

Repository Management with Nexus

Ed. 4.0

Contents

1	Introducing Sonatype Nexus	1
1.1	Introduction	1
1.2	Nexus Open Source	1
1.2.1	Nexus Open Source Features	2
1.2.2	Nexus Open Source License	3
1.3	Nexus Professional	3
1.3.1	Nexus Professional Features	4
1.3.2	Nexus Professional License	5
1.4	Choosing a Nexus Edition	6
1.4.1	Use Nexus Open Source	6
1.4.2	Use Nexus Professional	7
1.5	History of Nexus	8

2 Component Lifecycle and Repository Management	9
2.1 Introduction	9
2.2 Component Lifecycle Management	9
2.2.1 Increasing Component Usage and Open Source Components	10
2.2.2 Security Vulnerability and License Compliance Risks	10
2.2.3 Nexus and Component Lifecycle Management	10
2.3 Repository Management	11
2.3.1 Proxying Public Repositories	11
2.3.2 Managing Releases and Snapshots	12
2.3.3 Getting Control of Dependencies	12
2.3.4 A Nexus for Collaboration	13
2.4 What is a Repository?	13
2.4.1 Release and Snapshot Repositories	14
2.4.2 Repository Coordinates	15
2.4.3 Addressing Resources in a Repository	16
2.4.4 The Maven Central Repository	16
2.5 What is a Repository Manager	17
2.5.1 Core Capabilities of a Repository Manager	18

2.5.2	Additional Features of a Repository Manager	19
2.6	Reasons to Use a Repository Manager	20
2.6.1	Speed Up Your Builds	20
2.6.2	Save Bandwidth	20
2.6.3	Ease the Burden on Central	21
2.6.4	Gain Predictability and Scalability	21
2.6.5	Control and Audit Dependencies and Releases	21
2.6.6	Deploy 3rd Party Artifacts	22
2.6.7	Collaborate with Internal Repositories	22
2.6.8	Distribute with Public Repositories	22
2.7	Adopting a Repository Manager	23
2.7.1	Stage Zero: Before Using a Repository Manager	23
2.7.2	Stage One: Proxying Remote Repositories	24
2.7.3	Stage Two: Hosting a Repository Manager	24
2.7.4	Stage Three: Continuous Collaboration	26
2.7.5	Stage Four: Life-cycle Integration	26
3	Installing and Running Nexus	28
3.1	Nexus Prerequisites	28

3.2	Downloading Nexus	29
3.2.1	Downloading Nexus Open Source	29
3.2.2	Downloading Nexus Professional	31
3.3	Installing Nexus	32
3.4	Upgrading Nexus	33
3.5	Running Nexus	34
3.6	Post-Install Checklist	38
3.6.1	Step 1: Change the Administrative Password and Email Address	38
3.6.2	Step 2: Configure the SMTP Settings	38
3.6.3	Step 3: Configure Default HTTP Proxy Setting	39
3.6.4	Step 4: Enable Remote Index Downloads	39
3.6.5	Step 5: Change the Deployment Password	40
3.6.6	Step 6: If necessary, set the LANG Environment Variable	40
3.6.7	Step 7: Configure Routes	40
3.7	Configuring Nexus as a Service	40
3.7.1	Running as a Service on Linux	41
3.7.1.1	Add Nexus as a Service on Red Hat, Fedora, and CentOS	41
3.7.1.2	Add Nexus as a Service on Ubuntu and Debian	42

3.7.2	Running as a Service on Mac OS X	42
3.7.3	Running as a Service on Windows	43
3.8	Running Nexus Behind a Proxy	43
3.9	Installing the Nexus WAR	45
3.10	Installing a Nexus Professional License	46
3.10.1	License Expiration	49
3.11	Nexus Directories	49
3.11.1	Sonatype Work Directory	49
3.11.2	Nexus Configuration Directory	53
4	Configuring Maven to Use Nexus	55
4.1	Introduction	55
4.2	Configuring Maven to Use a Single Nexus Group	55
4.3	Adding Custom Repositories for Missing Dependencies	57
4.4	Adding a New Repository	57
4.5	Adding a Repository to a Group	59
5	Using the Nexus User Interface	62
5.1	Introduction	62
5.2	Browsing Repositories	63

5.2.1	Viewing the Artifact Information	65
5.2.2	Viewing the Maven Information	66
5.2.3	Using the Artifact Archive Browser	67
5.2.4	Viewing the Artifact Dependencies	68
5.2.5	Viewing the Artifact Insight Data	69
5.3	Browsing Groups	71
5.4	Searching for Artifacts	73
5.4.1	Search Overview	73
5.4.2	Advanced Search	76
5.4.3	Nexus OpenSearch Integration	77
5.5	Uploading Artifacts	79
5.6	Browsing System Feeds	81
5.7	System Files	83
5.8	Working with Your User Profile	84
5.8.1	Changing Your Password	85
5.8.2	Additional User Profile Tabs	86
5.9	Filing a Problem Report	86
6	Configuring Nexus	88

6.1	Customizing Server Configuration	88
6.1.1	SMTP Settings	89
6.1.2	HTTP Request Settings	90
6.1.3	Security Settings	90
6.1.4	Error Reporting Settings	91
6.1.5	Application Server Settings	93
6.1.6	Default HTTP Proxy Settings	94
6.1.7	System Notification Settings	95
6.1.8	PGP Key Server Information	96
6.1.9	New Version Availability	96
6.2	Managing Repositories	97
6.2.1	Proxy Repository	97
6.2.2	Hosted Repository	97
6.2.3	Virtual Repository	98
6.2.4	Configuring Repositories	98
6.2.5	Selecting Mirrors for Proxy Repositories	104
6.2.6	Adding a Mirror Entry for a Hosted Repository	105
6.2.7	Viewing Repository Summary Panel	106

6.2.8	Accessing The Central Repository Securely	107
6.2.9	Auto Block/Unblock of Remote Repositories	110
6.3	Managing Groups	110
6.4	Managing Routing	111
6.4.1	Automatic Routing	112
6.4.2	Manual Routing Configuration	114
6.5	Managing Scheduled Tasks	118
6.6	Accessing and Configuring Capabilities	123
6.7	Managing Security	125
6.8	Managing Privileges	126
6.9	Managing Repository Targets	128
6.10	Managing Roles	131
6.11	Managing Users	135
6.12	Network Configuration	137
6.13	Nexus Logging Configuration	138
6.14	Nexus Plugins and REST Interfaces	139
6.15	Security Setup with User Tokens	140
6.15.1	Introduction	140

6.15.2 Enabling and Resetting User Tokens	141
6.15.3 Accessing and Using Your User Tokens	143
6.15.4 Configuring User Token Behaviour	146
7 Nexus Smart Proxy	147
7.1 Introduction	147
7.2 Enabling Smart Proxy Publishing	148
7.3 Establishing Trust	149
7.4 Repository Specific Smart Proxy Configuration	150
7.5 Smart Proxy Security and Messages	152
7.6 Example Setup	153
8 Nexus LDAP Integration	155
8.1 Introduction	155
8.2 Enabling the LDAP Authentication Realm	155
8.3 Configuring Nexus LDAP Integration	157
8.4 Connection and Authentication	157
8.5 User and Group Mapping	159
8.6 Mapping Users and Groups with Active Directory	163
8.7 Mapping Users and Groups with posixAccount	165

8.8	Mapping Roles to LDAP Users	166
8.9	Mapping Nexus Roles for External Users	167
8.10	Mapping External Roles to Nexus Roles	171
8.11	Enterprise LDAP Support	174
8.11.1	Enterprise LDAP Fail-over Support	175
8.11.2	Support for Multiple Servers and LDAP Schemas	176
8.11.3	Enterprise LDAP Performance Caching and Timeout	177
8.11.4	User and Group Templates	178
8.11.5	Testing a User Login	179
9	Nexus Procurement Suite	180
9.1	Introduction	180
9.2	The Stages of Procurement	181
9.3	Two Approaches to Procurement	181
9.3.1	Procured Development Repository	182
9.3.2	Providing Access With A Repository Group	183
9.4	Setting up a Procured Repository	183
9.4.1	Enable Remote Index Downloads	184
9.4.2	Create a Hosted Repository	186

9.4.3 Configuring Procurement for Hosted Repository	187
9.4.4 Procured Repository Administration	189
9.5 Configuring Procurement	190
9.6 Stopping Procurement	196
10 Build Promotion with the Nexus Staging Suite	198
10.1 Introduction	198
10.1.1 Releasing Software without a Staging Repository	199
10.1.2 How the Staging Suite Works	199
10.2 Configuring the Nexus Staging Suite	201
10.2.1 Overview	201
10.2.2 Configuring a Staging Profile	202
10.2.3 Configuring Build Promotion Profiles	206
10.2.4 Staging Related Security Setup	207
10.2.5 Using Repository Targets for Staging	211
10.3 Configuring your Project for Deployment	211
10.3.1 Deployment with the Nexus Staging Maven Plugin	211
10.3.2 Deployment with the Nexus Staging Ant Tasks	221
10.3.3 Deployment with the Maven Deploy Plugin	224

10.3.4 Manually Uploading a Staged Deployment in Nexus	226
10.4 Managing Staging Repositories in Nexus	227
10.4.1 Closing an Open Repository	231
10.4.2 Using the Staging Repository	233
10.4.3 Releasing a Staging Repository	234
10.4.4 Promoting a Staging Repository	234
10.4.5 Releasing, Promoting, and Dropping Build Promotion Profiles	236
10.4.6 Multi-level Staging and Build Promotion	238
10.5 Enforcing Standards for Deployment and Promotion with Rulesets	239
10.5.1 Managing Staging Rulesets	239
10.5.2 Defining Rulesets for Promotion	241
11 Repository Health Check	243
11.1 Analyzing a Repository with Repository Health Check	243
11.2 Accessing the Detailed Repository Health Check Report	245
11.3 Using Repository Health Check Results For Component Lifecycle Management	248
11.3.1 Example: Analyzing a Spring Beans Vulnerability	248
11.3.2 Example: Resolving a License Issue	250
12 Sonatype CLM Integration	251

12.1 Introduction	251
12.2 Connecting Nexus to CLM Server	252
12.3 Configuring the CLM Server	253
12.4 Using CLM for Staging	253
13 Managing Maven Settings	257
13.1 Introduction	257
13.2 Manage Maven Settings Templates	257
13.3 Nexus M2Settings Maven Plugin	261
13.3.1 Running the Nexus M2Settings Maven Plugin	261
13.3.2 Configuring Nexus M2Settings Maven Plugin	262
13.3.3 Downloading Maven Settings	264
13.4 Summary	265
14 OSGi Bundle Repositories	267
14.1 Introduction	267
14.2 Proxy OSGi Bundle Repositories	268
14.3 Hosted OSGi Bundle Repositories	270
14.4 Virtual OSGi Bundle Repositories	271
14.5 Grouping OSGi Bundle Repositories	272

15 P2 Repositories	274
15.1 Introduction	274
15.2 Proxy P2 Repositories	274
15.3 Grouping P2 Repositories	275
16 .NET Package Repositories	277
16.1 Introduction	277
16.2 NuGet Proxy Repositories	278
16.3 NuGet Hosted Repositories	280
16.4 NuGet Virtual Repositories	281
16.5 NuGet Group Repositories	282
16.6 Accessing Packages in Repositories and Groups	283
16.7 Deploying Packages to NuGet Hosted Repositories	284
16.7.1 Creating a NuGet API-Key	284
16.7.2 Creating a Package for Deployment	284
16.7.3 Deployment with the NuPkg Upload User Interface	285
16.7.4 Command line based Deployment to a Nexus NuGet Hosted Repository	285
16.8 Integration of Nexus NuGet Repositories in Visual Studio	285
17 Deploying Sites to Nexus	287

17.1 Introduction	287
17.2 Creating a New Maven Project	287
17.3 Configuring Maven for Site Deployment	289
17.4 Adding Credentials to Your Maven Settings	290
17.5 Creating a Site Repository	291
17.6 Add the Site Deployment Role	293
17.7 Publishing a Maven Site to Nexus	293
18 Artifact Bundles	296
18.1 Introduction	296
18.2 Creating an Artifact Bundle from a Maven Project	297
18.3 Uploading an Artifact Bundle to Nexus	300
19 Nexus Best Practises	302
19.1 Introduction	302
19.2 Repositories per Project/Team	302
19.3 Partition Shared Repositories	303
19.3.1 Selecting an Approach	303
20 Using Nexus Plugins	305

20.1	Installing Additional Plugins	305
20.2	Nexus Outreach Plugin	306
20.3	Custom Metadata Plugin	306
20.3.1	Viewing Artifact Metadata	306
20.3.2	Editing Artifact Metadata	307
20.3.3	Searching Artifact Metadata	308
20.4	User Account Plugin	310
20.4.1	Installing the User Account Plugin	311
20.4.2	Configuring the User Account Plugin	311
20.4.3	Signing Up for an Account	312
20.4.4	Manual Activation of New Users	314
20.4.5	Modifying Default User Permissions	315
20.5	Nexus Atlassian Crowd Plugin	317
20.5.1	Installing the Crowd Plugin	317
20.5.2	Configuring the Crowd Plugin	317
20.5.3	Crowd Access Settings	319
20.5.3.1	Crowd HTTP Settings	320
20.5.3.2	Crowd HTTP Proxy Settings	320

20.5.3.3	Miscellaneous Settings	320
20.5.4	Adding the Crowd Authentication Realm	321
20.5.5	Configuring a Nexus Application in Crowd	322
20.5.6	Mapping Crowd Groups to Nexus Roles	324
20.5.7	Adding a Crowd Role to a Nexus User	326
20.6	Nexus Branding Plugin	329
21	Developing Nexus Plugins	330
21.1	Nexus Plugins	331
21.1.1	Nexus Plugin API	331
21.2	Nexus Extension Points	332
21.3	Nexus Plugin Extension Points	332
21.3.1	Nexus Plugin Extension	333
21.3.2	Nexus Index HTML Customizer	333
21.3.3	Static Plugin Resources	333
21.3.4	Plugin Templates	334
21.3.5	Event Inspectors	334
21.3.6	Content Generators	334
21.3.7	Content Classes	335

21.3.8 Storage Implementations	335
21.3.9 Repository Customization	335
21.3.10 Item and File Inspectors	335
21.3.11 Nexus Feeds	336
21.3.12 Nexus Tasks and Task Configuration	336
21.3.13 Application Customization	336
21.3.14 Request Processing	337
21.4 Using the Nexus Plugin Archetype	337
21.5 Set the Target Nexus Version	339
21.6 Building a Nexus Plugin Project	340
21.7 Creating a Complex Plugin	341
21.8 Nexus Plugin Descriptor Maven Plugin	345
21.9 The Nexus Plugin Descriptor	345
21.10 Defining Custom Repository Types	346
22 Migrating to Nexus	348
22.1 Migrating from Archiva	348
22.1.1 Introduction	348
22.1.2 Migrating Archiva Repositories	349

22.1.3 Migrating an Archiva Managed Repository	349
22.1.4 Migrating an Archiva Proxy Connector	353
22.2 Migrating from Artifactory	359
23 Configuring Nexus for SSL	361
23.1 Introduction	361
23.2 SSL Client Certificates	362
23.2.1 SSL Certificate Management	362
23.2.2 Proxying SSL Secured Repositories	364
23.2.3 Manually Configuring Trust Stores	365
23.2.3.1 Import the Server SSL Chain	367
23.2.3.2 Import the Client SSL Key/Certificate Pair	367
23.2.3.3 Configuring Nexus Start-up	368
23.3 Configuring Nexus to Serve SSL	368
23.3.1 Configure the Java Keystore	369
23.3.2 Configure Nexus/Jetty to Use the New Keystore	370
23.3.3 Modify the application-port for SSL connections	370
23.4 Redirecting Non-SSL Connections to SSL	371
24 Evaluating Nexus Step by Step	374

24.1 Prerequisites And Preparation	374
24.1.1 A Note About The Operating System	374
24.1.2 Java Runtime	375
24.1.3 Apache Maven	375
24.1.4 Gradle	376
24.1.5 Apache Ant And Apache Ivy	376
24.1.6 Microsoft Visual Studio And NuGet	377
24.2 Getting Started	377
24.2.1 Activating Your Nexus Trial	378
24.2.2 Logging Into Nexus As An Administrator	379
24.2.3 Getting Started With Your Nexus Professional Evaluation	379
24.3 The Basics: Proxying And Publishing	380
24.3.1 Proxying Components	380
24.3.2 Publishing Components	383
24.4 Governance	387
24.4.1 Identify Insecure OSS Components In Nexus	387
24.4.2 Track Your Exposure To OSS Licenses	389
24.4.3 Component Procurement	391

24.5 Process Improvements	393
24.5.1 Grouping Repositories	393
24.5.2 Staging A Release With Nexus	394
24.5.3 Hosting Project Web Sites	396
24.5.4 Process and Security Improvements With Maven Settings Management And User Token	397
24.6 .NET Integration	399
24.6.1 Consume .NET Components From NuGet Gallery	399
24.6.2 Publish And Share .NET Components With NuGet	401
24.7 Security	402
24.7.1 Integration With Enterprise LDAP Solutions	402
24.7.2 Single Sign On (SSO) Support With Atlassian Crowd	403
24.8 Enterprise Deployments	404
24.8.1 Scaling Nexus Deployments For Distributed Development	404
A Contributing to the Nexus Book	406
A.1 Contributor License Agreement (CLA)	406
A.2 Contributors, Authors, and Editors	407
A.3 How to Contribute	407

B Copyright	408
C Creative Commons License	410
C.1 Creative Commons BY-NC-ND 3.0 US License	411
C.2 Creative Commons Notice	415

List of Figures

3.1	Downloading Nexus Open Source	30
3.2	Selecting a Specific Version of Nexus Open Source to Download	31
3.3	Nexus Trial Activation Form	35
3.4	Nexus License Activation	35
3.5	Nexus Professional Evaluation Welcome Screen	36
3.6	Nexus Login Dialog (default login/password is admin/admin123)	36
3.7	Nexus Application Window	37
3.8	Nexus Professional Licensing Panel	46
3.9	Nexus Professional End-user License Agreement	47
3.10	License Upload Finished Dialog	48
3.11	Uninstall License Confirmation Dialog	48
3.12	License Uninstall Completed Dialog	49

3.13 The Sonatype Work Directory	50
4.1 Creating a New Proxy Repository	58
4.2 Configuring a Proxy Repository	58
4.3 Adding New Repositories to a Nexus Group	60
5.1 Nexus Interface for Anonymous Users	63
5.2 Browsing a Repository Storage	64
5.3 Browsing a Repository Index	65
5.4 Viewing the Artifact Information	66
5.5 Viewing the Maven Information	67
5.6 Using the Archive Browser	68
5.7 View an Artifact's Dependencies	69
5.8 Insight Data Displaying Security Vulnerabilities for an Old Version of Jetty	70
5.9 Common Vulnerabilities and Exposures CVE Entry for a Jetty Security Issue	71
5.10 Open Source Vulnerability DataBase OSVDB Entry for a Jetty Security Issue	71
5.11 Browsing a Nexus Group	72
5.12 Browsing a Nexus Group Index	73
5.13 Results of an Artifact Search for "junit"	74
5.14 Sort and Column Options in the Search Results Table	75

5.15 Advanced Search Results for a GAV Search Activated by the Show All Versions Link	76
5.16 Configuring Nexus as an OpenSearch Provider	78
5.17 OpenSearch Search Results in Nexus	78
5.18 Nexus Available as an Option in the Firefox OpenSearch Provider List	79
5.19 Artifact Upload Form	80
5.20 Browsing Nexus System Feeds	81
5.21 Browsing Nexus Logs and Configuration	83
5.22 Selecting the Update Frequency when Tailing a Log File	84
5.23 Drop Down on User Name with Logout and Profile Options	84
5.24 Summary Section of the Profile Tab	85
5.25 Changing Your Nexus Password	85
5.26 Generating a Nexus Problem Report	87
6.1 Administration Menu in the Left Hand Panel	89
6.2 Administration SMTP Settings	90
6.3 Administration HTTP Request Settings	90
6.4 Administration Security Settings	91
6.5 Administration Error Reporting Settings	91
6.6 Sonatype Issue Tracker	92

6.7	Signing Up for a Sonatype Issue Tracker Account	93
6.8	Administration Application Server Settings	94
6.9	Administration Default HTTP Proxy Settings	95
6.10	Administration System Notification Settings	95
6.11	Administration PGP Key Server Information	96
6.12	Administration New Version Availability	96
6.13	Repository Configuration Screen for a Proxy Repository	99
6.14	Repository Configuration Screen for a Proxy Repository	101
6.15	Repository Configuration Access Settings for a Hosted Repository	101
6.16	Configuring Mirrors for Proxy Repositories	105
6.17	Repository Summary Panel for a Hosted Repository	106
6.18	Repository Summary Panel for a Proxy Repository	106
6.19	Repository Summary Panel for a Virtual Repository	107
6.20	Default Configuration for the Central Repository Using HTTPS	108
6.21	Secure Central Capability	109
6.22	Configuring Remote Repository Auto Block/Unblock	110
6.23	Group Configuration Screen in Nexus	111
6.24	Automatic Routing for a Hosted Repository	112

6.25 Automatic Routing for a Proxy Repository	113
6.26 Routing Configuration Screen in Nexus	115
6.27 Managing Nexus Scheduled Tasks	118
6.28 Capabilities Management Interface with the Outreach Management Details Visible	124
6.29 Managing Security Privileges	126
6.30 Managing Security Privileges	127
6.31 Create, Delete, Read, and Update Privileges Created	128
6.32 Managing Repository Targets	129
6.33 Excluding Source Artifacts from a Repository Targets	130
6.34 Viewing the List of Defined Roles	131
6.35 Creating a New Role	132
6.36 Viewing an Internal Role	133
6.37 Managing Security Roles	134
6.38 Managing Users	135
6.39 Adding Roles to a User	136
6.40 Nexus User Role Tree	136
6.41 Nexus User Privilege Trace	137
6.42 The Log Configuration Panel	138

6.43	Plugin Console	139
6.44	Documentation Website for the Core API	140
6.45	User Token Administration Tab Panel	141
6.46	Selected Realms Server Security Settings with User Token Realm activated	142
6.47	User Token Reset for Specific User in Security Users Administration	143
6.48	User Token Panel for the Logged in Users in the Profile Section	144
6.49	Accessing the User Token Information	145
7.1	Global Configuration for Smart Proxy	148
7.2	Copying a Certificate	150
7.3	Adding a Trusted Certificate	150
7.4	Smart Proxy Settings for a Hosted Repository	151
7.5	Smart Proxy Settings for a Proxy Repository	151
7.6	Subscription with Smart Proxy Connected	152
7.7	Deployment Scenario for a Smart Proxy Use Case	153
8.1	Adding the LDAP Authentication Realm to Available Realms	156
8.2	Move the LDAP Authentication Realm after the XML Realms	156
8.3	Enterprise LDAP Option in the Security Menu	157
8.4	A Simple LDAP Connection and Authentication Setup	158

8.5	User & Group Templates Selection Drop Down	160
8.6	User Element Mapping	161
8.7	Dynamic Group Element Mapping	162
8.8	Static Group Element Mapping	162
8.9	Checking the User and Group Mapping in LDAP Configuration	166
8.10	Viewing All Configured Users	168
8.11	All Default Realm Users	168
8.12	All LDAP Users	169
8.13	Search LDAP Users	170
8.14	Mapping the Deployment Role to an External User	171
8.15	Selecting External Role Mapping in the Role Management Panel	172
8.16	Selecting an Externally Managed Role to Map to a Nexus Role	173
8.17	Mapping an External Role to a Nexus Role	174
8.18	Defining Multiple LDAP Servers in Nexus Professional	175
8.19	Use Multiple LDAP Servers in a Fail-over Scenario	176
8.20	Supporting Multiple LDAP Schemas with Nexus Professional	177
8.21	Setting the LDAP Query Cache Duration (in Seconds)	177
8.22	Setting the LDAP Connection Timeout (in Seconds)	178

8.23 Using User & Group Mapping Templates	178
8.24 Testing a User Login	179
8.25 Supply a User's Login Credentials	179
9.1 Procurement to a Certified Release Repository	182
9.2 Procurement to a Certified Development Repository	182
9.3 Enabling Remote Index Downloads for a Proxy Repository	185
9.4 Verification that the Remote Index has been Downloaded	186
9.5 Adding the "Approved From Central" Hosted Repository	187
9.6 Adding a Procured Repository	188
9.7 Configuring Procurement for a Hosted Repository	188
9.8 Hosted Repository is a Nexus Managed Proxy Repository while Procurement is Active	189
9.9 Viewing a Repository in the Artifact Procurement Interface	190
9.10 Applying a Rule to a Component Folder for <code>org/eclipse/aether</code>	191
9.11 Approving <code>org.eclipse.aether</code> Components	191
9.12 Accessing the Global Repository Configuration	192
9.13 Procurement Configurations Options for a Specific Component Version	193
9.14 Procurement Repository Tree View with Rule Visualization	194
9.15 Applied Rules for the Complete Procurement Repository	195

9.16 Adding a Freeform Rule	195
9.17 Stopping Procurement for a Procured Repository	197
10.1 Release Deployment Without the Nexus Staging Suite	199
10.2 Release Deployment with the Nexus Staging Suite	200
10.3 The Stages of a Staging Repository starting with Deployment and Ending with a Release or a Drop of the Repository	201
10.4 Adding a Staging Profile	202
10.5 Creating a New Staging Profile	203
10.6 Multi-level Staging and Build Promotion	206
10.7 Configuring a Build Promotion Profile	207
10.8 Adding a Role to a User	209
10.9 Available Roles for Staging with a Test Staging Profile	210
10.10 Uploading a Staged Deployment in Nexus	226
10.11 Staging Repositories List Panel	228
10.12 List of Activities Performed on a Promoted Staging Repository	230
10.13 Details of an Open Staging Repository as Displayed under the List of Staging Repositories	231
10.14 Confirmation and Description Dialog for Closing a Staging Repository	232
10.15 Viewing Nexus Managed Repositories	233

10.16 Confirmation Dialog for Releasing a Staging Repository	234
10.17 Confirmation Dialog for Promoting a Staging Repository	235
10.18 A Build Promotion Repository and its Members Panel	236
10.19 Releasing, Promoting, and Dropping Build Promotion Profiles	237
10.20 Promoting Multiple Repositories to the Same Build Promotion Profile	238
10.21 Multi-level Staging and Build Promotion	239
10.22 Creating a Staging Ruleset	240
10.23 Associating a Staging Ruleset with a Staging Profile	242
11.1 The Repositories List with Different Quality Status Indicators and Result Counts	244
11.2 A Result Summary Window for a Repository Health Check	244
11.3 Summary of the Detailed Repository Health Check Panel	245
11.4 The Security Data in the Detailed Repository Health Check Report	246
11.5 The License Data in the Detailed Repository Health Check Report	247
11.6 Insight tab for org.springframework:spring-beans:2.5.4	248
11.7 Viewing Multiple Versions of org.springframework:spring-beans:x	249
11.8 Viewing License Analysis Results for Hibernate	250
12.1 CLM configuration tab in Nexus	252
12.2 Staging and Release Configuration for a Policy in the CLM Server	254

12.3 Staging Profile with a CLM Application Configured	255
12.4 Staging Repository Activity with a CLM Evaluation Failure and Details	256
13.1 The Maven Settings Panel	258
14.1 Creating an OSGi Bundle Proxy Repository	269
14.2 Creating a Hosted OSGi Bundle Repository	271
14.3 Creating a Virtual OSGi Bundle Repository from a Maven Repository	272
14.4 Creating a new OSGi Bundle Repository Group	273
15.1 Creating a P2 Proxy Repository	275
15.2 Creating a new P2 Repository Group	276
16.1 NuGet Proxy Repository Configuration for nuget.org	278
16.2 NuGet Gallery with Package Source URL	279
16.3 NuGet Proxy Repository Scheduled Task	279
16.4 Example Configuration for a NuGet Hosted Repository for Release Packages	280
16.5 The NuPkg Upload Panel for a Hosted NuGet Repository	281
16.6 A Virtual NuGet Repository for the Releases Repository	282
16.7 A Public Nuget Group Combining a Proxy and Two Hosted Repositories	283
16.8 Viewing and Resetting the NuGet API Key in the NuGet Configuration Tab	284

16.9 Package Source Configuration for the Package Manager in Visual Studio to Access A Nexus NuGet Repository Group	286
17.1 Adding a Hosted Repository	291
17.2 Creating a New Maven Site Repository	292
17.3 Newly Created Site Repository	292
17.4 Adding the Site Deployment Role to the Deployment User	293
17.5 Sample Site Maven Project Web Site	295
18.1 Build Promotion Menu	300
18.2 Uploading an Artifact Bundle	300
18.3 Staging Repository Created from Artifact Bundle Upload	301
20.1 Viewing Artifact Metadata	307
20.2 Editing Artifact Metadata	308
20.3 Searching Artifact Metadata	309
20.4 Metadata Search Results for Custom Metadata	309
20.5 Metadata Search Results for Custom Metadata	310
20.6 Configuring the User Account Plugin	311
20.7 Nexus Sign Up Form	312
20.8 Nexus Sign Up Confirmation	313

20.9 Nexus Activation Email	313
20.10 Manually Activating a Signed Up User	315
20.11 User Interface with only the Base UI Privileges	316
20.12 Selecting Default Roles for New Users	316
20.13 Crowd Menu Link in the Security Section of the Nexus Menu	318
20.14 Crowd Configuration Panel	318
20.15 Crowd Access Settings	319
20.16 Crowd HTTP Settings	320
20.17 Crowd HTTP Proxy Settings	320
20.18 Crowd Miscellaneous Settings	321
20.19 Configuring the Crowd Authentication Realm	321
20.20 Creating a Nexus Crowd Application	322
20.21 Creating a Nexus Crowd Application Connection	323
20.22 Creating a Nexus Crowd Application Directories	323
20.23 Creating a Nexus Crowd Application Authorization	324
20.24 Adding an External Role Mapping	325
20.25 Mapping an External Crowd Group to a Nexus Role	325
20.26 Two Crowd Groups Mapped to Nexus Roles	326

20.27 Crowd Groups for User "brian"	327
20.28 Adding an External User Role Mapping	327
20.29 Locating a Crowd User in the User Role Mapping Dialog	328
20.30 Adding a Nexus Role to a Crowd User	328
22.1 Archiva Managed Repositories	350
22.2 Editing an Archiva Managed Repository	351
22.3 Creating a Nexus Hosted Repository	352
22.4 Rebuilding the Index of a Nexus Hosted Repository	353
22.5 Browsing Archiva Remote Repositories	354
22.6 Archiva Proxy Connectors	355
22.7 Archiva Proxy Connector Settings	356
22.8 Creating a Nexus Proxy Repository	357
22.9 Adding a Proxy Repository to a Repository Group	358
22.10 Defining Nexus Repository Groups	359
23.1 SSL Certificates Administration	362
23.2 Certificate Details Displayed After Successful Retrieval	363
23.3 Providing a Certificate in PEM Format	364
23.4 SSL Tab for a Proxy Repository with Remote Server Using HTTPS	365

24.1 Nexus User Interface With Login	379
24.2 Successfully Deployed Components In The Snapshots Repository	385
24.3 Repository Heath Check Summary	388
24.4 Security Vulnerability Summary Display From Repository Health Check	389
24.5 License Analysis Summary Display From Repository Health Check	390
24.6 Repository Health Check Details With License Issues List	391
24.7 Closing A Staging Repository In Nexus User Interface	395
24.8 NuGet Repositories In Repository List	400

List of Tables

8.1	Connection and Authentication Configuration for Active Directory	164
8.2	User Element Mapping Configuration for Active Directory	164
8.3	Group Element Mapping Configuration for Active Directory	164
8.4	User Element Mapping Configuration for posixAccount	165
8.5	Group Element Mapping Configuration for posixGroup	165
24.1	Commandline Invocation Examples	375

Preface

This book covers the concepts of component lifecycle and repository management in general and specifically usage of Sonatype Nexus Open Source and Sonatype Nexus Professional. It details all aspects of setup and running Nexus with the features of the latest release version 2.5.

This book was last updated and published on 2013-07-04.

Chapter 1

Introducing Sonatype Nexus

1.1 Introduction

Nexus manages software "artifacts" required for development, deployment, and provisioning. If you develop software, Nexus can help you share those artifacts with other developers and end-users. Maven's central repository has always served as a great convenience for users of Maven, but it has always been recommended to maintain your own repositories to ensure stability within your organization. Nexus greatly simplifies the maintenance of your own internal repositories and access to external repositories. With Nexus you can completely control access to, and deployment of, every artifact in your organization from a single location.

1.2 Nexus Open Source

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

1.2.1 Nexus Open Source Features

Hosting Repositories

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

Proxy Remote Repositories

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

Repository Groups

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

Hosting Project Web Sites

Nexus is a publishing destination for project web sites. While you very easily generate a project web site with Maven, without Nexus, you will need to set up a WebDAV server and configure both your web server and build with the appropriate security credentials. With Nexus, you can deploy your project's web site to the same infrastructure that hosts the project's build output. This single destination for binaries and documentation helps to minimize the number of moving parts in your development environment. You don't have to worry about configuring another web server or configuring your builds to distribute the project site using a different protocol, you simple point your project at Nexus and deploy the project site.

Fine-grained Security Model

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

Flexible LDAP Integration

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

Artifact Search

Nexus Open Source provides an intuitive search feature which allows you to search for software ar-

tifacts by identifiers such as groupId, artifactId, version, classifier, and packaging, names of classes contained in Java archives, keywords, and artifact checksums. Nexus search makes use of the industry standard for repository indexes, the Nexus Index format, and Nexus will automatically download a Nexus index from all remote repositories which create a Nexus index. Nexus will also automatically expose a Nexus index for any hosted repositories you create.

Scheduled Tasks

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

REST Services

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

Nexus Plugins

Nexus Open Source provides a rich API for extension in the form of Nexus Plugins. When you write a Nexus Plugin, you can customize REST services, the Nexus UI, repository formats, or write components that can intercept requests and add new capabilities to the platform. The plugin API which you have access to in Nexus Open Source is the same plugin API that is used to implement value-added features available in Nexus Professional.

Integration with m2eclipse

When you use Nexus as a repository manager it creates indexes that support some of the next-generation tools available in m2eclipse - Sonatype's Maven plugin for the Eclipse IDE. If you publish new artifacts and archetypes to Nexus, they are immediately available to m2eclipse project creation wizards and are included in m2eclipse search results.

1.2.2 Nexus Open Source License

Nexus Open Source is made available under the Eclipse Public License version 1.0. The text of this license is available from the Open Source Initiative (OSI) here: <http://www.opensource.org/licenses/eclipse-1.0.php>

1.3 Nexus Professional

Nexus Professional was designed to meet the needs of the enterprise. It is a central point of access to external repositories which provides the necessary controls to make sure that only approved artifacts enter

into your software development environment. It is also a central distribution point with the intelligence required to support the decision that go into making quality software. The extensibility provided by the custom metadata plugin coupled with REST services only available in Nexus Professional also lay the foundation for highly complex interactions within the enterprise. Once you start to use the work-flow and decision support features of Nexus Professional, you will start to see it as the "assembly line" - the central collaboration point for your software development efforts.

1.3.1 Nexus Professional Features

Nexus Procurement Suite

Consider the default behaviour of a proxy repository. Any developer can reference any artifact stored in a remote repository and cause Nexus to retrieve the artifact from the remote repository and serve back to a developer. Very often a company might want to control the set of artifacts which can be referenced in a proxy repository. Maybe the company has unique security requirements which require every third-party library to be subjected to a rigorous security audit before they can be used. Or, maybe another company has a legal team which needs to verify that every artifact referenced by your software adheres to an inflexible set of license guidelines. The Nexus Procurement Suite was designed to give organization this level of control over the artifacts that can be served from Nexus.

Nexus Staging Suite

When was the last time you did a software release to a production system? Did it involve a QA team that had to sign-off on a particular build? What was the process you used to re-deploy a new build if QA found a problem with the system at the last minute? Because few organizations use a mature process to manage binary software artifacts, there is little in the way of infrastructure designed to keep track of the output of a build. The Nexus Staging Suite changes this by providing work-flow support for binary software artifacts. If you need to create a release artifact and deploy it to a hosted repository, you can use the Staging Suite to post a collection of related, staged artifacts which can be tested, promoted, or discarded as a unit. Nexus keeps track of the individuals that are involved in a staged, managed release and can be used to support the decisions that go into producing quality software.

Support for OSGi Repositories

Instead of just supporting Maven repositories, Nexus Professional supports OSGi Bundle repositories and P2 repositories for those developers who are targeting OSGi or the Eclipse platform. Just like you can proxy, host, and group Maven repositories, Nexus Professional allows you to do the same with OSGi repositories.

Enterprise LDAP Support

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

Support for Atlassian Crowd

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

The User Account Plugin

When you are running a large, public instance of Nexus, it is often very useful to allow users to sign up for an account without the assistance of an administrator. Nexus Professional's User Account plugin allows for just this. With this plugin activate, a new user simply has to fill out a simple form and type in letters from a captcha. Once a user has signed up for Nexus, Nexus will then send an email with a validation link. If you are working in an environment with hundreds or thousand of users the user account plugin will allow you to support the tool without having to create logins for each individual user.

Maven Settings Management

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

Support for Artifact Bundles

When software is deployed to the Maven Central repository, it is deployed as a signed artifact bundle. Nexus Professional's Staging Suite allows you to upload artifact bundles to a staged repository.

Artifact Validation and Verification

The software artifacts you download from a remote repository are often signed with PGP signatures. Nexus Professional will make sure that these PGP signature are valid and the procurement plugin defines a few other rules that can be applied to artifacts which are downloaded from remote repositories. Nexus Professional also defines an API which allows you to create your own custom verification rules.

Custom Repository Metadata

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

1.3.2 Nexus Professional License

Nexus Professional is made available under a commercial license for businesses. Is available for free for use in qualifying Open Source projects, and is available at a discount for select Non-profits.

1.4 Choosing a Nexus Edition

If you are wondering which edition is appropriate for your organization. The following sections outline some reasons for choosing either Nexus Open Source or Nexus Professional with more information available on the [Nexus website](#).

1.4.1 Use Nexus Open Source...

...if you are new to Repository Management

If you are new to repository management, you should pick up a copy of Nexus Open Source, and experiment with Hosted and Proxy repositories. You should get a sense of how Maven Settings are configured to retrieve artifacts from a single Repository Group, and you should download a copy of the free Nexus book - Repository Management with Nexus. Once you've familiarized yourself with Nexus Open Source, you can easily upgrade to Nexus Professional by downloading and installing Nexus Professional. Nexus stores all of your repository data and configuration in a directory named `sonatype-work` which is separate from the Nexus application directory.

...if you are looking for more stability and control

If you depend directly on public repositories such as the Maven Central repository or the various repositories maintained by organizations like Codehaus or the Apache Software Foundation, you rely on these servers to be available to your developers 100% of the time. If a public repository goes down for maintenance, so does your development process. With a local proxy of Maven artifacts, you buy yourself a stable, isolated build. Even if a public repositories becomes unavailable, you will still be able to build your software against artifacts cached in your own Nexus installation.

...if you need to manage internal software distribution

If your organization needs to support collaboration between internal teams, you can use Nexus to support the distribution of internal software. With Nexus, sharing components between internal groups is as easy as adding a dependency from Maven Central. Just publish a JAR to Nexus, configure the appropriate repositories groups and inform others in our organization of the Maven coordinates. Using a repository management doesn't just make it easier to proxy external software artifacts, it makes it easier to share internal artifacts.

...if you need an intelligent local proxy

Many developers run Nexus on a local workstation as a way to gain more control over the repositories used by Nexus. This is also a great way to start evaluating Nexus. Download and install Nexus on your local workstation and point your Maven settings at <http://localhost:8081/nexus>. When you need to add a new repository, all you need to do is change the configuration of your local Nexus installation.

...if you need to integrate with an LDAP server

If you need to integrate Nexus with an an LDAP server, download Nexus Open Source. Nexus pro-

vides documented integration with popular LDAP servers such as OpenLDAP, Microsoft's Active Directory Server, and any other directory product which implements the LDAP standard.

1.4.2 Use Nexus Professional...

...if you are looking for Professional Support

When you purchase Nexus Professional, you are purchasing one year of support from the team that created the industry-standard in repository management. With Nexus Professional, you not only get a capable repository manager, you get the peace of mind that help is just a phone call away. Sonatype also offers an array of implementation and migration services for organizations looking for an extra level of assistance. Contact Sonatype Sales for more information, call +1 (888) 866-2836.

...if you need a repository manager that can support release and quality assurance decisions

Nexus Professional's Staging Suite can track the status of a software release and make sure that different decision makers are notified and supported during a software release. If you are looking for a repository manager that can automate and support software releases, download Nexus Professional and start learning about Staged repositories and Staging Rule-sets. When you start using Nexus Professional, your operations, quality assurance, and development teams can use the repository manager as a central point of collaboration.

...if you need more control over external artifacts

If you need more control over which external artifacts can be referenced and used in internal projects, you will need to use the Nexus Procurement Suite which is a part of Nexus Professional. While repositories like Maven Central are a great convenience, allowing your developers carte blanche access to any external library is often unacceptable in today's legal and regulatory environment. Nexus Professional's Procurement Suite allows you to enforce standards for external libraries. If you want to ensure that every dependency is evaluated for security or license compliance, download Nexus Professional.

...if you develop software for an Open Source project

Are you developing an open source project? If so, most open source projects qualify for a free Nexus Professional license. Open source projects can qualify for a free Professional license, or they can take advantage of free Nexus Professional hosting on <http://oss.sonatype.org>. Sonatype is very committed to supporting the development of quality open source and this is our way of giving back to the community.

...if you are developing and deploying to OSGi platforms

If you are developing OSGi components using OBR repositories, or if you are developing OSGi components using the P2 repository format, you will need to use the OSGi support available in the Nexus Professional distribution. Nexus Professional supports a wider array of repository formats than Nexus Open Source. As the industry moves toward OSGi as a standard, you should be using a product which supports these emerging standards as well as the existing repository formats used by millions of developers.

...if you need to integrate with Enterprise-level Security (LDAP and Crowd):: If you need to integrate Nexus with an Atlassian Crowd server or an enterprise LDAP deployment involving multiple servers or multiple LDAP schemas, download Nexus Professional. While Nexus Open Source provides extension points for writing custom security realms, Nexus Professional provides solid LDAP and Crowd support for the large, mission-critical LDAP deployments. If you need to support LDAP fail-over and federation, use Nexus Professional.

1.5 History of Nexus

Tamas Cservenak started working on Proximity in December 2005 as he was trying to find a way to isolate his own systems from an incredibly slow ADSL connection provided by a Hungarian ISP. Proximity started as a simple web application to proxy artifacts for a small organization with connectivity issues. Creating a local on-demand cache for Maven artifacts from the Maven Central repository gave an organization access to the artifacts on the Maven Central Repository, but it also made sure that these artifacts weren't downloaded over a very slow ADSL connection used by a number of developers.

In 2007, Sonatype asked Tamas to help create a similar product named Nexus. Nexus is currently considered the logical next step to Proximity. Nexus currently has an active development team, and portions of the indexing code from Nexus are also being used in m2eclipse.

Chapter 2

Component Lifecycle and Repository Management

2.1 Introduction

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

2.2 Component Lifecycle Management

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

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

2.2.1 Increasing Component Usage and Open Source Components

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

A large number of open source components in the form of libraries, reusable widgets or whole application, application servers and others is 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. E.g. creating a new web application framework and business workflow system just to create a website with a publishing workflow would be extremely inefficient.

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

2.2.2 Security Vulnerability and License Compliance Risks

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

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

2.2.3 Nexus and Component Lifecycle Management

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

Specific examples about using Nexus for CLM related tasks can be found in Chapter [11](#).

Nexus Professional secures your component supply chain as documented in Section [6.2.8](#), which forms an important base for your CLM efforts.

2.3 Repository Management

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

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

2.3.1 Proxying Public Repositories

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

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

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

2.3.2 Managing Releases and Snapshots

If your project is relying on a number of SNAPSHOT dependencies, Maven will need to regularly check for updated versions of these snapshots. Depending on the configuration of your remote repositories, Maven will check for SNAPSHOT updates periodically, or it might be checking for SNAPSHOT updates on every build. When Maven checks for a snapshot update it needs to interrogate the remote repository for the latest version of the SNAPSHOT dependency. Depending on your connection to the public Internet and the load on the Maven Central repository, a SNAPSHOT update can add seconds to your project's build for each SNAPSHOT dependency you rely upon.

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

2.3.3 Getting Control of Dependencies

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

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

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

2.3.4 A Nexus for Collaboration

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

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

2.4 What is a Repository?

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

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

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

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

you download and install Maven without any customization, Maven will retrieve artifacts from a Maven Central repository which serves millions of Maven users every single day. While you can configure Maven to retrieve binary software artifacts from a collection of mirrors, the best-practice is to install Nexus and use it to proxy and cache the contents of Central on your own network.

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

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

2.4.1 Release and Snapshot Repositories

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

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

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

Release

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

MD5 and SHA checksum which can be used to verify both the authenticity and integrity of the binary software artifact.

Snapshot

Snapshot artifacts are artifacts generated during the development of a software project. A Snapshot artifact has both a version number such as "1.3.0" or "1.3" and a timestamp in its name. For example, a snapshot artifact for commons-lang 1.3.0 might have the name commons-lang-1.3.0-20090314.182342-1.jar the associated POM, MD5 and SHA hashes would also have a similar name. To facilitate collaboration during the development of software components, Maven and other clients which know how to consume snapshot artifacts from a repository also know how to interrogate the metadata associated with a Snapshot artifact to retrieve the latest version of a Snapshot dependency from a repository.

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

2.4.2 Repository Coordinates

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

groupId

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

artifactId

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

version

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

qualifier would be a version like "1.2.3-BETA" where BETA signals a stage of testing meaningful to consumers of a software component.

packaging

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

2.4.3 Addressing Resources in a Repository

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

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

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

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

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

2.4.4 The Maven Central Repository

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

Artifact Metadata

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

Release Stability

Once published to the Central Repository, an artifact and the metadata describing that artifact never

change. This property of release repositories guarantees that projects which depend on releases will be repeatable and stable over time. While new software artifacts are being published every day, once an artifact is assigned a release number on the Central Repository, there is a strict policy against modifying the contents of a software artifact after a release.

Repository Mirrors

The Central Repository is a public resource, and it is currently used by the millions of developers who have adopted Maven and other build tools that understand how to interact with the Maven repository structure. There are a series of mirrors for the Central Repository which are constantly synchronized. Users are encouraged to query for project metadata and cryptographic hashes and they are encouraged to retrieve the actual software artifacts from one of Central's many mirrors. Tools like Nexus are designed to retrieve metadata from the Central Repository and artifact binaries from mirrors.

Artifact Security

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

2.5 What is a Repository Manager

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

Put simply, a repository manager provides two core features:

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

In addition to these two core features, a repository manager also allows you to manage binary software artifacts through the software development life-cycle, search and catalogue software artifacts, audit

development and release transactions, and integrate with external security systems such as LDAP. The following sections define the feature sets of Nexus Open Source and Nexus Professional.

2.5.1 Core Capabilities of a Repository Manager

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

Management of Software Artifacts

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

Management of Software Metadata

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

Proxying of External Repositories

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

Deployment to Hosted Repositories

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

Searching an Index of Artifacts

When you collect software artifacts and metadata in a repository manager, you gain the ability to create indexes and allow users and systems to search for artifacts. With the Nexus index, an IDE such as Eclipse has almost instantaneous access to the contents of all proxy repositories (including the Central repository) as well as access to your own internal and 3rd party artifacts. While the Central repository transformed the way that software is distributed, the Nexus index format brings the power of search to massive libraries of software artifacts.

Infrastructure for Artifact Management

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

2.5.2 Additional Features of a Repository Manager

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

Managing Project Dependencies

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

Managing a Software Release

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

Integration with LDAP

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

Advanced Security

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

Settings Templates

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

Support for Multiple Repository Formats

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

Archive Browsing

Nexus Professional allows users to browse the contents of archives directly in the user interface as described in Section [5.2.3](#).

2.6 Reasons to Use a Repository Manager

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

2.6.1 Speed Up Your Builds

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

2.6.2 Save Bandwidth

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

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

2.6.3 Ease the Burden on Central

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

2.6.4 Gain Predictability and Scalability

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

2.6.5 Control and Audit Dependencies and Releases

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

2.6.6 Deploy 3rd Party Artifacts

How do you deal with that one-off JAR from a vendor that is not open source, and not available on the Maven Central repository? You need to deploy these artifacts to a repository and configure your Maven instance to read from that repository. Instead of hand-crafting some POMs, download Nexus and take the two or three minutes it is going to take to get your hands on a tool that can create such a repository from 3rd-party artifacts. Nexus provides an intuitive upload form that you can use to upload any random free-floating JAR that finds its way into your project's dependencies.

2.6.7 Collaborate with Internal Repositories

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

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

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

2.6.8 Distribute with Public Repositories

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

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

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

2.7 Adopting a Repository Manager

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

2.7.1 Stage Zero: Before Using a Repository Manager

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

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

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

That is, until Repository Managers were introduced.

2.7.2 Stage One: Proxying Remote Repositories

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

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

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

2.7.3 Stage Two: Hosting a Repository Manager

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

organization.

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

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

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

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

2.7.4 Stage Three: Continuous Collaboration

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

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

2.7.5 Stage Four: Life-cycle Integration

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

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

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

Provisioning

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

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

Compliance

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

Procurement

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

Enterprise Security

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

Staging

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

Chapter 3

Installing and Running Nexus

3.1 Nexus Prerequisites

Nexus Open Source and Nexus Professional only have one prerequisite, a Java Runtime Environment (JRE) compatible with Java 6 or Java 7. Nexus is most often run with the JRE that is bundled with a Java Development Kit (JDK) installation, and it can be run with Oracle's JDK for Java 6 or Java 7. To download the latest release of the Oracle JDK, go to <http://www.oracle.com/technetwork/java/javase/downloads/index.html>, and download the latest Java 6 or Java 7 JDK.

At a minimum Java 6 Update 30 or Java 7u2 are required. IBM Java versions 6 and 7 are supported as well. In all instances we recommend the latest available version.

With the Nexus 2.5 release Java 6 support has become deprecated following the discontinued support for Java 6 itself. Java 6 is no longer supported for Nexus 2.6 and higher.

When encountering problems related to IPv6 usage with Nexus or Maven as is the default with Java 7, a known workaround is to configure

```
java.net.preferIPv4Stack=true
```

This setting is the default configuration for Nexus 2.5+.

**Warning**

While known to work for most use-cases OpenJDK is not officially supported.

3.2 Downloading Nexus

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

3.2.1 Downloading Nexus Open Source

To download Nexus Open Source go to <http://www.sonatype.org/nexus/go> and download the latest Nexus Open Source distribution by clicking on the appropriate button for a ZIP or a Gzip TAR archive (TGZ) shown in Figure 3.1. Your download will be file named nexus-2.2-01-bundle.zip or nexus-2.2-01-bundle.tar.gz

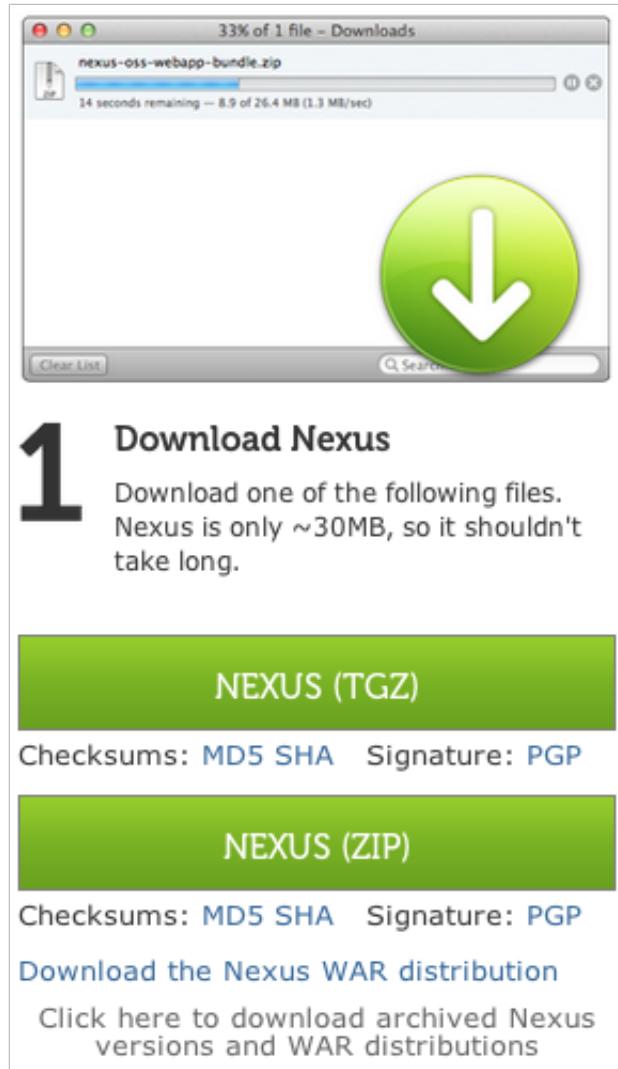


Figure 3.1: Downloading Nexus Open Source

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

The screenshot shows a two-step process for downloading Nexus. Step 1, 'Select a Version', lists various versions from 2.0.3 to 2.3.1 with their release dates. Step 2, 'Download a Distribution', provides download links for each version in tar.gz and zip formats, along with checksums.

Version	Release Date
Nexus 2.3.1	2013-01-04
Nexus 2.3	2013-01-04
Nexus 2.2	2012-10-25
Nexus 2.1.2	2012-07-20
Nexus 2.1.1.1	2012-08-14
Nexus 2.1.1	2012-08-01
Nexus 2.1	2012-07-30
Nexus 2.0.6	2012-06-25
Nexus 2.0.5	2012-06-04
Nexus 2.0.4-1	2012-04-27
Nexus 2.0.4	2012-04-25
Nexus 2.0.3	2012-03-29

Download Nexus 2.3.1

NEXUS (TGZ)

Checksums: [MD5](#) [SHA](#) [Signature](#)

NEXUS (ZIP)

Checksums: [MD5](#) [SHA](#) [Signature](#)

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

Nexus Open Source can also be deployed as a web application in a servlet container like Jetty or Tomcat or an application server like Glassfish or JBoss. Instructions for installing Nexus as a WAR are found in Section 3.9.

3.2.2 Downloading Nexus Professional

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

3.3 Installing Nexus

The following instructions are for installing Nexus Open Source or Nexus Professional as a stand-alone server. Nexus comes bundled with a Jetty instance which listens to all configured IP addresses on a host (0.0.0.0) and runs on port 8081 by default. If you would like to run Nexus as a web application in an existing application server or servlet container, please refer to the instructions in Section [3.9](#).

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

```
$ unzip nexus-2.2-01-bundle.zip
```

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

```
$ tar xvzf nexus-2.2-01-bundle.tar.gz
```

For Nexus professional the equivalent commands would be

```
$ unzip nexus-professional-2.2.1-bundle.zip  
$ tar xvzf nexus-professional-2.2.1-bundle.tar.gz
```

Note

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

Note

If you are installing Nexus on a server, you might want to use a directory other than your home directory. On a Unix machine, this book assumes that Nexus is installed in /usr/local/nexus-2.2.1 with a symbolic link /usr/local/nexus to the nexus directory. Using a generic symbolic link nexus to a specific version is a common practice which makes it easier to upgrade when a newer version of Nexus is made available.

```
$ sudo cp nexus-2.2-01-bundle.tar.gz /usr/local
```

```
$ cd /usr/local  
$ sudo tar xvzf nexus-2.2-01-bundle.tar.gz  
$ ln -s nexus-2.2-01 nexus
```

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

Note

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

The Nexus installation directory `nexus-2.2-01` or `nexus-professional-2.2.1` has a sibling directory named `sonatype-work`. This directory contains all of the repository and configuration data for Nexus and is stored outside of the Nexus installation directory to make it easier to upgrade to a newer version of Nexus.

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

3.4 Upgrading Nexus

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

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

If you have defined a symbolic link for the version of Nexus to use, stop the server and change that to point at the new Nexus application directory. When you start the new instance of Nexus it will read the existing repository configuration from the `sonatype-work` directory. Depending on the version you upgrade from and to, some maintenance tasks like rebuilding the internal indices can be necessary. Please refer to the

[upgrade notes](#) of the new release for more information on this. In addition a review of the [release notes](#) can be very useful to get a better understanding of potential, additional steps required.

If you are using any additional plugins supplied by Sonatype, the new version of Nexus you downloaded will contain a newer version of the plugin. Be sure to copy the new version from the optional-plugins folder to the plugin-repository folder as documented in Section [20.1](#) and restart Nexus.

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

Note

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

3.5 Running Nexus

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

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

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

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

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

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

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

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

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

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

The screenshot shows a web form titled 'Sonatype Nexus Professional'. At the top right is a link 'About, Help & Support'. The main title 'Trial Activation' is followed by the instruction 'Just complete this quick form to activate your trial. We'll email your license key and you'll be up and running in no time.' Below this is a section titled 'About Your Trial' with the sub-instruction 'Once you activate your trial, it will be active for 30 days. If you want to extend your trial, you need to re-activate your trial, just email us at support@nexuspro.com'. The form itself has fields for 'First name', 'Last name', 'Email', 'Organization', 'Country', 'Region', and 'Comments'. A 'Submit Activation Request' button is at the bottom.

Figure 3.3: Nexus Trial Activation Form

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



Figure 3.4: Nexus License Activation

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



Figure 3.5: Nexus Professional Evaluation Welcome Screen

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

Tip

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

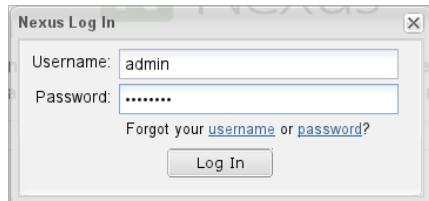


Figure 3.6: Nexus Login Dialog (default login/password is admin/admin123)

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

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

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

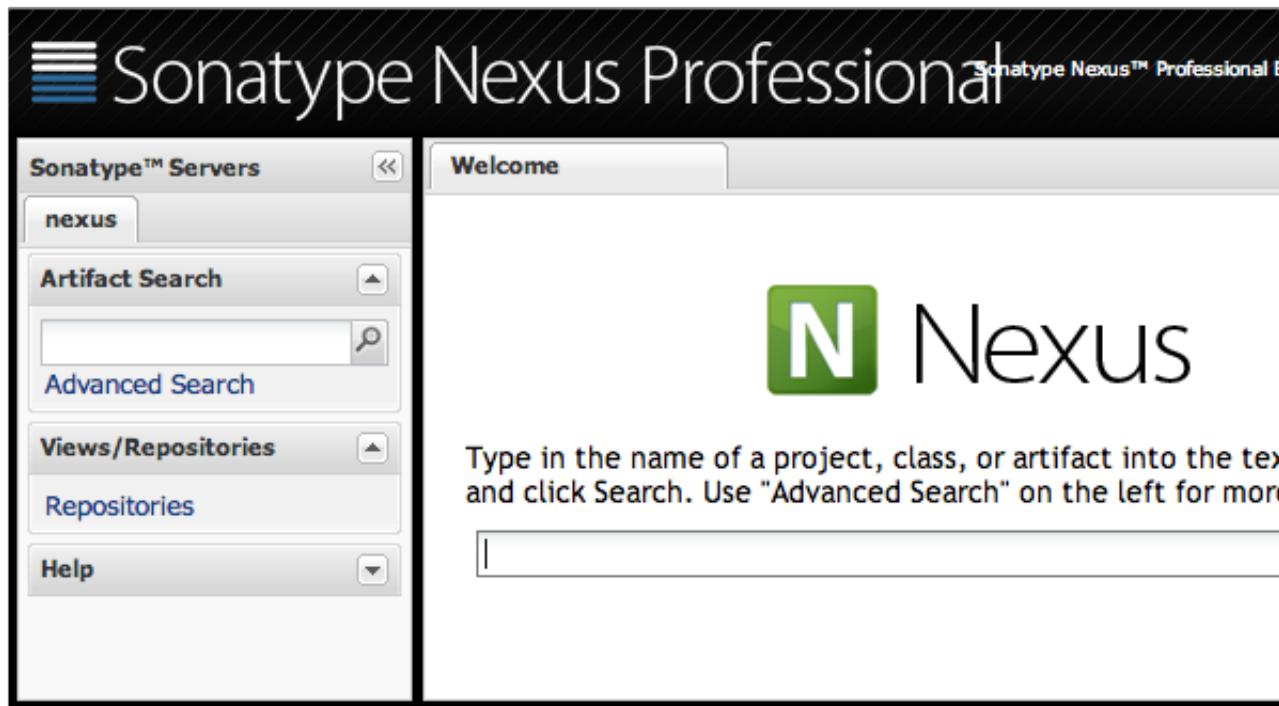


Figure 3.7: Nexus Application Window

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

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

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

```
solaris-sparc-32  
solaris-sparc-64  
solaris-x86-32  
windows-x86-32  
windows-x86-64
```

Tip

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

3.6 Post-Install Checklist

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

3.6.1 Step 1: Change the Administrative Password and Email Address

The administrative password defaults to `admin123`. The first thing you should do to your new Nexus installation is change this password. To change the administrative password login as "admin" with the password "`admin123`", and click on Change Password under the Security menu in the left-hand side of the browser window. For more detailed instructions, see Section [5.8](#).

3.6.2 Step 2: Configure the SMTP Settings

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

3.6.3 Step 3: Configure Default HTTP Proxy Setting

In many deployments the internet, and therefore any remote repositories that Nexus needs to proxy, can only be reached via a HTTP proxy server internal to the deployment company. In these cases the connection details to that proxy server need to be configured in Nexus, as documented in Section [6.1.6](#) in order for Nexus to be able to proxy remote repositories at all.

3.6.4 Step 4: Enable Remote Index Downloads

Nexus ships with three important proxy repositories for the Maven Central repository, Apache Snapshot repository, and the Codehaus Snapshot repository. Each of these repositories contains thousands (or tens of thousands) of artifacts and it would be impractical to download the entire contents of each. To that end, most repositories maintain an index which catalogues the entire contents and provides for fast and efficient searching. Nexus uses these remote indexes to search for artifacts, but we've disabled the index download as a default setting. To download remote indexes:

1. Click on Repositories under the Views/Repositories menu in the left-hand side of the browser window.
2. Select each of the three proxy repositories and change Download Remote Indexes to true in the Configuration tab. You'll need to load the dialog shown in Figure [6.13](#) for each of the three repositories.

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

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

3.6.5 Step 5: Change the Deployment Password

The deployment user's password defaults to deployment123. Change this password to make sure that only authorized developers can deploy artifacts to your Nexus installation. To change the deployment password: log in as an administrator, click on Security to expand the Security menu, then click on Users. You should then see a list of users. Right-click on the deployment user and select "Set Password".

3.6.6 Step 6: If necessary, set the LANG Environment Variable

If your Nexus instance needs to store configuration and data using an international character set, you should set the LANG environment variable. The Java Runtime will adapt to the value of the LANG environment variable and ensure that configuration data is saved using the appropriate character type. If you are starting Nexus as a service, place this environment variable in the start-up script found in /etc/init.d/nexus. For more information about locale settings in Ubuntu read <https://help.ubuntu.com/community/Locale>

3.6.7 Step 7: Configure Routes

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

3.7 Configuring Nexus as a Service

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

3.7.1 Running as a Service on Linux

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

1. Copy either \$NEXUS_HOME/bin/nexus to /etc/init.d/nexus
2. Make the /etc/init.d/nexus script executable - chmod 755 /etc/init.d/nexus
3. Edit this script changing the following variables:
 - a. Change NEXUS_HOME to the absolute folder location e.g. NEXUS_HOME="/usr/local/nexus"
 - b. Set the RUN_AS_USER to "nexus" or any other user with restricted rights that you want to use to run the service. You should not be running Nexus as root.
 - c. Change PIDDIR to a directory where this user has read/write permissions. In most Linux distributions, /var/run is only writable by root. The properties you need to add to customize the PID file location is "wrapper.pid". For more information about this property and how it would be configured in wrapper.conf, see: <http://wrapper.tanukisoftware.com/doc/english-properties.html>
4. Create a "nexus" user with sufficient access rights to run the service
5. Change the Owner and Group of your nexus directories to "nexus"
6. If Java is not on the default path for the user running Nexus, add a JAVA_HOME variable which points to your local Java installation and add a \$JAVA_HOME/bin to the PATH



Warning

While not recommended, it is possible to run Nexus as root user by setting RUN_AS_USER=root.

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

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

```
$ cd /etc/init.d  
$ chkconfig --add nexus  
$ chkconfig --levels 345 nexus on
```

```
$ service nexus start
Starting Sonatype Nexus...
$ tail -f /usr/local/nexus/logs/wrapper.log
```

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

3.7.1.2 Add Nexus as a Service on Ubuntu and Debian

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

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

3.7.2 Running as a Service on Mac OS X

The standard way to run a service on Mac OS X is by using launchd, which uses plist files for configuration. An example plist file for Nexus 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>/usr/local/nexus/bin/nexus</string>
<string>console</string>
</array>
<key>RunAtLoad</key>
<true/>
</dict>
</plist>
```

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

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

Tip

Consider setting up a different user to run Nexus and adapt permissions and the `RUN_AS_USER` setting in the `nexus` startup script.

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

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

3.7.3 Running as a Service on Windows

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

3.8 Running Nexus Behind a Proxy

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

separate proxy for performance improvements unnecessary.

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

Some brief instructions for establishing such a setup with Apache httpd follow as an example. We assume that you've already installed Apache 2, and that you are using a Virtual Host for www.somecompany.com.

Let's assume that you wanted to host Nexus behind Apache HTTPD at the URL <http://www.somecompany.com>. To do this, you'll need to change the context path that Nexus is served from.

1. Edit nexus.properties in `$NEXUS_HOME/conf`. You'll see an element named `nexus-webapp-context-path`. Change this value from "/nexus" to "/"
2. Restart Nexus and Verify that it is available on <http://localhost:8081/>
3. Clear the Base URL in Nexus as shown in Figure 6.8 under Application Server Settings.

At this point, edit the HTTPD configuration file for the www.somecompany.com virtual host. Include the following to expose Nexus via mod_proxy at <http://www.somecompany.com/>.

```
ProxyRequests Off
ProxyPreserveHost On

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

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

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

Apache configuration is going to vary based on your own application's requirements and the way you intend to expose Nexus to the outside world. If you need more details about Apache HTTPD and mod_proxy, please see <http://httpd.apache.org>

3.9 Installing the Nexus WAR

The Nexus Open Source WAR can run on most Java application servers. To download the Nexus Open Source WAR, go to <http://www.sonatype.org/nexus/go>. Click on the Download Site link and then download the Nexus WAR. Once you have downloaded the Nexus Open Source WAR, you can install it in a servlet container or application server.

Warning

 Testing of the WAR file install is currently only done on Tomcat and Jetty. The complexity of the task to get Nexus to run on an application server may vary depending on the server and the server version. It is strongly recommended to use the bundle install with the included Jetty application server instead of the WAR file. Support for Nexus Professional is only provided for the bundle install.

The process for installing a WAR in a servlet container or application server is going to vary for each specific application. Often, this installation process is as simple as dropping a WAR file in a special directory and restarting the container. In many cases it will be required to expand the war into a folder rather than deploying the unextracted WAR file for the plugin manager to work with all installed plugins and allow installation of additional plugins.

For example, to install the Nexus WAR in Tomcat, drop the `nexus-2.2-01.war` file in `$TOMCAT_HOME/webapps` and restart your Tomcat instance. Assuming that Tomcat is configured on port 8080 once Tomcat is started, Nexus will be available on <http://localhost:8080/nexus-2.2-01>

If you would like a less verbose URL, copy `nexus-2.2-01.war` to a file named `nexus.war` before copying the distribution to `$TOMCAT_HOME/webapps`

Note

When installing Nexus as a WAR in an application server or servlet container, it automatically creates a `sonatype-work` directory in the home directory of the user running the application server. This directory contains all of the necessary configuration and repository storage for Nexus.

3.10 Installing a Nexus Professional License

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

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

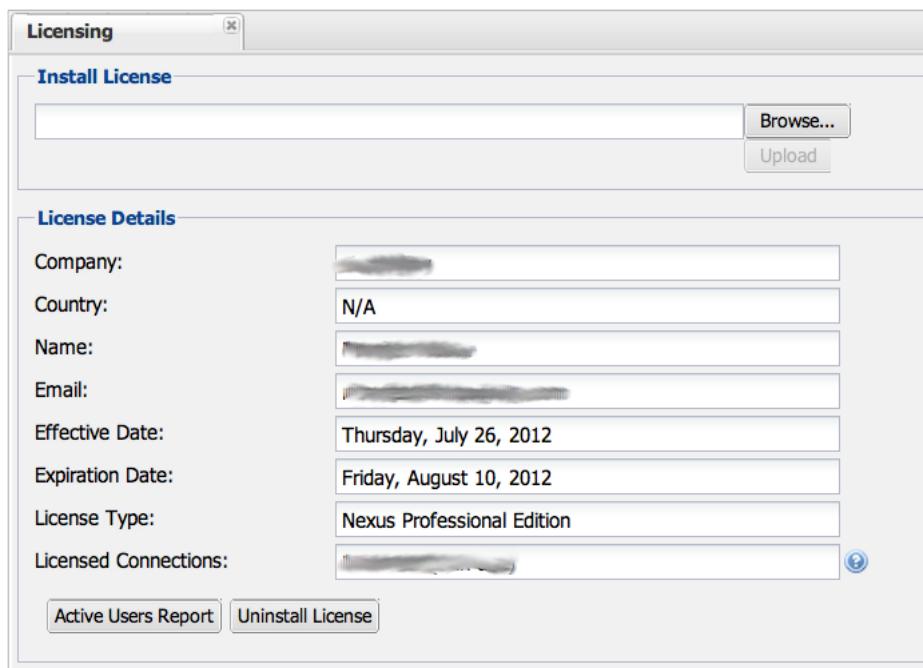


Figure 3.8: Nexus Professional Licensing Panel

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

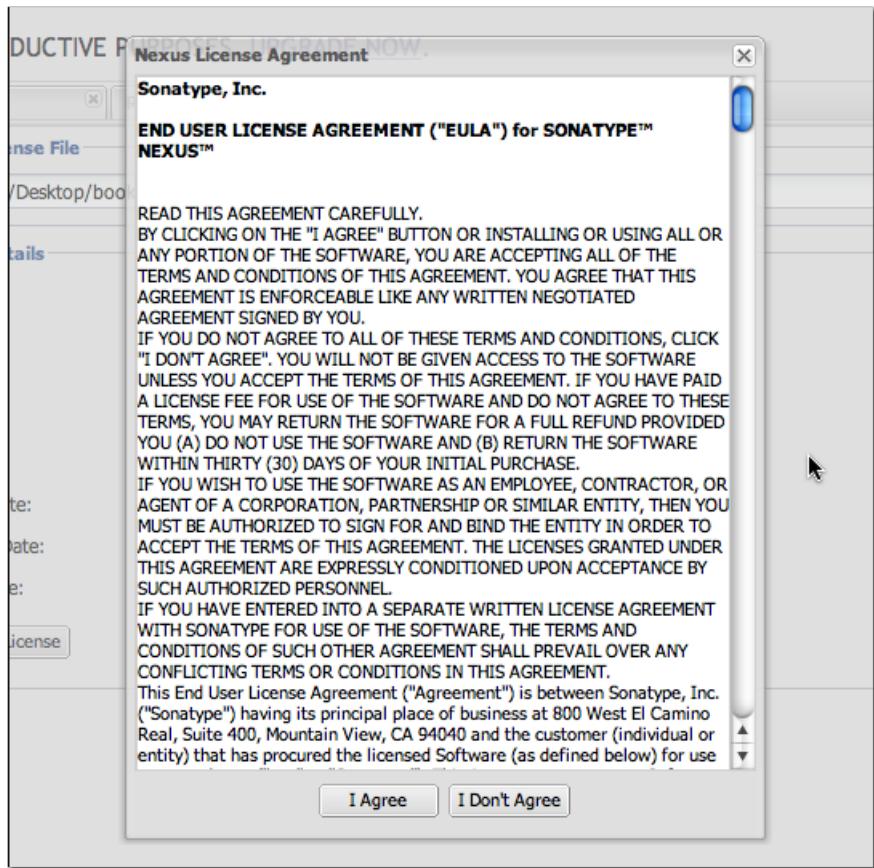


Figure 3.9: Nexus Professional End-user License Agreement

Once you have agreed to the terms and conditions contained in the End User License Agreement, Nexus Professional will then display a dialog box confirming the installation of a Nexus Professional license as shown in Figure 3.10.

