain:** A text input field containing "LDAP". Above the field is the label "The domain for the privilege".
- Actions:** A text input field containing "read". Above the field is the label "The comma-delimited list of actions".

At the bottom of the form are two buttons: "Create privilege" (in blue) and "Cancel" (in grey).

Figure 5.4: Creating an Application Privilege

The form provides *Name*, *Description*, *Domain*, and *Actions*. In Figure 5.4 the form is completed for a privilege only that allows read access to the LDAP administration. If assigned this privilege, a user is able to view LDAP administration configuration but not edit it, create a new LDAP configuration, nor delete any existing LDAP configurations.

In another example, a *Repository View* privilege type is created.



The screenshot shows the 'Privileges' management interface in Nexus. The breadcrumb trail at the top indicates the path: 'Privileges / Select Privilege Type / Create Repository View...'. The form contains the following fields:

- Name:** A text input field containing 'system-repository-docker-internal-publish'.
- Description:** A text input field containing 'Publish access for the hosted Docker repo docker-internal'.
- Format:** A text input field with the label 'The format(s) for the repository' containing 'docker'.
- Repository:** A dropdown menu with the label 'The repository name' showing 'docker-internal'.
- Actions:** A text input field with the label 'The comma-delimited list of actions' containing 'add,browse,create,read'.

At the bottom of the form are two buttons: 'Create privilege' (in blue) and 'Cancel' (in grey).

Figure 5.5: Creating a Repository View Privilege

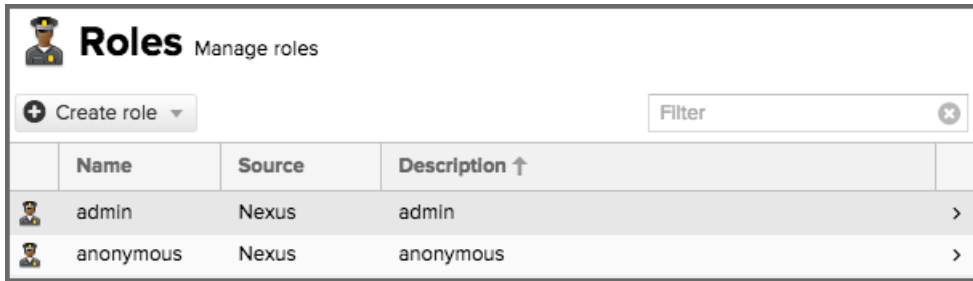
The form provides *Name*, *Description*, *Format*, *Repository*, and *Actions*. In Figure 5.5 the form is completed for a privilege granting sufficient access to publish images to a specific hosted repository. A user with this privilege can view and read the contents of the repository as well as publish new images to it, but not delete images.

5.4 Roles

Available in Nexus Repository OSS and Nexus Repository Pro

Roles aggregate privileges into a related context and can, in turn, be grouped to create more complex roles.

The repository manager ships with a predefined *admin* as well as an *anonymous* role. These can be inspected in the *Roles* feature view accessible via the *Roles* item in the *Security* section of the *Administration* main menu. A simple example is shown in Figure 5.6. The list displays the *Name* and *Description* of the role as well as the *Source*, which displays whether the role is internal (*Nexus*) or a mapping to an external source like LDAP.






	Name	Source	Description ↑	
	admin	Nexus	admin	>
	anonymous	Nexus	anonymous	>

Figure 5.6: Viewing the List of Defined Roles

To create a new role, click on the *Create role* button, select *Nexus Role* and fill out the Role creation feature view shown in Figure 5.7.

 **Roles** /  Create Role

Role ID:

Role name:

Role description:

Privileges:

Available

✕

- nx-all
- nx-analytics-all
- nx-apikey-all
- nx-atlas-all
- nx-blobstores-all
- nx-blobstores-create

>

<

Given

Roles:

Available

✕

- nx-admin

>

<

Contained

- nx-anonymous

Create role

Cancel

Figure 5.7: Creating a New Role

When creating a new role, you will need to supply a *Role ID* and a *Name* and optionally a *Description*. Roles are comprised of other roles and individual privileges. To assign a role or privilege to a role, drag and drop the desired privileges from the *Available* list to the *Given* list under the *Privileges* header. You can use the *Filter* input to narrow down the list of displayed privileges and the arrow buttons to add or remove privileges.

The same functionality is available under the *Roles* header to select among the *Available* roles and add them to the list of *Contained* roles.

Finally press the *Create Role* button to get the role created.

An existing role can be inspected and edited by clicking on the row in the list. This role-specific view allows you to delete the role with the *Delete role* button. The built-in roles are managed by the repository manager and cannot be edited or deleted. The *Settings* section displays the same section as the creation view as displayed in Figure 5.7.

In addition to creating an internal role, the *Create role* button allows you to create an *External role mapping* to an external authorization system configured in the repository manager such as *LDAP*. This is something you would do, if you want to grant every member of an externally managed group (such as an *LDAP* group) a number of privileges and roles in the repository manager.

For example, assume that you have a group in *LDAP* named `scm` and you want to make sure that everyone in the `scm` group has administrative privileges.

Select *External Role Mapping* and *LDAP* to see a list of roles managed by that external realm in a dialog. Pick the desired `scm` group and confirm by pressing *Create mapping*.

Tip

For faster access or if you cannot see your group name, you can also type in a portion or the whole name of the group and it will limit the dropdown to the selected text.

Once the external role has been selected, creates a linked role. You can then assign other roles and privileges to this new externally mapped role like you would do for any other role.

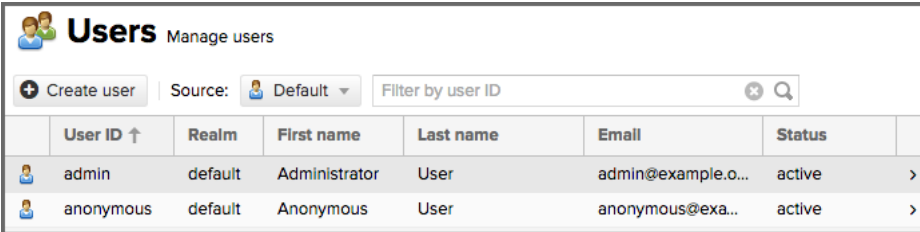
Any user that is part of the `scm` group in *LDAP*, receives all the privileges defined in the created role allowing you to adapt your generic role in *LDAP* to the repository manager-specific use cases you want these users to be allowed to perform.

5.5 Users

Available in Nexus Repository OSS and Nexus Repository Pro

The repository manager ships with two users: *admin* and *anonymous*. The *admin* user has all privileges and the *anonymous* user has read-only privileges. The default password for the *admin* user is *admin123*.

The *Users* feature view displayed in Figure 5.8 can be accessed via the *Users* item in the *Security* section of the *Administration* menu. The list shows the users *User ID*, *First Name*, *Last Name* and *Email* as well as what security *Realm* is used and if the accounts *Status* is *active* or *disabled*.

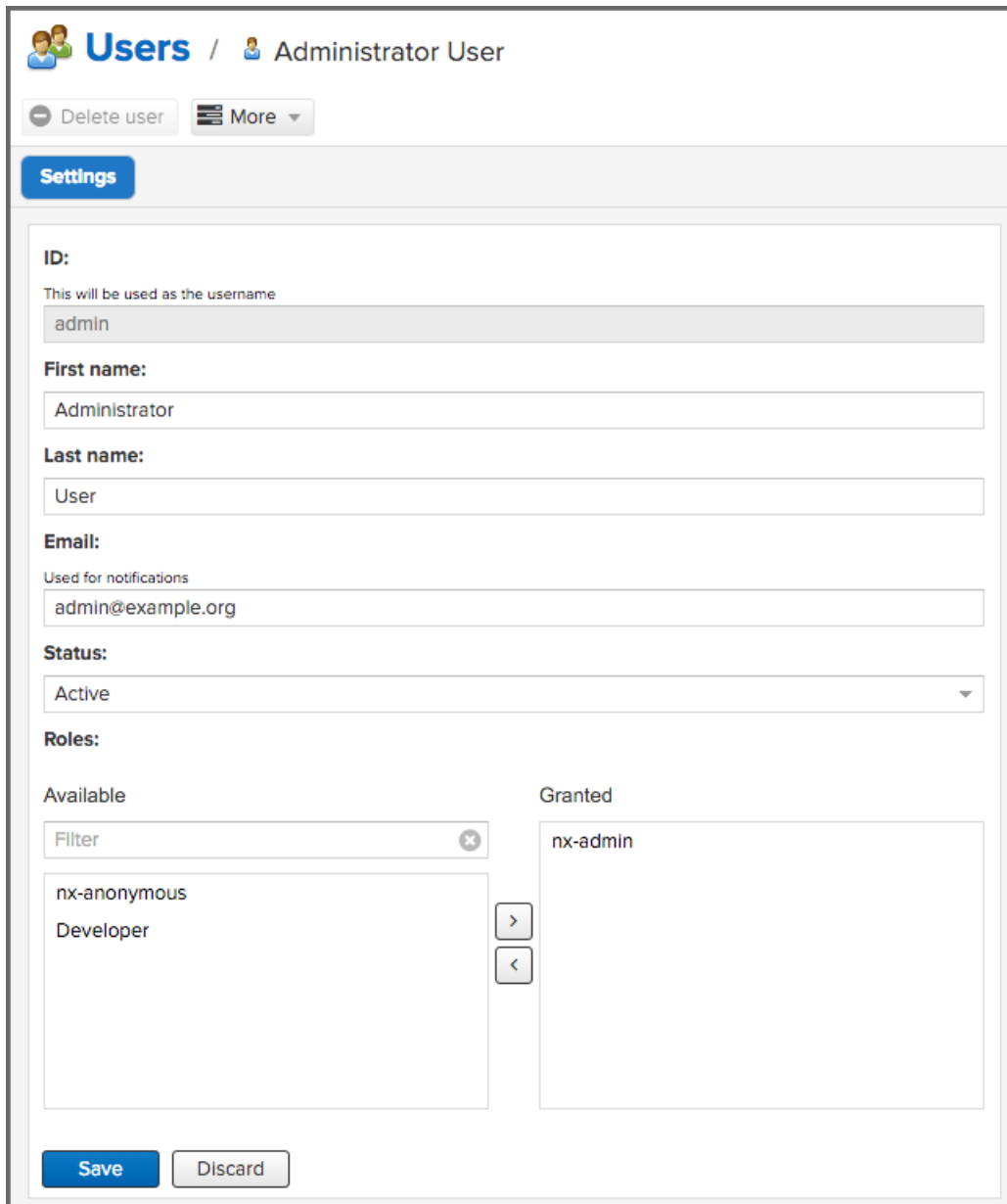


User ID ↑	Realm	First name	Last name	Email	Status
admin	default	Administrator	User	admin@example.o...	active
anonymous	default	Anonymous	User	anonymous@exa...	active

Figure 5.8: Feature View with List of Users

Clicking on a user in the list or clicking on the *Create user* button displays the details view to edit or create the account shown in Figure 5.9. The *ID* can be defined upon initial creation and remains fixed thereafter. In addition you can specify the users *First Name*, *Last Name* and *Email* address. The *Status* allows you to set an account to be *Disabled* or *Active*.

The *Roles* control allows you to add and remove defined [roles](#) to the user and therefore control the [privileges](#) assigned to the user. A user can be assigned one or more roles that in turn can include references to other roles or to individual privileges.



The screenshot shows the 'Users' management interface in Nexus. The breadcrumb is 'Users / Administrator User'. There are buttons for 'Delete user' and 'More'. The 'Settings' tab is selected. The form contains the following fields:

- ID:** This will be used as the username. Value: admin
- First name:** Value: Administrator
- Last name:** Value: User
- Email:** Used for notifications. Value: admin@example.org
- Status:** Active (dropdown)
- Roles:**
 - Available:** Filter (input), nx-anonymous, Developer
 - Granted:** nx-admin
 - Navigation buttons: > and <

At the bottom are 'Save' and 'Discard' buttons.

Figure 5.9: Creating or Editing a User

The *More* button in the allows you to select the *Change Password* item in the drop down. The password

can be changed in a dialog, provided the user is managed by the built-in security realm.

**Important**

Ensure to change the password of the *admin* user to avoid security issues. Alternatively create other users with administrative rights and disable the default *admin* user.

5.6 Anonymous Access

Available in Nexus Repository OSS and Nexus Repository Pro

By default, the user interface as well as the repositories and the contained components are available to unauthenticated users for read access. The *Anonymous* feature view is available via the *Anonymous* item in the *Security* section of the *Administration* main menu and shown in Figure 5.10.

The privileges available to these users are controlled by the roles assigned to the *anonymous* user from the *NexusAuthorizingRole*. By changing the privileges assigned to this user in the [Users feature view](#).

Figure 5.10: Configuring Anonymous Access

If you want to disable unauthenticated access to the repository manager entirely, you can uncheck the

Allow *anonymous users to access the server* checkbox. The *Username* and *Realm* controls allow you to change the details for the anonymous user. E.g. you might have a *guest* account defined in your LDAP system and desire to use that user for anonymous access.

5.7 LDAP

Available in Nexus Repository OSS and Nexus Repository Pro

5.7.1 Introduction

Nexus Repository Manager Pro and Nexus Repository Manager OSS can use the Lightweight Directory Access Protocol (LDAP) for authentication via external systems providing LDAP support such as Microsoft Exchange/Active Directory, [OpenLDAP](#), [ApacheDS](#) and others.

Configuring LDAP can be achieved in a few simple steps:

- Enable LDAP Authentication Realm
- Create LDAP server configuration with connections and user/group mapping details
- Create external role mappings to adapt LDAP roles to repository manager specific usage

In addition to handling authentication, the repository manager can be configured to map roles to LDAP user groups. If a user is a member of a LDAP group that matches the ID of a role, the repository manager grants that user the matching role. In addition to this highly configurable user and group mapping capability, the repository manager can augment LDAP group membership with specific user-role mapping.

The repository manager can cache authentication information and supports multiple LDAP servers and user/group mappings. Connection details to the LDAP server and the user/group mappings as well as specific account logins can be tested directly from the user interface.

All these features allow you to adapt to any specific LDAP usage scenario and take advantage of the central authentication set up across your organization in all your repository managers.

5.7.2 Enabling the LDAP Authentication Realm

As seen in Figure 5.1, activate your *LDAP Realm* by following these steps:

- Navigate to the [Realms](#) administration section
- Select the *LDAP Realm* and add it to the list of *Active* realms on the right
- Ensure that the *LDAP Realm* is located beneath the *Nexus Authenticating Realm* in the list
- Press *Save*

Best practice is to leave the *Nexus Authenticating Realm* and the *Nexus Authorizing Realm* activated so that the repository manager can be used by *anonymous*, *admin* and other users configured in this realm even with LDAP authentication offline or unavailable. Any user account not found in the *Nexus Authenticating Realm*, will be passed through to LDAP authentication.

5.7.3 LDAP Connection and Authentication

The *LDAP* feature view displayed in Figure 5.11 is available via the *LDAP* item in the *Security* section of the *Administration* main menu.

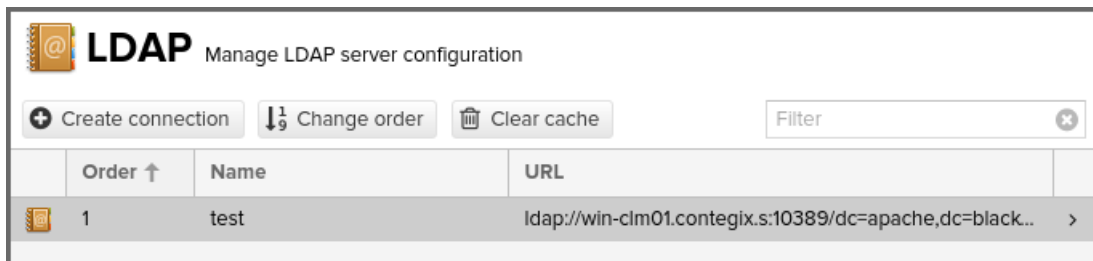


Figure 5.11: LDAP Feature View

The *Order* determines in which order the repository manager connects to the LDAP servers when authenticating a user. The *Name* and *URL* columns identify the configuration and clicking on a individual row provides access to the *Connection* and *User and group* configuration.