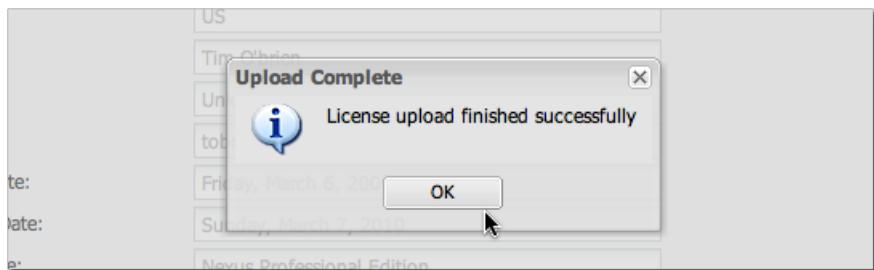


Figure 3.10: License Upload Finished Dialog

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

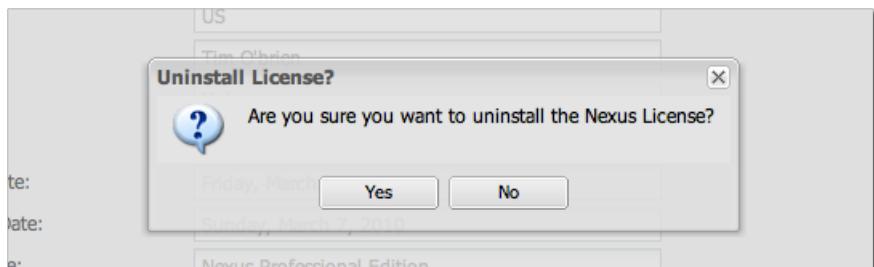


Figure 3.11: Uninstall License Confirmation Dialog

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

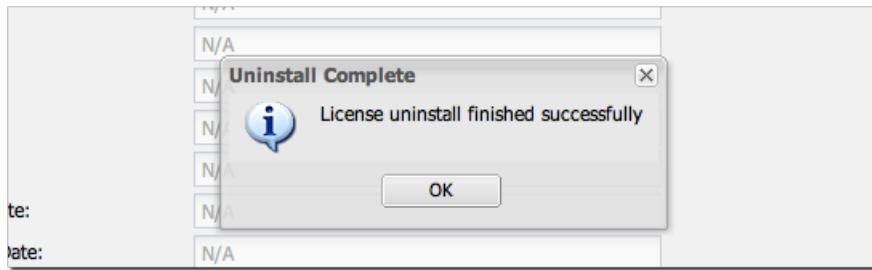


Figure 3.12: License Uninstall Completed Dialog

3.10.1 License Expiration

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

3.11 Nexus Directories

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

3.11.1 Sonatype Work Directory

The Sonatype Work directory `sonatype-work` is installed as a sibling to the `nexus` application directory, and the location of this directory can be configured via the `nexus.properties` file which is described in Section 3.11.2. Figure 3.13 shows the Sonatype Nexus work directory with some of its sub-directories.

```
$ tree sonatype-work/ -L 2
sonatype-work/
└── nexus
    ├── access
    ├── aether-local-repository
    ├── broker
    ├── conf
    ├── error-report-bundles
    ├── health-check
    ├── indexer
    ├── logs
    ├── nuget
    ├── plugin-repository
    ├── proxy
    ├── storage
    ├── template-store
    ├── timeline
    └── trash
    README.txt
```

Figure 3.13: The Sonatype Work Directory

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

access/

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

aether-local-repository/

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

backup/

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

broker/

The broker directory and its sub-directories contains the storage backend for the Smart Proxy messaging component.

conf/

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

conf/keystore/

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

db/

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

error-report-bundles/

Contains the bundled archives of data assembled for problem reporting.

health-check/

Holds cached reports from the Repository Health Check plugin.

indexer/

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

logs/

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

nuget/

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

p2/

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

plugin-repository/

This directory contains any additionally installed plugins from third parties as documented in Section [20.1](#).

proxy/

Stores data about the files contained in a remote repository. Each proxy repository has a sub-directory in the proxy/attributes/ directory and every file that Nexus has interacted with in the remote repository has an XML file which captures such data as the: last requested timestamp, the remote URL for a particular file, the length of the file, and the digests for a particular file among other things. If you need to backup the local cached contents of a proxy repository, you should also back up the contents of the proxy repository's directory under proxy/attributes/

storage/

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

template-store/

Contains templates for default repositories. If you examine the XML files in this directory, you will see that they contain default templates for each different type of repository. For example, the repository-default_proxy_release.xml file contains defaults for a Proxy repository with a release policy.

timeline/

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

tmp/

Folder used for temporary storage.

trash/

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

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

broker.groovy

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

capabilities.xml

Further Smart Proxy backend configuration.

healthcheck.properties

Configuration for the Repository Health Check.

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

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

lvo-plugin.xml

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

nexus.xml

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

pgp.xml

Contains PGP key server configuration.

nexus-obr-plugin.properties

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

procurement.xml

Contains configuration for the Nexus Procurement plugin in Nexus Professional.

security-configuration.xml

Contains global security configuration.

security.xml

Contains security configuration about users and roles.

staging.xml

Contains configuration for the Nexus Staging Plugin in Nexus Professional.

3.11.2 Nexus Configuration Directory

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

The files and folders contained in this directory are:

classworlds.conf

Defines the order in which resources and classes are loaded from the classpath. It is unlikely that even the most advanced Nexus users will ever need to customize the contents of this file.

nexus.properties

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

jetty.xml

If this file is present in the conf/ directory, it will be used to configure Jetty.

The conf/examples/ directory contains sample Jetty configuration files which can be used to customize the behaviour of the Jetty servlet container:

jetty.xml

contains a jetty.xml sample with no customizations. This sample file listens on the "application-port" defined in nexus.properties

jetty-ajp.xml

Contains a jetty.xml sample which will configure Nexus to listen on an AJP port 8009. This configuration can be used if you are proxying your Nexus server with web server which understands the AJP protocol such as Apache httpd with the mod_proxy_ajp module.

jetty-dual-ports-with-ssl.xml

Contains a jetty.xml sample which configures Nexus to listen on both the "application-port" and "application-port-ssl" (as defined in nexus.properties). This sample configuration also contains the SSL redirect rule.

jetty-faster-windows.xml

Contains a jetty.xml sample which configures a response buffer size that will address performance issues on Windows 2003 Server, for more information about this fix see [the Jetty Wiki](#)

jetty-header-buffer.xml

Contains a jetty.xml sample which increases the headerBufferSize to 8k from the default of 4k. Documentation about the header buffer size can be found on [the Jetty Wiki](#)

jetty-simple-https-proxy.xml

Contains a jetty.xml sample which should be used if you are proxying a Nexus instance with a web server that is handling SSL. For example, if you were proxying Nexus with Apache httpd server using mod_ssl you would use this configuration to configure the Jetty RewriteHandler

jetty-ssl.xml

Contains a jetty.xml sample which will only serve SSL encrypted content from "application-port" (as defined in nexus.properties)

The conf/examples/proxy-https/ directory contains two files: apache2.conf and jetty.xml contains sample mod_proxy directives to configure Apache httpd to handle SSL.

Chapter 4

Configuring Maven to Use Nexus

4.1 Introduction

To use Nexus, you will configure Maven to check Nexus instead of the public repositories. To do this, you'll need to edit your mirror settings in your `~/.m2/settings.xml` file. First, we're going to demonstrate how to configure Maven to consult your Nexus installation instead of retrieving artifacts directly from the Maven Central repository. After we override the central repository and demonstrate that Nexus is working, we'll circle back to provide a more sensible set of settings that will cover both releases and snapshots.

4.2 Configuring Maven to Use a Single Nexus Group

If you are adopting Nexus for internal development you should configure a single Nexus group which contains both releases and snapshots. To do this, add snapshot repositories to your public group, and add the following mirror configuration to your Maven settings in `~/.m2/settings.xml`

Configuring Maven to Use a Single Nexus Group

```
<settings>
  <mirrors>
    <mirror>
```

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

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

The `mirrorOf` pattern of `*` causes any repository request to be redirected to this mirror and therefore 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 nexus group is listed as an active profile in the activeProfiles element.

4.3 Adding Custom Repositories for Missing Dependencies

If you've configured your Maven settings.xml to list the Nexus public group as a mirror for all repositories, you might encounter projects which are unable to retrieve artifacts from your local Nexus installation. This usually happens because you are trying to build a project which has defined a custom set of repositories and snapshotRepositories in a pom.xml. When you encounter a project which contains a custom repository element in a pom.xml add this repository to Nexus as a new proxy repository and then add the new proxy repository to the public group.

4.4 Adding a New Repository

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

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

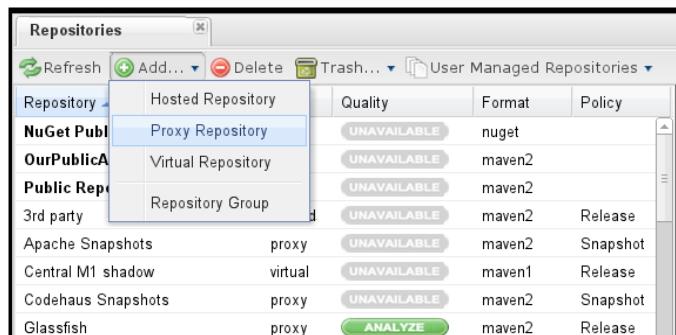


Figure 4.1: Creating a New Proxy Repository

Once you do this, you will see a screen resembling Figure 4.2. Populate the required fields Repository ID and the Repository Name. The Repository ID will be part of the URL used to access the repository, so it is recommended to avoid characters that could cause problems. Set the Repository Policy to "Release", and the Remote Storage Location to the public URL of the repository you want to proxy.

Repository	Type	Quality	Format	Policy	Repository Status
New Proxy Repository	proxy	UNAVAILABLE			
NuGet Public Group	group	UNAVAILABLE			
OurPublicAPIs	group	UNAVAILABLE			

New Proxy Repository

Repository ID: **jboss-releases**

Repository Name: **JBoss Releases**

Repository Type: **proxy**

Provider: **Maven2**

Format: **maven2**

Repository Policy: **Release**

Default Local Storage Location:

Override Local Storage Location:

Remote Repository Access

Remote Storage Location: **https://repository.jboss.org/nexus/content/repositories/releases/**

Download Remote Indexes: **True**

Auto Blocking Enabled:

Save **Cancel**

Figure 4.2: Configuring a Proxy Repository

Once you've filled out this screen, click on the Save button. Nexus will then be configured to proxy the repository.

4.5 Adding a Repository to a Group

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

Note

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

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

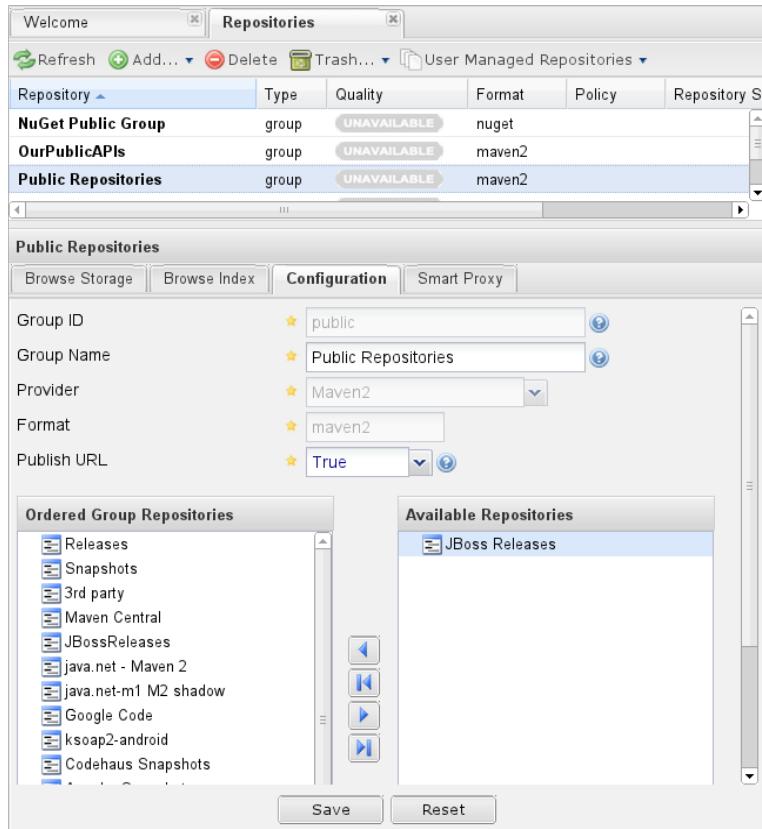


Figure 4.3: Adding New Repositories to a Nexus Group

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

Note

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

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

If you were not using a repository manager, you would have added these repositories to the repository element of your project's POM, or you would have asked all of your developers to modify `~/.m2/settings.xml` to reference two new repositories. Instead, you used the Nexus repository manager to add the two repositories to the public group. If all of the developers are configured to point to the public group in Nexus, you can freely swap in new repositories without asking your developers to change local configuration, and you've gained a certain amount of control over which repositories are made available to your development team.

Chapter 5

Using the Nexus User Interface

5.1 Introduction

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

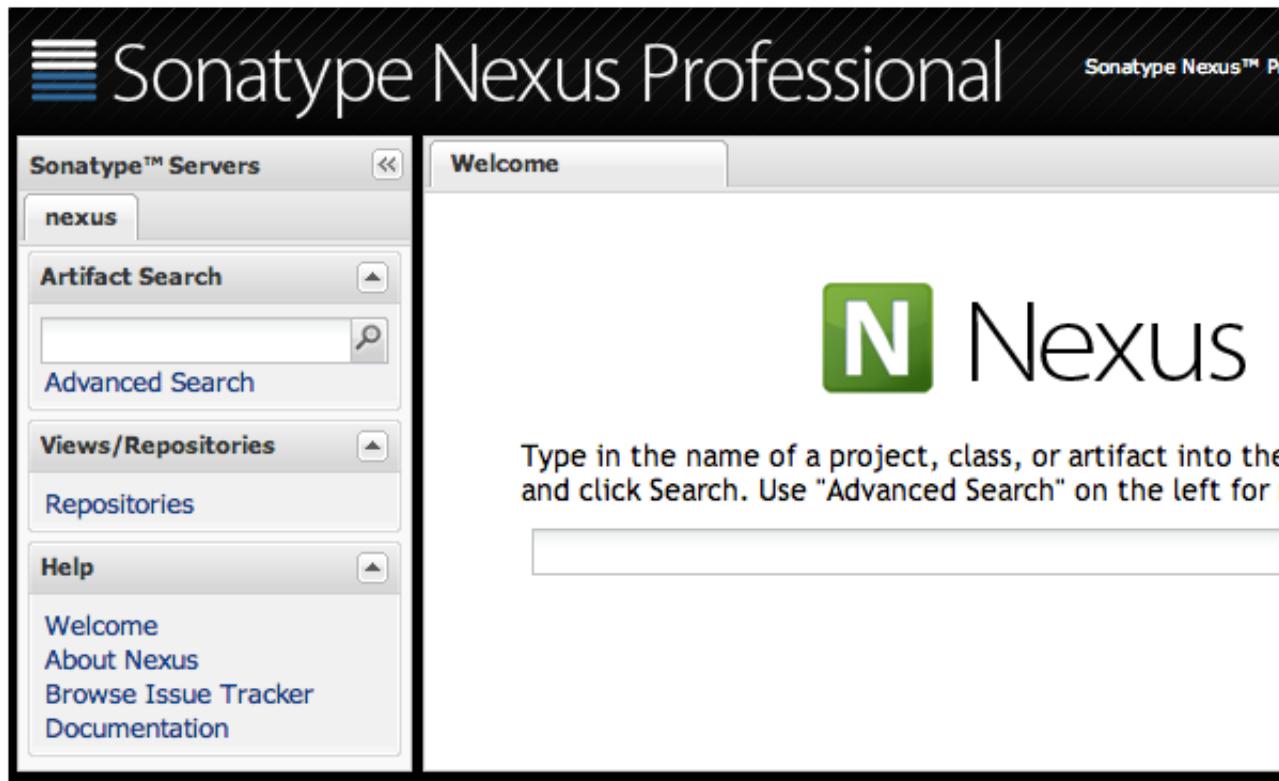


Figure 5.1: Nexus Interface for Anonymous Users

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

5.2 Browsing Repositories

One of the most straightforward uses of the Nexus is to browse the structure of a repository. If you click on the Repositories menu item in the Views.Repositories menu, you should see the following display. The top-half of Figure 5.2 shows you a list of groups and repositories along with the type of the repository and the repository status. To browse the artifacts that are stored in a local Nexus instance, click on the Browse Storage tab for a repository as shown in Figure 5.2.

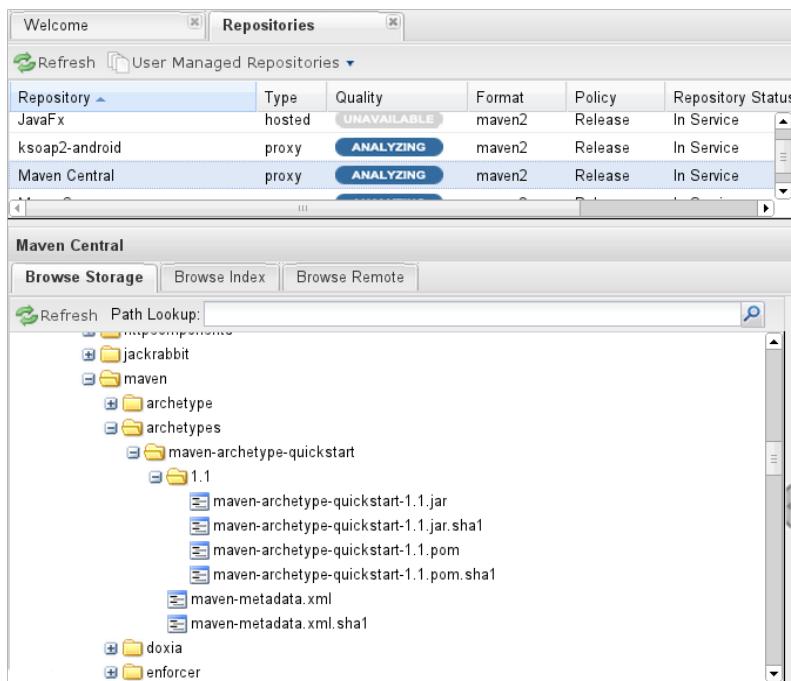


Figure 5.2: Browsing a Repository Storage

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

Note

When browsing a remote repository you might notice that the tree doesn't contain all of the artifacts in a repository. When you browse a proxy repository, Nexus is displaying the artifacts which have been cached locally from the remote repository. If you don't see an artifact you expected to see through Nexus, it only means that Nexus has yet to cache the artifact locally. If you have enabled remote repository index downloads, Nexus will return search results that may include artifacts not yet downloaded from the remote repository. Figure 5.2, is just an example, and you may or may not have the example artifact available in your installation of Nexus.

A Nexus proxy repository acts as a local cache for a remote repository, in addition to downloading and caching artifacts locally, Nexus will also download an index of all the artifacts stored in a particular

repository. When searching or browsing for artifacts, it is often more useful to search and browse the repository index. To view the repository index, click on the Browse Index tab for a particular repository to load the interface shown in Figure 5.3.

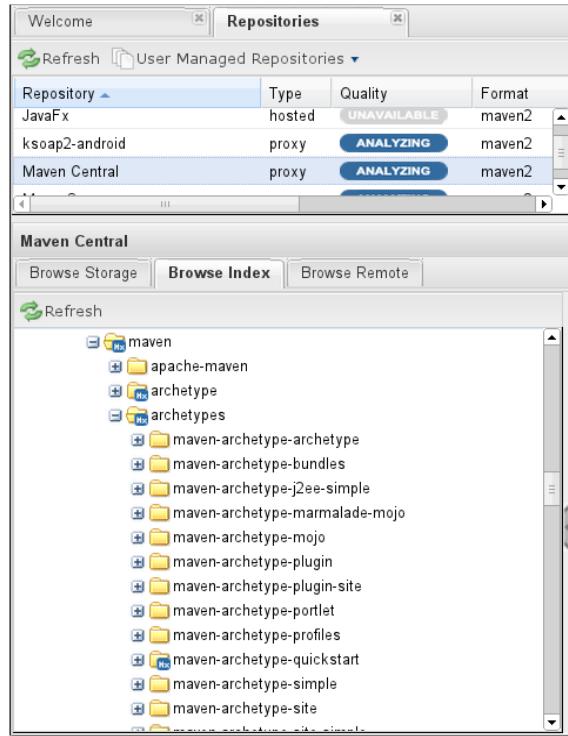


Figure 5.3: Browsing a Repository Index

As shown in Figure 5.3, if an artifact has been downloaded from a remote repository and cached in Nexus, the artifact or folder will display a small Nexus logo.

5.2.1 Viewing the Artifact Information

Once you located an archive in the repository index or storage the right hand panel will at minimum show the Artifact Information tab as visible in Figure 5.4. Besides showing details like the Repository Path, Size, Checksums, location of the artifact and other details you are able to download and delete the artifact with the respective buttons.

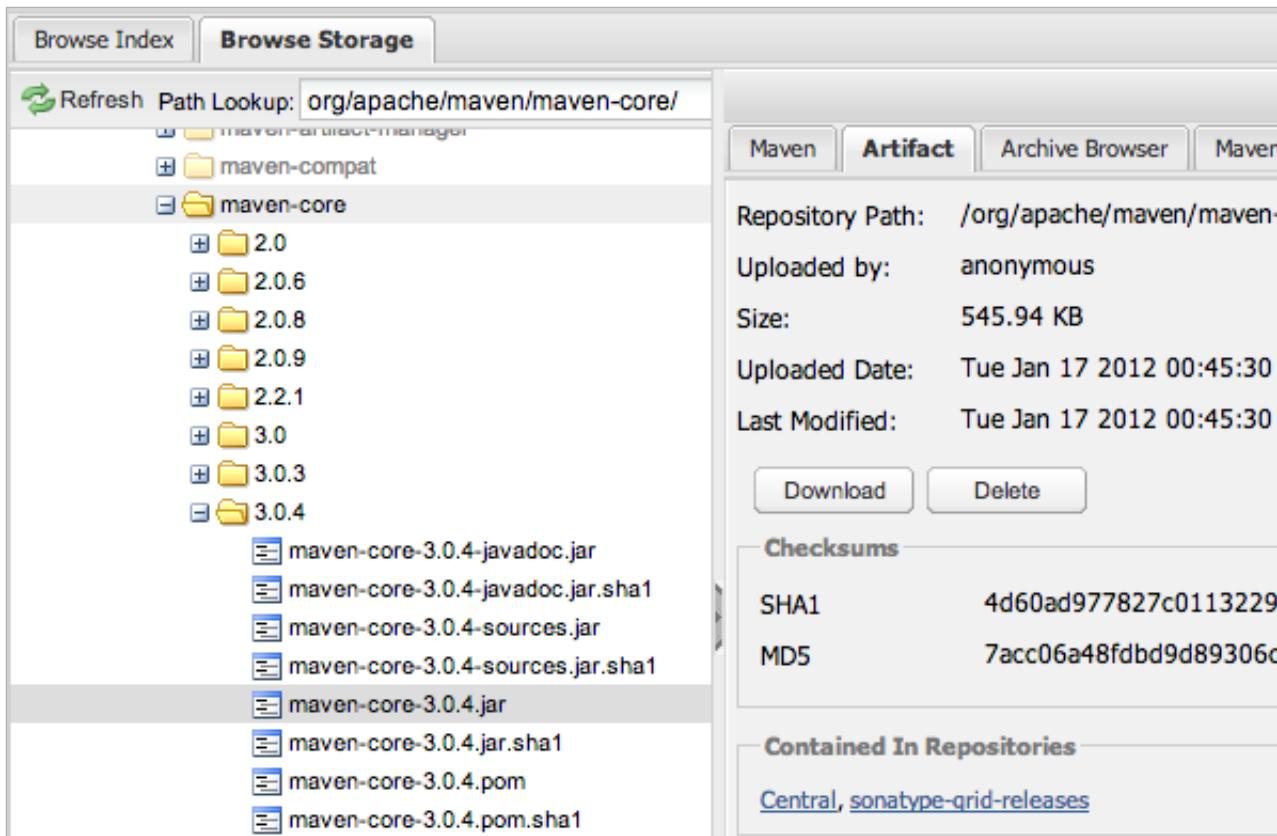


Figure 5.4: Viewing the Artifact Information

5.2.2 Viewing the Maven Information

If the artifact you are looking at in the browser is a Maven related artifact like a pom file or a jar you will see the Maven Information tab in the right hand panels. As visible in Figure 5.5 the GAV parameters are displayed above an XML snippet identifying the artifact that you can just cut and paste into a Maven pom.xml file.

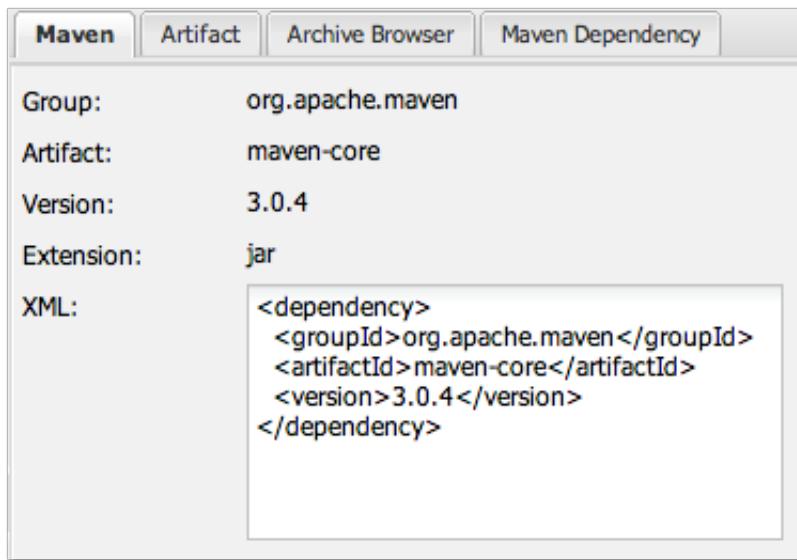


Figure 5.5: Viewing the Maven Information

5.2.3 Using the Artifact Archive Browser

For binary artifacts like jar files Nexus displays an Archive Browser panel as visible in Figure 5.6 that allows you to view the contents of the archive. Clicking on individual files in the browser will download them and potentially display them in your browser. This can be useful for quickly checking out the contents of an archive without manually downloading and extracting it.

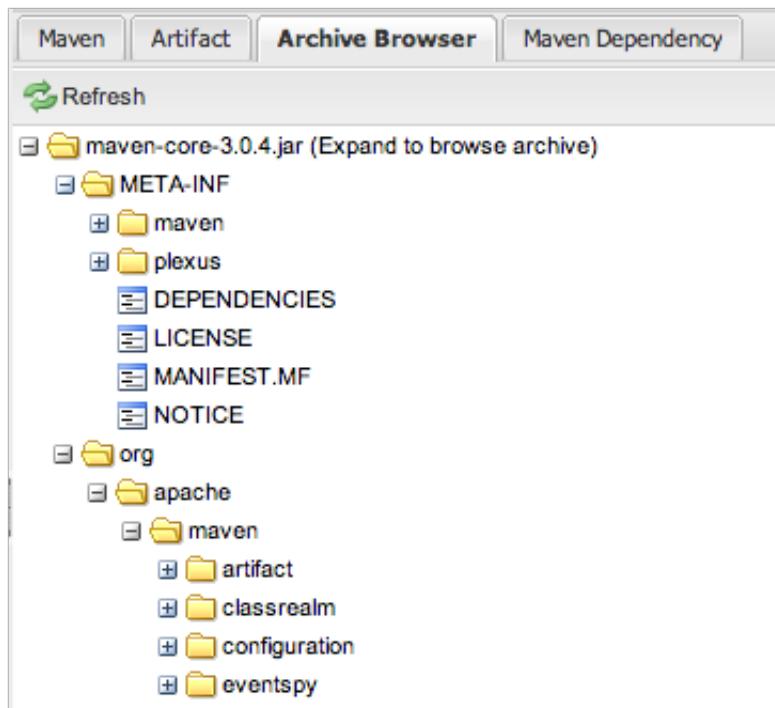


Figure 5.6: Using the Archive Browser

5.2.4 Viewing the Artifact Dependencies

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

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

Repository

When resolving an artifact's dependencies, Nexus will query an existing repository or repository group. In many cases it will make sense to select the same repository group you are referencing in your Maven Settings. If you encounter any problems during the dependency resolution, you need

to make sure that you are referencing a repository or a group which contains these dependencies.

Mode

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

Once you have selected a repository to resolve against and a mode to display an artifact's dependencies, click on the Resolve button as shown in Figure 5.7. Clicking on this button will start the process of resolving dependencies, depending on the number of artifacts already cached by Nexus, this process can take anywhere from a few seconds to a minute. Once the resolution process is finished, you should see the artifact's dependencies as shown in Figure 5.7.

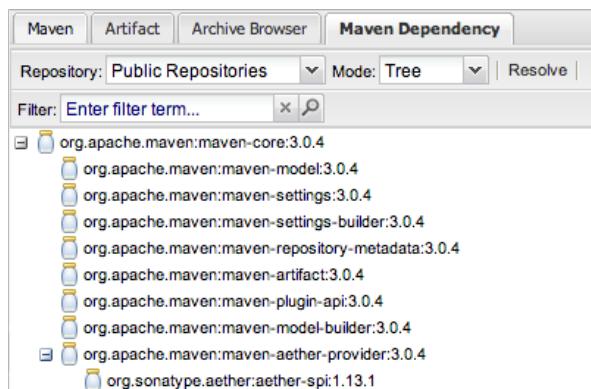


Figure 5.7: View an Artifact's Dependencies

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

5.2.5 Viewing the Artifact Insight Data

One of the added features of Nexus Professional is the usage of data from Sonatype Insight. This data contains security and license information about artifacts and is accessible for a whole repository in the

Repository Health Check feature described in Chapter 11. Details about the vulnerability and security issue ratings and others can be found there as well.

The Insight tab displays the security and licence information available for a specific artifact. It is available in browsing or search results, once you have selected an artifact in the search results list or repository tree view. An example search for Jetty, with the Insight tab visible, is displayed in Figure 5.8. It displays the results from the License Analysis and any found Security Issues.

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

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

Threat Level	Problem Code	Summary
5	CVE-2007-5615	CRLP injection vulnerability in Mortay Jetty before 6.1.6rc0 allows remote attackers to inject arbitrary HTTP headers and conduct HTTP response splitting attacks via unspecified vectors.

Figure 5.8: Insight Data Displaying Security Vulnerabilities for an Old Version of Jetty

The Security Issues section displays two issues of Threat Level 5. The Summary column contains a small summary description of the security issue. The Problem Code column contains the codes, which link to the respective entries in the Common Vulnerabilities and Exposures CVE list as well as the Open Source Vulnerability DataBase OSVDB displayed in Figure 5.9 and Figure 5.10.

CVE-2011-4461 Learn more at National Vulnerability Database (NVD)
(under review)

References

Related Items

ITEMS OF INTEREST

Figure 5.9: Common Vulnerabilities and Exposures CVE Entry for a Jetty Security Issue

Description

Classification

Solution

Products

Figure 5.10: Open Source Vulnerability DataBase OSVDB Entry for a Jetty Security Issue

5.3 Browsing Groups

Nexus contains ordered groups of repositories which allow you to expose a series of repositories through a single URL. More often than not, an organization is going to point Maven at the two default Nexus groups: Public Repositories and Public Snapshot Repositories. Most end-users of Nexus are not going to know what artifacts are being served from what specific repository, and they are going to want to be able to browse the Public Repository. To support this use case, Maven allows you to browse the contents of a Nexus Group as if it were a single merged repository with a tree structure. Figure 5.11, shows the browsing storage interface for a Nexus Group. There is no difference to the user experience of browsing a Nexus Group vs. browsing a Nexus Repository.

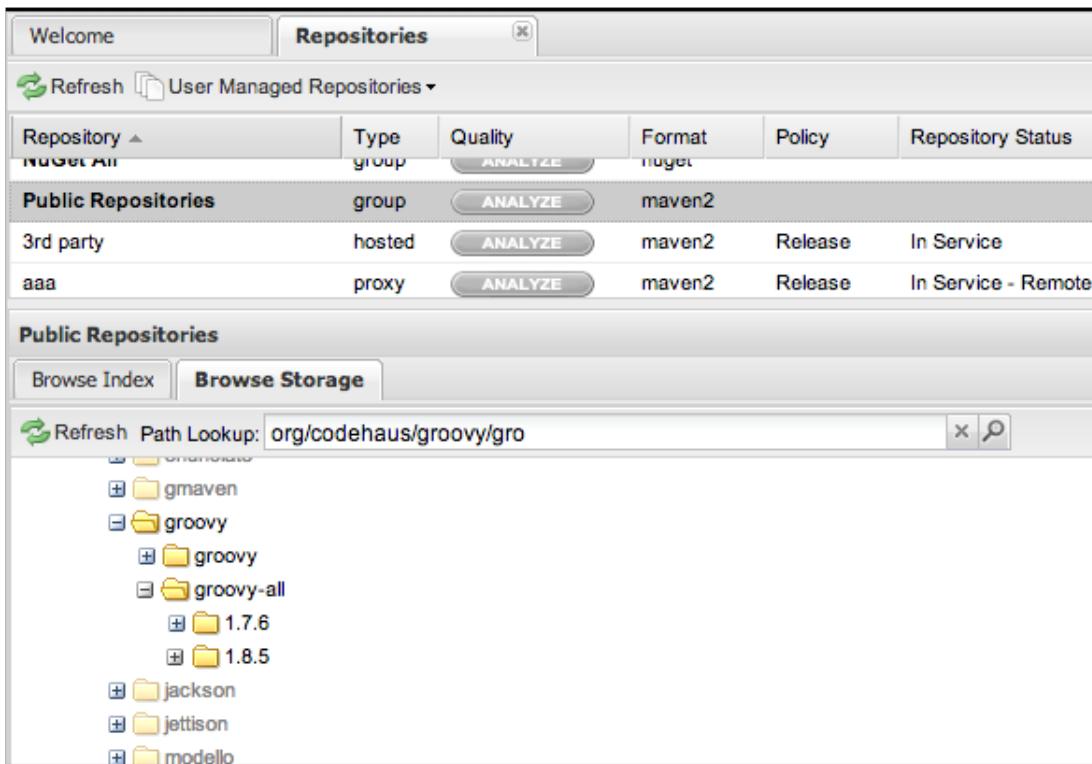


Figure 5.11: Browsing a Nexus Group

When browsing a Nexus group's storage, you are browsing the underlying storage for all of the repositories which are contained in a group. If a Nexus group contains proxy repositories, the Browse Storage tab will show all of the artifacts in the Nexus group that have been download from the remote repositories. To browse and search all artifacts available in a Nexus group, click on the Browse Index tab to load the interface shown in Figure 5.12.

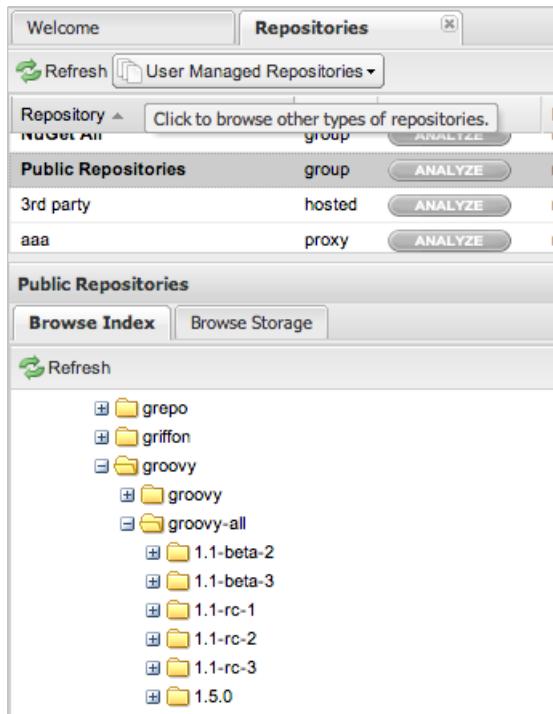


Figure 5.12: Browsing a Nexus Group Index

5.4 Searching for Artifacts

5.4.1 Search Overview

In the left-hand navigation area, there is an Artifact Search text field next to a magnifying glass. To search for an artifact by groupId or artifactId, type in some text and click the magnifying glass. Typing in the search term "junit" and clicking the magnifying glass should yield a search result similar to Figure 5.13.

The screenshot shows the Nexus Repository Manager interface. At the top, there is a search bar with the query "junit". Below the search bar is a table with columns: Group, Artifact, Version, Most Popular Version, and Download. The table contains several rows, with the first row being "junit" (Artifact: junit-dep, Version: 4.10, Most Popular Version: 4.10, Download: pom). A message "Too many results, please refine the search condition." is displayed above the table. Below the table, it says "Displaying Top 36 records". On the left, there is a tree view of the repository structure under "Central", showing the "junit" group with versions 3.7, 3.8, 3.8.1, 3.8.2, 4.0, 4.1, 4.10, and 4.11, along with their respective artifacts (javadoc.jar, sources.jar, jar). On the right, there is a detailed view for the selected "junit" entry (Version 4.11). It shows the Maven tab selected, with fields for Group (junit), Artifact (junit), Version (4.11), Extension (jar), and XML (the corresponding Maven dependency XML code). There are also tabs for Artifact, Archive Browser, and Maven Dependency.

Figure 5.13: Results of an Artifact Search for "junit"

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

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

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

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

Each of the columns in the search results table can be used to sort the table in Ascending or Descending order. In addition you can choose to add and remove columns with the sort and column drop down options visible in Figure 5.14.

The screenshot shows a search results table for 'junit'. The table has columns for Version, Most Popular Version, and Download. A context menu is open over the 'Version' header, showing 'Sort Ascending' and 'Sort Descending' options. Another context menu is open over the 'Columns' button, listing various filter and sorting criteria. The 'Most Popular Version' checkbox is checked. Other visible checkboxes include 'Group', 'Artifact', 'Version', 'Age', 'Popularity', 'Security Issues', 'License Threat', and 'Download'. The 'Download' checkbox is also checked. The table displays several rows of artifact information, with the first row showing 'pom' and 'pom, jar, sources.jar, javadoc.jar'.

Figure 5.14: Sort and Column Options in the Search Results Table

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

Warning

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

5.4.2 Advanced Search

Clicking on the (Show All Versions) link in the Version column visible in Figure 5.13 will kick off an Advanced Search by the groupId and artifactId of the row and result in a view similar to Figure 5.15.

Group	Artifact	Version	Age	Popularity	Security Issues	License Threat	Download
junit	junit	4.10	1.1 yrs	<div style="width: 80%;">80%</div>	0	■	pom, jar
junit	junit	3.8.2	5.5 yrs	<div style="width: 60%;">60%</div>	0	■	pom, jar, sources.jar, javadoc.jar
junit	junit	4.8.2	2.1 yrs	<div style="width: 40%;">40%</div>	0	■	pom, jar, sources.jar, javadoc.jar
junit	junit	3.8.1	5.5 yrs	<div style="width: 20%;">20%</div>	0	■	pom, jar
junit	junit	4.8.1	2.7 yrs	<div style="width: 10%;">10%</div>	0	■	pom, jar, sources.jar
junit	junit	4.4	5.2 yrs	<div style="width: 5%;">5%</div>	0	■	pom, jar, sources.jar, javadoc.jar

Figure 5.15: Advanced Search Results for a GAV Search Activated by the Show All Versions Link

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

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

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

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

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

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

The following advanced searches are available:

Keyword Search

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

Classname Search

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

GAV Search

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

Checksum Search

Sometimes it is necessary to determine the version of a jar artifact in order to migrate to a qualified version. When attempting this and neither the filename nor the contents of the manifest file in the jar contain any useful information about the exact version of the jar you can use Checksum Search to identify the artifact. Create a sha1 checksum, e.g. with the sha1sum command available on Linux, and use the created string in a Checksum search. This will return one result, which will provide you with the GAV coordinates to replace the jar file with a dependency declaration.

Tip

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

5.4.3 Nexus OpenSearch Integration

OpenSearch a standard which facilitates searching directly from your browser's search box. If you are using Internet Explorer 7+ or Firefox 2+ you can add any Nexus instance as an OpenSearch provider.

Then you can just type in a search term into your browser's search field and quickly search for Maven artifacts. To configure OpenSearch, load Nexus in a browser and then click on the drop-down next to the search tool that is embedded in your browser. Figure 5.16 shows the Add Nexus option that is present in Firefox's OpenSearch provider drop-down.

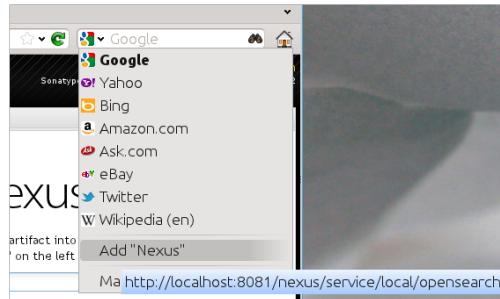


Figure 5.16: Configuring Nexus as an OpenSearch Provider

Once you have added Nexus to the list of OpenSearch providers, click on the drop-down next to the search term and select Nexus (localhost) from the list of OpenSearch providers. Type in a groupId, artifactId, or portion of a Maven identifier and press enter. Your opensearch-friendly web browser will then take you to the search results page of Nexus displaying all the artifacts that match your search term.

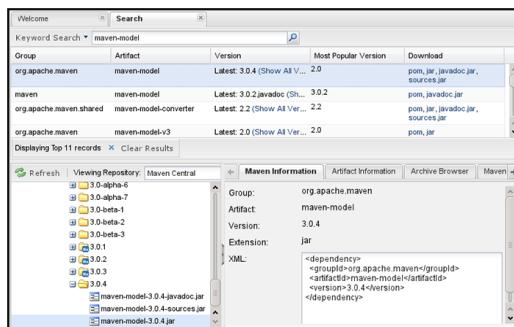


Figure 5.17: OpenSearch Search Results in Nexus

Once you have configured your browser to use Nexus as an OpenSearch provider, searching for a Maven artifact is as simple as typing in a groupId or artifactId, selecting Nexus from the drop-down shown in Figure 5.18, and performing a search.

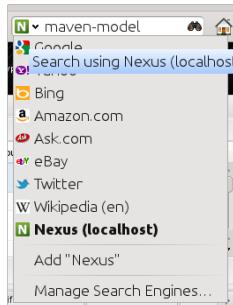


Figure 5.18: Nexus Available as an Option in the Firefox OpenSearch Provider List

5.5 Uploading Artifacts

When your build makes use of proprietary or custom dependencies which are not available from public repositories, you will often need to find a way to make them available to developers in a custom Maven repository. Nexus ships with a pre-configured 3rd Party repository that was designed to hold 3rd Party dependencies which are used in your builds. To upload artifacts to a repository, select a hosted repository in the Repositories panel and then click on the Artifact Upload tab. Clicking on the Artifact Upload tab will display the tab shown in Figure 5.19.

The screenshot shows the 'Artifact Upload' tab selected in the '3rd party' navigation bar. The 'Select GAV Definition Source' section contains a dropdown menu labeled 'GAV Definition: Select...'. Below it is a note: 'Select a source for the GAV definition. GAV can be specified either manually or from a POM file. These settings will be applied to all artifacts specified below.' The 'Select Artifact(s) for Upload' section includes a 'Select Artifact(s) to Upload...' button, fields for 'Filename', 'Classifier', and 'Extension', and an 'Add Artifact' button. A large list area titled 'Artifacts' displays a single entry, with 'Remove' and 'Remove All' buttons nearby. At the bottom are 'Upload Artifact(s)' and 'Reset' buttons.

Figure 5.19: Artifact Upload Form

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

If the artifact you are uploading is a JAR file that was created by Maven it will already have POM information embedded in it. If you are uploading a JAR from a vendor you will likely need to set the Group Identifier, Artifact Identifier, and Version manually. To do this, select GAV Parameters from the GAV Definition drop-down at the top of this form. Selecting GAV Parameters will expose a set of form fields which will let you set the Group, Artifact, Version, and Packaging of the artifacts being uploaded.

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

The Artifact Upload panel supports multiple artifacts with the same Group, Artifact, and Version identifiers. For example, if you need to upload multiple artifacts with different classifiers, you may do so by

clicking on Select Artifact(s) for Upload and Add Artifact multiple times.

5.6 Browsing System Feeds

Nexus provides feeds that capture system events, you can browse these feeds by clicking on System Feeds under the View menu. Clicking on System Feeds will show the panel in Figure 5.20. You can use these simple interface to browse the most recent reports of artifact deployments, cached artifacts, broken artifacts, and storage changes that have occurred in Nexus.

The screenshot shows the 'System Feeds' view in Nexus. At the top, there are 'Refresh' and 'Subscribe' buttons. Below that is a table with two columns: 'Feed' and 'URL'. The 'Feed' column lists several system events, and the 'URL' column provides the corresponding RSS feed URLs. The table has a scroll bar on the right. Below the table, there is a section titled 'New artifacts in all Nexus repositories (cached or deployed)' with a 'Date' dropdown. Underneath this section, a list of artifact deployments is shown, each with a title, date, and a brief description of the deployment action. The list includes entries for 'com.simpligility.maven:stagingexample:1.6.1', 'com.simpligility.maven:stagingexample:1.6', 'com.simpligility.maven:stagingexample:1.6-20120125.205105-2', and 'com.simpligility.maven:stagingexample:1.6-20120125.204958-1'.

Feed	URL
Authentication and Authorization Events	http://localhost:8081/nexus/service/local/rssfeeds/authentication_and_authorization_events
Broken artifacts in all Nexus repositories (checksum errors...)	http://localhost:8081/nexus/service/local/rssfeeds/broken_artifacts_in_all_nexus_repositories_checksum_errors...
Broken files in all Nexus repositories (checksum errors,...)	http://localhost:8081/nexus/service/local/rssfeeds/broken_files_in_all_nexus_repositories_checksum_errors,...
Error and Warning events	http://localhost:8081/nexus/service/local/rssfeeds/error_and_warning_events
New artifacts in all Nexus repositories (cached or deployed)	http://localhost:8081/nexus/service/local/rssfeeds/new_artifacts_in_all_nexus_repositories_cached_or_deployed
New cached artifacts in all Nexus repositories (cached)	http://localhost:8081/nexus/service/local/rssfeeds/new_cached_artifacts_in_all_nexus_repositories_cached

New artifacts in all Nexus repositories (cached or deployed).

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

Figure 5.20: Browsing Nexus System Feeds

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

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

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

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

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

5.7 System Files

The System Files is only visible to Administrative users under the Administrationmenu. Click on this option brings up the dialog shown in Figure 5.21. From this screen you can view the Nexus log file as well as the configuration files documented in Section 3.11.1.

The nexus.log file is the general application log for Nexus. Unless you are an administrative user, you might not have much interest in the information in this log. If you are trying to debug an error, or if you have uncovered a bug in Nexus, you can use this log viewer to diagnose problems with Nexus.

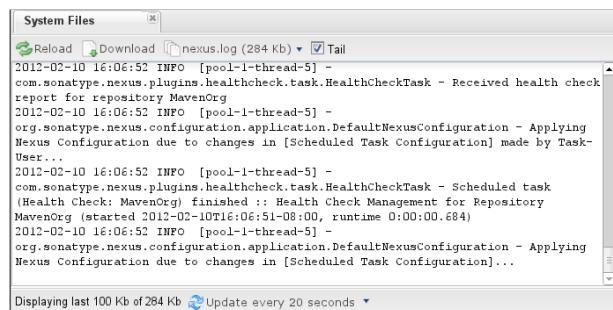


Figure 5.21: Browsing Nexus Logs and Configuration

You can load, view and download the various files by using the buttons and the dropdown to select the files to examine.

In Figure 5.21 there is a "tail" checkbox. If this box is checked, then Nexus will always show you the end of a log file. This can come in handy if you want to see a continuously updated log file. When this tail box is checked, a drop-down at the bottom of the panel allows you to set the update frequency. The contents of this drop-down are shown in Figure 5.22. If an update interval is selected, Nexus will periodically refresh the selected log file.

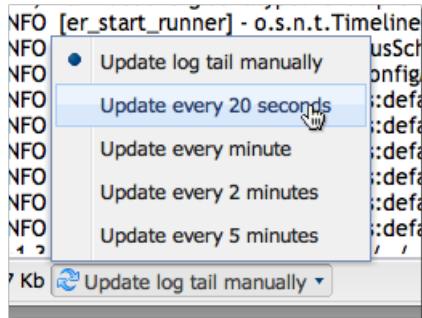


Figure 5.22: Selecting the Update Frequency when Tailing a Log File

5.8 Working with Your User Profile

As a logged in user, you can click on your user name in the top right hand corner of the Nexus user interface to expose a drop down with an option to Logout as well as to access your user Profile displayed in Figure 5.23.

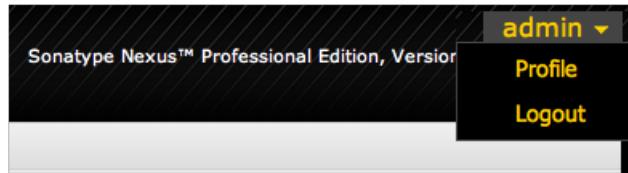


Figure 5.23: Drop Down on User Name with Logout and Profile Options

Once you have selected to display your profile you will get access to the Summary section of the Profile tab as displayed in Figure 5.24.

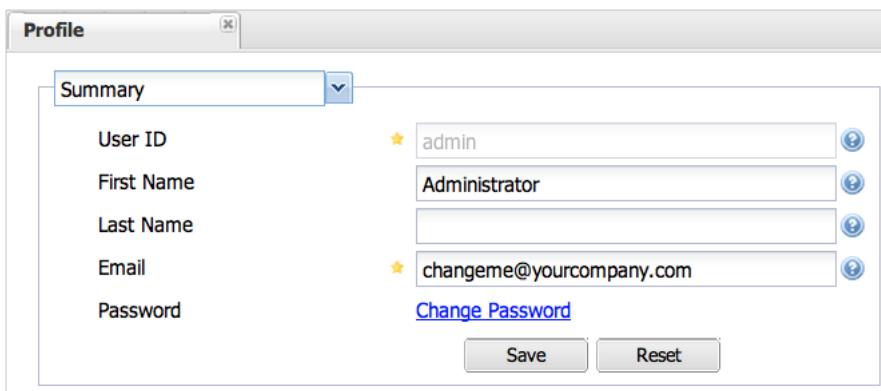


Figure 5.24: Summary Section of the Profile Tab

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

5.8.1 Changing Your Password

In addition to changing your name and email, the user profile allows you to change your password by clicking on the Change Password text. The dialog displayed in Figure 5.25 will be displayed and allow you to supply your current password, and choose a new password. When you click on Change Password, your Nexus password will be changed.



Figure 5.25: Changing Your Nexus Password

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

5.8.2 Additional User Profile Tabs

The Profile tab can be used by other plugins and features to change or access user specific data and functionality. One such use case is the User Token access documented in Section [6.15](#).

5.9 Filing a Problem Report

If you encounter a problem with Nexus, you can use the Nexus UI to report a bug or file an issue against the Nexus project in Sonatype's JIRA instance.

To file a problem report, you will first need to sign up for an account on <http://issues.sonatype.org>. You can click on Report Problem in the left hand Help menu, supply your Sonatype JIRA credentials, and file a problem report. Supply your JIRA username and password along with a short title and a description as shown in the following figure.

When you file a Nexus problem report, Nexus will create a new issue in JIRA and attach your configuration and logs to the newly filed issue.



Figure 5.26: Generating a Nexus Problem Report

The submitted files are stored in zip archive files in sonatype-work/nexus/error-report-bundles.

Chapter 6

Configuring Nexus

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

6.1 Customizing Server Configuration

In a production installation of Nexus, you'll probably want to customize the administrative password to something other than "admin123", and you might want to override the default directories that Nexus uses to store repository data. To do this, log in as the administrative user and click on Server under Administration in the left-hand navigation menu as visible in Figure 6.1.



Figure 6.1: Administration Menu in the Left Hand Panel

The server configuration screens subsections are documented in the following.

6.1.1 SMTP Settings

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

This section of the form takes an SMTP Hostname and Port as well as Username and Password. In addition you can enable SSL and/or TLS for the connection to the SMTP server. The System Email parameter defines the email address used in the From: header of an email from Nexus. Typically this would be configured as a "Do-Not-Reply" email address or a mailbox or mailing list monitored by the administrators of the Nexus server.

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

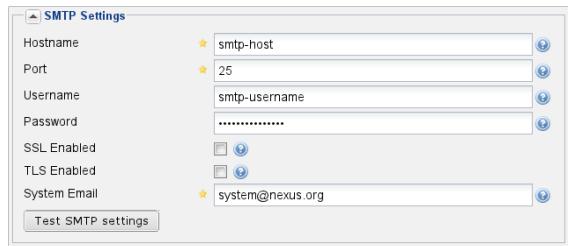


Figure 6.2: Administration SMTP Settings

6.1.2 HTTP Request Settings

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

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

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



Figure 6.3: Administration HTTP Request Settings

6.1.3 Security Settings

You can choose to enable or disable security, enable or disable anonymous access, and set the username and password for anonymous access. If you choose to enable security, you are telling Nexus to enforce role-based access control to enforce read and write access to repositories.

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

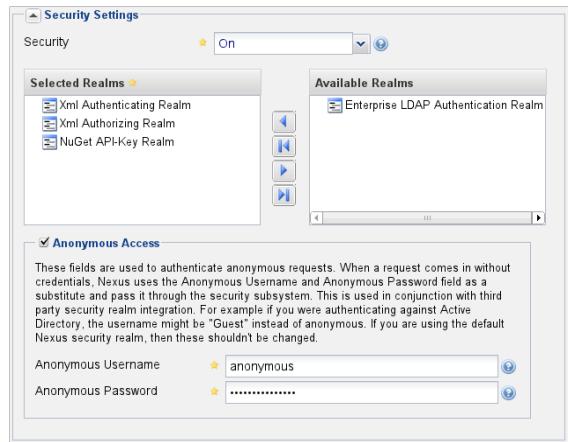


Figure 6.4: Administration Security Settings

6.1.4 Error Reporting Settings

Nexus can be configured to automatically file exception and error reports with the Nexus project in the Sonatype issue tracker. Activating this setting in your own Nexus installation helps to improve Nexus as the development team will receive automatic error reports if your Nexus instance experiences an error or a failure. The Nexus Server configuration's Automated Error Reporting Settings section is shown in Figure 6.5. This section accepts a JIRA username and password, and allows you to configure Nexus to use the default HTTP Proxy Settings when Nexus attempts to file an error report with the Sonatype issue tracker.

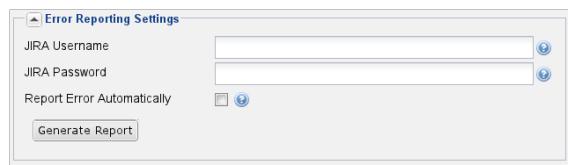


Figure 6.5: Administration Error Reporting Settings

To sign up for an account on the Sonatype JIRA instance, go to <http://issues.sonatype.org>. Once you see the web site shown in Figure 6.6, click on the "Signup" link below the Login form.

The screenshot shows the Sonatype Issue Tracker interface. At the top, there is a navigation bar with links for Dashboards, Projects, Issues, Agile, Test Sessions, and a Quick Search bar. On the left, there is a sidebar titled 'Projects' listing several projects: Aether Ant Tasks (AETHERANT), Book - M2Eclipse Book (MEBOOK), Book - Maven by Example (MVNEX), Book - Maven: The Complete Reference (MVNREF), and another entry for Book - Maven by Example (MVNEX). Each project entry includes a 'Lead' name. To the right of the sidebar is a 'Login' form. The 'Login' form contains fields for 'Username' and 'Password', a checkbox for 'Remember my login on this computer', and links for 'Sign Up for an account' and 'Can't access your account?'. Below the login form is an 'Activity Stream' section.

Figure 6.6: Sonatype Issue Tracker

Fill out the sign-up form shown in Figure 6.7, and choose a username and password. This is the username and password you should use in the Automated Error Reporting Settings section of the Server configuration shown in Figure 6.5.

The screenshot shows the Sonatype sign-up interface. At the top, there's a navigation bar with links for Dashboards, Projects, Issues, Agile, and Test Sessions. The main title "Sign up" is displayed prominently. A blue callout box contains the text: "To sign up for JIRA simply enter your details below." Below this, there are five input fields with asterisks indicating they are required: "Username", "Password", "Confirm Password", "Full Name", and "Email". Following these fields is a CAPTCHA challenge: "Please enter the word as shown below" with a text input field containing the word "futhers". At the bottom of the form are two buttons: "Sign up" and "Cancel".

Figure 6.7: Signing Up for a Sonatype Issue Tracker Account

6.1.5 Application Server Settings

This section allows you to change the Base URL for your Nexus installation. It is used when generating links in emails and RSS feeds. The Sonatype Nexus repository is available on <http://repository.sonatype.org>, and it makes use of this Base URL field to ensure that links in emails and RSS feeds point to the correct URL. If you are hosting Nexus behind a proxy server and you want to make sure that Nexus always uses

the specified Base URL, check the "Force Base URL" checkbox. If the Force Base URL is not checked, Nexus will craft URLs in HTTP responses based on the request URL, but it will use the Base URL when it is generating emails.

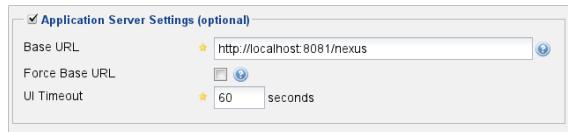


Figure 6.8: Administration Application Server Settings

TIP

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

6.1.6 Default HTTP Proxy Settings

If you Nexus instance needs to reach public repositories like the Central Repository via a proxy server, you can configure one or multiple servers in this settings section.

There are a number of HTTP Proxy settings for Nexus installations which need to be configured to use an HTTP Proxy. You can specify a host, port, and a number of authentication options which might be required by your proxy server.

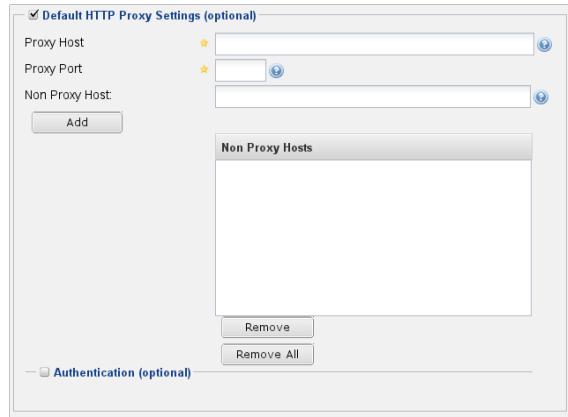


Figure 6.9: Administration Default HTTP Proxy Settings

Tip

This is a critical, initial step for Nexus deployment in many enterprise deployments of Nexus, since these environments are typically secured via a HTTP proxy server.

6.1.7 System Notification Settings

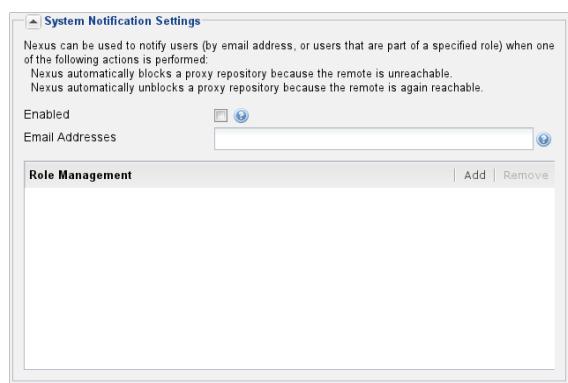


Figure 6.10: Administration System Notification Settings

6.1.8 PGP Key Server Information

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



Figure 6.11: Administration PGP Key Server Information

6.1.9 New Version Availability

Nexus can notify you of new versions of Nexus via the Nexus interface. To enable this feature, check the Enable checkbox in the New Version Notification section of the Nexus server settings as shown in Figure 6.12.



Figure 6.12: Administration New Version Availability

6.2 Managing Repositories

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

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

6.2.1 Proxy Repository

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

Apache Snapshots

This repository contains snapshot releases from the Apache Software Foundation <http://repository.apache.org/snapshots/>

Codehaus Snapshots

This repository contains snapshot released from Codehaus <http://nexus.codehaus.org/snapshots/>

Central

This is the central repository (for releases). For Nexus OSS the URL <http://repo1.maven.org/maven2/> is used, while Nexus Professional has the SSL secured version <https://secure.central.sonatype.com/maven2/> preconfigured.

6.2.2 Hosted Repository

A hosted repository is a repository which is hosted by Nexus. Maven ships with the following configured hosted repositories:

3rd Party

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

Releases

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

Snapshots

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

6.2.3 Virtual Repository

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

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

6.2.4 Configuring Repositories

The Repositories window displayed in Figure 6.13 allows you to create, update and delete different repositories with the Add, Delete and Trash button. Use the Refresh button to update the displayed list of repositories and repository groups. The Trash button allows you to empty the trash folder into which deleted components are copied, when any delete operations are performed from the Nexus user interface.

By default the list of repositories displays the repositories configured and managed by the administrator. The drop down on the right of the Trash button allows you to switch the list of repositories and view the repositories managed by Nexus. There are staging repositories as documented in Chapter 10 or procurement repositories as documented in Chapter 9.

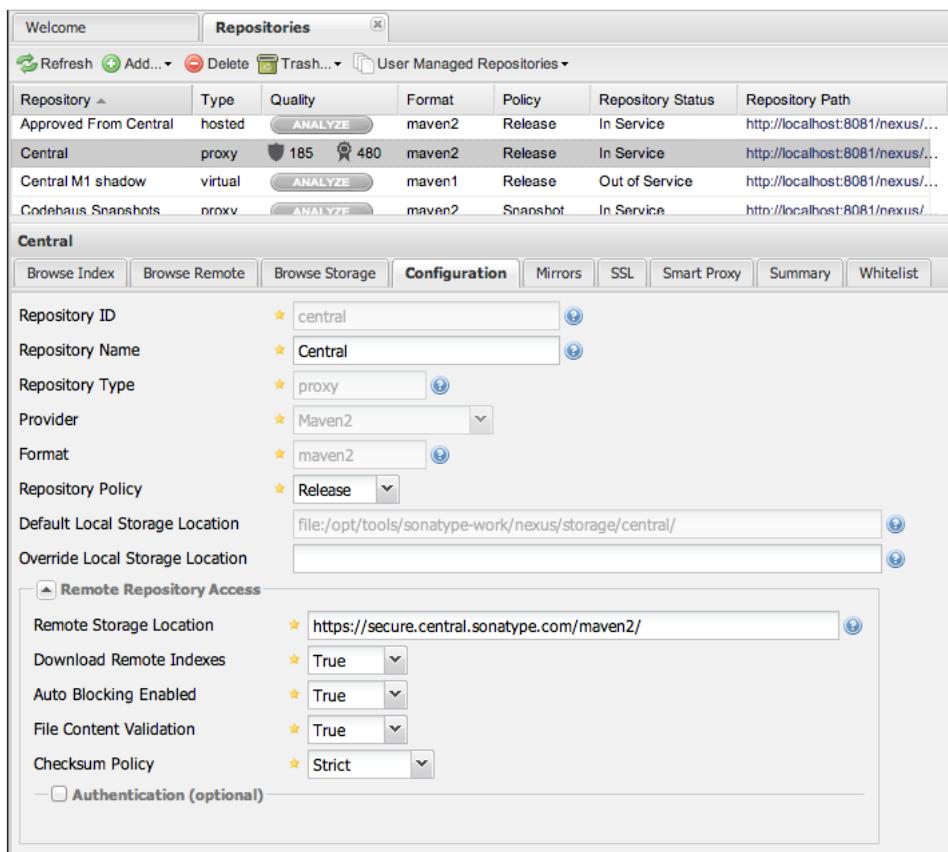


Figure 6.13: Repository Configuration Screen for a Proxy Repository

The list of repositories visible in Figure 6.13 allows you to access more details for each repository by selecting a specific row and displays some information for each repository in the following columns:

Repository

the name of the repository with repository groups displayed in bold

Type

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

Quality

a button to trigger the creation or access the results of a repository health check as documented in Chapter 11

Format

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

Policy

the deployment policy that applies to this repository. Not all repository policies. The typical Maven format allows Snapshot and Release policies.

Repository Status

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

Repository Path

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

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

If you perform a right clicking on a row you can trigger a number of actions on the current repository. These actions depend on the repository type and include:

Expire Cache

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

Rebuild Metadata

rebuild the metadata of a hosted Maven 2 repository

Block Proxy / Allow Proxy

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

Put Out Of Service / Put in Service

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

Repair Index / Update Index

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

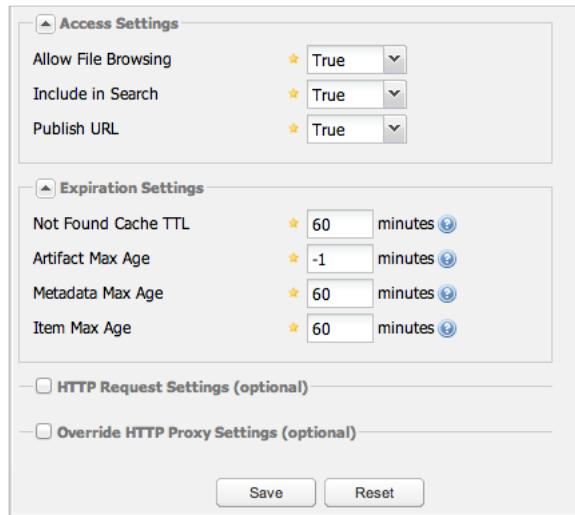


Figure 6.14: Repository Configuration Screen for a Proxy Repository

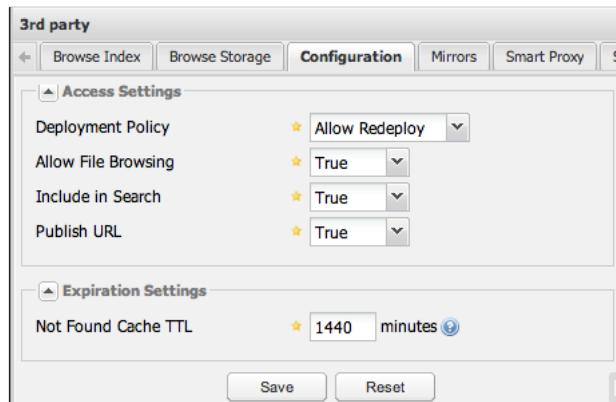


Figure 6.15: Repository Configuration Access Settings for a Hosted Repository

Figure 6.13 and Figure 6.14 show the Repository configuration screen for a Proxy repository in Nexus. From this screen, you can manage the settings for proxying an external repository:

Repository ID

The repository ID is the identifier which will be used in the Nexus URL. For example, the central

proxy repository has an ID of "central", this means that maven can access the repository directly at <http://localhost:8081/nexus/content/repositories/central>. The Repository ID must be unique in a given Nexus installation. ID is required.

Repository Name

The display name for a repository. Name is required.

Repository Type

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

Provider and Format

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

Repository Policy

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

Default Storage Location

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

Override Storage Location

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

Remote Repository Access

This section tells Nexus where to look for and how to interact with the remote Maven repository being proxied.

Remote Storage Location

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

Download Remote Indexes

This field controls the downloading of the remote indexes. If enabled, Nexus will download the index, if it exists, and use that for its searches as well as serve that up to any clients which ask for the index (like m2eclipse). The default for new proxy repositories is enabled, but all of the default repositories included in Nexus have this option disabled. To change this setting

for one of the proxy repositories that ship with Nexus, change the option, save the repository, and then re-index the repository. Once this is done, artifact search will return every artifact available on the Maven Central repository.

Auto Blocking Enabled

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

File Content Validation

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

Checksum Policy

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

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

Authentication

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

Access Settings

This section configures access settings for a repository.

Deployment Policy

This setting controls how a Hosted repository allows or disallows artifact deployment. If this policy is set to "Read Only", no deployment is allowed. If this policy is set to "Disable Redeploy", a client can only deploy a particular artifact once and any attempt to redeploy an artifact will result in an error. If this policy is set to "Allow Redeploy", clients can deploy artifacts to this repository and overwrite the same artifact in subsequent deployments. This option is visible for Hosted repositories as shown in Figure 6.15.

Allow File Browsing

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

Include in Search

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

Publish URL

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

Expiration Settings

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

Not Found Cache TTL

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

Artifact Max Age

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

Metadata Max Age

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

Item Max Age

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

HTTP Request Settings

This section lets you change the properties of the HTTP request to the remote repository. In this section you can configure the User Agent of the request, add parameters to a request, and set the timeout and retry behaviour. This section refers to the HTTP request made from Nexus to the remote Maven repository being proxied.

Override HTTP Proxy Settings

This section lets you configure the HTTP Proxy for the request from Nexus to the remote repository. You can configure a proxy host and port plus an authentication settings you need tell Nexus to use an HTTP Proxy for all requests to a remote repository.

6.2.5 Selecting Mirrors for Proxy Repositories

Nexus also allows you to select which mirrors Nexus will consult for a particular Proxy repository. Clicking on the Mirrors tab will show the figure shown in Figure 6.16.

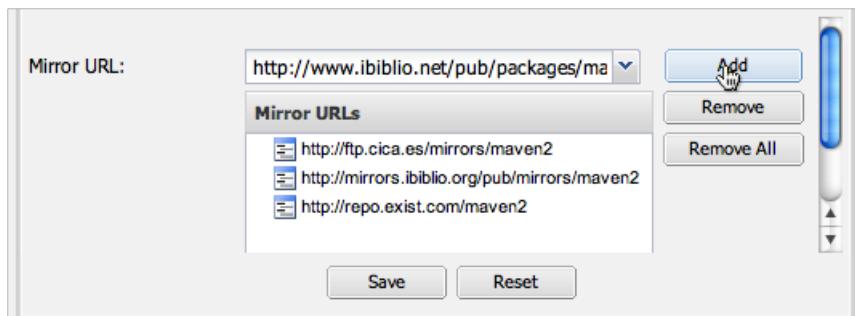


Figure 6.16: Configuring Mirrors for Proxy Repositories

To configure a mirror repository, click on the Mirror URL drop-down and select a mirror for the Proxy repository. Click the Add button, and Nexus will then be configured to download artifacts from the selected mirror. Nexus will always download checksums and metadata from the original (or Canonical) URL for a proxy repository. For example, if Nexus is going to download an artifact, it will retrieve the MD5 checksum from the original Maven Central repository and then retrieve the artifact from the selected mirror.

6.2.6 Adding a Mirror Entry for a Hosted Repository

If you are logged in as a user with Administrative privilege, there will be a Mirrors tab available when you are viewing a Hosted repository, clicking on this Mirrors tab will show the form shown in Figure 6.16. This tab contains a list of mirror URLs for this hosted repository. If there are other sites which mirror the contents of this hosted repository, this tab allows you to populate the repository mirror metadata with those URLs.

This repository mirror metadata can then be consumed by other systems that interact with your hosted repository. For example, if you have a release repository which is used by your customers or by the general public, if one of people consuming your Hosted repository is also running a Nexus, they can configure a Proxy repository that targets your Hosted repository and they can use the mirror metadata to configure their instance of Nexus to consume artifacts from mirrors of your Hosted repository.

6.2.7 Viewing Repository Summary Panel

The Repository Summary panel can be loaded by selecting a Hosted, Proxy, or Virtual repository and then clicking on the Summary tab. When viewing the Summary tab of a Hosted repository, as shown in Figure 6.17, you will also see the Distribution Management settings which can be used to configure Maven to publish artifacts to a Hosted repository.

The screenshot shows the Nexus interface with the 'Repositories' tab selected. In the main pane, there is a table of repositories:

Repository	Type	Quality	Format	Policy	Repository Status
NuGet Public Group	group	UNAVAILABLE	nuget		
OurPublicAPIs	group	UNAVAILABLE	maven2		
Public Repositories	group	UNAVAILABLE	maven2		
3rd party	hosted	UNAVAILABLE	maven2	Release	In Service
Apache Snapshots	proxy	UNAVAILABLE	maven2	Snapshot	In Service

Below the table, the 'Summary' tab is active. The 'Repository Information' section displays details for the selected '3rd party' repository, including its ID, name, type, policy, and format. The 'Distribution Management' section shows the XML configuration for publishing artifacts to this repository.

Figure 6.17: Repository Summary Panel for a Hosted Repository

The Repository Summary panel for a Proxy repository, as shown in Figure 6.18, contains all of the repository identifiers and configuration as well as a list of groups, in which the repository is contained.

The screenshot shows the Nexus interface with the 'Repositories' tab selected. In the main pane, there is a table of repositories:

Repository	Type	Quality	Format	Policy	Repository Status
Javav	hosted	UNAVAILABLE	maven2	Release	In Service
ksoap2-android	proxy	ANALYZING	maven2	Release	In Service
Maven Central	proxy	ANALYZING	maven2	Release	In Service

Below the table, the 'Summary' tab is active. The 'Repository Information' section displays details for the selected 'Maven Central' repository, including its ID, name, type, policy, and format. The 'Distribution Management' section is present but appears to be empty or not applicable for this type of repository.

Figure 6.18: Repository Summary Panel for a Proxy Repository

The Repository Summary panel for a Virtual repository, as shown in Figure 6.19, displays repository identifiers and configuration.

The screenshot shows the Nexus Repository Management interface. At the top, there's a toolbar with 'Welcome', 'Repositories', 'Refresh', 'Add...', 'Delete', 'Trash...', and 'User Managed Repositories'. Below the toolbar is a table titled 'Repositories' with columns: Repository, Type, Quality, Format, Policy, and Repository Status. Three repositories are listed: 'Central M1 shadow' (Type: virtual, Quality: UNAVAILABLE, Format: maven1, Policy: Release, Status: In Service), 'Codehaus Snapshots' (Type: proxy, Quality: UNAVAILABLE, Format: maven2, Policy: Snapshot, Status: In Service), and 'Glassfish' (Type: proxy, Quality: ANALYZE, Format: maven2, Policy: Release, Status: In Service). A large central panel is titled 'Central M1 shadow' and contains tabs for 'Summary' (which is selected) and 'Smart Proxy'. Under 'Summary', it shows 'Repository Information' with details: Repository ID: central-m1, Repository Name: Central M1 shadow, Repository Type: virtual, Repository Policy: Release, Repository Format: maven1, and Contained in groups: (empty).

Figure 6.19: Repository Summary Panel for a Virtual Repository

6.2.8 Accessing The Central Repository Securely

One part of component lifecycle management is securing your component supply chain. The most important and widely used source for components for Java development and beyond is the Central Repository available at <http://search.maven.org>. It is the preconfigured default repository in Apache Maven and easily configured in other build systems as well.

Nexus Professional supports access to the Central Repository using HTTPS. This secure access to the Central Repository is the default configuration for Nexus Professional 2.2 and newer. It prevents anybody from gaining insight into the components you are downloading as well as compromising these components via Cross Build Injection XBI attacks.

The Remote Storage Location configured for the "Central" proxy repository is "<https://secure.central.sonatype.com/maven2>" as displayed in Figure 6.20.

Central

Configuration

Repository ID: central

Repository Name: Central

Repository Type: proxy

Provider: Maven2

Format: maven2

Repository Policy: Release

Default Local Storage Location: file:/opt/tools/sonatype-work/nexus/storage/central/

Override Local Storage Location:

Remote Repository Access

Remote Storage Location: https://secure.central.sonatype.com/maven2/

Download Remote Indexes: True

Auto Blocking Enabled: True

File Content Validation: True

This screenshot shows the configuration for the 'Central' repository in Nexus. The 'Configuration' tab is selected. The repository is set up as a proxy (Type: proxy) for the Maven2 provider (Format: maven2). It uses a local storage location at /opt/tools/sonatype-work/nexus/storage/central/. The 'Remote Repository Access' section is expanded, showing the remote storage location as https://secure.central.sonatype.com/maven2/, and download remote indexes, auto blocking, and file content validation are all enabled (set to True).

Figure 6.20: Default Configuration for the Central Repository Using HTTPS

The secure connection relies on an authentication token as well as Nexus running on a JVM with high-strength RSA cipher keys. The status of the secured access to the Central Repository can be inspected by accessing the "Secure Central" capability displayed in Figure 6.21.

The screenshot shows the Nexus interface for managing repositories. At the top, there is a toolbar with icons for Refresh, Add, Delete, and Show advanced. Below the toolbar is a table listing repositories:

Enabled	Active	Type	Description	Notes
true	true	Smart Proxy: Messaging	ID:Manfreds-MacBook-Pro.local-49485-135213...	
true	true	Secure Central	Ready: true, Tracked: 1	Auto...
true	true	User Token	Records: 1	Auto...
true	true	Smart Proxy: Publish	snapshots	
true	true	Smart Proxy: Secure Connector	ssl://0.0.0.0:0	

Below the table, a specific repository configuration is shown for "Secure Central - Ready: true, Tracked: 1". The configuration includes:

- Enabled:** checked
- Active:** checked
- Type:** Secure Central
- Notes:** Automatically added on Thu Oct 25 19:12:51 PDT 20: [details]

A tooltip for the "Type" field explains: "Automatically configures proxy repositories which point to Secure Central with the appropriate authentication parameters."

The interface also displays sections for **Settings** (Auth token) and **Status** (Configuration, Auth token, Tracked Repositories).

Figure 6.21: Secure Central Capability

You can use the secure connection to the Central Repository on a version of Nexus that was either upgraded from Nexus Open Source or from an older version, where the Central location was "<http://repo1.maven.org/maven2>". On Nexus 2.2 and newer you simply replace the Remote Storage Location for the "Central" proxy repository with "<https://secure.central.sonatype.com/maven2/>". The authentication token will automatically be requested and configured.

The secure access can be used on older versions of Nexus as well, although the preferred approach is to update to Nexus 2.2 or higher. If you require secure access to the Central Repository on an older version of Nexus please contact Sonatype support to receive your authentication token and configuration instructions.

6.2.9 Auto Block/Unblock of Remote Repositories

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

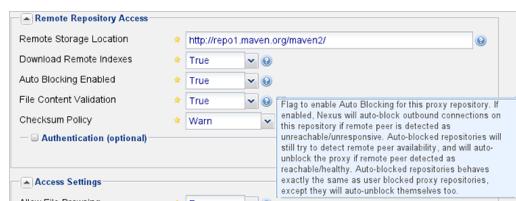


Figure 6.22: Configuring Remote Repository Auto Block/Unblock

6.3 Managing Groups

Groups are a powerful feature of Nexus. They allow you to combine multiple repositories and other repository groups in a single URL. Use the left hand panel Repositories menu item in the Views/Repositories menu to access the repositories and groups management interface.

Nexus ships with one group: public. The Public Repositories group combines the multiple important external proxy repositories like the Central Repository with the hosted repositories: 3rd Party, Releases, and Snapshots.

In Section 4.2 we configured Maven via the settings.xml to look for artifacts in the public group managed by Nexus. Figure 6.23 shows the group configuration screen in Nexus, in this figure you can see the contents of the public

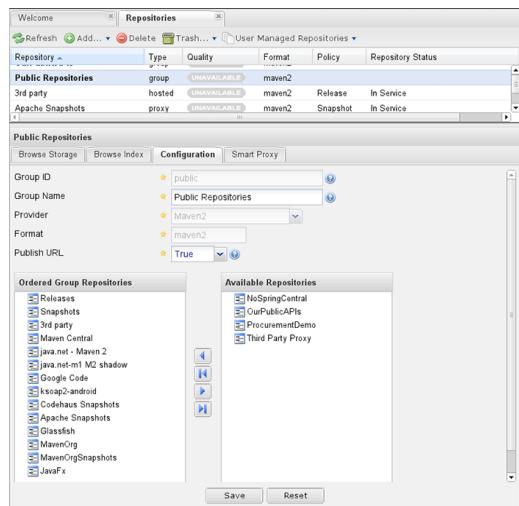


Figure 6.23: Group Configuration Screen in Nexus

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

The order of repositories or other groups in a group can be used to influence the effective metadata that will be retrieved by Maven from a Nexus Repository Group. We recommend placing release repositories higher in the list than snapshot repositories so that LATEST and RELEASE versions are merged appropriately.

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

6.4 Managing Routing

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

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

6.4.1 Automatic Routing

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

The Routing information consists of the top two levels of the directory structure of the repository and is stored in a prefixes.txt file. It allows Nexus to automatically route only component requests with the corresponding groupId values to a repository avoid unnecessary index or even remote repository access.

Nexus generates the prefixes.txt file for a hosted repository and makes it available for remote downloads. Each deployment of a new component will trigger an update of the file for the hosted repository as well as the prefix files for any repository groups that contain the hosted repository. You can access it in the Routing tab of a hosted repository as displayed in Figure 6.24 by clicking on the *Show prefix file* link on the right. In addition the Publishing section shows the *Status* of the routing information, a *Message* with further details and the date and time of the last update in the *Published On* field.

The screenshot shows the Nexus interface for managing repositories. At the top, there's a navigation bar with tabs for 'Releases', 'hosted', 'ANALYZE' (which is highlighted), 'maven2', 'Release', and 'In Service'. Below this, a table lists repositories: 'Site Internal' (hosted, ANALYZE, site, In Service) and 'Sonatype' (hosted, ANALYZE, maven2, Snapshot, In Service). A 'Releases' tab is selected, showing sub-tabs for 'Browse Index', 'Browse Storage', 'Configuration', 'Mirrors', 'Routing' (which is highlighted in blue), 'Smart Proxy', and 'Summary'. Under the 'Routing' tab, there's a 'Publishing' section with the following details:

Status:	Published.
Message:	Prefix file published successfully.
Published on:	Mon Mar 25 2013 13:48:47 GMT-0700 (PDT)

Next to the 'Published on' timestamp is a link labeled 'Show prefix file'.

Figure 6.24: Automatic Routing for a Hosted Repository

The Routing tab for a proxy repository displayed in Figure 6.25 contains the Discovery section. It displays the *Status* and a more detailed *Message* about the prefix file access. The *Last run* field displays the date

and time of the last execution of the prefix file discovery. Such an execution can be triggered by pressing the *Update now* button. Otherwise the *Update Interval* allows you to trigger a new discovery every one, two, three, six, nine or twelve hours or as a daily or weekly execution.

The screenshot shows the Nexus Central interface. At the top, there are tabs for Central, proxy, 188, 505, maven2, Release, and In Service. Below this, a table lists 'Central' and 'Central M1 shadow' with columns for status (virtual), last update (ANALYZE), and mirrors (maven1). A prominent 'ANALYZE' button is highlighted. Below the table, the 'Central' tab is selected, showing the 'Routing' section. The 'Discovery' section is checked. It displays the status as 'Successful.', the message as 'Remote publishes prefix file (is less than a day old), using it.', the last run as 'Tue Apr 02 2013 09:39:44 GMT-0700 (PDT)', and the update interval as 'Daily'. A 'Show prefix file' link is also present. A 'Update now' button is located at the bottom left of the 'Discovery' section.

Figure 6.25: Automatic Routing for a Proxy Repository

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

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

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

6.4.2 Manual Routing Configuration

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

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

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

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

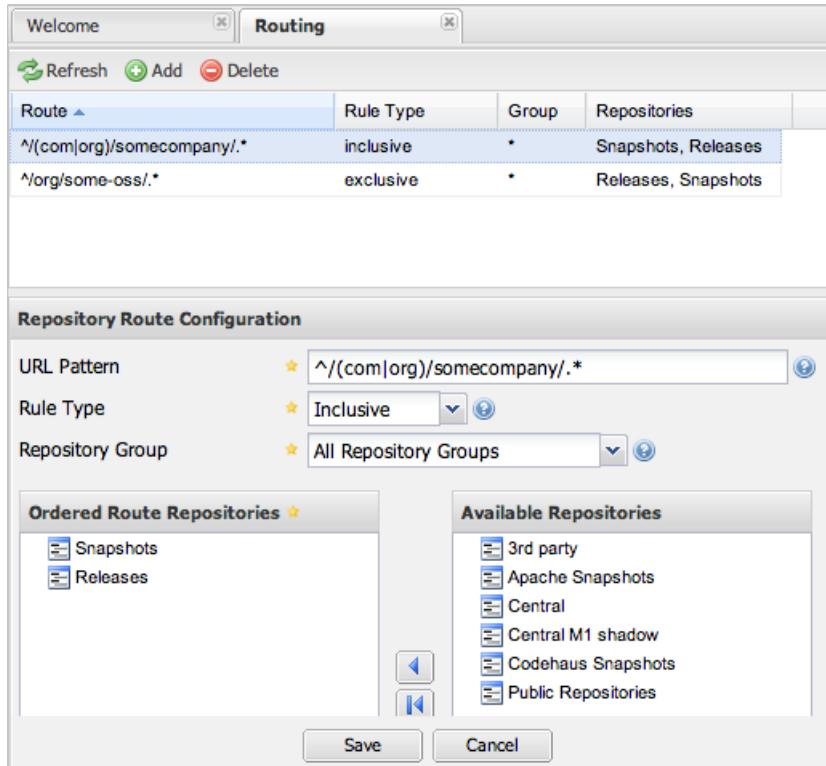


Figure 6.26: Routing Configuration Screen in Nexus

Figure 6.26 shows the Routing configuration screen. Clicking on a route will bring up a screen which will allow you to configure the properties of a route. The configuration options available for a route are:

URL Pattern

This is the pattern which Nexus will use to match a request to Nexus. If the regular expression in this pattern is matched, Nexus will either include or exclude the listed repositories from a particular artifact query. In Figure 6.26 the two patterns are:

".*/(com|org)/somecompany/.*"

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

".*/org/some-oss/.*"

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

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

Example "(?!/org/some-oss/.)."**

Using this pattern in an exclusive route allows you to exclude everything, but the "org/some-oss" project(s).

Rule Type

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

Ordered Route Repositories

This is an ordered list of repositories which Nexus will search to locate a particular artifact. Nexus searches top to bottom; if it's looking for an artifact, it will return the first match. When Nexus is looking for metadata, all repositories in a group are checked and the results are merged. The merging is applied giving preference to the earlier repositories. This is relevant when a project is looking for a LATEST or a RELEASE version. Within a Nexus Group, you should define the release repositories before the snapshot repositories, otherwise LATEST may incorrectly resolve to a snapshot version.

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

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

Tip

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

What if there is a conflict between two routes? Nexus will process inclusive routes before it will process the exclusive routes. Remember that routes only affect Nexus' resolution of artifacts when it is searching a Group. When Nexus starts to resolve an artifact from a repository group it will start with the list of repositories in a group. If there are matching inclusive routes, Nexus will then take the intersection of the repositories in the group and the repositories in the inclusive route. The order as defined in the group will not be affected by the inclusive route. Nexus will then take the result of applying the inclusive route and apply the exclusive route to that list of repositories. The resulting list is then searched for a matching artifact.

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

Tip

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

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

Tip

Check out Chapter 9 for fine grained control of artifact availability and use blocking routes sparingly.

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

6.5 Managing Scheduled Tasks

Nexus allows you to schedule tasks that will be applied to all repositories or to specific repositories on a configurable schedule. Use the Scheduled Tasks menu item in the Administration menu visible in Figure 6.1 to access the screen shown in Figure 6.27, that allows you to manage your Scheduled Tasks.

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

Figure 6.27: Managing Nexus Scheduled Tasks

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

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

Enabled

allows you to enable or disable a specific task

Name

provide a name to identify the task in the user interface

Task Type

specify the type of action the scheduled task can execute. The list of available task type is documented in more detail below.

Task Settings

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

Alert Email

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

Recurrence

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

The following kinds of scheduled task types are available:

Backup all Nexus Configuration Files (Nexus Professional only)

This scheduled task will archive the contents of the sonatype-work/nexus/conf directory. Once a backup has been run, the contents of the backup will be available in sonatype-work/nexus/backup in a series of ZIP archives which include the date and a timestamp.

Download Indexes

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

Download NuGet Feed

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

Drop Inactive Staging Repositories

Staging repositories can be dropped by user interaction or automated systems using the Nexus Staging Maven Plugin or Ant Task or a REST API call. Heavy users of the Nexus staging features observe that some staging repositories are inevitably left behind. This scheduled task can be used to drop these repositories. You can configure the duration of inactivity in days after which the repositories should be dropped as well as the status of the repositories to include in the check. Any change of the staging repository like a state change from open to closed to promoted or released as well other changes to the repository meta data like a description update are counted as an activity.

You can configure to scan open, closed and released repositories for inactivity and therefore potentially drop them with this task. This will allow you to avoid accumulating a large number of stale staging repositories.

Empty Trash

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

Evict Unused Proxied Items From Repository Caches

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

Expire Repository Caches

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

Yum: Generate Metadata

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

Optimize Repository Index

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

Rebuild P2 metadata and Rebuild P2 repository

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

Publish Indexes

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

Purge Orphaned API Keys

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

Purge Nexus Timeline

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

Rebuild Maven Metadata Files

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

Rebuild NuGet Feed

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

Remove Releases From Repository

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

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

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

Optionally the Repository Target parameter can be used to narrow down the content of the repository that is analysed, to determine if any deletion should occur. Choosing All (Maven2) is suitable to cause the full repository to be analysed. If you want to only target a specific groupId and artifactId combination or a number of them you can create a suitable repository target as documented in Section [6.9](#) and use it in the configuration of the scheduled task.

Repair Repositories Index

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

Remove Snapshots from Repository

Often, you will want to remove snapshots from a snapshot repository to preserve storage space. Note that configuring and running this job is not enough to reclaim disk space. You will also need to configure a scheduled job to empty the trash folder. Files are not deleted by the Remove Snapshots job, they are only moved into the Trash folder. When you create a scheduled task to remove snapshots, you can specify the Repository/Group to affect as well as:

Minimum Snapshot Count - This configuration option allows you to specify a minimum number of SNAPSHOTs to preserve per artifact. For example, if you configured this option with a value of 2, Nexus will always preserve at least two SNAPSHOT artifacts. -1 indicates to preserve all SNAPSHOTs.

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

Remove if released - If enabled and a released artifact with the same GAV coordinates is detected all SNAPSHOTs will be removed.

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

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

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

Synchronize Shadow Repository

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

Update Repositories Index

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

Mirror Eclipse Update Site (Nexus Professional only)

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

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

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

Tip

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

Setting up scheduled tasks adapted to your usage of Nexus is an important first step when setting up a Nexus instance. Go through the list of task types and consider your usage patterns of Nexus. Also update your scheduled tasks when changing e.g. from not deploying SNAPSHOTS to running deployments of a CI server or when introducing usage of user tokens with a strict LDAP password change policy.

6.6 Accessing and Configuring Capabilities

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

**Warning**

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

Nexus Professional ships with a number of capabilities pre-installed and allows you to enable/disable them. An example capability is the Outreach Management displayed in Figure 6.28. The Capabilities management interface supports adding new capabilities by pressing the Add button and deleting a selected capability with the Delete button.

The screenshot shows the Nexus Capabilities Management interface. At the top, there's a toolbar with Refresh, Add, and Delete buttons. Below it is a table with columns: Enabled, Active, Type, Description, and Notes. A single row is selected, showing 'true' for both Enabled and Active, 'Outreach : Management' for Type, and a long hex string for Description. The Notes column contains a truncated message about automatically adding capabilities.

Below the table, a section titled 'Outreach : Management -' is expanded. It contains fields for Enabled (checked), Active (checked), Type (selected to 'Outreach : Management'), and Notes (empty). To the right of these fields is a tooltip providing a detailed description of the Outreach plugin and its default base URL.

At the bottom of this section is a 'Settings' tab, which includes fields for Base URL (set to 'http://links.sonatype.com/products/nexus/outreach'), Disable Caching (unchecked), and Override URL (empty).

Figure 6.28: Capabilities Management Interface with the Outreach Management Details Visible

Every capability can be enabled or disabled with the Enabled checkbox. The Active checkbox shows if the capability is currently operating correctly. It can not be changed by the user. If the checkbox is deselected the capability is inactive and a help text will display the reason for that status. Depending on the capability the reasons can vary widely. For example the Secure Central capability requires Nexus to run on a JVM with specific security features and an error message will indicate, if the JVM is not suitable and an error message regarding this will be displayed. In addition the Type of the capability is displayed in the drop down and Notes can be added and edited in the input field.

A small description of the capability is displayed on the right hand side of the generic configuration.

Creating a new capability by pressing the Add button will display a new form allowing you to configure the capability. You can configure if the capability should be enabled with the Enabled checkbox. The Notes field allows you to provide a simple text note, that is visible in the list of capabilities. The main configuration is the Type selector, which determines what further customization can be done specific to the type of capability and will provide the necessary user interface components in the capability configuration section below the Notes input.

Many of the built-in capabilities can be configured in the Capabilities administration section, but also in other more user friendly, targetted user interface sections. E.g. the User Token feature administrated by using the interface available via the User Token menu item in the Security left hand menu as well as by editing the User Token capability. Other capabilities are internal to Nexus functionality and sometimes managed automatically by the responsible plugin.

6.7 Managing Security

Nexus has role-based access control (RBAC) which gives administrators very fine-grained control over who can read from a repository (or a subset of repositories), who can administer the server, and who can deploy to repositories. The security model in Nexus is also so flexible as to allow you to specify that only certain users or roles can deploy and manage artifacts in a specific repository under a specific groupId or asset class. The default configuration of Nexus ships with four roles and four users with a standard set of permissions that will make sense for most users. As your security requirements evolve, you'll likely need to customize security settings to create protected repositories for multiple departments, or development groups. Nexus provides a security model which can adapt to any scenario. The Security configuration is done via menu items in the left hand Security menu.

Nexus' Role-based access control (RBAC) system is designed around the following four security concepts:

Privileges

Privileges are rights to read, update, create, or manage resources and perform operations. Nexus ships with a set of core privileges which cannot be modified, and you can create new privileges to allow for fine-grained targeting of role and user permissions for specific repositories.

Targets

Privileges are usually associated with resources or targets. In the case of Nexus, a target can be a specific repository or a set of repositories grouped in something called a repository target. A target can also be a subset of a repository or a specific asset classes within a repository. Using a target you can apply to a specific privilege to apply to a single groupId.

Roles

Collections of privileges can be grouped into roles to make it easier to define collections of privileges common to certain classes of users. For example, deployment users will all have similar sets of permissions. Instead of assigning individual privileges to individual users, you use Roles to make it easier to manage users with similar sets of privileges. A role has one or more privilege and/or one or more roles.

Users

Users can be assigned roles and privileges, and model the individuals who will be logging into

Nexus and read, deploying, or managing repositories.

6.8 Managing Privileges

Nexus has three types of privileges: application privileges which cover actions a user can execute in Nexus, repository target privileges which govern the level of access a user has to a particular repository or repository target, and repository view privileges which control whether a user can view a repository. Behind the scenes, a privilege is related to a single REST operation and method like create, update, delete, read.

The screenshot shows the 'Privileges' management interface in Nexus. At the top, there's a toolbar with 'Refresh' and 'Add...' buttons. Below is a table listing existing privileges:

Name	User...	Type	Target	Repository	Method
New Repository Target Privilege		Repository Target			
3rd party - (view)	false	Repository View	3rd party		
Administrator privilege (ALL)	false	Application	*		
All M1 Repositories - (create)	false	Repository Target	All (Maven1)	All Repositories	create,read
All M1 Repositories - (delete)	false	Repository Target	All (Maven1)	All Repositories	delete,read

Below the table is a 'New Repository Target Privilege' form:

Name	Apache Snapshot
Description	
Repository	Apache Snapshots (Repo)
Repository Target	All (Maven2)

At the bottom of the form are 'Save' and 'Cancel' buttons.

Figure 6.29: Managing Security Privileges

To create a new privilege, click on the Add... button in the Privileges panel and choose Repository Target privilege. Creating a privilege will load the New Repository Target Privilege form shown in Figure 6.30. This form takes a privilege name, a privilege description, the repository to target, and a repository target.

The screenshot shows the 'Privileges' management screen in Nexus. At the top, there's a toolbar with 'Refresh' and 'Add...' buttons. Below the toolbar is a table with columns: Name, User..., Type, Target, Repository, and Method. The table lists several existing privileges:

Name	User...	Type	Target	Repository	Method
New Repository Target Privilege		Repository Target			
3rd party - (view)	false	Repository View		3rd party	
Administrator privilege (ALL)	false	Application		*	
All M1 Repositories - (create)	false	Repository Target	All (Maven1)	All Repositories	create,read
All M1 Repositories - (delete)	false	Repository Target	All (Maven1)	All Repositories	delete,read

Below the table, a new privilege is being created:

New Repository Target Privilege

Name	Apache Snapshot
Description	
Repository	Apache Snapshots (Repo)
Repository Target	All (Maven2)

At the bottom right of the form are 'Save' and 'Cancel' buttons.

Figure 6.30: Managing Security Privileges

Once you create a new privilege, it will create four underlying privileges: create, delete, read, and update. The four privileges created by the form in Figure 6.30 are shown in Figure 6.31.

The screenshot shows the 'Privileges' section of the Nexus management interface. At the top, there is a table with columns: Name, User..., Type, Target, Repository, and Method. The table lists four entries related to 'Apache Snapshot' operations:

Name	User...	Type	Target	Repository	Method
Apache Snapshot - (create)	true	Repository Target	All (Maven2)	Apache Snaps...	create,read
Apache Snapshot - (delete)	true	Repository Target	All (Maven2)	Apache Snaps...	delete,read
Apache Snapshot - (read)	true	Repository Target	All (Maven2)	Apache Snaps...	read
Apache Snapshot - (update)	true	Repository Target	All (Maven2)	Apache Snaps...	update,read

Below the table is a section titled 'API-Key Access' with the following fields:

Name	API-Key Access
Description	Give permission to use an API-Key to access the server
Type	Application
Method	*
Permission	apikey:access

Figure 6.31: Create, Delete, Read, and Update Privileges Created

6.9 Managing Repository Targets

A Repository Target is a set of regular expressions to match on the path of artifacts in a repository (in the same way as the routing rules work). Nexus is preconfigured with a number of repository targets and allows you to create additional ones. Access the management interface visible in Figure 6.32 via the Repository Targets menu item in the left hand Views.Repositories menu.

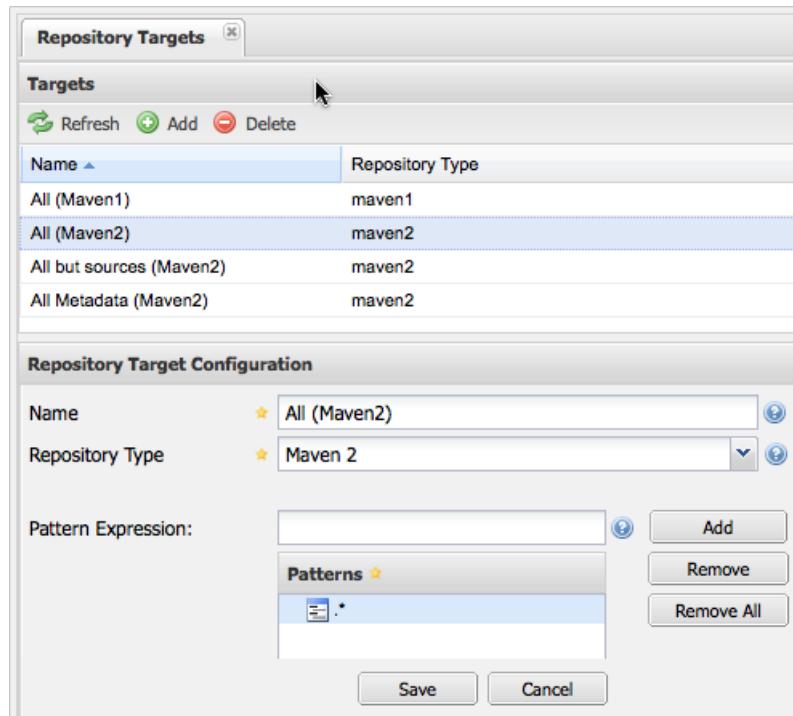


Figure 6.32: Managing Repository Targets

Repository targets allow you to define for example a target called Apache Maven with a pattern of `^/org/apache/maven/.*`. This would match all artifacts with a groupId of *org.apache.maven* and any artifacts within nested groupIds like *org.apache.maven.plugins*.

A pattern that would capture more artifacts like all artifacts with any part of the path containing *maven* could be `.*maven.*`.

The regular expressions can also be used to exclude artifacts as visible with the pattern `(?!.*-sources.*).*` in Figure 6.33 where artifacts with the qualifier *-sources* are excluded. The syntax used for the expressions is the [Java syntax](#), which is similar but not identical to the Perl syntax.

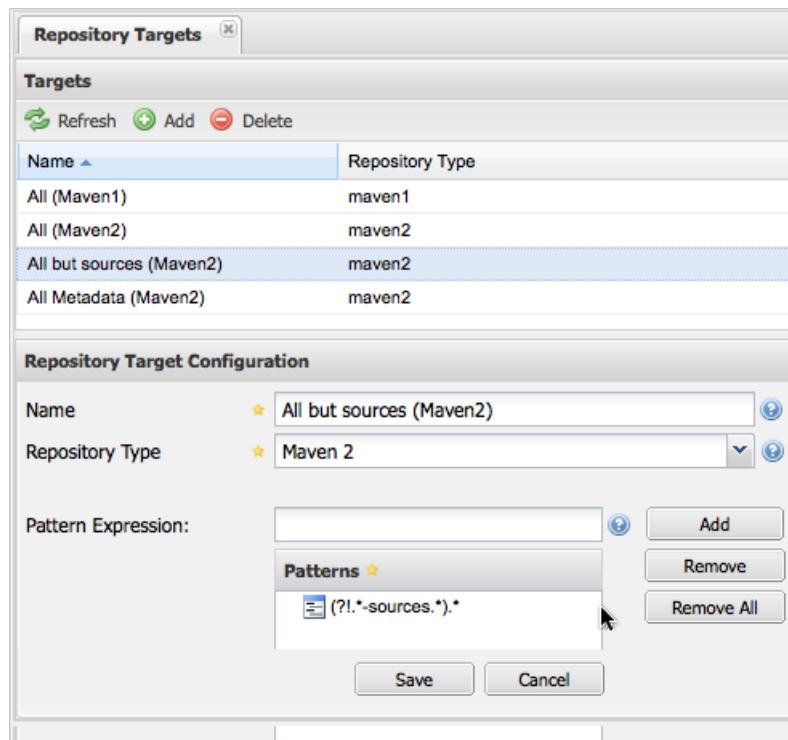


Figure 6.33: Excluding Source Artifacts from a Repository Targets

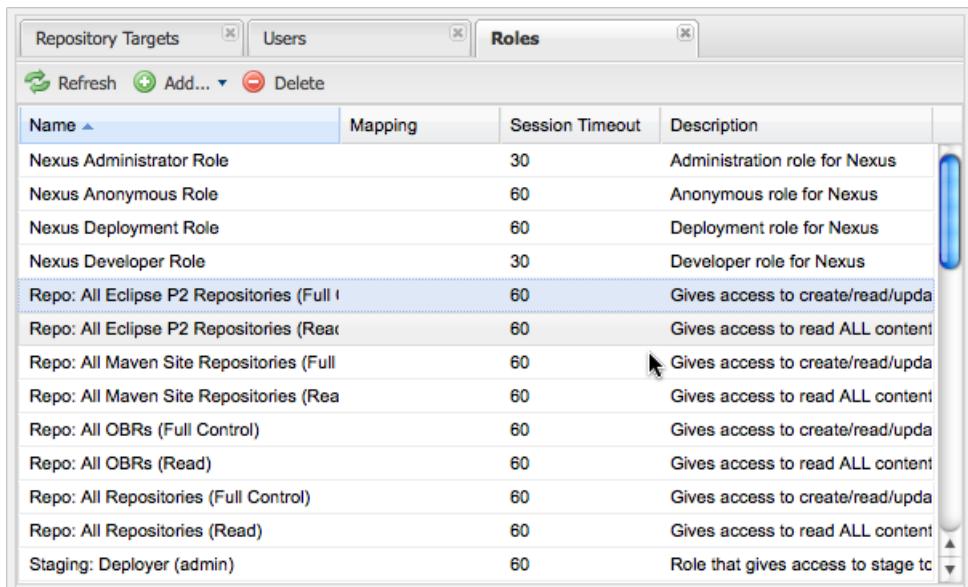
By combining multiple patterns in a repository target you can establish a fine grained control of artifacts included and excluded.

Once you have created a repository target you can add it as part of your security setup. You can add a new privilege that relates to the target and controls the CRUD operations for artifacts matching that path. The privilege can even span multiple repositories. With this setup you can delegate all control of artifacts in `org.apache.maven` to a "Maven" team. In this way, you don't need to create separate repositories for each logical division of your artifacts.

Repository targets are also used for matching artifacts for implicit capture in the Staging Suite as documented in Chapter 10.

6.10 Managing Roles

Nexus ships with four roles: Nexus Administrator Role, Nexus Anonymous Role, Nexus Developer Role, and Nexus Deployment Role. Click on the Roles link under Security in the Nexus menu to show the list of roles shown in Figure 6.34.



The screenshot shows a table titled 'Roles' with columns: Name, Mapping, Session Timeout, and Description. The table lists various roles and their properties. A cursor is hovering over the 'Description' column of the second-to-last row.

Name	Mapping	Session Timeout	Description
Nexus Administrator Role		30	Administration role for Nexus
Nexus Anonymous Role		60	Anonymous role for Nexus
Nexus Deployment Role		60	Deployment role for Nexus
Nexus Developer Role		30	Developer role for Nexus
Repo: All Eclipse P2 Repositories (Full)		60	Gives access to create/read/upda
Repo: All Eclipse P2 Repositories (Read)		60	Gives access to read ALL content
Repo: All Maven Site Repositories (Full)		60	Gives access to create/read/upda
Repo: All Maven Site Repositories (Read)		60	Gives access to read ALL content
Repo: All OBRs (Full Control)		60	Gives access to create/read/upda
Repo: All OBRs (Read)		60	Gives access to read ALL content
Repo: All Repositories (Full Control)		60	Gives access to create/read/upda
Repo: All Repositories (Read)		60	Gives access to read ALL content
Staging: Deployer (admin)		60	Role that gives access to stage to

Figure 6.34: Viewing the List of Defined Roles

To create a new role, click on the Add... button and fill out the New Nexus Role form shown in Figure 6.35.

When creating a new role, you will need to supply a role identifier, a role name, a description, and a session timeout. Roles are comprised of other roles and individual privileges, to assign a role or privilege to a role, click on the role or privilege under Available Roles/Privileges and drag the role or privilege to the Selected Roles/Privileges list.

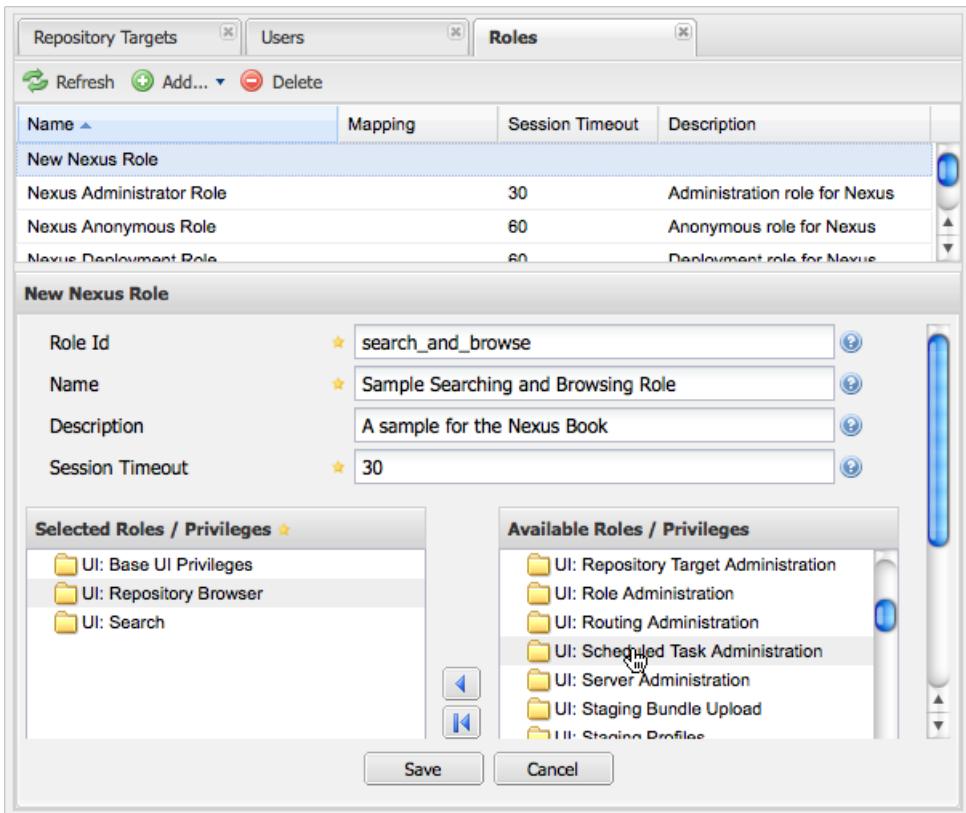


Figure 6.35: Creating a New Role

The built-in roles Nexus Administrator Role, Nexus Anonymous Role, Nexus Deployment Role, and Nexus Developer Role are managed by Nexus and can not be edited or deleted. Selecting one of these built-in roles will load the form shown in Figure 6.36.

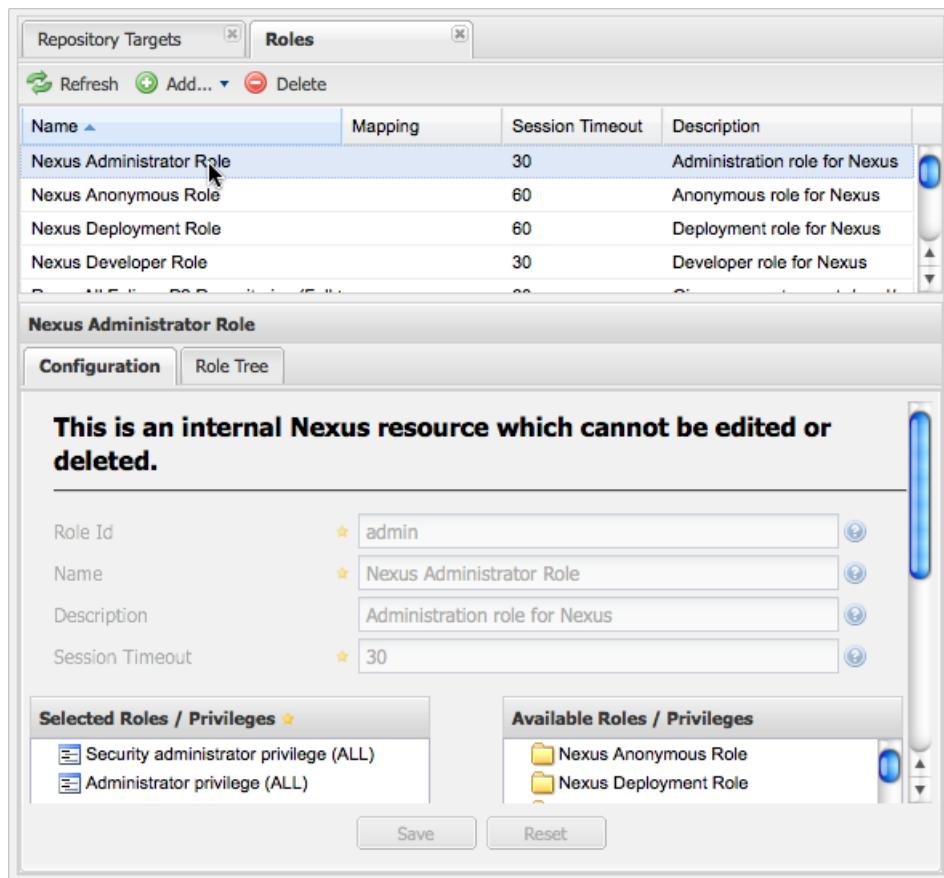


Figure 6.36: Viewing an Internal Role

A Nexus role is comprised of other Nexus roles and individual Nexus privileges. To view the component parts of a Nexus Role, select the role in the Roles panel and then choose the Role Tree tab as shown in Figure 6.37.

The screenshot shows the Nexus Roles management interface. At the top, there are tabs for 'Repository Targets' and 'Roles'. Below the tabs is a toolbar with 'Refresh', 'Add...', and 'Delete' buttons. A table lists four roles: 'Nexus Administrator Role', 'Nexus Anonymous Role', 'Nexus Deployment Role', and 'Nexus Developer Role'. The 'Nexus Anonymous Role' row is selected, highlighted with a blue background. The 'Description' column for this role states 'Anonymous role for Nexus'. Below the table, a section titled 'Nexus Anonymous Role' displays its configuration. It includes a 'Role Tree' tab and a 'Configuration' tab. The 'Role Tree' tab is active, showing a hierarchical tree of permissions. The tree starts with 'Nexus Anonymous Role' and branches into 'UI: Repository Browser', 'UI: Search', and 'UI: System Feeds'. Under 'UI: Repository Browser', permissions include 'Read Repository Status', 'Repositories - (read)', and 'Repository Groups - (read)'. Under 'UI: Search', permissions include 'Checksum Search' and 'Search Repositories'. Under 'UI: System Feeds', permissions include 'Artifact Download', 'Repository Content Classes Component - (read)', 'Repository Types - (read)', 'Status - (read)', 'User Forgot Password - (create,read)', and 'User Forgot User Id - (create,read)'.

Figure 6.37: Managing Security Roles

With the Repository Targets, you have fine grained control over every action in the system. For example you could make a target that includes everything except sources `(.?!-sources)\.` and assign that to one group while giving yet another group access to everything. This means you can host your public and private artifacts in a single repository without giving up control of your private artifacts.

6.11 Managing Users

Nexus ships with three users: admin, anonymous, and deployment. The admin user has all privileges, the anonymous user has read-only privileges, and the deployment user can both read and deploy to repositories. If you need to create users with a more focused set of permissions, you can click on Users under Security in the left-hand navigation menu. Once you see the list of users, you can click on a user to edit that specific user's user ID, name, email, or status. You can also assign or revoke specific roles or permissions for a particular user.

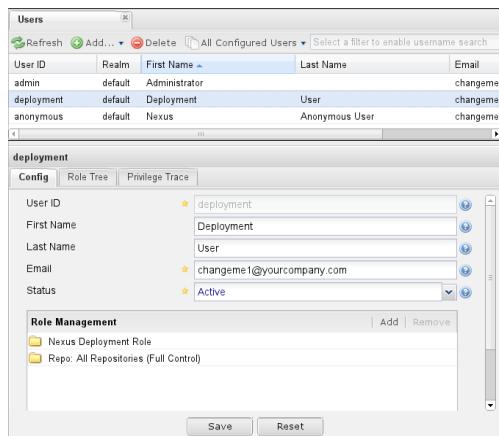


Figure 6.38: Managing Users

Clicking the Add button in the Role Management section will bring up the list of available roles in a pop up window visible in Figure 6.39. It allows you filter and search for roles and add one or multiple roles to the user.

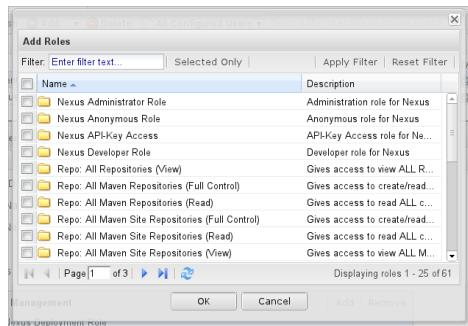


Figure 6.39: Adding Roles to a User

A user can be assigned one or more roles which in turn can include references to other Nexus roles or to individual Nexus privileges. To view a tree of assigned Nexus roles and privileges, select the Role Tree for a particular user as shown in Figure 6.40.

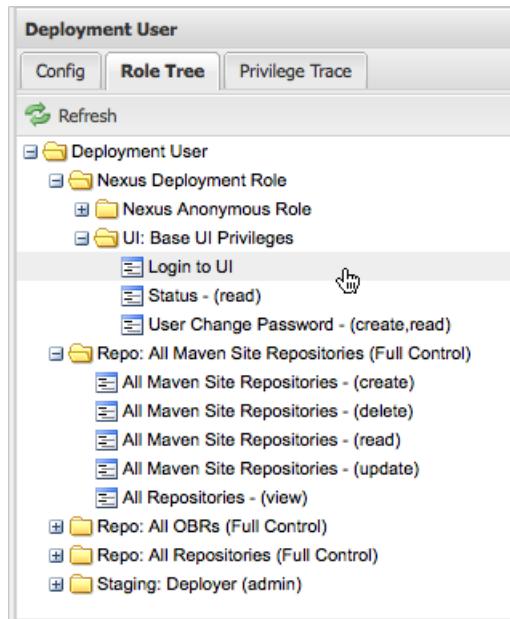


Figure 6.40: Nexus User Role Tree

If you need to find out exactly how a particular user has been granted a particular privilege, you can use the Privilege Trace panel as shown in Figure 6.41. The Privilege Trace pane lists all of the privileges that have been granted to a particular user. Clicking on a privilege loads a tree of roles that grant that particular privilege to a user. If a user has been assigned a specific privilege by more than one Role or Privilege assignment, you will be able to see this reflected in the Role Containment list.

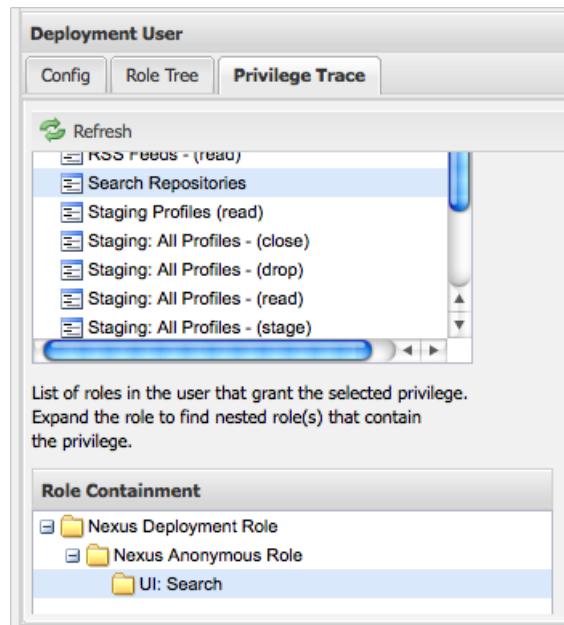


Figure 6.41: Nexus User Privilege Trace

Additional plugins can contribute further panels for the security configuration of a user. An example of an additional panel is the User Token panel, added by the User Token feature of Nexus Professional as documented in Section 6.15.

6.12 Network Configuration

By default, Nexus listens on port 8081. You can change this port, by changing the value in `$NEXUS_HOME/conf/nexus.properties`. This file is shown in [Contents of conf/nexus.properties](#). To change the port, stop Nexus, change the value of `applicationPort` in this file, and then restart Nexus. Once you do this, you should see a log statement in `$NEXUS_HOME/logs/wrapper.log` telling you that Nexus is listening on the altered port.

Contents of conf/nexus.properties

```
# Sonatype Nexus
# =====
# This is the most basic configuration of Nexus.

# Jetty section
application-port=8081
application-host=0.0.0.0
nexus-webapp=${bundleBasedir}/nexus
nexus-webapp-context-path=/nexus

# Nexus section
nexus-work=${bundleBasedir}../sonatype-work/nexus
runtime=${bundleBasedir}/nexus/WEB-INF
```

6.13 Nexus Logging Configuration

You can configure the level of logging from within the Nexus interface. To do this, click on Log Configuration under the Administration menu in the left-hand navigation menu. Clicking on this link will display the panel shown in Figure 6.42.

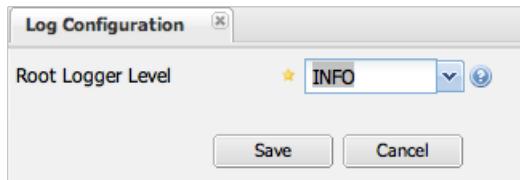


Figure 6.42: The Log Configuration Panel

From this panel you can configure the Root Logger Level. It controls how verbose the Nexus logging will be. If set to DEBUG, Nexus will be very verbose printing all log messages include debugging statements. If set to ERROR, Nexus will be far less verbose only printing out a log statement if Nexus encounters an error. INFO represents an intermediate amount of logging.

Advanced logging configurations can be done by editing the logback-nexus.xml file found in sonatype-work/nexus/conf. Other logback* file found in the same directory do not need to be edited for most logging modifications.

6.14 Nexus Plugins and REST Interfaces

As documented in Section 20.1 Nexus is built as a collection of plugins supported by a core architecture and additional plugins can be installed.

You can use the Nexus Plugin Console to list all installed Nexus plugins and browse REST services made available by the installed plugins. To open the Nexus Plugin Console, click on the *Plugin Console* link in the *Administration* menus shown in Figure 6.1.

Once you open the Plugin Console, you will see a list of plugins installed in your Nexus installation. Clicking on a plugin in this list will display information about the plugin including name, version, status, a description, SCM information about the plugin, and the URL of the plugin's project web site and links to the plugin documentation.

The screenshot shows the Nexus Plugin Console interface. At the top, there is a navigation bar with tabs for 'Welcome' and 'Plugin Console'. Below the navigation bar is a search bar containing the text 'Refresh'. The main content area displays a table of installed plugins. The table has three columns: 'Name', 'Version', and 'Description'. The 'Name' column lists the plugin names, the 'Version' column lists their versions, and the 'Description' column provides a brief description of each plugin's function. One row in the table is highlighted in grey, corresponding to the 'Nexus Restlet 1.x Plugin'. Below the table, there is a section titled 'Nexus Restlet 1.x Plugin' which contains detailed information about this specific plugin, including its name, version, status, description, SCM version, timestamp, site URL, and documentation link.

Name	Version	Description
Nexus PGP Plugin	2.4.0-09	Adds PGP Rules for Staging and Procurement
Nexus Plugin Console Plugin	2.4.0-09	Adds a UI to view installed plugins.
Nexus Plugin-kit Plugin	1.3.2	Provides support for Nexus plugins
Nexus Procurement Plugin	2.4.0-09	Allows you to block and allow specific artifacts
Nexus Remote Repository Browsing Plugin	2.4.0-09	Adds support for browsing proxy repositories
Nexus Restlet 1.x Plugin	2.4.0-09	Provides Restlet 1.x REST API.
Nexus SSL Plugin	1.0.4	SSL Certificates Validation
Nexus Siesta Plugin	2.4.0-09	Support for Siesta (Jersey) REST API.
Nexus Site Repository Plugin	1.1	Adds a repository type for deploying web sites
Nexus Smart Proxy Plugin	1.4.1	Provides Smart Proxy features

Nexus Restlet 1.x Plugin

Name	Nexus Restlet 1.x Plugin
Version	2.4.0-09
Status	Activated
Description	Provides Restlet 1.x REST API.
SCM Version	5e710cc30b91879a736c6d148394564e570d6c1b
SCM Timestamp	N/A
Site	http://links.sonatype.com/products/nexus/oss/home
Documentation	Restlet 1.x API

Figure 6.43: Plugin Console

An example for the plugin documentation is the main documentation for the core Nexus API linked off the Nexus Restlet 1.x Plugin from Figure 6.43 and displayed in Figure 6.44

The screenshot shows a web page titled "NEXUS RESTLET 1.x PLUGIN REST API". At the top right are links for "REST" and "Data Model". A yellow navigation bar at the top contains a "Home" link. The main content area has a section titled "REST Resources". Below it, a text block states: "This API supports a [Representational State Transfer \(REST\)](#) model for accessing a set of resources through a fixed set of operations. The following resources are accessible through the RESTful model:" followed by a bulleted list of URLs.

- [/all_repositories](#)
- [/artifact/maven](#)
- [/artifact/maven/content](#)
- [/artifact/maven/redirect](#)
- [/artifact/maven/resolve](#)
- [/assigned_privileges/{userId}](#)
- [/attributes](#)

Figure 6.44: Documentation Website for the Core API

6.15 Security Setup with User Tokens

6.15.1 Introduction

When using Apache Maven with Nexus, the user credentials for accessing Nexus have to be stored in clear text in the user's settings.xml file. Maven has the ability to encrypt passwords in setting.xml, but the need for it to be reversible in order to be used, limits its security. In addition the general setup and use is cumbersome and the potential need for regular changes due to strong security requirements e.g. with regular, required password changes triggers the need for a simpler and more secure solution.

The User Token feature of Nexus fills that need for Apache Maven as well as other build systems and users. It introduces a two part token for the user, replacing the username and password with a user code and a pass code that allows no way of recovering the username and password from the user code and pass code values, yet can be used for authentication with Nexus from the command line e.g. via Maven as well as in the UI.

This is especially useful for scenarios where single sign on solutions like LDAP are used for authentication against Nexus and other systems and the plain text username and password can not be stored in the settings.xml following security policies. In this scenario the generated user tokens can be used instead.

User token usage is integrated in the Maven settings template feature of Nexus documented in Chapter 13 to further simplify its use.

6.15.2 Enabling and Resetting User Tokens

The User Token based authentication can be activated by a Nexus administrator or user with the role usertoken-admin or usertoken-all by accessing the User Token item in the Security menu on the left hand navigation.

Once User Token is Enabled by activating the checkbox in the administration tab displayed in Figure 6.45 and pressing Save, the feature is activated and the additional section to Reset All User Tokens is available as well.

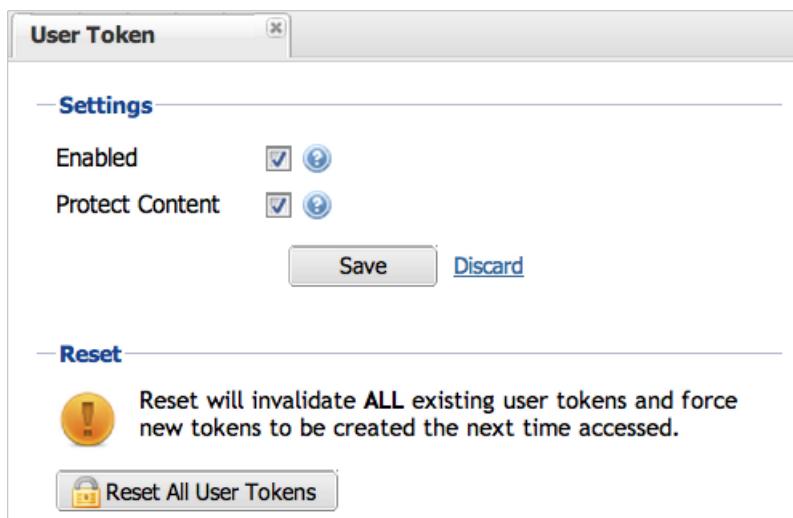


Figure 6.45: User Token Administration Tab Panel

Selecting the Protect Content feature configures Nexus to require a user token for any access to the content urls of Nexus, which includes all repositories and groups. This affects read access as well as write access for example for deployments from a build execution.

Activating User Token as a feature automatically adds the User Token Realm as a Selected Realm in the Security Settings section as displayed in Figure 6.46 and available in the Server section of the left hand Administration menu. If desired, you can reorder the security realms used, although the default settings with the User Token Realm as a first realm is probably the desired setup. This realm is not removed when the User Token feature is disabled, however it will cleanly pass through to the next realm and with the realm remaining any order changes stay persisted in case the feature is reactivated at a later stage.

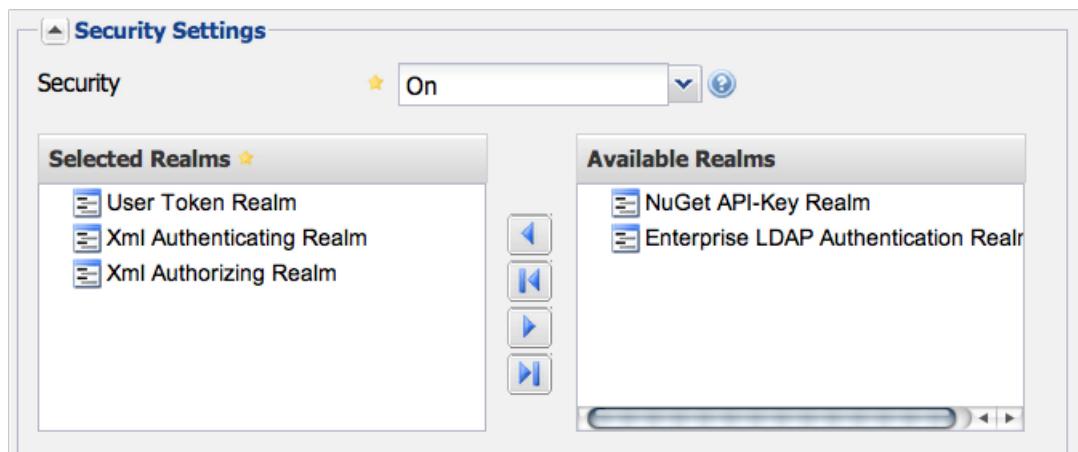


Figure 6.46: Selected Realms Server Security Settings with User Token Realm activated

Besides resetting all user tokens, an administrator can reset the token of an individual user by selecting the User Token tab in the Users administration from the Security menu in the left hand navigation displayed in Figure 6.47. The password requested for this action to proceed is the password for the currently logged in administrator resetting the token(s)

User ID	Realm	First Name	Last Name
admin	default	Administrator	
deployment	default	Deployment	User
anonymous	default	Nexus	Anonymous User

Figure 6.47: User Token Reset for Specific User in Security Users Administration



Warning

Resetting user tokens forces the users to update the `settings.xml` with the newly created tokens and potentially breaks any command line builds using the tokens until this change is carried out. This specifically also applies to continuous integration servers using user tokens or any other automated build executions.

6.15.3 Accessing and Using Your User Tokens

With User Token enabled, any user can access their individual tokens via their Profile panel. To access the panel, select Profile when clicking on the user name in the top right hand corner of the Nexus user interface. Then select User Token in the drop down to get access to the User Token screen in the Profile panel displayed in Figure 6.48.

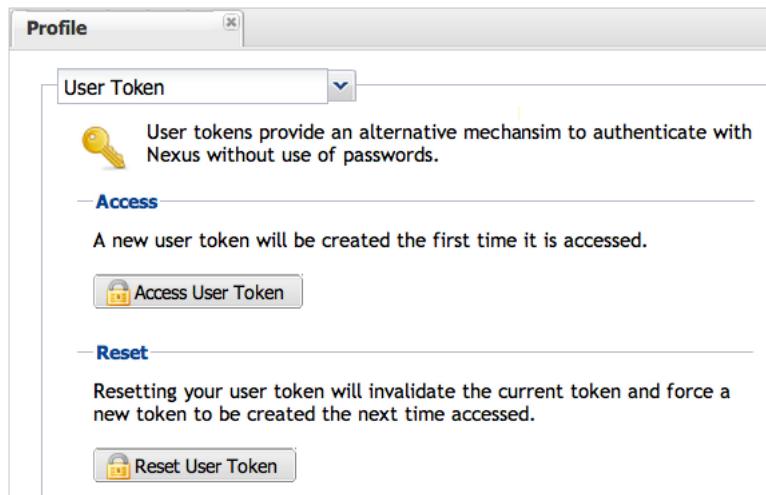


Figure 6.48: User Token Panel for the Logged in Users in the Profile Section

In order to be able to see this User Token panel the user has to have the usertoken-basic role or the usertoken-user privilege. To access or reset the token you have to press the respective button in the panel and then provide your username and password in the dialog.

Resetting the token will show and automatically hide a dialog with a success message and accessing the token will show the dialog displayed in Figure 6.49.

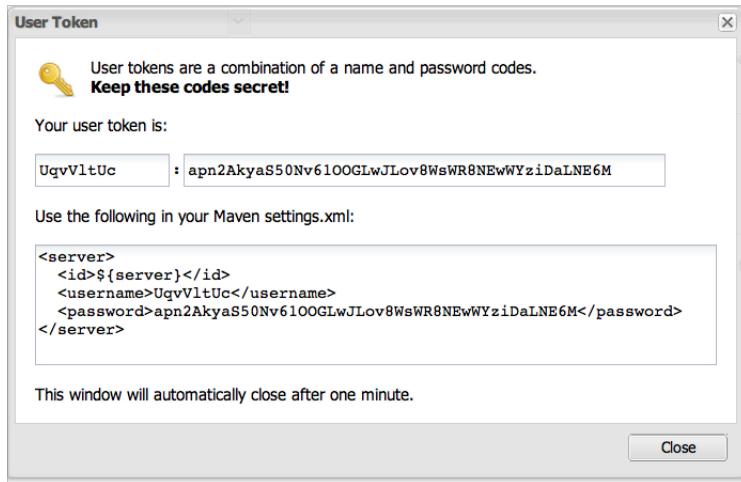


Figure 6.49: Accessing the User Token Information

The User Token dialog displays the user code and pass code tokens in separate fields in the top level section as well as a server section ready to be used in a Maven settings.xml file. When using the server section you simply have to replace the \${server} placeholder with the repository id that references your Nexus server you want to authenticate against with the user token. The dialog will close automatically after one minute or can be closed with the Close button.

The user code and pass code values can be used as replacements for username and password in the login dialog for Nexus. It is also possible to use the original username and the pass code to log in to Nexus.

With content protection enabled command line access to Nexus will require the tokens to be supplied. Access to e.g. the releases repository via

```
curl -v --user admin:admin http://localhost:9081/content/repositories/ →
    releases/
```

has to be replaced with the usage of user code and pass code separated by colon in the curl command line like this

```
curl -v --user HdeHuL4x:Y7ZH6ixZFdOVwNpRhaOV+phBISmipsfwVxPRUH1gkV09
http://localhost:9081/content/repositories/releases/
```

User token values can be accessed as part of the Maven settings template feature automating updates as documented in Chapter 13.

Note

The user tokens are created at first access whether that is by using the Nexus user interface or the Nexus Maven Plugin.

6.15.4 Configuring User Token Behaviour

The user token feature is preconfigured with built-in parameters and no external configuration file is created by default. It is however possible to customize some behaviour by creating a file *sonatype-work/nexus/conf/usertoken.properties*'.

The following properties can be configured:

usertoken.userTokenServiceImpl.allowLookupByUserName

This parameter controls if username lookup is allowed when using a pass code. The default is set to true. If set to false user code and pass code have to be used to authenticated, otherwise username and pass code is also possible. This would be the more secure setting.

usertoken.userTokenServiceImpl.restrictByUserAgent

With this value set to true, which is the default, any access to the Nexus content with content protection enabled will only be allowed to web browser based access even without credentials. Other tools like curl or wget or other command line tools will be blocked. With the more secure setting of *false* any access without correct codes will be disallowed.

The *usertoken.* prefix is optional when the properties are loaded from the *usertoken.properties* file.

Chapter 7

Nexus Smart Proxy

7.1 Introduction

Default is Polling

Typically an organization runs a single Nexus instance to proxy external components as well as host internally produced components. When a build is running against this Nexus instance it will look for any new components in the proxied remote repositories. This adds additional network traffic that in many cases will just be a response from the remote server indicating that there are no changes.

This polling approach is fine for smaller deployment. It will however not result in immediately updated components as soon as they become available upstream. In distributed teams with multiple Nexus instances, this delay can result in build failures and delays. The only way you are going to achieve assurance that everything is up to date is by setting your expiration times to zero and constantly polling.

Smart Proxy Introduces Publish-Subscribe

Increasingly Nexus is used in globally distributed teams or used by projects that span multiple organizations. In many cases it is advisable for each physical location to host their own Nexus instance. This local instance hosts their own components and proxies the other servers.

An example deployment scenario is displayed in Figure 7.7. Using the traditional polling approach,

specifically when used with snapshot repositories, can result in significant traffic and a performance hit for all involved servers.

The Smart Proxy feature replaces this constant polling approach with a Publish-Subscribe based messaging approach between Nexus instances sharing a mutual trust. The result is a significantly improved performance due to nearly immediate availability of upstream artifact information directly in the downstream Nexus instances.

7.2 Enabling Smart Proxy Publishing

In order to enable the Smart Proxy feature on your Nexus instance, you need to navigate to the global Smart Proxy configuration screen. It is available in the left hand navigation in the Enterprise section. Selecting Smart Proxy will show you the configuration screen displayed in Figure 7.1.

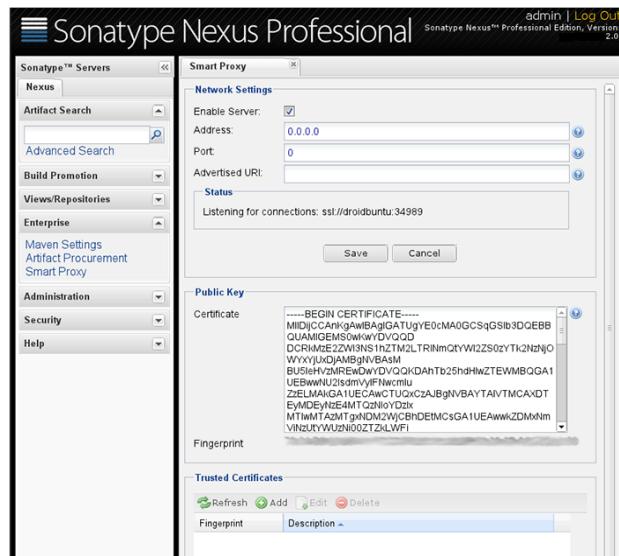


Figure 7.1: Global Configuration for Smart Proxy

The Network Settings section allows you to enable the Smart Proxy server with a check-box. This will need to be enabled on all servers that will publish events in the Smart Proxy network, while servers that will only act as subscribers can leave this option disabled.

In addition you can configure the address and port at which the publishing server will be available. The default address of 0.0.0.0 will cause the proxy to listen on all addresses. The default port number of 0 will trigger usage of a random available port number for connection listening. If a random port is used, it will be chosen when the server (re)starts.

With the Advertised URI field it is possible to configure a specific address to be broad-casted by the proxy to the subscribing Smart Proxy clients enabling e.g. usage of a publicly available fully qualified host-name including the domain or also just usage of an externally reachable IP number.

Tip

It is important to enable the configured connection ports on any firewall between the servers to allow the direct socket connection between the servers and to avoid using random ports.

The Status field below the form will show the current status of the Smart Proxy including the full address and port.

The Public Key field displays the key identifying this server instance. It is automatically populated with the certificate associated with the public/private key pair that is generated when the server is first run.

Tip

The key is stored in sonatype-work/nexus/conf/keystore/private.ks and identifies this server. If you copy the sonatype work folder from one server to another as part of a migration or a move from testing to staging or production you will need to ensure that keys are not identical between multiple servers. To get a new key generated simply remove the key-store file and restart Nexus.

7.3 Establishing Trust

The servers publishing as well as subscribing to events identify themselves with their Public Key. This key has to be registered with the other servers in the Trusted Certificates section of the Smart Proxy configuration screen.

To configure two Nexus repository servers as trusted Smart Proxies, you copy the Public Key from the certificate of the other server in the Trusted Certificates configuration section by adding a new trusted certificate with a meaningful description as displayed in Figure 7.2 and Figure 7.3.

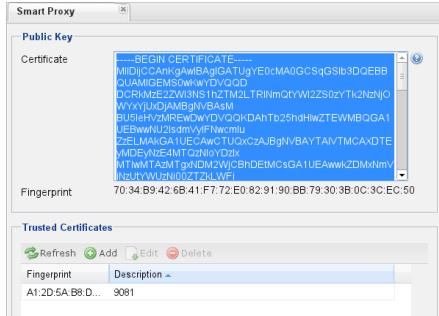


Figure 7.2: Copying a Certificate

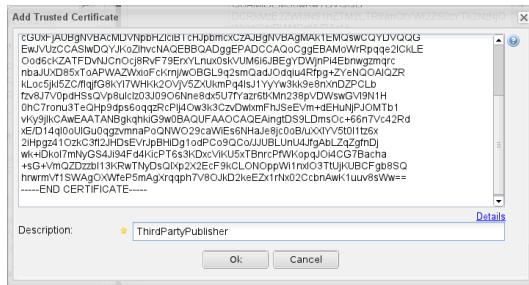


Figure 7.3: Adding a Trusted Certificate

All the key generation and certificates related to the trust management is handled by Nexus itself and no external configuration or usage of external keys is necessary or possible.

7.4 Repository Specific Smart Proxy Configuration

Once Smart Proxy has been configured and enabled as described above, you have to configure, which repositories' content should be synchronized between the servers. This is done in the Repositories administration interface in a separate configuration tab titled Smart Proxy, which allows you to configure repository specific details as compared to server wide details described above.

On the publishing Nexus server you have to enable Smart Proxy on the desired hosted, virtual or proxy repositories in the repository configuration. This is accomplished by selecting the Publish Updates check-

box in the Publish section of the Smart Proxy configuration for a specific repository as displayed in Figure 7.4 and pressing save.

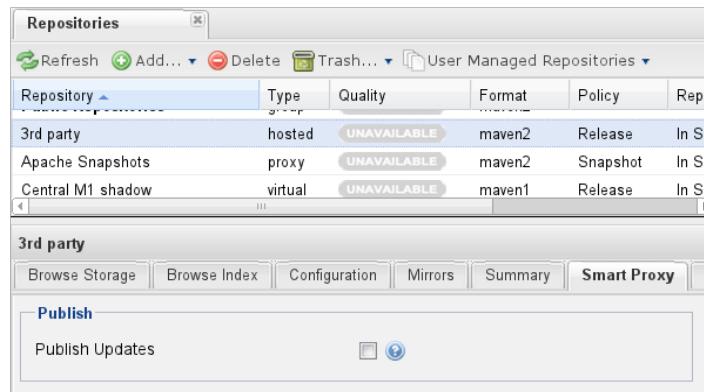


Figure 7.4: Smart Proxy Settings for a Hosted Repository

On the Nexus instance subscribing to the publishing server you have to create a new proxy repository to hold the synchronized artifacts. The Smart Proxy configuration for this repository displayed in Figure 7.5 allows you to activate the Receive Updates check-box in the Subscribe configuration section. You can enable pre-fetching the components by selecting *Immediately* in the *Download Updated Artifacts* dropdown. The default behaviour is to only download components *Upon Request*.

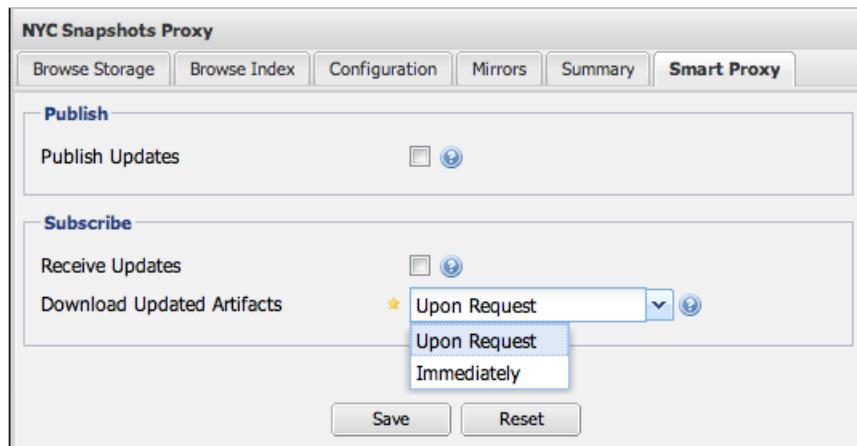


Figure 7.5: Smart Proxy Settings for a Proxy Repository

With a working trust established between the publishing and subscribing Nexus servers the Smart Proxy configuration of the proxy repository on the subscribing Nexus will display connection status as displayed in Figure 7.6.

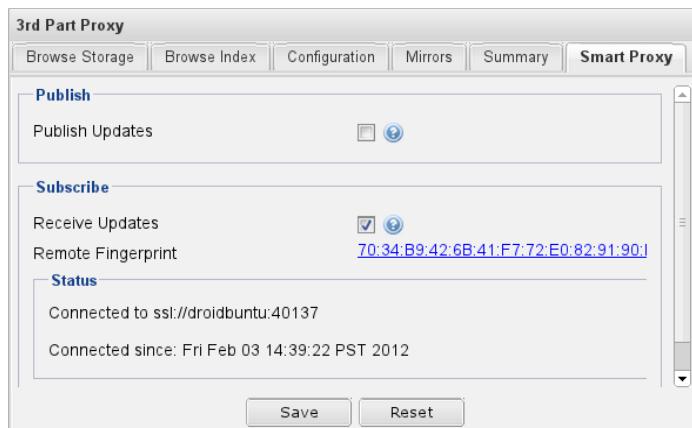


Figure 7.6: Subscription with Smart Proxy Connected

7.5 Smart Proxy Security and Messages

Smart Proxy messages are started with an initial handshake via HTTP. This handshake allows the two server to exchange their keys and confirm that they are configured with a valid trust relationship to communicate. After successful handshake, messages are sent in the middleware layer and can be configured to be sent via SSL encrypted messages.

The following events are broad-casted via Smart Proxy.

- a new artifact has been deployed
- an artifact has been deleted
- an artifact has been changed
- repository cache or a part of it has been cleared
- Smart Proxy publishing has been disabled

On the recipient side this will cause the changes to be applied mimicking what happened on the publisher. Only deletion will not be executed on the client, but ignored. If Smart Proxy is disabled the subscription will be stopped.

7.6 Example Setup

The deployment scenario displayed in Figure 7.7 is a typical use case for Smart Proxy. Component development is spread out across four distributed teams located in New York, London, Bangalore and San Jose. Each of the teams has a Nexus instance deployed in their local network to provide the best performance for each developer team and any locally running continuous integration server and other integrations

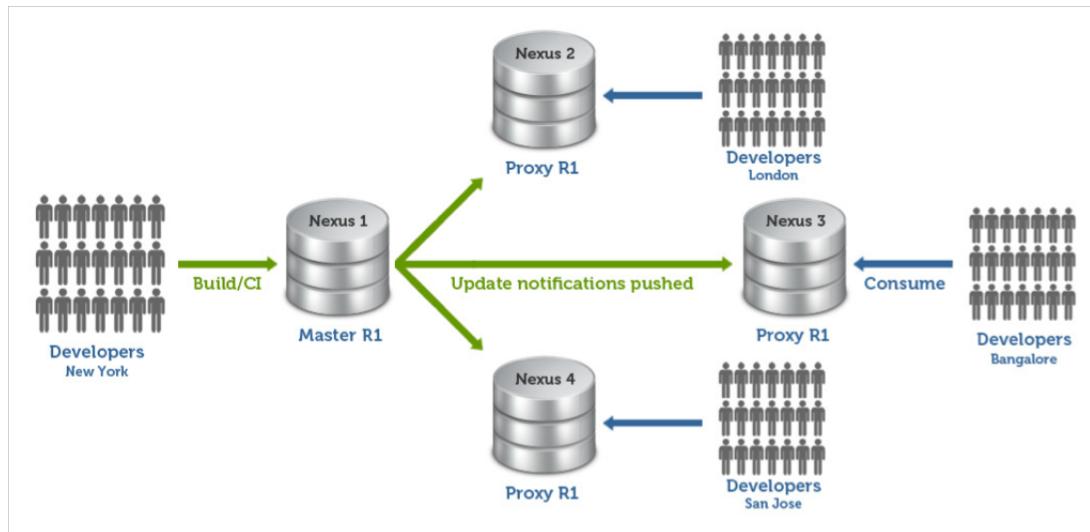


Figure 7.7: Deployment Scenario for a Smart Proxy Use Case

When the development team in New York does a commit to their component build a continuous integration server deploys a new component snapshot version to the Nexus 1 instance.

With Smart Proxy enabled, this deployment is immediately followed by notifications, sent to the trusted smart proxy subscribers in Nexus 2, Nexus 3 and Nexus 4. These are collocated with the developers in London, Bangalore and San Jose and can be configured to immediately fetch the new components

available. At a minimum they will know about the availability of new component versions, without the need to poll Nexus 1 repeatedly, therefore keeping performance high for everyone.

When a user of Nexus 2, 3 or 4 build a component that depends on a snapshot version of the component from Nexus 1, smart proxy guarantees that the latest version published to Nexus 1 is used.

To configure Smart Proxy between these servers for the snapshots repository you have to

1. add the public key of Nexus 1 as trusted certificate to Nexus 2, 3 and 4
2. add the public keys of Nexus 2, 3 and 4 as trusted certificate to Server 2
3. enable smart proxy publishing on the snapshot repository on Nexus 1
4. set up new proxy repositories to proxy the Nexus 1 snapshot repository on Nexus 2, 3 and 4
5. enable smart proxy subscription on the new proxy repositories
6. optionally enable pre-fetching of components
7. add the new proxy repositories to the public group on Nexus 2, 3 and 4

With this setup any snapshot deployment from the New York team on Nexus 1 is immediately available to the development team in London, Bangalor and San Jose.

Chapter 8

Nexus LDAP Integration

8.1 Introduction

Nexus Open Source has a Lightweight Directory Access Protocol (LDAP) Authentication realm which provides Nexus with the capability to authenticate users against an LDAP server. In addition to handling authentication, Nexus can be configured to map Nexus roles to LDAP user groups. If a user is a member of a group that matches the ID of a Nexus role, Nexus will grant that user the matching Nexus role. In addition to this highly configurable user and group mapping capability, Nexus can augment LDAP group membership with Nexus-specific user-role mapping.