The *Create connection* button can be used to create a new LDAP server configuration. Multiple configurations can be created and are accessible in the list.

The *Change order* button can be used to change the order in which the repository manager queries the LDAP servers in a pop up dialog.

Successful authentications are cached so that subsequent logins do not require a new query to the LDAP server each time. The *Clear cache* button can be used to remove these cached authentications.

Tip

Contact the administrator of your LDAP server to figure out the correct parameters, as they vary between different LDAP server vendors, versions and individual configurations performed by the administrators.

The following parameters allow you to create an LDAP connection:

Name

Enter a unique name for the new configuration.

LDAP server address

Enter *Protocol*, *Hostname*, and *Port* of your LDAP server.

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 server.

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 further qualifies the connection to the LDAP server. The search base usually corresponds to the domain name of an organization. For example, the search base could be `dc=example,dc=com`.

You can configure one of four authentication methods to be used when connecting to the LDAP Server with the *Authentication method* drop-down.

Simple Authentication

Simple authentication consists of a *Username* and *Password*. 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

The anonymous authentication uses the server address and search base without further authentication.

Digest-MD5

This is an improvement on the CRAM-MD5 authentication method. For more information, see [RFC-2831](#).

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 [RFC-2829](#) and [RFC-2251](#).

SASL Realm

The Simple Authentication and Security Layer (SASL) realm used to connect to the LDAP server. It is only available if the authentication method is Digest-MD5 or CRAM-MD5.

Username or DN

Username or Distinguished Name DN of an LDAP user with read access to all necessary users and groups. It is used to connect to the LDAP server.

Password

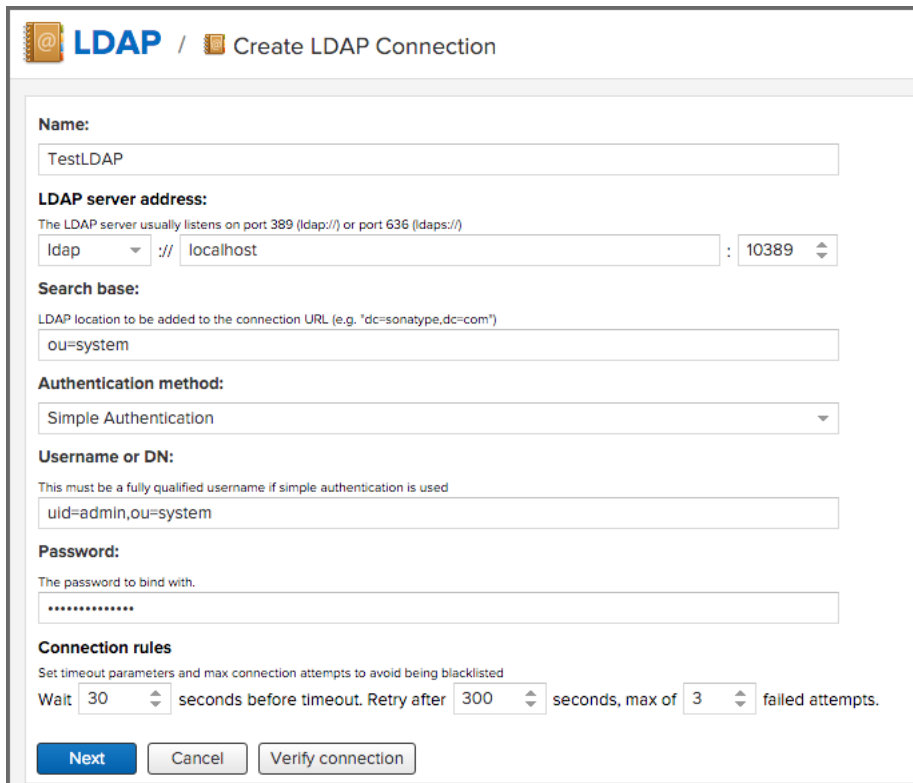
Password for the Username or DN configured above.

To test your connection to the external LDAP server, click *Verify connection*. A successful connection is confirmed with notification pop up.

The connection details can be further refined by configuring timeout period, retry period and number of connection attempts in *Connection rules*.

Click *Next* to proceed to configure [user and group mappings](#) for the LDAP configuration.

Figure [5.12](#) shows a LDAP connection configuration for the repository manager configured to connect to an LDAP server running on localhost port 10389 using the search base of `ou=system`.



The screenshot shows a web-based configuration form titled "LDAP / Create LDAP Connection". The form contains several sections with input fields and dropdown menus:

- Name:** A text input field containing "TestLDAP".
- LDAP server address:** A section with a dropdown menu set to "ldap", a text input field containing "://localhost", and a port spinner set to "10389". A note below states: "The LDAP server usually listens on port 389 (ldap://) or port 636 (ldaps://)".
- Search base:** A text input field containing "ou=system". A note below states: "LDAP location to be added to the connection URL (e.g. 'dc=sonatype,dc=com')".
- Authentication method:** A dropdown menu set to "Simple Authentication".
- Username or DN:** A text input field containing "uid=admin,ou=system". A note above states: "This must be a fully qualified username if simple authentication is used".
- Password:** A password input field with masked characters "*****". A note above states: "The password to bind with.".
- Connection rules:** A section with a note: "Set timeout parameters and max connection attempts to avoid being blacklisted". It includes three spinner controls: "Wait" set to 30, "seconds before timeout. Retry after" set to 300, and "seconds, max of" set to 3, followed by the text "failed attempts."

At the bottom of the form are three buttons: "Next" (highlighted in blue), "Cancel", and "Verify connection".

Figure 5.12: Create LDAP Connection

5.7.4 User and Group Mapping

The LDAP connection panel contains a section to manage *User and group* mappings. This configuration is the next step after you configure and verify the *LDAP Connection*. It is a separate panel called *Choose Users and Groups*.

This panel provides a *Configuration template* drop-down, shown in Figure 5.13. Based on your template selection the rest of the field inputs will adjust to the appropriate user and group template requirements. These templates are suggestions for typical configurations used on servers such as *Active Directory*, *Generic Ldap Server*, *Posix with Dynamic Groups* and *Posix with Static Groups*. The values are suggestions only and have to be adjusted to your specific needs based on your LDAP server configuration.

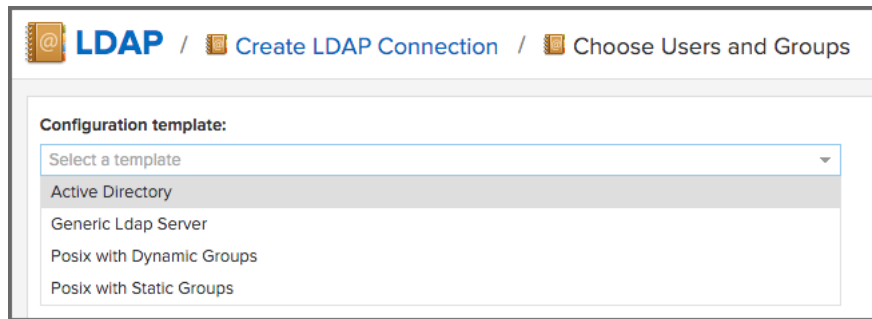


Figure 5.13: Configuration Template for Users and Groups

The following parameters allow you to configure your user and group elements with the repository manager:

Base DN

Corresponds to the collection of distinguished names used as the base for user entries. This DN is relative to the Search Base. 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 use a value of `ou=users`.

User subtree

Check the box if *True*. Uncheck if *False*. 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 is a standard object class defined in [RFC-2798](#), and specifies the object class for users. Common values are `inetOrgPerson`, `person`, `user` or `posixAccount`.

User filter

This allows you to configure a filter to limit the search for user records. It can be used as a performance improvement.

User ID attribute

This is the attribute of the object class specified above, that supplies the identifier for the user from the LDAP server. The repository manager uses this attribute as the *User ID* value.

Real name attribute

This is the attribute of the Object class that supplies the real name of the user. The repository

manager uses this attribute when it needs to display the real name of a user similar to usage of the internal *First name* and *Last name* attributes.

Email attribute

This is the attribute of the Object class that supplies the email address of the user. The repository manager uses this attribute for the *Email* attribute of the user. It is used for email notifications of the user.

Password attribute

It can be used to configure the Object class, which supplies the password ("userPassword"). If this field is blank the user will be authenticated against a bind with the LDAP server. The password attribute is optional. 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. The repository manager uses this attribute when it is authenticating a user against an LDAP server.

An automatically checked box will allow you to *Map LDAP groups as roles*. With the configuration any LDAP group configured for a specific users is used to query the roles in the repository manager. Identical names trigger the user to be granted the privileges of the roles.

Groups in LDAP systems are configured to be dynamic or static. A dynamic group is a list of groups to which users belong. A static group contains a list of users. Select *Dynamic Groups* or *Static Groups* from the *Group type* drop-down to proceed with the appropriate configuration.

The screenshot shows a configuration form for 'Static Groups'. It includes fields for 'Group type' (a dropdown menu), 'Group base DN' (with a descriptive note and a text input), 'Group subtree' (with a checkbox), 'Group object class' (with a descriptive note and a text input), 'Group ID attribute' (with a text input), 'Group member attribute' (with a descriptive note and a text input), and 'Group member format' (with a descriptive note and a text input). At the bottom are four buttons: 'Save', 'Discard', 'Verify user mapping', and 'Verify login'.

Group type:
Static Groups
Group base DN:
The base location in the LDAP that groups are found. This is relative to the search base (e.g. ou=Group).
cn=builtin
Group subtree:
<input type="checkbox"/> Are groups located in structures below the group base DN.
Group object class:
LDAP class for group objects (e.g. posixGroup)
group
Group ID attribute:
sAMAccountName
Group member attribute:
LDAP attribute containing the usernames for the group.
member
Group member format:
The format of user ID stored in the group member attribute (e.g. "uid=\${username},ou=people,o=sonatype")
\${dn}
<input type="button" value="Save"/> <input type="button" value="Discard"/> <input type="button" value="Verify user mapping"/> <input type="button" value="Verify login"/>

Figure 5.14: Static Group Element Mapping

Static groups with an example displayed in Figure 5.14, are configured with the following parameters:

Group base DN

This field is similar to the *Base DN* field described for User Element Mapping, but applies to groups instead of users. For example, 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, but configures groups instead of users. If all groups are defined under the entry defined in Base DN, set the field to false. If a group can be defined in a tree of organizational units under the Base DN, set the field to true.

Group object class

This value in this field is a standard object class defined in [RFC-2307](#). The class is simply a collection of references to unique entries in an LDAP directory and can be used to associate user entries with a group. Examples are `groupOfUniqueNames`, `posixGroup` or custom values.

Group ID attribute

Specifies the attribute of the object class that specifies the group identifier. If the value of this field corresponds to the ID of a role, members of this group will have the corresponding privileges.

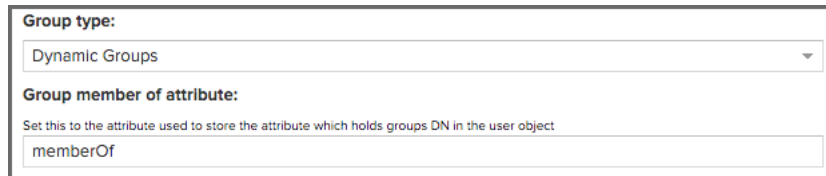
Group member attribute

Specifies the attribute of the object class which specifies a member of a group. An example value is *uniqueMember*.

Group member format

This field captures the format of the Group Member Attribute, and is used by the repository manager to extract a username from this attribute. An example values is `${dn}`.

If your installation does not use static groups, you can configure the LDAP connection to refer to an attribute on the user entry to derive group membership. To do this, select *Dynamic Groups* in the *Group type* drop down.



The screenshot shows a configuration form with two main sections. The first section, labeled 'Group type:', contains a dropdown menu with 'Dynamic Groups' selected. The second section, labeled 'Group member of attribute:', includes a descriptive text 'Set this to the attribute used to store the attribute which holds groups DN in the user object' and a text input field containing the value 'memberOf'.

Figure 5.15: Dynamic Group Element Mapping

Dynamic groups are configured via the *Group member of attribute* parameter. The repository manager inspects this attribute of the user entry to get a list of groups of which the user is a member. In this configuration, seen in Figure 5.15, a user entry would have an attribute that would contain the name of a group, such as *memberOf*.

Once you have configured the user and group settings on the *Choose Users and Groups* form, you can check the correctness of your user mapping by pressing the *Verify user mapping* button. A successful mapping will result in the retrieval of a list of user records, which will be shown in the *User Mapping Test Result* dialog.

The repository manager provides you with the ability to test a user login directly. To test a user login, go to the *Choose Users and Groups* page after all appropriate field inputs of the form are filled. Scroll to the bottom and click the *Verify login* button.

The *Verify login* button can be used to check if authentication and user/group mappings work as expected for a specific user account besides the global account used for the LDAP configuration.

After your LDAP the successful configuration of your connection and user and group mappings, you can proceed to configure external role mappings. This allows you to define the repository manager specific security for a LDAP group. More details are available in Section 5.4.

5.8 Authentication via Remote User Token

Available in Nexus Repository OSS and Nexus Repository Pro

The repository manager allows integration with external security systems that can pass along authentication of a user via the `Remote_User` HTTP header field for all requests - Remote User Token *Rut* authentication. This typically affects all web application usage in a web browser.

These 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 the repository manager. 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. the repository manager 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 4.2.2 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 *Active* realms in the *Realms* configuration described in Section 5.2.

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 installation with the internal authorization scheme enabled, a value of *admin* would grant the user the access rights in the user interface as the *admin* user.

A seamless integration can be set up for users if the external security system is exposed via LDAP and configured in the repository manager 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.

5.9 Configuring SSL

Using Secure Socket Layer (SSL) communication with the repository manager is an important security feature and a recommended best practice. Secure communication can be inbound or outbound.

Outbound client communication may include integration with

- a remote proxy repository over HTTPS - documented in [Section 4.3](#)
- SSL/TLS secured servers - e.g. for SMTP/email integration documented in [Section 4.2.3](#)
- LDAP servers configured to use LDAPS
- specialized authentication realms such as the Crowd realm.

Inbound client communication includes

- web browser HTTPS access to the user interface,
- tool access to repository content,
- and manual or scripted usage of the REST APIs.

5.9.1 Outbound SSL - Trusting SSL Certificates of Remote Repositories

Available in Nexus Repository OSS and Nexus Repository Pro

When the SSL certificate of a remote proxy repository is not trusted, the repository may be automatically blocked or outbound requests fail with a message similar to *PKIX path building failed*.

The *Proxy* configuration for each proxy repository documented in [Section 4.3.5](#) includes a section titled *Use the Nexus truststore*. It allows you to manage the SSL certificate of the remote repository and solves these problems. It is only displayed, if the remote storage uses a HTTPS URL.

The *View certificate* button triggers the display of the *SSL Certificate Details* dialog. An example is shown in [Figure 5.16](#).