In addition to the basic LDAP support from Nexus Open Source, Nexus Professional offers LDAP support features for enterprise LDAP deployments. These include the ability to cache authentication information, support for multiple LDAP servers and backup mirrors, the ability to test user logins, support for common user/group mapping templates, and the ability to support more than one schema across multiple servers.

8.2 Enabling the LDAP Authentication Realm

In order to use LDAP authentication in Nexus, you will need to add the Nexus LDAP Authentication Realm to the Selected Realms in the Security section of the Server configuration panel. To load the Server configuration panel, click on the Server link under Administration in the Nexus menu. Once you

have the Server configuration panel loaded, select Enterprise LDAP Authentication Realm (or OSS LDAP Authentication Realm) in the Available Realms list under the Security section and click the Add button (or Left Arrow) as shown in Figure 8.1.

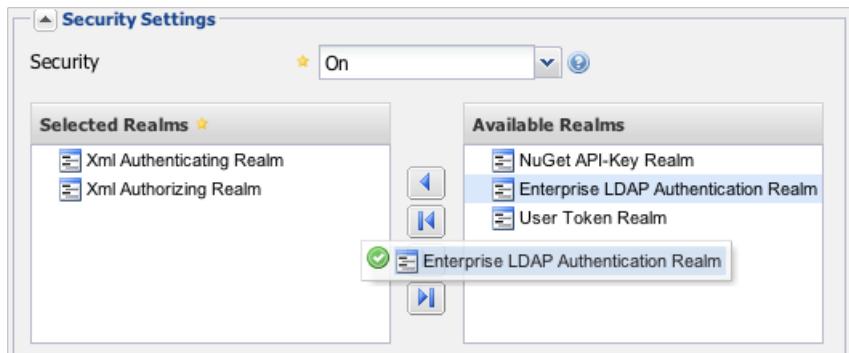


Figure 8.1: Adding the LDAP Authentication Realm to Available Realms

Next, ensure that the LDAP realms is located below the XML realms in the list. If the LDAP realm is on top of the list after adding, drag it after the XML realms as shown in Figure 8.2.

This is necessary, so that Nexus can be used by anonymous, admin and other users configured in the XML realms even with LDAP authentication offline or unavailable. Any user account not found in the XML realms, will be passed through to LDAP authentication.

Next, click on the Save button at the bottom of the Server configuration panel to have the change applied.

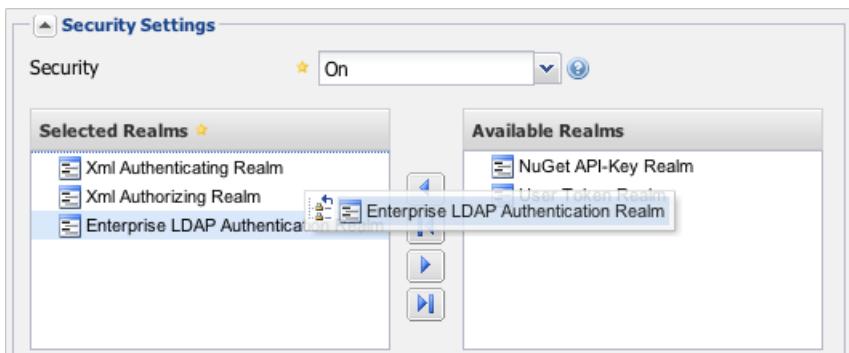


Figure 8.2: Move the LDAP Authentication Realm after the XML Realms

8.3 Configuring Nexus LDAP Integration

To configure LDAP integration, click on the Enterprise LDAP menu item in Nexus Professional or the LDAP Configuration menu item in Nexus Open Source in the Security menu as shown in Figure 8.3.



Figure 8.3: Enterprise LDAP Option in the Security Menu

Clicking on the Enterprise LDAP/LDAP Configuration menu item will load the LDAP Configuration panel. The following sections outline the configuration options available in the LDAP Configuration Panel.

8.4 Connection and Authentication

Figure 8.4 shows a simplified LDAP configuration for Nexus configured to connect to an LDAP server running on localhost port 10389 using the search base of "ou=system". On a more standard installation, you would likely not want to use Simple Authentication as it sends the password in clear text over the network, and you would also use a search base which corresponds to your organization's top-level domain components such as "dc=sonatype,dc=com".

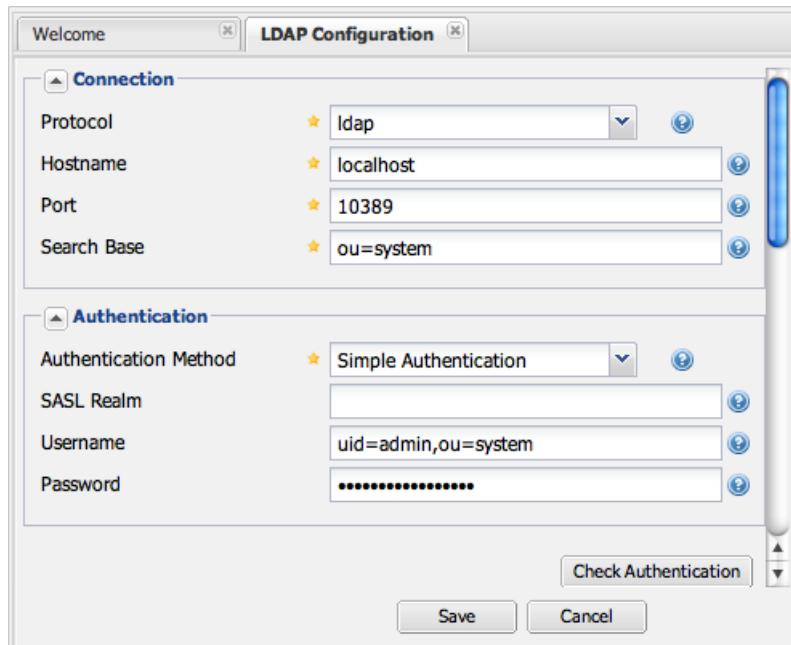


Figure 8.4: A Simple LDAP Connection and Authentication Setup

The following parameters can be configured in the Connection and Authentication sections of the LDAP Configuration panel.

Protocol

Valid values in this drop-down are ldap and ldaps which correspond to the Lightweight Directory Access Protocol and the Lightweight Directory Access Protocol over SSL.

Hostname

The hostname or IP address of the LDAP

Port

The port on which the LDAP server is listening. Port 389 is the default port for the ldap protocol, and port 636 is the default port for the ldaps.

Search Base

The search base is the Distinguished Name (DN) to be appended to the LDAP query. The search base usually corresponds to the domain name of an organization. For example, the search base on the Sonatype LDAP server could be "dc=sonatype,dc=com".

Authentication Method

Nexus provides four distinct authentication methods to be used when connecting to the LDAP Server:

Simple Authentication

Simple authentication is not recommended for production deployments not using the secure ldaps protocol as it sends a clear-text password over the network.

Anonymous Authentication

Used when Nexus only needs read-only access to non-protected entries and attributes when binding to the LDAP

Digest-MD5

This is an improvement on the CRAM-MD5 authentication method. For more information, see <http://www.ietf.org/rfc/rfc2831.txt>

CRAM-MD5

The Challenge-Response Authentication Method (CRAM) 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 which the server compares to an expected value. For more information, see RFC 2195

For a full discussion of LDAP authentication approaches, see <http://www.ietf.org/rfc/rfc2829.txt> and <http://www.ietf.org/rfc/rfc2251.txt>

SASL Realm

The Simple Authentication and Security Layer (SASL) Realm to connect with. The SASL Realm is only available if the authentication method is Digest-MD5 or CRAM-MD5.

Username

Username of an LDAP User to connect (or bind) with. This is a Distinguished Name of a user who has read access to all users and groups

Password

Password for an Administrative LDAP User

8.5 User and Group Mapping

The LDAP Configuration panel in Nexus Open Source contains sections to manage User Element Mapping and Group Element Mapping in the User and Group Settings tab. These configuration sections are located in a separate panel called User and Group Settings in Nexus Professional. This panel provided a User & Group Templates drop down displayed in Figure 8.5 that will adjust the rest of the user interface based on your template selection.

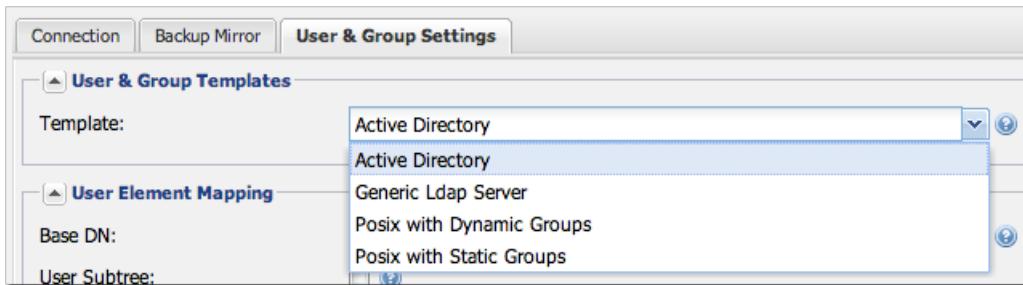


Figure 8.5: User & Group Templates Selection Drop Down

The User Element Mapping displayed in Figure 8.6 has been prepopulated by the Active Directory selection in the template drop down and needs to be configured as required by your LDAP server. The available fields are:

Base DN

Corresponds to the Base DN containing user entries. This DN is going to be relative to the Search Base which was specified in Figure 8.4. For example, if your users are all contained in "ou=users,dc=sonatype,dc=com" and you specified a Search Base of "dc=sonatype,dc=com" you would use a value of "ou=users"

User Subtree

True if there is a tree below the Base DN which can contain user entries. False if all users are contained within the specified Base DN. For example, if all users are in "ou=users,dc=sonatype,dc=com" this field should be false. If users can appear in organizational units within organizational units such as "ou=development,ou=users,dc=sonatype,dc=com" this field should be true.

Object Class

This value defaults to `inetOrgPerson` which is a standard object class defined in [RFC 2798](#). `inetOrgPerson` contains standard fields such as `mail`, `uid`. Other possible values are `posixAccount` or a custom class.

User ID Attribute

This is the attribute of the Object class which supplies the User ID. Nexus will use this attribute as the Nexus User ID.

Real Name Attribute

This is the attribute of the Object class which supplies the real name of the user. Nexus will use this attribute when it needs to display the real name of a user.

E-Mail Attribute

This is the attribute of the Object class which supplies the email address of the user. Nexus will use this attribute when it needs to send an email to a user.

Password Attribute

This control is only available in Nexus Open Source and replaced by the Use Password Attribute section from [?informalfigure] in Nexus Professional. It can be used to configure the Object class, which supplies the password ("userPassword").

The screenshot shows the 'User Element Mapping' configuration page. It includes fields for Base DN (cn=users), User Subtree (checkbox), Object Class (user), User Filter, User ID Attribute (sAMAccountName), Real Name Attribute (cn), and E-Mail Attribute (mail). Each field has a blue question mark icon to its right.

Figure 8.6: User Element Mapping

Once the checkbox for Use Password Attribute has been selected, the interface from [?informalfigure] allows you to configure the optional attribute. When not configured authentication will occur as a bind to the LDAP server. Otherwise this is the attribute of the Object class which supplies the password of the user. Nexus will use this attribute when it is authenticating a user against an LDAP server.

The screenshot shows the 'Use Password Attribute' configuration page. It features a single field for 'Password Attribute' with a yellow star icon to its left, and a blue question mark icon to its right.

The Group Type drop down displayed in Figure 8.7 and Figure 8.8 determines, which fields are available in the user interface. Groups are generally one of two types in LDAP systems - static or dynamic. A static group contains a list of users. A dynamic group is where the user contains a list of groups the user belongs to. In LDAP a static group would be captured in an entry with an Object class groupOfUniqueNames which contains one or more uniqueMember attributes. In a dynamic group configuration, each user entry in LDAP contains an attribute which lists group membership.

The screenshot shows a configuration dialog titled 'Group Element Mapping'. Under 'Group Type', the dropdown is set to 'Dynamic Groups'. Under 'Member of Attribute', the input field contains 'memberOf'. There are two blue question mark icons next to the dropdown and input field.

Figure 8.7: Dynamic Group Element Mapping

Dynamic groups are configured via the Member of Attribute parameter. Nexus will inspect this attribute of the user entry to get a list of groups that the user is a member of. In this configuration, a user entry would have an attribute such as memberOf which would contain the name of a group.

The screenshot shows a configuration dialog titled 'Group Element Mapping'. Under 'Group Type', the dropdown is set to 'Static Groups'. Other fields include 'Base DN' (empty), 'Group Subtree' (unchecked), 'Object Class' (empty), 'Group ID Attribute' (empty), 'Group Member Attribute' (empty), and 'Group Member Format' (empty). At the bottom are four buttons: 'Check User Mapping', 'Check Login', 'Save', and 'Cancel'.

Figure 8.8: Static Group Element Mapping

Static groups are configured with the following parameters:

Base DN

This field is similar to the Base DN field described for User Element Mapping. If your groups were defined under "ou=groups,dc=sonatype,dc=com", this field would have a value of "ou=groups"

Group Subtree

This field is similar to the User Subtree field described for User Element Mapping. If all groups are defined under the entry defined in Base DN, this field should be false, if a group can be defined in a tree of organizational units under the Base DN, this field should be true.

Object Class

This value defaults to groupOfUniqueNames which is a standard object class defined in [RFC 4519](#). groupOfUniqueNames is simply a collection of references to unique entries in an LDAP directory and can be used to associate user entries with a group. Other possible values are posixGroup or a custom class.

Group ID Attribute

Specifies the attribute of the Object class which specifies the Group ID. If the value of this field corresponds to the ID of a Nexus Role, members of this group will have the corresponding Nexus privileges. Defaults to "cn".

Group Member Attribute

Specifies the attribute of the Object class which specifies a member of a group. A groupOfUniqueNames has multiple uniqueMember attributes for each member of a group. Defaults to "uniqueMember".

Group Member Format

This field captures the format of the Group Member Attribute and it is used by Nexus to extract a username from this attribute. For example, if the Group Member Attribute has the format "uid=brian,ou=users,dc=sonatype,dc=com", then the Group Member Format would be "uid=\$username,ou=users,dc=sonatype,dc=com". If the Group Member Attribute had the format "brian", then the Group Member Format would be "\$username".

If your installation does not use Static Groups, you can configure Nexus LDAP Integration to refer to an attribute on the User entry to derive group membership. To do this, select Dynamic Groups in the Group Type field in Group Element Mapping.

Once you have configured the User & Group Settings you can check the correctness of your user mapping by pressing the Check user Mapping button visible in Figure 8.8.

Nexus Professional offers a button "Check Login" to check an individual users login and can be used as documented in Section [8.11.5](#).

Press the Save button after successful configuration.

8.6 Mapping Users and Groups with Active Directory

When mapping users and groups to an Active Directory installation, try the common configuration values listed in Table [8.2](#) and Table [8.3](#).

Table 8.1: Connection and Authentication Configuration for Active Directory

Configuration Element	Configuration Value
Protocol	ldap
Hostname	Hostname of Active Directory Server
Port	389 (or port of AD server)
Search Base	DC=yourcompany,DC=com (customize for your organization)
Authentication	Simple Authentication
Username	CN=Administrator,CN=Users,DC=yourcompany,DC=com

Table 8.2: User Element Mapping Configuration for Active Directory

Configuration Element	Configuration Value
Base DN	cn=users
User Subtree	false
Object Class	user
User ID Attribute	sAMAccountName
Real Name Attribute	cn
E-Mail Attribute	mail
Password Attribute	(Not Used)

Table 8.3: Group Element Mapping Configuration for Active Directory

Configuration Element	Configuration Value
Group Type	Dynamic Groups
Member Of Attribute	memberOf



Warning

You should connect to the AD through port 3268 if you have a multi-domain, distributed Active Directory forest. Connecting directly to port 389 might lead to errors.

8.7 Mapping Users and Groups with posixAccount

When mapping users and groups to LDAP entries of type posixAccount, try the common configuration values listed in Table 8.4 and Table 8.5.

Table 8.4: User Element Mapping Configuration for posixAccount

Configuration Element	Configuration Value
Base DN	(Not Standard)
User Subtree	false
Object Class	posixAccount
User ID Attribute	sAMAccountName
Real Name Attribute	uid
E-Mail Attribute	mail
Password Attribute	(Not Used)

Table 8.5: Group Element Mapping Configuration for posixGroup

Configuration Element	Configuration Value
Group Type	Static Groups
Base DN	(Not Standard)
Group Subtree	false
Object Class	posixGroup
Group ID Attribute	cn
Group Member Attribute	memberUid
Group Member Format	

8.8 Mapping Roles to LDAP Users

Once User and Group Mapping has been configured, you can start verifying how LDAP users and groups are mapped to Nexus Roles. If a user is a member of an LDAP group that has a Group ID corresponding to the ID of a Nexus Role, that user is granted the appropriate permissions in Nexus. For example, if the LDAP user entry in "uid=brian,ou=users,dc=sonatype,dc=com" is a member of a groupOfUniqueNames attribute value of admin, when this user logs into Nexus, it will be granted the Nexus Administrator Role if Group Element Mapping is configured properly. To verify the User Element Mapping and Group Element Mapping, click on Check User Mapping in the LDAP Configuration panel directly below the Group Element Mapping section, Figure 8.9 shows the results of this check.

The screenshot shows the 'LDAP Configuration' panel with the following sections:

- User Element Mapping:** Contains fields for Base DN (ou=users), User Subtree (checkbox), Object Class (inetOrgPerson), User ID Attribute (uid), Real Name Attribute (cn), E-Mail Attribute (mail), Password Attribute (userpassword), and Password Encoding (Crypt).
- Group Element Mapping:** Contains fields for Group Type (Static Groups) and Base DN (ou=groups).
- User Mapping Test Results:** A table with columns User ID, Name, Email, and Roles. It shows one row: brian, brian, brian@example.com, admin.

Figure 8.9: Checking the User and Group Mapping in LDAP Configuration

In Figure 8.9, Nexus LDAP Integration locates a user with a User ID of "brian" who is a member of the "admin" group. When brian logs in, he will have all of the rights that the admin Nexus Role has.

8.9 Mapping Nexus Roles for External Users

If you are unable to map all of the Nexus roles to LDAP groups, you can always augment the Role information by adding a specific user-role mapping for an external LDAP user in Nexus. In other words, if you need to make sure that a specific user in LDAP gets a specific Nexus role and you don't want to model this as a group membership, you can add a role mapping for an external user in Nexus. Nexus will keep track of this association independent of your LDAP server. Nexus continues to delegate authentication to the LDAP server for this user, Nexus will continue to map the user to Nexus roles based on the group element mapping you have configured, but Nexus will also add any roles specified in the User panel. You are augmenting the role information that Nexus gathers from the group element mapping.

Once the User and Group Mapping has been configured, click on the Users link under Security in the Nexus menu. The Users tab is going to contain all of the "configured" users for this Nexus instance as shown in Figure 8.10. A configured user is a user in a Nexus-managed Realm or an External User which has an explicit mapping to a Nexus role. In Figure 8.10, you can see the three default users in the Nexus-managed default realm plus the brian user from LDAP. The brian user appears because this user has been mapped to a Nexus role.

The screenshot shows the Nexus 'Users' management interface. At the top, there's a header bar with tabs for 'Welcome', 'Users', and 'LDAP Configuration'. Below the header are standard UI controls: Refresh, Add..., Delete, and a dropdown for 'All Configured Users' which is currently set to 'Select a filter to enable username search'. A search input field is also present.

User ID	Realm	Name	Email	Roles
admin	default	Administrator	changeme@yourcompany.com	Nexus Administrator Role
brian	LDAP	Brian Fox		Nexus Deployment Role
deployment	default	Deployment User	changeme1@yourcompany.com	Repo: All Repositories (Full Control), Nexus Deployment Role
anonymous	default	Nexus Anonymous User	changeme2@yourcompany.com	Nexus Anonymous Role

Below the table, a section titled 'Deployment User' is expanded, showing a form to edit the 'deployment' user:

- User ID:** deployment
- Name:** Deployment User
- Email:** changeme1@yourcompany.com
- Status:** Active

Two panels below the form show role assignments:

- Selected Roles:** Nexus Deployment Role
- Available Roles:** Nexus Administrator Role

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

Figure 8.10: Viewing All Configured Users

The list of users in Figure 8.10 is a combination of all of the users in the Nexus default realm and all of the External Users with role mappings. To explore these two sets of users, click on the All Configured Users drop-down and choose "Default Realm Users". Once you select this, click in the search field and press Enter. Searching with a blank string in the Users panel will return all of the users of the selected type. In Figure 8.11 you see a dialog containing all three default users from the Nexus default realm.

This screenshot shows the same Nexus 'Users' interface as Figure 8.10, but with a different configuration. The 'All Configured Users' dropdown is now set to 'Default Realm Users'. The search input field at the top contains a blank string, which is highlighted with a cursor.

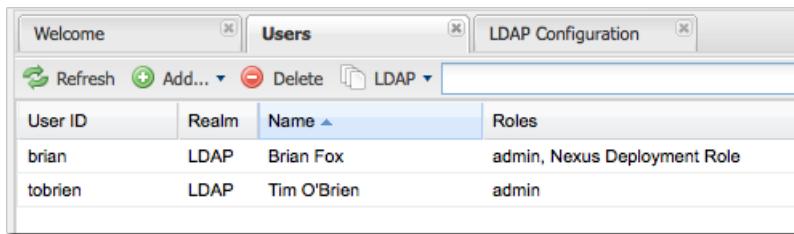
User ID	Realm	Name	Roles
admin	default	Administrator	Nexus Administrator Role
deployment	default	Deployment User	Repo: All Repositories (Full Control), Nexus Deployment Role
anonymous	default	Nexus Anonymous User	Nexus Anonymous Role, Repo: All Repositories (Read Only)

Figure 8.11: All Default Realm Users

If you wanted to see a list of all LDAP users, select "LDAP" from the "All Configured Users" drop-down shown in Figure 8.10 and click on the search button (magnifying glass) with an empty search field. Clicking search with an empty search field will return all of the LDAP users as shown in Figure 8.12.

Note

Note that the user "tobrien" does not show up in the "All Configured Users" list. This is by design. Nexus is only going to show you information about users with external role mappings. If an organization has an LDAP directory with thousands of developers, Nexus doesn't need to retain any configuration information for users that don't have custom Nexus role mappings.



The screenshot shows the Nexus web interface with the 'Users' tab selected. At the top, there are buttons for Refresh, Add..., Delete, and a dropdown menu set to 'LDAP'. Below the header is a table with four columns: 'User ID', 'Realm', 'Name', and 'Roles'. There are two rows of data:

User ID	Realm	Name	Roles
brian	LDAP	Brian Fox	admin, Nexus Deployment Role
tobrien	LDAP	Tim O'Brien	admin

Figure 8.12: All LDAP Users

To add a mapping for an external LDAP user, you would click on the "All Configured Users" drop-down and select LDAP. Once you've selected LDAP, type in the user ID you are searching for and click the search button (magnifying glass icon to right of the search field). In Figure 8.13, a search for "brian" yields one user from the LDAP server.

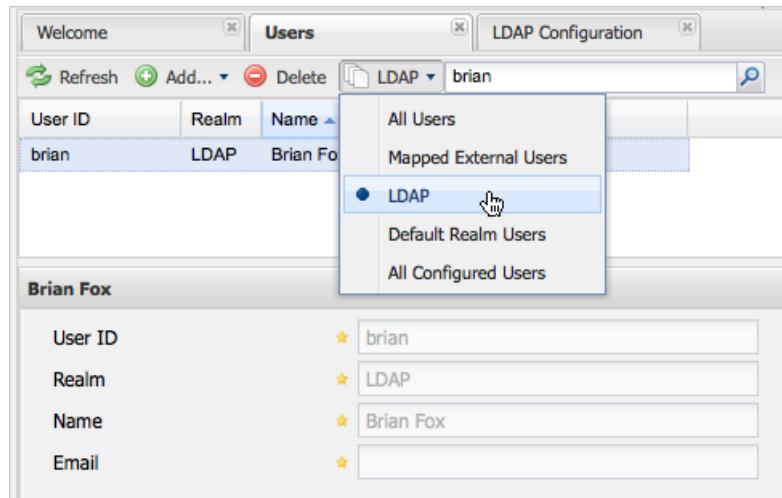


Figure 8.13: Search LDAP Users

To add a Nexus role mapping for the external user "brian" shown in Figure 8.13, click on the user in the results table and drag a role from Available Roles to Selected Roles as shown in Figure 8.14. In this case, the user "brian" is mapped to the Administrative group by virtue of his membership in an "admin" group in the LDAP server. In this use case, a Nexus administrator would like to grant Brian the Deployment Role without having to create a LDAP group for this role and modifying his group memberships in LDAP

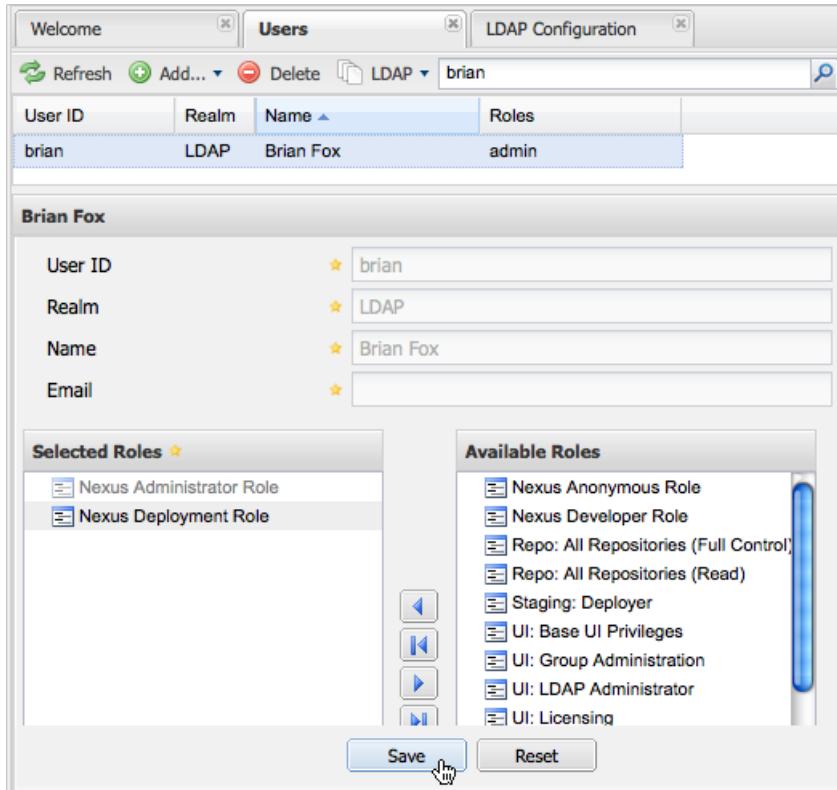


Figure 8.14: Mapping the Deployment Role to an External User

The end result of this operation is to augment the Group-Role mapping that is provided by the LDAP integration. You can use LDAP groups to manage coarse-grained permissions to grant people administrative privileges and developer roles, and if you need to perform more targeted privilege assignments in Nexus you can Map LDAP users to Nexus roles with the techniques shown in this section.

8.10 Mapping External Roles to Nexus Roles

Nexus makes it very straightforward to map an external role to an internal Nexus role. This is something you would do, if you want to grant every member of an externally managed group (such as an LDAP group) a certain privilege in Nexus. For example, assume that you have a group in LDAP named "svn" and you want to make sure that everyone in the "svn" group has Nexus Administrative privileges. To do

this, you would click on the Add.. drop-down in the Role panel as shown in Figure 8.15. This drop-down can be found in the Role management panel which is opened by clicking on Roles in the Security menu.

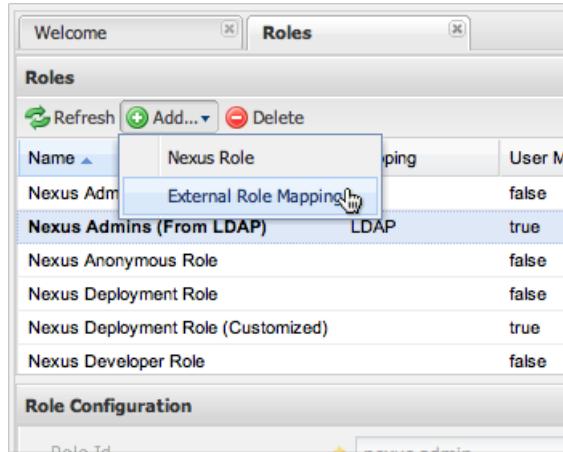


Figure 8.15: Selecting External Role Mapping in the Role Management Panel

Selecting External Role Mapping under Add... will show you a dialog which contains a drop-down of External Realms. Selecting an external realm such as LDAP will then bring up a list of roles managed by that external realm. The dialog shown in Figure 8.16 shows the external realm LDAP selected and the role "svn" being selected to map to a Nexus role.

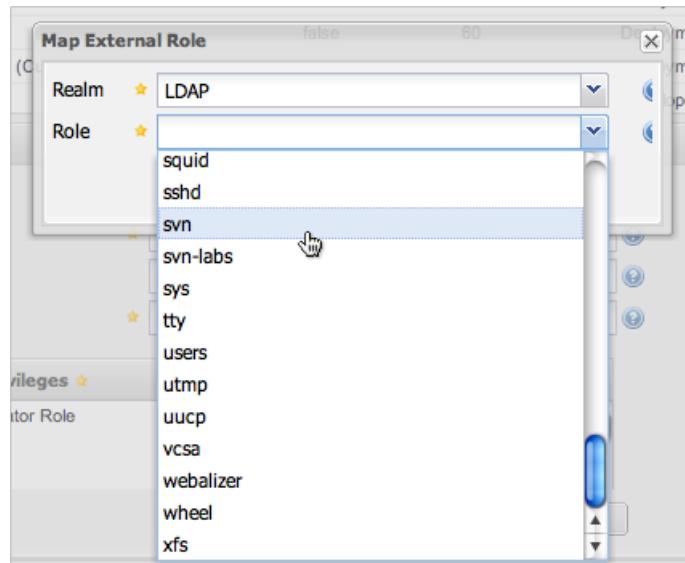


Figure 8.16: Selecting an Externally Managed Role to Map to a Nexus Role

Once the external role has been selected, Nexus will create a corresponding Nexus Role. You can then assign other Roles to this new externally mapped Role. Figure 8.17 shows that the SVN role from LDAP is being assigned the Nexus Administrator Role. This means that any user that is authenticated against the external LDAP Realm who is a member of the svn LDAP group will be assigned a Nexus role that maps to the Nexus Administrator Role.

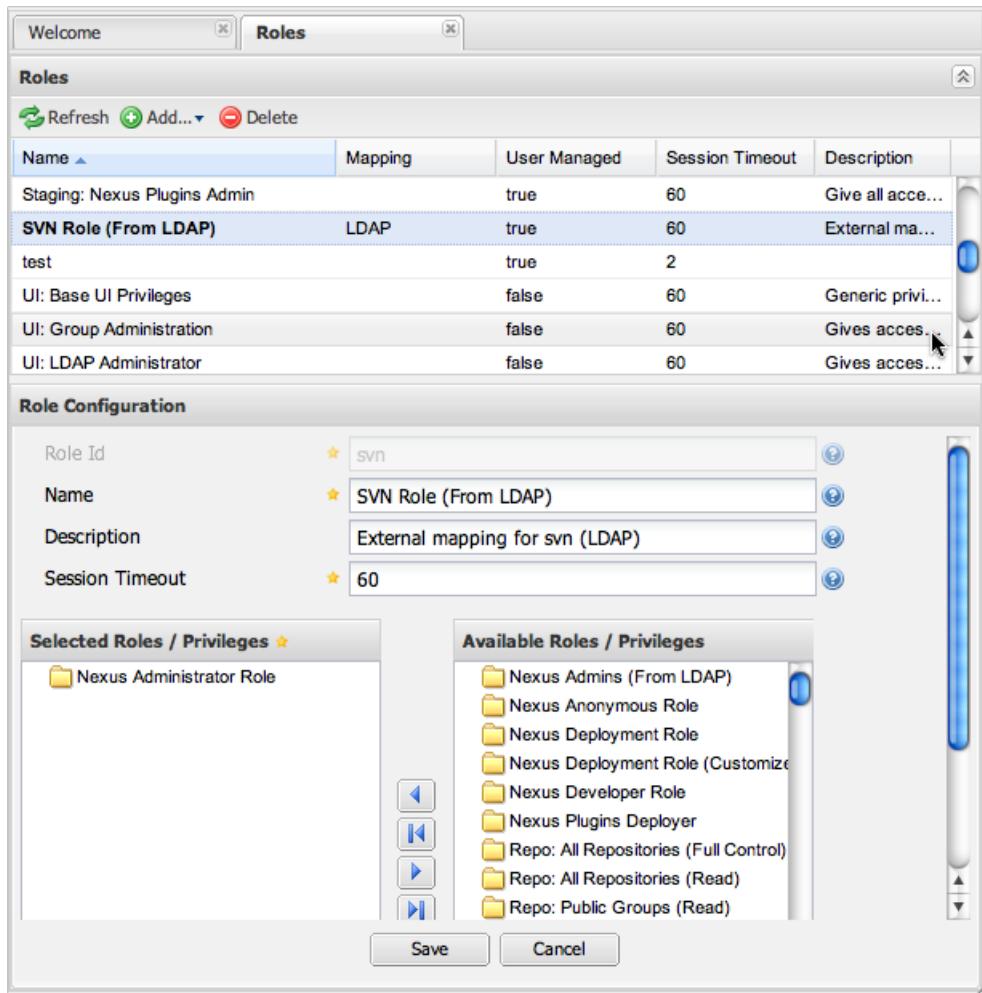


Figure 8.17: Mapping an External Role to a Nexus Role

8.11 Enterprise LDAP Support

The following sections outline Enterprise LDAP features which are available in Nexus Professional.

8.11.1 Enterprise LDAP Fail-over Support

When an LDAP server fails, the applications authenticating against it can also become unavailable. Because a central LDAP server is such a critical resource, many large software enterprises will install a series of primary and secondary LDAP servers to make sure that the organization can continue to operate in the case of an unforeseen failure. Nexus Professional's Enterprise LDAP plugin now provides you with the ability to define multiple LDAP servers for authentication. To configure multiple LDAP servers, click on Enterprise LDAP under Security in the Nexus application menu. You should see the Enterprise LDAP panel shown in the following figure.

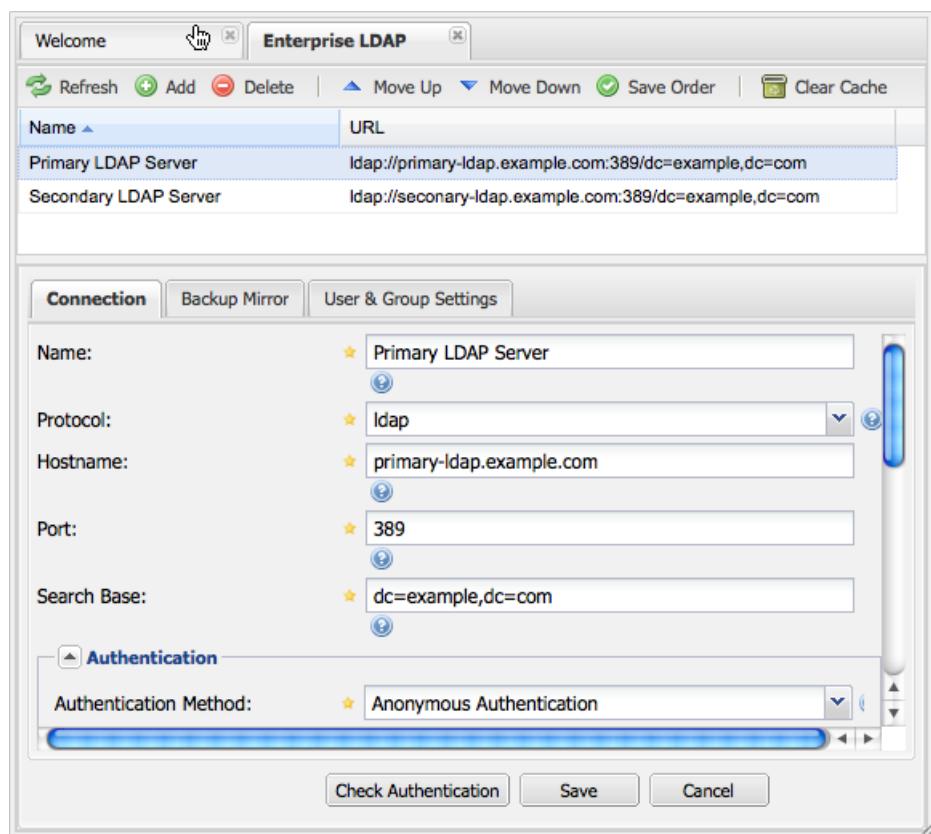


Figure 8.18: Defining Multiple LDAP Servers in Nexus Professional

You can use the Backup Mirror setting for an LDAP repository. This backup mirror is another LDAP server which will be consulted if the original LDAP server cannot be reached. Nexus Professional assumes

that the backup mirror is a carbon copy of the original LDAP server, and it will use the same user and group mapping configuration as the original LDAP server. Instead of using the backup mirror settings, you could also define multiple LDAP backup mirrors in the list of configured LDAP servers shown in the previous figure. When you configure more than one LDAP server, Nexus Professional will consult the servers in the order they are listed in this panel. If Nexus can't authenticate against the first LDAP server, Nexus Professional will move on to the next LDAP server until it either reaches the end of the list or finds an LDAP server to authenticate against.

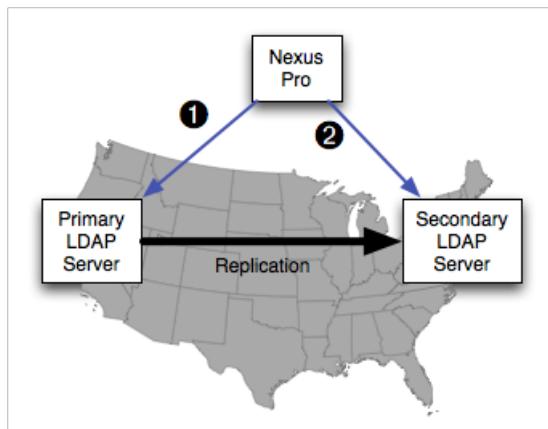


Figure 8.19: Use Multiple LDAP Servers in a Fail-over Scenario

The feature just described is one way to increase the reliability of your Nexus instance. In the previous case, both servers would have the same user and group information. The secondary would be a mirror of the primary. But, what if you wanted to connect to two LDAP servers that contained different data? Nexus Professional also provides...

8.11.2 Support for Multiple Servers and LDAP Schemas

The same ability to list more than one LDAP server also allows you to support multiple LDAP servers which may or may not contain the same user authentication information. Assume that you had an LDAP server for the larger organization which contained all of the user information across all of the departments. Now assume that your own department maintains a separate LDAP server which you use to supplement this larger LDAP installation. Maybe your department needs to create new users that are not a part of the larger organization, or maybe you have to support the integration of two separate LDAP servers that use different schema on each server.

A third possibility is that you need to support authentication against different schema within the same LDAP server. This is a common scenario for companies which have merged and whose infrastructures has not yet been merged. To support multiple servers with different user/group mappings or to support a single server with multiple user/group mappings, you can configure these servers in the Enterprise LDAP panel shown above. Nexus will iterate through each LDAP server until it can successfully authenticate a user against an LDAP server.

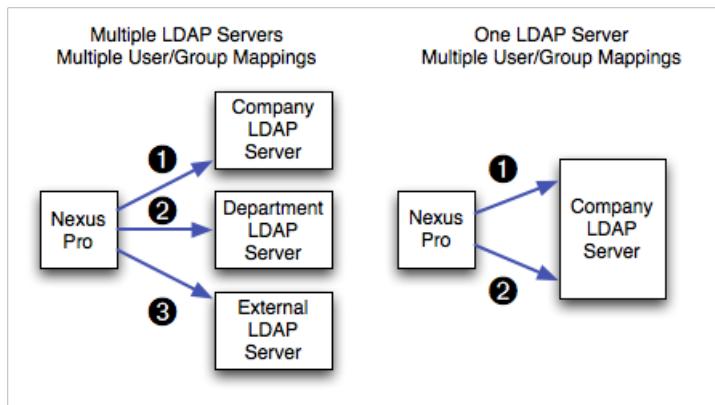


Figure 8.20: Supporting Multiple LDAP Schemas with Nexus Professional

8.11.3 Enterprise LDAP Performance Caching and Timeout

If you are constantly authenticating against a large LDAP server, you may start to notice a significant performance degradation. With Nexus Professional you can cache authentication information from LDAP. To configure caching, create a new server in the Enterprise LDAP panel, and scroll to the bottom of the Connect tab. You should see the following input field which contains the number of seconds to cache the results of LDAP queries.

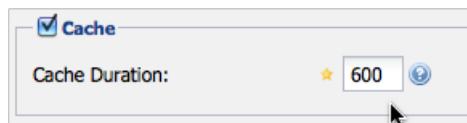


Figure 8.21: Setting the LDAP Query Cache Duration (in Seconds)

You will also see options to alter the connection timeout and retry interval for an LDAP server. If you are

configuring a number of different LDAP servers with different user and group mappings, you will want to make sure that you've configured low timeouts for LDAP servers at the beginning of your Enterprise LDAP server list. If you do this properly, it will take Nexus next to no time to iterate through the list of configured LDAP servers.

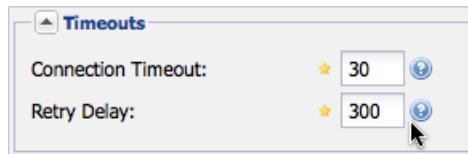


Figure 8.22: Setting the LDAP Connection Timeout (in Seconds)

We improved the overall caching in this release. The cache duration is configurable and applies to authentication and authorization, which translates into pure speed! Once you've configured LDAP caching in Nexus Professional, authentication and other operations that involve permissions and credentials once retrieved from an external server will run in no time.

8.11.4 User and Group Templates

If you are configuring your Nexus Professional instance to connect to an LDAP server there is a very good chance that your server follows one of several, well-established standards. Nexus Professional's LDAP server configuration includes these widely used user and group mapping templates which great simplify the setup and configuration of a new LDAP server. To configure user and group mapping using a template, select a LDAP server from the Enterprise LDAP panel, and choose the User and Group Settings. You will see a User & Group Templates section as shown in the following figure.



Figure 8.23: Using User & Group Mapping Templates

8.11.5 Testing a User Login

Nexus Professional provides you with the ability to test a user login directly. To test a user login, go to the User and Group Settings tab for a server listed in the Enterprise LDAP panel. Scroll to the bottom of the form, and you should see a button named "Check Login".

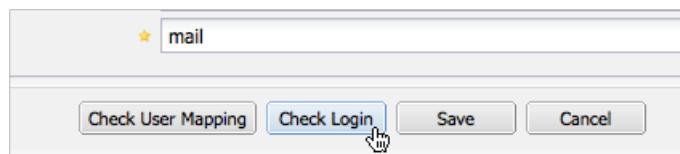


Figure 8.24: Testing a User Login

If you click on Check Login, you will then be presented with the login credentials dialog shown below. You can use this dialog to login as an LDAP user and test the user and group mapping configuration for a particular server. This feature allows you to test user and group mapping configuration directly. This feature allows you to quickly diagnose and address difficult authentication and access control issues via the administrative interface.

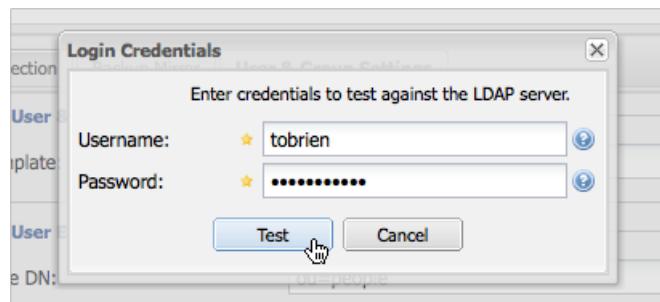


Figure 8.25: Supply a User's Login Credentials

Chapter 9

Nexus Procurement Suite

9.1 Introduction

Nexus Procurement Suite provides an organization with control over what components are allowed into a repository from an external, proxied repository such as the Central Repository. Such control can be a prerequisite for organizations unwilling or unable to trust the entire contents of an external public repository. If an organization is developing mission critical code, they will likely want to subject every third party dependency to intense scrutiny and testing before making the component available to build a release or support a team of developers. In most Enterprise development environments, a developer can't just decide to add in a new dependency to Hibernate or to the Spring Framework on a whim; the decision to add dependencies to third-party libraries will need to be funnelled through an oversight process that relies on an architect or an administrator to promote components to a certified release repository.

Another, more common experience is an organization which needs to proxy something like the Central Repository, but wants to limit access to specific versions of components or prevent dependencies on everything contained under a specific group. Some organizations are more amenable to trusting the contents of a remote, proxied repository like the Central Repository, but they also need the ability to block certain dependencies. Maybe you work on a team that needs to limit access to dependencies with a certain license, or maybe you just want to make sure no one uses a problematic version of Hibernate with a known bug? The procurement suite is the tool that provides for both coarse and fine-grained control of the artifacts that can appear in a repository.

9.2 The Stages of Procurement

A procured repository is a hosted Repository which procures components from a Proxy Repository, while procurement is enabled. For example, one could create a hosted repository named "Approved From Central" and then configure this hosted repository to procure components from the "Central" repository. Once the hosted repository has been created and the source of procurement has been configured, the repository will obtain components from the proxy repository as long as procurement is activated. If you start procurement for a hosted repository, the hosted repository will fetch artifacts from the proxy repository specified in the procurement settings. If you stop procurement for a hosted repository, no additional components will be retrieved from the proxy repository specified in the procurement settings. Without procurement active it is a plain, static hosted repository.

The ability to enable or disable procurement for a hosted repository comes in very handy when you want to "certify" a hosted repository as containing all of the components (no more and no less) required for a production build. You can start procurement, run a build which triggers artifact procurement, and then stop procurement knowing that the procured repository now contains all of the components required for building a specific project. Stopping procurement assures you that the contents of the repository will not change if the third-party, external proxied repository does. This is an extra level of assurance that your release components depend on a set of components under your complete control.

9.3 Two Approaches to Procurement

There are two main use cases for the Procurement Suite. In the first use case, the *Procured Release Repository*, the procurement features are used to create a procured release repository to make sure that the organization has full control over the components that are making it into a production release. The other use case, the *Procured Development Repository*, is for organizations that need more up-front control over which artifacts are allowed during the development of a project. The following sections describe these two uses cases in more detail.

```
[ [procure-sect-cert] ]  
===== Procured Release Repository
```

The Procurement Suite can be used in two different ways. In the "Procured Release" mode, developers work with a proxied 3rd party repository exactly as they would without the Procurement Suite. When a developer needs to add a dependency on a new artifact, Nexus will retrieve the artifact from the third-party repository (like Central or Apache Snapshots) and this artifact will be served to Maven via a proxied Nexus repository. When a QA or Release engineer needs to build a release or staging artifact, the Release

or QA build would be configured to execute against a procured repository or repository group with only approved and procured repositories. A procured repository is one which only serves the components which have been explicitly approved using the Procurement Suite.

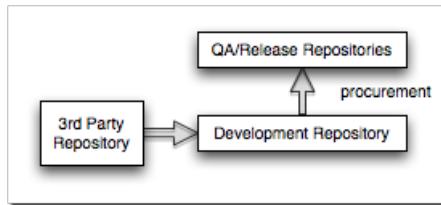


Figure 9.1: Procurement to a Certified Release Repository

In this model, developers can add as many third-party dependencies as they want, and it is the responsibility of the QA and release engineers to approve (or procure) components from the development Repository to the QA/Release repository. Developers can move forward, adding dependencies freely from a third-party, proxied repository, but once it is time to populate a release repository, a Nexus administrator can audit the required components, create a hosted repository, turn on procurement, populate the repository, and then deactivate procurement. This has the effect of "locking down" the components that are involved in a production release.

9.3.1 Procured Development Repository

There are some development environments, which require even more control over which components can be used and referenced by developers. In these situations, it might make sense to only allow developers to work with a procured repository. In this mode, a developer must ask a Nexus administrator for permission to add a dependency on a particular third-party artifact. A procurement manager would then have to approve the component, or group of components so that they would be made available to the developers. This is the "ask-first" model for organizations that want to control which components make it into the development cycle.

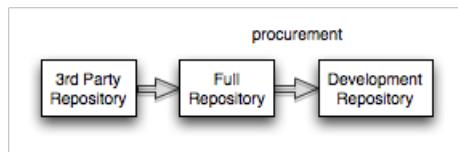


Figure 9.2: Procurement to a Certified Development Repository

This is a model common in industries that have strict oversight requirements. More often than not, banks, hospitals, and government agencies have fairly strict regulations on the software that can be used by large development teams. With the Procured Development Repository approach, an architecture group can have full control over what components can be referenced by a large development team.

9.3.2 Providing Access With A Repository Group

In a typical usage a software build relies on approved components that have successfully passed procurement and as well as additional components that have been authored internally in the organization and are available on Nexus as well.

In order to use a combination of such components together with the procured component, you should set up a repository group that contains all repositories with pre-approved components as well as the procurement repository. E.g. the release and snapshot repositories could be added to the group, based on the assumption that any internally authored components deployed there are automatically approved. In addition you could add the 3rd party repository, if all uploads to it are done with prior approval of the specific components.

Once this repository group is set up, you can reference it from any tool just like the public group e.g. in a separate settings.xml used by builds that can only have access to the approved components.

Tip

Ensure to run clean builds without any components in the local repository of the build to ensure that only approved components are used in the build.

9.4 Setting up a Procured Repository

If you installed Nexus Professional, the Nexus Procurement Suite is already installed and available via the Artifact Procurement option in the Enterprise menu of the Nexus interface.

This section will walk through the process of creating and configuring a hosted repository named "Approved From Central" which will be procured from the "Central" proxy repository. Setting up a procured repository consists of the following steps:

- Enabling Remote Index Downloads for a Proxy Repository
- Creating a Hosted Repository to be Procured
- Configuring Procurement for the Hosted Repository
- Configuring Procurement Rules

Before configuring a procured repository, you need to make sure that you have enabled Remote Index downloading for the proxied repository that will serve as the source for your procured repository.

Note

If you are attempting to procure components from a remote repository which does not have a repository index, you can still use the procurement suite. Without a remote repository index, you will need to configure procurement rules manually without the benefit of the already populated repository tree shown in Section 9.5.

9.4.1 Enable Remote Index Downloads

When you configure procurement rules for a hosted repository, the administrative interface displays the repository as a tree view using the Maven repository format of groups and components using populated from remote repository's index. Nexus ships with a set of proxy repositories, but remote index downloading is disabled by default.

To use procurement effectively, you will need to tell Nexus to download the remote indexes for a proxy repository. Click on "Repositories" under Views/Repositories in the Nexus menu, then click on the Central Repository in the list of repositories. Click on the Configuration tab, locate Download Remote Indexes, and switch this option to "True" as shown in Figure 9.3.

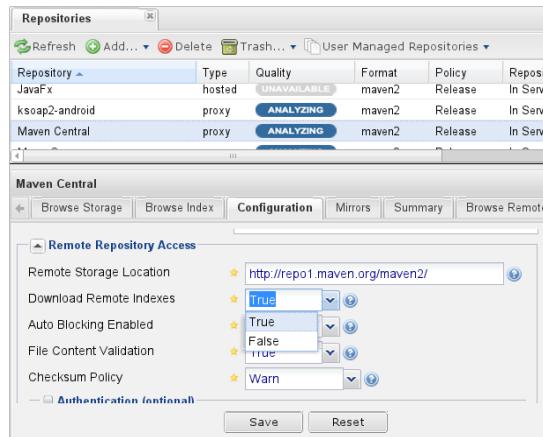


Figure 9.3: Enabling Remote Index Downloads for a Proxy Repository

Click on the Save button in the dialog shown in Figure 9.3. Right-click on the repository row in the Repositories list and select "Update Index". Nexus will then download the remote repository index and recreate the index for any Repository Groups that contain this proxied repository.

Nexus may take a few minutes to download the remote index for a large repository. Depending on your connection to the Internet, this process can take anywhere from under a minute to a few minutes. The size of the remote index for the Central Repository currently exceeds 50MB and is growing in parallel to the size of the repository itself.

To check on the status of the remote index download, click on System Feeds under Views in the Nexus menu. Click on the last feed to see a list of "System Changes in Nexus". If you see a log entry like the one highlighted in Figure 9.4, Nexus has successfully downloaded the Remote Index from Maven Central.

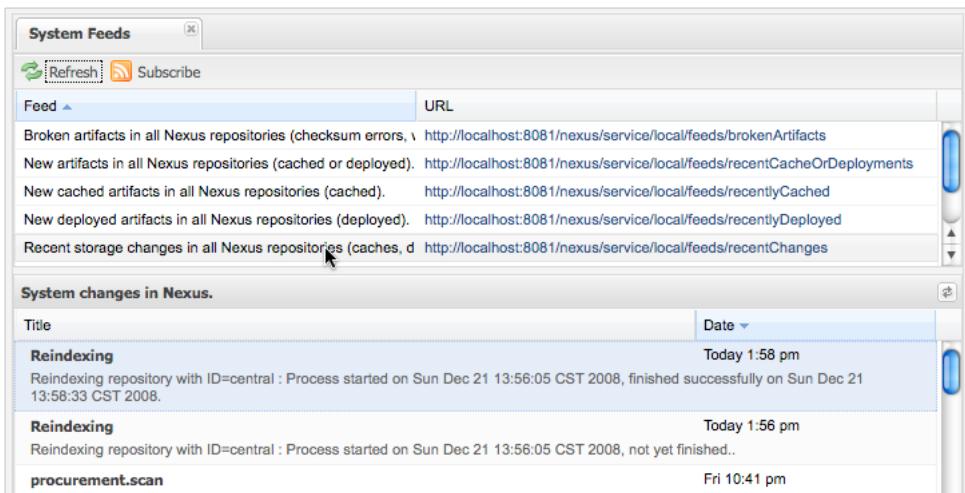


Figure 9.4: Verification that the Remote Index has been Downloaded

9.4.2 Create a Hosted Repository

When you configure procurement you are establishing a relationship between a proxy repository and a hosted repository. The hosted repository will be the static container for the components, while the proxy repository acts as the component source. To create a hosted repository, select *Repositories* from the *Views.Repositories* section of the Nexus menu, and click on the *Add* button selecting *Hosted Repository* as shown in Figure 9.5.

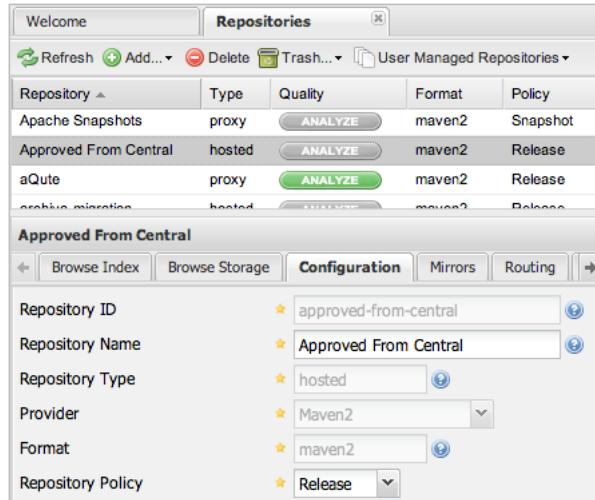


Figure 9.5: Adding the "Approved From Central" Hosted Repository

Selecting Hosted Repository will then load the Configuration form. Create a repository with a Repository ID of "approved-from-central" and a name of "Approved From Central". Make the release policy "Release". Click the Save button to create the new Hosted Repository.

9.4.3 Configuring Procurement for Hosted Repository

At this point, the list of Repositories will have a new Hosted repository named "Approved From Central". The next step is to start procurement for the new repository. When you do this, you are establishing a relationship between the new hosted repository and another repository as source of components. Typically this source is a proxy repository. In this case, we're configuring procurement for the repository and we're telling the Procurement Suite to procure artifacts from the *Central* proxy repository. To configure this relationship and to start procurement, click on Artifact Procurement under the Enterprise menu. In the Procurement panel, click on Add Procured Repository as shown in Figure 9.6.



Figure 9.6: Adding a Procured Repository

You will then be presented with the Start Procurement dialog as shown in Figure 9.7. Select the "Central" proxy repository from the list of available Source repositories.

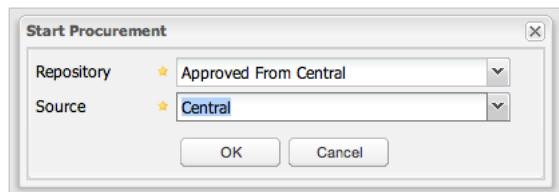


Figure 9.7: Configuring Procurement for a Hosted Repository

Procurement is now configured and started, if you are using an instance of Nexus installed on localhost port 8081, you can configure your clients to reference the new repository at <http://localhost:8081/nexus/content/repositories/approved-from-central>

By default, all artifacts are denied and without further customization of the procurement rules no components will be available in the new repository.

One interesting thing to note about the procured repository is that the repository type changed, once procurement was started. When procurement is activate for a hosted repository, the repository will not show up in the repositories list as a *User Managed Repository*. Instead it will show up as a proxy repository in the list of *Nexus Managed Repositories*. Use the drop down for User Managed/Nexus Managed Repositories in the Repositories list. Click Refresh in the Repositories list, and look at the *Approved From Central* repository in the list of Nexus Managed Repositories. You will see that the repository type

column contains "proxy" as shown in Figure 9.8. When procurement is started for a hosted repository it is effectively a proxy repository, and when it is stopped it will revert back to being a normal hosted repository.

The screenshot shows the Nexus 'Repositories' screen. A table lists repositories, with one entry highlighted: 'Approved From Central' (Type: proxy). Below the table, a detailed view for 'Approved From Central' is displayed, showing its configuration: Repository ID: approved-from-central, Repository Name: Approved From Central, Repository Type: proxy, Repository Policy: Release, Repository Format: maven2, Contained in groups: Approved Artifacts Only, and Remote URL: repositories:central/. The 'ANALYZE' button is highlighted in green.

Figure 9.8: Hosted Repository is a Nexus Managed Proxy Repository while Procurement is Active

9.4.4 Procured Repository Administration

Once you've defined the relationship between a hosted repository and a proxy repository and you have started procurement, you can start defining the rules which will control which components are allowed in a procured repository and which components are denied. You can also start and stop procurement. This section details some of the administration panels and features which are available for a procured repository.

A procurement rule is a rule to allow or deny the procurement of a group, artifact, or a collection of groups or artifacts. You load the Artifact Procurement interface by selecting Artifact Procurement in the Enterprise menu of the Nexus left hand navigation. Clicking on this link will load a list of procured repositories. Clicking on the repository will display the proxied source repository and the current content of the procured repository in a tree as shown in Figure 9.9.

This section will illustrate the steps required for blocking access to an specific component and then selectively allowing access to a particular version of that same component. This is a common use-case in organizations which want to standardize on specific versions of a particular dependency.

Note

If you are attempting to procure components from a remote repository which does not have a repository index, you can still use the procurement suite. Without a remote repository index, you will need to configure procurement rules manually without the benefit of the already populated repository tree shown in this section.

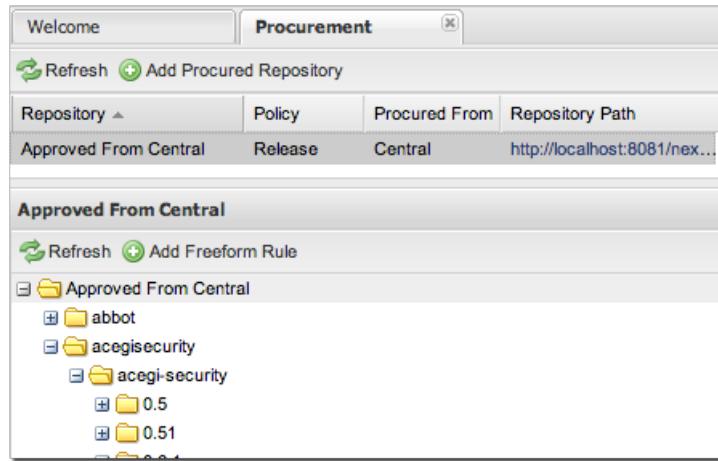


Figure 9.9: Viewing a Repository in the Artifact Procurement Interface

The directory tree in Figure 9.9 is the index of the proxy repository from which artifacts are being procured.

9.5 Configuring Procurement

To configure a procurement rule, right click on a folder in the tree. Figure 9.10 displays the procurement interface after right clicking on the `org/eclipse/aether` component folder.

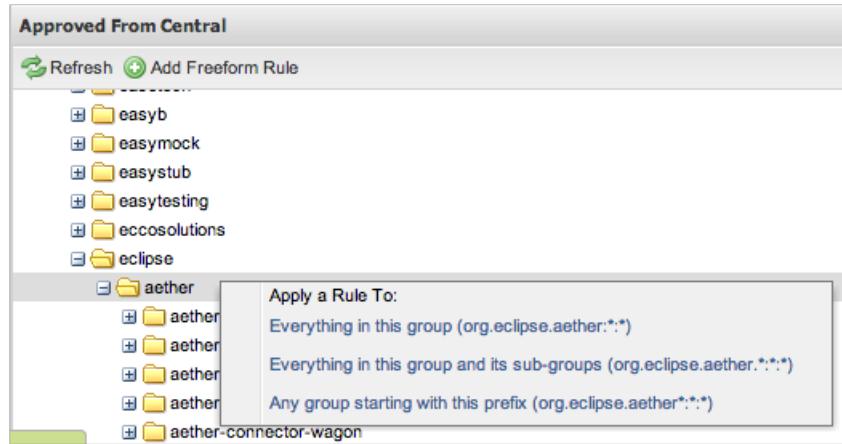


Figure 9.10: Applying a Rule to a Component Folder for `org/eclipse/aether`

In this dialog, we are deciding to configure a rule for everything within the group and its sub-groups, which will bring up the rule configuration dialog displayed in Figure 9.11. The dialog to add rules allows you to select the available rule, e.g. a Forced Approve/Deny Rule, and configure the rule properties. The displayed dialog approves all components Eclipse Aether components.

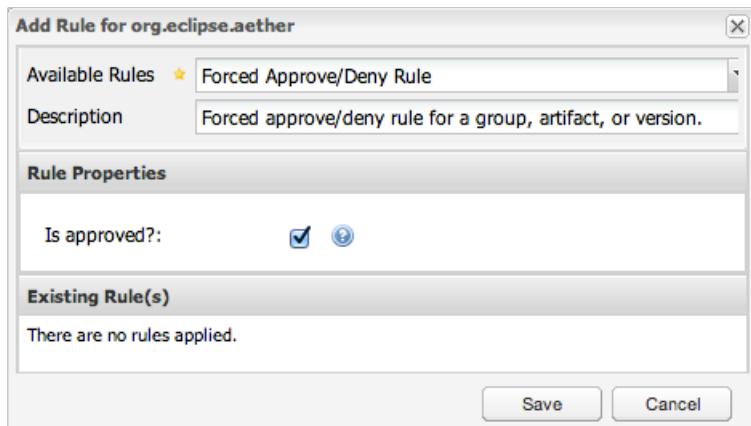


Figure 9.11: Approving `org.eclipse.aether` Components

By right clicking on the top level folder of the repository as displayed in Figure 9.12 you can configure rules for the complete repository as well as access all configured rules via the *Applied Rules* option.



Figure 9.12: Accessing the Global Repository Configuration

This allows you to setup a global rule like blocking all components from the repository. Once you have configured this you can then selectively allow specific versions of a component. Figure 9.13 displays the options available for configuring rules for a specific component version of the Apache Commons Collections component.

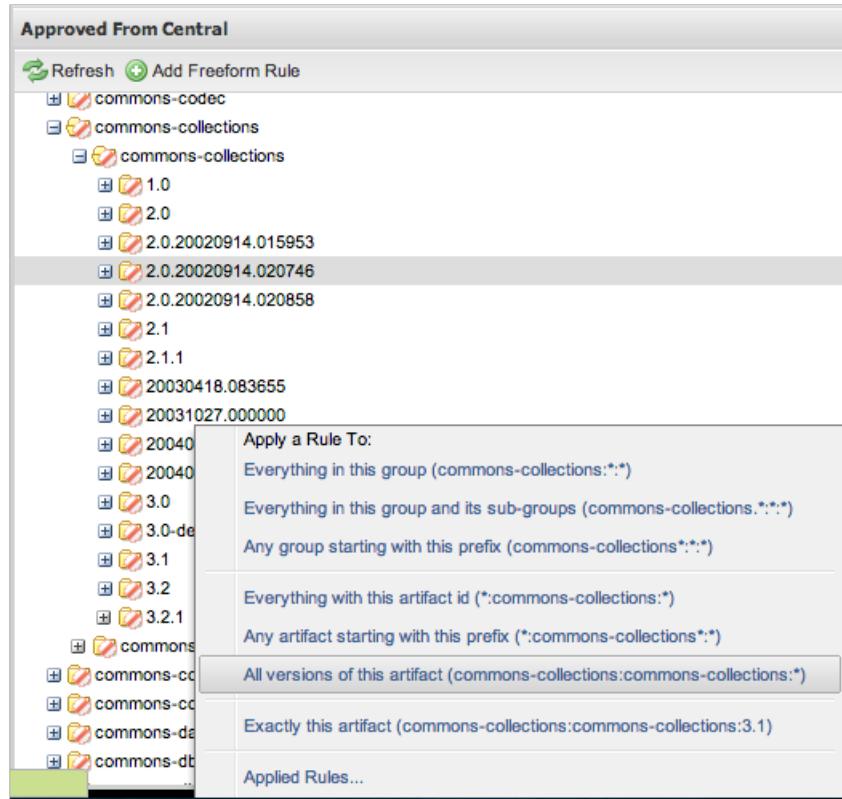


Figure 9.13: Procurement Configurations Options for a Specific Component Version

Once you approve a specific version the tree view will change the icons for the component displaying green checkmarks for approved components and red cross lines for denied components as visible in Figure 9.14. The icons are updated for signature validation rule violations, if applicable, showing a yellow icon.

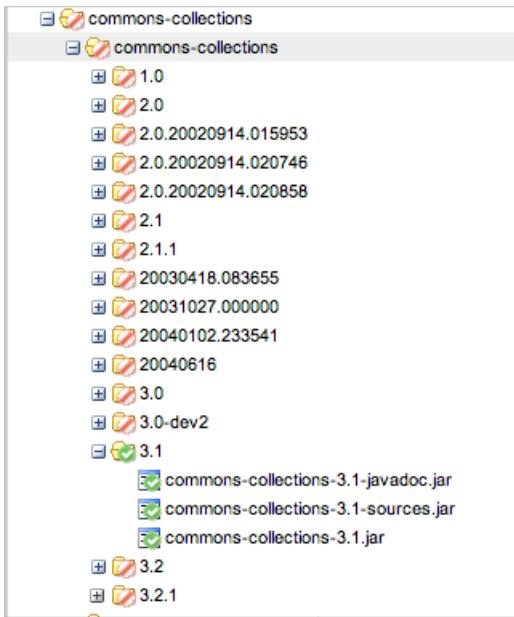


Figure 9.14: Procurement Repository Tree View with Rule Visualization

An example dialog of Applied Rules for the complete repository, as configured by `::*`, is visible in Figure 9.15. This repository currently denies access to all components, only approving components within `org/apache/maven` and `org/eclipse/aether`.

This dialog gives the procurement administrator a fine-grained view into the rules that apply to the complete repository. A view of all Applied Rules for a specific repository folder can be accessed by right-clicking on the folder and selecting Applied Rules. The dialog allows you to remove specific rules or all rules as well.

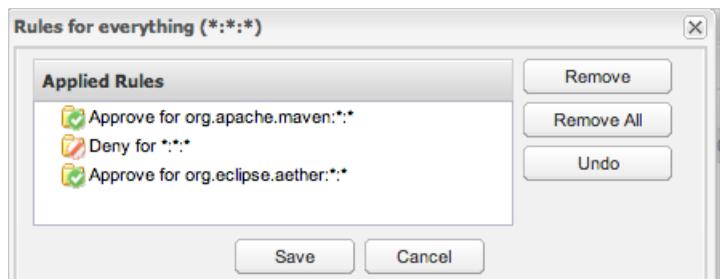


Figure 9.15: Applied Rules for the Complete Procurement Repository

The *Refresh* button above the tree view of a repository tree view allows you to update the tree view and the visualization of all applied rules. The *Add Freeform Rule* button allows you to bring up the dialog to manually configure a procurement rule displayed in Figure 9.16. This is especially useful if the tree view is not complete due to a missing repository index or if you have detailed knowledge of the component you want to apply a rule to. The format for entering the a specific component to be in the *Enter GAV* input field is the short form for a Maven component coordinate using the groupId, artifactId and version separated by :. The * character can be used as a wildcard for a complete coordinate.

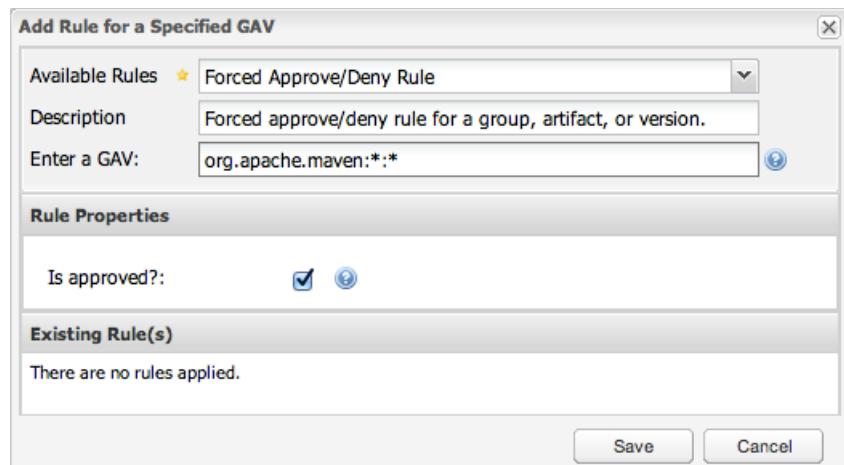


Figure 9.16: Adding a Freeform Rule

Examples for a freeform rule coordinates are:

`::*` matches any component in the complete repository

`org.apache.ant:::*`

matches any component with the groupId org.apache.ant located in org/apache/ant

`org.apache.ant:*::*`

matches any component with the groupId org.apache.ant located in org/apache/ant

`org.apache.ant.*::*`

matches any component with the groupId org.apache.ant located in org/apache/ant as well as any sub-groups e.g. `org.apache.ant.ant`

These coordinates are displayed as part of a Maven built when retrieving a component fails as part of the error message with the addition of the packaging type. It is therefore possible to cut and paste the respective coordinates from the build output and insert them into a freeform rule. Once you have done that you can kick off the build again, potentially forcing downloads with the option `-U` and continue procurement configuration for further components.

9.6 Stopping Procurement

Some organizations may want to "lock down" the components that a release build can depend upon, and it is also a good idea to make sure that your build isn't going to be affected by changes to a repository not under your control. A procurement administrator can configure a procured repository, start procurement, and run an enterprise build against the repository to populate the procured, hosted repository with all of the necessary components. After this process, the procurement administrator can stop procurement and continue to run the same release build against the hosted repository which now contains all of the procured components, while being a completely static repository.

To stop procurement, go to the Artifact Procurement management interface by clicking on Artifact Procurement under the Enterprise section of the Nexus menu. Right click on the repository and choose Stop Procurement as shown in Figure 9.17.

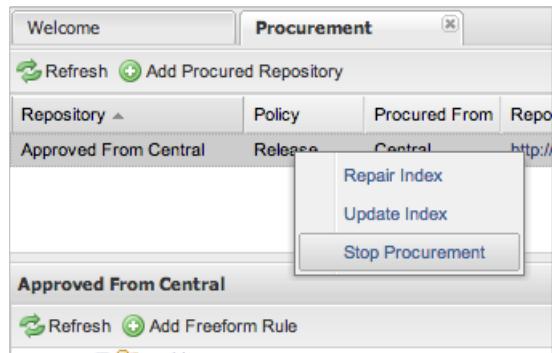


Figure 9.17: Stopping Procurement for a Procured Repository

After choosing Stop Procurement, you will then see a dialog confirming your decision to stop procurement. Once procurement is stopped, the Secure repository will revert back to being a plain-old Hosted Repository.

In order to add further components you create a procurement repository off the hosted repository as you did initially. If the repository contains components already, activating procurement will automatically generate rules that allows all components already within the repository.

Chapter 10

Build Promotion with the Nexus Staging Suite

10.1 Introduction

If you release software, you will often need to test a release before deploying it to a production system or an externally accessible repository. For example, if you are developing a large, enterprise web application you may want to stage a release candidate to a staging system and perform a series of rigorous tests before a release manager makes a decision to either return a system to development or deploy a system to production.

The Staging Suite in Nexus Professional allows an organization to create a temporary staging repository and to manage the promotion of artifacts from a staging repository to a release repository. This ability to create an isolated, release candidate repository that can be discarded or promoted makes it possible to support the decisions that go into certifying a release, while the certification process is done on the same binaries that will ultimately be released.

10.1.1 Releasing Software without a Staging Repository

Without the Staging Suite, when a developer deploys an artifact to a hosted repository such as the Release repository, this artifact is published and immediately made available - there is no oversight, there is no approval or certification process. There is no chance to test the artifact before writing the artifact to a hosted repository. If there is a mistake in the release, often the only option available is to republish the artifacts to the release repository or deploy a new version of the artifacts.

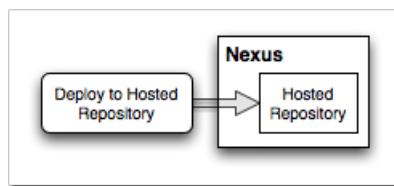


Figure 10.1: Release Deployment Without the Nexus Staging Suite

While this is acceptable for some users, organizations and enterprises with a QA cycle often need a temporary storage for potential release candidates: a staging repository. With the Nexus Staging Suite, an organization can automatically stage releases to a temporary repository which can then be used to test and certify a set of artifacts, before they are published to a final release repository. This temporary repository can then be promoted as a whole or dropped depending on the results of testing. When used the binary artifacts being tested and certified are the identical artifacts that will ultimately be released and not a clean fresh build kicked off after the certification finished as is often the case without a staging suite being used.

10.1.2 How the Staging Suite Works

Here's how staging works in Nexus Professional:

1. A developer deploys an artifact (or a set of artifacts) to Nexus Professional
2. The staging suite intercepts this deployment and determines if the deployment matches for a staging profile
3. If a match is found a temporary staging repository is created and the artifacts are deployed to this repository.
4. Once the developer has deployed a set of artifacts to Nexus, they will then "Close" the staging repository.

5. The Staging Suite will then add this temporary staging repository to one or more Target Repository Groups.

Once the staging repository is closed and has been added to a target repository group, the artifacts in the staging repository are available to Nexus users for testing and certification via a repository group. Tests can be performed on the artifacts as if they were already published in a hosted repository. At this point different actions can be performed with the staging repository:

Release

A Nexus user can "release" a staging repository and select a hosted repository to publish artifacts to. Releasing the contents of a repository publishes all artifacts from the staging repository to a hosted repository and deletes the temporary staging repository.

Drop

A Nexus user can "drop" a staging repository. Dropping a staging repository will remove it from any groups and delete the temporary staging repository.

Promote

If your Nexus installation contains Build Promotion profiles, you will also see an option to "promote" a staging repository to a Build Promotion Group. When you promote a staging repository you expose the contents of that staging repository via additional groups. Build Promotion profiles are explained in detail in the next section.

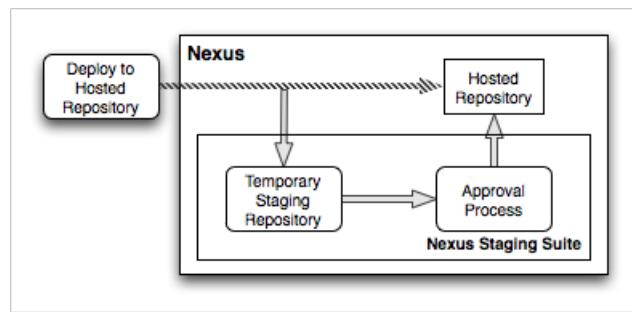


Figure 10.2: Release Deployment with the Nexus Staging Suite

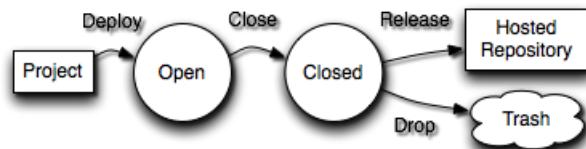


Figure 10.3: The Stages of a Staging Repository starting with Deployment and Ending with a Release or a Drop of the Repository

10.2 Configuring the Nexus Staging Suite

10.2.1 Overview

The Staging Suite is part of the default Nexus Professional install and is accessible with the menu items **Staging Profiles**, **Staging Repositories**, **Staging Ruleset**, and **Staging Upload** options in the left-hand navigation menu of the Nexus interface called **Build Promotion**.

Staging Profiles define the rules by which artifact deployments from your project are intercepted by Nexus and staged in Staging Repositories.

Staging Repositories are dynamically created by Nexus as they are needed. They are temporary holding repositories for your artifacts that are used for the different staging related steps. Using them in the Nexus user interface users can promote the contents of the staging repository to a hosted repository, discard them and more.

Staging Rulesets allow you to define rules that the artifacts being deployed have to follow in order to allow successful deployment.

Staging Upload allows you to manually upload artifacts to Nexus via the user interface rather than by using your build system.

10.2.2 Configuring a Staging Profile

Staging profiles control the process by which artifacts are selected for staging. When you define a Staging profile, you are defining a set of rules which will control the way in which Nexus intercepts an artifact deployment and what actions to perform during and after staging the artifacts. When you click on Staging Profiles in the Nexus menu, you will see a list of configured staging profiles. This list allows you to Add... and Delete staging profiles. Click on an existing staging profile in the list and the panel below the list will display the configuration of the profile.

The list of staging profiles displayed also determines the order in which the profiles are examined when a component is deployed to staging. Going down the list each profile is checked for a match of the deployed component characteristics to the configuration of the profile. If a match is found a staging repository for this profile with the deployed components is created. Otherwise the next profile in the list is examined. Specifically with implicit matching criteria being used for your deployments as explained in more detail below, this order becomes important and can be controlled by selecting a staging profile and using the Move Up and Move Down buttons on the top of the list. Once you have created the desired order, press the Save Order button and confirm the order in the dialog.

Clicking on Add... will display the drop-down menu shown in Figure 10.4.

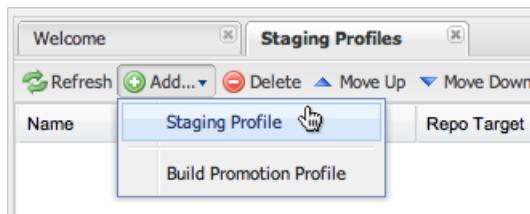


Figure 10.4: Adding a Staging Profile

Selecting Staging Profile will create a new Staging Profile and display the form shown in Figure 10.5.

Figure 10.5 defines a Staging Profile named "Test". It is configured to only intercept explicit deployments in the Profile Selection Strategy using the Profile ID and the Nexus Staging Maven Plugin. It uses the template "Maven2 (hosted, release)" for newly created temporary staging repositories, and it will automatically add closed staging repositories to the Public Repositories group. In addition it is configured to verify the deployment against the rules defined in Sonatype CLM for the CLM Application Id "bom1-12345678".

Name	Mode	Repo Target	Release Re...	Target Groups
nexusclass	Deploy and UI Upload	Sonatype Sample	Releases	Public Repositories
TrainingRelease	Deploy and UI Upload	Simpligility	Releases	Public Repositories
CLMTest	Deploy	All (Maven2)	Releases	Approved Artifacts Only, Public Repositories
Test	Deploy and UI Upload	All (Maven2)	Releases	Public Repositories

Test

Profile ID: 12f3a105e1212751
 Deploy URL: http://localhost:8081/nexus/service/local/staging/deploy/maven2

Profile Name: Test

Profile Selection Strategy: Explicit Only - only selected by ID

Searchable Repositories:

Staging Mode: Deploy and UI Upload

Template: Maven2 (hosted, release)

Repository Target: All (Maven2)

Release Repository: Releases

CLM Application Id: bom1-12345678

Content Type: maven2

Target Groups

- Public Repositories

Available Groups

- Approved Artifacts Only

Figure 10.5: Creating a New Staging Profile

The form allows you to configure a profile with the following fields:

Profile ID and Deploy URL

These two fields are only available as read only display once a profile has been created. The Profile ID displays the unique identifier that can be used for staging to this repository using the Nexus Staging Maven Plugin. The Deploy URL displays the generic staging URL that can be used with the default Maven Deploy Plugin together with the Repository Target configuration to intercept the deployment and move the artifacts into the Staging Suite instead.

Profile Name

The name of the staging profile. This can be an arbitrary value. It is simply a convenience for the

Nexus Administrator, and it is also used to create Nexus roles that are used to grant permissions to view and manipulate staging repositories created by this profile.

Profile Selection Strategy

Select the strategy used by Nexus to select this staging profile. Explicit or Implicit is the default behaviour and causes Nexus to select the profile by the provided staging profile identifier and if none is provided fall back to an automatic determination. It is necessary to be used with the Maven Deploy Plugin and the correct staging profile is determined using Repository Targets together with the generic Deploy URL of Nexus.

When using the Nexus Staging Maven Plugin for deployments, and therefore an explicitly defined staging profile in the project POM, the setting should be changed to Explicit Only. This will prevent the profile from implicitly capturing a deployment in this repository due to the matching defined and allow Nexus to ensure that the deployment reaches the Staging Profile with the configured Staging Profile ID even if the default matching and staging profile order could potentially cause a deployment to end up in a different profile.

Searchable Repositories

The default value of enabling this feature will cause any new artifacts in this staging profile to be added to the indexes and therefore be available in search queries. Disable this feature to "hide" artifacts in staging.

Staging Mode

This field contains the options "Deploy," "UI Upload," and "Deploy and UI Upload." This controls how artifacts can be staged to this staging profile. If Deploy is selected, artifacts can only be deployed using Maven to upload build artifacts. If UI Upload is selected, users can upload artifacts to Nexus using the Nexus user interface.

Template

Defines the template for the format of the temporary staging repositories created by this staging profile. The current version of Nexus Professional provides the option "Maven2 (hosted, release)" only. Additional templates can be supplied by plugins that enable staging for other repository types. An example for such a plugin is the [Nexus Yum Plugin](#).

Repository Target

When a developer deploys an artifact to the generic Deploy URL, the Staging Suite will check to see if the artifact matches the patterns defined in this Repository Target. The repository target defines the "trigger" for the creation of a staging repository from this staging profile and is only needed for implicit deployments with the Deploy URL and not for explicit deployments using the Profile ID.

Release Repository

Staged artifacts are stored in a temporary staging repository which is made available via Target Groups. Once a staged deployment has been successfully tested, artifacts contained in the temporary staging repository are promoted to a hosted repository as their final storage place. The Release Repository setting configures this target release repository for this staging profile.

CLM Application Id

Configures the application identifier defined in the Sonatype CLM server, allowing you to use the rules defined there for staging within Nexus. More details can be found in Chapter [12](#).

Content Type

Nexus can create staging repositories for repositories of type Maven2. This value is automatically selected based on the chosen template.

Target Groups

When a Staging Repository is "closed" and is made available to users and developers involved in the testing process, the temporary Staging Repository is added to one or more Repository Groups. This field defines those groups.

Close Repository Notification Settings

After a developer has deployed a set of related release artifacts, a staging repository is "closed". This means that no further artifacts can be deployed to the same staging repository. A repository would be closed when a developer is satisfied that a collection of staged artifacts is ready to be certified by a manager or a quality assurance resource. In this setting, it is possible to define email addresses and roles which should be notified of a staging repository being closed. A notification email will be sent to all specified email addresses, as well as all Nexus users in the specified roles, informing that a staging repository has been closed. It is also possible to select that the creator of the staging repository receives this notification.

Promote Repository Notification Settings

Once a closed staging repository has been certified by whoever is responsible for testing and checking a staged release, it can then be promoted (published) or dropped (discarded). In this setting, it is possible to define email addresses and Nexus security roles which should be notified of a staging repository being promoted. A notification email will be sent to all specified email addresses, as well as all Nexus users in the specified roles, informing that a staging repository has been promoted. It is also possible to select that the creator of the staging repository receives this notification.

Drop Repository Notification Settings

In this setting, it is possible define email addresses and roles which should be notified of a staging repository being dropped. A notification email will be sent to all specified email addresses, as well as all Nexus users in the specified roles, informing that a staging repository has been dropped. It is also possible to select that the creator of the staging repository receives this notification.

Close Repository Staging Rulesets

This defines the rulesets which will be applied to a staging repository before it can be closed. If the staging repository does not pass the rules defined in the specified rulesets, you will be unable to close it. For more information about rulesets, see Section [10.5](#).

Promote Repository Staging Rulesets

This defines the rulesets which will be applied to a staging repository on promotion. If the staging repository does not pass the rules defined in the specified rulesets, the promotion will fail with an error message supplied by the failing rule. For more information about rulesets, see Section [10.5](#).

10.2.3 Configuring Build Promotion Profiles

A Build Promotion profile is used when you need to add an additional step between initial staging and final release. To add a new Build Promotion profile, open the Staging Profiles link from the Nexus menu and click on Add... to display the drop-down menu shown in Figure 10.6. Select Build Promotion Profile from this drop-down to create a new Build Promotion Profile.

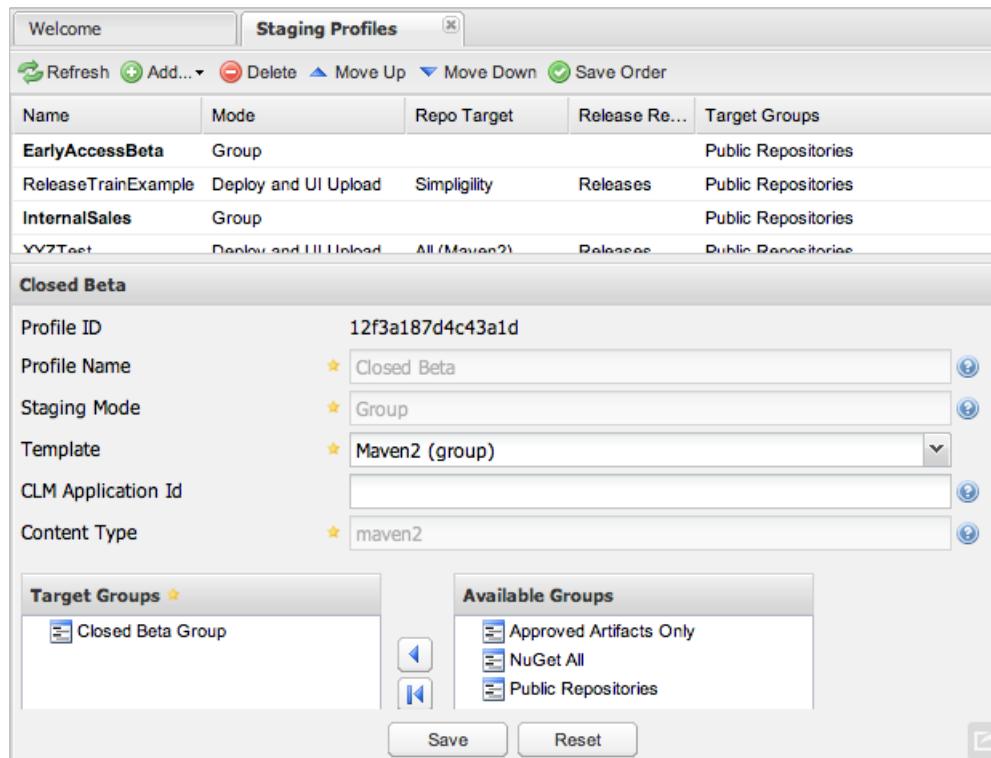


Figure 10.6: Multi-level Staging and Build Promotion

After creating a new Build Promotion profile, you will see the form shown in Figure 10.7. This form contains the following configuration fields:

Profile Name

This is the name for the Build Promotion profile which will be displayed in the promotion dialog and be associated with repositories created from this promotion profile.

Template

This is the template for repositories generated by this Build Promotion profile. The default value for this field is "Maven2 (group)".

Target Groups

This is the most important configuration field for a Build Promotion profile. It controls the group that promoted artifacts will be made available through. Artifacts can be made available through one or more groups.

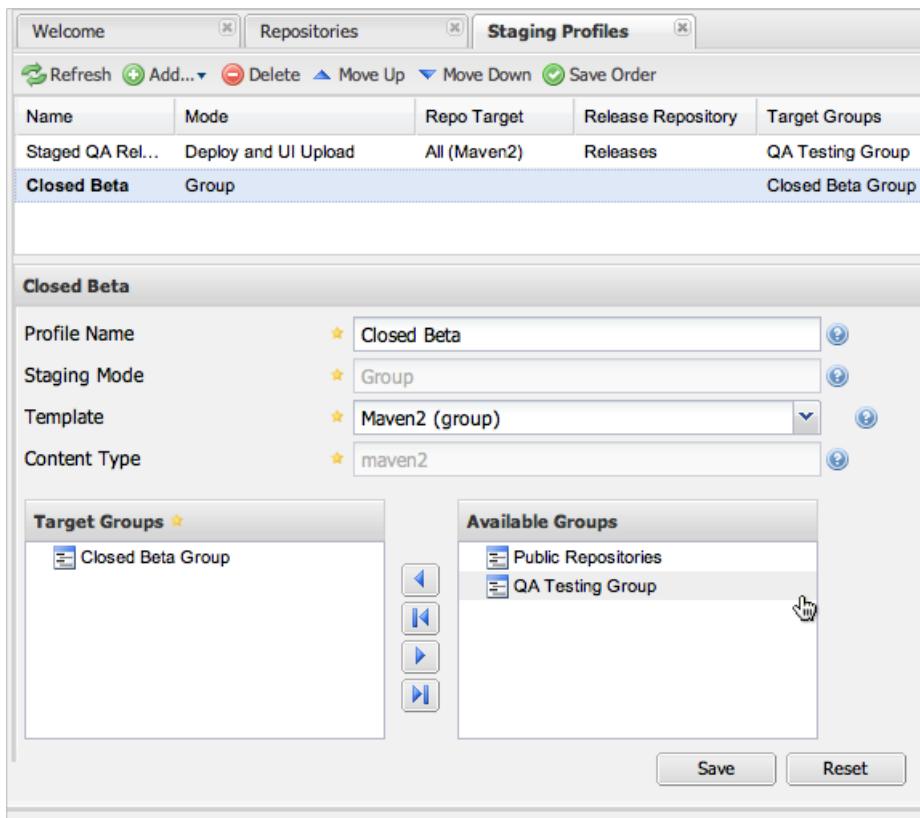


Figure 10.7: Configuring a Build Promotion Profile

10.2.4 Staging Related Security Setup

Staging Suite is controlled by three roles:

- Staging: Deployer
- Staging: Promoter
- Staging: Repositories

These roles are available as general *admin* roles that apply to all staging profiles with the respective access. When you create a new staging profile, Nexus will create new roles that grant permissions specific to that staging profile. If you created the staging profile named *Test*, Nexus created the three new and profile specific roles:

Staging: Repositories (Test)

This role grants a user read and view access to the staging repositories created by the *Test* staging profile.

Staging: Deployer (Test)

This role grants all of the privileges from the Staging: Repositories role and in addition grants the user permission to deploy artifacts, close and drop any staging repository created by the *Test* staging profile.

Staging: Promoter (Test)

This role grants the user the right to promote staging repositories created by the *Test* staging profile.

To perform a staged deployment, the user deploying the artifact must have the "Staging: Deployer (admin)" role or the "Staging: Deployer" role for a specific Staging Profile.

To configure the deployment user with the appropriate staging role, click on Users under the Security menu in the Nexus menu. Once you see the Users panel , click on the deployment user to edit this user's roles. Click on the Add button in the Role Management section of the Config tab visible in Figure 10.8 for the user to be able to add new roles to the user.

The screenshot shows the Nexus User Management interface. At the top, there is a header bar with buttons for Refresh, Add..., Delete, All Configured Users, and a search field. Below the header is a table listing users:

User ID	Realm	First Name	Last Name	Email	Roles
admin	default	Administrator		changeme@yourcompany.com	Nexus...
deployment	default	Deployment	User	changeme1@yourcompany.com	Repo:...
anonymous	default	Nexus	Anonymous User	changeme2@yourcompany.com	UI: U...

Below the table, a section titled "deployment" is expanded, showing configuration details for this user:

User ID	deployment
First Name	Deployment
Last Name	User
Email	changeme1@yourcompany.com
Status	Active

Underneath the configuration section is a "Role Management" panel. It contains a list of available roles:

- Nexus Deployment Role
- Repo: All Repositories (Full Control)
- Staging: Deployer (admin)

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

Figure 10.8: Adding a Role to a User

Use the Filter section with the keyword Staging and press the Apply Filter button to see all available staging related roles as displayed in Figure 10.8.

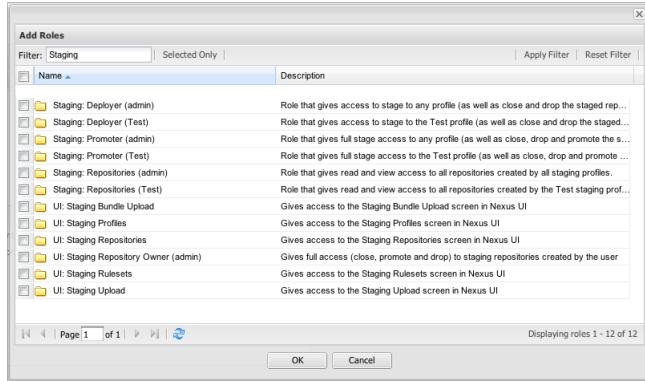


Figure 10.9: Available Roles for Staging with a Test Staging Profile

You should see the "Staging: Deployer (admin)" role listed as well as the *Test* staging profile specific role, the promoter and repositories ones for *admin* and *Test* and a few staging user interface related roles. These roles are required if interaction with the staging suite in the Nexus user interface is desired and allow you to control the details about this access. If you need to add a specific permission to activate a single Staging Profile, you would select that specific role.

Once the deployment user has the "Staging: Deployer (admin)" role, you can then use this user to deploy to the staging URL and trigger any staging profile. Without this permission, the deployment user would not be able to publish a staged artifact.

In a similar fashion you can assign the promoter role to users.

In addition to the roles created a number of specific privileges is available to further customize the access to the staging suite:

Staging Profiles

allows control of create, read, delete and update operations on staging propfiles.

Staging Repository: test-001

separate privileges for each staging repository allowing create, read, update and delete operations are generated automatically.

Staging: All Profiles, Owner All Profiles and Profile xyz

these staging profile specific privileges can be granted for drop, promote, read and finish operations.

Staging: Rule Set and Staging: Rule Types

control access to staging rules and rule types

Staging: Upload

controls access to the manual staging upload user interface

Staging: Repositories, Promote Repository, Profile Ordering, Close Staging and others

a number of application user interface specific privileges allow fine grained control over access in the user interface

10.2.5 Using Repository Targets for Staging

The Staging Suite intercepts deployments to Nexus using Repository Targets as documented in Section 6.9 when using implicit matching as a profile selection strategy based on the artifacts path in the repository.

For example, if you wanted to intercept all deployments to the com.sonatype.sample groupId, you would create a repository target with a pattern with a regular expression of `^/com/sonatype/sample/.*` and use that repository target in your Staging Profile configuration.

10.3 Configuring your Project for Deployment

Once Nexus is configured to receive artifacts in the staging suite as documented in Section 10.2, you will have to update your project build configuration to deploy to the staging suite.

The preferred way to do this is to take advantage of the features provided by the Nexus Staging Maven Plugin or the Nexus Staging Ant Tasks as documented in Section 10.3.1 and Section 10.3.2.

If you need to continue to use the Maven Deploy Plugin you can read about using it with the Nexus Staging Suite in Section 10.3.3.

With all tools you can use the manual upload of your artifacts documented in Section 10.3.4.

10.3.1 Deployment with the Nexus Staging Maven Plugin

The Nexus Staging Maven Plugin is a Nexus specific and more powerful replacement for the Maven Deploy Plugin with a number of features specifically geared towards usage with the Nexus Staging Suite. The simplest usage can be configured by adding it to the project build plugins section as an extension:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.sonatype.plugins</groupId>
      <artifactId>nexus-staging-maven-plugin</artifactId>
      <version>1.4.4</version>
      <extensions>true</extensions>
      <configuration>
        <serverId>local-nexus</serverId>
        <nexusUrl>http://localhost:8081/nexus/</nexusUrl>
      </configuration>
    </plugin>
```

Note

It is important to use a version of the plugin that is compatible with your Nexus server. Version 1.2 is compatible with Nexus 2.3, Version 1.4.4 is compatible with Nexus 2.4. The latest version of the plugin available is always compatible with the latest available version of Nexus.

Following Maven best practices the version should be pulled out into a pluginManagement section in a company POM or parent POM.

This configuration works only in Maven 3 and automatically replaces the deploy goal invocation of the Maven Deploy Plugin in the deploy Maven lifecycle phase with the deploy goal invocation of the Nexus Staging Maven Plugin.

The minimal required configuration parameters for the Nexus Staging Maven Plugin are:

serverId

the id of the server element in `settings.xml` from which the user credentials for accessing Nexus should be retrieved

nexusUrl

the base URL at which the Nexus server to be used for staging is available

With this configuration the Nexus Staging Maven Plugin will stage the artifacts locally and connect to Nexus. Nexus will try to determine the appropriate Staging Profile by matching the artifact path with any repository targets configured with staging profiles with an activated implicit profile selection strategy.

If an appropriate staging profile is found a staging repository is created on the fly and the artifacts are deployed into it. If no profile is found the upload will fail.

To successfully deploy to your Nexus instance, you will need to update your Maven Settings with the credentials for the deployment user. These credentials are stored in the Maven Settings file in `~/.m2/settings.xml`.

To add these credentials, add the following element to the servers element in your `~/.m2/settings.xml` file as shown in [Listing deployment credentials in Maven Settings](#).

Listing deployment credentials in Maven Settings

```
<settings>
  ...
  <servers>
    ...
    <server>
      <id>nexus</id>
      <username>deployment</username>
      <password>deployment123</password>
    </server>
  </servers>
  ...
</settings>
```

Note that the server identifier listed in [Listing deployment credentials in Maven Settings](#) should match the serverId parameter you are passing to the Nexus Staging Maven Plugin and in the example contains the default password for the Nexus deployment user - `deployment123`. You should change this password to match the deployment password for your Nexus installation.

If more control is desired over when the plugins deploy goal is activated or if Maven 2 is used, you have to explicitly deactivate the Maven Deploy Plugin and replace the Maven Deploy Plugin invocation with the Nexus Staging Maven Plugin like visible in [Usage of Nexus Staging Maven Plugin for Maven 2](#).

Usage of Nexus Staging Maven Plugin for Maven 2

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-deploy-plugin</artifactId>
      <configuration>
        <skip>true</skip>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```
<plugin>
  <groupId>org.sonatype.plugins</groupId>
  <artifactId>nexus-staging-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>default-deploy</id>
      <phase>deploy</phase>
      <goals>
        <goal>deploy</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <serverId>local-nexus</serverId>
    <nexusUrl>http://localhost:8081/nexus/</nexusUrl>
    <!-- explicit matching using the staging profile id -->
    <stagingProfileId>129341e09f2ee275</stagingProfileId>
  </configuration>
</plugin>
...
...
```

The implicit matching relies on the setup of repository targets as well as the correct order of staging profiles and is therefore an error prone approach when many staging profiles are in use.

The preferred way to work in this scenario is to change the profile selection strategy on all staging profiles to explicit only and pass the staging profile id to the Nexus Staging Maven Plugin using the `stagingProfileId` configuration parameter as documented above. A full example pom.xml for deployment of snapshot as well as release builds with the Nexus Staging Maven Plugin using explicit matching for the staging profile and locally staged builds and atomic uploads is available in [Full example pom.xml for Nexus Staging Maven Plugin usage](#).

Full example pom.xml for Nexus Staging Maven Plugin usage

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.sonatype.training.nxs301</groupId>
  <artifactId>explicit-staging-example</artifactId>
  <version>1.0.0</version>

  <distributionManagement>
    <snapshotRepository>
      <id>nexus-snapshots</id>
      <url>http://localhost:8081/nexus/content/repositories/snapshots</url>
    </snapshotRepository>
  </distributionManagement>
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.sonatype.plugins</groupId>
      <artifactId>nexus-staging-maven-plugin</artifactId>
      <version>1.2</version>
      <extensions>true</extensions>
      <configuration>
        <serverId>nexus-releases</serverId>
        <nexusUrl>http://localhost:8081/nexus/</nexusUrl>
        <!-- update this to the correct id! -->
        <stagingProfileId>1296f79efe04a4d0</stagingProfileId>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

In order to deploy project artifacts to Nexus with the above setup you would invoke a build with

```
mvn clean deploy
```

The build will locally stage the artifacts for deployment in `target/nexus-staging`, on the console and create a closed staging repository in Nexus holding the build artifacts. This execution of the `deploy` goal of the Nexus Staging Maven Plugin performs the following actions:

- the artifacts are staged locally,
- a staging profile is selected either implicitly or explicitly
- a staging repository is either created on the fly if needed or just selected
- an atomic upload to the staging repository is performed
- and the staging repository is closed (or dropped if upload fails)

The log of a successful deployment would look similar to this:

```
[INFO] --- nexus-staging-maven-plugin:1.1.1:deploy (injected-nexus-deploy) ---
@ staging-example ---
[INFO] Using server credentials with ID="nexus-releases" from Maven settings.
[INFO] Preparing staging against Nexus on URL http://localhost:8081/nexus/
```

```
[INFO] * Remote Nexus reported itself as version 2.2.1 and edition " ←
Professional"
[INFO] * Using staging profile ID "12a1656609231352" (matched by Nexus).
[INFO] Staging locally (stagingDirectory=
"/Users/manfred/dev/explicit-staging-example/target/nexus-staging/12 ←
a1656609231352")...
Uploading: file: ... explicit-staging-example-1.0.0.jar
Uploaded: file: ... explicit-staging-example-1.0.0.jar (4 KB at 1051.1 KB/ ←
sec)
Uploading: file: ... explicit-staging-example-1.0.0.pom
Uploaded: file: ... explicit-staging-example-1.0.0.pom (4 KB at 656.2 KB/ ←
sec)
Downloading: file: ...maven-metadata.xml
Uploading: file: ...maven-metadata.xml
Uploaded: file: ... maven-metadata.xml (322 B at 157.2 KB/sec)
[INFO] Staging remotely...
[INFO] Uploading locally staged directory: 12a1656609231352
[INFO] Performing staging against Nexus on URL http://localhost:8081/nexus ←
/
[INFO] * Remote Nexus reported itself as version 2.2.1 and edition " ←
Professional"
[INFO] * Created staging repository with ID "test-002",
applied tags: {javaVersion=1.6.0_37, localUsername=manfred}
[INFO] * Uploading locally staged artifacts to:
http://localhost:8081/nexus/service/local/staging/deployByRepositoryId/ ←
test-002
[INFO] * Upload of locally staged artifacts done.
[INFO] * Closing staging repository with ID "test-002".
[INFO] Finished staging against Nexus with success.
```

Failures are accompanied by error reports that reveal further details:

```
[ERROR] Error while trying to close staging repository with ID "test-003".
[ERROR]
[ERROR] Nexus Staging Rules Failure Report
[ERROR] =====
[ERROR]
[ERROR] Repository "Test-003 (u:admin, a:127.0.0.1)" (id=n/a) failures
[ERROR]   Rule "RepositoryWritePolicy" failures
[ERROR]     * Artifact updating: Repository ='releases:Releases' does
not allow updating
artifact='/com/sonatype/training/nexus/explicit-staging-example/t1.0.0/ ←
staging-example-1.0.0.jar'
[ERROR]     * Artifact updating: Repository ='releases:Releases' does
not allow updating
artifact='/com/sonatype/training/nexus/explicit-staging-example/1.0.0/ ←
staging-example-1.0.0.pom'
[ERROR]
```

```
[ERROR]
```

If the configuration parameter `skipStagingRepositoryClose` set to `true` is passed to the plugin execution, the remote staging repository will not be closed.

Instead of Nexus creating a staging repository based on the implicit or explicit staging profile selection, you can explicitly configure the staging repository to use by providing the staging repository name as value of the `stagingRepositoryId` configuration property via the plugin configuration or command line invocation.

The identifier of a staging repository can be determined by looking at the name column in the list of staging repositories. The name column used the capitalized id and adds the username and address the staging was deployed from in brackets. For example a name could be `Test-003 (u: admin, a: 127.0.0.1)`. The ID of this staging repository is `test-003`.

Together with skipping the closing of the repository using `skipStagingRepositoryClose` it is possible to get multiple builds to deploy to the same staging repository and therefore have a number of artifacts go through the staging workflow together. An alternative to this approach would be to create an aggregating project that assembles all artifacts together e.g. in an assembly and then use this project for staging.

Finally to override all staging you can define the full repository URL to deploy to with the `deployUrl` configuration parameter e.g.

```
http://localhost:8081/nexus/content/repositories/releases/
```

This would cause any staging to be skipped and a straight upload of the artifacts to the repository to occur.

As part of the configuration section for the plugin you can define tags with arbitrary key and value names. For example you could create a tag with key `localUsername` and a value of the current user picked up from the `USER` environment variable:

```
...
<configuration>
...
<tags>
  <localUsername>${env.USER}</localUsername>
  <javaVersion>${java.version}</javaVersion>
</tags>
...
```

Once artifacts are released these tags are transformed into attributes stored along the artifacts in the release repository and can be accessed via the REST interface and therefore any plugin and user interface integration.

In addition to the above documented configuration options that determine the behaviour of the Nexus Staging Maven Plugin, further configuration can be provided with the following parameters:

altStagingDirectory

defaulting to target/nexus-staging you can set the property to set a different folder for the local staging

description

allows you to provide a description for the staging repository action like close or drop carried out as part of the plugin execution. The description will then be used in any notification just like a description provided in the user interface.

keepStagingRepositoryOnFailure

setting this flag to true will cause the plugin to skip any clean up operations like dropping a staging repository for failed uploads, by default these clean up operations occur

keepStagingRepositoryOnCloseRuleFailure

with the default setting of false the Nexus Staging Maven Plugin will drop the created staging repository if any staging rule violation occurs. If this flag is set to true, it will not drop the staging repository. This allows you to inspect the deployed components in order to figure out why a rule failed causing the staging failure.

skipStagingRepositoryClose

set this to true to turn off the automatic closing of a staging repository after deployment

skipNexusStagingDeployMojo

set to false by default this flag will cause to skip any execution of the *deploy* goal of the plugin when set to true similar to *maven.deploy.skip*

skipStaging

set to false by default this flag will cause to skip any execution of the plugin when set to true

skipRemoteStaging

if this flag is set to true any step related to remote staging will be skipped and only local staging will be performed, the default setting is false

skipLocalStaging

by default set to *true* causes the Nexus Staging Maven Plugin to use local staging, setting this parameter to *false* turns off local staging, which emulates the immediate upload as performed by the Maven Deploy Plugin

With `skipRemoteStaging` set to true, only the local staging happens. This local staging can then be picked up for the remote staging and closing by running the `deploy-staged` goal of the plugin explicitly like this

```
mvn nexus-staging:deploy-staged
```

Besides the default `deploy` goal the Nexus Staging Maven Plugin supports a number of additional goals. By configuring executions of the goals as part of your POM or manually invoking them further automation of a staged release process can be achieved.

deploy-staged

perform full staging deployment workflow for a locally staged project e.g. with the artifacts in `target/nexus-staging`

deploy-staged-repository

perform an upload of a repository from the local filesystem to a staging repository.

close

close the staging repository for current context

drop

drop the staging repository for current context

release

release the staging repository for current context

promote

promote the staging repository for the current context

Closing, dropping and releasing the staging repository using the goals relies on content of a local staging folder .

Promoting additionally needs the build promotion profile name passed in via the `buildPromotionProfileId` configuration parameter.

The `deploy-staged-repository` goal can be used to stage a repository. Typically a local repository is created with an invocation of the deploy similar to

```
mvn deploy -DaltDeploymentRepository=local::default::file://path
```

To deploy this file system repository with the goal, you have to provide the path to this repository with the `repositoryDirectory` parameter as well as `nexusUrl`, `serverId` and `stagingProfileId` . Optionally you can

configure the repository to stage into with *stagingRepositoryId*. This aggregated command is then be run outside any specific Maven project.

While the above goals need the context of a project with configuration for the Nexus Staging Plugin in the POM file, it is possible to execute staging repository related tasks without a project as well. The Nexus Staging Maven Plugin offers remote-control goals to control staging in Nexus:

rc-close

close a specified staging repository

rc-drop

`drop a specified staging repository`

rc-release

release a specified staging repository

rc-promote

promote a specified staging repository

When invoking these goals outside a project context you need to have the Nexus Staging Maven Plugin groupId specified as a pluginGroup in your settings.xml:

```
<pluginGroups>
    <pluginGroup>org.sonatype.plugins</pluginGroup>
</pluginGroups>
```

In addition you need to specify all parameters on the command line as properties passed in via `-Dkey=value`.

At a minimum the required parameters `serverId` and `nexusUrl` have to be specified:

```
mvn nexus-staging:rc-close -DserverId=local-nexus -DnexusUrl=http://localhost:8081/nexus
```

Depending on the goal you will have to configure the staging repositories you want to close, drop or release with

-DstagingRepositoryId=repo-001, repo-002

and you can also supply a description like this

-Ddescription="Dropping since QA of issue 123 failed"

For promoting you need to add the required parameter that specifies the build promotion profile identifier:

```
-DbuildPromotionProfileId=12a25eabf8c8b3f2
```

A successful remote control drop would be logged in the command line similar to this

```
-- nexus-staging-maven-plugin:1.2:rc-drop (default-cli) @ standalone-pom --
[INFO] Connecting to Nexus...
[INFO] Using server credentials with ID="nexus-releases" from Maven    ↵
      settings.
[INFO] RC-Dropping staging repository with IDs=[test-003]
[INFO]   ↵
----- ↵
[INFO] BUILD SUCCESS
[INFO]   ↵
----- ↵
```



Warning

The Nexus Maven Plugin in versions earlier than 2.1.0 had goals to work with staging repositories. These goals have been deprecated in favour of the remote control goals of the Nexus Staging Maven Plugin.

10.3.2 Deployment with the Nexus Staging Ant Tasks

The Nexus Staging Ant Tasks provide equivalent features to the Nexus Staging Maven Plugin for Apache Ant users covering all use cases for interacting with the Nexus Staging Suite.

To use the Ant tasks in your Ant build file you need to download the complete jar with the included dependencies. You can find it at the Central Repository. Simply search for *nexus-staging-ant-tasks* and download the jar file with the *uber* classifier e.g. *nexus-staging-ant-tasks-1.0-uber.jar*.

After downloading, put the jar file somewhere in your project or in your system so you can add it to the classpath in your build file with a task definition. In the following example the jar file is placed in a *tasks* folder within the project.

```
<taskdef uri="antlib:org.sonatype.nexus.ant.staging"
        resource="org/sonatype/nexus/ant/staging/antlib.xml">
```

```
<classpath>
  <fileset dir="tasks" includes="nexus-staging-ant-tasks-*uber.jar" />
</classpath>
</taskdef>
```

The deployment related information for your project is captured in a `nexusStagingInfo` section in your build file that contains all the necessary configuration.

```
<staging:nexusStagingInfo id="target-nexus"
    stagingDirectory="target/local-staging">
  <staging:projectInfo groupId="org.sonatype.nexus.ant"
    artifactId="nexus-staging-ant-tasks"
    version="1.0" />
  <staging:connectionInfo
    baseUrl="http://localhost:8081/nexus">
    <staging:authentication
      username="deployment"
      password="deployment123" />
  </staging:connectionInfo>
</staging:nexusStagingInfo>
```

nexusStagingInfo:id

the identifier that allows you to reference the staging information in the Ant build file

stagingInfo:stagingDirectory

the local staging directory, a place where local staging will happen. Ensure that this directory is cleaned up by "clean" tasks (or alike, if any).

projectInfo

the project information targetting a staging profile. This can be done explicitly with the stagingProfileId or implicitly with groupId, artifactId and version. stagingRepositoryId can also be part of projectInfo identifying a staging repository for interaction

connectionInfo:baseUrl

the base URL of the Nexus server you want to deploy to and interact with

If necessary the connection Info can have a nested proxy section

```
<staging:proxy
  host="proxy.mycorp.com"
  port="8080">
  <staging:authentication
    username="proxyUser"
    password="proxySecret" />
</staging:proxy>
```

With the above setup you are ready to add a deploy target to your build file that stages the artifacts locally as well as remotely and closes the staging repository.

```
<target name="deploy" description="Deploy: Local and Remote Staging">

    <staging:stageLocally>
        <staging:nexusStagingInfo
            refid="target-nexus" />
        <fileset dir="target/local-repo"
            includes="**/*.*" />
    </staging:stageLocally>

    <staging:stageRemotely>
        <staging:nexusStagingInfo
            refid="target-nexus" />
    </staging:stageRemotely>

</target>
```

Similarly you can create a target that releases the staged artifacts by adding the releaseStagingRepository task to the end of the target:

```
<staging:releaseStagingRepository>
    <staging:nexusStagingInfo
        refid="target-nexus" />
</staging:releaseStagingRepository>
```

The stageLocally task takes a fileset as configuration. The stageRemotely task has additional configuration options

keepStagingRepositoryOnFailure

set to true this causes the remote staging repository to be kept rather than deleted in case of a failed upload, default setting is false

skipStagingRepositoryClose

by default a staging repository is automatically closed, setting this parameter to true will cause the staging repository to remain open

In addition to the tasks for local and remote staging the Nexus Staging Ant Tasks includes tasks for closing, dropping, releasing and promoting a staging repository:

- closeStagingRepository

- dropStagingRepository
- releaseStagingRepository
- promoteStagingRepository

All these tasks take the context information from the local staging directory or from the optional parameter `stagingRepositoryId`. The task to promote a repository has the additional, mandatory attribute `buildPromotionProfileId` to specify the build promotion profile to promote.

10.3.3 Deployment with the Maven Deploy Plugin

When using the Maven Deploy Plugin with the Nexus Staging Suite, you rely on implicit matching of the artifacts against a staging profile based on a repository target definition.

To deploy a staged release, a developer needs to deploy to the staging URL. To configure a project to deploy to the Staging URL, add the a `distributionManagement` element to your project's POM.

Listing the Staging URL in distributionManagement

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
...
<distributionManagement>
  <repository>
    <id>nexus</id>
    <name>Nexus Staging Repo</name>
    <url>http://localhost:8081/nexus/service/local/staging/deploy/maven2 ←
        /</url>
  </repository>
</distributionManagement>
...
</project>
```

This configuration element, `distributionManagement`, defines the repository to which our deployment will be made. It references the Staging Suite's Staging URL: <http://localhost:8081/nexus/service/local/-staging/deploy/maven2>

This URL acts as a something of a virtual repository to be published to. If an artifact being published matches one of the Repository Targets in a Staging Profile, that Staging Profile is "activated" and a temporary Staging Repository is created.

Once the sample project's distributionManagement has been set to point at the Nexus Staging URL and your deployment credentials are updated in your `~/.m2/settings.xml` file, you can deploy to the Staging URL. To do this, run `mvn deploy`

```
$ mvn deploy
[INFO] Scanning for projects...
[INFO]   ↵
-----
[INFO] Building staging-test
[INFO]   task-segment: [deploy]
[INFO]   ↵
-----
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory: /private/tmp/staging-test/target/ ↵
      surefire-reports

...
[INFO] [jar:jar]
[INFO] [install:install]
[INFO] Installing /private/tmp/staging-test/target/staging-test-1.0.jar to ↵
  \
~/m2/repository/com/sonatype/sample/staging-test/1.0/staging-test-1.0.jar
[INFO] [deploy:deploy]
altDeploymentRepository = null
Uploading: http://localhost:8081/nexus/service/local/staging/deploy/maven2 ↵
  /
com/sonatype/sample/staging-test/1.0/staging-test-1.0.jar
2K uploaded
[INFO] Uploading project information for staging-test 1.0
[INFO] Retrieving previous metadata from nexus
[INFO] repository metadata for: 'artifact com.sonatype.sample:staging-test ↵
  ,
could not be found on repository: nexus, so will be created
[INFO] Uploading repository metadata for: 'artifact com.sonatype.sample: ↵
  staging-test'
[INFO]   ↵
-----
[INFO] BUILD SUCCESSFUL
```

If the Staging Suite is configured correctly, any deployment to the staging URL matching in a repository target configured for a staging profile should be intercepted by the Staging Suite and placed in a temporary staging repository. Deployment with the Maven Deploy Plugin will not automatically close the staging repository. Closing the staging repository has to be done with the Nexus user interface or the Nexus Staging Maven Plugin. Once this repository has been closed, it will be made available in the Target Group you selected when you configured the Staging Profile.

10.3.4 Manually Uploading a Staged Deployment in Nexus

You can also upload a staged deployment via the Nexus interface. To upload a staged deployment, select Staging Upload from the Nexus menu. Clicking Staging Upload will show the panel shown in Figure 10.10.

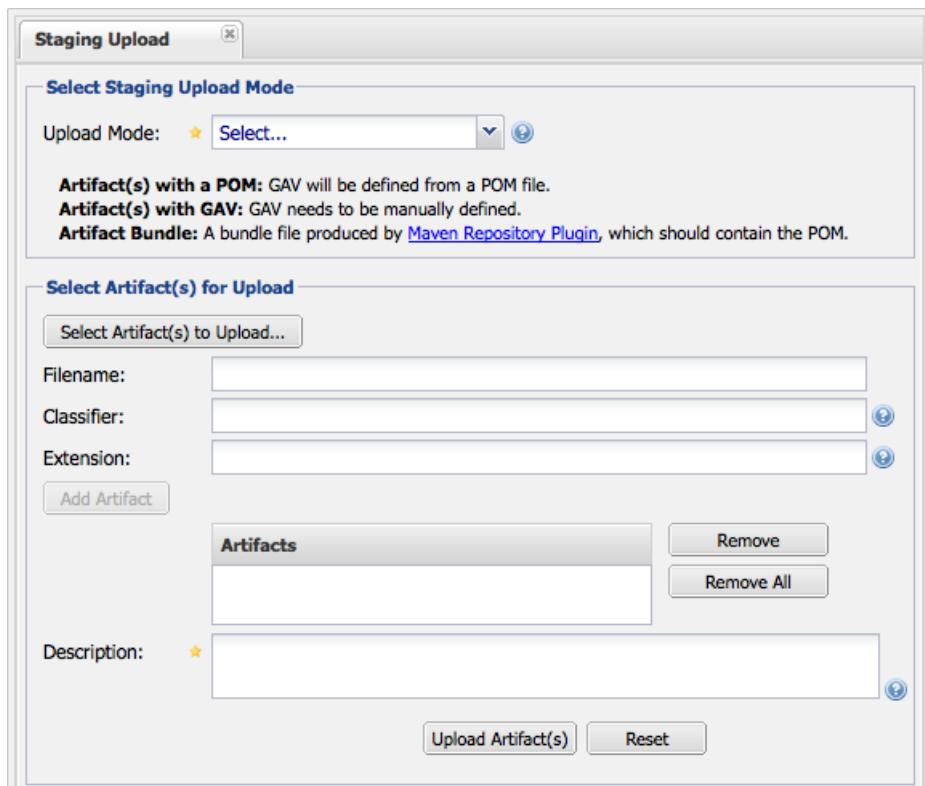


Figure 10.10: Uploading a Staged Deployment in Nexus

To upload an artifact, click on Select Artifact(s) for Upload... and select an artifacts from the filesystem to upload. Once you have selected an artifact, you can modify the classifier and the extension before clicking on the Add Artifact button. Repeat this process to upload multiple artifacts for the same Group, Artifact and Version (GAV) coordinates like a jar, the pom and maybe a sources and javadoc jar in addition. Once you have added all the artifacts, you can then configure the source of the Group, Artifact, Version (GAV) parameters.

If the artifact you are uploading is a JAR file that was created by Maven it will already have POM information embedded in it, but if you are uploading a JAR from a vendor you will likely need to set the Group Identifier, Artifact Identifier, and Version manually. To do this, select GAV Parameters from the GAV Definition drop-down at the top of this form. Selecting GAV Parameters will expose a set of form fields which will let you set the Group, Artifact, Version, and Packaging of the artifacts being uploaded. If you would prefer to set the Group, Artifact, and Version from a POM file which was associated with the uploaded artifact, select From POM in the GAV Definition drop-down. Selecting From POM in this drop-down will expose a button labelled "Select POM to Upload". Once a POM file has been selected for upload, the name of the POM file will be displayed in the form field below this button.

The Staging Upload panel supports multiple artifacts with the same Group, Artifact, and Version identifiers. For example, if you need to upload multiple artifacts with different classifiers, you may do so by clicking on Select Artifact(s) for Upload and Add Artifact multiple times. This interface also accepts an Artifact Bundle which is a JAR that contains more than one artifact, which is documented in more detail in Chapter 18.

Once a staging artifact upload has been completely configured, click on Upload Artifact(s) button to begin the upload process. Nexus will upload the artifacts to the Staging URL which will trigger any staging profiles that are activated by the upload by explicitly matching using the repository targets configured with the staging profiles. If a staging profile is activated, a new staging repository will be created and can be managed using the procedures outlined in Section 10.4.

10.4 Managing Staging Repositories in Nexus

With a staging profile configured and a deployment completed as outlined in Section 10.2 and Section 10.3, you will have an automatically generated Staging Repository. All list of all staging repositories can be accessed by selecting the Staging Repositories item in the Build Promotion menu and is displayed in Figure 10.11

Staging Repositories					
	Repository	Profile	Status	Updated	Description
<input checked="" type="checkbox"/>	nxs301-001	NXS301	closed	2013-Mar-27 10:10:14	Implicitly created (aut)

Figure 10.11: Staging Repositories List Panel

The header of this view provides buttons to Close, Promote, Release or Drop the staging repository currently selected in the list below. The Refresh button can be used to force a reload of repositories. The Filter by profile drop down allows you to select one or multiple staging profiles, from which the repositories in the list were created. The list of repositories itself displays a number of columns with details for each repository. Further columns can be added by pressing on the drop down triangle beside the currently selected column. Sorting by a single column in Ascending or Descending order can be set from the same drop down as the column addition.

NOTE

When triggering a transition for a staging repository from e.g. the open state to a the closed state a background task performs all the necessary operations. Since these are potentially longer running the user interface is not immediately updated and displays a in progress icon. You are required to press Refresh to get the latest state of all repositories.

By default the following columns are displayed:

Checkbox

a checkbox to allow operations on multiple repositories

Status Icon

an icon symbolizing the status of the staging repository

Repository

the name of the staging repository

Profile

the name of the staging profile, that was used to create the staging repository

Status

status of the repository

Updated

date and time of the last update

Description

textual description of the repository

Additional columns are:

Release To

target repository for the components in the staging repository after release

Promoted To

the build promotion profile, to which a staging repository was optionally promoted to

Owner

the username of the creator of the staging repository

Created

date and time of the creation of the staging repository

User Agent

user agent string sent by the tool used for the deployment e.g. Apache-Maven/3.0.5...

Tip

You can also access staging repositories in the list of repositories available in the **Repositories** panel available via the **Views / Repositories** as a Nexus managed repository.

In the following sections, you will walk through the process of managing staging repositories. Once you have deployed a set of related components, you must close the repository moving it from an "Open" to a "Closed" state unless the deployment tool automatically closed the staging repository.

A repository in the "Closed" state, is added to a Repository Group and is made available for testing purposes or other inspection and can no longer receive additional components in it.

When the component examination is complete, you can either "Promote", "Release" or "Drop" the closed repository.

If the repository is dropped, the repository is discarded and removed from the Repository Group and the components are moved to the Trash.

If the repository is promoted, it is assigned to a build promotion profile for further staging activities.

If the repository is released, its components are moved to the target repository configured in the staging profile.

Note

A scheduled task documented in Section 6.5 can be used to clean up inactive staging repositories automatically.

Selecting a staging repository in the list displays further details about the repository in the Summary, Activity and Content tabs below the list. An example for an open repository is displayed in Figure 10.12.

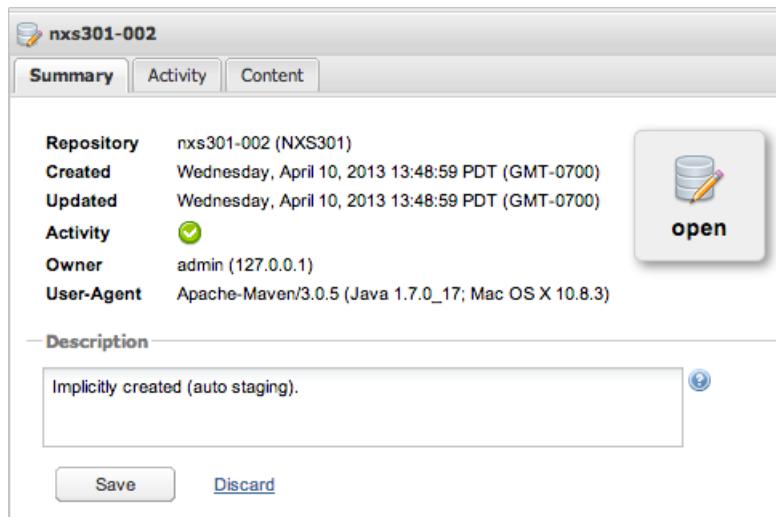


Figure 10.12: List of Activities Performed on a Promoted Staging Repository

The Summary tab displays a number of properties of the staging repository and allows you to edit the Description. The properties include the name of the repository, creation and update time and date stamps, an activity indicator, the owner and originating IP number of the deployment as well as the user agent string sent by the deployment. All staging operations have a default description that is used if the input field is left blank.

The Activity tab shows all the activities that occurred on a specific staging repository. An example for a promoted repository is displayed in Figure 10.13. The activities are separated per activity and list all

events that occurred in an activity. Selecting an event displays further details about the event on the right side of the tab.



Figure 10.13: Details of an Open Staging Repository as Displayed under the List of Staging Repositories

The Content tab displays a repository browser view of the staging repository content and allows you to filter and display the components in the tree view. Selecting a specific component triggers the display of further panels with further information about the component, in the same manner as other repository browser views. The tabs include Maven and Artifact information and others.

A Members tab is additionally shown for build promotion profile. It displays the source repositories and build promotion profiles from which this current build promotion profile was created.

10.4.1 Closing an Open Repository

Once you deploy a component that triggers a staging profile, Nexus Staging Suite will create a repository that contains the components you deployed. A separate staging repository is created for every combination of User ID, IP Address, and User Agent. This means that you can perform more than one deployment to a single Staging Repository as long as you perform the deployment from the same IP, with the same deployment user, and the same installation of Maven.

You can perform multiple deployments to an open staging repository. Depending on the deployment tool and your configuration the staging repository might be automatically closed during deployment or left open until manually closed.

Once you are ready to start testing the staging repository content, you will need to transition the repository from the open state to the closed state. This will close the staging repository to more deployments.

To close a repository, select the open staging repository in the list and by clicking the checkbox in the list or anywhere else in the row. For a open repository the **Close** and the **Drop** buttons above the table will be activated. Pressing the **Close** button will bring up the dialog for a staging deployer to describe the contents of the staging repository and confirm . This description field can be used to pass essential information to the person that needs to test a deployment.

In Figure 10.14, the description field is used to describe the release for the user that needs to certify and promote a release.

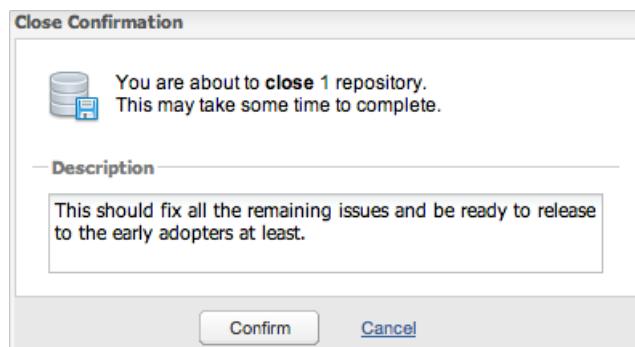


Figure 10.14: Confirmation and Description Dialog for Closing a Staging Repository

Confirming this state transition will close the repository and add the repository to the repository groups configured in the staging profile. The updated status will be visible in the list of staging repositories after a Refresh, since the transition could take longer depending on the configured staging rules and potential validation against Sonatype CLM.

10.4.2 Using the Staging Repository

Once the staging repository has been closed, it will automatically be added to the repository group(s) that are specified as target groups in the staging profile configuration.

This has the effect of making the staged artifacts available to everyone who is referencing this group. Developers who are referencing this repository group can now test and interact with the staged artifacts as if they were published to a Hosted repository.

While the artifacts are made available in a repository group, the fact that they are held in a temporary staging directory gives the staging user the option of promoting this set of artifacts to a hosted repository. Or alternatively the user can drop this temporary staging repository, if there are problems discovered during the testing and certification process for a release.

Once a staging repository is closed, you can also browse and search the repository in the staging repositories list.

Alternatively to view all staging repositories, click on the Repositories item in the Views.Repositories menu and then select Nexus Managed Repositories as shown in Figure 10.15.

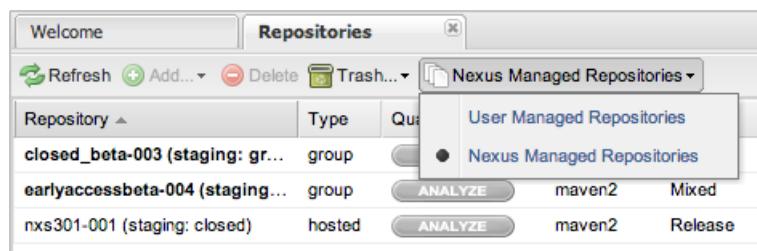


Figure 10.15: Viewing Nexus Managed Repositories

This list allows you to access all Nexus Managed Repositories, just like the User Managed Repositories including browsing the content and accessing detailed information about the components in the repository. In addition to staging repositories, the list included procured repositories as documented in Chapter 9.

10.4.3 Releasing a Staging Repository

When you are finished testing or certifying the contents of a staging repository, you are ready to either release, promote or drop the staging repository. Dropping the staging repository will delete the temporary it from Nexus and remove any reference to this repository from the groups it was associated with. Releasing the staging repository allows you to publish the contents of this temporary repository to a hosted repository. Promoting the repository will move it to a build promotion profile.

You can release a staging repository by pressing **Release**, after selecting a closed staging repository from the staging repositories list. The **Release Confirmation** dialog displayed in Figure 10.16 will allow you to supply a description and configure if the staging repository should be automatically dropped after the components have been released to the hosted repository.

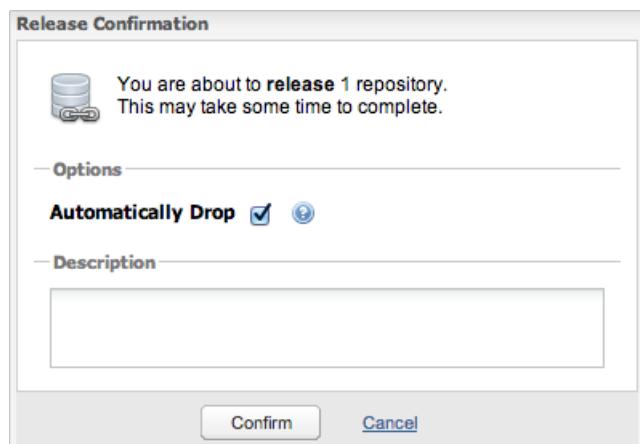


Figure 10.16: Confirmation Dialog for Releasing a Staging Repository

10.4.4 Promoting a Staging Repository

If you have a closed staging repository that you want to promote to a Build Promotion Profile, open the list of Staging Repositories and click the **Promote** button to bring up the **Promote Confirmation** dialog displayed in Figure 10.16. It allows you to select the build promotion profile to which you want to stage the repository to as well as provide a description.

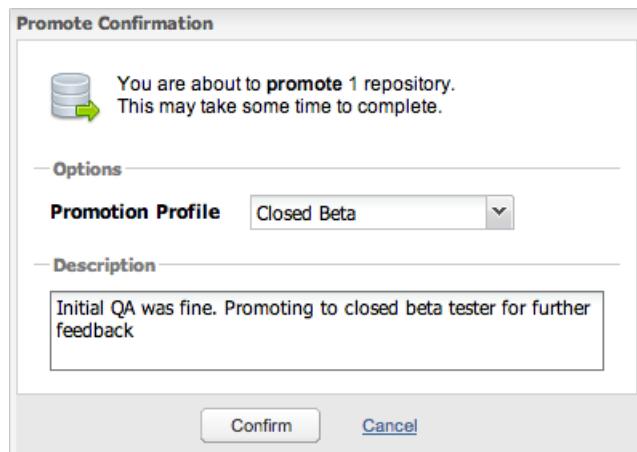


Figure 10.17: Confirmation Dialog for Promoting a Staging Repository

Clicking on the `Promote` button in the dialog will promote the staging repository to a build promotion repository and expose the contents of the selected staging repository through the target group(s) associated with the build promotion profile.

The build promotion repository is accessible in the staging repository list as displayed in Figure 10.18. If you add the column `Promoted To` to the list you will observe that Nexus keeps track of the promotion source. The `Members` tab for a build promotion repository displays the path of a build promotion repository back to a staging repository. One or more staging repositories can be promoted to a single build promotion profile.

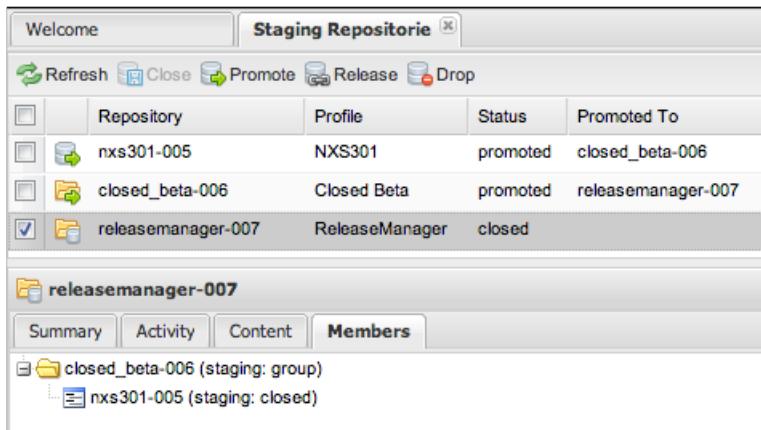


Figure 10.18: A Build Promotion Repository and its Members Panel

10.4.5 Releasing, Promoting, and Dropping Build Promotion Profiles

When you configure a build promotion profile and promote staging repositories to promotion profiles, each build promotion profile creates a repository which contains one or more staging repositories. Just like you can promote the contents of a staging repository to a build promotion profile, you can also promote the contents of a build promotion profile to another build promotion profile. When you do this you can create hierarchies of staging repositories and build promotion profiles which can then be dropped or released together.

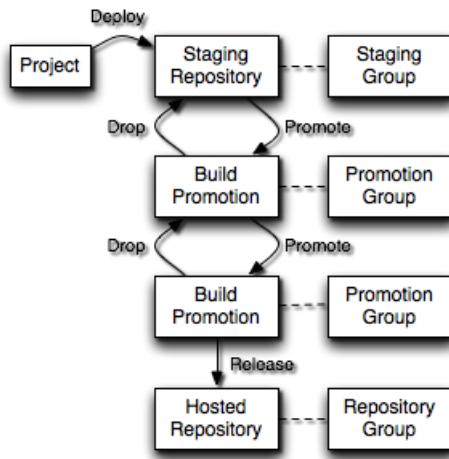


Figure 10.19: Releasing, Promoting, and Dropping Build Promotion Profiles

When you promote a staging repository to a build promotion profile, you make the contents of a staging repository available via a repository group associated with a build promotion profile.

For example, if you staged a few artifacts to a QA staging repository and then subsequently promoted that repository to a Closed Beta build promotion group, the contents of the QA staging repository would initially be made available via a QA repository group. After a build promotion, these artifacts would also be available via a Closed Beta repository group.

You can take it one step further and promote the contents of the Closed Beta Build Promotion profile to yet another build promotion profile. In this way you can have an arbitrary number of intermediate steps between the initial staging deployment and the final release.

If you drop the contents of a build promotion profile, you roll back to the previous state. For example, if you decided to drop the contents of the Closed Beta build promotion group, Nexus will revert the status of the staging repository from promoted to closed, and make the artifacts available via the QA staging repository. The effects of promoting, dropping, and releasing artifacts through a series of Staging Profiles and Build Promotion Profiles is shown in Figure 10.19.

When you perform a release on a build promotion profile, it rolls up to release all its members ultimately reaching a staging repository. Each staging repository is releases its components to the release repository configured in Figure 10.5. Because a build repository can contain one or more promoted staging repositories, this means that releasing a build promotion profile can cause components to be published to more

than one release repository.

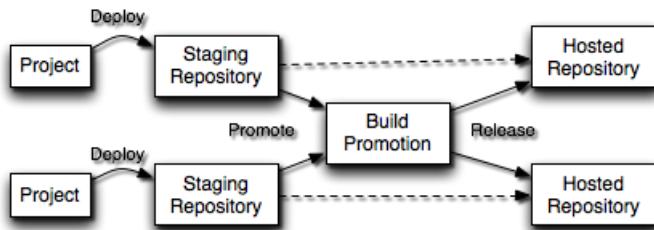


Figure 10.20: Promoting Multiple Repositories to the Same Build Promotion Profile

Build promotion profiles are not directly related to release repositories, only staging profiles are directly associated with target release repositories. Figure 10.20 illustrates this behaviour with two independent staging repositories each configured with a separate release repository. Releasing the build promotion profile causes Nexus to publish each staging repository to a separate hosted repository.

10.4.6 Multi-level Staging and Build Promotion

Nexus also supports multi-level staging and build promotion. With multi-level staging, a staging repository can be tested and then promoted to multiple separate build promotion profiles consecutively and exposed through different repository groups to allow for additional testing and qualification before a final frelease. Figure 10.21 illustrates a potential use for multi-level staging:

Stage

A developer publishes components to a QA staging profile which exposes the staged components in a QA repository group used by an internal quality assurance team for testing.

Promote to Beta

Once the QA team has successfully completed testing, they promote the temporary staging repository to a build promotion profile which will expose the staged components to a limited set of customers who have agreed to act as a beta testers for a new feature.

Release

Once this closed beta testing period is finished, the staged repository is then released and the artifacts it contains are published to a hosted release repository and exposed via the public repository group.

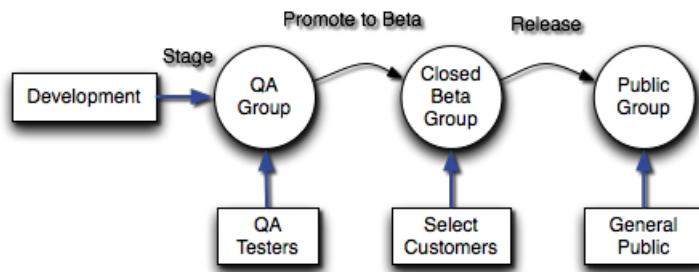


Figure 10.21: Multi-level Staging and Build Promotion

To support this multi-level staging feature, you can configure Build Promotion profiles as detailed in Section 10.2.3. Once you have promoted a Staging Repository to a Build Promotion profile, you can drop, promote, or release the artifacts it contains as detailed in Section 10.2.

10.5 Enforcing Standards for Deployment and Promotion with Rule-sets

Nexus has the ability to define staging rules that must be satisfied to allow successful deployment or before a staging repository can be promoted.

10.5.1 Managing Staging Rulesets

Staging rulesets are customizable groups of rules that are validated against the components in a staging repository, when the repository is closed or promoted. If any rules can not be validated closing or promoting the repository will fail.

A staging repository associated with a staging ruleset configured in the staging profile can not be closed or promoted until all of the rules associated with the rulesets have been satisfied. This allows you to set standards for your own hosted repositories, and it is the mechanism that is used to guarantee the consistency of components stored in the Central Repository.

To create a Staging Ruleset, click on the **Staging Ruleset** item in the **Build Promotion** menu.

This will load the interface shown in Figure 10.22. The Staging Ruleset panel is used to define sets of rules that can be applied to staging profiles.

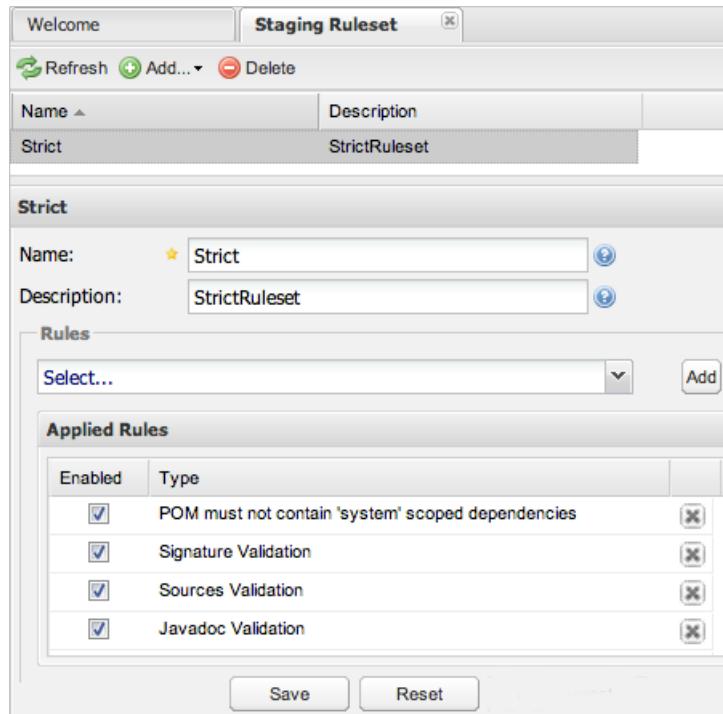


Figure 10.22: Creating a Staging Ruleset

Nexus contains the following rules:

Artifact Uniqueness Validation

This rule checks to see that the component being released, promoted, or staged is unique in a particular Nexus instance.

Checksum Validation

This rule validates that file checksum files are present and correct for the published components.

Javadoc Validation

The Javadoc Validation rule will verify that every project has a component with the javadoc classifier. If you attempt to promote a staging repository which contains components not accompanied by "-javadoc.jar" artifacts, this validation rule will fail.

POM Validation

The Staging POM Validation rule will verify Project URL - project/url, Project Licenses - project/licenses and Project SCM Information - project/scm. Any of these POM elements can not be missing or empty.

POM must not contain *system* scoped dependencies

ensures that no dependency is using the scope system, that allows for a path definition ultimately making the component rely on a specific relative path.

POM must not contain release repository

This rule can ensure that no repository element is defined in the POM. This is important since it potentially would circumvent the usage of the repository manager and could point to other repositories that are not actually available to a user of the component

Signature Validation

The Signature Validation rule verifies that every item in the repository has a valid PGP signature. If you attempt to promote a staging repository which contains artifacts not accompanied by valid PGP signature, this validation will fail.

Sources Validation

The Sources Validation rule will verify that every project has an artifact with the sources classifier. If you attempt to promote a staging repository which contains artifacts not accompanied by "-sources.jar" artifacts, this validation rule will fail.

10.5.2 Defining Rulesets for Promotion

To define a ruleset to be used for closing or promotion, edit the staging profile by selecting it in the staging profile list. Scroll down to the sections Close Repository Staging Rulesets and Promote Repository Staging Rulesets as shown in Figure 10.23 and add the desired available rulesets to the left hand list of activated rulesets for the current staging profile.

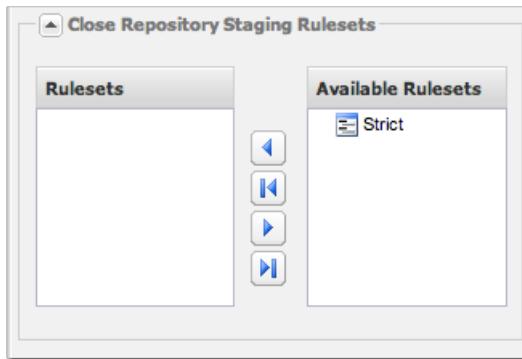


Figure 10.23: Associating a Staging Ruleset with a Staging Profile

The next time you attempt to close or promote a staging repository that was created with this profile, Nexus Professional will check that all of the rules in the associated rulesets are being adhered to.

Chapter 11

Repository Health Check

Repository Health Check is a feature of Nexus that integrates data from [Sonatype Insight](#). Sonatype Insight is suite of separate products that consists of tools to monitor and manage license, quality and security data about artifacts used in your software development life cycle for your Component Lifecycle Management CLM efforts.

Repository Health Check provides access to a limited subset of the available data in Sonatype Insight via the Sonatype Insight Service right in your Nexus server. The Sonatype Insight Service is part of the Central Repository and exposes data about the artifacts there, including license information, security vulnerability data and other statistics like relative usage data and age.

11.1 Analyzing a Repository with Repository Health Check

To perform a Repository Health Check about the artifacts in your Nexus instance, you have to click the Analyze button in the Quality column of the list of Repositories displayed in Figure 11.1. In order to be able to do that you have to be logged in as an administrator. Once the data gathering analysis is completed the Quality column will display the number of security and license issues found.

Repositories						
Repository	Type	Quality	Format	Policy	Repository Status	Repository Path
Apache Snapshots	proxy	UNAVAILABLE	maven2	Snapshot	In Service	https://repository.sonatype.org/content/repositories/snapshots/
Apache Staging	proxy	ANALYZE	maven2	Release	In Service	https://repository.sonatype.org/content/repositories/staging/
Atlassian	proxy	ANALYZE	maven2	Release	In Service	https://repository.sonatype.org/content/repositories/atlassian-releases/
Atlassian User Contrib	proxy	ANALYZE	maven2	Release	In Service	https://repository.sonatype.org/content/repositories/atlassian-user-contrib-releases/
ben-tests	hosted	UNAVAILABLE	maven-site		In Service	https://repository.sonatype.org/content/repositories/ben-tests/
Central Proxy	proxy	2117	19162	maven2	Release	In Service
CentralM1	virtual	UNAVAILABLE	maven1		Release	https://repository.sonatype.org/content/repositories/centralm1/
Codehaus Snapshots	proxy	UNAVAILABLE	maven2	Snapshot	In Service	https://repository.sonatype.org/content/repositories/codehaus-snapshots/
Concierge	proxy	ANALYZE	maven2	Release	In Service - Remo..	https://repository.sonatype.org/content/repositories/concierge-releases/
DDSteps	proxy	ANALYZE	maven2	Release	In Service - Remo..	https://repository.sonatype.org/content/repositories/ddsteps-releases/
Docbkx Snapshots	proxy	UNAVAILABLE	maven2	Snapshot	In Service	https://repository.sonatype.org/content/repositories/docbkx-snapshots/

Figure 11.1: The Repositories List with Different Quality Status Indicators and Result Counts

Hovering your mouse pointer over that value will display the Repository Health Check summary data in a pop up window. A sample window is displayed in Figure 11.2.

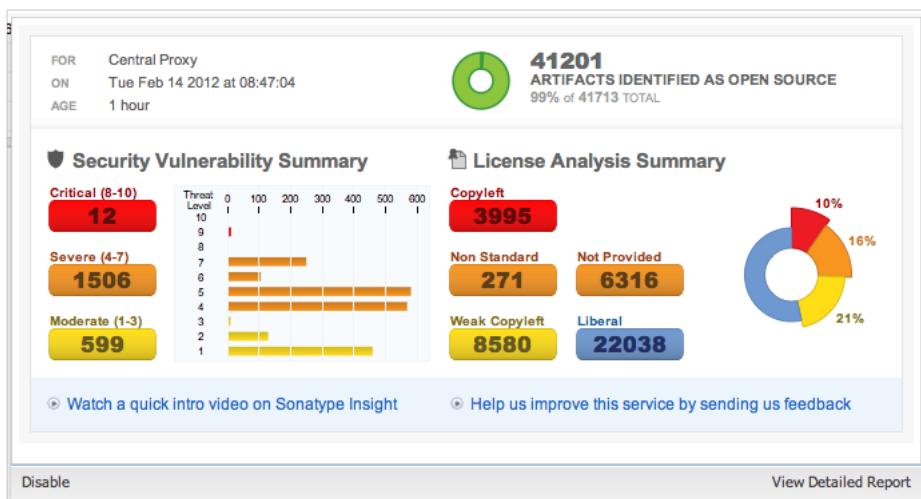


Figure 11.2: A Result Summary Window for a Repository Health Check

At the bottom of the pop up window you find a button to disable the analysis for this repository as well as the button View Detailed Report to access the detailed report. It will show up in another tab in the main area of the Nexus user interface.

11.2 Accessing the Detailed Repository Health Check Report

The detailed report contains the same overview data and charts for security and license information at the top displayed in Figure 11.3 .

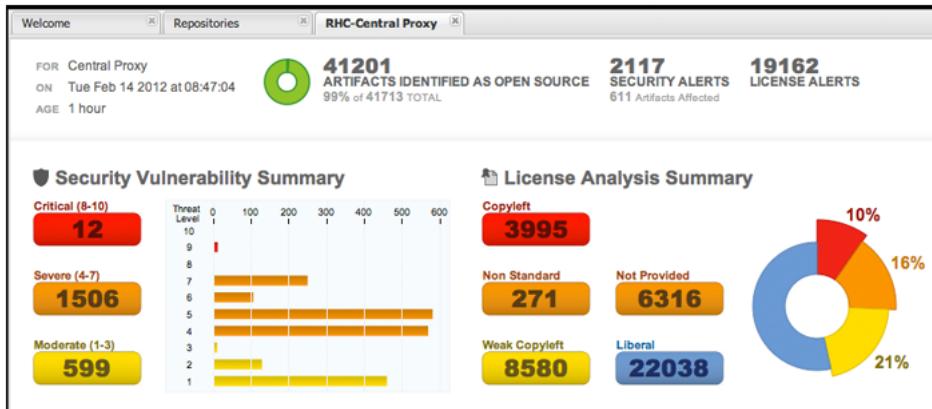


Figure 11.3: Summary of the Detailed Repository Health Check Panel

Below this overview a drop down for Security and License information allows you to toggle between two lists displaying further details. Both lists have a filter for each column at the bottom of the list that allows you to narrow down the number of rows in the table and therefore find specific entries easily.

The security list as visible in Figure 11.4 contains columns for Threat Level, Problem Code and the GAV parameters identifying the affected artifact. The Problem Code column is a link to the security warning referenced and commonly links a specific entry in the [Open Source Vulnerability Database](#) or the [Common Vulnerabilities and Exposures](#) list. Both of these databases have a descriptive text for the vulnerability and further information and reference links.

View By: Vulnerabilities

Threat Level	Problem Code	Group	Artifact	Version
7	CVE-2010-2076	org.apache.cxf	cxf-common-utilities	2.2.4
	CVE-2011-3190	org.apache.tomcat	coyote	6.0.33
	osvdb-24364	struts	struts	1.1-rc1
	osvdb-67294	org.apache.cxf	cxf-common-utilities	2.2
	osvdb-24363	struts	struts	1.1-rc1
	osvdb-67294	org.apache.cxf	cxf-common-utilities	2.2.4
	CVE-2011-3190	org.openl.rules	org.openl.rules.tomcat.lib	5.7.2
	osvdb-74818	org.ow2.jonas.assemblies.profiles	jonas-full	5.3.0-M2
	CVE-2006-1547	struts	struts	1.1-rc1
	osvdb-67294	org.apache.cxf	cxf-common-utilities	2.1.2
	CVE-2010-2076	org.apache.cxf	cxf-common-utilities	2.2
	CVE-2010-2076	org.apache.cxf	cxf-common-utilities	2.1.2
	osvdb-65697	org.apache.axis2	axis2-webapp	1.5.1
	osvdb-67294	org.apache.cxf	cxf-bundle-jaxrs	2.2
	CVE-2006-1546	struts	struts	1.1-rc1
	CVE-2010-2076	org.apache.cxf	cxf-bundle-jaxrs	2.2
	osvdb-65697	org.apache.ws.commons.axiom	axiom-api	1.2.6
	osvdb-65697	org.apache.servicemix	apache-servicemix	4.3.0
	osvdb-74818	org.openl.rules	org.openl.rules.tomcat.lib	5.7.2
	osvdb-74818	org.apache.tomcat	coyote	6.0.32

Figure 11.4: The Security Data in the Detailed Repository Health Check Report

The Threat Level is rated in values used by the vulnerability databases and ranges from 0 for a low threat to 10 for the highest threat. Values from 8-10 are classified as Critical and are displayed as red notification, values from 4 to 7 are classified as Severe and use orange and values from 1 to 3 are classified as Moderate and use yellow as notification color.

The license list as visible in Figure 11.5 shows a derived threat in the Effective License Threat column. The Declared License column details the license information found in pom file. The Observed Licenses in Source columns lists all the licences found in the actual source code of the library in the form of file headers and license files. The next columns for the GAV parameters allow you to identify the artifact. The last column Security Issues displays an indicator for potentially existing security issue for the same artifact.

Licences such as GPL-2.0 or GPL-3.0 are classified as the highest License Threat and labelled as Copyleft and use red as signalling color.

Non Standard or Not Provided license are classified as a moderate threat and use orange. Non Standard as a classification is triggered by the usage of atypical licenses for open source software such as [CharityWare license](#), [BeerWare](#), [NCSA Open Source License](#) and many others. Not Provided is triggered as classification if no license information was found anywhere.

Licenses such as CDDL-1.0, EPL-1.0 or GPL-2.0-CPE receive a Weak Copyleft classification and yellow as notification color.

Liberal licences that are generally friendly to inclusion in commercial products are using blue and include licences such as Apache-2.0, MIT or BSD.

A general description about the implications of the different licenses is available when hovering over the specific category in the License Analysis Summary. Further information about the different licenses can be obtained from the [Open Source Initiative](#). Mixed license scenarios like a mixture of licenses such as Apache-1.1, Apache-2.0, LGPL and LGPL-2.1 can be complicated to assess in its impact and might be legally invalid depending on the combination of licenses observed. Detailed implications to your business and software are best discussed with your lawyers.

License Threat	Declared License	Observed Licenses in	Group	Artifact	Version
GPL	Apache-2.0	Apache-2.0, GPL	org.sonatype.configuration	base-configuration	1.1
GPL-2.0+	Apache-2.0+, BSD, EPL-	Apache-2.0, BSD, EPL-1	biz.sourceforge_code	base64coder	2010-12-19
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	org.glassfish.core	glassfish	3.1-b13
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	org.glassfish	jaxws.jms	3.1
GPL-2.0, GPL-2.0+	Apache-2.0	Apache-1.1, Apache-2.0,	org.apache.servicemix	servicemix-scripting	2008.01
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	org.glassfish	jaxws.transaction	10.0-b28
GPL	Apache-2.0	Apache, Apache-2.0, GP	org.apache.camel	camel-jms	2.3.0
GPL	AFL-2.1, Apache-2.0, BS	AFL-2.1, Apache-2.0, BS	org.cometd	cometd-demo	1.1.3
GPL-2.0+	GPL-2.0-with-classpath=1	GPL-2.0+	me.springframework	spring-me-sample-j2	1.0
GPL	Apache-2.0	Apache-2.0, GPL	org.apache.camel	camel-core	2.1.0
GPL	GPL	Not Provided	org.glassfish.pdf	pfl-asm	3.2.0-b01
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	com.sun.xml.ws	policy	2.0-b01
GPL	Not Provided	GPL, MIT	org.scala-tools.sxr	sxr_2.7.2	0.2
GPL, GPL-3.0+	GPL	GPL-3.0+	org.sonatype.nexus.res	nexus-restlight-m2-s	1.8.0
GPL, GPL-2.0	Apache-2.0	Apache-2.0, CDDL-1.0, C	org.apache.servicemix.j	org.apache.servicemix	2.1.13_1
GPL-3.0	Apache-2.0	Apache-2.0, GPL-3.0	org.ujoframework	uj-o-core	1.20
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	com.sun.grizzly	grizzly-websockets	1.9.36
GPL, GPL-2.0	CDDL, GPL, GPL-2.0	Not Provided	org.glassfish.web	web-gui-plugin-comm	3.1.1

Figure 11.5: The License Data in the Detailed Repository Health Check Report

Nexus will report all artifacts in the local storage of the respective repository in the detail panel. This means that at some stage a build running against your Nexus instance required these artifacts and caused Nexus to download them to local storage.

To determine which project and build caused this download to be able to fix the offending dependency by upgrading to a newer version or removing it with an alternative solution with a more suitable license you will have to investigate all your projects.

Sonatype Insight itself helps with these tasks by enabling monitoring of builds and products, analyzing

release artifacts and creating bill of material and other reports.

11.3 Using Repository Health Check Results For Component Life-cycle Management

11.3.1 Example: Analyzing a Spring Beans Vulnerability

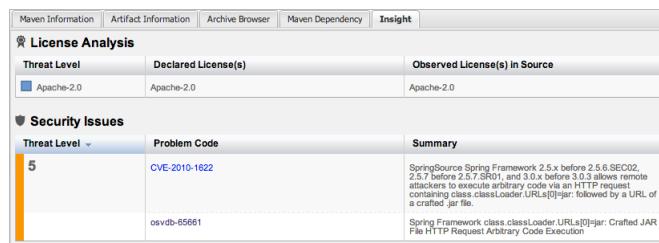
The following example details how you can analyze security issues of an artifact found in your repository health check and determine a solution with the help of information available in Nexus.

After performing a repository health check as documented in the prior sections of Chapter 11, you noticed the artifact with the Group `org.springframework`, the Artifact `spring-beans` and Version `2.5.4`. Upon further inspection of your software build and the components used, you can confirm that this artifact is indeed part of your shipping software.

Tip

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

A GAV search for the artifact in Nexus as documented in Section 5.4 allows you to inspect the Insight tab for the artifact displayed in Figure 11.6.



The screenshot shows the 'Insight' tab for the artifact `org.springframework:spring-beans:2.5.4`. The 'License Analysis' section shows that the declared license is Apache-2.0 and the observed license in the source is also Apache-2.0. The 'Security Issues' section lists a single issue with a threat level of 5, identified as CVE-2010-1622, which is described as a remote code execution vulnerability. The summary notes that this is due to SpringSource Spring Framework 2.5.x before 2.5.6 SEC-200, 2.5.7 before 2.5.7.2R01, and 3.0.x before 3.0.5, allowing remote attackers to execute arbitrary code via an HTTP request containing `classLoaderURLs[0].jar`: followed by a URL of a crafted .jar file. It also mentions a related issue: Spring Framework class.classLoader.URLs[0].jar: Crafted JAR File HTTP Request Arbitrary Code Execution.

Figure 11.6: Insight tab for `org.springframework:spring-beans:2.5.4`

After reading the summary and inspecting the entries for the security issues in the security databases linked in the Problem Code column, you decide that these issues affect your software and a fix is required. In order to determine your next steps you search for all versions of the spring-beans artifact. As a result you receive the list of all versions available partially displayed in Figure 11.7. The security column in the search results list displays the count of two security issues for the version 2.5.4 of the library.

Welcome		Search					
GAV Search	Group:	Artifact:	Version:	Packaging:			
Group	Artifact	Version ▲	Age	Popularity	Security Issues	License Threat	
org.springframework	spring-beans	3.0.4.RELEASE	2.2 yrs	<div style="width: 10%;">█</div>	0	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	3.0.3.RELEASE	2.4 yrs	<div style="width: 20%;">█</div>	0	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	3.0.2.RELEASE	2.6 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	3.0.1.RELEASE	2.7 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	3.0.0.RELEASE	2.9 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.6.SEC03	1.1 yrs	<div style="width: 10%;">█</div>	0	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.6.SEC02	2.4 yrs	<div style="width: 10%;">█</div>	0	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.6.SEC01	3.5 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.6	4.0 yrs	<div style="width: 50%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.5	4.4 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.4	4.5 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.3	4.6 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.2	4.7 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	
org.springframework	spring-beans	2.5.1	4.8 yrs	<div style="width: 10%;">█</div>	2	<div style="width: 10%;">█</div>	

Figure 11.7: Viewing Multiple Versions of org.springframework:spring-beans:x

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

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

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

The necessary steps will depend on the build system used, but in all cases you now have the information at your hands why you should upgrade and what version to upgrade to, which allows you to carry out your component lifecycle management effectively.

11.3.2 Example: Resolving a License Issue

The following example details how you can analyze a license issue of an artifact found in your repository health check and determine a solution with the help of information available in Nexus.

Your repository health check detail report indicated that Hibernate 3.2.7.GA might have issues due to its Threat Level declared as Non-Standard. Looking at your software artifacts you found that you are indeed using this version of Hibernate. Searching for the artifact in Nexus provides you with the search results list and the Insight tab for the specific version displayed in Figure 11.8.

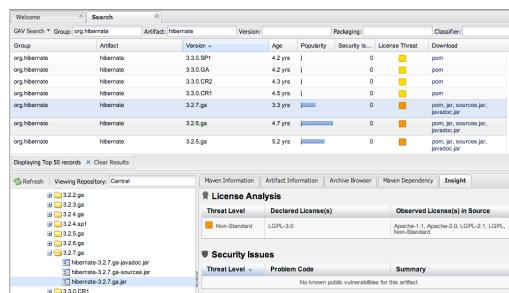


Figure 11.8: Viewing License Analysis Results for Hibernate

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

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

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

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

Chapter 12

Sonatype CLM Integration

12.1 Introduction

As discussed in Chapter 2, Component Lifecycle Management (CLM) and Repository Management are closely related activities. The Sonatype CLM suite of tools provides a server application for administering your component usage policies and other features that integrate with other tools of the suite. It has access to extensive security vulnerability and license information data from the Sonatype CLM backend, that can be used as input for your policies. For example you could establish a policy is logged as violated if any components in your software has a known security vulnerability or uses a license that is incompatible with your business model.

Find out more about the Sonatype CLM tools in the book "Optimized Component Lifecycle Management with Sonatype CLM" available in [PDF](#) or [HTML format](#).

Nexus is an important component that can take advantage of the CLM server. This chapter goes into the details of configuring and using the integration of the CLM server and Nexus.

12.2 Connecting Nexus to CLM Server

The Sonatype CLM server is a separate server application that Nexus integrates with via API calls. To configure Nexus to be able to contact the CLM server, you need to click on the **CLM** menu item in the left hand Administration menu in Nexus, which will open the tab visible in Figure 12.1.

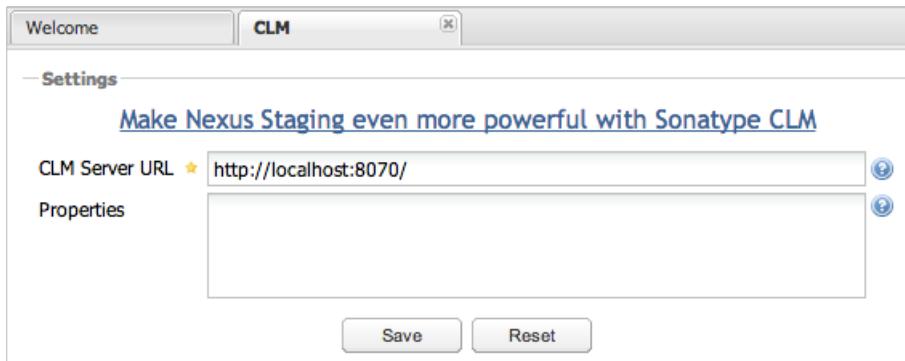


Figure 12.1: CLM configuration tab in Nexus

The CLM connection is established by providing the URL to the CLM server in the `CLM Server URL` input field.

Additional details can be configured in the `Properties` input field using a `key=value` definition per line. An example is