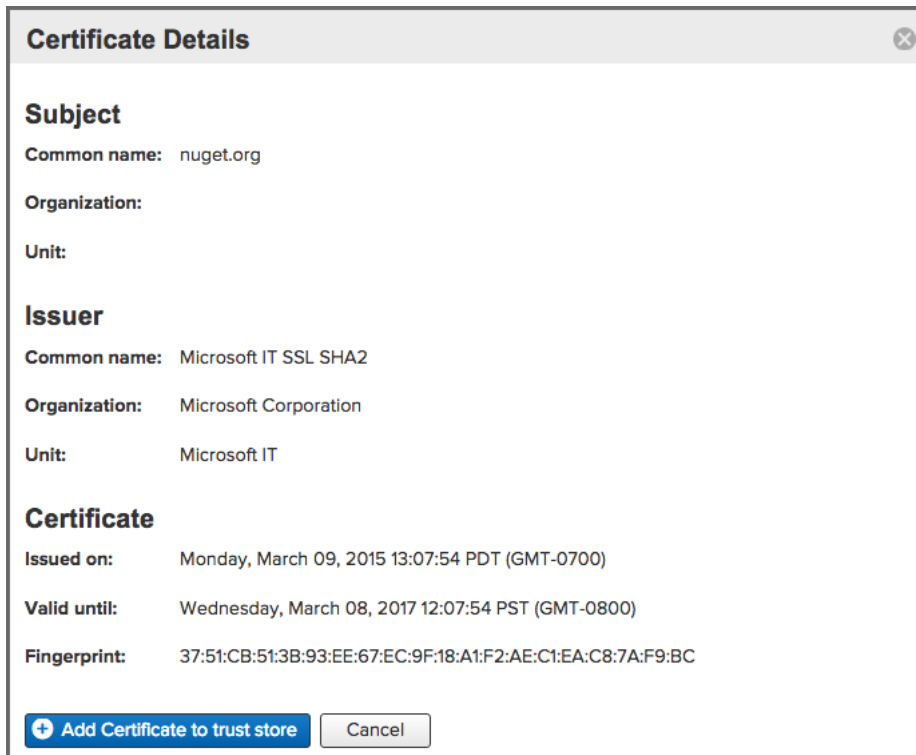


Figure 5.16: Certificate Details Dialog to Add an SSL to the Nexus Truststore

Use the *Certificate Details* dialog when the remote certificate is not issued by a well-known public certificate authority included in the default Java trust store. This specifically also includes usage of self-signed certificates used in your organization. To confirm trust of the remote certificate, click the *Add certificate to truststore* button in the dialog. This feature is analogous to going to the [SSL Certificates](#) user interface and using the *Load certificate* button found there as described in Section 5.9.2. If the certificate is already added, the button can undo this operation and will read *Remove certificate from trust store*.

The checkbox labelled *Use certificates stored in Nexus to connect to external systems* is used to confirm that the repository manager should consult the internal truststore as well as the JVM truststore when confirming trust of the remote repository certificate. Without adding the certificate to the private truststore and enabling the checkbox, the repository will not trust the remote.

The default JVM truststore of the JVM installation used to run the repository manager and the private truststores are merged. The result of this merge is used to decide about the trust of the remote server. The default Java truststore already contains public certificate authority trust certificates. If the remote

certificate is signed by one of these authorities, then explicitly trusting the remote certificate will not be needed.

**Warning**

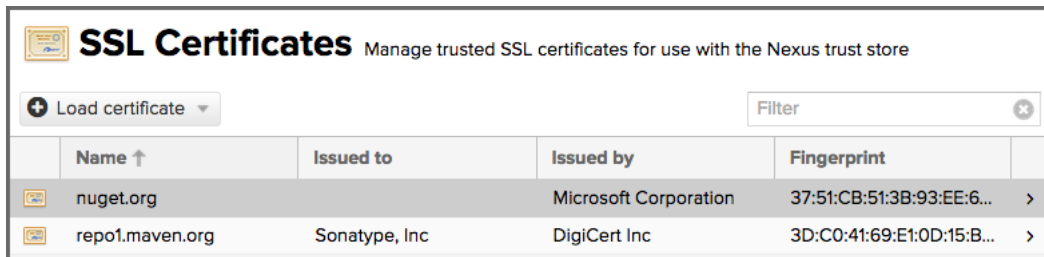
When removing a remote trusted certificate from the truststore, a repository manager restart is required before a repository may become untrusted.

5.9.2 Outbound SSL - Trusting SSL Certificates Globally

Available in Nexus Repository OSS and Nexus Repository Pro

The repository manager allows you to manage trust of all remote SSL certificates in a centralized user interface. Use this interface when you wish to examine all the currently trusted certificates for remote repositories, or manage certificates from secure remotes that are not repositories.

Access [the feature view for SSL Certificates administration](#) by selecting the *SSL Certificates* menu items in the *Security* submenu in the *Administration* main menu.





	Name ↑	Issued to	Issued by	Fingerprint	
	nuget.org		Microsoft Corporation	37:51:CB:51:3B:93:EE:6...	>
	repo1.maven.org	Sonatype, Inc	DigiCert Inc	3D:C0:41:69:E1:0D:15:B...	>

Figure 5.17: SSL Certificates Administration

The list shows any certificates that are already trusted. Clicking on an individual row allows you to inspect the certificate. This detail view shows further information about the certificate including *Subject*, *Issuer* and *Certificate* details. The *Delete certificate* button allows you to remove a certificate from the truststore.

The button *Load certificate* above the list of certificates can be used to add a new certificate to the truststore by loading it directly from a server or using a PEM file representing the certificate.

The common approach is to choose *Load from server* and enter the full `https://` URL of the remote site, e.g, `https://repo1.maven.org`. The repository manager will connect using HTTPS and use the HTTP proxy server settings if applicable. When the remote is not accessible using `https://`, only enter the host name or IP address, optionally followed by colon and the port number. For example: `example.com:8443`. In this case the repository manager will attempt a direct SSL socket connection to the remote host at the specified port. This allows you to load certificates from SMTP or LDAP servers, if you use the correct port.

Alternatively you can choose the *Paste PEM* option to configure trust of a remote certificate. Copy and paste the Base64 encoded X.509 DER certificate to trust. This text must be enclosed between lines containing `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`.

Typically this file is supplied to you by the certificate owner. An example method to get the encoded X.509 certificate into a file on the command line using `keytool` is:

```
keytool -printcert -rfc -sslserver repo1.maven.org > repo1.pem
```

The resulting `repo1.pem` file contains the encoded certificate text that you can cut and paste into the dialog in the user interface. An example of inserting such a certificate is shown in Figure 5.18.



Figure 5.18: Providing a Certificate in PEM Format

If the repository manager can successfully retrieve the remote certificate or decode the pasted certificate, the details will be shown allowing you to confirm details as shown in Figure 5.19. Please review the

displayed information carefully before clicking *Add Certificate* to establish the truststore addition.

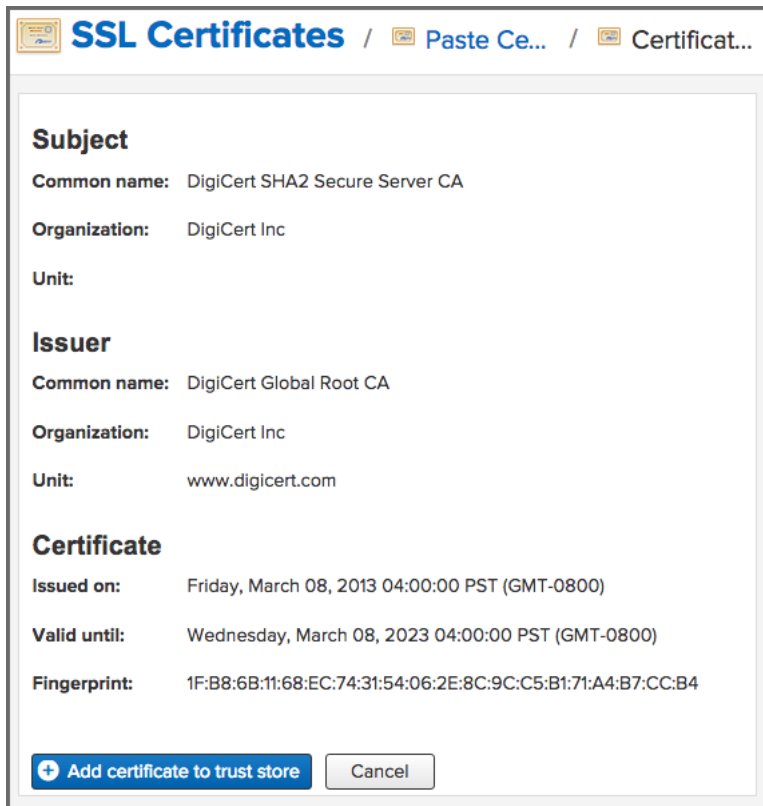


Figure 5.19: Certificate Details Displayed after Successful Retrieval or Parsing

In some organizations, all of the remote sites are accessed through a globally configured proxy server which rewrites every SSL certificate. This single proxy server is acting as a private certificate authority. In this case, you can **follow special instructions for trusting the proxy server root certificate**, which can greatly simplify your certificate management duties.

5.9.3 Outbound SSL - Trusting SSL Certificates Using Keytool

Available in Nexus Repository OSS and Nexus Repository Pro

Managing trusted SSL certificates from the command line using [keytool](#) and system properties is an alternative and more complex option than using the SSL certificate management features of the repository manager.

Before you begin the process of trusting a certificate from the command line you will need:

- a basic understanding of [SSL certificate technology and how the Java VM implements this feature](#)
- command line access to the host operating system and the *keytool* program
- network access to the remote SSL server you want to trust from the host running the repository manager. This must include any HTTP proxy server connection details.

If you are connecting to servers that have certificates which are not signed by a public CA, you will need to complete these steps:

1. Copy the default JVM truststore file (`$JAVA_HOME/jre/lib/security/cacerts`) to a specific location for editing.
2. Import additional trusted certificates into the copied truststore file.
3. Configure JSSE system properties for the repository manager process so that the custom truststore is consulted instead of the default file.

Some common commands to manually trust remote certificates can be found in our [SSL Certificate Guide](#).

After you have imported your trusted certificates into a truststore file, you can add the JVM parameters configuring the truststore file location and password as separate configuration lines into the file `etc/system.properties`.

```
javax.net.ssl.trustStore=<truststore>
javax.net.ssl.trustStorePassword=<truststore_password>
```

Once you have added the properties shown above, restart the repository manager and attempt to proxy a remote repository using the imported certificate. The repository manager will automatically register the certificates in the truststore file as trusted.

5.9.4 Inbound SSL - Configuring to Serve Content via HTTPS

Available in Nexus Repository OSS and Nexus Repository Pro

Providing access to the user interface and content via HTTPS is a best practice.

You have two options:

- Using a separate reverse proxy server in front of the repository manager to manage HTTPS
- Configure the repository manager itself to serve HTTPS directly

5.9.4.1 Using A Reverse Proxy Server

A common approach is to access the repository manager through a dedicated server which answers HTTPS requests on behalf of the repository manager - these servers are called reverse proxies or SSL/TLS terminators. Subsequently requests are forwarded to the repository manager via HTTP and responses received via HTTP are then sent back to the requestor via HTTPS.

There are a few advantages to using these which can be discussed with your networking team. For example, the repository manager can be upgraded/installed without the need to work with a custom JVM keystore. The reverse proxy could already be in place for other systems in your network. Common reverse proxy choices are Apache httpd, nginx, Eclipse Jetty or even dedicated hardware appliances. All of them can be configured to serve SSL content, and there is a large amount of reference material available online.

Serving SSL Directly The second approach is to use the Eclipse Jetty instance that is distributed with the repository manager to accept HTTPS connections.

5.9.4.2 How to Enable the HTTPS Connector

1. Create a Java keystore file at `$install-dir/etc/ssl/keystore.jks` which contains the Jetty SSL certificate to use. Instructions are available on the [Eclipse Jetty documentation site](#).
2. Edit `$install-dir/etc/org.sonatype.nexus.cfg`. Add a property on a new line `application-port-ssl=8443`. Change 8443 to be your preferred port on which to expose the HTTPS connector.

3. Edit `$install-dir/etc/org.sonatype.nexus.cfg`. Change the `nexus-args` property comma delimited value to include `${karaf.etc}/jetty-https.xml`. Save the file.
4. Restart Nexus. Verify HTTPS connections can be established.
5. Update the Base URL to use `https` in your repository manager configuration using the [Base URL capability](#).

Tip

This configuration process is available [as a video demonstration](#).

5.9.4.3 How to Redirect All Plain HTTP Requests to HTTPS

Some organizations need to remind their users that Nexus should only be used over HTTPS - redirecting HTTP requests to HTTPS can help.

1. Follow all the steps under [How to Enable the HTTPS Connector](#). Make sure the `nexus-args` property value still includes the reference to `${karaf.etc}/jetty-http.xml`
2. Edit `$install-dir/etc/org.sonatype.nexus.cfg`. Change the `nexus-args` property comma delimited value to include `${karaf.etc}/jetty-http-redirect-to-https.xml`. Save the file.
3. Restart Nexus. Verify all plain HTTP requests get redirected to the equivalent HTTPS url.

Tip

Redirecting HTTP requests is not recommended because it introduces implied security and creates increased network latency. Clients which send Basic Authorization headers preemptively may unintentionally expose credentials in plain text.

5.9.4.4 How to Disable the HTTP Connector

1. Edit `$install-dir/etc/org.sonatype.nexus.cfg`. Change the `nexus-args` property comma delimited value to not include `${karaf.etc}/jetty-http.xml`. Save the file.
 2. Restart Nexus. Verify plain HTTP requests are no longer serviced.
-

Chapter 6

Maven Repositories with Apache Maven and Other Tools

Available in Nexus Repository OSS and Nexus Repository Pro

6.1 Introduction

Historically Nexus Repository Manager started as a repository manager supporting the Maven repository format and it continues to include excellent support for users of Apache Maven , Apache Ant/Ivy, Eclipse Aether, Gradle and others.

This chapter explains the default configuration included in Nexus Repository Manager Pro and Nexus Repository Manager OSS, instructions for creating further Maven repositories as well as searching and browsing the repositories. Build tool configuration for Apache Maven, Apache Ant, Gradle and others tools follow. The configuration examples take advantage of the repository manager merging many repositories and exposing them via a repository group.

6.2 Maven Repository Format

Apache Maven created the most widely used repository format in the Java development ecosystem with the release of Apache Maven 2. It is used by all newer versions of Apache Maven and many other tools including Apache Ivy, Gradle, sbt, Eclipse Aether and Leiningen. Further information about the format can be found in Section 1.3.

The format is used by many publicly available repositories. The **Central Repository** is the largest repository of components aimed at Java/JVM-based development and beyond and is used the Maven repository format for release components of numerous open source projects. It is configured as a proxy repository by default in Apache Maven and widely used in other tools.

In addition to the generic repository management features documented in Section 4.3, specifics of the Maven repository format can be configured for each repository in the *Maven 2* section.

Version policy

Release

A Maven repository can be configured to be suitable for release components with the *Release* version policy. The Central Repository uses a release version policy.

Snapshot

Continuous development is typically performed with snapshot versions supported by the *Snapshot* version policy. These version values have to end with *-SNAPSHOT* in the POM file. This allows repeated uploads where the actual number used is composed of a date/timestamp and an enumerator and the retrieval can still use the *-SNAPSHOT* version string. The repository manager and client tools manage the metadata files that manage this translation from the snapshot version to the timestamp value.

Mixed

The *Mixed* version policy allows you to support both approaches within one repository.

Layout policy

The Maven repository format defines a directory structure as well as a naming convention for the files within the structure. Apache Maven follows these conventions. Other build tools, such as sbt, and custom tools have historically created usages that use the directory structure less strictly, violating the file naming conventions. Components based on these tools' different conventions have been published to public repositories, such as the Central Repository. These tools rely on these changed conventions.

Permissive

You can configure a layout policy of *Permissive* to allow assets in the repository that violate the default format.

Strict

The default value of *Strict* requires publishing and accessing tools to follow the Apache Maven conventions. This is the preferred setting if you are using Apache Maven, Eclipse Aether, and other strictly compatible tools.

Strict Content Type Validation

Maven repositories can be configured to validate any new components to see if the MIME-type corresponds to the content of the file by enabling this setting. Any files with a mismatch are rejected.

6.3 Proxying Maven Repositories

A default installation of Nexus Repository Manager includes a proxy repository configured to access the Central Repository via HTTPS using the URL `https://repo1.maven.org/maven2/`. To reduce duplicate downloads and improve download speeds for your developers and CI servers, you should proxy all other remote repositories you access as proxy repositories as well.

To proxy a Maven repository, you simply create a new repository using the recipe *maven2 (proxy)* as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Define URL for *Remote storage* e.g. `https://repo1.maven.org/maven2/`
- Select a *Blob store* for *Storage*

This creates a repository using the a *Release* version policy and a *Strict* layout policy. Both can be configured as appropriate for the remote repository.

If the remote repository contains a mixture of release and snapshot versions, you have to set the version policy to *Mixed*.

Usage of the repository with build tools such as sbt, potentially requires the layout policy to be set to *Permissive*.

6.4 Hosting Maven Repositories

A hosted Maven repository can be used to deploy your own as well as third-party components. A default installation of Nexus Repository Manager includes a two hosted Maven repositories. The *maven-releases* repository uses a release version policy and the *maven-snapshots* repository uses a snapshot version policy.

To create another hosted Maven repository, add a new repository with the recipe *maven2 (hosted)* as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for *Storage*

6.5 Grouping Maven Repositories

A repository group is the recommended way to expose all your Maven repositories from the repository manager 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 as well as other repository groups with one URL for tool configuration. This is possible for Maven repositories by creating a new repository with the *maven2 (group)* recipe as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for *Storage*
- Add Maven repositories to the *Members* list in the desired order

A typical, useful example is the *maven-public* group that is configured by default. It aggregates the *maven-central* proxy repository with the *maven-releases* and *maven-snapshots* hosted repositories. Using the *URL* of the repository group gives you access to the packages in all three repositories with one URL. Any new component added as well as any new repositories added to the group will automatically be available.

6.6 Browsing and Searching Maven Repositories

You can browse Maven repositories in the user interface inspecting the components and assets and their details as documented in [Section 3.4](#).

Components can be searched in the user interface as described in [Section 3.3](#). A search finds all components and assets that are currently stored in the repository manager, either because they have been deployed to a hosted repository or they have been proxied from an upstream repository and cached in the repository manager.

TIP

You can change the default column order in the search and browse user interfaces to the familiar order of *Group* (groupId), *Name* (artifactId) and *Version*. Simply drag the *Group* column from the middle to the left using the header. This setting will be persisted as your preference in your web browser.

6.7 Configuring Apache Maven

To use repository manager with **Apache Maven**, configure Maven to check the repository manager 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 [Listing: Configuring Maven to Use a Single Repository Group](#).

Listing: Configuring Maven to Use a Single Repository Group

```
<settings>
  <mirrors>
    <mirror>
      <!--This sends everything else to /public -->
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://localhost:8081/repository/maven-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 [Listing: Configuring Maven to Use a Single Repository Group](#) a single profile called `nexus` is defined. 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 repository group. This repository group can, therefore, contain release as well as snapshot components 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.

Deployment to a repository is configured in the `pom.xml` for the respective project in the `distributionManagement` section. Using the default repositories of the repository manager:

```
<project>
...
<distributionManagement>
  <repository>
    <id>nexus</id>
    <name>Releases</name>
    <url>http://localhost:8081/repository/maven-releases</url>
  </repository>
  <snapshotRepository>
    <id>nexus</id>
    <name>Snapshot</name>
    <url>http://localhost:8081/repository/maven-snapshots</url>
  </snapshotRepository>
</distributionManagement>
...
```

The credentials used for the deployment are looked from a `server` section in a users `settings.xml` using the `nexus` value used in the `id` fields:

```
<settings>
....
  <servers>
    <server>
      <id>nexus</id>
      <username>admin</username>
      <password>admin123</password>
    </server>
  </servers>
```

Full example projects can be found in the `maven` folder of the [documentation examples project](#) in the `nexus-3.x` branch. A full build of the `simple-project`, including downloading the declared dependencies and uploading the build output to the repository manager can be invoked with `mvn clean deploy`.

6.8 Configuring 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 repository manager running on `localhost` is shown in [Listing: Minimal Ivy Configuration in an Ant file](#).

Listing: Minimal Ivy Configuration in an Ant file

```
<ivysettings>
  <settings defaultResolver="nexus"/>
  <property name="nexus-public"
    value="http://localhost:8081/repository/maven-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 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 [documentation examples project](#) in the `nexus-3.x` branch. A full build of the `simple-project`, including downloading the declared dependencies and uploading the build output to the repository manager can be invoked with

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

6.9 Configuring 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` files 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 repository manager running on `localhost` is shown in

[Listing: Minimal Aether Configuration in an Ant file.](#)

Listing: Minimal Aether Configuration in an Ant file

```
<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/repository/maven- ↵
    public/" mirrorOf="*" />
  ...
</project>
```

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

To deploy build outputs to a 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 [documentation examples project](#) in the `nexus-3.x` branch. A full build of the `simple-project`, including downloading the declared dependencies and uploading the build output to the repository manager can be invoked with

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

6.10 Configuring 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 [Listing: Gradle Repositories Configuration](#) has to be declared.

Listing: Gradle Repositories Configuration

```
repositories {
  maven {
    url "http://localhost:8081/repository/maven-public/"
```

```
}  
}
```

These minimal settings allow Gradle to download the declared dependencies.

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

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

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

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

Full example projects can be found in the `gradle` folder of the [documentation book examples project](#) in the `nexus-3.x` branch. A full build of the `simple-project`, including downloading the declared dependencies and uploading the build output to the repository manager can be invoked with

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

6.11 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 [Listing: SBT Resolvers Configuration](#) has to be declared.

Listing: SBT Resolvers Configuration

```
resolvers += "Nexus" at "http://localhost:8081/repository/maven-public/"
```

These minimal settings allow sbt to download the declared dependencies.

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

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

The `publishTo` configuration:

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

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

6.12 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 [Listing:](#)

[Leiningen Configuration](#) has to be declared.

Listing: Leiningen Configuration

```
:mirrors {  
  "central" {:name "Nexus"  
             :url "http://localhost:8081/repository/maven- ←  
                 public/"  
             :repo-manager true}  
  #"clojars" {:name "Nexus"  
             :url "http://localhost:8081/repository/maven- ←  
                 public/"  
             :repo-manager true}  
}
```

These minimal settings allow Leiningen to download the declared dependencies.

To deploy build outputs to a repository with the `deploy` command, the target repositories have to be added 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 `public` repository group used in `mirrors`.

```
:deploy-repositories [  
  ["snapshots" "http://localhost:8081/repository/maven-snapshots"]  
  ["releases" "http://localhost:8081/repository/maven-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 7

.NET Package Repositories with NuGet

Available in Nexus Repository OSS and Nexus Repository Pro

7.1 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 and 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.

Nexus Repository Manager Pro and Nexus Repository Manager OSS support the NuGet repository format for hosted and proxy repositories as well as exposing them to the client-side tools as a repository group and has related repositories preconfigured.

Nexus Repository Manager and NuGet allow 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 using a repository manager when working in the .NET architecture.

To share a library or tool with NuGet, you create a NuGet package and store it in the NuGet repository on the repository manager. 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.

7.2 NuGet Repository Format

The NuGet repository format uses OData queries for communication between the client and the repository. These queries include metadata information about available packages and other data.

When the repository manager receives queries from the nuget client, it passes these queries on to the remote repositories, configured as proxy repository, if necessary.

To avoid sending identical queries to the remote repository, the repository manager caches the queries and will rely on previously stored metadata if the same query is received again before the cache expires.

The *NuGet* section is included in for NuGet Proxy repositories to allow configuration of this caching. The parameters *Query cache size* and *Query cache age* can be used to configure the size of this cache in terms of how many queries are cached as well as the rate at which queries expire and are subsequently re-run.

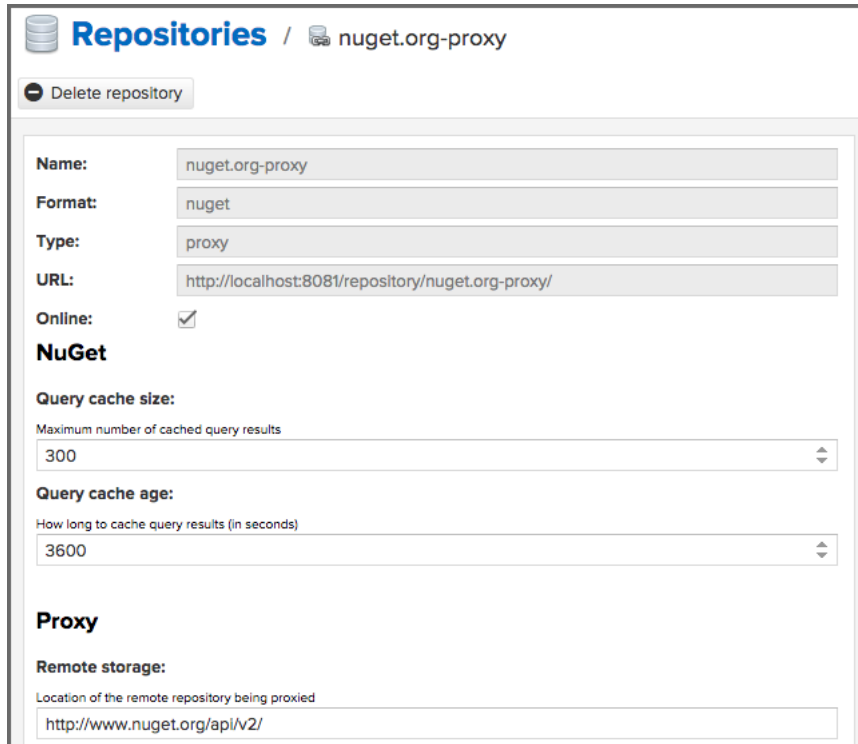
7.3 NuGet Proxy Repositories

The NuGet Gallery is the common 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 the repository manager. If you use other external repositories, you should also proxy these. A default installation of the repository manager has the NuGet gallery set up as a proxy repository with the name *nuget.org-proxy*.

To proxy another external NuGet repository, you simply create a new *nuget (proxy)* as documented in

Section 4.3. The *Remote Storage* has to be set to the URL of the remote repository you want to proxy.

The default configuration for proxying the NuGet Gallery is partially visible in Figure 7.1 and uses the URL of the API `http://www.nuget.org/api/v2/`.



The screenshot shows the 'Repositories' management interface in Nexus. The breadcrumb trail is 'Repositories / nuget.org-proxy'. A 'Delete repository' button is at the top left. The configuration form includes the following fields:

- Name:** nuget.org-proxy
- Format:** nuget
- Type:** proxy
- URL:** http://localhost:8081/repository/nuget.org-proxy/
- Online:** ☒

NuGet

- Query cache size:** Maximum number of cached query results: 300
- Query cache age:** How long to cache query results (in seconds): 3600

Proxy

- Remote storage:** Location of the remote repository being proxied: http://www.nuget.org/api/v2/

Figure 7.1: NuGet Proxy Repository Configuration for the NuGet Gallery

By default, searches in NuGet proxy repositories in the repository manager initiated by a client like `nuget` or VisualStudio will be passed through to the remote repositories. 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. These counts should therefore be considered approximate numbers. The cache can be configured in the *NuGet* section documented in Section 7.2.

7.4 NuGet Hosted Repositories

A hosted repository for NuGet can be used to upload your own packages as well as third-party packages. The repository manager includes a hosted NuGet repository named *nuget-hosted* by default.

To create another NuGet hosted repository, simply create a new *nuget (hosted)* repository. An example configuration from the default *nuget-hosted* repository is displayed in Figure 7.2.

The screenshot displays the 'Repositories' management interface in Nexus, specifically for the 'nuget-hosted' repository. At the top, there are buttons for 'Delete repository' and 'Rebuild Index'. Below this is a 'Settings' tab. The configuration is organized into several sections:

- Name:** A unique identifier for this repository. The value is 'nuget-hosted'.
- Format:** The format of the repository (i.e. maven2, docker, raw, nuget...). The value is 'nuget'.
- Type:** The type of repository (i.e. group, hosted, or proxy). The value is 'hosted'.
- URL:** The URL used to access this repository. The value is 'http://localhost:8081/repository/nuget-hosted/'.
- Online:** A checkbox labeled 'If checked, the repository accepts incoming requests' is checked.
- Storage:**
 - Blob store:** A dropdown menu showing 'default'.
 - Strict Content Type Validation:** A checkbox labeled 'Validate that all content uploaded to this repository is of a MIME type appropriate for the repository format' is checked.
- Hosted:**
 - Deployment policy:** A dropdown menu labeled 'Controls if deployments of and updates to artifacts are allowed' is set to 'Allow redeploy'.

At the bottom of the settings panel are 'Save' and 'Discard' buttons.

Figure 7.2: Example Configuration for a NuGet Hosted Repository

The NuGet feed is immediately updated as packages are deployed or deleted from the host repository.

7.5 NuGet Repository Groups

A repository group is the recommended way to expose all your NuGet repositories from the repository manager 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.

Nexus Repository Manager includes a *nuget-group* repository group by default. This typical, useful example groups the *nuget.org-proxy* proxy repository that proxies the NuGet Gallery and the *nuget-hosted* hosted repository.

The *URL* of the repository group can be used in your client tool and will give you access to the packages in all repositories from the group with one URL. Any new packages added as well as any new repositories added to the group will automatically be available.

7.6 Accessing Packages in Repositories and Groups

You can access the repository group or individual repositories with the `nuget` tool on the command line using their *URL* e.g.:

```
nuget sources add -name nuget-group -source http://localhost:8081/ repository/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 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 7.7.

7.7 Deploying Packages to NuGet Hosted Repositories

In order to authenticate a client against a NuGet repository, NuGet uses an API key for deployment requests. The API key acts as an alias for the user account, so the same API key is used for all NuGet repositories within the repository manager. This user-specific key is generated separately by a user and can be regenerated at any time. At regeneration, all previous keys generated for that user are invalid.

7.7.1 Accessing your NuGet API Key

For usage with the repository manager, NuGet API keys are only needed when packages are going to be deployed. Users with the necessary *apikey-all* security privilege can access the *NuGet API Key* feature view via the *User* menu by pressing on their username in the main toolbar.

You can access your API key by pressing on the *Access API Key* button and providing your username and password again. The resulting dialog as well as the surrounding user interface context is displayed in Figure 7.3. It shows the API key itself as well as the full command line to register the key for usage with *nuget*.

The *Reset API Key* button can be used to invalidate an existing API key and create a new one.

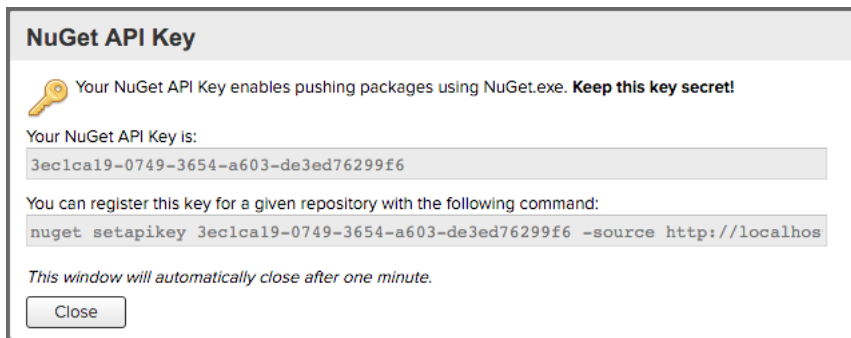


Figure 7.3: Accessing your NuGet API Key

**Important**

Usage of the API key requires the *NuGet API-Key Realm* to be activated. To do this, simply add the realm to the active realms in the *Realms* feature of the *Security* menu from the *Administration* menu.

7.7.2 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](#).

7.7.3 Command line based Deployment to a NuGet Hosted Repository

The `nuget` command line tool allows you to deploy packages to a repository with the `push` command. The command requires you to use the *NuGet API Key* and the *URL* of the target hosted repository. For example, you could push to the hosted repository created in Section 7.4 using the URL `http://localhost:8081/repository/nuget-hosted`.

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](#).

7.8 Integration with Visual Studio

In order to access a NuGet repository or preferably all NuGet repositories exposed in a repository group, you provide the *URL* from the repository manager to configure *Name* and *Source* in the Visual Studio configuration for the *Package Sources* of the *NuGet Package Manager* as displayed in Figure 7.4.