```
procArch=false  
ipAddresses=true  
operatingSystem=false
```

These properties are passed to the CLM server and can, for example, determine what properties are logged as part of a validation. Consult the CLM server documentation for suitable parameters. In most use cases you will not need to configure any properties.

Press **Save** once you have entered the desired URL and properties, and Nexus will attempt to contact the CLM server and potentially display an error message if the CLM server could not be contacted.

12.3 Configuring the CLM Server

With the connection between the CLM Server and Nexus established, you can configure any applications, application policies and constraints in the CLM server. Nexus will be accessing the CLM server using an application identifier, so you will have to configure one application for each different application use case in Nexus.

Find out more about configuring the CLM server in the book "Optimized Component Lifecycle Management with Sonatype CLM" referenced earlier.

12.4 Using CLM for Staging

Before using CLM for Staging you should familiarize with the general setup and usage patterns of the Nexus Staging Suite documented in Chapter [10](#).

To use the rich configuration of policies and rules of the CLM Server for staging component releases in Nexus, you have to configure the Stage Release and Release actions for each policy as desired.

An example policy that would warn for a staging deployment and fail a release is visible in Figure [12.2](#)

Edit Policy

Name	OlderThan1year	Threat Level	3		
Constraints	OlderThan1year	<input type="button"/> <input type="button"/> <input type="button"/>			
Actions	Stage <input type="button"/>	Fail	Warn	Do Nothing	Notify
	Procure			✓	👤 0
	Develop			✓	👤 0
	Build			✓	👤 0
	Stage Release		✓		👤 0
	Release	✓			👤 0
	Operate			✓	👤 0

Figure 12.2: Staging and Release Configuration for a Policy in the CLM Server

This configuration can then be used for a staging profile or a build promotion profile by configuring the CLM Application Id with the identifier for the application in the CLM server. Figure 12.3 shows an example staging profile with a CLM application configured.

The screenshot shows the configuration of a staging profile named 'sonatype-sample'. The configuration includes:

- Profile ID: 12ab45c1f7c026f6
- Deploy URL: http://localhost:8081/nexus/service/local/staging/deploy/maven2
- Profile Name: sonatype-sample
- Profile Selection Strategy: Explicit or Implicit - selected by ID or Nexus will calculate match
- Searchable Repositories: checked
- Staging Mode: Deploy and UI Upload
- Template: Maven2 (hosted, release)
- Repository Target: Sonatype Sample
- Release Repository: Releases
- CLM Application Id: bom1-12345678
- Content Type: maven2

Figure 12.3: Staging Profile with a CLM Application Configured

The configuration of the Stage Release action of a policy in the CLM server is used for closing the staging repository. If it is set to Fail and a constraint is violated, the staging repository closing fails. A Warn setting will validate the policy and produce a warning. Do Nothing skips the validation.

The Release configuration similarly can be configured to fail, warn or do nothing and is used for releasing or promoting the staging repository.

Once the staging profile is configured with the CLM application identifier any deployment potentially triggers a CLM rule evaluation, which will be visible as Activity for the staging repository. Any rule failures are provided with further information in the detail panel. Figure 12.4 displays a staging repository with CLM rule validations and a failure. The View Full Report buttons links back to the Sonatype CLM server, which displays a detailed report.

The screenshot shows the Nexus Staging Repository interface. At the top, there's a toolbar with 'Refresh', 'Close', 'Promote', 'Release', and 'Drop' buttons. A 'Filter by profile' dropdown is also present. Below the toolbar is a table listing repositories:

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

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

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

To the right of the activity tree, a summary of a recent event is displayed:

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

Validation failed.

Found 28 violations in 28 components

7 CRITICAL **21 SEVERE** **View Full Report**

Figure 12.4: Staging Repository Activity with a CLM Evaluation Failure and Details

CLM for staging in Nexus combines the powerful controls for your release process from Nexus with the rich information and validation available in the CLM server. Using them together you can ensure that any releases you produce are actively and automatically validated against up to date information in terms of security vulnerabilities and license characteristics of all the components you use and any whitelists or blacklists you maintain are enforced.

Chapter 13

Managing Maven Settings

13.1 Introduction

When you move an organization to a repository manager such as Nexus, one of the constant challenges is keeping everyone's Maven Settings synchronized to ensure the Nexus server is used and any further configuration in the settings file is consistent. In addition different users or use cases require different settings files. You can find out more about the Maven settings file in Chapter 4. Nexus Professional allows you to define templates for Maven Settings stored on the server and provide them to users via the user interface or automated download.

If a Nexus administrator makes a change which requires every developer to modify his or her `~/.m2/settings.xml` file, this feature can be used to manage the distribution of Maven Settings changes to the entire organization. Once you have defined a Maven Settings template in Nexus Professional, developers can then use the Nexus M2Settings Maven Plugin to retrieve the new Maven settings file directly from Nexus Professional.

13.2 Manage Maven Settings Templates

To manage Maven Settings templates, click on `Maven Settings` in the `Enterprise` section of the Nexus menu on the left side of the Nexus UI. This will load the panel shown in Figure 13.1.

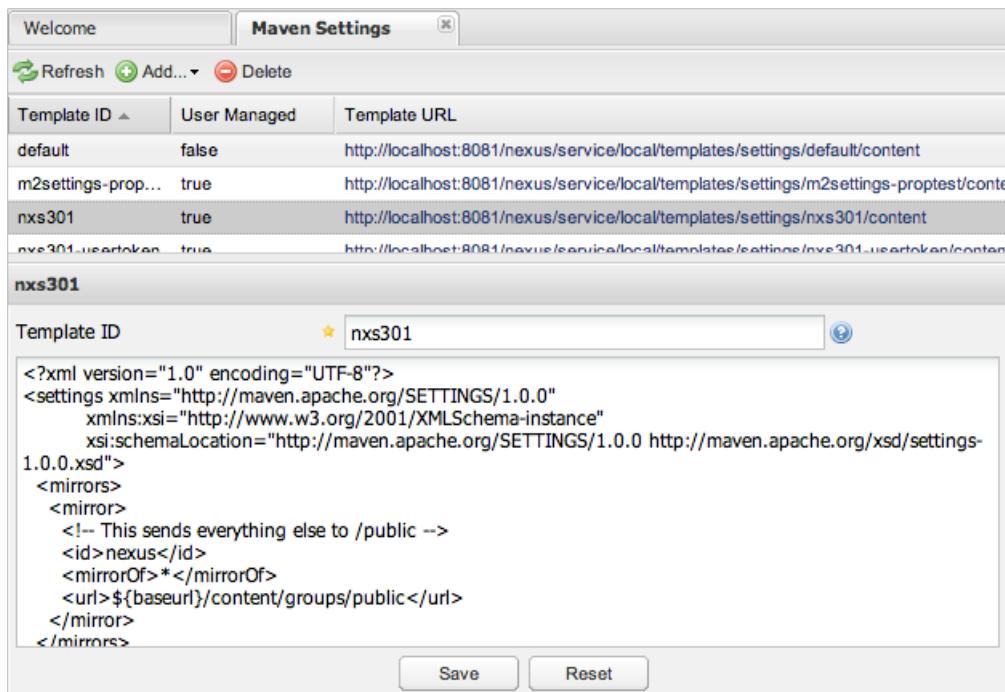


Figure 13.1: The Maven Settings Panel

The Maven Settings panel allows you to add, delete, and edit Maven Settings templates. The default template has an id of "default" and can not be changed. It contains the recommended settings for a standard Nexus installation. To create a new Maven Settings template, click on the Add... button and select Settings Template. Once the new template is created, assign a name to the template in the Template ID text input and click the Save button.

To edit a template, click on a template that has a User Managed value of true in the list and edit the template in the tab below the list. Once you are finished editing the template, click Save to save the template. When editing the template you can insert some property references that will be replaced on the server with their values at request time:

baseurl

the base URL of the Nexus installation

userId

the user id of the user that is generating a Maven Settings file from this template

Server side interpolation takes effect even when the download of the settings template is done with tools like curl. These properties can

```
<settings>
  <mirrors>
    <mirror>
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>${baseurl}/content/groups/public</url>
    </mirror>
  ...
...
```

To preview a Maven Settings template, click on the `Template URL` in the list. Clicking on this URL loads a dialog window which contains the Maven Settings file generated from this template. This rendered view of the Maven Settings template has all variable references replaced using the current context of the user. This is the result of running the property replacement on the Nexus server.

The Nexus M2Settings Maven Plugin supports the more powerful and feature rich, client side replacement of properties using a `$ [property]` syntax.

Client side properties supported by the Nexus M2Settings Maven Plugin are

baseurl

the base URL of the Nexus installation.

userId or username

the username of the user that is requesting a Maven Settings file from this template.

password

the password of the user

userToken

the formatted user token composed of name code, : and pass code.

userToken.nameCode

the name code part of the user token

userToken.passCode

the pass code part of the user token

userToken.encrypted

the encrypted, formatted user token

userToken.nameCode.encrypted

the encrypted name code part of the user token

userToken.passCode.encrypted

the encrypted pass code part of the user token

Client side interpolation allows you to fully populate a `<server>` section with the required properties either with the plain text username and password,

```
<server>
  <id>nexus</id>
  <username>$[username]</username>
  <password>$[password]</password>
</server>
```

the usertoken equivalent

```
<server>
  <id>nexus</id>
  <!-- User-token: ${userToken} -->
  <username>$[userToken.nameCode]</username>
  <password>$[userToken.passCode]</password>
</server>
```

or with [Maven master-password encryption](#) with the master keyword in `settings-security.xml`:

```
<server>
  <id>nexus-client-side-interp-encrypted</id>
  <!-- User-token: ${userToken.encrypted} -->
  <username>$[userToken.nameCode.encrypted]</username>
  <password>$[userToken.passCode.encrypted]</password>
</server>
```

The usage of the encrypted enwhich will result in values using the encrypted value syntax based on the master keyword similar to

```
<server>
  <id>nexus-client-side-interp-encrypted</id>
  <!-- User-token: {2Sn+...} -->
  <username>{3HQg...}</username>
  <password>{fsx2f...}</password>
</server>
```

**Warning**

`userToken.*` properties are only expanded to values if the User Token feature as documented in Section [6.15](#) is enabled and configured.

13.3 Nexus M2Settings Maven Plugin

Once you have defined a set of Maven templates, you can use the Nexus M2Settings Maven Plugin to distribute changes to the settings file to the entire organization.

13.3.1 Running the Nexus M2Settings Maven Plugin

To invoke a goal of the Nexus M2Settings Maven Plugin, you will initially have to use a fully qualified groupId and artifactId in addition to the goal. An example invocation of the `download` goal is:

```
mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:download
```

In order to be able to use an invocation with the simple plugin prefix like this

```
mvn nexus-m2settings:download
```

you have to have the appropriate plugin group `org.sonatype.plugins` configured in your Maven Settings file:

```
<settings>
  ...
  <pluginGroups>
    <pluginGroup>org.sonatype.plugins</pluginGroup>
  </pluginGroups>
  ...
```

An initial invocation of the `download` goal will update your settings file, with a template from Nexus Professional. The default template in Nexus Professional adds the `org.sonatype.plugins` group to the `pluginGroups`, so you will not have to do this manually. It is essential that you make sure that any new, custom templates also includes this plugin group definition. Otherwise, there is a chance that a developer

could update his or her Maven Settings and lose the ability to use the Nexus Maven plugin with the short identifier.

Tip

This practice of adding pluginGroups to the settings file is useful for your own Maven plugins or other plugins that do not use the default values of `org.apache.maven.plugins` or `org.codehaus.mojo` as well, since it allows the short prefix of a plugin to be used for an invocation outside a Maven project using the plugin.

The download goal of the Nexus M2Settings Maven Plugin downloads a Maven Settings file from Nexus Professional and stores it locally. The default file name for the settings file is the Maven default for the current user of `~/.m2/settings.xml`. If you are replacing a Maven Settings file, this goal can be configured to make a backup of an existing Maven Settings file.

Note

The download with the Nexus Maven Plugin is deprecated and has been replaced with the Nexus M2Settings Maven Plugin.

13.3.2 Configuring Nexus M2Settings Maven Plugin

The download goal of the Nexus M2Settings Maven plugin prompts the user for all required parameters, which include the Nexus server URL, the username and password and the template identifier.

Note

For security reason the settings download requires a HTTPS connection to your Nexus instance. If you are running Nexus via plain HTTP you will have to set the `secure` parameter to `false`.

The required configuration parameters can either be supplied as invocation parameters or when prompted by the plugin and are:

nexusUrl

points to the Nexus server installation's base URL. If you have installed Nexus on your local ma-

chine, this would be <http://localhost:8081/nexus/>. Access via http only works with the secure configuration parameter set to false.

username

the username to use for authenticating to Nexus. Default value is the Java System property user.name

password

password to use for authenticating to Nexus

templateId

the Template ID for the settings template as defined in the Nexus user interface

Additional general configuration parameters are related to the security of the transfer and the output file:

secure

by default set to true, this parameter forces a Nexus URL access with HTTPS. Overriding this parameter and setting it to false allows you to download a settings file via HTTP. When using this override it is important to keep in mind that the username and password transferred via HTTP can be intercepted.

outputFile

defines the filename and location of the downloaded file and defaults to the standard `~/.m2/settings.xml`

backup

If true and there is a pre-existing settings.xml file in the way of this download, backup the file to a date-stamped filename, where the specific format of the date-stamp is given by the backupTimestampFormat parameter. Default value is true.

backup.timestampFormat

When backing up an existing settings.xml file, use this date format in conjunction with SimpleDateFormat to construct a new filename of the form: settings.xml-\$format. Date stamps are used for backup copies of the settings.xml to avoid overwriting previously backed up settings files. This protects against the case where the download goal is used multiple times with incorrect settings, where using a single static backup-file name would destroy the original, pre-existing settings. Default value is: yyyyMMddHHmmss.

encoding

Use this optional parameter to define a non-default encoding for the settings file.

As a Maven plugin the Nexus M2Settings Maven Plugin relies on Apache Maven execution and therefore on the fact that the Central Repository can be contacted for downloading the required plugins and dependencies. If this access is only available via a proxy server you can configure the proxy related parameters proxy, proxy.protocol, proxy.host, proxy.port, proxy.username and proxy.password.

13.3.3 Downloading Maven Settings

You can download the Maven Settings from Nexus Professional with a simple invocation, and rely on the plugin to prompt you for the required parameters:

```
$ mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:download
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO]
[INFO] --- nexus-m2settings-maven-plugin:1.4.2:download (default-cli) @ ←
      standalone-pom ---
Nexus URL: https://localhost:8081/nexus
Username [manfred]: admin
Password: *****
[INFO] Connecting to: https://localhost:8081/nexus (as admin)
[WARNING] Insecure protocol: https://localhost:8081/nexus/
[INFO] Connected: Sonatype Nexus Professional 2.4.0-07
Available Templates:
  0) default
  1) example
Select Template: 0
[INFO] Fetching content for templateId: default
[INFO] Backing up: /Users/manfred/.m2/settings.xml to: /Users/manfred/.m2/ ←
      settings.xml-20130404120146
[INFO] Saving content to: /Users/manfred/.m2/settings.xml
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 29.169s
[INFO] Finished at: Thu Apr 04 12:01:46 PDT 2013
[INFO] Final Memory: 12M/153M
[INFO] -----
```

If your Nexus server is hosted internally and does not use https you can download a settings file with

```
$ mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:download -Dsecure ←
      =false
```

As displayed the plugin will query for all parameters and display a list of the available templates. Alternatively, you can specify the username, password, Nexus URL and template identifier on the command line.

```
$ mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:download \
```

```
-DnexusUrl=https://localhost:8081/nexus \
-Dusername=admin \
-Dpassword=admin123 \
-DtemplateId=default
```

Enabling proxy access with `-Dproxy=true` will trigger the plugin to query the necessary configuration:

```
[INFO] Connecting to: https://localhost:8081/nexus (as admin)
Proxy Protocol:
  0) http
  1) https
Choose: 1
Proxy Host: myproxy.example.com
Proxy Port: 9000
Proxy Authentication:
  0) yes
  1) no
Choose: 0
Proxy Username [manfred]: proxy
Proxy Password: *****
[INFO] Proxy enabled: proxy@https:myproxy.example.com:9000
```

In some scenarios you have to get an initial settings file installed on a computer that does not have internet access and can therefore not use the Maven plugin. For this first initial configuration, that connects the computer to Nexus for following Maven invocations, a simple HTTP GET command to retrieve an unmodified settings file can be used:

```
curl -u admin:admin123 -X GET "http://localhost:8081/nexus/service/local/ \
  templates/settings/default/content" > ~/.m2/settings.xml
```

Modify the commandline above by changing the `username:password` supplied after `-u` and adapting the url to Template URL visible in the Nexus user interface. This invocation will however not do the client side replacement of parameters, so you will have to manually change any `username` or `password` configuration, if applicable.

13.4 Summary

Overall the Maven Settings integration in Nexus allows you to maintain multiple settings template files on the central Nexus server. You can configure settings files for different use cases like referencing a repository group containing only approved components in the mirror section for your release or QA

builds, while at the same time providing an open public group mirror reference to all your developers for experimentation with other components.

By using the Nexus M2Settings Maven Plugin you can completely automate initial provisioning and updates of these settings files to your users.

Chapter 14

OSGi Bundle Repositories

14.1 Introduction

Nexus Professional supports the OSGi Bundle Repository format. The OSGi Bundle format is defined by the [OSGi RFC 112 "Bundle Repository"](#). It is a format for the distribution of OSGi "bundles" which includes any components that are described by the OSGi standards set forth in RFC 112. An OBR repository has a single XML file which completely describes the contents of the entire repository. Nexus Professional can read this OBR repository XML and create proxy repositories which can download OSGi bundles from remote OBR repositories. Nexus Professional can also act as a hosting platform for OSGi bundles, you can configure your builds to publish OSGi bundles to Nexus Professional, and then you can expose these bundle repositories to internal or external developers using Nexus Professional as a publishing and distribution platform.

Nexus Professional can also act as a bridge between Maven repositories and OSGi bundle repositories. When you configure a virtual OBR repository which uses a Maven 2 repository as a source repository, Nexus Professional will expose artifacts with the appropriate metadata from the Maven repository as OSGi bundles. In this way, you can unify your OSGi and non-OSGi development efforts and publish artifacts with the appropriate OSGi metadata to Nexus Professional. Non-OSGi clients can retrieve software artifacts from a Maven repository, and OSGi-aware clients can retrieve OSGi bundles from a virtual OBR repository.

The following sections detail the procedures for creating and managing OBR repositories.

14.2 Proxy OSGi Bundle Repositories

Nexus can proxy an OSGi Bundle Repository, using the OBR repository XML as the remote storage location. To create a new proxy OBR repository:

1. Login as an Administrator.
2. Click Repositories in the Left Navigation Menu.
3. Click the Add.. button above the list of Nexus repositories, and choose Proxy repository from the drop-down of repository types.
4. In the New Proxy Repository window,
 - a. Select OBR as the Provider.
 - b. Supply an id and a repository name.
 - c. Enter the URL to the remote repository OBR XML as the Remote Storage location.
 - d. Click Save.

Figure 14.1 provides some sample configuration used to create a proxy of the Apache Felix OBR repository.



Figure 14.1: Creating an OSGi Bundle Proxy Repository

To verify that the OBR proxy repository has been properly configured, you can then load the OBR XML from Nexus Professional. If Nexus Professional is properly configured, you will be able to load the obr.xml by navigating to the obr.xml directory:

```
$curl http://localhost:8081/nexus/content/repositories/felix-proxy/.meta/ ↔
  obr.xml
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type='text/xsl' href='http://www2.osgi.org/www/obr2html. ↔
  xsl'?>
<repository name='Felix OBR Repository' lastmodified='1247493075615'>
  <resource id='org.apache.felix(javax.servlet)/1.0.0'
    presentationname='Servlet 2.1 API'
    symbolicname='org.apache.felix(javax.servlet)'
    uri='../../bundles/org.apache.felix(javax.servlet)-1.0.0.jar'
    version='1.0.0'>
    <description>
      Servlet 2.1 API
    </description>
    <documentation>
      http://www.apache.org/
    </documentation>
    <license>
      http://www.apache.org/licenses/LICENSE-2.0.txt
    </license>
```

```
</license>
...

```

14.3 Hosted OSGi Bundle Repositories

Nexus can host an OSGi Bundle Repository, providing you with a way to publish your own OBR bundles. To create an OBR hosted repository:

1. Login as an Administrator.
2. Click Repositories in the Left Navigation Menu.
3. Click the Add.. button above the list of Nexus repositories, and choose Hosted repository from the drop-down of repository types.
4. In the New Hosted Repository window,
 - a. Select OBR as the Provider.
5. Supply an id and a repository name.
6. Click Save.

Figure 14.2 provides some sample configuration used to create a hosted OBR repository.

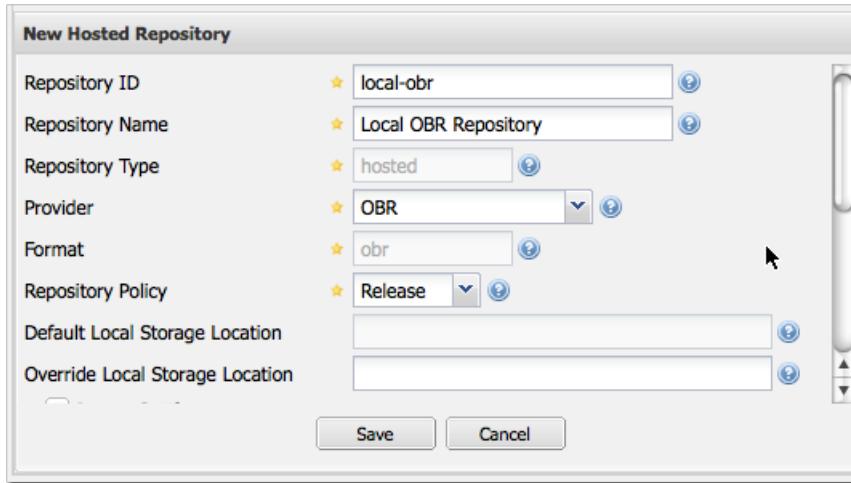


Figure 14.2: Creating a Hosted OSGi Bundle Repository

14.4 Virtual OSGi Bundle Repositories

Nexus Professional can also be configured to convert a traditional Maven repository into an OSGi Bundle repository using a virtual OBR repository. To configure a virtual OBR repository:

1. Login as an Administrator.
2. Click Repositories in the Left Navigation Menu.
3. Click the Add.. button above the list of Nexus repositories, and choose Virtual repository from the drop-down of repository types.
4. In the New Virtual Repository window,
 - a. Select OBR as the Provider.
 - b. Select another repository's ID in the Source Nexus Repository ID drop-down
 - c. Supply an id and a repository name.
 - d. Click Save.

The next figure provides some sample configuration used to create a virtual OBR repository which transforms the proxy repository for Maven Central into an OBR repository.

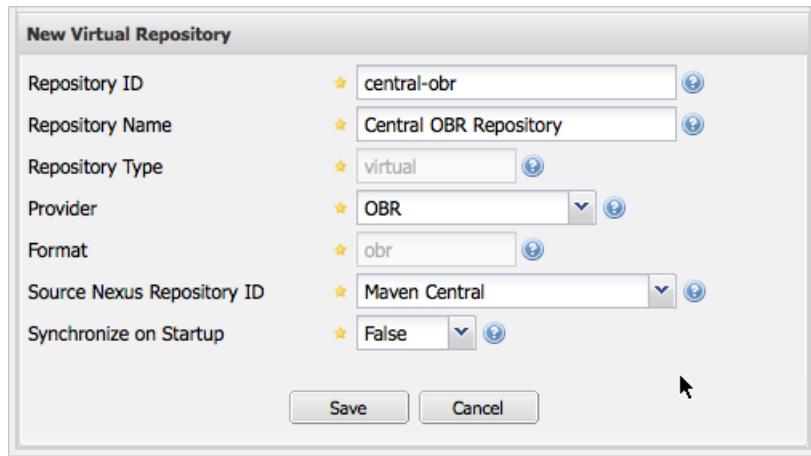


Figure 14.3: Creating a Virtual OSGi Bundle Repository from a Maven Repository

14.5 Grouping OSGi Bundle Repositories

Just like Nexus can group Maven repositories, Eclipse update sites, and P2 repositories, Nexus can also be configured to group OSGi Bundle Repositories. To group OSGi bundle repositories:

1. Login as an Administrator.
2. Click Repositories in the Left Navigation Menu.
3. Click the Add.. button above the list of Nexus repositories, and choose Repository Group from the drop-down of repository types.
4. In the New Repository Group window,
 - a. Select OBR Group as the Provider.
 - b. Drag and drop one or more hosted, proxy, or virtual OSGi Bundle repositories into the new group.
 - c. Supply an id and a repository name.
 - d. Click Save.

Figure 14.4 shows an example of the a new repository group which contains a hosted OSGi Bundle repository, a virtual OSGi Bundle repository, and a OSGi Bundle proxy repository.

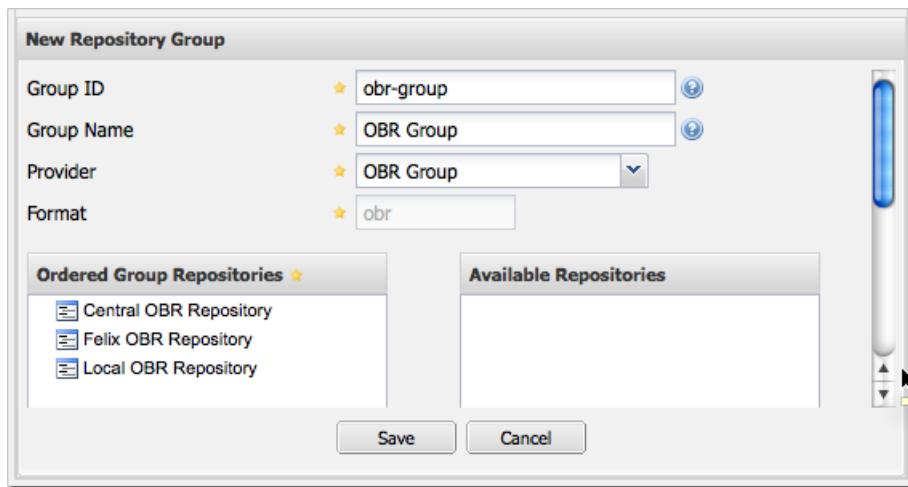


Figure 14.4: Creating a new OSGi Bundle Repository Group

Chapter 15

P2 Repositories

15.1 Introduction

Nexus Professional supports the P2 Repository format. The P2 repository format is a provisioning platform for Eclipse components. For more information about the P2 repository format, see the [Equinox P2 documentation](#) on the Eclipse Wiki.

The following sections detail the procedures for creating and managing P2 repositories.

15.2 Proxy P2 Repositories

Nexus can proxy a P2 Repository. To create a new proxy P2 repository:

1. Login as an Administrator.
2. Click **Repositories** in the Left Navigation Menu.
3. Click the **Add..** button above the list of Nexus repositories, and choose **Proxy repository** from the drop-down of repository types.
4. In the New Proxy Repository window,

- a. Select P2 as the Provider.
- b. Supply an id and a repository name.
- c. Enter the URL to the remote P2 repository as the Remote Storage location.
- d. Click Save.

Figure 15.1 provides some sample configuration used to create a proxy of the Indigo Simultaneous Release P2 repository.

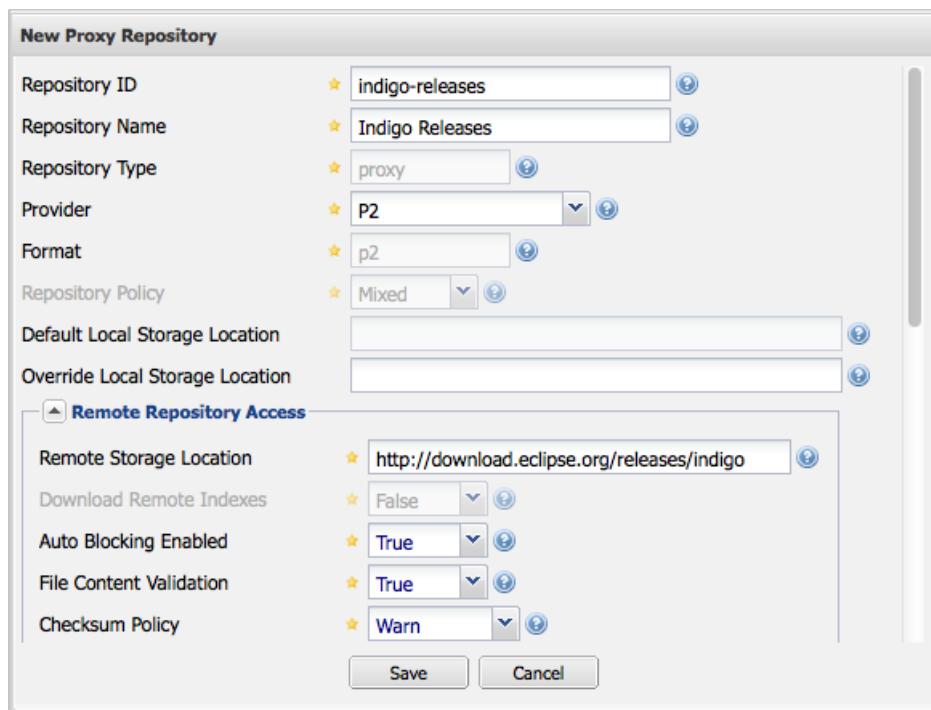


Figure 15.1: Creating a P2 Proxy Repository

15.3 Grouping P2 Repositories

Just like Nexus can group Maven repositories and OBR repositories, Nexus can also be configured to group P2 Repositories. To group P2 repositories:

1. Login as an Administrator.
2. Click Repositories in the Left Navigation Menu.
3. Click the Add.. button above the list of Nexus repositories, and choose Repository Group from the drop-down of repository types.
4. In the New Repository Group window,
 - a. Select P2 as the Provider.
 - b. Drag and drop one or more P2 repositories into the new group.
 - c. Supply an id and a group name.
 - d. Click Save.

Figure 15.2 shows an example of a repository group which contains two P2 proxy repositories.

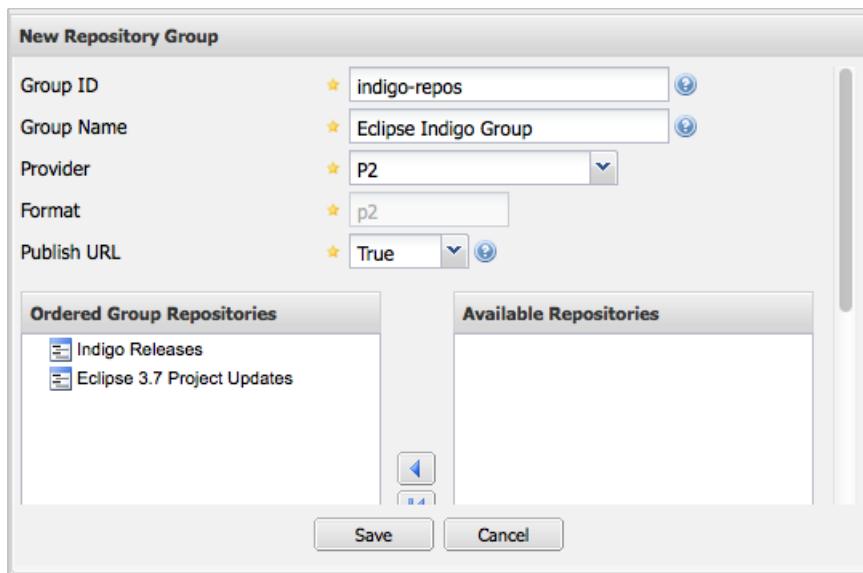


Figure 15.2: Creating a new P2 Repository Group

Chapter 16

.NET Package Repositories

16.1 Introduction

With the recent creation of the [NuGet](#) project a package management solution for .NET developers has become available. Similar to 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 [nuget.org](#) and [nuget.codeplex.com](#) host tool downloads, 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 supports the NuGet repository format for hosted and proxy repositories. Nexus also supports aggregation of NuGet repositories and conversion of other repositories containing ".nupkg" artifacts to the NuGet format. This allows you to improve collaboration and control while speeding up .NET development facilitating open source libraries and sharing of internal artifacts across teams. When you standardize on a single repository for all your development and use it for internal artifacts as well you will get all the benefits of Nexus when working in the .NET architecture.

To share a library or tool with NuGet you create a NuGet package and store it in the Nexus based NuGet repository. Similarly you can use packages others have created and made available in their NuGet repository by proxying it or downloading the package and installing it 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.

16.2 NuGet Proxy Repositories

To proxy an external NuGet repository you simply create a new Proxy Repository as documented in Section 6.2. The Provider has to be set to NuGet. The Remote Storage Location has to be set to the source URL of the repository you want to proxy.

A complete configuration for proxying nuget.org is visible in Figure 16.1.

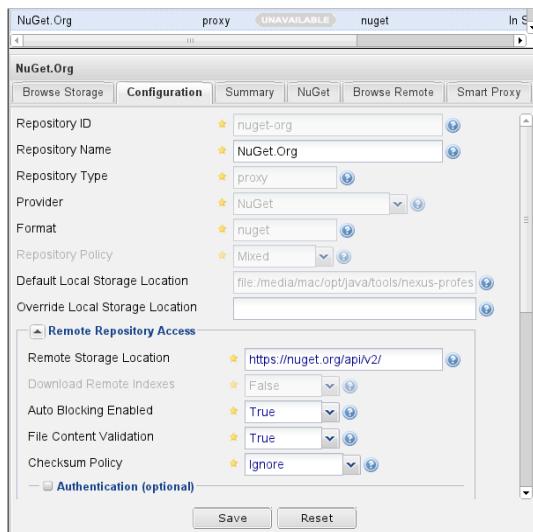


Figure 16.1: NuGet Proxy Repository Configuration for nuget.org

The source URL for the main NuGet.org repository is

```
http://nuget.org/api/v2/
```

The repository configuration for a NuGet proxy repository has an additional tab titled NuGet as visible in Figure 16.2, which displays the Nexus URL at which the repository is available as a NuGet repository.

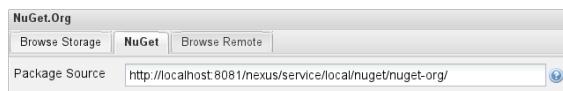
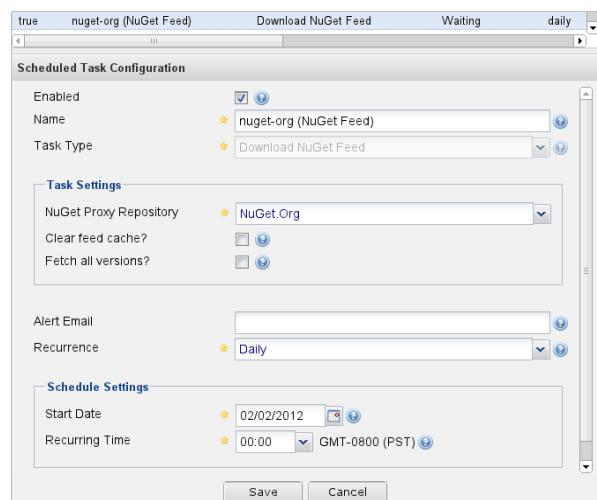


Figure 16.2: NuGet Gallery with Package Source URL

When creating a NuGet proxy repository a Scheduled Task is automatically created to download the index data about the content of the remote NuGet repository. This task is available in the Scheduled Tasks administration section and by default created for a daily schedule. To modify the task access it via the Administration panel in the left hand navigation area and the Scheduled Tasks menu item. The task will be using the name of the proxy repository with (NuGet Feed) appended. A user interface as displayed in Figure 16.3 will allow you to adjust the task as desired.



16.3 NuGet Hosted Repositories

A hosted repository for NuGet can be used to upload your own packages as well as third party packages. It is good practice to create two separate hosted repositories for these purposes.

To create a NuGet hosted repository simply create a new Hosted Repository and set the Provider to NuGet. A sample configuration for an internal releases NuGet hosted repository is displayed in Figure 16.4.

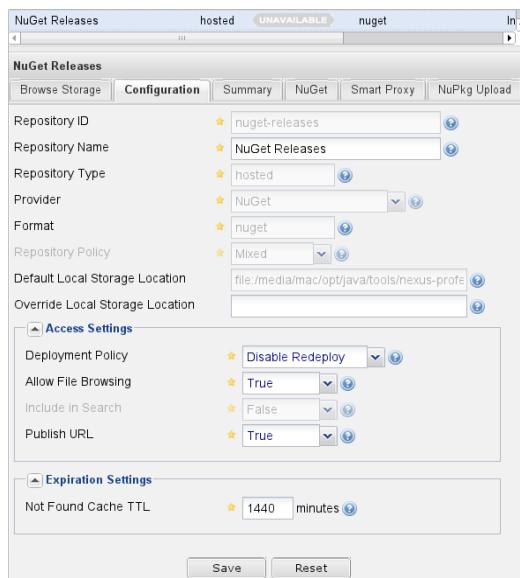


Figure 16.4: Example Configuration for a NuGet Hosted Repository for Release Packages

Besides the NuGet tab the configuration for the repository has a NuPkg Upload tab as displayed in Figure 16.5, that allows you to manually upload one or multiple packages.

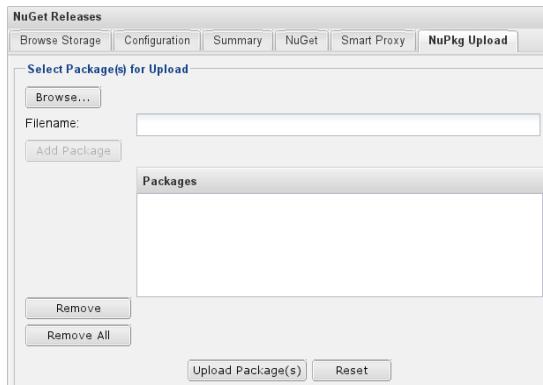


Figure 16.5: The NuPkg Upload Panel for a Hosted NuGet Repository

The NuGet feed is immediately updated as packages are deployed or deleted from the host repository. If for some reason you ever need to rebuild the feed for a hosted NuGet repository you can manually schedule a Rebuild NuGet Feed task.

16.4 NuGet Virtual Repositories

If you have deployed NuGet packages to a Maven repository in the past you can expose them to Visual Studio by creating a Virtual repository and setting the Format to NuGet Shadow Repository. The setup displayed in Figure 16.6 shows a virtual repository set up to expose the content of the regular Maven Releases repository in the form of a NuGet repository, so that NuGet can access any NuGet packages deployed to the releases repository.

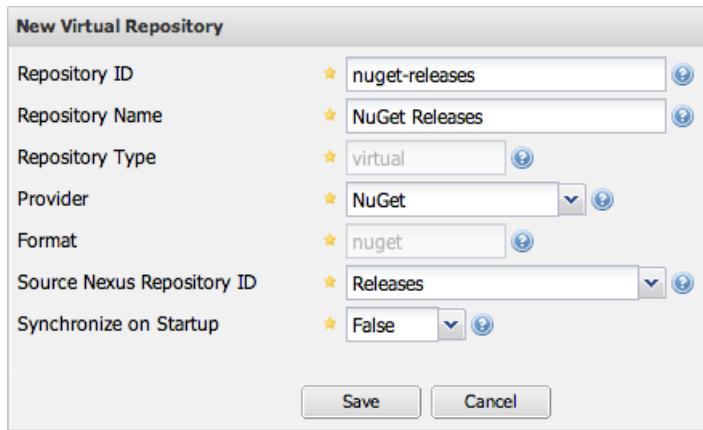


Figure 16.6: A Virtual NuGet Repository for the Releases Repository

The NuGet feed is immediately updated as packages are deployed or deleted from the shadowed repository. If for some reason you ever need to rebuild the feed for a virtual NuGet repository, you can manually schedule a Synchronize Shadow Repository task.

16.5 NuGet Group Repositories

A repository group allows you to expose the aggregated content of multiple proxy and hosted repositories with one URL to your tools. This is possible for NuGet repositories by creating a new Repository Group with the Format set to NuGet.

A typical useful example would be to group the proxy repository that proxies nuget.org, an internal releases NuGet hosted repository and a third party Nuget hosted repository. The configuration for such a setup is displayed in Figure 16.7.

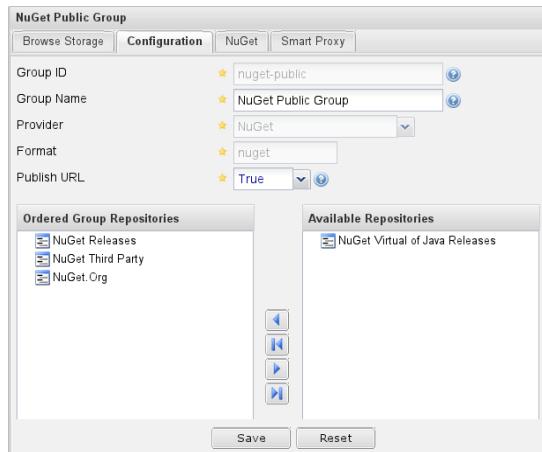


Figure 16.7: A Public Nuget Group Combining a Proxy and Two Hosted Repositories

Using the Repository Path of the repository group as your NuGet repository URL in your client tool will give you access to the packages in all three repositories with one URL.

16.6 Accessing Packages in Repositories and Groups

Once you have set up your hosted and proxy repositories for NuGet packages and potentially created a group you can access them with the nuget tool on the command line. Copy the Package Source url from the NuGet tab of the repository/group configuration you want to access and add it to nuget on the command line with e.g.:

```
nuget sources add -name NuGetNexus -source http://localhost:8081/nexus/ ↵
  service/local/nuget/nuget-public
```

Replace localhost with the public hostname or url of your Nexus server and nuget-public with the name of the repository you want to proxy. Ideally this will be your NuGet group.

After this source was added you can list the available packages with

```
nuget list
```

16.7 Deploying Packages to NuGet Hosted Repositories

In order to authenticate a client against a NuGet repository NuGet uses an API key for deployment requests. These keys are generated separately on-request from a user account on the NuGet gallery and can be re-generated at any time. At regeneration all previous keys generated for that user are invalid.

16.7.1 Creating a NuGet API-Key

For usage with Nexus, API keys are only needed when packages are going to be deployed. Therefore API key generation is by default not exposed in the user interface to normal users. Only users with the Deployer role have access to the API keys.

Other users that should be able to access and create an API key have to be given the Nexus API-Key Access role in the Users Security administration user interface.

In addition the NuGet API-Key Realm has to be activated. To do this, simply add the realm to the selected realms in the Security Settings section of the Server Administration.

Once this is set up you can view, as well as reset, the current Personal API Key in the NuGet tab of any NuGet proxy or hosted repository as visible in Figure 16.8

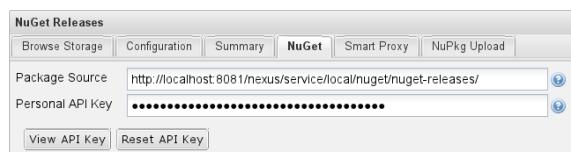


Figure 16.8: Viewing and Resetting the NuGet API Key in the NuGet Configuration Tab

16.7.2 Creating a Package for Deployment

Creating a package for deployment can be done with the pack command of the nuget command line tool or within Visual Studio. Detailed documentation can be found on the [NuGet website](#).

16.7.3 Deployment with the NuPkg Upload User Interface

Manual upload of one or multiple packages is done on the NuPkg Upload tab of the repository displayed in Figure 16.5. Press the Browse button to access the package you want to upload on the file system and press Add Package. Repeat this process for all the packages you want upload and press Upload Package(s) to complete the upload.

16.7.4 Command line based Deployment to a Nexus NuGet Hosted Repository

The nuget command line tool allows you to deploy packages to a repository with the push command. The command requires you to use the API Key and the Package Source path. Both of them are available in the NuGet tab of the hosted NuGet repository you want to deploy to. Using the delete command of nuget allows you to remove packages in a similar fashion.

Further information about the command line tool is available in the [on-line help](#).

16.8 Integration of Nexus NuGet Repositories in Visual Studio

In order to access a Nexus NuGet repository or preferable all Nexus NuGet repositories exposed in a group you provide the Repository Path in the Visual Studio configuration for the Package Sources of the Package Manager as displayed in Figure 16.9.

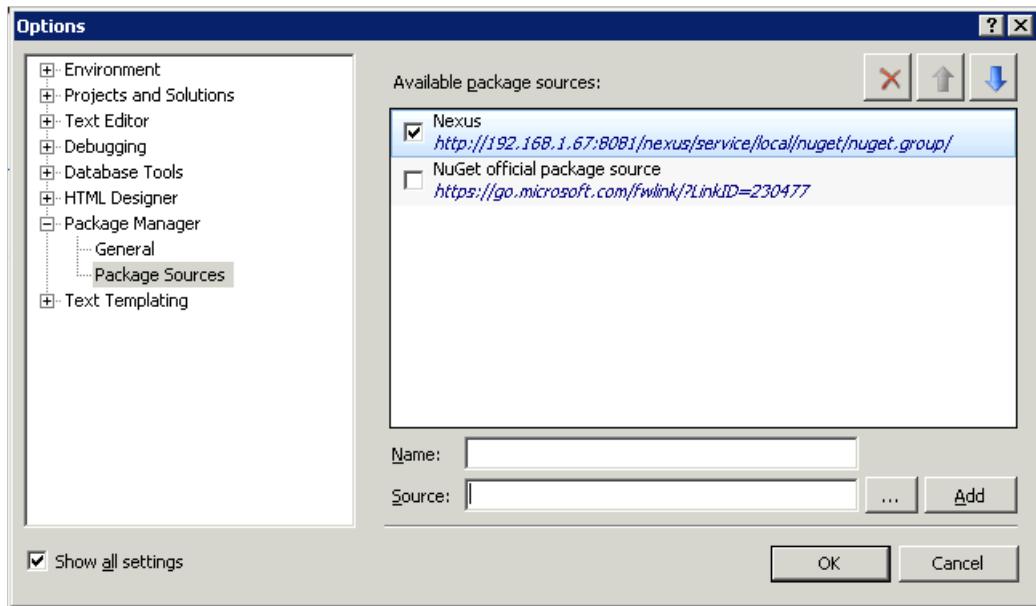


Figure 16.9: Package Source Configuration for the Package Manager in Visual Studio to Access A Nexus NuGet Repository Group

With this configuration in place all packages available in your Nexus NuGet repository will be available in the Package Manager in Visual Studio ready for install.

Chapter 17

Deploying Sites to Nexus

17.1 Introduction

Nexus provides a repository provider for hosting static websites - the "Site" format. Hosted repositories with this format can be used to hold a Maven-generated web site. This chapter details the process of configuring a site repository and configuring a simple Maven project to publish a Maven-generated project site to an instance of Nexus.

17.2 Creating a New Maven Project

In this chapter, you will be creating a simple Maven project with a simple web site that will be published to a Nexus Site repository. To create a new Maven project, use the archetype plugin's archetype:generate goal on the command line, and supply the following identifiers:

- groupId: org.sonatype.books.nexus
 - artifactId: sample-site
 - version: 1.0-SNAPSHOT
 - package: org.sonatype.books.nexus
-

```
~/examples$ mvn archetype:generate
[INFO] [archetype:generate {execution: default-cli}]
[INFO] Generating project in Interactive mode
Choose archetype:
1: internal -> appfuse-basic-jsf
...
13: internal -> maven-archetype-portlet (A simple portlet application)
14: internal -> maven-archetype-profiles ()
15: internal -> maven-archetype-quickstart ()

...
Choose a number: (...14/15/16...) 15: : 15
Define value for groupId: : org.sonatype.books.nexus
Define value for artifactId: : sample-site
Define value for version: 1.0-SNAPSHOT: : 1.0-SNAPSHOT
Define value for package: org.sonatype.books.nexus: : org.sonatype.books. ↵
    nexus
Confirm properties configuration:
groupId: org.sonatype.books.nexus
artifactId: sample-site
version: 1.0-SNAPSHOT
package: org.sonatype.books.nexus
Y: :
[INFO] Parameter: groupId, Value: org.sonatype.books.nexus
[INFO] Parameter: packageName, Value: org.sonatype.books.nexus
[INFO] Parameter: package, Value: org.sonatype.books.nexus
[INFO] Parameter: artifactId, Value: sample-site
[INFO] Parameter: basedir, Value: /private/tmp
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] OldArchetype created in dir: /private/tmp/sample-site
[INFO] ↵
----- ↵
[INFO] BUILD SUCCESSFUL
[INFO] ↵
----- ↵
[INFO] Total time: 23 seconds
[INFO] Finished at: Sat Oct 03 07:09:49 CDT 2009
[INFO] Final Memory: 13M/80M
[INFO] ↵
----- ↵
```

After running the archetype:generate command you will have a new project in a sample-site/ sub-directory.

17.3 Configuring Maven for Site Deployment

To deploy a site to a Nexus Site repository, you will need to configure the project's distribution management settings, add site deployment information, and then update your Maven settings to include the appropriate credentials for Nexus.

Add the following section to sample-site/pom.xml before the dependencies element. This section will tell Maven where to publish the Maven-generated project web site:

Distribution Management for Site Deployment to Nexus

```
<distributionManagement>
  <site>
    <id>nexus-site</id>
    <url>dav:http://localhost:8081/nexus/content/sites/site/</url>
  </site>
</distributionManagement>
```

Note

In [Distribution Management for Site Deployment to Nexus](#), there is a Nexus installation running on localhost port 8081. In your environment you will need to customize this URL to point to your own Nexus instance.

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 you can change the url to a parameterized setup.

Parameterized Distribution Management for Site Deployment

```
<url>
dav:http://localhost:8081/nexus/content/sites/site/${project.groupId}/${{ →
  project.artifactId}}/${project.version}
</url>
```

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 release of your software.

The dav protocol used by for deployment to Nexus requires that you add the implementing library as a build extension to your Maven project:

Build Extension for DAV Support

```
<build>
  <extensions>
    <extension>
      <groupId>org.apache.maven.wagon</groupId>
      <artifactId>wagon-webdav-jackrabbit</artifactId>
      <version>2.2</version>
    </extension>
  </extensions>
```

In addition to the distributionManagement element and the build extension, you will want to add the following build element that will configure Maven to use version 3.2 of the Maven Site plugin.

Configuring Version 3.2 of the Maven Site Plugin

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.2</version>
    </plugin>
  </plugins>
</build>
```

17.4 Adding Credentials to Your Maven Settings

When the Maven Site plugin deploys a site to Nexus, it needs to supply the appropriate deployment credentials to Nexus. To configure this, you need to add credentials to your Maven Settings. Open up your `~/.m2/settings.xml` and add the following server configuration to the servers element.

Configuring Deployment Credentials for Nexus Site Deployment

```
<settings>
  <servers>
    <server>
      <id>nexus-site</id>
      <username>deployment</username>
```

```
<password>deployment123</password>
</server>
</servers>
</settings>
```

Note

[Configuring Deployment Credentials for Nexus Site Deployment](#), uses the default deployment user and the default deployment user password. You will need to configure the username and password to match the values expected by your Nexus installation.

17.5 Creating a Site Repository

To create a site repository, log in as a user with Administrative privileges, and click on "Repositories" under Views.Repositories in the Nexus menu. Under the Repositories tab, click on the Add... drop-down and choose "Hosted Repository" as shown in Figure 17.1.

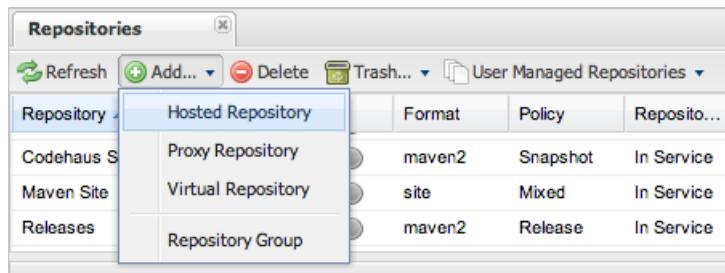


Figure 17.1: Adding a Hosted Repository

In the New Hosted Repository form, click on the Provider drop-down and chose the Site provider as shown in Figure 17.2. Although you can use any arbitrary name and identifier for your own Nexus repository, for the chapter's example, use a Repository ID of "site" and a Repository Name of "Maven Site".

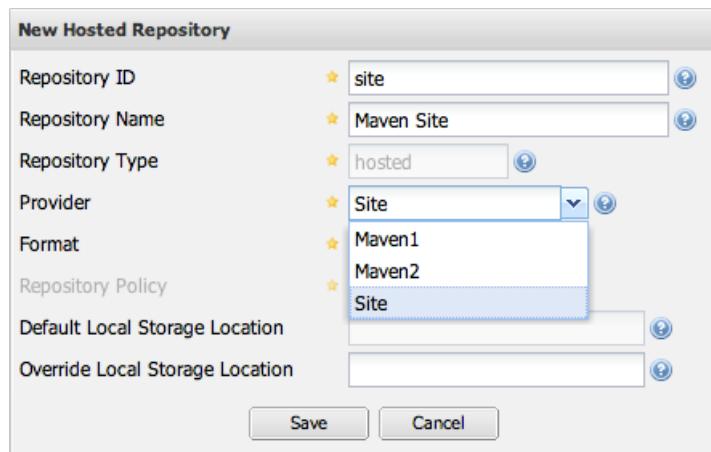


Figure 17.2: Creating a New Maven Site Repository

After creating a new Site repository, it should appear in the list of Nexus repositories as shown in Figure 17.3. Note that the Repository Path shown in Figure 17.3, is the same as the repository path referenced in [Distribution Management for Site Deployment to Nexus](#).

Repositories						
Repository	Type	Quality	Format	Policy	Repo...	Repository Path
Codehau...	proxy	ANALYZE	maven2	Snapshot	In Service	http://localhost:8081/nexus/content/repositories
Maven Site	hosted	ANALYZE	site	Mixed	In Service	http://localhost:8081/nexus/content/sites/site
Releases	hosted	ANALYZE	maven2	Release	In Service	http://localhost:8081/nexus/content/repositories

Figure 17.3: Newly Created Site Repository

Tip

The Site provider support is implemented in the Nexus Site Repository Plugin and is installed by default in Nexus Open Source as well as Nexus Professional.

17.6 Add the Site Deployment Role

In the Maven Settings shown in [Configuring Deployment Credentials for Nexus Site Deployment](#), you configured your Maven instance to use the default deployment user and password. To successfully deploy a site to Nexus, you will need to make sure that the deployment user has the appropriate role and permissions. To add the site deployment role to the deployment user, click on Users under the Security section of the Nexus menu, and click on the Add button the Role Management section. This will trigger the display of the Add Role dialog that will allow you to apply a filter value of Site to locate the applicable roles as shown in Figure 17.4.

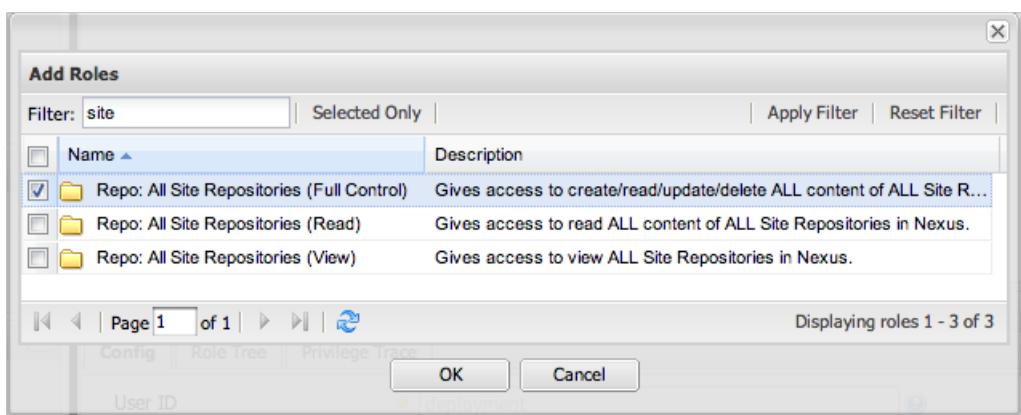


Figure 17.4: Adding the Site Deployment Role to the Deployment User

Check the box beside the "Repo: All Site Repositories (Full Control)" role in the list and press OK in the dialog. After the dialog closed, you should see the new role in the Role Management section. Click on the Save button to update the roles for the deployment user. The deployment user now has the ability to publish sites to a Maven Site repository.