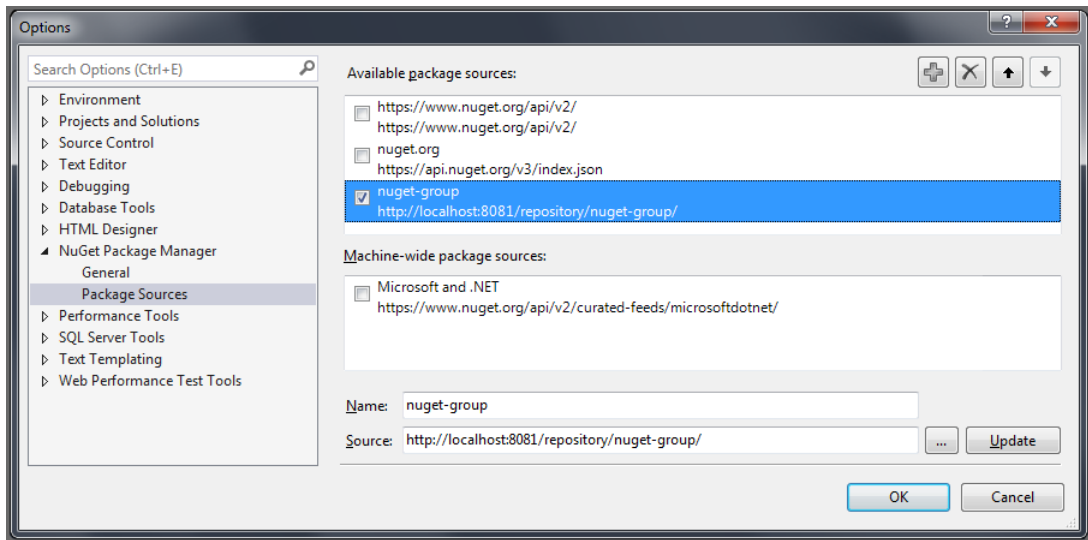


Figure 7.4: Package Source Configuration for the NuGet Package Manager in Visual Studio

With this configuration in place, all packages available in your NuGet repository will be available in the *NuGet Package Manager* in Visual Studio.

Chapter 8

Private Registry for Docker

Available in Nexus Repository OSS and Nexus Repository Pro

8.1 Introduction

Docker containers and their usage have revolutionized the way applications and the underlying operating system are packaged and deployed to development, testing and production systems.

The creation of the **Open Container Initiative**, and the involvement of a large number of stakeholders, guarantees that the ecosystem of tools around the lightweight containers and their usage will continue to flourish.

Docker Hub is the original registry for Docker container images and it is being joined by more and more other publicly available registries such as the **Google Container Registry** and others.

Nexus Repository Manager Pro and Nexus Repository Manager OSS support Docker registries as the Docker repository format for hosted and proxy repositories. You can expose these repositories to the client-side tools directly or as a repository group, which is a repository that merges and exposes the contents of multiple repositories in one convenient URL.

This allows you to reduce time and bandwidth usage for accessing Docker images in a registry as well

as share your images within your organization in a hosted repository. Users can then launch containers based on those images, resulting in a completely private Docker registry with all the features available in the repository manager.

Tip

Docker is a fast moving project and requires usage of current operating system versions and tools. For example, usage of Red Hat Enterprise Linux 6 is simply not supported. Please use the [official documentation](#) as reference and help for your usage.

8.2 SSL and Repository Connector Configuration

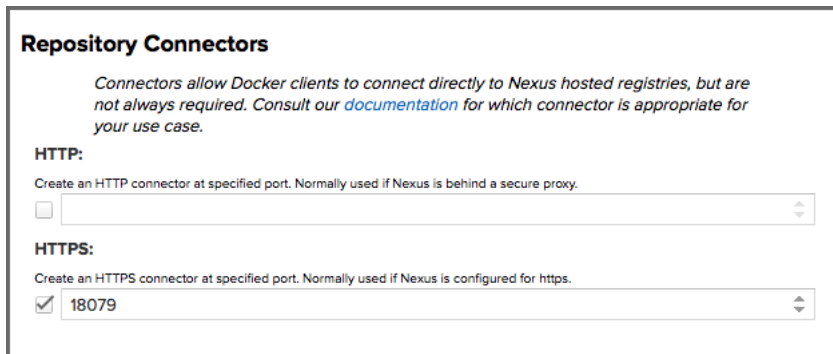
Docker relies on secure connections using SSL to connect to the repositories. You are therefore required to expose the repository manager to your client tools via HTTPS. This can be configured via an external proxy server or directly with the repository manager. Further details can be found in [Section 5.9.4](#).

Interaction of the `docker` client with repositories requires specific ports to be used. These can be configured in the repository configuration in the *Repository Connectors* section. In order for this to work on your network, you need to ensure that the chosen ports are available in your organization and not used by some other application, and that no firewall or other network configuration prevents connectivity.

Note

The `docker` client does not allow a context as part of the path to a registry, as the namespace and image name are embedded in the URLs it uses. This is why requests to repositories on the repository manager are served on a specific and separate port from the rest of the application instead of how most other repositories serve content via a path i.e. `<nexus-hostname>/<repositoryName>/<path to content>`.

The recommended minimal configuration requires one port for a Docker repository group used for read access to all repositories and one port for each hosted Docker repository that will receive push events from your users. The *Repository Connectors* configuration displayed in [Figure 8.1](#) is available in the configuration for proxy and hosted Docker repositories as well as Docker repository groups.



Repository Connectors

Connectors allow Docker clients to connect directly to Nexus hosted registries, but are not always required. Consult our [documentation](#) for which connector is appropriate for your use case.

HTTP:

Create an HTTP connector at specified port. Normally used if Nexus is behind a secure proxy.

☐

HTTPS:

Create an HTTPS connector at specified port. Normally used if Nexus is configured for https.

☒ 18079

Figure 8.1: Repository Connector Configuration

If you have configured the repository manager to use HTTPS directly, you have to configure a HTTPS repository connector. If an external proxy server translates incoming HTTPS requests to HTTP and forwards the request to the repository manager via HTTP you have to configure the respective HTTP port.

Tip

A configured context-path for the user interface does not affect the repository connector URLs used by Docker. E.g. if your repository manager instance is configured to be available at <http://localhost:8081/-nexus> instead of the default root context <http://localhost:8081/>, the URLs for your Docker repositories will still only use the configured port for the repository and omit the context path in the URL. This is a side-effect of the fact that Docker does not support context paths in the registry API.

8.2.1 Tips for SSL Certificate Usage

Nexus Repository Manager is not configured with HTTPS connectors by default as it requires an SSL certificate to be generated and configured manually.

The requirement of Docker to use HTTPS forces the usage of SSL certificates. By default, Docker looks up the validity of the certificate by checking with certificate authorities. If you purchased a certificate that is registered with these authorities, all functionality works as desired.

If you create a certificate yourself with tools such as `openssl`, it is self-signed and not registered. Using a self-signed certificate requires further configuration steps to ensure that Docker can explicitly trust it.

**Warning**

Docker Daemon can stand up instances with the `--insecure-registry` flag to skip validation of a self-signed certificate. But the repository manager does not support the use of the flag, as it generates known bugs and other implementation issues.

To generate a trustworthy self-signed certificate for the repository manager use `keytool`, a utility that lets you manage your own private key pairs and certificates. See our [knowledge base article](#) to learn how to configure the utility.

8.3 Support for Docker Registry API

The Docker client tools interact with a repository via the registry API. It is available in version 1 (V1) and version 2 (V2). The newer V2 will completely replace the old V1 in the future. Currently Docker Hub and other registries as well as other tools use V2, but in many cases fall back to V1. E.g., search is currently only implemented in V1.

Nexus Repository Manager supports V1 as well as V2 of the API. All Docker repository configurations contain a section to configure *Docker Registry API Support*. If you activate *Enable Docker V1 API* for a repository it is enabled to use V1 as a fallback from V2. Without this option any V1 requests result in errors from the client tool.

Tip

Generally V1 support is only needed for repository groups that will be used for command line-based searches, when any client side tools in use require V1 or when a upstream proxy repository requires V1. If you are unsure if your setup uses these or V1, it is recommended to activate V1 support as there should be no harm if it is not needed.

8.4 Proxy Repository for Docker

Docker Hub is the common registry used by all image creators and consumers. To reduce duplicate downloads and improve download speeds for your developers and CI servers, you should proxy Docker Hub and any other registry you use for Docker images.

To proxy a Docker registry, you simply create a new *docker (proxy)* as documented in Section 4.3 in details.

Minimal configuration steps are:

- Define *Name*
- Define URL for *Remote storage*
- *Enable Docker VI API* support, if required by the remote repository
- Select correct *Docker index*, further configure *Location of Docker index* if needed
- Select *Blob store* for *Storage*

Optionally you can configure *Repository Connectors* as explained in Section 8.2, although typically read access is done via a repository group and not a proxy repository directly, and write access is done against a hosted repository.

The *Remote Storage* has to be set to the URL of the remote registry you want to proxy. The configuration for proxying Docker Hub uses the URL `https://registry-1.docker.io` for the *Remote storage* URL.

The *Proxy* configuration for a Docker proxy repository includes a configuration URL to access the *Docker Index*. The index is used for requests related to searches, users, docker tokens and other aspects. The registry and the index are typically co-hosted by the same provider, but can use different URLs. E.g. the index for Docker Hub is exposed at `https://index.docker.io/`.

The default option *Use proxy registry (specified above)* will attempt to retrieve any index data from the same URL configured as the *Remote storage* URL.

The option to *Use Docker Hub* fulfills any index related requests by querying the Docker Hub index at `https://index.docker.io/`. This configuration is desired when the proxy repository is Docker Hub itself or any of its mirrors.

The option to use a *Custom index* allows you to specify the URL of the index for the remote repository.

It is important to configure a correct pair of *Remote Storage* URL and *Docker Index* URL. In case of a mismatch, search results potentially do not reflect the content of the remote repository and other problems can occur.

Tip

Just to recap, in order to configure a proxy for Docker Hub you configure the *Remote Storage* URL to <https://registry-1.docker.io>, enable Docker V1 API support and for the choice of *Docker Index* select the *Use Docker Hub* option.

8.5 Hosted Repository for Docker (Private Registry for Docker)

A hosted repository using the Docker repository format is typically called a private Docker registry. It can be used to upload your own container images as well as third-party images. It is common practice to create two separate hosted repositories for these purposes.

To create a Docker hosted repository, simply create a new *docker (hosted)* repository as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for *Storage*

If you add a *Repository Connectors* configuration as documented in Section 8.2 you can push images to this repository, and subsequently access them directly from the hosted repository or ideally from the Docker repository group as documented in Section 8.6.

By default this setup will allow repeated deployment of images. If you want to enforce new deployments using different versions, set the *Deployment Policy* to *Disable Redeploy*.

8.6 Repository Groups for Docker

A repository group is the recommended way to expose all your repositories for read access to your users. It allows you to pull images from all repositories in the group without needing any further client side configuration after the initial setup. A repository group allows you to expose the aggregated content of multiple proxy and hosted repositories with one URL to your tools.

To create a Docker repository group, simply create a new *docker (group)* repository as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for *Storage*
- Add Docker repositories to the *Members* list in the desired order

Typically the member list includes a mixture of proxy and hosted repositories to allow access to public as well as private images.

Using the *Repository Connectors* port of the repository group and the URL of the repository manager in your client tool gives you access to the container images in all repositories from the group. Any new images added as well as any new repositories added to the group will automatically be available.

Tip

Check out this repository configuration demonstrated in [a video](#).

8.7 Authentication

If access to a repository requires the user to be authenticated, `docker` will check for authentication access in the `~/.docker/config.json` file. If authentication is not found, some actions will prompt for authentication but otherwise a `docker login` command will be required before the actions can be performed. Typically this is required when [anonymous access](#) to the repository manager is disabled or the operation requires authentication.

The `docker login` command observes the following syntax for the desired repository or repository group:

```
docker login <nexus-hostname>:<repository-port>
```

Provide your repository manager credentials of username and password as well as an email address. This authentication is persisted in `~/.docker/config.json` and reused for any subsequent interactions

against that repository. Individual login operations must be performed for each repository and repository group you want to access in an authenticated manner.

Tip

Specifically when planning to push to a repository a preemptive login operation is advisable as it removes the need for user interaction and is therefore suitable for continuous integration server setups and automated scenarios.

8.8 Accessing Repositories

You can browse Docker repositories in the user interface and inspect the components and assets and their details as documented in [Section 3.4](#).

When using the *docker* command line client, or any other tools using the repository manager indirectly, the common structure for commands can be:

```
docker <command> <nexus-hostname>:<repository-port>/<namespace>/<image>:<tag>
docker search <nexus-hostname>:<repository-port>/<search-term>
```

with

command

a docker command such as *push* or *pull*

nexus-hostname

the IP number or hostname of your repository manager

repository-port

the port configured as the repository connector for the specific repository or repository group

namespace

the optional namespace of the specific image reflecting the owner, if left out this will silently default to */library* and utilize Docker Hub

image

the name of the Docker image

tag

the optional tag of the image, defaulting to *latest* when omitted

search-term

the search term or name of the image to search for

The most important aspects are to know and use the correct hostname for the repository manager and the port for the desired repository or repository group.

8.9 Searching

Searching for Docker images can be performed in the user interface as described in Section 3.3. This search will find all Docker images that are currently stored in repositories, either because they have been pushed to a hosted repository or they have been proxied from an upstream repository and cached in the repository manager.

The more common use case for a Docker user is to search for images on the command line:

```
$ docker search postgres
NAME                DESCRIPTION                                STARS   OFFICIAL   ←
AUTOMATED
postgres            The PostgreSQL object-relational database... 1025    [OK]
...
```

By default this search uses Docker Hub as preconfigured in `docker` and will only find images available there. A more powerful search is provided by the repository manager when searching against a repository group. An example looking for a `postgres` image on Nexus Repository Manager OSS running on the host `nexus.example.com` and exposing a repository group with a repository connector port of 18443 looks like this:

```
docker search nexus.example.com:18443/postgres
```

The results include all images found in the repositories that are part of the repository group. This includes any private images you have pushed to your hosted repositories. In addition it includes all results returned from the remote repositories configured as proxy repositories in the group. Searching in a specific repository can be achieved by using the repository connector port for the specific repository.

8.10 Pulling Images

Downloading images, also known as pulling, from the repository manager can be performed with the `docker pull` command. The only necessary additions are the hostname or IP address of the repository manager as well as the repository connector port for the repository or repository group to download from:

```
docker pull <nexus-hostname>:<repository-port>/<image>
```

The preferred setup is to proxy all relevant sources of public/private images you want to use, with Docker Hub being the most common choice. Then configure one or more hosted repositories to contain your own images, and expose these repositories through one repository group.

Examples for various images from Nexus Repository Manager running on the host `nexus.example.com` and exposing a repository group with a repository connector port of 18443 are:

```
docker pull nexus.example.com:18443/ubuntu
docker pull nexus.example.com:18443/bitnami/node
docker pull nexus.example.com:18443/postgres:9.4
```

These snippets download the official `ubuntu` image, the `node` image from the user `bitnami` and the version 9.4 of the `postgres` image. Official images such as `ubuntu` or `postgres` belong to the `library` user on Docker Hub and will therefore show up as `library/ubuntu` and `library/postgres` in the repository manager.

After a successful `pull` you can start the container with `run`.

8.11 Pushing Images

Sharing an image can be achieved, by publishing it to a hosted repository. This is completely private and requires you to `tag` and `push` the image. When tagging an image, you can use the image identifier (`imageId`). It is listed when showing the list of all images with `docker images`. Syntax and an example (using `imageId`) for creating a tag are:

```
docker tag <imageId or imageName> <nexus-hostname>:<repository-port>/< ←
  image>:<tag>
docker tag af340544ed62 nexus.example.com:18444/hello-world:mytag
```


Once the tag, which can be equivalent to a version, is created successfully, you can confirm its creation with `docker images` and issue the push with the syntax:

```
docker push <nexus-hostname>:<repository-port>/<image>:<tag>
```

**Important**

Note that the port needs to be the repository connector port configured for the **hosted** repository to which you want to push to. You can not push to a repository group or a proxy repository.

A sample output could look like this:

```
$ docker push nexus.example.com:18444/hello-world:labeltest
The push refers to a repository [nexus.example.com:18444/hello-world] (len ↵
: 1)
Sending image list
Pushing repository nexus.example.com:18444/hello-world (1 tags)
535020c3e8ad: Image successfully pushed
af340544ed62: Image successfully pushed
Pushing tag for rev [af340544ed62] on
{https://nexus.example.com:18444/repository/docker-internal/v1/ ↵
repositories/hello-world/tags/labeltest}
```

Now, this updated image is available in the repository manager and can be pulled by anyone with access to the repository, or the repository group, containing the image. Pulling the image from the repository group exposed at port 18443 can be done with:

```
docker pull nexus.example.com:18443/hello-world:labeltest
```

Prior to push, and depending on your configuration, repository manager login credentials may be required before a push or pull can occur.

Tip

Searching, Browsing, Pushing and Pulling are all showcased in [this video](#).

Pushing large images can result in failures due to network interruptions and other issues. These partial uploads result in temporary storage for these transfers in the repository manager filling up. The task *Purge incomplete docker uploads* can be configured to delete these files. Further documentation can be found in Section [4.2.6](#).

Chapter 9

Node Packaged Modules and npm Registries

Available in Nexus Repository OSS and Nexus Repository Pro

9.1 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 `npmjs` website, available at <https://www.npmjs.org>, provides search and other convenience features to access the public registry at <https://registry.npmjs.org/>. It 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 Repository Manager Pro and Nexus Repository Manager OSS support 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 the repository manager.

In addition, Nexus Repository Manager 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 Repository Manager supports aggregation of npm registries. This allows you to expose all the external packages 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 npm registry hosted by the repository manager. 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.

9.2 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>. By default npm accesses this registry directly. You can also proxy any other registries you require.

To proxy an external npm registry, you simply create a new *npm (proxy)* as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Define URL for *Remote storage* e.g. `https://registry.npmjs.org`
- Select *Blob store* for *Storage*

9.3 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 the repository manager. 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 *npm (hosted)* as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for *Storage*

The npm registry information is immediately updated as packages are deployed or deleted from the repository.

9.4 Grouping npm Registries

A repository group is the recommended way to expose all your npm registries repositories from the repository manager 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 *npm (group)* as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for *Storage*
- Add npm repositories to the *Members* list in the desired order

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.

Using the *URL* 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.

9.5 Browsing npm Registries and Searching Modules

You can browse npm repositories in the user interface inspecting the components and assets and their details as documented in [Section 3.4](#).

Searching for npm modules can be performed in the user interface as described in [Section 3.3](#). This search will find all npm modules images that are currently stored in the repository manager, either because they have been pushed to a hosted repository or they have been proxied from an upstream repository and cached in the repository manager.

9.6 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 with the `npm config` command and the public URL of your repository group in the repository list in the *Repository Path* column.

```
npm config set registry http://localhost:8081/repository/npm-all/
```

The command inserts the configuration in the `.npmrc` file in your users home directory.

Registry configuration in `.npmrc`

```
registry = http://localhost:8081/repository/npm-all/
```

With this configuration any npm commands will use the new registry from the repository manager. The command line output will reference the URLs in `--verbose` mode or with `info` logging for the downloads of the requested packages:

```
$ npm --loglevel info install grunt
...
npm http fetch GET http://localhost:8081/repository/npmjs-org/grunt/-/ ←
  grunt-0.4.5.tgz
npm http fetch 200 http://localhost:8081/repository/npmjs-org/grunt/-/ ←
  grunt-0.4.5.tgz
...
```

```
npm http fetch GET http://localhost:8081/repository/npm-all/underscore/-/ ↵
underscore-1.7.0.tgz
npm http fetch 200 http://localhost:8081/repository/npm-all/underscore/-/ ↵
underscore-1.7.0.tgz
...
```

9.7 npm Security

By default any anonymous user has read access to the repositories and repository groups. If anonymous access, as documented in Section 5.6 is disabled, or write access is required for publishing a package, the user needs to authenticate to the repository manager.

This authentication requires the `npm Bearer Token Realm`. Simply add the realm to the active realms in the `Realms` feature of the `Security` menu from the `Administration` menu to activate it as documented in Section 5.2.

Once the realm is activated, a user can establish the authentication to a repository with the `npm login` command.

```
npm login --registry=http://localhost:8081/repository/npm-internal
```

Provide your repository manager username and password as well as your email address when prompted. Upon successful completion, a line for authentication of this combination is automatically added to your `.npmrc` configuration file for the specific repository.

Further details on `npm login` can be found on the [npm website](#).

9.8 Publishing npm Packages

Publishing your own packages to a npm hosted repository allows you to share packages across your organization or with external partners. With authentication configured you can publish your packages with the `npm publish` command.

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 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/repository/npm-internal/
```

Alternately, you can edit your `package.json` file and add a `publishConfig` section:

```
"publishConfig" : {  
  "registry" : "http://localhost:8081/repository/npm-internal/"  
},
```

Detailed information about package creation can be found on the [npm website](#).

If your package requires the use of `npm scope`, the repository manager supports this functionality. Packages published to the repository manager with a defined scope are reflected with the scope value populating the repository group field in Browse and Search. Details on scoping are available on the [npm website](#) also.

Once a package is published to the private registry in the repository manager, any other developers or build servers that access the repository manager via the repository group have instant access to it.

9.9 Deprecating npm Packages

Once your packages have been pushed to an npm hosted repository, you can mark them as deprecated. This is useful when a newer version of the package is available, and you want to warn people that the old package has reached end of life or you want to avoid usage and warn your users for some other reason.

The `npm deprecate` command uses a `registry` configuration value to inform where the package lives. To deprecate an existing package, use a command like the following:

```
npm deprecate --registry http://localhost:8081/repository/npm-internal/ ↵  
testproject1@0.0.1 "This package is deprecated"
```

If you change your mind, you can reverse this action using the same command. To undeprecate a package, pass an empty string to the `deprecate` command:

```
npm deprecate --registry http://localhost:8081/repository/npm-internal/ ↵  
testproject1@0.0.1 ""
```

The message text is persisted in the `deprecated` attribute of the *packageJson* section for the asset and can be viewed in the user interface.

Chapter 10

Bower Repositories

Available in Nexus Repository OSS and Nexus Repository Pro

10.1 Introduction

Bower is a package manager for front-end web development. JavaScript developers using Bower gain convenient access to a large amount of packages from the remote Bower registry. This reduces the complexity of their development efforts and improves the resulting applications.

Nexus Repository Manager Pro and Nexus Repository Manager OSS support the Bower registry format for hosted and proxy repositories. This allows the repository manager to take advantage of the packages in the official Bower registry and other public registries without incurring repeated downloads of packages.

The official Bower registry is available for searches at <http://bower.io/search> and for package retrieval via the URL <http://bower.herokuapp.com>.

You can publish your own packages to a private Bower registry as a hosted repository on the repository manager and then expose the remote and private repositories to Bower as a repository group, which is a repository that merges and exposes the contents of multiple repositories in one convenient URL. This allows you to reduce time and bandwidth usage for accessing Bower packages a registry as well as share your packages within your organization in a hosted repository.

10.2 Proxying Bower Repositories

You can set up a Bower proxy repository to access a remote repository location, for example the official Bower registry at <http://bower.herokuapp.com> that is configured as the default on Bower.

To proxy a Bower registry, you simply create a new *bower (proxy)* as documented in Section 4.3 in details. Minimal configuration steps are:

- Define *Name*
- Define URL for *Remote storage* e.g. <http://bower.herokuapp.com>
- Select a *Blob store* for *Storage*

The *Bower* specific configuration section include the setting to *Enable rewrite of package URLs*. This causes Bower to retrieve components and their dependencies through the repository manager even if original meta data has hard-coded URLs to remote repositories. This setting *Force Bower to retrieve packages via the proxy repository* is enabled by default.

If deactivated, no rewrite of the URL occurs. As a result, the original component URL is exposed. Turning off rewrite capabilities proxies the information directly from the remote registry without redirecting to the repository manager to retrieve components.

10.3 Hosting Bower Repositories

Creating a Bower hosted repository allows you to register packages in the repository manager. The hosted repository acts as an authoritative location for these components. This effectively creates an asset that becomes a pointer to an external URL (such as a Git repository).

To add a hosted Bower repository, create a new repository with the recipe *bower (hosted)* as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name* - e.g. `bower-internal`
-

- Select *Blob store* for *Storage*

10.4 Bower Repository Groups

A repository group is the recommended way to expose all your Bower repositories from the repository manager to your users, with minimal additional client side configuration. A repository group allows you to expose the aggregated content of multiple proxy and hosted repositories as well as other repository groups with one URL in tool configuration. This is possible for Bower repositories by creating a new repository with the *bower (group)* recipe as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name* - e.g. `bower-all`
- Select *Blob store* for *Storage*
- Add Bower repositories to the *Members* list in the desired order

10.5 Installing Bower

Bower is typically installed with `npm`. Since the repository manager supports NPM repositories for proxying, we recommend to configure the relevant NPM repositories and `npm` as documented in Chapter 9 prior to installing Bower. Once this is completed you can install Bower with the usual command.

```
npm install -g bower
```

Bower version 1.5 or higher is required and can be verified with

```
$ bower -v  
1.7.7
```

In addition Bower requires a custom URL resolver to allow integration with Nexus Repository Manager Pro and Nexus Repository Manager OSS. The resolver is an API introduced in Bower version 1.5. Bower fetches component and version information through the repository manager, then automatically searches and saves the component in the repository. You can install the resolver with:

```
npm install -g bower-nexus3-resolver
```

Alternatively you can install the resolver on a per-project basis instead by adding it as a dependency in your `package.json`:

```
"devDependencies" : {  
  "bower-nexus3-resolver" : "*"   
}
```

10.6 Configuring Bower Package Download

Once you have set up your repositories for Bower packages, and installed Bower and the custom resolver, you can create a `.bowerrc` JSON file to access registry URLs. The `registry` value is configured to access the Bower repository group that exposes the proxy and hosted repositories together. The `resolvers` configuration is necessary, so that Bower uses the required custom resolver.

Global `.bowerrc` file in your home directory for package download via the group `bower-all`

```
{  
  "registry" : {  
    "search" : [ "http://localhost:8081/repository/bower-all" ]  
  },  
  "resolvers" : [ "bower-nexus3-resolver" ]  
}
```



Important

The `.bowerrc` file can be located in various locations. For global configuration for a specific developer working on multiple projects the users home directory is a suitable location. If multiple files exist, they are merged. Details can be found in [the documentation](#).

With this configuration in place, any further Bower command invocations trigger package downloads via the repository manager.

Running an `install` command logs the download via the repository manager:

```
$ bower install jquery
bower jquery#*          not-cached nexus+http://localhost:8081/ ↵
  repository/bower-all/jquery#*
bower jquery#*          resolve nexus+http://localhost:8081/ ↵
  repository/bower-all/jquery#*
bower jquery#*          resolved nexus+http://localhost:8081/ ↵
  repository/bower-all/jquery#2.2.0
bower jquery#^2.2.0      install jquery#2.2.0
jquery#2.2.0 bower_components/jquery
```

If anonymous access to the repository manager is disabled, you have to specify the credentials for the accessing the repository manager as part of the URL like `http://username:password@host:port/repository/bower-all` and add a nexus section to your `.bowerrc` file.

```
{
  "nexus" : {
    "username" : "myusername"
    "password" : "mypassword"
  }
}
```

Downloaded packages are cached, do not have to be retrieved from the remote repositories again and can be inspected in the user interface.

10.7 Browsing Bower Repositories and Searching Packages

You can browse Bower repositories in the user interface inspecting the components and assets and their details, as described in Section 3.3.

Searching for Bower packages can be performed in the user interface, too. It finds all packages that are currently stored in the repository manager, either because they have been pushed to a hosted repository or they have been proxied from an upstream repository and cached in the repository manager.

10.8 Registering Bower Packages

If you are authoring your own packages and want to distribute them to other users in your organization, you have to register them to a hosted repository on the repository manager. This establishes a metadata file in the repository that links to the source code repository. Typically this is a git repository. The consumers can then download it via the repository group as documented in Section 10.6.

You can specify the URL for the target hosted repository in the `register` value in your `.bowerrc` file. If you are registering all packages you create in the same hosted repository you can configure in the your global configuration file e.g. located in your users home directory:

```
{
  "registry" : {
    "search" : [
      "http://localhost:8081/repository/bower-all"
    ],
    "register" : "http://localhost:8081/repository/bower-internal"
  },
  "resolvers" : [ "bower-nexus3-resolver" ]
}
```

Alternatively, if you desire to use a per-project `.bowerrc` file that you potentially version in your source code management system with the rest of the package code, you can use a simplified file:

```
"registry": {
  "register": "http://localhost:8081/repository/bower-hosted"
}
```

Authentication is managed in the same manner as for proxying with anonymous access disabled as documented in Section 10.6.

With this configuration you can run a command such as

```
bower register example-package git://gitserver/project.git
```

All semantic version tags on the git repository are now exposed as version for this package and consumers can install the package via the repository group like any other package.

```
bower install example-package
```

Chapter 11

PyPI Repositories

Available in Nexus Repository OSS and Nexus Repository Pro

11.1 Introduction

The Python Package Index, or PyPI, is a vast repository of open-source Python packages supplied by the worldwide community of Python developers. The official index is available at <https://pypi.python.org/pypi>, and the site itself is maintained by the [Python Software Foundation](#).

Both Nexus Repository Manager Pro and Nexus Repository Manager OSS support proxying the Python Package Index. This allows the repository manager to take advantage of the packages in the official Python Package Index without incurring repeated downloads. This will reduce time and bandwidth usage for accessing Python packages.

Also, you can publish your own packages to a private index as a hosted repository on the repository manager, then expose the remote and private repositories as a repository group, which is a repository that merges and exposes the contents of multiple repositories in one convenient URL.

Tip

If using pip with the repository manager, you should consider setting up your repository manager to use SSL as documented in Section 5.9. Otherwise, you will likely need to put `--trusted-host` additions at the end of many commands or further configure pip to trust your repository manager.

11.2 Proxying PyPI Repositories

You can set up a PyPI proxy repository to access a remote repository location, such as the PyPI repository at <https://pypi.python.org/pypi>. The index is maintained as the default location for Python packages.

To proxy a PyPI package, you simply create a new *pypi(proxy)* recipe as documented in Section 4.3.2, in detail. Minimal configuration steps are:

- Define *Name*
- Define URL for *Remote storage* e.g. <https://pypi.python.org/>
- Pick a *Blob store* for *Storage*

The repository manager can access Python packages and tools from the index. The proxy repository for PyPI packages provides a cache of files available on the index. This allows the local network client to access components from the Python Package Index more reliably.

The proxy configuration for a PyPI proxy repository includes a configuration URL to access the index. Users will be able to browse and search assets against a remote repository, as mentioned in Section 11.8.

11.3 Hosting PyPI Repositories

Creating a PyPI hosted repository allows you to upload packages in the repository manager. The hosted repository acts as an authoritative location for packages fetched from the Python index.

To host a PyPI package, create a new *pypi(hosted)* recipe as documented in Section 4.3.3, in detail. Minimal configuration steps are:

- Define *Name* - e.g. `pypi-internal`
- Pick a *Blob store* for *Storage*

11.4 PyPI Repository Groups

A repository group is the recommended way to expose all your PyPI repositories from the repository manager to your users, with minimal additional client side configuration. A repository group allows you to expose the aggregated content of multiple proxy and hosted repositories as well as other repository groups with one URL in tool configuration. PyPI group repositories can be created with the *pypi(group)* recipe as documented in Section 4.3.4.

Minimal configuration steps are:

- Define *Name* - e.g. `pypi-all`
- Pick a *Blob store* for *Storage*
- Add PyPI repositories to the *Members* list in the desired order

11.5 Installing PyPI Client Tools

The latest versions of such Linux distributions as CentOS and Ubuntu come packaged with Python 2.7 and **pip**, a tool for installing and managing Python packages from the index. For Mac OS X and Microsoft Windows, download and install a Python version compatible with the repository manager from <https://www.python.org/downloads/>. Download the pip installer from <https://pip.pypa.io/en/stable/-installing/>.

Note

Nexus Repository Manager Pro and Nexus Repository Manager OSS support specific versions of Python, pip, and setuptools. For Python the repository manager supports the latest of releases 2 and 3, as well as some earlier versions (i.e. 2.7 and earlier, 3.5 and earlier). For pip versions 7 and 8 are supported. The latest two versions of setuptools, used to build and distribute Python dependencies, are compatible with the repository manager.

11.6 Configuring PyPI Client Tools

Note

Depending on your preference for either `setuptools`, `twine`, `distutils`, and `pip` your proxy and hosted configuration may vary.

Once you have installed all necessary client tools from the Python Package Index, you can create and configure a `.pypirc` file to reference packages stored in the repository manager. Depending on your Python configuration you can manage your repository groups with `pip.conf` or `setup.cfg` to have all commands, such as search and install, run against your project.

Configuring a proxy repository to use `easy_install`

You can create a `setup.cfg`, if using `easy_install`. The `index-url` is the tag created to specify the base URL for the PyPI package. In this example `index-url` is set for a proxy repository:

```
[easy_install]
index-url = http://localhost:8081/repository/pypi-proxy/simple
```

If you prefer to configure `easy_install` for hosted (`pypi-internal`) or group (`pypi-all`) adjust the file accordingly.

Configuring your hosted repository with `.pypirc`

If you are authoring your own packages and want to distribute them to other users in your organization, you have to upload them to a hosted repository on the repository manager. The `.pypirc` holds your credentials for authentication when hosting a PyPI repository.

In the example `.pypirc` file below, specify the URL you want to deploy to the target hosted repository in the `repository` value. Add `username` and `password` values to access the repository manager. The `.pypirc` file contains `distutils`, a default server used by PyPI that provides upload commands that stores assets and authentication information.