17.7 Publishing a Maven Site to Nexus

To publish a site to a Maven Site repository in Nexus, run mvn site-deploy from the sample-site/ project created earlier in this chapter. The Maven Site plugin will deploy this site to Nexus using the credentials stored in your Maven Settings.

```
~/examples/sample-site$ mvn site-deploy
```

```
[INFO] Scanning for projects...
[INFO]  ↵
-----
[INFO] Building sample-site
...
[INFO] Generating "About" report.
[INFO] Generating "Issue Tracking" report.
[INFO] Generating "Project Team" report.
[INFO] Generating "Dependencies" report.
[INFO] Generating "Project Plugins" report.
[INFO] Generating "Continuous Integration" report.
[INFO] Generating "Source Repository" report.
[INFO] Generating "Project License" report.
[INFO] Generating "Mailing Lists" report.
[INFO] Generating "Plugin Management" report.
[INFO] Generating "Project Summary" report.
[INFO] [site:deploy {execution: default-cli}]
http://localhost:8081/nexus/content/sites/site/ - Session: Opened
Uploading: ./css/maven-base.css to http://localhost:8081/nexus/content/ ↵
sites/site/
#http://localhost:8081/nexus/content/sites/site//./css/maven-base.css \
- Status code: 201

Transfer finished. 2297 bytes copied in 0.052 seconds
Uploading: ./css/maven-theme.css to http://localhost:8081/nexus/content/ ↵
sites/site/
#http://localhost:8081/nexus/content/sites/site//./css/maven-theme.css \
- Status code: 201

Transfer finished. 2801 bytes copied in 0.017 seconds

Transfer finished. 5235 bytes copied in 0.012 seconds
http://localhost:8081/nexus/content/sites/site/ - Session: Disconnecting
http://localhost:8081/nexus/content/sites/site/ - Session: Disconnected
[INFO]  ↵
-----
[INFO] BUILD SUCCESSFUL
[INFO]  ↵
-----
[INFO] Total time: 45 seconds
[INFO] Finished at: Sat Oct 03 07:52:35 CDT 2009
[INFO] Final Memory: 35M/80M
[INFO] -----
```

Once the site has been published, you can load the site in a browser by going to <http://localhost:8081/nexus/content/sites/site/>

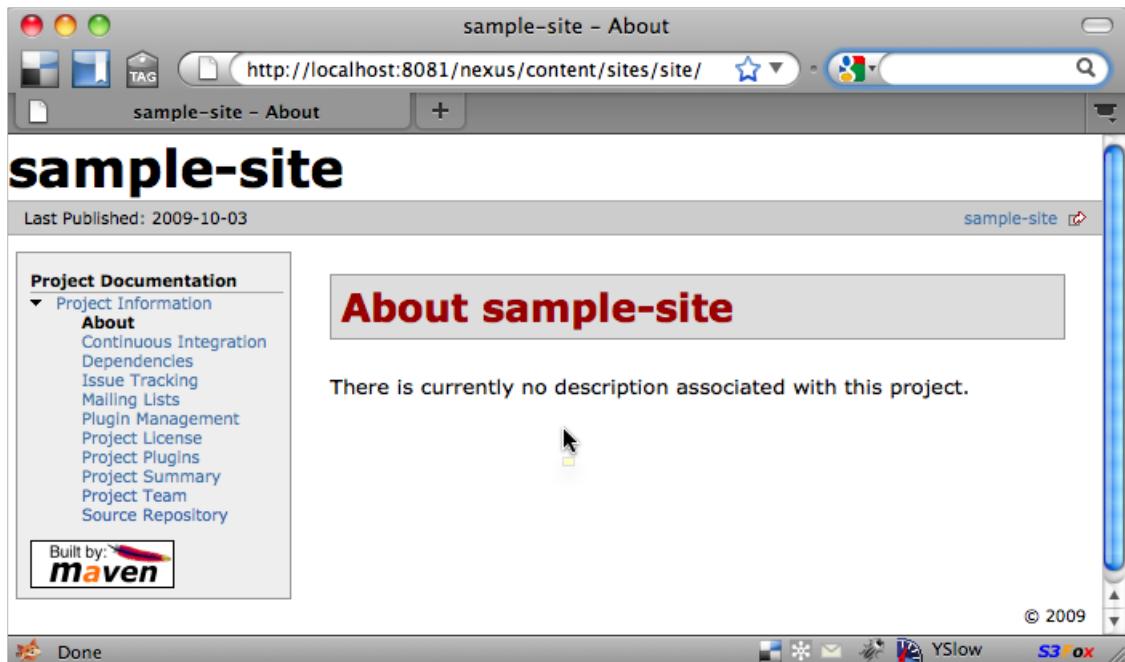


Figure 17.5: Sample Site Maven Project Web Site

Chapter 18

Artifact Bundles

18.1 Introduction

Artifact bundles are groups of related artifacts which are all related by the same groupId, artifactId, and version (GAV) coordinate. They are used by projects that wish to upload artifacts to the Central Repository.

Bundles must contain the following POM elements:

- modelVersion
 - groupId
 - artifactId
 - packaging
 - name
 - version
 - description
 - url
 - licenses
-

- scm
 - url
 - connection

18.2 Creating an Artifact Bundle from a Maven Project

Artifact bundles are created with the Maven Repository Plugin. For more information about the Maven Repository plugin, see <http://maven.apache.org/plugins/maven-repository-plugin/>

[Sample POM Containing all Required Bundle Elements](#), lists a project's POM which satisfies all of the constraints that are checked by the Maven Repository plugin. The following POM contains, a description and a URL, SCM information, and a reference to a license. All of this information is required before an artifact bundle can be published to the Maven Central repository.

Sample POM Containing all Required Bundle Elements

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.sonatype.sample</groupId>
  <artifactId>sample-project</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>sample-project</name>
  <description>A Sample Project for the Nexus Book</description>
  <url>http://books.sonatype.com</url>
  <licenses>
    <license>
      <name>The Apache Software License, Version 2.0</name>
      <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
      <distribution>repo</distribution>
    </license>
  </licenses>
  <scm>
    <connection>
      scm:git:git://github.com/sonatype/sample-project.git
    </connection>
    <url>http://github.com/sonatype/sample-project.git</url>
    <developerConnection>
      scm:git:git://github.com/sonatype-sample-project.git
    </developerConnection>
  </scm>

```

```
</developerConnection>
</scm>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
</project>
```

To create a bundle from a Maven project, run the repository:bundle-create goal. This goal will check the POM to see if it complies with the standards for publishing a bundle to a public repository, it will then bundle all of the artifacts generated by a particular build. To build a bundle that only contains the standard, unclassified artifact from a project, run mvn repository:bundle-create. To generate a bundle which contains more than one artifact, run mvn javadoc:jar source:jar repository:bundle-create

```
~/examples/sample-project$ mvn javadoc:jar source:jar repository:bundle- ↵
  create
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'javadoc'.
[INFO]   ←
----- ←
[INFO] Building sample-project
[INFO]   task-segment: [javadoc:jar, source:jar, repository:bundle-create ←
  ]
[INFO]   ←
----- ←
[INFO] [javadoc:jar {execution: default-cli}]
Loading source files for package com.sonatype.sample...
Constructing Javadoc information...
Standard Doclet version 1.6.0_15
Building tree for all the packages and classes...
...
[INFO] Preparing source:jar
[INFO] No goals needed for project - skipping
[INFO] [source:jar {execution: default-cli}]
...
----- ←
```

TESTS

```
Running com.sonatype.sample.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.03 sec
```

```
Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: ~/temp/sample-project/target/sample-project-1.0.jar
[INFO] [repository:bundle-create {execution: default-cli}]
[INFO] The following files are marked for inclusion in the repository ←
  bundle:

0.) Done
1.) sample-project-1.0.jar
2.) sample-project-1.0-javadoc.jar
3.) sample-project-1.0-sources.jar

Please select the number(s) for any files you wish to exclude, or '0' when ←
  \
you're done. Separate the numbers for multiple files with a comma (',').

Selection:
0
[INFO] Building jar: ~/temp/sample-project/target/sample-project-1.0- ←
  bundle.jar
[INFO] ←
----- ←

[INFO] BUILD SUCCESSFUL
[INFO] ←
----- ←

[INFO] Total time: 11 seconds
[INFO] Finished at: Sat Oct 10 21:24:23 CDT 2009
[INFO] Final Memory: 36M/110M
[INFO] ←
----- ←
```

Once the bundle has been created, there will be a bundle JAR in the target/ directory. As shown in the following command output, the bundle JAR contains: a POM, the project's unclassified artifact, the javadoc artifact, and the sources artifact.

```
~/examples/sample-project$ cd target
~/examples/sample-project/target$ jar tvf sample-project-1.0-bundle.jar
0 Sat Oct 10 21:24:24 CDT 2009 META-INF/
98 Sat Oct 10 21:24:22 CDT 2009 META-INF/MANIFEST.MF
1206 Sat Oct 10 21:23:46 CDT 2009 pom.xml
2544 Sat Oct 10 21:24:22 CDT 2009 sample-project-1.0.jar
20779 Sat Oct 10 21:24:18 CDT 2009 sample-project-1.0-javadoc.jar
```

```
891 Sat Oct 10 21:24:18 CDT 2009 sample-project-1.0-sources.jar
```

18.3 Uploading an Artifact Bundle to Nexus

To upload an artifact bundle to Nexus Professional you have to have a repository target for the project configured as described in Section [6.9](#).

Once that is done, select Staging Upload from the Build Promotion section of the Nexus menu as shown in Figure 18.1.



Figure 18.1: Build Promotion Menu

Selecting the Staging Upload link will load the Staging Upload dialog. Choose Artifact Bundle from the Upload Mode drop-down the Staging Upload panel will switch to the form shown in Figure 18.2. Click on Select Bundle to Upload... and then select the JAR that was created with the Maven Repository plugin used in the previous sections. Once a bundle is selected, click on Upload Bundle.

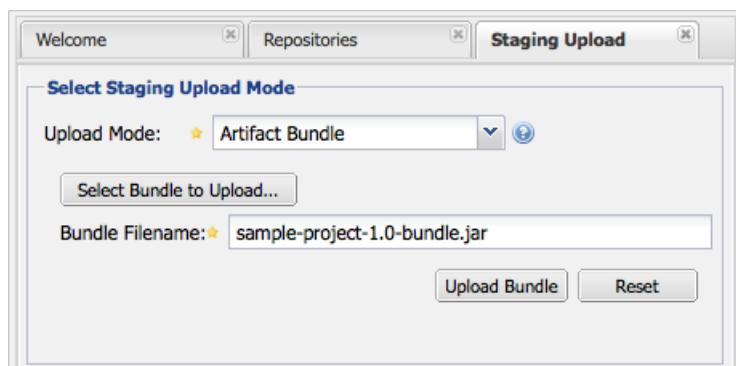


Figure 18.2: Uploading an Artifact Bundle

After a successful upload, a dialog displays the name of the created staging repository in a URL that links to the content of the repository. To view the staging repository, click on the Staging Repositories link in the Build Promotion section of the Nexus menu as shown in Figure 18.1, and you should see that the Staging Artifact Upload created and closed a new staging repository as shown in Figure 18.3. This repository contains all of the artifacts contained in the uploaded bundle. It allows you to promote or drop the artifacts contained in a bundle as a single unit.

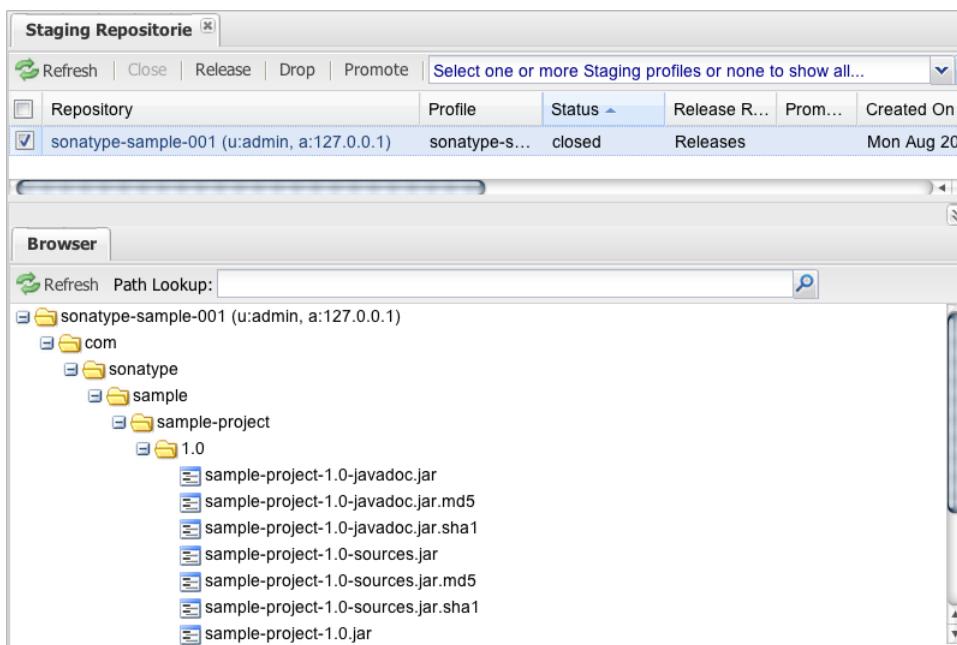


Figure 18.3: Staging Repository Created from Artifact Bundle Upload

Once the staging repository is closed you can promote it to a Build Promotion Profile or release it to the target repository of the staging profile as documented in Section 10.4.

Chapter 19

Nexus Best Practises

19.1 Introduction

Once you decide to install a Repository Manager, the next decision is how to setup your repositories, particularly if you have multiple teams sharing the same instance. Nexus is very flexible in this area and supports a variety of configurations. I'll first describe the options and then discuss the thought process used to decide what makes sense for your organization.

19.2 Repositories per Project/Team

The first and most obvious way to support multiple teams is to configure a pair of repositories per team (one release, one snapshot). The team is then given the appropriate C.R.U.D. permissions and they are able to use the system for their artifacts.

Our <http://oss.sonatype.org> instance is for the most part configured in this manner, where each project like Jetty has their own repositories separate from everyone else.

19.3 Partition Shared Repositories

Another option is to have a single (or a few) pair of Release/Snapshot repositories for your entire organization. In this case, the access is controlled by a mechanism we call "Repository Targets."

Simply put, a Repository Target is a way to manage a set of artifacts based on their paths. A Repository Target is simply a list of regular expressions and a Name. For example, a Repo Target for Maven would be `./org/apache/maven/`. and Nexus OSS would be `./org/sonatype/nexus/`.

Note

While it is most common to manage artifacts based on the path of their groupId, the Regular Expression is matched against the entire path, and so it is also possible, for example, to define "Sources" as `.*-sources.jar` . . . it's also worth noting that Repository Targets are not mutually exclusive. It is perfectly valid for a given path to be contained by multiple targets.

In this model, you would create a Repo Target for each project in your system. You are then able to take the Repo Target and associate it with one or more Repositories or Groups in your system. When you do this, new, specific, C.R.U.D. privileges are created. For example, I could take the Maven Repo target, associate it with my Release and Snapshot repository, and now I get privileges I can assign to Create, Read, Update, Delete "Maven" (`./org/apache/maven/`) artifacts in my Release and Snapshot repositories.

This method is used to manage the <http://repository.apache.org> instance, where we have just one Release and Snapshot repository and each project team gets permissions to their artifacts based on the path.

19.3.1 Selecting an Approach

First of all, these choices aren't mutually exclusive. In fact, the first option builds upon the default Repository Target of `".*"` which simply gives you access to all artifacts regardless of the path. You still associate the default Repo Target with specific repositories to create the assignable privileges

In general, it's my opinion that fewer repositories will scale better and are easier to manage. It's also easier to start off with a single pair of repositories, with the default "All M2″ (`.*`) target and simply refine the permissions as you scale. Most things that are configured per repository (Cache, Storage location, Snapshot purging, etc) will generally be applicable for all projects, so this mode avoids the duplication of these tasks. Since everything will be stored together in a single folder on disk, it makes backups easier as well.

The reasons why you would want multiple sets of repositories is essentially the opposite of above: If you need different expiration, Snapshot purging or storage folders, then a single shared repo won't work. Replication and fail-over strategies may also make this method easier to support. If you absolutely must maintain total separation between Project teams, i.e. they can't read each other's artifacts, then this solution might be more applicable as well. (but is still possible with Repo Targets...just grant Read to only the appropriate targets)

In Summary, Nexus allows you to control the security of your artifacts based on the repository and/or the path of the artifact, meaning it is possible to slice and dice the system any way you see fit. My default position is to use a few Hosted Repositories as possible and control the permissions by the Repository Target.

Chapter 20

Using Nexus Plugins

Nexus Open Source and Nexus Professional are built using a plugin architecture, where Professional simply includes additional plugins. Besides the default installed plugins you can install optional plugins shipped with Nexus as well as plugins you or somebody else created.

20.1 Installing Additional Plugins

Optional plugins supplied by Sonatype can be found in the directory \$NEXUS_HOME/nexus/WEB-INF/optional-plugins. To install any of these simply copy the folder containing the desired plugin into \$NEXUS_HOME/nexus/WEB-INF/plugin-repository.



Warning

When updating Nexus you will have to redo the install of any optional plugins using the newest version shipping with the download of the new Nexus version. Any configuration of the plugin will be preserved from one version to the other.

Plugins supplied by third parties like the [Nexus Yum Plugin](#) and others, are installed by copying the folder with the plugin code into sonatype-work/nexus/plugin-repository.

After a restart of Nexus the new plugins will be active and ready to use. Upgrades are done by simply copying over the newer plugin and removing the older one.

Plugins can be removed by deleting the respective folder in the plugin-repository and restarting Nexus.

20.2 Nexus Outreach Plugin

The Nexus Outreach Plugin is installed by default in Nexus Open Source and Nexus Professional. It allocates space in the user interface left navigation and welcome page for linking to further documentation and support resources. This data is retrieved from Sonatype servers.

In a case where this outgoing traffic from your Nexus instance or the resulting documentation and links are not desired, the plugin can be disabled. The plugin can be disabled by disabling the advanced *Outreach : Management* capability in the Capabilities section found in the Administration left hand navigation menu.

You can safely remove the plugin as well without any other negative side effects. To do so simply remove the *nexus-outreach-plugin-X.Y.Z* folder in *\$NEXUS_HOME/nexus/WEB-INF/plugin-repository/* and restart your Nexus instance.

20.3 Custom Metadata Plugin

The custom metadata plugin is an optional plugin that comes as part of any Nexus download. It allows you to view, edit and search for additional metadata about any artifact in your Nexus repositories.

The directory containing the plugin code is called *nexus-custom-metadata-plugin-X.Y.Z*. Install the plugin following the instructions in Section [20.1](#).

20.3.1 Viewing Artifact Metadata

The Custom Metadata Plugin gives you the ability to view artifact metadata. When browsing repository storage or a repository index, clicking on an artifact will load the Artifact Information panel. Selecting the Artifact Metadata tab will display the interface shown in Figure [20.1](#).

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

Figure 20.1: Viewing Artifact Metadata

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

20.3.2 Editing Artifact Metadata

The Custom Metadata Plugin gives you the ability to add custom attributes to artifact metadata. To add a custom attribute, click on an artifact in Nexus, and select the Artifact Metadata tab. On the Artifact Metadata tab, click on the Add... button and a new row will be inserted into the list of attributes. Supply a key and value and click the Save button to update an artifact's metadata. Figure 20.2 shows the Artifact Metadata panel with two custom attributes: "approvedBy" and "approved".

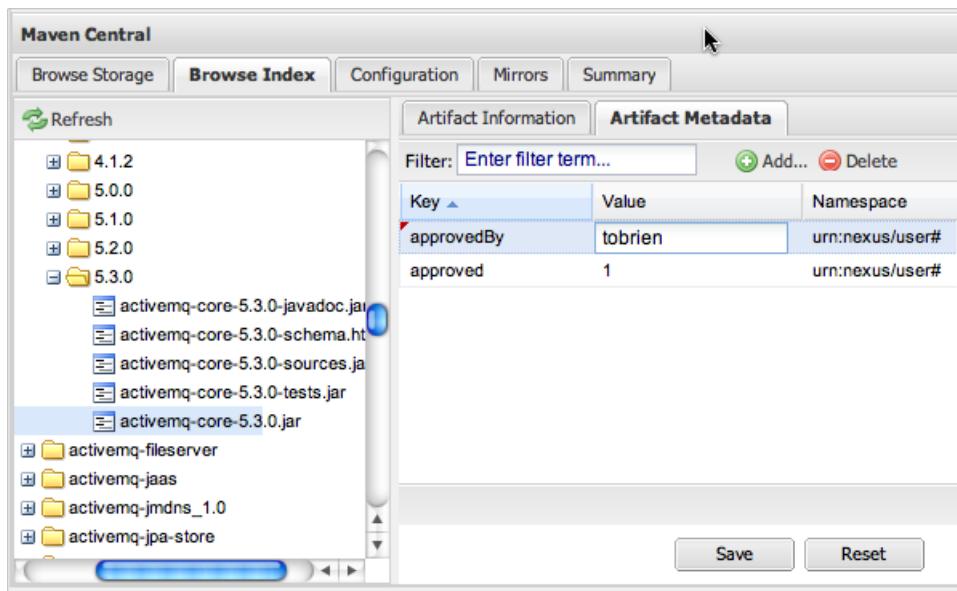


Figure 20.2: Editing Artifact Metadata

20.3.3 Searching Artifact Metadata

The Custom Metadata Plugin provides you with the ability to configure custom artifact metadata and search for artifacts with specific metadata. To search for artifacts using metadata, click on the Advanced Search link directly below the search field in the Nexus application menu to open the Search panel. Once in the search panel, click on the Keyword Search and click on Metadata Search in the search type drop-down as shown in Figure 20.3.

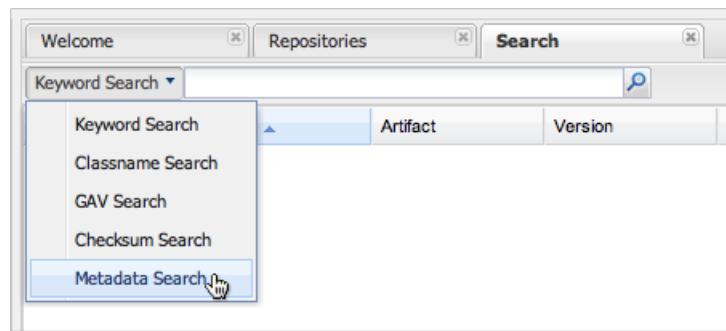


Figure 20.3: Searching Artifact Metadata

Once you select the Metadata Search you will see two search fields and an operator drop-down. The two search fields are the key and value of the metadata you are searching for. The key corresponds to the key of the metadata you are searching for, and the value contains the value or value range you are searching for. The operator drop-down can be set to Equals, Matches, Bounded, or Not Equal.

A screenshot of the Nexus search results for custom metadata. The search interface is similar to Figure 20.3, with tabs for 'Welcome', 'Repositories', and 'Search'. The 'Search' tab is active. The search criteria are set to 'Metadata Search' with 'Key: approved' and 'Value: 1'. The results table shows two records from the 'Maven Central (hosted)' index. The first record is 'abbot' (Group: abbot, Artifact: abbot, Version: 0.12.3, Packaging: jar). The second record is 'activemq-core' (Group: org.apache.activemq, Artifact: activemq-core, Version: 5.3.0, Packaging: jar). Below the results, it says 'Displaying 2 records' and 'Clear Results'. At the bottom, there are tabs for 'Artifact Information' and 'Artifact Metadata'. Under 'Artifact Information', the values are: Group: org.apache.act, Artifact: activemq-core, Version: 5.3.0, Download: pom, artifact. Under 'Artifact Metadata', the XML code is displayed: <dependency> <groupId>org.apache.activemq</groupId> <artifactId>activemq-core</artifactId> <version>5.3.0</version> </dependency>. A scroll bar is visible on the right side of the results table.

Figure 20.4: Metadata Search Results for Custom Metadata

Once you locate a matching artifact in the Metadata Search interface, click on the artifact and then select the Artifact Metadata to examine an artifacts metadata as shown in Figure 20.5.

The screenshot shows the Nexus Metadata Search interface. At the top, there is a search bar with 'approvedBy' as the key and 'to*' as the value. Below the search bar, a table displays search results for Maven Central, showing a single record for org.apache.activemq.activemq-core version 5.3.0. The table has columns for Source Index, Group, Artifact, Version, and Packaging. Below the table, it says 'Displaying 1 records'. Underneath, there are two tabs: 'Artifact Information' and 'Artifact Metadata'. The 'Artifact Metadata' tab is selected. It contains a table with columns for Key, Value, and Namespace. The table lists various metadata entries such as approved (value 1, namespace urn:nexus:user#), approvedBy (value tobrien, namespace urn:nexus:user#), artifactId (value activemq-core, namespace urn:maven#), and so on. There are buttons for 'Save' and 'Reset' at the bottom.

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

Figure 20.5: Metadata Search Results for Custom Metadata

20.4 User Account Plugin

Nexus Professional's User Account Plugin allows anonymous users to sign-up for a Nexus account without administrative intervention. This "self-serve" capability for user account creation is especially important when an open source project is using Nexus as a primary means for distribution. In such an environment, there may be hundreds of Nexus users who all need a basic level of read-only access, and making these users wait for an administrator to create an account makes little sense. In such a setting,

anyone can create an account, activate the account via a verification email, and then access a repository with a default level of access you can define.

20.4.1 Installing the User Account Plugin

The user account plugin is an optional plugin that comes as part of the Nexus Professional download. The directory containing the plugin code is called `nexus-user-account-plugin-X.Y.Z`. Install the plugin following the instructions in Section [20.1](#).

20.4.2 Configuring the User Account Plugin

After a restart of Nexus the installed plugin will be available for use but still be deactivated. To configure the User Account Plugin, click on Server under the Administration section of the Nexus menu, and scroll down to the section named "User Sign Up". The User Sign Up section is shown in Figure [20.6](#).

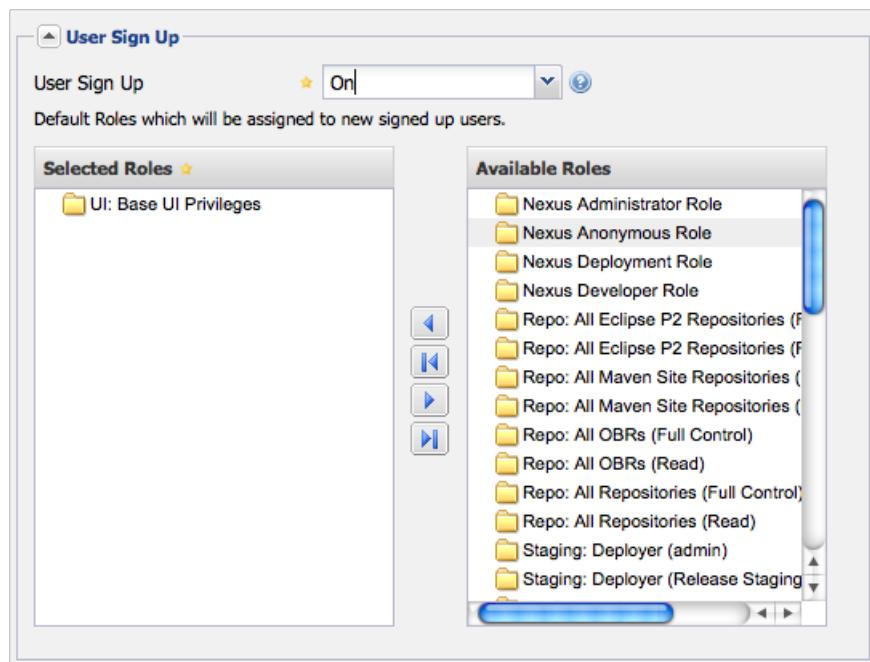


Figure 20.6: Configuring the User Account Plugin

To activate the User Sign Up feature, set the "User Sign Up" feature to "On". This will expose a "Sign Up" link next to the "Log In" link in the Nexus user interface. The Selected Roles in this configuration section are the default roles assigned to users who successfully signed up for an account.

In addition it is necessary for the anonymous user to have the role "UI: User Account Sign Up" assigned. This should be automatically configured during the plugin install. You can use the Role Tree tab for the anonymous user to inspect if that is correctly configured, and if necessary add the role

20.4.3 Signing Up for an Account

Once User Sign Up has been activated via the Server settings as shown in Section 20.4.2, users will see a Sign Up link next to the Log In link in the top right hand corner of the Nexus interface. Clicking on this Sign Up link will display the Nexus Sign Up dialog shown in Figure 20.7. This form accepts a username, password, the first and last name of the new user, and an email account. It also asks the users to type in some text from a captcha form element. If a user cannot read the text in the captcha, they can click on the captcha to refresh it with new text.

The screenshot shows the "Nexus Sign Up" dialog box. It contains fields for entering user information and acaptcha. The fields are as follows:

- Username: mmoser
- Password: (represented by four dots)
- Confirm Password: (represented by four dots)
- First Name: Manfred
- Last Name: Moser
- Email: mmoser@sonatype.com
- Type the text you see below: wfe7m

Below the text input field is a CAPTCHA image containing the text "wfe7m" with a diagonal line through it. To its left is the text "Too blurred? Click to change it:". At the bottom of the dialog is a "Sign Up" button.

Figure 20.7: Nexus Sign Up Form

Once the new user clicks on the Sign Up button, they will receive a confirmation dialog which instructs them to check for an activation email.



Figure 20.8: Nexus Sign Up Confirmation

The user will then receive an email containing an activation link.



Warning

The SMTP settings in your Nexus Server configuration need to be set up for the activation email to be sent successfully.

When a user signs up for a Nexus account, the newly created account is disabled until they click on the activation link contained in this email. A sample of the activation email is shown in Figure 20.9.

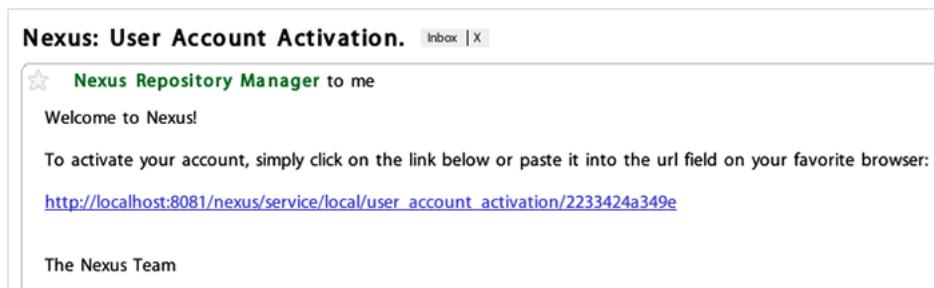


Figure 20.9: Nexus Activation Email

Upon successful login from the activation email link, the user will be directed to the Summary panel of the users Profile.

Note

The example activation email in Figure 20.9, points to localhost:8081. You can change this URL by changing the Base URL setting in the Application Server Settings section of the Server configuration. To change this setting, click on the Server link under Administration in the Nexus menu.

20.4.4 Manual Activation of New Users

If a user does not receive the activation email after signing up for a new account, an Administrator may need to manually activate a new user. To do this, go to the list of Nexus users by clicking on the Users link under Security in the Nexus menu. Locate and select the new user in the list of Nexus users, and change the Status from Disabled to Enabled as shown in Figure 20.10.

The screenshot shows the Nexus UI interface. At the top, there's a navigation bar with tabs for 'Welcome' and 'Users'. Below the navigation bar is a toolbar with icons for Refresh, Add..., Delete, and All Configured Users, along with a search bar. The main area displays a table of users:

User ID	Realm	First Name	Last Name	Email	Role
admin	default	Administrator		changeme@yourcompany.com	Nex
deployment	default	Deployment	User	changeme1@yourcompany.com	Re
mmoser	default	Manfred	Moser	manfred@simpligility.com	UI:
anonymous	default	Nexus	Anonymous User	changeme2@yourcompany.com	UI:

Below the user table, a specific user profile for 'mmoser' is shown. The profile includes fields for User ID (mmoser), First Name (Manfred), Last Name (Moser), Email (manfred@sonatype.com), and Status (Active). A dropdown menu for Status shows 'Active' and 'Disabled'. Under 'Role Management', it lists 'UI: Base UI Privileges'. At the bottom of the profile view are 'Save' and 'Reset' buttons.

Figure 20.10: Manually Activating a Signed Up User

20.4.5 Modifying Default User Permissions

The default user permissions in the User Sign Up feature only includes "UI: Base UI Privileges". If a user signs up with just this simple permission, the only thing they will be able to do is login, change their password, and logout. Figure 20.11, shows the interface a user would see after logging in with only the base UI privileges.



Figure 20.11: User Interface with only the Base UI Privileges

To provide some sensible default permissions, click on the Server under the Administration section of the Nexus menu and scroll down to the User Sign Up section of the Server settings. Make sure that the selected default roles for new users contain some ability to browse, search, and view repositories.

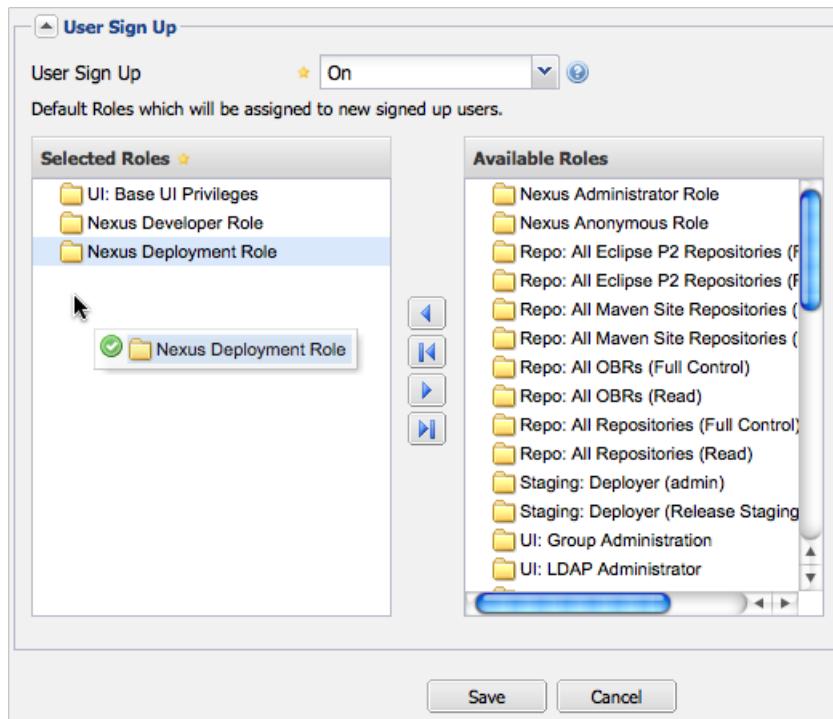


Figure 20.12: Selecting Default Roles for New Users

Warning

Figure 20.12 shows a default User Sign Up role containing the Nexus Deployment Role. If your server were available to the public this wouldn't be a wise default role as it would allow anyone to sign up for an account, activate an account, and start publishing artifacts to hosted repositories with little or no oversight. Such a default role may only make sense if you are running an internal, corporate instance of Nexus Professional and you are comfortable granting any developer in the organization deployment permissions.

20.5 Nexus Atlassian Crowd Plugin

Atlassian's Crowd is a single sign-on and identity management product that many organizations use to consolidate user accounts and control which users and groups have access to which applications. Nexus Professional contains an optional security plugin that allows you to configure Nexus to authenticate against an Atlassian Crowd instance. For more information about Atlassian Crowd, go to <http://www.atlassian.com/software/crowd/>

20.5.1 Installing the Crowd Plugin

The Nexus Atlassian Crowd plugin is an optional plugin that comes as part of any Nexus Professional download. The directory containing the plugin code is called enterprise-crowd-plugin-X.Y.Z. Install the plugin following the instructions in Section 20.1.

20.5.2 Configuring the Crowd Plugin

Once the Atlassian Crowd plugin is installed, restart Nexus and login as a user with Administrative privileges. To configure the Crowd plugin, click on the Crowd Configuration in the Security section of the Nexus menu as shown in Figure 20.13.



Figure 20.13: Crowd Menu Link in the Security Section of the Nexus Menu

Clicking on the Crowd Configuration link will load the form shown in Figure 20.14. This configuration panel contains all of the options that need to be configured to connect your Nexus instance to Crowd for authorization and authentication.

A screenshot of the "Crowd Configuration" dialog box. It has tabs for "Access Settings", "HTTP Settings", "HTTP Proxy Settings", and "Miscellaneous Settings". Under "Access Settings", fields include "Application Name" (nexus), "Application Password" (redacted), "Crowd Server URL" (http://localhost:8095/crowd/), "Authentication Interval" (30), and "Use Groups" (checked). Buttons at the bottom include "Test Connection", "Save", and "Cancel".

Figure 20.14: Crowd Configuration Panel

The following sections outline all of the settings in the Crowd Configuration Pane.

20.5.3 Crowd Access Settings

The Access Settings section of the Crowd configuration is shown in Figure 20.15. This section contains the following fields:

Application Name

This field contains the application name of a Crowd application. This value should match the value in the Name field of the form shown in Figure 20.20.

Application Password

This field contains the application password of a Crowd application. This value should match the value in the Password field of the form shown in Figure 20.20.

Crowd Server URL

This is the URL of the Crowd Server, this URL should be accessible to the Nexus process as it is the URL that Nexus will use to connect to Crowd's SOAP services.

Authentication Interval

This is the number of minutes that a Crowd authentication is valid for. This value is in units of minutes, and a value of 30 means that Nexus will only require re-authentication if more than 30 minutes have elapsed since a previously authenticated user has accessed Nexus.

Use Groups

If clicked, Use Groups allows Nexus to use Crowd Groups when calculating Nexus Roles. When selected, you can map a Nexus Role to a Crowd Group.

Access Settings	
Application Name	<input type="text" value="nexus"/> ⓘ
Application Password	<input type="password" value="*****"/> ⓘ
Crowd Server URL	<input type="text" value="http://localhost:8095/crowd/"/> ⓘ
Authentication Interval	<input type="text" value="30"/> ⓘ
Use Groups	<input checked="" type="checkbox"/> ⓘ

Figure 20.15: Crowd Access Settings

20.5.3.1 Crowd HTTP Settings

You can control the concurrency of connections to Crowd in the HTTP Settings section shown in Figure 20.16. If you have a high-traffic instance of Nexus, you will want to limit the number of simultaneous connections to the Crowd server to a reasonable value like 20. The HTTP Timeout specifies the number of milliseconds Nexus will wait for a response from Crowd. A value of zero for either of these properties indicates that there is no limit to either the number of connections or the timeout.



Figure 20.16: Crowd HTTP Settings

20.5.3.2 Crowd HTTP Proxy Settings

If your Nexus installation is connecting to Crowd via an HTTP Proxy server, the HTTP Proxy Settings section of the Crowd Configuration allows you to specify the host, port, and credentials for a HTTP Proxy server. The HTTP Proxy Settings section is shown in Figure 20.17.

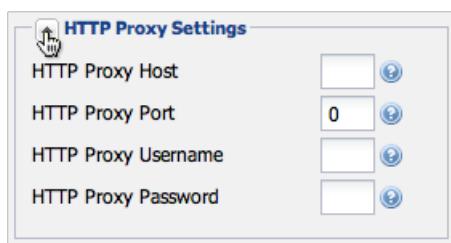


Figure 20.17: Crowd HTTP Proxy Settings

20.5.3.3 Miscellaneous Settings

The miscellaneous settings section shown in Figure 20.18, allows you to configure settings that control the name of the Single Sign-on cookie and the various keys that are used to retrieve values that relate

to authentication and the auth token. This dialog is only relevant if you have modified optional Crowd settings in your \$CROWD_HOME/etc/crowd.properties. For more information about customizing these options see the [Atlassian Crowd documentation](#)

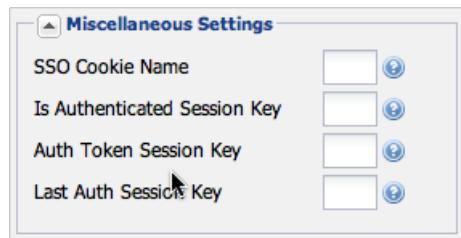


Figure 20.18: Crowd Miscellaneous Settings

20.5.4 Adding the Crowd Authentication Realm

Once you have configured Nexus to connect to Crowd, you must select the Crowd authorization realm from the list of available realms in your Nexus Server settings. Figure 20.19, shows the Security settings section in the Nexus Server configuration panel. To load the Nexus server configuration panel, click on Server under Administration in the Nexus menu. Drag Crowd from the list of available realms to the list of selected realms and then save the Nexus server configuration.

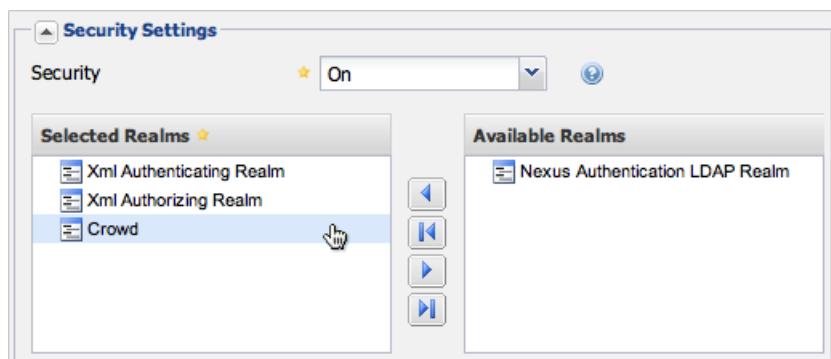


Figure 20.19: Configuring the Crowd Authentication Realm

20.5.5 Configuring a Nexus Application in Crowd

To connect Nexus to Atlassian's Crowd, you will need to configure Nexus as an application in Crowd. To do this, login to Crowd as a user with Administrative rights, and click on the Applications tab. Once you click on this tab, you should see two options under the Applications tab: Search Applications and Add Application. Click on Add Application to display the form shown in Figure 20.20, and create a new application with the following values in the Details tab of the Add Application form:

- Application Type: Generic Application
- Name: nexus
- Description: Sonatype Nexus Professional

Choose a password for this application. Nexus will use this password to authenticate with the Crowd server. Click on the Next button.

The screenshot shows the 'Add Application' dialog box. The 'Details' tab is active. The 'Application Type' dropdown is set to 'Generic Application'. The 'Name' field contains 'nexus'. The 'Description' field contains 'Sonatype Nexus Professional'. Both the 'Password' and 'Confirm Password' fields contain masked text. At the bottom are 'Next >' and 'Cancel' buttons.

Figure 20.20: Creating a Nexus Crowd Application

Clicking on Next will advance the form to the Connection tab shown in Figure 20.21. In this tab you need to supply the URL Nexus and the remote IP address for Nexus. Figure 20.21, shows the Connection

form configured for a local instance of Nexus. If you were configuring Crowd and Nexus in a production environment, you would supply the URL that users would use to load Nexus in a web browser and you would supply the IP address that Nexus will be connecting from. Once you have completed the Connection form, click on Next to advance to the Directories form.

The screenshot shows the 'Add Application - nexus' interface. At the top, there are five tabs: 1. Details, 2. Connection, 3. Directories, 4. Authorisation, and 5. Confirmation. The '2. Connection' tab is selected. Below the tabs, there are two input fields: 'URL:' containing 'http://localhost:8081/nexus' and 'Remote IP Address:' containing '127.0.0.1'. To the right of the URL field is a 'Resolve IP Address' button. Below each field is a descriptive tooltip. At the bottom are 'Next »' and 'Cancel' buttons.

Figure 20.21: Creating a Nexus Crowd Application Connection

Clicking on Next advances to the Directories form shown in Figure 20.22. In this example, the Nexus application in Crowd is going to use the default "User Management" directory. Click on the Next button to advance to the "Authorisation" form.

The screenshot shows the 'Add Application - nexus' interface at the '3. Directories' step. It asks to select directories for authentication and authorisation. A single checkbox labeled 'User Management' is checked, with a tooltip explaining it's a user management directory. At the bottom are 'Next »' and 'Cancel' buttons.

Figure 20.22: Creating a Nexus Crowd Application Directories

Clicking on the Next button advances to the "Authorisation" form shown in Figure 20.23. If any of the directories selected in the previous form contain groups, each group is displayed on this form next to a

checkbox. You can select "Allow all users" for a directory, or you can select specific groups which are allowed to authenticate to Crowd through Nexus. This option would be used if you wanted to limit Nexus access to specific subgroups within a larger Crowd directory. If your entire organization is stored in a single Crowd directory, you may want to limit Nexus access to a group that contains only Developers and Administrators.

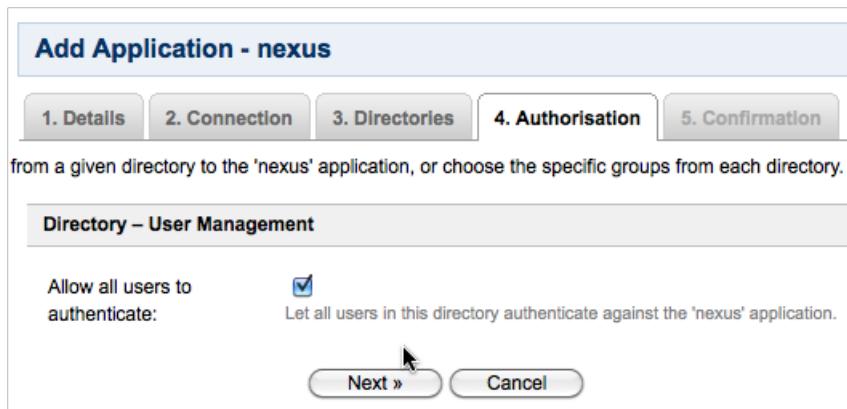
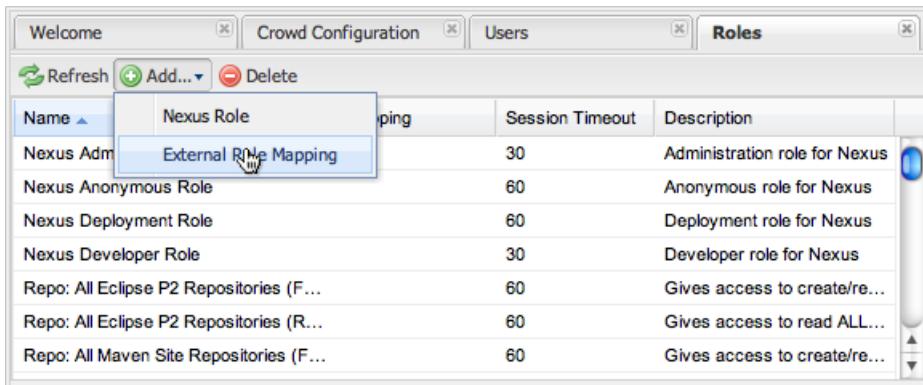


Figure 20.23: Creating a Nexus Crowd Application Authorization

20.5.6 Mapping Crowd Groups to Nexus Roles

To map a Crowd Group to a Nexus Role, open up the Roles panel by clicking on the Roles link under the Security section of the Nexus menu. Click on the Add... button and select External Role Mapping as shown in Figure 20.24.



Name	Nexus Role	Mapping	Session Timeout	Description
Nexus Admin	External Role Mapping		30	Administration role for Nexus
Nexus Anonymous Role			60	Anonymous role for Nexus
Nexus Deployment Role			60	Deployment role for Nexus
Nexus Developer Role			30	Developer role for Nexus
Repo: All Eclipse P2 Repositories (F...			60	Gives access to create/re...
Repo: All Eclipse P2 Repositories (R...			60	Gives access to read ALL...
Repo: All Maven Site Repositories (F...			60	Gives access to create/re...

Figure 20.24: Adding an External Role Mapping

Selecting External Role Mapping will show the Map External Role dialog shown in Figure 20.25.

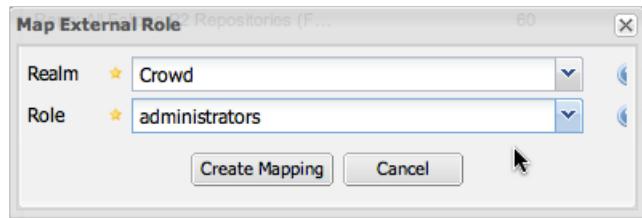


Figure 20.25: Mapping an External Crowd Group to a Nexus Role

Once you have mapped a Crowd Group to a Nexus Role, these Roles will appear in the list of Nexus Roles with a mapping value of "Crowd" as shown in Figure 20.26.

The screenshot shows the 'Roles' configuration screen in the Nexus interface. At the top, there are tabs for 'Welcome', 'Crowd Configuration', 'Users', and 'Roles'. Below the tabs is a toolbar with 'Refresh', 'Add...', and 'Delete' buttons. A main table lists roles with columns for 'Name', 'Mapping', 'Session Timeout', and 'Description'. Two rows are selected: 'administrators' (Crowd mapping) and 'developers' (Crowd mapping). Below the table, a section titled 'administrators' contains configuration details: Role Id (administrators), Name (administrators), Description (External mapping for administrators (Crowd)), and Session Timeout (60). On the left, a 'Selected Roles / Privileges' panel shows 'Nexus Administrator Role'. On the right, an 'Available Roles / Privileges' panel lists several options, including 'Nexus Anonymous Role', 'Nexus Deployment Role', 'Nexus Developer Role', 'Repo: All Eclipse P2 Repositories (Full Control)', and 'Repo: All Eclipse P2 Repositories (Read)'. At the bottom are 'Save' and 'Reset' buttons.

Figure 20.26: Two Crowd Groups Mapped to Nexus Roles

20.5.7 Adding a Crowd Role to a Nexus User

To illustrate this feature, consider the crowd-manager user with an id of "brian". This user's groups are shown in Figure 20.27.

The screenshot shows the Crowd web interface. At the top, there's a navigation bar with links for Applications, Users, Groups, Roles, Directories, and Administration. The User: Default Admin link is visible. Below the navigation bar, on the left, is a sidebar with links for Search Users, Add User, Import Users, Reset Password, and Remove User. The main content area is titled "View User – brian". It has tabs for Details, Attributes, Groups (which is selected), Roles, and Applications. A sub-section titled "These are the groups the user is a member of." displays a table with one row: Group (developers), Description (Developers), and Active (true). At the bottom of this section are "Add Groups" and "Remove Groups" buttons.

Figure 20.27: Crowd Groups for User "brian"

To add an external user role mapping, open up the Users panel by clicking on Users in the Security section of the Nexus panel. Click on the Add... button and select External User Role Mapping from the drop-down as shown in Figure 20.28.

The screenshot shows the Nexus interface with a tab bar for Welcome, Crowd Configuration, Users, and Roles. The Users tab is active. Below the tab bar is a toolbar with Refresh, Add..., Delete, and All Configured Users buttons. A search bar says "Select a filter to enable username search". The main content is a table with columns for User ID, Name, Email, and Roles. The first row shows admin with the name "Nexus User". The second row shows deployment with the name "External User Role Mapping". The third row shows anonymous with the name "Nexus Anonymous User". The "External User Role Mapping" row is highlighted with a blue background and has a cursor pointing at it. The table also includes a header row with column names.

Figure 20.28: Adding an External User Role Mapping

Selecting External User Role Mapping will show the dialog shown in Figure 20.29.

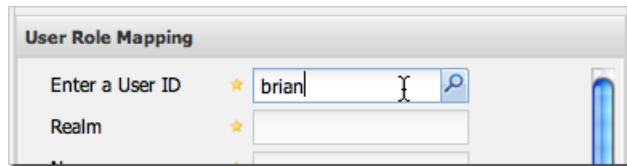


Figure 20.29: Locating a Crowd User in the User Role Mapping Dialog

Once you locate the Crowd user that you want to add a Nexus Role to... You can use the configuration panel shown in Figure 20.30, to add a Role to the Crowd-managed "brian" user.

A screenshot of the Crowd Configuration interface. The top navigation bar shows "Welcome", "Crowd Configuration", "Users", and "Roles". Below the navigation is a table titled "All Configured Users" with columns: User ID, Realm, Name, Email, and Roles. The table data is as follows:

User ID	Realm	Name	Email	Roles
admin	default	Administrator	changeme@yourcompany.com	Nexus...
brian	Crowd	Brian Fox	brian@sonatype.com	Nexus...
deployment	default	Deployment User	changeme1@yourcompany.com	Repo...
anonymous	default	Nexus Anonymous User	changeme2@yourcompany.com	Repo...

A detailed view of the "Brian Fox" configuration dialog. It has tabs for "Config", "Role Tree", and "Privilege Trace". The "Config" tab is active. It displays the user's details: User ID (brian), Realm (Crowd), Name (Brian Fox), and Email (brian@sonatype.com).

Selected Roles *		Available Roles	
<input type="checkbox"/> developers	<input type="checkbox"/> Nexus Deployment Role	<input type="checkbox"/> administrators	<input type="checkbox"/> Nexus Administrator Role
<input type="checkbox"/> Nexus Deployment Role		<input type="checkbox"/> Nexus Anonymous Role	

At the bottom are "Save" and "Reset" buttons.

Figure 20.30: Adding a Nexus Role to a Crowd User

20.6 Nexus Branding Plugin

The branding plugin is an optional plugin that comes as part of Nexus Professional. It features the possibility to brand your Nexus instance by replacing the default Sonatype Nexus logo with your logo (or an image of your choice).

The new logo needs to be a PNG image. To blend in well in the UI, it is recommended that it is of 60 pixels height and has a transparent background.

By default, the branding plugin will look for the new logo in a file called

```
sonatype-work/nexus/conf/branding.png
```

However, it is possible to configure this path by adding a `branding.image.path` property to the `nexus.properties` file in `$NEXUS_HOME/conf/`:

```
branding.image.path=/data/images/nexus_logo.png
```

If it fails to find a new logo, the plugin will fall back to using the default Sonatype Nexus logo.

Chapter 21

Developing Nexus Plugins

Among the many benefits of using an technology with an open source core is the ability to customize behaviour and create extensions. To this end, Sonatype has spent a great deal of time designing an intuitive Plugin API that will allow you to take Nexus where you need it to go. This chapter summarizes some of these extension points and presents a walk through of how you would start to develop your own Nexus plugins.

Our community has already created a number of compelling and useful plugins, some of which have been integrated into the set of plugins that are distributed with both Nexus Open Source and Nexus Professional. Sonatype tried to make the Plugin API as lightweight and extensible as possible with the following goals in mind:

- Providing a clear set of extension points for plugin developers
 - Providing isolated plugin classpaths to avoid compatibility issues between plugins and to prevent a plugin from disturbing another, unrelated part of Nexus.
 - Giving developers the ability to load and unload Nexus plugins at runtime
-

21.1 Nexus Plugins

The Nexus API is contained in a module named `nexus-api`. If you are developing a Nexus plugin, you will need to familiarize yourself with the extension points that are defined in this project.

21.1.1 Nexus Plugin API

Nexus provides an extra module for plugin developers - the "nexus-plugin-api". This module provides some extra annotations for plugins developers, and it allows a plugin developer to implement a plugin without having to know anything about Plexus or Nexus internals.

The Nexus Plugin API uses the `@javax.inject.Inject` annotation, an emerging standard for dependency injection which allows Nexus plugins to be developed in a way that is container-neutral.

The plugin API also introduces some additional annotations to make things easier:

```
@org.sonatype.plugin.Managed
```

When a `@Managed` annotation is present on an interface, it marks the interface as "component contract" (Plexus role). Any non-abstract classes implementing it will be made managed by current container.

```
@org.sonatype.nexus.plugins.RepositoryType
```

Used on interfaces, to mark it as new repository type, and to be registered with other core repository types in Nexus Repository Type Registry. It holds the basic information about the new type (the path where to mount it).

```
@org.sonatype.nexus.plugins.RestResource
```

Used on classes, to mark them as REST Resources.

21.2 Nexus Extension Points

The simplest Nexus plugin contain a single class, SampleEventInspector, which contributes an EventInspector to the Nexus Application. This simple event inspector will do nothing more than print a message every time it accepts and inspects an event.

A Simple Event Inspector

```
package org.sample.plugin;

import org.sonatype.nexus.proxy.events.EventInspector;
import org.sonatype.plexus.appevents.Event;

public class SampleEventInspector implements EventInspector {
    public boolean accepts( Event<?> evt ) {
        return true;
    }

    public void inspect( Event<?> evt ) {
        System.out.println( "Invoked with event: " +
            evt.toString() + " with sender " +
            evt.getEventSender().toString() );
    }
}
```

During the build of this nexus plugin, this class is compiled and then scanned for concrete classes that implement extension point interfaces defined in the following section. The EventInspector interface in the nexus-api project has been marked with the @ExtensionPoint annotation. The plugin build takes the @ExtensionPoint, @Named, and @Inject annotations that may be present and generates a plugin descriptor which is packaged in the plugin's JAR.

When the plugin is present in Nexus during start-up, the Nexus plugin manager reads the plugin metadata and instantiates the appropriate components. To implement a plugin, you simply implement some of these interfaces.

21.3 Nexus Plugin Extension Points

The following sections outline the available Nexus extension points.

21.3.1 Nexus Plugin Extension

Interface: org.sonatype.nexus.plugins.NexusPlugin

This extension component is meant to be used in Nexus plugins only. If it is found in a plugin, it will be invoked during install/uninstall/init phases of a plugin installation/uninstallation/initialization. Typical usage would be a need to perform some specific tasks on plugin install (i.e. it uses native code to do some magic and those needs to be copied somewhere, register them with OS, etc).

21.3.2 Nexus Index HTML Customizer

Interface: org.sonatype.nexus.plugins.rest.NexusIndexHtmlCustomizer

This extension is able to customize the "index.html" returned by Nexus. Using this component, a plugin is able to add markup or Javascript to the pages generated by the Nexus web application. Every plugin that has a UI component uses this extension point to add Javascript customizations to the interface.

21.3.3 Static Plugin Resources

Interface: org.sonatype.nexus.plugins.rest.NexusResourceBundle

This extension gathers and publishes static resources over HTTP. These resources are usually JavaScript files, CSS files, images, etc. Plugin developers do not need to use this extension directly since some of the features it exposes are automatic for all plugins. When the Nexus plugin manager discovers resources in plugin JAR under the path "/static", the Plugin Manager will create a special "plugin NexusResourceBundle" component on the fly.

If you do not want the plugin manager to automatically add a resource bundle you can define your own resource bundle implementation. The plugin manager will not add a resource bundle if:

- no resources found on "/static" path within plugin classpath, or
- a user created component of NexusResourceBundle exists within plugin

The "default plugin" resource bundle component uses MimeUtil from core to select MIME types of re-

sources found within plugin, and will use same path to publish them (i.e. in plugin JAR "/static/image.png" will be published on "http://nexushost/nexus/static/image.png").

21.3.4 Plugin Templates

Interface: org.sonatype.nexus.templates.TemplateProvider

Template provider is a component providing repository templates to Nexus. Every plugin which provides a "new" repository type should add a TemplateProvider as it is the only way to instantiate a repository instance. The core of Nexus provides a "default" template provider with templates for all core repository types, and all custom repository plugins (P2, OBR) provide template providers for their types.

21.3.5 Event Inspectors

Interface: org.sonatype.nexus.proxy.events.EventInspector

Event inspectors are used to inspect events in Nexus. One example of where this extension point is used is the index generation. To generate a Nexus index, there is an event inspector which listens for RepositoryItemEvent subclasses and updates the index in response to repository activity.

21.3.6 Content Generators

Interface: org.sonatype.nexus.proxy.item.ContentGenerator

A content generator is a component that is able to generate content dynamically, on the fly, instead of just serving a static resource. The content generator is registered to respond to a path that corresponds to a file. When the resource is retrieved, Nexus discards the file content and uses the registered content generator to generate content. The Nexus Archetype plugin uses a content generator to generate the archetype-catalog.xml. Every time a client requests the archetype-catalog.xml, the archetype catalog is generated using information from the index.

21.3.7 Content Classes

Interface: org.sonatype.nexus.proxy.registry.ContentClass

Content class controls the compatibility between repository types. It defines the type of content that can be stored in a repository, and it also affects how repositories can be grouped into repository groups. Every plugin contributing a new repository type should provide an instance of this extension point. Nexus has a ContentClass implementation for every core supported repository type, and the P2 and OBR plugins define custom ContentClass implementations.

21.3.8 Storage Implementations

Interface: org.sonatype.nexus.proxy.storage.local.LocalRepositoryStorage

Interface: org.sonatype.nexus.proxy.storage.remote.RemoteRepositoryStorage

A plugin developer can override the default file-based local repository storage and the default remote HTTP repository storage interface. If your plugin needs to stores repository artifacts and information in something other than a filesystem, or if your remote repository isn't accessible via HTTP, your plugin would provide an implementation of one of these interfaces. Nexus provides one of the each: a file-system LocalRepositoryStorage and CommonsHttpClient 3.x based RemoteRepositoryStorage.

21.3.9 Repository Customization

Interface: org.sonatype.nexus.plugins.RepositoryCustomizer

This extension component will be invoked during configuration of every Repository instance, and may be used to add some "extra" configuration to repositories. The procurement plugin uses this mechanism to "inject" RequestProcessor that will evaluate rules before allowing execution of request.

21.3.10 Item and File Inspectors

Interface: org.sonatype.nexus.proxy.attributes.StorageItemInspector

Interface: org.sonatype.nexus.proxy.attributes.StorageFileItemInspector

Attribute storage ItemInspectors are able to "decorate" items in repositories with custom attributes. Every file stored/cached/uploaded in Nexus will be sent to these components for inspection and potentially decoration. The StorageItemInspector will get all item types for inspection (file, collections, links), while StorageFileItemInspector will only get file items. Currently only one ItemInspector is used in Nexus: the checksumming inspector, that decorates all file items in Nexus with SHA1 checksum and stores it into item attributes.

21.3.11 Nexus Feeds

Interface: org.sonatype.nexus.rest.feeds.sources.FeedSource

To add new RSS feeds, a plugin may provide implementation of this extension point. Nexus provides implementation for all the "core" RSS feeds.

21.3.12 Nexus Tasks and Task Configuration

Interface: org.sonatype.nexus.scheduling.NexusTask<T>

Interface: org.sonatype.nexus.tasks.descriptors.ScheduledTaskDescriptor

NexusTask is an extension point to implement new Nexus Scheduled Tasks.

If a contributed task needs UI, then the plugin which provides the NexusTask should provide a ScheduledTaskDescriptor which allows the UI customization for the task creation and management interface.

21.3.13 Application Customization

Interface: org.sonatype.nexus.rest.NexusApplicationCustomizer

This extension component is able to intercept URLs routed in the Nexus REST API layer.

21.3.14 Request Processing

Interface: org.sonatype.nexus.proxy.repository.RequestProcessor

This extension point can affect how a repository reacts to an item request.

21.4 Using the Nexus Plugin Archetype

To create a new Nexus Plugin, you can use the Nexus Plugin Archetype by with the Maven Archetype Plugins generate goal with the filter set to "nexus-plugin-archetype" as displayed in [Creating a Sample Nexus Plugin using the Archetype](#).

Enter the number of the nexus-plugin-archetype at the first prompt and select the version corresponding to the version of Nexus you are using. Supply the desired values for the groupId, artifactId, version, and package name for the generated plugin next and confirm your values provided.

Creating a Sample Nexus Plugin using the Archetype

```
$ mvn archetype:generate -Dfilter=nexus-plugin-archetype
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
...
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart
(org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
Choose archetype:
1: remote -> org.sonatype.nexus.archetypes:nexus-plugin-archetype (-)
Choose a number or apply filter (format: [groupId:]artifactId,
case sensitive contains): : 1
Choose org.sonatype.nexus.archetypes:nexus-plugin-archetype version:
1: 1.0
2: 1.1
3: 1.2
4: 2.1-RC1
5: 2.1
6: 2.1.1
7: 2.1.2
8: 2.2-SNAPSHOT
```

```
9: 2.2
10: 2.2-01
11: 2.2.1
Choose a number: 11: 11
Define value for property 'groupId': : org.sonatype.nexus.plugins
Define value for property 'artifactId': : sample-plugin
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': org.sonatype.nexus.plugins: :
[INFO] Using property: nexusVersion = 2.2.1
Confirm properties configuration:
groupId: org.sonatype.nexus.plugins
artifactId: sample-plugin
version: 1.0-SNAPSHOT
package: org.sonatype.nexus.plugins
nexusVersion: 2.2.1
Y: :
[INFO] -----
[INFO] Using following parameters for creating project from Archetype:
nexus-plugin-archetype:2.2.1
[INFO] -----
[INFO] Parameter: groupId, Value: org.sonatype.nexus.plugins
[INFO] Parameter: artifactId, Value: sample-plugin
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: org.sonatype.nexus.plugins
[INFO] Parameter: packageInPathFormat, Value: org/sonatype/nexus/plugins
[INFO] Parameter: package, Value: org.sonatype.nexus.plugins
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: nexusVersion, Value: 2.2.1
[INFO] Parameter: groupId, Value: org.sonatype.nexus.plugins
[INFO] Parameter: artifactId, Value: sample-plugin
[INFO] project created from Archetype in dir:
          /Users/manfred/Projects/sample-plugin
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 45.655s
[INFO] Finished at: Thu Nov 08 20:55:03 PST 2012
[INFO] Final Memory: 15M/153M
[INFO] -----
```

Once the Archetype plugin has created the project, you will have a project with the layout shown in [Layout of a Nexus Plugin Project](#) in a directory using the name of the artifactId value supplied.

Layout of a Nexus Plugin Project

```
pom.xml
src
```

```
main
  java
    org
      sonatype
        nexus
          plugins
            VirusScanner.java
            VirusScannerRepositoryCustomizer.java
            VirusScannerRequestProcessor.java
            XYVirusScanner.java
          events
            InfectedItemFoundEvent.java
        rest
          HelloWorldPlexusResource.java
```

The generated project contains the following classes:

VirusScanner

A VirusScanner interface described in Section [21.7](#).

VirusScannerRepositoryCustomizer

A class which customizes the repositories affected by the VirusScanner

VirusScannerRequestProcessor

A class which customizes the request handling process in Nexus

XYVirusScanner

An implementation of the VirusScanner Interface

InfectedItemFoundEvent

A simple event which is fired by the VirusScanner

HelloWorldPlexusResource

what is that.. TODO

21.5 Set the Target Nexus Version

When creating a new Nexus Plugin project with the Nexus Plugin archetype, a default target Nexus version will be set. To change the target Nexus version of the new Nexus Plugin project, you will need to edit the project's POM.

Open up the POM of your Nexus Plugin and find the section that defines properties. Here you can change the value of the nexus-version property to target a specific Nexus version.

```
<project>
  ...
  <properties>
    <nexus.version>2.2.1</nexus.version>
  </properties>
  ...
</project>
```

It is also possible to override the archetype's default value of this property during the creation of a new project. To do that, you would set the "nexusVersion" system property value on the command-line:

```
$ mvn archetype:generate \
-DarchetypeGroupId=org.sonatype.nexus.archetypes \
-DarchetypeArtifactId=nexus-plugin-archetype \
-DarchetypeVersion=2.2.1 \
-DnexusVersion=2.2
```

21.6 Building a Nexus Plugin Project

To build your Nexus plugin project, just run

```
mvn clean install
```

in the newly generated project directory. Once the build is completed, your plugin's JAR will be available in the project's target/ folder.



Warning

Nexus plugin development requires you to use Apache Maven 3.

The Nexus Plugin project, as created by the Nexus Plugin archetype, depends on a number of artifacts which may not be available from the Maven Central repository. If you experience missing artifacts during your Nexus plugin project build, you should make sure that the Sonatype Public Grid Repository is available as release repository.