```
[distutils]
index-servers =
    nexus

[nexus]
repository = http://localhost:8081/repository/pypi-internal/
```

```
username = admin
password = admin123
```

Note

If you have multiple hosted repositories, you can add them to the `.pypirc` file, each with a different name, pointing to the corresponding repository URL.

After this is configured, you can upload packages to the hosted repository, as explained in [Section 11.9](#).

Global `pip.conf` file with a repository group

If you want your `pip.conf` to install or search Python within a group, configure the file to include the repository group URL.

```
[global]
index = http://localhost:8081/repository/pypi-all/pypi
index-url = http://localhost:8081/repository/pypi-all/simple
```

If you prefer to configure your global `pip.conf` for proxy (`pypi-proxy`) or hosted (`pypi-internal`) adjust the file accordingly.

11.7 SSL Usage for PyPI Repositories

You can proxy Python packages over HTTPS to ensure a secure connection with a self-signed certificate. This works for proxy, hosted, and group repositories. To set up the repository manager to serve HTTPS follow the configuration steps in [Section 5.9](#).

Also, you can set up `pip` to use the certificate to enable SSL and fetch packages securely. Additional configuration is necessary for the HTTPS client implementation to work. This assumes the repository manager has already been set up to use SSL, so verify your certificate works. Run the following command:

```
openssl verify <example-certificate>
```

When your certificate is proven to work, update your `pip.conf`. Here is an example configuration file for a repository group:

```
[global]
index = https://localhost:8443/repository/pypi-all/pypi
index-url = https://localhost:8443/repository/pypi-all/simple
cert = nexus.pem
```

11.8 Browsing PyPI Repositories and Searching Packages

You can browse PyPI repositories in the user interface inspecting the components and assets and their details, as described in [Section 3.4](#).

Searching for PyPI packages can be performed in the user interface, as described in [Section 3.3](#). It finds all packages that are currently stored in the repository manager, either because they have been pushed to a hosted repository or they have been proxied from an upstream repository and cached in the repository manager.

From the command line you can search available PyPI packages defined in your configuration. This method is limited to `pip` (`pip.conf`). To search, run:

```
pip search example-package
```

11.9 Uploading PyPI Packages

Note

The steps to upload a PyPI package will vary if your system is configured with `setuptools` or `twine`.

After you configure your `.pypirc` you can upload packages from the index to the repository manager.

In the example below, `twine` is invoked to tell your repository what server to use when uploading a package. The `-r` flag is used to find the `nexus` server in your `.pypirc`.

```
twine upload -r nexus <filename>
```

Chapter 12

Ruby, RubyGems and Gem Repositories

Available in Nexus Repository OSS and Nexus Repository Pro

12.1 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

ing 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 Repository Manager Pro and Nexus Repository Manager OSS support gem repositories. A user can connect to the repository manager to download gems from RubyGems, create proxies to other repositories, and host their own or third-party gems. Any gem downloaded via the repository manager needs to be downloaded from the remote repository, like RubyGems, only once and is then available internally from the repository manager. Gems pushed to the repository manager automatically become available to everyone else in your organization. Using the repository manager 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 version 3.1 and higher

The following features are included as part of the gem repository support:

- Proxy repository for connecting to remote gem repositories and caching gems on the repository manager to avoid duplicate downloads and wasted bandwidth and time
- Hosted repository for hosting gem packages and providing them to 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 the repository manager.

12.2 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, you simply create a new repository using the recipe *rubygems (proxy)* as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Define URL for *Remote storage* e.g. `https://rubygems.org` (the official URL for RubyGems.org)
- Select a *Blob store* for *Storage*

Further configuration details are available in Section 4.3.5.

12.3 Private Hosted Gem Repositories

A private gem repository can be used as a target to push your own gems as well as third-party gems and subsequently provide them to your users. It is a good practice to create two separate hosted gem repositories for internal and third-party gems.

To create a hosted gem repository, create a new repository using the recipe *rubygems (hosted)* as documented in Section 4.3.

Minimal configuration steps are:

- Define *Name*
- Select a *Blob store* for *Storage*

The gem repository information is immediately updated as gems are pushed to the repository or deleted from it.

12.4 Grouping Gem Repositories

A repository group is the recommended way to expose all your gem repositories 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.

To create a gem group repository, create a new repository using the recipe *rubygems (group)* as documented in Section [4.3](#).

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for Storage
- Add RubyGems repositories to the *Members* list in the desired order

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.

Using the repository *URL* of the repository group as your gem repository URL in your client tool gives you access to the gems in all member repositories with one URL.

Any gem added to a hosted or proxy repository becomes immediately available to all users of the gem repository group. Adding a new proxy gem repository to the group makes all gems in that proxy immediately available to the users as well.

12.5 Using Gem Repositories

Once you have configured the repository manager 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 gem repository or group using the *URL* from the repository list with a command like

```
$ gem sources --add http://localhost:8081/repository/rubygems-group/
```

In order to take full advantage of the repository manager and the proxying of gems, you should remove any other sources. By default `https://rubygems.org/` is configured in `gem` and this can be removed with

```
$ gem sources --remove https://rubygems.org/
```

Subsequently you should clear the local cache with

```
$ gem sources -c
```

To check a successful configuration you can run

```
$ gem sources
*** CURRENT SOURCES ***

http://localhost:8081/repository/rubygems-group/
```

With this setup completed any installation of new gems with `gem install gemname` (e.g. `gem install rake`) will download from the repository manager.

By default read access is available to anonymous access and no further configuration is necessary. If your repository manager requires authentication, you have to add the *Basic Auth* authentication details to the sources configuration:

```
$ gem sources --add http://myuser:mypassword@localhost:8081/repository/ ↵
    rubygems-group/
```

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 the repository manager with Bundler, you have to configure the gem repository group as a mirror:

```
$ bundle config mirror.http://rubygems.org
http://localhost:8081/repository/rubygems-group/
```

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/repository/rubygems-group"
```

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 gem repository group configured as a mirror.

12.6 Pushing Gems

At this point you have set up the various gem repositories on the repository manager (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 Repository Manager Pro 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 can push your gem to a desired repository. The initial invocation will request the URL for the gem repository and the 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/repository/rubygems-hosted
The Nexus URL has been stored in ~/.gem/nexus
Enter your Nexus credentials
Username:  admin
Password:
Your Nexus credentials has been stored in /Users/manfred/.gem/nexus
Uploading gem to Nexus...
Created
```

By default pushing an identical version to the repository, 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 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/repository/rubygems-hosted
```

Chapter 13

Raw Repositories, Maven Sites and More

Available in Nexus Repository OSS and Nexus Repository Pro

13.1 Introduction

Nexus Repository Manager Pro and Nexus Repository Manager OSS include support for hosting, proxying and grouping static websites - the *raw* format. Hosted repositories with this format can be used to store and provide a Maven-generated website. Proxy repositories can subsequently proxy them in other servers. The *raw* format can also be used for other resources than HTML files exposed by straight HTTP-like browsable directory structures.

This chapter details the process of configuring raw repositories, configuring a simple Maven project to publish a Maven-generated project site and other use cases for raw repositories.

13.2 Creating a Hosted Raw Repository

To create a raw repository for hosting a static website, you simply create a new repository using the *raw* (*hosted*) recipe as documented in [Section 4.3](#).

For the Maven site example in Section 13.3, set the *Name* to `site` and change the *Deployment policy* to *Allow redeploy*.

After creating the new raw repository, it appears in the list of repositories with the name *site* provided earlier. The *URL* in the list can be used for deployment and access usage.

13.3 Creating and Deploying a Maven Site

13.3.1 Creating a New Maven Project

In this section, you are be creating a minimal Maven project with a simple website that can be published to the hosted raw repository created in Section 13.2.

The following steps can be used to create a new Maven project:

- Run the command `mvn archetype:generate` in a command line interface
- Confirm the first prompt using the default selection (number will vary)
- Confirm the default selection for the archetype version
- Set the `groupId` to `org.sonatype.books.nexus`
- Set the `artifactId` to `sample-site`
- Confirm the default version of `1.0-SNAPSHOT`
- Confirm the preset package of `org.sonatype.books.nexus`
- Confirm the properties configuration

After running the `archetype:generate` command, you will have a new project in a `sample-site` directory.

13.3.2 Configuring Maven for Site Deployment

To deploy a site to a raw repository in the repository manager, you need to configure the project's `distributionManagement`, add site deployment information, and then update your Maven settings to

include the appropriate credentials.

Add the following section to `sample-site/pom.xml` before the `dependencies` element. This section tells Maven where to publish the Maven-generated project website:

Distribution Management for Site Deployment

```
<distributionManagement>
  <site>
    <id>nexus</id>
    <url>dav:http://localhost:8081/repository/site/</url>
  </site>
</distributionManagement>
```

The URL in the distribution management is not parameterized, 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 change the whole URL between deployments.

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 the repository manager requires that you add the implementation library as a dependency of the Maven site plugin in your Maven project:

Dependency for the Maven Site Plugin for 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>2.8</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

13.3.3 Adding Credentials to Your Maven Settings

When the Maven site plugin deploys a site, it needs to supply the appropriate deployment credentials to the repository manager. To configure this, you need to add credentials to your Maven settings. Edit your `~/.m2/settings.xml` file and add the following server configuration to the `servers` element.

Configuring Deployment Credentials for Site Deployment

```
<settings>
  <servers>
    <server>
      <id>nexus</id>
      <username>admin</username>
      <password>admin123</password>
    </server>
  </servers>
</settings>
```

Note

[Configuring Deployment Credentials for Site Deployment](#) uses the default `admin` username and password. For real world usage you would use the username and password of a user with the privilege to write to the target repository.

13.3.4 Publishing a Maven Site

To publish the site to the hosted raw repository in the repository manager, run `mvn site-deploy` from the `sample-site` directory. The Maven site plugin will deploy this site to the repository using the credentials stored in your Maven settings:

Sample Maven Log from Deploying a Site

```
$ mvn site-deploy
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building sample-site 1.0-SNAPSHOT
...
[INFO] --- maven-site-plugin:3.4:site (default-site) @ sample-site ---
...
[INFO] Generating "About" report.
```

```
...
[INFO] --- maven-site-plugin:3.4:deploy (default-deploy) @ sample-site ---
http://localhost:8081/repository/site/ - Session: Opened
[INFO] Pushing /Users/manfred/training/sample-site/target/site
[INFO] >>> to http://localhost:8081/repository/site/.
...
Transfer error: java.io.IOException: Unable to create collection: http:// ↵
localhost:8081/repository/; status code = 400
Uploading: ../project-summary.html to http://localhost:8081/repository/ ↵
site/

##http://localhost:8081/repository/site/./project-summary.html - Status ↵
code: 201

Transfer finished. 5078 bytes copied in 0.075 seconds
http://localhost:8081/repository/site/ - Session: Disconnecting
http://localhost:8081/repository/site/ - Session: Disconnected
...
[INFO] BUILD SUCCESS
...
```

Once the site has been published, you can load the site in a browser by going to [http://localhost:8081/-repository/site/index.html](http://localhost:8081/repository/site/index.html).



Figure 13.1: Maven-Created Sample Site Hosted in a Raw Repository

Tip

A complete Maven project example can be found in the [documentation book examples](#).

13.4 Proxying and Grouping Raw Repositories

Beside the common use case using hosted raw repositories for site deployments, the repository manager supports proxying as well as grouping of raw repositories.

The creation follows the same process as documented in Section 4.3 using the *raw (proxy)* and the *raw (group)* recipes.

A raw proxy repository can be used to proxy any static website. This includes a Maven site hosted in a raw repository in another Nexus Repository Manager server or a plain static website hosted on another web server like Apache httpd. It can also be used to proxy directory structures exposed via a web server to distribute archives such as `https://nodejs.org/dist/`.

Note

No content is modified when proxied. This means that e.g., any absolute URL used with HTML document remain absolute and therefore bypass the proxying mechanism.

Grouping raw repositories is possible and can e.g., be used to aggregate multiple site repositories. However keep in mind that the raw format does not contain any logic to resolve conflicts between the different repositories in the group. Any request to the group causes the repository manager to check the member repositories in order and return the first matching content.

13.5 Uploading Files to Hosted Raw Repositories

Many other tools, besides using Maven, can be used to upload files to a hosted raw repository. A simple HTTP PUT can upload files. The following example uses the `curl` command and the default credentials of the `admin` user to upload a `test.png` file to a hosted raw repository with the name `documentation`.

An Example Upload Command Using curl:

```
curl -v --user 'admin:admin123' --upload-file ./test.png http://localhost ↵  
:8081/repository/documentation/test.png
```

After a completed upload the repository manager provides the file at the URL `http://localhost:8081/repository/documentation/test.png`. Using this approach in a script entire static websites or any other binary resources can be uploaded.

Chapter 14

REST and Integration API

Available in Nexus Repository OSS and Nexus Repository Pro

14.1 Introduction

Automation via scripts is a common scenario. The Nexus Repository Manager provides APIs that simplify provisioning and other tasks in the repository manager. These APIs can be invoked from scripts that are published to the repository manager and executed there.

14.2 Writing Scripts

The scripting language used on the repository manager is **Groovy**. Any editor can be used to author the scripts.

The available APIs are contained in a number of JAR files. All these files, including JavaDoc and Sources archives, are available from the **Central Repository**. They can be manually downloaded and extracted. E.g. the different versions and the specific JAR files for `org.sonatype.nexus:nexus-core` are available in versioned directories at `http://repo1.maven.org/maven2/org/sonatype/nexus/`

nexus-core/.

This manual process can be simplified and improved by the usage of a Maven project declaring the relevant components as dependencies. An example project with this setup called `nexus-script-example` and a few scripts are available in the [documentation examples project](#).

Maven Project `pom.xml` Declaring the API Dependencies for Scripting

```
<?xml version="1.0" encoding="UTF-8"?>
<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example.automation</groupId>
  <artifactId>nexus-script-demo</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <nx-version>3.0.2-02</nx-version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.sonatype.nexus</groupId>
      <artifactId>nexus-core</artifactId>
      <version>${nx-version}</version>
    </dependency>
    <dependency>
      <groupId>org.sonatype.nexus</groupId>
      <artifactId>nexus-script</artifactId>
      <version>${nx-version}</version>
    </dependency>
    <dependency>
      <groupId>org.sonatype.nexus</groupId>
      <artifactId>nexus-security</artifactId>
      <version>${nx-version}</version>
    </dependency>
    <dependency>
      <groupId>org.sonatype.nexus.plugins</groupId>
      <artifactId>nexus-script-plugin</artifactId>
      <version>${nx-version}</version>
    </dependency>
  </dependencies>
</project>
```

Development environments such as IntelliJ IDEA or Eclipse IDE can download the relevant JavaDoc and Sources JAR files to ease your development. Typically you would create your scripts in `src/main/groovy` or `src/main/scripts`.

The scripting API exposes specific tooling for IntelliJ IDEA that allows you to get access to code completion and similar convenience features, while writing your scripts in this Maven project. Currently the API exposes four main providers with numerous convenient methods:

- `core`
- `repository`
- `blobStore`
- `security`

The API is deliberately designed to be simple to use. It encapsulates complex configuration in single method invocations. Many of the included methods use default values that can be omitted. For example, the method to create a hosted repository using the Maven format in the simplest usage simply requires a name.

```
repository.createMavenHosted("private")
```

This method simply uses the default values for the rest of the parameters and is therefore equivalent to

```
repository.createMavenHosted("private", BlobStoreManager. ←  
    DEFAULT_BLOBSTORE_NAME, VersionPolicy.RELEASE,  
    WritePolicy.ALLOW_ONCE, LayoutPolicy.STRICT)
```

You can inspect the default values in the API documentation available by inspecting the declaration of the specific methods in your IDE or by viewing the JavaDoc.

In terms of overall complexity of the scripts created, it is best to break large tasks up into multiple scripts and therefore invocations.

14.3 Managing and Running Scripts

Once you have completed the creation of your script, you need to publish it to the repository manager for execution. This is done by REST API invocations against the endpoint:

```
http://localhost:8081/service/siesta/rest/v1/script
```

This endpoint accepts JSON-formatted payloads with your script as the content.

Example JSON formatted file maven.json with a simple repository creation script

```
{
  "name": "maven",
  "type": "groovy",
  "content": "repository.createMavenHosted('private')"
}
```

The JSON file `maven.json` located in the current directory can be published to the repository manager with an HTTP POST like

```
curl -v -X POST -u admin:admin123 --header "Content-Type: application/json" \
  "http://localhost:8081/service/siesta/rest/v1/script" -d @maven.json
```

A list of scripts stored on the repository manager can be accessed with

```
curl -v -X GET -u admin:admin123 'http://localhost:8081/service/siesta/ \
  rest/v1/script'
```

The same call with a script name appended returns the actual script content.

A script can be executed by sending a POST to the `run` method of the specific script

```
curl -v -X POST -u admin:admin123 --header "Content-Type: text/plain" ' \
  http://localhost:8081/service/siesta/rest/v1/script/maven/run'
```

A successful execution should result in a `HTTP/1.1 200 OK` result.

Scripts can be removed with a HTTP DELETE operation to the specific script:

```
curl -v -X DELETE -u admin:admin123 'http://localhost:8081/service/siesta/ \
  rest/v1/script/maven'
```

Scripts can receive run-time parameters via the REST API

```
curl -v -X POST -u admin:admin123 --header "Content-Type: text/plain" ' ↵  
  http://localhost:8081/service/siesta/rest/v1/script/ ↵  
  updateAnonymousAccess/run' -d 'false'
```

and receive them as arguments that have to be parsed by the script as desired

```
security.setAnonymousAccess (Boolean.valueOf (args) )
```

Interaction with the REST API for scripts can be done with any scripting language capable of HTTP calls as mentioned above. In the following section you can find some further detailed examples.

14.4 Examples

The API for scripts is capable of a number of different tasks. This section provides examples for script writing, publishing and executing them.

The `simple-shell-example` project in the [scripting section of the documentation examples project](#) includes a number of JSON file with simple scripts:

maven.json

simplest script to create a hosted Maven repository

npm.json

simple script to create a hosted and proxy repository as well as a repository group for npm usage

bower.json

simple script to create a hosted and proxy repository as well as a repository group for bower usage

anonymous.json

parameterized script to enable or disable anonymous access

Simple shell scripts are added to contain the curl invocations to manage scripts via the REST API:

create.sh

Upload a specified JSON file

delete.sh

Delete a script specified by its name

list.sh

List all deployed scripts

run.sh

Run a script specified by its name

setAnonymous.sh

Run the anonymous script on the server with the parameter `true` or `false`

update.sh

Update an existing script by specifying the name and the JSON file to use for the update

And example sequence of creating and running a script is:

```
./create.sh maven.json
./run.sh maven
```

Subsequently you could list all scripts and delete the maven script with

```
./list.sh
./delete.sh maven
```

Since scripts are typically longer than a single line and creating them in a separate file in the IDE is recommended, using a helper script that formats a `.groovy` file into a JSON file and submits it to the repository manager can be a convenient approach.

The `complex-script` project in the [scripting section of the documentation examples project](#) includes an example implementation using Groovy invoked from a shell script. All scripts in this folder can be published and executed via the `provision.sh` file. This results in the download of all required dependencies and the upload and execution of the referenced script. Alternatively you can provision the scripts individually:

```
groovy addUpdateScript.groovy -u "admin" -p "admin123" -n "raw" -f "↵
rawRepositories.groovy" -h "http://localhost:8081"
curl -v -X POST -u admin:admin123 --header "Content-Type: text/plain" "↵
http://localhost:8081/service/siesta/rest/v1/script/raw/run"
```

The following scripts are available:

npmAndBowerRepositories.groovy

configures a set of proxy and hosted repositories as well as repository groups for NPM and Bower repositories suitable for server-side and client JavaScript-based development

rawRepositories.groovy

creates a new blob store and uses it for a hosted raw repository

security.groovy

disables anonymous access, creates a new administrator account, creates a new role with a simple expansion to anonymous user role and a user, creates a new role with publishing access to all repositories and a user

core.groovy

configures the base URL capability and a proxy server

Logging from your scripts into the repository manager logs is automatically available and performed with the usual calls

```
log.info('User jane.doe created')
```

The result of the last script line is by default returned as a string. This can be a message as simple as *Success!* or more complex structured data. For instance, you can easily return JSON using built-in Groovy classes like:

```
return groovy.json.JsonOutput.toJson([result: 'Success!'])
```

which looks like

```
{
  "result": "Success!"
}
```

Passing parameters to the script can use JSON encoded arguments like

```
{
  "id": "foo",
  "name": "bar",
  "description": "baz",
  "privilegeIds": ["nx-all"],
  "roleIds": ["nx-admin"]
}
```

which in turn can be parsed using the `JsonSlurper` class in the script:

```
import groovy.json.JsonSlurper

//expects json string with appropriate content to be passed in
def role = new JsonSlurper().parseText(args)

security.addRole(role.id, role.name, role.description, role.privilegeIds, ↵
    role.roleIds)
```

You can read more about how to work with XML and JSON with Groovy on <http://groovy-lang.org/-processing-xml.html> and <http://groovy-lang.org/json.html>.

Chapter 15

Bundle Development

Available in Nexus Repository OSS and Nexus Repository Pro

15.1 Introduction

Nexus Repository Manager is built on top of the OSGi container [Apache Karaf](#). The supporting core infrastructure, known as Nexus platform, provides a foundation for these editions. The functionality is encapsulated in a number of OSGi bundles. Each edition is composed of a number of bundles, that provide the specific features.

Bundles can provide further functionality for the back-end such as support for new repository formats, specific behaviour for components, new tasks, 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 bundle.

This chapter provides a high level overview and information to begin developing your own bundles for the Nexus platform, and specifically the Nexus Repository Manager.

Knowledge of Apache Maven and Java are required for your bundle development efforts. OSGi-related knowledge is highly relevant and beneficial. Please ensure to consult the documentation for the relevant projects, when necessary.

If you work on any bundle development and require any help or assistance, please contact the development team:

- the [users mailing list](#)
- the [community chat channel](#)
- or via email to nexus-feedback@sonatype.com

**Warning**

This documentation is not complete and Sonatype encourages you to provide feedback to help us answer any questions you might have and improve this chapter as needed.

15.2 Installing Bundles

In order to have your features from your bundle available as part of the repository manager, the bundle needs to be loaded by the OSGi container.

The default build assembles multiple bundles into features that form the foundation of Nexus Repository Manager OSS and Nexus Repository Manager Pro. A number of these definitions can be found in [the assemblies](#) module. The supported distributions are defined in the modules `nexus-oss-feature` and `nexus-pro-feature`, which are part of the internal code-base.

An installation of the repository manager defines the feature it loads in `etc/org.sonatype.nexus.cfg` and additional features can be declared to be loaded there. E.g. to add `my-custom-feature` to an Nexus Repository Manager OSS installation you can change to

```
nexus-features=nexus-oss-feature,my-custom-feature
```

The feature `my-custom-feature` is a Maven project that includes the desired bundles as dependencies. Alternatively you can add a specific feature via Karaf commands.

Bundles can be loaded via the Karaf console. To enable the console, set `karaf.startLocalConsole` in `bin\nexus.vmoptions` to `true`. This allows you to access the Karaf console by pressing enter after starting the repository manager with the `run` option.

The `bundle:install` command can be used to load a bundle into the container.

For development usage, you can set the local Maven repository as the source for any bundle loading with e.g.

```
config:property-set -p org.ops4j.pax.url.mvn org.ops4j.pax.url.mvn. ↵  
    defaultRepositories "file:${user.home}/.m2/repository@id=system. ↵  
    repository@snapshots"
```

Once your bundle is installed will be display a part of the output from `bundle:list`. With the local Maven repository configured as a source, you can rebuild your bundle and get it reloaded and the repository completed restarted with

```
bundle:update 270  
bundle:refresh  
system:shutdown -f -r
```

This process ensures that bundles are updated, imports are correctly picking up any changes and the full repository manager runs through the full start-up life-cycle.

15.3 Bundle Development Overview

The preferred way to write bundles is to use Java as the implementation language and Apache Maven as the build system. The [public code-base of Nexus Repository Manager OSS](#) can be used as a starting point to investigate existing bundles and their source code. The easiest way to create a new bundle project is to replicate a bundle with a similar functionality. Inspect the source code of bundles with similar functionality, and read the JavaDoc documentation for the involved classes.

To gain access to all the components needed for your bundle development, you have to proxy the Sonatype grid repository with the URL:

```
https://repository.sonatype.org/content/groups/sonatype-public-grid/
```

Set up your project to include inheriting from the parent of all the Nexus Repository Manager OSS bundles with the version 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>3.0.0-SNAPSHOT</version>
</parent>
```

**Warning**

It is best to use the identical version of the parent as the Nexus Repository Manager instance on which you want to run your bundle. When developing a bundle you are using large parts of internals, which are subject to change from one version to another. This same logic applies to any dependencies as well.

A bundle Maven project creates a custom build output file in the form of an OSGi bundle. Enable this by changing the packaging to `bundle`. In addition, you need to add the `karaf-maven-plugin` and any needed dependencies. Inspect the `pom.xml` files for specific bundle in the `plugins` directory for further details.

These dependencies pull in a large number of transitive dependencies that expose Nexus Repository Manager functionality and other libraries to your project. Depending on the type of bundle 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 bundle development.

With the exception of interfaces and code that is required to be accessible from modules outside of the format bundle, all code should be nested within an *internal* directory that will be isolated by the OSGi run-time container.

Once you have created your Maven project as described above, you can build the bundle with `mvn clean install`.

15.4 Support for a New Repository Format

This chapter examines the efforts required to implement support for a new repository format in the Nexus Repository Manager. By default, the repository manager includes support for various repository formats including `raw-format`, `maven2-format` and others.

When considering to implement support for a new repository format, it is important to get a good understanding of the format itself as well as the tools and the community working with the format. It might

even be necessary to read and understand the source code of potential native, upstream implementations to support a format.

Following are a few questions that can provide some useful answers leading to a better understanding of the format and necessary steps for implementation;

- What is this format all about?
- What tools (client/server) are involved?
- What communication is performed between client and server?
- Do any protocols or specifications exist?
- What authentication method needs to be supported by the repository manager?
- How can the repository manager authenticate against a proxied remote server?
- How does the concepts of components and assets used in Nexus Repository Manager map to the format?
- What is the best way to map the component identifier of name, version and group to the format?
- What format specific attributes should be stored as components and assets?
- Is it necessary to rewrite proxied metadata? E.g. proxied metadata contains absolute URLs to proxied server that it has to rewrite to point to repository manager.
- Are there any special features that should be considered?

To provide sufficient support for users, a new repository format needs to include a number of features:

- proxying components from remote repositories
- storing and managing components in a hosted repository
- exposing multiple repositories to users as a single repository group
- format-specific search criteria and the related search functionality

Depending on the specific of the repository format being implemented a number of other features have to be provided or can optionally provide additional value to the user:

- any required tasks for maintenance of the repositories and their content
-

- client side tools to allow the standard tools to interact with the repositories on the repository manager
- custom features to display information about the repositories or their content in the user interface

The implementation of all these features for the `raw-format` can be found in the module `plugins/nexus-repository-raw`. The `raw` format is a good example code-base to expose as it presents the most simplistic repository format.

The Maven repository format as used by Apache Maven and many other tools is implemented in the `plugins/nexus-repository-maven` module. It can serve as another, slightly more complex, example. Examining the code base can be especially useful, if you know the Maven repository format.

15.4.1 Format, Recipe and Facet

Extending `Format` allows you define support for your new repository format. Proxy, hosted and group functionality are implemented in a corresponding `Recipe` implementation each. The recipe enables the system to configure the *view* of a repository. It configures the facets that decorate the implementation, and matches up routes with appropriate handlers. Some handlers like the `SecurityHandler` are required for all repositories, while others are used to implement format specific functionality like managing the content (i.e. `RawContentHandler`).

Facets are used to decorate the format and provide additional functionality like proxy support (e.g. `RawProxyFacet`).

Each format plugin is required to extend `RepositoryFormatSecurityConfigurationResource` to provide security configuration. This simple implementation can be used to enhance the security rules as necessary.

15.4.2 Storage

An addressable component in a repository is described as a `Component`. Typically it defines common metadata like name and version and acts as the parent for one or multiple assets. An `Asset` represents binary content of any type - usually a JAR file, ZIP archive or some other binary format and additional files associated with the package (i.e. `pom.xml` for maven). Some metadata is automatically collected for Assets, like check-sums, while each format can also contribute its own specific metadata. An asset should always have a `sha1` check-sum, but certain formats may require other types of check-sum and should extend the `Asset.attributes.checksum` map as required to store these.

15.4.3 User Interface

The user interface for supporting a new repository format is following a standard-pattern and is implemented as a recipe in the `nexus-coreui-plugin` bundle in `src/main/resources/static/rapture/NX/coreui/view/repository/recipe/`. These merely compose configuration for specific facets are implemented in `..../repository/facet`.

If a given format requires any additional specific configuration you have to add a new facet configuration screen with the required fields. They have to be mapped to the key/value map called `attributes` of the repository. E.g. a repository format `foo` has to be mapped to `attributes.foo.someConfigProperty`. New format configurations need to be registered in the `views` configuration of the controller in `../coreui/controller/Repositories.js`.

15.4.4 Tasks

Tasks are implemented for scheduled maintenance and similar task, that operate on a repository as a whole. The Maven repository bundle includes a number of tasks that can serve as an example in the `org.sonatype.nexus.repository.maven.tasks` package.

15.5 Contributing Bundles

Ideally any new bundles created, yields significant benefits for the overall community of users. Sonatype encourages contribution of such bundles to the upstream repository and is offering support and help for such efforts.

The minimum steps for such contributions are:

- Sign and submit a [contributor license agreement](#) to Sonatype
- Create a pull request with the relevant changes to the [nexus-public repository](#)

In further collaboration Sonatype will decide upon next steps on a case-by-case basis and work with you to

- Create sufficient tests
- Provide access to upstream repositories
- Facilitate other infrastructure such as CI server builds
- Help you with verification and testing
- Work with you on user documentation and outreach
- Expose your work to the user community
- And many others.

Appendix A

Contributing

The Nexus Repository Manager documentation 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 Sonatype values 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 this appendix.

Contributor License Agreement (CLA)

In order to contribute to the Nexus Repository Manager documentation, 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.

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.

Contributors The following people have been authors or contributors to the book in the past:

Manfred Moser, Tim O'Brien, Jason Van Zyl, Damian Bradicich, John Casey, Tamas Cservenak, Brian Demers, Brian Fox, Marvin Froeder, Anders Hammar, Rich Seddon, Peter Lynch, Juven Xu, Joe Tom, Jeff Wayman, Kelly Robinson, Dulani Wallace, Jeffry Hesse

Appendix B

Copyright

Copyright © 2011-Present 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 Derivate 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/>.