```
http://repository.sonatype.org/content/groups/sonatype-public-grid/
```

If you are using Nexus, and you have configured your build to work against a public group, you will want to make sure that you have added the repository to your public group.

21.7 Creating a Complex Plugin

In this section, we will step through the skeletal, sample project that implements a virus scanner plugin for Nexus as created with the archetype. This plugin will consist of:

- A managed "virus scanner" component
- A RequestProcessor that sends all "incoming" artifacts for scanning
- A repository customizer to inject a RequestProcessor to all proxy repositories

We start with creating a @Managed component contract for the VirusScanner. While this class could just as easily be a non-managed component, this example uses the @Managed and @Singleton annotations to demonstrate dependency injection.

VirusScanner Interface

```
package org.sonatype.book.nexus;

import javax.inject.Singleton;

import org.sonatype.nexus.proxy.item.StorageFileItem;
import org.sonatype.plugin.Managed;

@Managed
@Singleton
public interface VirusScanner
{
    boolean hasVirus( StorageFileItem file );
}
```

Once we have the interface for VirusScanner, we need to define a named instance XYVirusScanner which implements the interface. The following example shows how the @Named annotation is used to assign a name of "XY" to this implementation.

XYVirusScanner Implementation

```
package org.sonatype.book.nexus;

import javax.inject.Named;

import org.sonatype.nexus.proxy.item.StorageFileItem;

@Named( "XY" )
public class XYVirusScanner implements VirusScanner {

    public boolean hasVirus( StorageFileItem file ) {

        // DO THE JOB HERE
        System.out.println( "Kung Fu VirusScanner --- " +
            "scanning for viruses on item: " + file.getPath() );

        // simulating virus hit by having the filename
        // contain the "infected" string
        return file.getName().contains( "infected" );
    }

}
```

The next class is a request processor which scans an artifact for viruses before it is cached locally. If a virus is found in an artifact, this plugin will refuse to cache the artifact and trigger an event which will signal that a virus was found in a file item. Note the use of @Named which assigns the name "virusScanner" to this component. Also note the two uses of @Inject. The first use of @Inject will fetch the default implementation of ApplicationEventMulticaster, and the second use of @Inject will fetch the "XY" virus scanner.

Virus Scanning Request Processor

```
package org.sonatype.book.nexus;

import javax.inject.Inject;
import javax.inject.Named;

import org.sonatype.book.nexus.events.InfectedItemFoundEvent;
import org.sonatype.nexus.proxy.ResourceStoreRequest;
import org.sonatype.nexus.proxy.access.Action;
import org.sonatype.nexus.proxy.item.AbstractStorageItem;
import org.sonatype.nexus.proxy.item.StorageFileItem;
import org.sonatype.nexus.proxy.repository.ProxyRepository;
import org.sonatype.nexus.proxy.repository.Repository;
import org.sonatype.nexus.proxy.repository.RequestProcessor;
import org.sonatype.plexus.appevents.ApplicationEventMulticaster;
```

```
@Named( "virusScanner" )
public class VirusScannerRequestProcessor
implements RequestProcessor {

    @Inject
    private ApplicationEventMulticaster applicationEventMulticaster;

    @Inject
    private @Named( "XY" )
    VirusScanner virusScanner;

    // @Inject
    // private @Named("A") CommonDependency commonDependency;

    public boolean process( Repository repository,
                           ResourceStoreRequest request, Action action )
    {
        // Check dependency
        // System.out.println( "VirusScannerRequestProcessor " +
        //--- CommonDependency data: " +
        commonDependency.getData()
    }

    // don't decide until have content
    return true;
}

public boolean shouldProxy( ProxyRepository repository,
                           ResourceStoreRequest request )
{
    // don't decide until have content
    return true;
}

public boolean shouldCache( ProxyRepository repository,
                           AbstractStorageItem item )
{
    if ( item instanceof StorageFileItem ) {
        StorageFileItem file = (StorageFileItem) item;

        // do a virus scan
        boolean hasVirus = virusScanner.hasVirus( file );

        if ( hasVirus ) {
            applicationEventMulticaster
                .notifyEventListeners(
                    new InfectedItemFoundEvent( item
                        .getRepositoryItemUid().getRepository(),

```

```
        file ) );
    }

    return !hasVirus;
} else {
    return true;
}
}
```

The last component is the RepositoryCustomizer. It simply injects our virus scanner RequestProcessor into proxy repositories only. For his example local uploads are considered safe, so the only way to get an artifact into Nexus from the uncontrolled Internet is a proxy repository. Note how the request processor is injected into this repository customizer with @Inject and @Named.

The Virus Scanner Repository Customizer

```
package org.sonatype.book.nexus;

import javax.inject.Inject;
import javax.inject.Named;

import org.sonatype.configuration.ConfigurationException;
import org.sonatype.nexus.plugins.RepositoryCustomizer;
import org.sonatype.nexus.proxy.repository.ProxyRepository;
import org.sonatype.nexus.proxy.repository.Repository;
import org.sonatype.nexus.proxy.request.RequestProcessor;

public class VirusScannerRepositoryCustomizer
    implements RepositoryCustomizer {

    @Inject
    private @Named( "virusScanner" )
    RequestProcessor virusScannerRequestProcessor;

    public boolean isHandledRepository( Repository repository ) {
        // handle proxy repos only
        return repository.getRepositoryKind()
            .isFacetAvailable( ProxyRepository.class );
    }

    public void configureRepository( Repository repository )
        throws ConfigurationException {
        repository.getRequestProcessors()
            .put( "virusScanner",
                  virusScannerRequestProcessor );
    }
}
```

```
}
```

21.8 Nexus Plugin Descriptor Maven Plugin

Nexus plugins have a custom packaging "nexus-plugin" which is introduced by the app-lifecycle-maven-plugin. A "nexus-plugin" packaged plugin:

- is a plain JAR
- has a META-INF/nexus/plugin.xml embedded Nexus Plugin Metadata embedded
- has static resources embedded into the plugin JAR

The plugin introduces a new project path (i.e. src/main/static-resources). Static resources such as JavaScript files, images, and CSS should be located in this folder and will be packaged appropriately.

21.9 The Nexus Plugin Descriptor

Every Nexus plugin has a plugin descriptor which is generated during the build process for a plugin. This plugin descriptor is packaged with the plugin JAR and can be found in \$basedir/target/classes/META-INF/nexus/plugin.xml

A Nexus Plugin Descriptor

```
<plugin>
  <modelVersion>1.0.0</modelVersion>
  <groupId>org.sonatype.sample</groupId>
  <artifactId>sample-plugin</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>Nexus Plugin Archetype</name>
  <applicationId>nexus</applicationId>
  <applicationEdition>OSS</applicationEdition>
  <applicationMinVersion>1.4.0</applicationMinVersion>
</plugin>
```

If your Nexus plugin has any dependencies, they will be included in this plugin descriptor automatically. For example, if the Nexus plugin you were developing had a dependency on commons-beanutils version 1.8.2, your plugin descriptor will include the following classpathDependency

```
<plugin>
  <modelVersion>1.0.0</modelVersion>
  <groupId>org.sonatype.book.nexus</groupId>
  <artifactId>sample-plugin</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>Nexus Plugin Archetype</name>
  <applicationId>nexus</applicationId>
  <applicationEdition>OSS</applicationEdition>
  <applicationMinVersion>1.4.0</applicationMinVersion>
  <classpathDependencies>
    <classpathDependency>
      <groupId>commons-beanutils</groupId>
      <artifactId>commons-beanutils</artifactId>
      <version>1.8.2</version>
      <type>jar</type>
    </classpathDependency>
  </classpathDependencies>
</plugin>
```

21.10 Defining Custom Repository Types

When you need to introduce a custom repository type, you should implement the Repository interface. The following example extends the HostedRepository class and adds a repository type with the path prefix "sample".

Creating a Custom Repository Type Interface

```
package org.sample.plugin;

import org.sonatype.nexus.plugins.RepositoryType;
import org.sonatype.nexus.proxy.repository.HostedRepository;

@RepositoryType( pathPrefix="sample" )
public interface SampleRepository extends HostedRepository {
  String boo();
}
```

If you want to implement a custom repository type, you should reference the nexus-proxy module as dependency which contains the AbstractRepository class which is a useful super-class for repository implementations. To implement the SampleRepository interface, you can then extend the AbstractRepository as shown in the following example.

Creating a Custom Repository Type Implementation

```
package org.sample.plugin;

public class DefaultSampleRepository extends AbstractRepository
    implements SampleRepository {
    .... implement it
}
```

Your newly introduced repository type will appear under <http://localhost:8081/nexus/content/sample/>.

Chapter 22

Migrating to Nexus

If you have been running another repository manager, such as Artifactory, Archiva, or Proximity, and you want to migrate this repository to Nexus, you can do so by copying the files from a standard Maven 2 repository file layout to Nexus.

Depending on your repository managers you will have to use different approaches to get access to a repository in Maven 2 format on disk.

Nexus stores it's artifacts in standard maven 2 layout, and they are served directly from disk, and can therefore be easily integrated into an existing Nexus instance as a new hosted repository.

22.1 Migrating from Archiva

22.1.1 Introduction

This appendix walks you through the process of migrating an existing [Archiva](#) installation to a new Nexus installation.

22.1.2 Migrating Archiva Repositories

Archiva uses the filesystem to store hosted repositories and proxied repositories, because of this migrating from Archiva to Nexus couldn't be simpler. The following sections outline the process for migrating existing Archiva repositories to a new Nexus instance.

22.1.3 Migrating an Archiva Managed Repository

Archiva Managed Repositories are the equivalent of Nexus Hosted repositories. To migrate a Managed Repository from Archiva to Nexus, all you need to do is:

- Create a New Hosted Repository in Nexus
- Copy the Contents of the Archiva Managed Repository to the Storage Directory of the newly-created Nexus Hosted Repository
- Rebuild the Index for the New Nexus Hosted Repository

The following example will walk through the process of migrating the Archiva repository named "internal" to a new Nexus Hosted repository named "internal". To view your managed repositories in Archiva, login to Archiva as an administrative user and click on the "Repositories" link in the left-hand navigation menu. Clicking on "Repositories" will list all of your Archiva Managed repositories as shown in Figure 22.1.

The screenshot shows the 'Managed Repositories' interface for Archiva. At the top right are 'Add', 'Edit', 'Delete', and 'RSS' buttons. The repository details are listed below:

Archiva Managed Internal Repository	
Identifier	internal
Name	Archiva Managed Internal Repository
Directory	./data/repositories/internal
WebDAV URL	http://localhost:8080/archiva/repository/internal/
Type	Maven 2.x Repository
Releases Included	
Snapshots Included	
Scanned	
Scanning Cron	0 0 * * *
Actions	Scan Repository Now
Stats	No Statistics Available.
POM Snippet	Show POM Snippet

Figure 22.1: Archiva Managed Repositories

To migrate this Managed repository to a Nexus Hosted repository, you will need to find the directory in which Archiva stores all of the repository artifacts. To do this, click on the Edit link listed next to the name of the repository you want to migrate as shown in Figure 22.1. Click on Edit should load the form shown in Figure 22.2.

Admin: Edit Managed Repository

ID: internal

Name*: Archiva Managed Internal Repository

Directory*: ./data/repositories/internal

Index Directory: []

Type: Maven 2.x Repository

Cron*: 0 0 * * * ?

Repository Purge By Days Older Than: 30

Repository Purge By Retention Count: 2

Releases Included

Snapshots Included

Scannable

Delete Released Snapshots

Update Repository

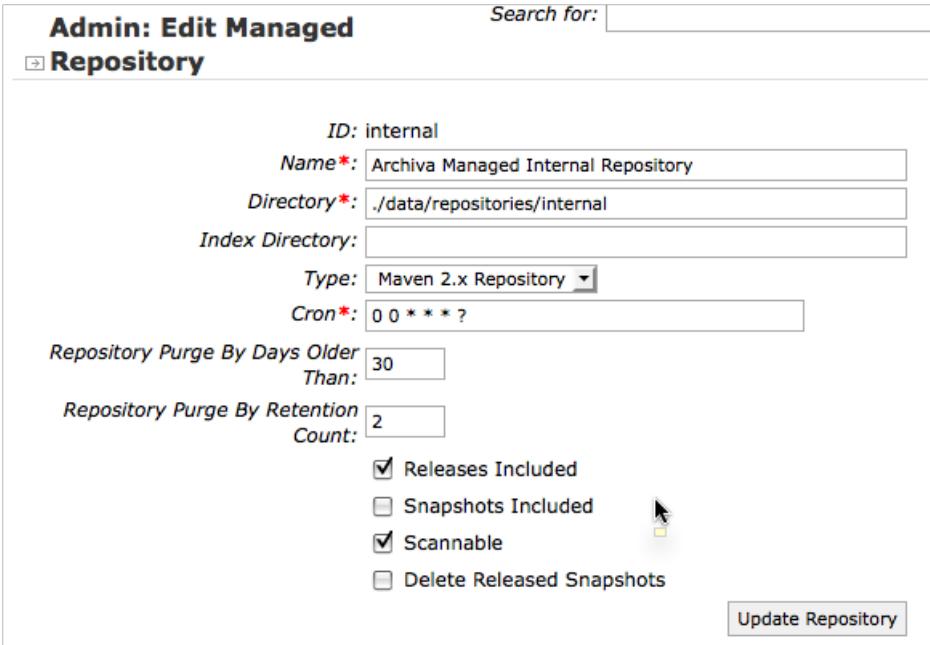


Figure 22.2: Editing an Archiva Managed Repository

Take note of the file path for Directory. The file path shown in Figure 22.2 is ./data/repositories/internal. If Archiva is installed in /usr/local/archiva-1.2.1, this would correspond to the directory /usr/local/archiva-1.2.1/data/repositories/internal. You will use this path later in this section to copy the contents of your old Archiva Managed Repository to your new Nexus Hosted Repository.

Next, create a new Nexus repository with the same identifier and Name as the old Archiva Managed Repository. To do this, log into Nexus as an administrative user, click on Repositories in the left-hand Nexus navigation menu, and then click on the Add drop-down as shown in Figure 22.3. Select "Hosted Repository" and then fill out the Repository ID and Repository Name to match the name of the old Archiva repository. If you are migrating a Snapshot repository, select a Repository Policy of Snapshot, and if you are migrating a Release repository select a Snapshot Policy of Release.

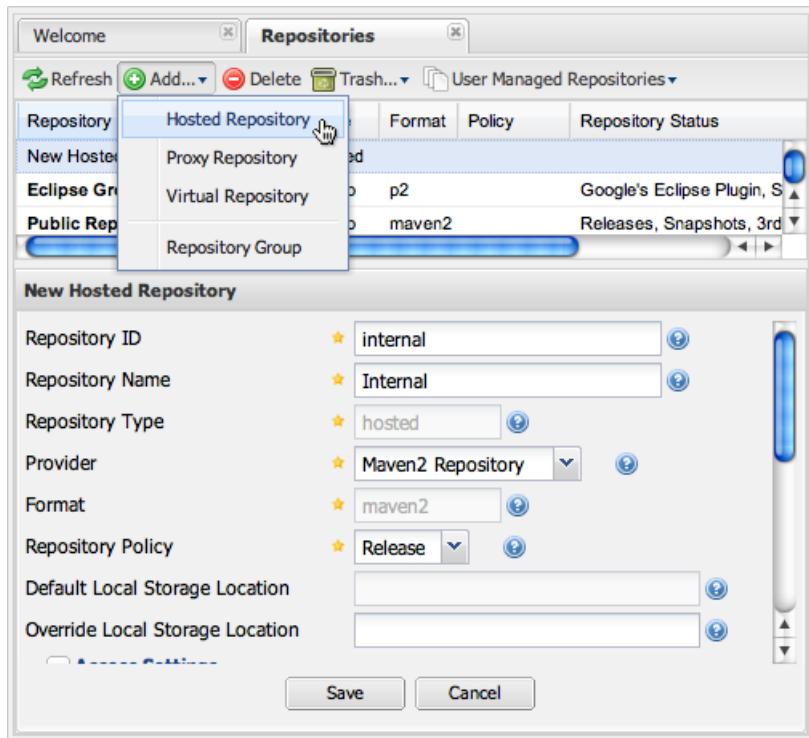


Figure 22.3: Creating a Nexus Hosted Repository

Now, you'll need to copy the Archiva repository to the Nexus repository. You can do this, by copying the contents of the Archiva repository directory to the Nexus repository storage directory. If we assume that Archiva is install in /usr/local/archiva-1.2.1, Nexus is install in /usr/local/nexus-1.3.6, and the Sonatype Work directory is /usr/local/sonatype-work. You can copy the contents of the Archiva managed repository to the new Nexus hosted repository by executing the following command:

```
$ cp -r /usr/local/archiva-1.2.1/data/repositories/internal/* \
/usr/local/sonatype-work/nexus/storage/internal/
```

If you are migrating to a Nexus instance which is on a different server, you can simply create an archive of the /usr/local/archiva-1.2.1/data/repositories/internal directory, copy it to the new server, and then decompress your repository archive in the appropriate directory.

Warning

 Archiva stores artifacts from proxied remote repositories in the same directory as artifacts in a managed repository. If you have been proxying a remote repository, you might want to remove artifacts that have been proxied from a remote repository. For example, if your organization uses a groupId of org.company for internal project, you can make sure to only copy the artifacts under the corresponding org/company/

Once the contents of the repository have been copied to the Nexus Hosted repository, you must rebuild the repository index as shown in Figure 22.4. Right-clicking on the repository in the list of Nexus repositories will display the context menu shown in the following figure.

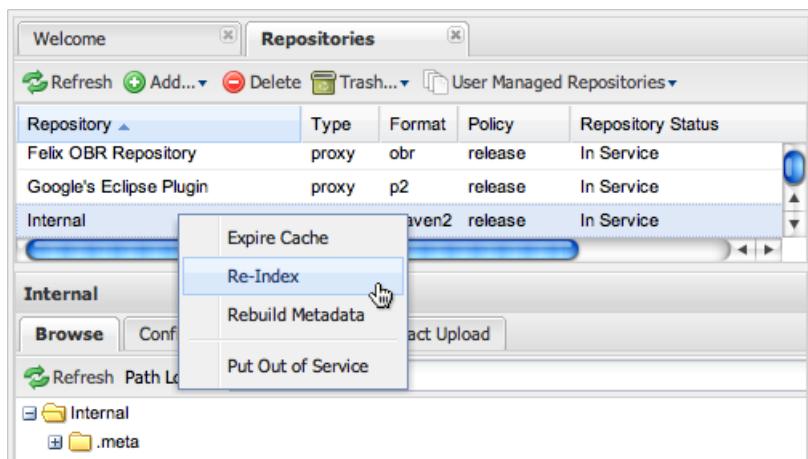


Figure 22.4: Rebuilding the Index of a Nexus Hosted Repository

Once the migration is complete, you will be able to search and browse the contents of your newly migrated Nexus Hosted repository.

22.1.4 Migrating an Archiva Proxy Connector

Archiva allows you to define remote repositories and repository connectors to proxy remote repositories and cache remote artifacts from remote repositories in Archiva Managed Repositories. While Nexus also provides Proxy repositories, there is one major difference between Nexus and Archiva. Where Nexus maintains a separate local storage directory for each proxy repository, Archiva combines cached remote

artifacts into a single filesystem with the contents of a managed repository. In other words, there is no good way to transfer an existing local cache of artifacts between Archiva and Nexus without manually manipulating the contents of Archiva's Managed Repository directory.

To recreate an Archiva repository connector in Nexus as a Proxy repository and to preserve the local cache of artifacts from this repository. You'll need to create a Proxy repository in Nexus, copy the contents of the existing proxy repository to the Nexus storage location for your new Proxy repository, and then rebuild the metadata of your new Nexus Proxy repository.

First step is to take a look at the Remote Repositories in your Archiva installation: log in as an administrative user and then click on "Repositories" under the Administration menu in the left-hand Archiva navigation menu. Once you've clicked this link and loaded the list of repositories, scroll to the bottom of the page to see the list of remote repositories as shown in Figure 22.5.

The screenshot shows the 'Remote Repositories' section of the Archiva administration interface. It lists two remote repositories:

Identifier	Name	URL	Type
central	Central Repository	http://repo1.maven.org/maven2	Maven 2.x Repository
maven2-repository.dev.java.net	Java.net Repository for Maven 2	http://download.java.net/maven/2/	Maven 2.x Repository

Figure 22.5: Browsing Archiva Remote Repositories

Defining a proxy repository in Archiva involves associating one of the remote repositories defined in Figure 22.5 with one of the Managed Repositories defined in Figure 22.1. Once you do this, requests for artifacts from the managed repository will also query the remote repository. If an artifact is found in the remote repository, it will be retrieved and stored in the managed repository's storage directory. To see a list of proxy connectors and the managed repositories they are associated with, click on "Proxy Connectors" in the left-hand Archiva menu, you will see a list similar to that shown in Figure 22.6.

The screenshot shows the 'Repository Proxy Connectors' section of the Archiva interface. At the top right is a green '+' icon labeled 'Add'. Below it, there's a header for 'internal' repositories: 'Archiva Managed Internal Repository' with a logo. Two proxy connectors are listed:

- Proxy Connector**: 'central' (id: 1). Description: 'Central Repository' with URL '<http://repo1.maven.org/maven2>'. It includes a 'Settings' link and edit/pencil icons.
- Proxy Connector**: 'maven2-repository.dev.java.net' (id: 2). Description: 'Java.net Repository for Maven 2' with URL '<http://download.java.net/maven/2/>'. It includes a 'Settings' link and edit/pencil icons.

Figure 22.6: Archiva Proxy Connectors

Click on the Edit Icon (or Pencil) next to second Proxy Connector listed in Figure 22.6, this will then load the settings form for this proxy connector shown in Figure 22.7. You should use the settings for this proxy connect to configure your new Nexus Proxy repository.

The screenshot shows the configuration interface for a Proxy Connector in Archiva. At the top, there are three dropdown menus: 'Network Proxy*' set to '(direct connection)', 'Managed Repository*' set to 'internal', and 'Remote Repository*' set to 'maven2-repository.dev.java.net'. Below these are several policy settings:

Policies:	Return error when:	always
	On remote error:	stop
	Releases:	once
	Snapshots:	never
	Checksum:	fix
	Cache failures:	yes

Under 'Properties', there is a text input field followed by an 'Add Property' button. A note below says 'No properties have been set.' In the 'Black List' section, there is a text input field followed by an 'Add Pattern' button. A note below says 'No black list patterns have been set.' In the 'White List' section, there is a text input field followed by an 'Add Pattern' button. Three entries are listed: "'javaxax/**'" with a red X, "'org/jvnet/**'" with a red X, and "'com/sun/**'" with a red X. At the bottom right is a 'Save Proxy Connector' button.

Figure 22.7: Archiva Proxy Connector Settings

To create a Proxy repository that will correspond to the Proxy Connector in Archiva, log into Nexus as an administrative user, and click on Repositories in the left-hand Nexus menu. Once you can see a list of Nexus repositories, click on Add... and select Proxy Repository from the drop-down of repository types. In the New Proxy Repository form (shown in Figure 22.8) populate the repository ID, repository Name, and use the remote URL that was displayed in Figure 22.5. You will need to create a remote repository for every proxy connector that was defined in Archiva.

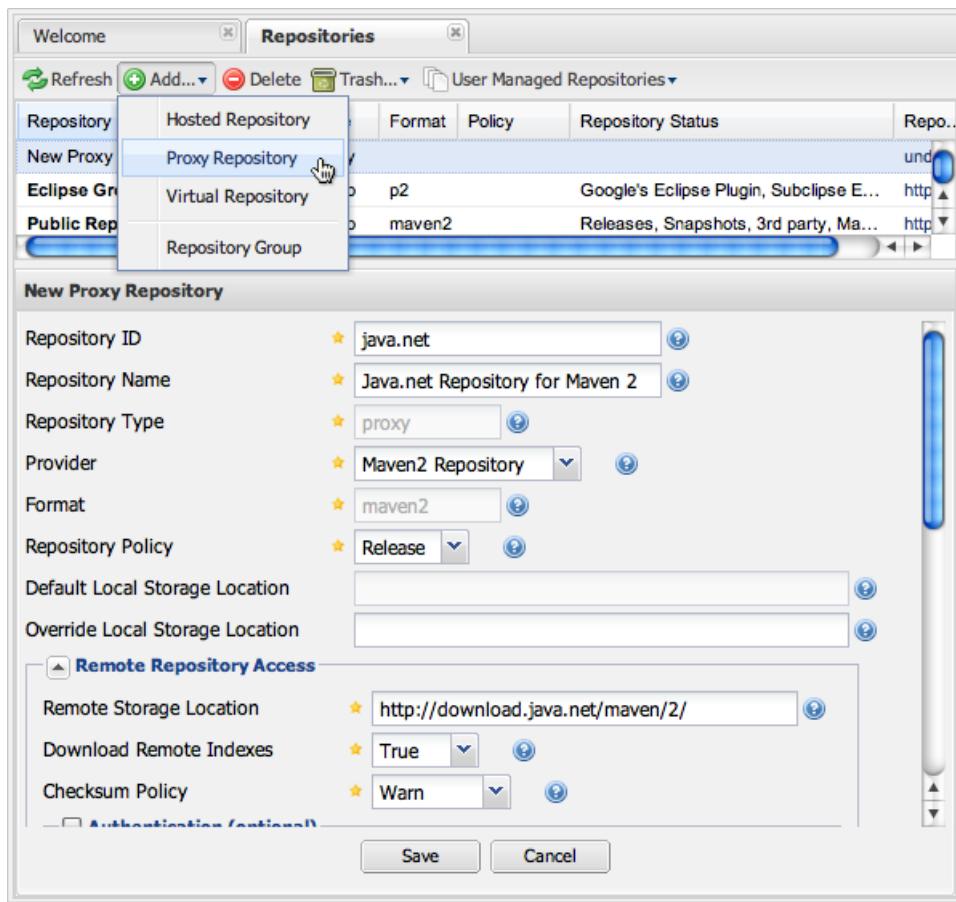


Figure 22.8: Creating a Nexus Proxy Repository

To expose this new Proxy repository in a Repository Group, create a new Nexus Repository group or select an existing group by clicking on Repositories in the left-hand Nexus menu. Click on a repository group and then select the Configuration tab to display the form shown in Figure 22.9. In the Configuration tab you will see a list of Order Group Repositories and Available Repositories. Click and drag your new Nexus Proxy repository to the list of Ordered Group Repositories, and click Save.

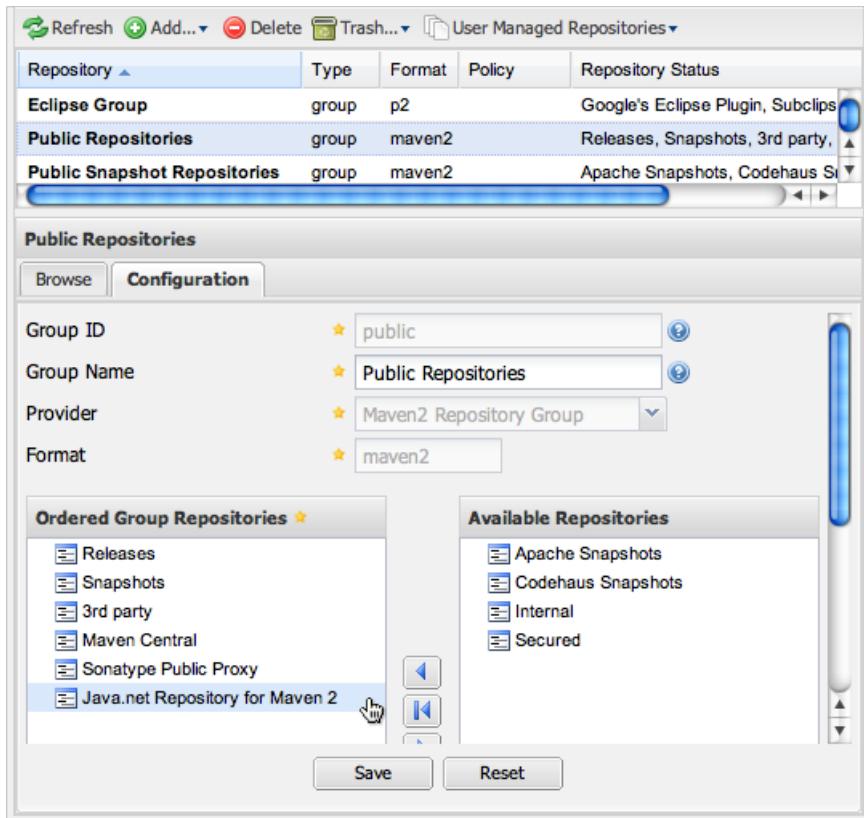


Figure 22.9: Adding a Proxy Repository to a Repository Group

Next, you will need to define repository groups that will tell Nexus to only locate certain artifacts in the newly created proxy repository. In , Archiva defined three patterns that were used to filter artifacts available from the proxy connector. These three patterns were "javax/", "com/sun/", and "org/jvnet/**". To recreate this behaviour in Nexus, define three Routes which will be applied to the group you configured in Figure 22.9. To create a route, log in as an administrative user, and click on Routes under the Administration menu in the left-hand Nexus menu. Click on Add.. and add three inclusive routes that will apply to the repository group you configured in Figure 22.9.

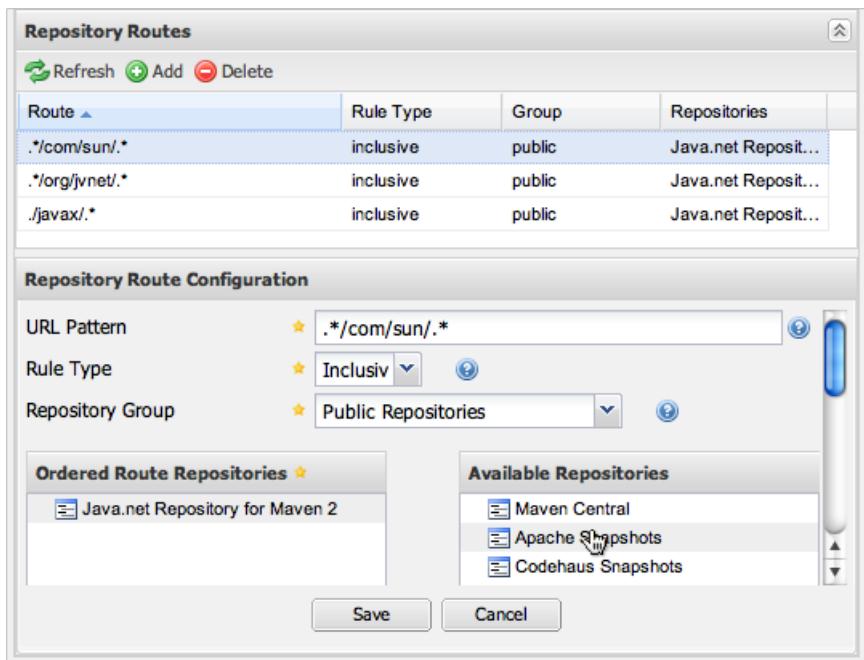


Figure 22.10: Defining Nexus Repository Groups

22.2 Migrating from Artifactory

This appendix provides a guideline for migrating a Maven repository from Artifactory to Nexus.

Typically migrating from Artifactory revolves around migrating hosted repositories only, since any proxy repositories configured in Artifactory can just be set up with the same configuration in Nexus and all data will be retrieved from the upstream repositories again.

Hosted repositories on the other hand have to be migrated. The best practice for migration is to use the import/export feature of Artifactory and migrate one hosted repository after another. Please consult the Artifactory documentation for step by step instructions on how to export a repository.

After the export you have to create a hosted repository in Nexus e.g. with the name "old-releases" as documented in Section 4.4. This will create a folder in sonatype-work/nexus/storage/old-releases.

Now you are ready to take the exported repository and copy it into the newly created storage folder.

Going back to the Nexus user interface, navigate to the repository administration and select the Browse Storage panel. Right-click on the root folder of the repository and select Rebuild Metadata first and as a second step select Update Index. Once these tasks are completed, the migrated repository is ready to be used.

After these task are completed you will probably want to add the migrated repository to the Public Repositories group or any other group in which you want the migrated repository content to be available.

If you want to ensure that the repository does not get any further content added you can set the Deployment Policy to Read Only in the Access Settings of the repository Configuration panel.

Chapter 23

Configuring Nexus for SSL

23.1 Introduction

Using Secure Socket Layer SSL to secure protocols like http, ldap and smtp is a critical step of securing your Nexus setup. Since Nexus is serving content as well as connecting to external sources there are two aspects of SSL configuration related to Nexus:

- Configuring SSL certificate usage when connecting to external systems including
 - Proxying a remote repository available via https
 - Connecting to a SSL secured SMTP server
 - Connecting to an LDAP server via ldaps
- Exposing the Nexus user interface and content via https

Securing all connections to external systems with SSL as well as exposing Nexus via SSL are both recommended best practices for any deployment.

Especially when you set up a repository manager for a team of developers spread out over a variety of locations both internal and external to a corporate network, you will likely want to secure your repository using SSL.

23.2 SSL Client Certificates

23.2.1 SSL Certificate Management

Nexus allows you to manage all SSL certificates directly in the user interface. The administration interface for SSL certificates as visible in Figure 23.1 and can be accessed by selecting *SSL Certificates* in the left hand *Administration* menu. The list of certificates displayed shows the certificate for the SSL secured Central Repository preconfigured in Nexus Professional and a self signed certificate registered in Nexus.

Note

The SSL Certificate Management is a Nexus Professional feature.

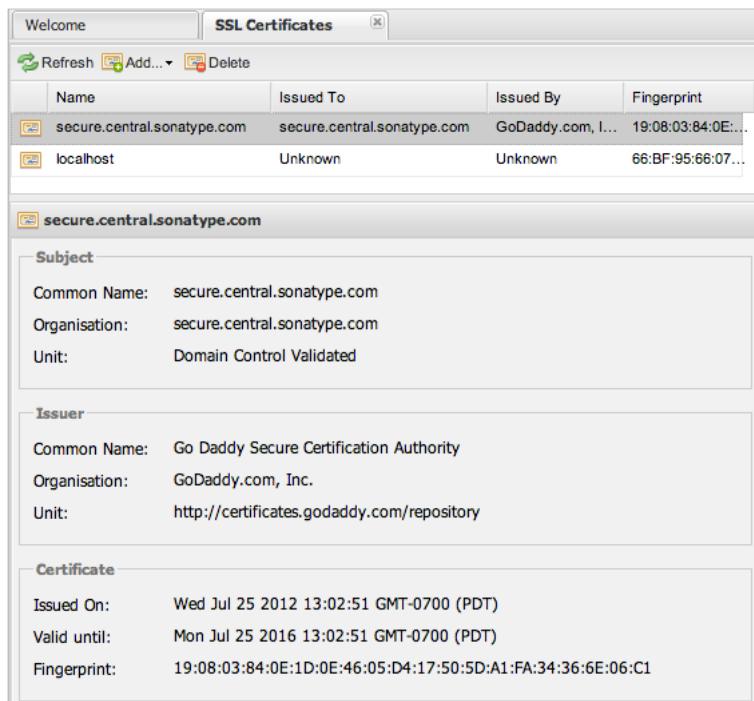


Figure 23.1: SSL Certificates Administration

The actual list of SSL certificates can be reloaded by clicking the *Refresh* button above the list. In addition certificates can be added and deleted with the *Add* and *Delete* buttons.

Pressing the add button provides a choice to load a certificate from a server with the *Load from server* option or to insert a certificate in PEM format with the *Paste PEM*.

The dialog to load a certificate from a server allows you to provide a hostname, a hostname:port string or a full URL. When providing a hostname a connection via http:// using the default SSL port 443 will be attempted. Using a full URL on the other hands gives the most control.

As an example you could retrieve the certificate for the secured Central Repository using the url <https://secure.central.sonatype.org/nexus/service/local/repositories/central/ssl/certs/smtp.gmail.com>. Besides retrieving certificates for servers running https you can retrieve and therefore register the certificate for email and directory servers. An LDAP directory server certificate can be loaded with a URL using the ldaps protocol and the desired hostname and port similar to *ldaps://localhost:10636*. A SMTP server can be queried with a similar pattern using *smtps://localhost:465*. After successful retrieval the details of the certificate as displayed in a dialog. Figure 23.2 shows the result from querying a certificate from *smtps://smtp.gmail.com:465*. Pressing the *Add Certificate* button will save the certificate within Nexus and allow you to connect to the associated services.

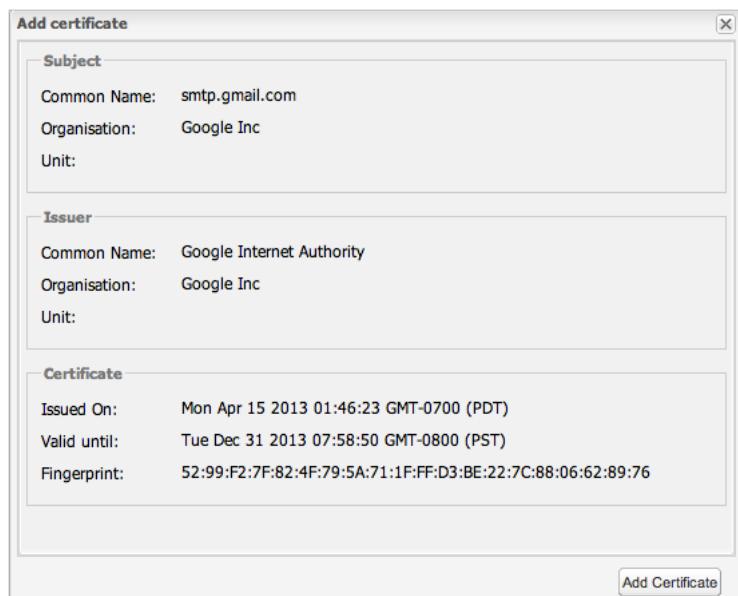


Figure 23.2: Certificate Details Displayed After Successful Retrieval

The dialog displays details about the certificate owner in the *Subject* section, the certificate issuer in the *Issuer* section and the certificate itself in the *Certificate* section. The same data is displayed below the list of certificates when you select a specific certificate in the list.

The alternate method of registering a certificate with Nexus uses the PEM format of the [X.509 certificate](#) as used by SSL. An example of inserting such a certificate in the dialog is shown in Figure 23.3.

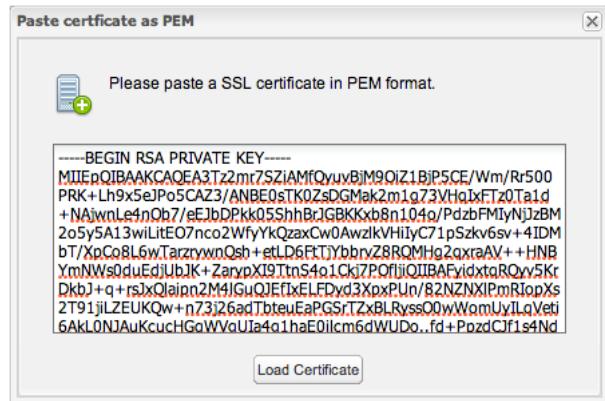


Figure 23.3: Providing a Certificate in PEM Format

Once a certificate for an LDAP server or SMTP server has been registered in Nexus, you can configure connections to these servers in the LDAP and Server/SMTP Settings administration user interfaces.

23.2.2 Proxying SSL Secured Repositories

When setting up a proxy repository with a remote storage location secured with https the repository administration will display an *SSL* configuration tab under the list of repositories, if the proxy repository is selected. For a repository using a self signed certificate the repository status will initially be set to be in service, but the remote will be automatically blocked and set to be unavailable, since the certificate of the remote server is not trusted. Remote repositories that use a CA signed certificate will be automatically trusted.

The *SSL* tab displays as visible in Figure 23.4 the details of the certificate and allows you to add the certificate to the truststore or to remove it from it with the button on the top right hand corner named *Add to trust store* and *Remove from trust store* respectively.

In addition the checkbox on the top left corner allows you to store the certificate in the Nexus internal SSL trust store. Otherwise the certificate is installed into the trust store of the JVM running Nexus. Using the Nexus internal trust store will work fine even when migrating Nexus from one machine to another or when switching the Java runtime and JVM between restarts for example during upgrades, and is therefore recommended. At runtime the JVM and Nexus trust stores are merged and both used so you can use a combination, if your organization e.g. maintains a default trust store for all JVM installations.

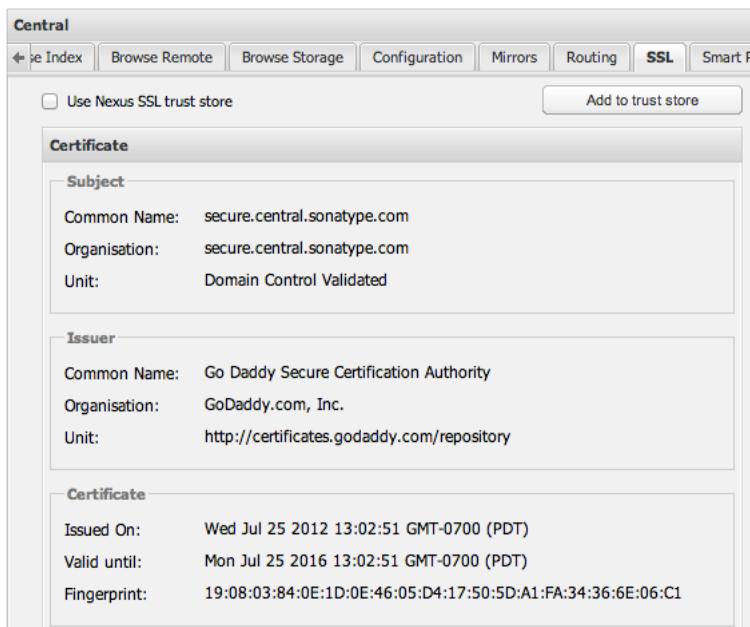


Figure 23.4: SSL Tab for a Proxy Repository with Remote Server Using HTTPS

When removing a certificate from the trust store, a Nexus restart is required.

23.2.3 Manually Configuring Trust Stores

The Nexus user interface should be sufficient to work with the trust stores and certificates. However in older versions of Nexus as well as some use cases you need to manually configure the trust store.

Sonatype provides an import-ssl tool, which can be downloaded from: <http://central.sonatype.com/help-import-ssl.jar>. It allows you to import a client certificate in two steps:

- importing the server's SSL chain and
- importing the client SSL key/certificate pair.

The Java Virtual Machine running Nexus uses the [Java Secure Socket Extension \(JSSE\)](#) to enable secure Internet communication. It uses two certificate stores - the truststore and the keystore.

A truststore contains certificates from servers run by other parties that you expect to communicate with, or from Certificate Authorities that you trust to identify other parties. This truststore ships with a number of CA's out of the box - trusted root certificates.

A keystore contains private keys, and the certificates with their corresponding public keys.

Typically they are stored in separate files stored in the default location of

Some notes about the location of the key-store and default key-store passwords:

- If you are using the default JSSE key-store locations on either a Linux or OS X platform, you must run the commands below as the root user. You can do this either by changing to the root user (`su -`), or by using the `sudo` command: `sudo [command]`.
- The default password used by Java for the built-in key-stores is *changeit*. If your key-store uses a different password, you'll need to specify that password as the last parameter on the command lines above.
- If you want to specify your own key-store/truststore file, provide that in place of `<keystore_dir>` in the examples below.
- If you're using a password other than *changeit* for your keystore, you should supply it immediately following the keystore path in the commands below.
- If you specify a keystore location that doesn't exist, the `import-ssl` utility will create it on-demand.

Before you begin the process of importing a Server SSL Chain and a client certificate you will need three things:

- Network access to the SSL server you are connecting to,
 - An SSL client certificate,
 - and a certificate password.
-

For server certificates you should either import directly into import into that.

**Warning**

If you replace the existing truststore rather than adding to it or if you override the truststore location, you will lose all of the trusted CA root certificates of the JRE and no SSL sites will be accessible.

23.2.3.1 Import the Server SSL Chain

The first command imports the entire self-signed SSL certificate chain for central.sonatype.com into your JSSE keystore:

```
$ java -jar import-ssl.jar server central.sonatype.com <keystore>
```

You would substitute the server name used in the previous listing with the server name you are attempting to connect to. This particular command will connect to <https://central.sonatype.com>, retrieve, and import the server's SSL certificate chain.

23.2.3.2 Import the Client SSL Key/Certificate Pair

The second command imports your client-side SSL certificate into the JSSE keystore, so Nexus can send it along to the server for authentication:

```
$ java -jar import-ssl.jar client <your-certificate.p12> \  
<your-certificate-password> keystore
```

When the client command completes, you should see a line containing the keystore path, like the one that follows. This path is important; you will use it in your Nexus configuration below, so make a note of it!

```
...  
Writing keystore: /System/Library/Frameworks/JavaVM.framework/\  
Versions/1.6.0/Home/lib/security/jsssecacerts
```

If you want to make a new keystore to import your keys into, you will have to use the keytool that ships with your Java installation to create an empty keystore:

```
keytool -genkey -alias foo -keystore keystore  
keytool -delete -alias foo -keystore keystore
```

Tip

Make sure to use the keytool commands for your Java version used to run Nexus. The documentation for keytool is available online for [Java 6](#) as well as [Java 7](#).

23.2.3.3 Configuring Nexus Start-up

Once both sets of SSL certificates are imported to your keystore and/or truststore, you can modify the *wrapper.conf* file located in *\$NEXUS_HOME/bin/jsw/conf/* to inject the JSSE system properties necessary to use these certificates, as seen below adapting the iterator number (10, 11..) to start at the last used value, which depends on the rest of your configuration.

```
warpper.java.additional.10=-Djavax.net.ssl.keyStore=<keystore>  
warpper.java.additional.11=-Djavax.net.ssl.keyStorePassword=< ↵  
    keystore_password>  
warpper.java.additional.12=-Djavax.net.ssl.trustStore=<truststore>  
warpper.java.additional.13=-Djavax.net.ssl.trustStorePassword=< ↵  
    truststore_password>
```

Once you have configured the Nexus start-up option shown above, restart Nexus and attempt to proxy a remote repository which requires an SSL client certificate. Nexus will use the keystore location and keystore password to configure the SSL interaction to accept the server's SSL certificate and send the appropriate client SSL certificate using the manual configuration you have complete with the import-ssl tool.

23.3 Configuring Nexus to Serve SSL

Providing access to the Nexus user interface and content via https only is a recommended best practice for any deployment.

The recommended approach to implementation is to proxy Nexus behind a server that is configured to serve content via SSL and leave Nexus configured for http. The advantage of this approach is that Nexus

can easily be upgraded and there is no need to work with the JVM truststore. In addition you can use the expertise of your system administrators and the preferred server for achieving the proxying, which in most cases will already be in place for other systems.

Common choices are servers like Apache httpd, nginx, Eclipse Jetty or even dedicated hardware appliances. All of them can easily be configured to serve SSL content and there is a large amount of reference material available for configuring these servers to serve secure content. For example Apache httpd would be configured to use mod_ssl.

Alternatively the Jetty instance that is part of the default Nexus install can be configured to serve SSL content directly, and if you would like to avoid the extra work of putting a web server like Apache httpd in front of Nexus, this section shows you how to do that.

Tip

Keep in mind that you will have to redo some of these configurations each time you upgrade Nexus, since they are modifications to the embedded Jetty instance located in `$NEXUS_HOME`.

To configure Nexus to serve SSL directly to clients, you'll need to perform the following steps.

Note

All examples given here can be found in the Nexus distribution under `$(NEXUS_HOME)/conf/examples`. Before you customize your Nexus configuration to serve SSL, keep in mind the following:

- Any custom Jetty configuration must be contained in the `$(NEXUS_HOME)/conf/jetty.xml` file, or else in the location referenced by the `jetty.xml` property in `$(NEXUS_HOME)/conf/nexus.properties` (in case you've customized this location).
- While the instructions below will work with Nexus Open Source, these instructions assume the filesystem of Nexus Professional. If you are missing Jetty JAR files, you should obtain them from the Jetty project page: <http://www.eclipse.org/jetty/>

23.3.1 Configure the Java Keystore

Follow the instructions on the [How to configure SSL](#) page on the Jetty Wiki to setup the appropriate keys and certificates in a form that Jetty can use.

The jetty-util jar and the main Jetty jar can be found in \$NEXUS_HOME/lib. The command line used to import an OpenSSL key+cert in PKCS12 format is:

```
$ keytool -importkeystore -srckeystore <your-certificate.p12> - ←  
srcstoretype PKCS12 -destkeystore <keystore> -deststoretype JKS
```

The command line used to generate an obfuscated password hash is:

```
$ java -cp jetty-util-8.1.8.v20121106.jar org.eclipse.jetty.util.security. ←  
Password <your-password>
```

The OBF line that is the out of the command above will be used in the jetty.xml three times. You'll need to run the previous command three times to generate the obfuscated hash-codes for three passwords:

- The Key Password
- The Trust Store Password
- The Key Store Password

In the next section, the key store and trust store are the same file, with the same password.

23.3.2 Configure Nexus/Jetty to Use the New Keystore

A jetty.xml with the modifications of the jetty.xml required can be found in \$NEXUS_HOME/conf/examples/jetty-ssl.xml, inside your Nexus distribution.

Insert the OBF output from earlier command in the *addConnector* section in the setters for *password*, *keyPassword* and *trustPassword*.

23.3.3 Modify the application-port for SSL connections

has a default configuration that many people would more naturally associate with non-SSL connections. You may wish to modify this port to something like 8443, or even 443 (if you have root access from which to start Nexus). To change this property, modify the \$(basedir)/conf/nexus.properties

Note

You may wish to enable both types of connections, with appropriate rewrite rules between them. Such a configuration is beyond the scope of this section; if you're interested, please refer to the [Jetty Documentation Hub](#) for some information to get you started. Additionally, you may need to add extra port properties to the `nexus.properties` configuration file to accommodate both SSL and non-SSL connections.

23.4 Redirecting Non-SSL Connections to SSL

If you want to make it very easy for people to use your Nexus repository, you will want to configure the automatic redirect from the non-SSL port (default 80) to the SSL port (default 443).

With the recommended practice of using an external proxy server to for SSL, you would setup a redirect in the respective proxy server. With a web server like Apache httpd, you could configure mod_rewrite to automatically redirect browsers to the SSL port, or you can configure Jetty to perform this redirection.

If you however configured Nexus to directly serve SSL as documented in Section [23.3](#), the following instructions can be used to configure Nexus to redirect appropriately.

When this feature is configured, browsers and clients that attempt to interact with the non-SSL port will be seamlessly redirected to the SSL port. If you do not turn on the automatic redirect to SSL, users who attempt to load the Nexus interface via the default port 80 will see a network error.

To do this in Jetty

To enable this feature, configure Jetty to serve SSL directly as demonstrated in Section [23.3](#). After you have configured Jetty to serve SSL directly, you use a custom rewrite rule for Jetty that is bundled with Nexus. Open your `jetty.xml` and replace the existing handler/context-collection declaration with a stand-alone context-collection declaration, by replacing the `handler` section starting with

```
<Set name="handler">
  <New id="Contexts" class="org.eclipse.jetty.handler.←
    ContextHandlerCollection">
  ...

```

with this one:

```
<New id="Contexts" class="org.eclipse.jetty.handler. ↵
    ContextHandlerCollection">
    <!-- The following configuration is REQUIRED, and MUST BE FIRST.
        It makes the Plexus container available for use in the Nexus webapp ↵
        . -->
<Call name="addLifeCycleListener">
    <Arg>
        <New
            class="org.sonatype.plexus.jetty.custom. ↵
                InjectExistingPlexusListener" />
    </Arg>
</Call>

<!-- The following configuration disables JSP taglib support, the
     validation of which slows down Jetty's start-up significantly. -->
<Call name="addLifeCycleListener">
    <Arg>
        <New class="org.sonatype.plexus.jetty.custom.DisableTagLibsListener" ↵
            />
    </Arg>
</Call>
</New>
```

Now, configure the rewrite handler for Jetty by adding the following section just above the line with stopAtShutdown in it:

```
<Set name="handler">
    <New id="Handlers" class="org.eclipse.jetty.handler.rewrite. ↵
        RewriteHandler">
        <Set name="rules">
            <Array type="org.eclipse.jetty.handler.rewrite.Rule">
                <Item>
                    <New id="redirectedHttps"
                        class="org.sonatype.plexus.jetty.custom.RedirectToHttpsRule">
                        <Set name="httpsPort">${application-port-ssl}</Set>
                    </New>
                </Item>
            </Array>
        </Set>
        <Set name="handler">
            <New id="Handlers" class="org.eclipse.jetty.handler. ↵
                HandlerCollection">
                <Set name="handlers">
                    <Array type="org.eclipse.jetty.Handler">
                        <Item><Ref id="Contexts"/></Item>
                        <Item>
                            <New id="DefaultHandler"
                                class="org.eclipse.jetty.handler.DefaultHandler"/></Item>
                        </Item>
                    </Array>
                </Set>
            </New>
        </Set>
    </New>
</Set>
```

```
<Item>
    <New id="RequestLog"
        class="org.eclipse.jetty.handler.RequestLogHandler"/></ -->
    Item>
</Array>
</Set>
</New>
</Set>
</New>
</Set>
```

Modify \$NEXUS_HOME/conf/nexus.properties and add a new property, application-port-ssl. This will allow you to customize both the SSL and non-SSL ports independently:

```
application-port-ssl=8443
```

Chapter 24

Evaluating Nexus Step by Step

24.1 Prerequisites And Preparation

The following guide for evaluating Sonatype Nexus is based on an assumption of installing Nexus itself as well as the various technologies used in the specific evaluation example all on one computer. A more extended evaluation of Nexus in a team environment should follow the instructions for a full installation as documented in the book [Repository Management with Nexus](#). Consult the book for further in-depth documentation about all features of Nexus.

Besides the installation of Nexus itself, various evaluations will need different prerequisites installed on the machine you use for your evaluation. The installation instructions of these technologies follow below. Only follow the instructions referenced from the examples you are interested in. For example you will only need to install Visual Studio and NuGet if you want to evaluate the .Net Integration of Nexus.

24.1.1 A Note About The Operating System

Some of the tasks described are referencing command line calls. Where that is the case, this guide will use Unix typical commands and syntax as used on a bash shell. This is the most common environment on Linux and Mac OSX computers. On Windows machines a bash shell can be installed as well, using e.g. the cygwin system. However the typical usage would be to use the Windows command prompt with slightly different calls. Table 24.1 displays a number of examples for typical tasks carried out in the

evaluations with their bash as well as Windows shell commands.

Table 24.1: Commandline Invocation Examples

Task	Bash Shell	Window Shell
Delete a file	rm filename	del filename
Delete a directory	rm -rf directoryname	rmdir directoryname
Delete a directory in users home directory	rm -rf ~/.m2/repository	rmdir %HOMEPATH%\ .m2\repository
Change to the users home directory	cd ~	cd %HOMEPATH%
Script invocation	./build	build.bat
Gradle Wrapper script invocation	./gradlew	gradlew.bat

24.1.2 Java Runtime

Nexus itself as well as some of the technologies used in the evaluation require a Java runtime or development kit, which is available for most operating systems. We recommend to install the latest Oracle Java 6 or Java 7 JDK available from the [download web page](#) and following the installation instructions on the same site.

After a successful installation, you can verify it by running the command `java -version`, which should result in an output similar to

```
java version "1.7.0_17"
Java(TM) SE Runtime Environment (build 1.7.0_17-b02)
Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)
```

24.1.3 Apache Maven

Apache Maven can be retrieved from the [download page](#) and installed following the instructions available there. We recommend the usage of the latest available Maven 3 version.

After a successful installation you can verify it with running the command `mvn --version`, which should result in an output similar to

```
Apache Maven 3.0.5 (r01de1472...; 2013-02-19 05:51:28-0800)
Maven home: /opt/tools/apache-maven-3.0.5
Java version: 1.7.0_17, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.7.0_17.jdk/Contents/Home ←
/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.8.2", arch: "x86_64", family: "mac"
```

24.1.4 Gradle

The examples in this guide use the so-called Gradle wrapper script. It allows you to get gradle installed automatically by the wrapper and invoke all gradle commands via it. To use it you simple invoke all gradle commands with `./gradlew` on Unix based systems and `gradlew.bat` on Windows instead of `gradle`.

Alternatively Gradle can be retrieved from the [download page](#) and installed following the instructions available in the [User Guide](#). We recommend the usage of the latest available Gradle version.

After a successful installation you can verify it with running the command `gradle -v`, which should result in an output similar to

```
Gradle 1.4

Gradle build time: Monday, January 28, 2013 3:42:46 o'clock AM UTC
Groovy: 1.8.6
Ant: Apache Ant(TM) version 1.8.4 compiled on May 22 2012
Ivy: 2.2.0
JVM: 1.7.0_17 (Oracle Corporation 23.7-b01)
OS: Mac OS X 10.8.2 x86_64
```

24.1.5 Apache Ant And Apache Ivy

Apache Ant can be retrieved from the [download page](#) and installed following the instructions available in the [manual](#). We recommend the usage of the latest available Ant version.

After a successful completion you can verify your Ant installation by running the command `ant -version`, which should result in an output similar to

```
Apache Ant (TM) version 1.8.4 compiled on May 22 2012
```

The example projects used in this guide, contain ant targets in their build files that will automatically install Apache Ivy as part of the build. Alternatively you can retrieve Apache Ivy from the [download page](#) and install it following the [instructions](#).

24.1.6 Microsoft Visual Studio And NuGet

Microsoft Visual Studio and NuGet are needed to evaluate the .Net support of Nexus Professional. There are a number of different Visual Studio distributions. Some of these distributions, may have NuGet already installed, while others do not. Even if your Visual Studio installation is bundled with NuGet, you will want to make sure that you have upgraded to the latest version of the tool.

NuGet is a fast-paced project, and you'll find that new packages available on NuGet Gallery may not be compatible with older versions of the NuGet package manager.

For detailed instructions on installing NuGet in Visual Studio, please go to the [NuGet project's documentation site](#) and refer to the [Installing NuGet](#) instructions.

24.2 Getting Started

This guide is based on the usage of Nexus Professional. A lot of the core features are available in Nexus Open Source as well and some examples are suitable to assess the open source version as well.

- **Step 1:** [Download the Nexus Professional Trial Installer](#) for your operating system
- **Step 2:** Run the Nexus Professional Installer
- **Step 3:** Start Nexus from the Nexus Professional Installer

When Nexus has started just select to open the url in the wizard or go to <http://localhost:8081/nexus> in a browser window.

Note

This guide and the examples reference the Nexus URL <http://localhost:8081/nexus>. If you have chosen to use a different port during the installation of the trial e.g. 9081, simple change the URLs.

In case something goes wrong and Nexus seems to be unavailable, you can examine the Nexus log file to identify problems. It is located in

```
nexus-pro-trial-<version>/logs/wrapper.log
```

Nexus tries to listen on port 8081. If you have another application listening on this port, Nexus will not be able to start.

You can change the port Nexus listens on. Open this file

```
nexus-pro-trial-<version>/conf/nexus.properties
```

Edit the line that looks like this:

```
application-port=8081
```

For example, to access Nexus on port 9090 instead, change the line to

```
application-port=9090
```

Save the file and restart Nexus.

24.2.1 Activating Your Nexus Trial

Once Nexus is started and you are accessing the user interface the first time, you will see the trial activation form. Provide your full name, email address, organization, and location, click on Submit Activation Request.

You will immediately receive an email from Sonatype with the subject “Your Nexus Professional Trial License”, which contains your trial license key. Paste this license key into the license field in the Nexus Professional user interface. Click Activate to activate your 14-day Nexus Professional Trial. Once your trial is activated you will be presented with the Nexus user interface.

24.2.2 Logging Into Nexus As An Administrator

After activating your Nexus install, you can log into Nexus as an administrator. Go to <http://localhost:8081/nexus/> and click on the Login button in the upper right-hand corner of the interface.

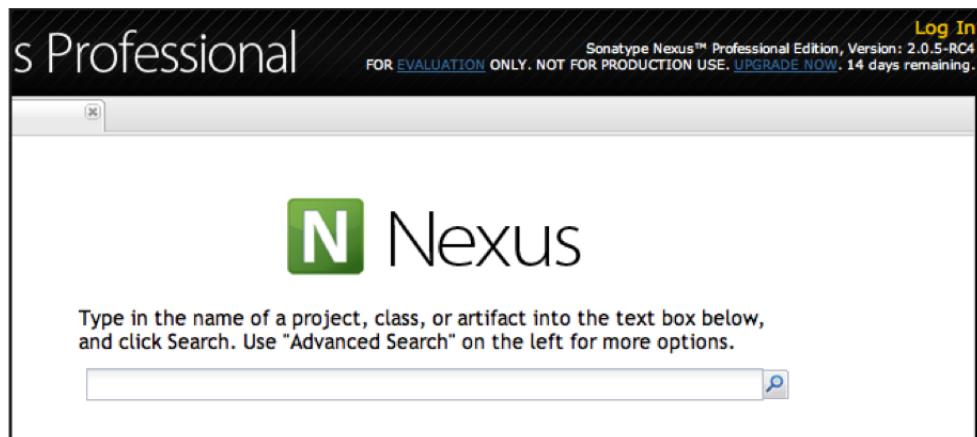


Figure 24.1: Nexus User Interface With Login

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

The Nexus Professional Trial evaluation guide assumes that you are logged in as an administrator.

24.2.3 Getting Started With Your Nexus Professional Evaluation

To make it easier to evaluate Nexus, we've created a set of projects to demonstrate the features of Nexus Open Source and Nexus Professional. These example projects are bundled with the trial installer for your convenience.

In addition they are available as the `nexus-book-examples` project on GitHub at <https://github.com/sonatype/nexus-book-examples> for you to download and inspect separately, if desired. The latest version of all the examples is available as a zip archive at <https://github.com/sonatype/nexus-book-examples/archive/master.zip>.

When you downloaded the trial distribution of Nexus Professional, your server is also preconfigured to demonstrate important features.

The Nexus trial distribution contains the following customizations:

- Nexus has been preconfigured to download the search index from the Central Repository.
- A Staging profile has been configured to demonstrate release management.
- Procurement has been preconfigured so you can quickly define rules for the OSS components.
- Nexus proxies NuGet Gallery so that you can quickly evaluate support for .NET development.

24.3 The Basics: Proxying And Publishing

After a few weeks the importance of having a repository manager is so obvious no one on my team can believe we used to develop software without one.

— Build Engineer *Financial Industry*

If you are new to repository management, the first step is to evaluate the two basic benefits of running a repository manager: proxying and publishing.

You can reap these benefits with any build Java/JVM build system that includes declarative dependency management and understands the Maven repository format. In the following we are going to cover the details for Apache Maven, Gradle and Apache Ant/Apache Ivy based builds. Build tools like SBT, Leiningen, Gant/Grails and others can be configured to do the same and get access to the same benefits.

24.3.1 Proxying Components

If you use a dependency in your software, your build downloads components from a remote repository, such as the [Central Repository](#) and others. Your systems depend on these components. If one of these critical remote repositories becomes unavailable, your productivity can grind to a halt.

This is where Nexus can help. Nexus is pre-configured to proxy the Central Repository and other remote repositories can be easily added. Once set up, Nexus maintains a local cache of the needed components

from the remote repositories for you. Your build is more reliable when all the components you require are cached by Nexus. It is providing you with dramatic efficiency and speed improvements across your entire development effort.

In this example, you will...

- Configure your build to download components from Nexus
- Pre-cache dependencies and build components with an initial build
- Note organization-wide improvements in build reliability

Let's get started using the provided scripts:

The eval bundle includes an installation of Apache Maven as well scripts that isolate your evaluation from the rest of your system and make it extremely easy for you to follow. The Gradle examples use a wrapper script to allow you to simply follow the example. To follow the Ant/Ivy examples you will have to install Apache Ant as explained in Section 24.1.5.

1. Go to the Nexus evaluation guide directory you configured during the Nexus Professional install, which is named evalguide by default and can be found in your users home directory, and run the command

```
$ cd maven  
$ ./build -f simple-project/pom.xml clean install
```

to use Apache Maven or if you want to try Gradle use

```
$ cd gradle/simple-project  
$ ./gradlew build
```

With Apache Ant and Ivy you can run

```
$ cd ant-ivy/simple-project  
$ ant jar
```

2. As the project builds, you will notice that all components are downloaded from your local Nexus instance installed with requests from Apache Maven like

```
Downloading: http://localhost:8081/nexus/content/groups/public/org  
/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5. pom  
Downloaded: http://localhost:8081/nexus/content/groups/public/org
```

```
/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5. ←  
    pom  
(4 KB at 1.3 KB/sec)  
...
```

or from Gradle

```
Download http://localhost:8081/nexus/content/groups/public/org/  
    codehaus/jackson/jackson-core-asl/1.8.0/jackson-core-asl-1.8.0.jar  
Download http://localhost:8081/nexus/content/groups/public/org/  
    codehaus/jackson/jackson-mapper-asl/1.8.0/jackson-mapper-asl-1.8.0. ←  
        jar  
Download http://localhost:8081/nexus/content/groups/public/com/  
    google/sitebricks/sitebricks-converter/0.8.5/sitebricks-converter ←  
        -0.8.5.jar  
...
```

or from Apache Ivy

```
[ivy:retrieve] downloading http://localhost:8081/nexus/content/  
    groups/public/asm/asm-commons/3.2/asm-commons-3.2.jar ...  
[ivy:retrieve] .. (32kB)  
[ivy:retrieve] .. (0kB)  
[ivy:retrieve] [SUCCESSFUL ] asm#asm-commons;3.2!asm-commons.jar (313 ←  
    ms)  
...
```

3. After the build has successfully completed, delete the local Maven repository cache in the eval guide directory and re-run the build as before

```
$ cd maven  
$ rm -rf repository
```

Delete the Gradle cache with

```
$ rm -rf ~/.gradle
```

or the Ivy cache with

```
$ ant clean-cache clean
```

4. Notice how the downloads are occurring much faster. The components are no longer retrieved from the remote repositories before being served by Nexus, but rather are supplied straight from the proxy repository cache in Nexus.
5. To verify that components are being cached in Nexus, open the Repositories panel by clicking on **Repositories** in the left-hand navigation menu. Once the list of repositories is displayed, select Central. Click on the **Browse Storage** tab and observe the tree of components downloaded and successfully cached in Nexus.

Alternatively using your own Apache Maven setup:

1. Ensure that Apache Maven is installed as a prerequisite as documented in Section [24.1.3](#).
2. Go to the Nexus evaluation guide directory you configured during the Nexus Professional install and configure Maven to access Nexus with the provided *settings.xml*. Ensure to back up any existing settings file and adapt the port in the mirror url, if you have chosen to use a different port than 8081 in the Nexus trial installer.

```
$ cp maven/settings/settings.xml ~/.m2/
```

3. Optionally, if you do not want to use the default local repository location of Maven in `~/.m2/repository`, change the localRepository settings in the *settings.xml* file to an absolute path.
4. Build the simple-project

```
$ cd maven/simple-project/  
$ mvn clean install
```

5. And observe the downloads from the Nexus repository as described above
 6. After the build has successfully completed, delete the local Maven repository cache and re-run the build.
- ```
$ rm -rf ~/.m2/repository
```
7. And notice the improved build performance and the cached components in Nexus as described earlier.

**Conclusion**

Your builds will be faster and more reliable now that you are caching components in Nexus and retrieve them from there.. Once Nexus has cached a component locally, there is no need to make another round-trip to the remote repository server. The caching benefits all tools configured to access Nexus.

### **24.3.2 Publishing Components**

Nexus makes it easier to share components internally. How do you distribute and deploy your own applications? Without Nexus, internal code is often distributed and deployed using an SCM, a shared file system, or some other inefficient method for sharing binary components.

---

With Nexus you create hosted repositories, giving you a place to upload your own components to Nexus. You can then feed your components back into the same repositories referenced by all developers in your organization.

### In this example, you will...

- Publish a component to Nexus
- Watch another project download this component as a dependency from Nexus

### Let's get started using the provided scripts:

1. Follow the proxying evaluation example from Section [24.3.1](#)
2. Go to the Nexus evaluation guide directory and publish the simple-project to Nexus with the Maven wrapper script.

```
$ cd maven
$./build -f simple-project/pom.xml clean deploy
```

With your own Maven installation you can use

```
$ cd maven/simple-project/
$ mvn clean deploy
```

To deploy the project with Gradle you can run the commands

```
$ cd gradle/simple-project
$./gradlew upload
```

The equivalent Ant invocation is

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

3. The simple-project has been preconfigured to publish its build output in the form of a jar component to your local instance of Nexus Professional.
4. Observe how the build tools log the deployment to Nexus e.g. Maven

```
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/
org/sonatype/nexus/examples/simple-project/1.0.0-SNAPSHOT/
simple-project-1.0.0-20130311.231302-1.jar
Uploaded: http://localhost:8081/nexus/content/repositories/snapshots/
org/sonatype/nexus/examples/simple-project/1.0.0-SNAPSHOT/
simple-project-1.0.0-20130311.231302-1.jar (3 KB at 38.2 KB/sec)
```

### Gradle

```
Uploading:
org/sonatype/nexus/examples/simple-project/1.0-SNAPSHOT/
 simple-project-1.0-20130306.173412-1.jar
to repository remote at
http://localhost:8081/nexus/content/repositories/snapshots
```

### or Ivy

```
[ivy:publish] :: publishing :: org.sonatype.nexus.examples#simple- ←
 project
[ivy:publish] published simple-project to http://localhost:8081
 /nexus/content/repositories/snapshots/org/sonatype/nexus/examples/
 simple-project/1.0-SNAPSHOT/simple-project-1.0-SNAPSHOT.jar
```

5. To verify that the simple-project component was deployed to Nexus, click on Repositories and then select the Snapshots repository. Select the Browse Storage tab as shown in this illustration.

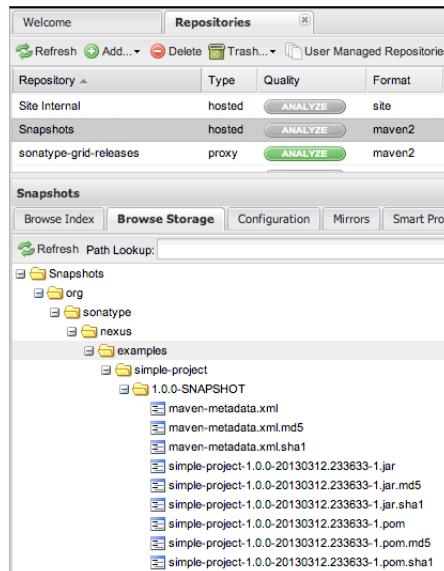


Figure 24.2: Successfully Deployed Components In The Snapshots Repository

6. Once this component has been published, return to the evaluation sample projects directory and run a build of another-project:

```
$ cd maven
$ build -f another-project/pom.xml clean install
```

With your own Maven installation you can use

```
$ cd maven/another-project
$ mvn clean install
```

To build the second project with Gradle, simply use

```
$ cd gradle/another-project
$./gradlew build
```

Perform the same action with Ant using

```
$ cd ant-ivy/another-project
$ ant jar
```

7. This second project has a dependency on the first project declared in the Maven pom.xml with

```
<dependency>
 <groupId>org.sonatype.nexus.examples</groupId>
 <artifactId>simple-project</artifactId>
 <version>1.0.0-SNAPSHOT</version>
</dependency>
```

and in the Gradle build.gradle file as

```
dependencies {
 compile "org.sonatype.nexus.examples:simple-project:1.0.0-SNAPSHOT"
}
```

Ivy declares the dependency in ivy.xml and it looks like this

```
<dependencies>
 <dependency org="org.sonatype.nexus.examples" name="simple-project"
 rev="1.0.0-SNAPSHOT"/>
</dependencies>
```

During the build, it is relying on Nexus when it attempts to retrieve the component from simple-project.

Now that you are sharing components of your projects internally, you do not need to build each others software projects anymore. You can focus on writing the code for your own components and the integration of all components to create a larger software component. In fact it does not even matter, which build tool created the component, since the Maven repository format is understood by all of them.f

---

**Conclusion**

Sonatype Nexus Open Source and Professional can serve as an important tool for collaboration between different developers and different development groups. It removes the need to store binaries in source control or shared file-systems and makes collaboration more efficient.

## 24.4 Governance

### 24.4.1 Identify Insecure OSS Components In Nexus

The Repository Health Check in Nexus Professional turns your repository manager into the first line of defence against security vulnerabilities. Nexus Professional scans components and finds cached components with known vulnerabilities from the Common Vulnerabilities and Exposures (CVE) database. You can get an immediate view of your exposure from the Repository Health Check summary report with vulnerabilities grouped by severity according to the Common Vulnerability Scoring System (CVSS).

As your developers download components, they may be unwittingly downloading components with critical security vulnerabilities, that might expose your applications to known exploits. According to a joint study by Aspect Security and Sonatype released in 2012, Global 500 corporations downloaded 2.8 million flawed components in one year. Nexus becomes an effective way to discover flawed components in your repositories allowing you to avoid falling victim to known exploits.

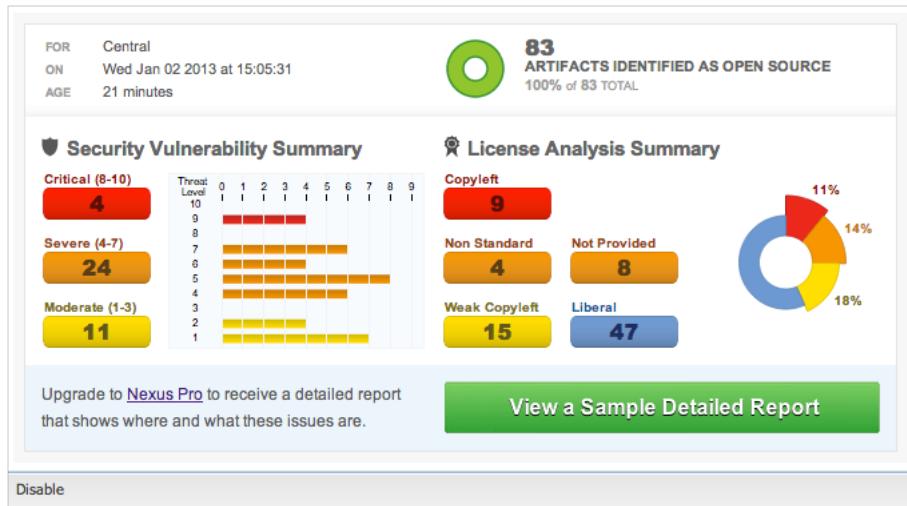


Figure 24.3: Repository Health Check Summary

### In this example, you will...

- Start an analysis of all components proxied from the Central Repository
- Inspect the number of security vulnerabilities found

### Let's get started

1. Follow the proxying examples in Section 24.3 to seed the Central proxy repository of your Nexus instance. These examples include several components with security vulnerabilities and license issues as dependencies.
2. Once your Nexus instance has cached the components, open the Nexus interface, log in as administrator and click on the green Analyze button next to your Central proxy repository
3. After the completion of the analysis, the button will change into an indicator of the number of security and license issues found
4. Hover your mouse over the indicator and Nexus will show you a summary report detailing the number and type of security vulnerabilities present in your repository.
5. Optionally build some of your own applications to get further components proxied and see if additional security issues appear.

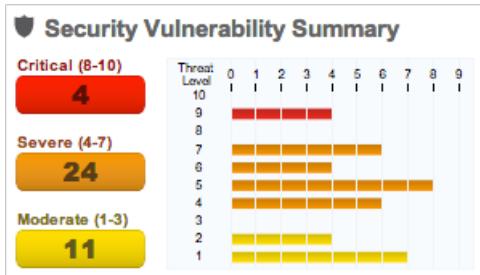


Figure 24.4: Security Vulnerability Summary Display From Repository Health Check

Nexus Professional users gain access to further details about all the components with security vulnerabilities including their repository coordinates to uniquely identify the component as well as links to the vulnerability database records for further details.

### Conclusion

The Repository Heath Check of Nexus allows you to get an understanding of all the security vulnerabilities affecting the components you have proxied into your environment and which might potentially be part of the software you are creating, distributing and deploying in production environments.

#### 24.4.2 Track Your Exposure To OSS Licenses

With Open Source Software (OSS) component usage as the de-facto standard for enterprise application development, the importance of tracking and identifying your exposure to OSS licenses is an essential part of the software development lifecycle. Organizations need tools that let them govern, track, and manage the adoption of open source projects and the evaluation of the licenses and obligations, that are part of OSS development and OSS component usage.

With Nexus Professional's Repository Health Check, your repository becomes more than just a place to store binary components. It becomes a tool to implement policies and govern the open source licenses used in development to create your applications.

**In this example, you will...**

- Start an analysis of all components proxied from the Central Repository
- Inspect the number of license issues found

## Let's get started

1. Follow the proxying examples in Section 24.3 to seed the Central proxy repository of your Nexus instance. These examples include several components with security vulnerabilities and license issues as dependencies.
2. Once your Nexus instance has cached the components, log in to the Nexus interface as administrator and click on the green Analyze button next to your Central proxy repository in the *Repositories* list
3. After the completion of the analysis, the button will change into an indicator of the number of security and license issues found
4. Hover your mouse over the indicator and Nexus will show you a summary report detailing the number and type of license issues of components present in your repository.
5. Optionally build some of your own applications to get further components proxied and see if additional license issues appear.

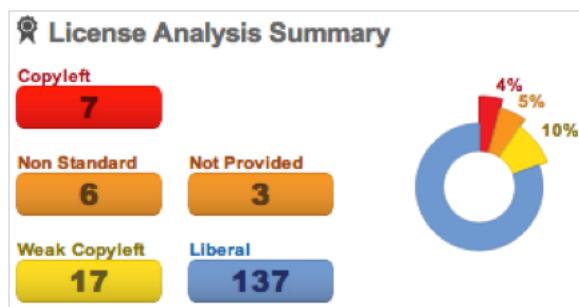


Figure 24.5: License Analysis Summary Display From Repository Health Check

Nexus Open Source and the Trial version show the summary information found by the analysis.

Nexus Professional customers can access a detailed report to identify specific components with known security vulnerabilities or unacceptable licenses. The component lists can be sorted by OSS license or security vulnerabilities, and Nexus Professional provides specific information about licenses and security vulnerabilities. A detailed walkthrough of this report is available on the [Sonatype website](#).

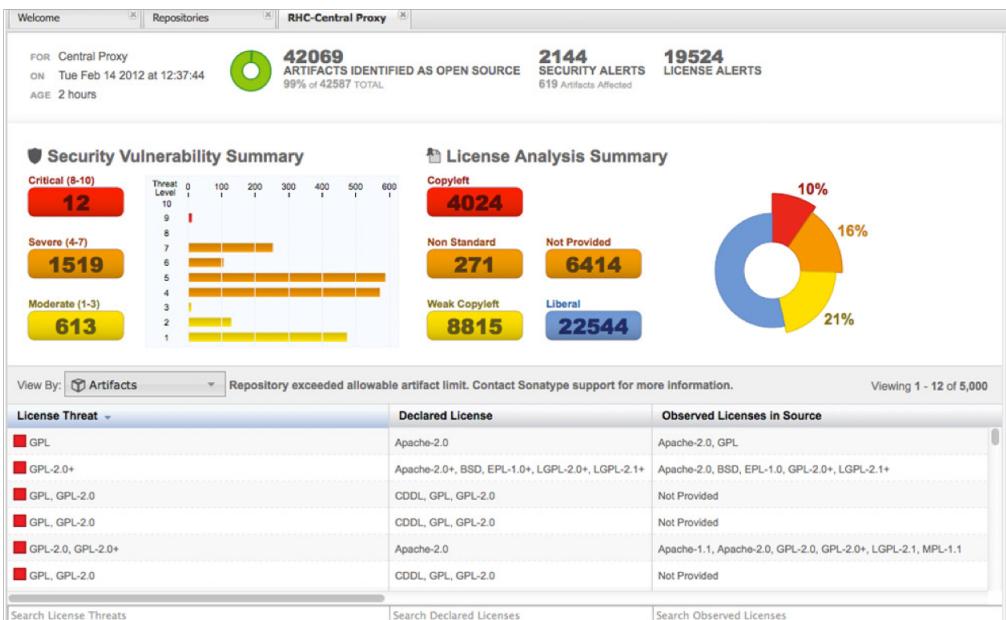


Figure 24.6: Repository Health Check Details With License Issues List

## Conclusion

OSS License compliance and security assessments are not something you do when you have the time. It is something that should be a part of your everyday development cycle. With Nexus Professional's Repository Health Check, it is.

### 24.4.3 Component Procurement

Consider the default behaviour of a proxy repository. Any developer can reference any component stored in a remote repository and cause Nexus to retrieve it from the remote repository. Any developer, anywhere in your organization, can add any dependency to your software. This is possible regardless of the license or security issues of that component or any of its dependencies, that are automatically added as well as - so called transitive dependencies..

If you want control over the components used in a proxy repository, the Nexus Procurement feature was designed to give organizations a mechanism to limit the components that are served from Nexus to your

users. This valuable governance tool can give you the certainty you need to deliver reliable software.

### In this example, you will...

- Configure access rules for components that can be referenced in this Procured version

### Let's get started

1. Your evaluation instance of Nexus has been preconfigured with the following steps
  - a. A hosted repository named *Procured Central* has been created
  - b. Artifact Procurement was enabled with the *Central* proxy repository as the source for the procuring into the newly created *Procured Central* repository
2. Click on *Artifact Procurement* in the *Enterprise* menu in the left hand navigation of the Nexus user interface
3. Select *Procured Central* from the list
4. Define rules for procurement by right clicking on the desired sections of the repository structure including disallowing some components → [Read more...](#)
5. Modify your settings.xml to use the *Repository Path* of the procured repository in the url section of the mirror element
6. Try building a Maven project that references one of the disallowed components, after deleting the local Maven repository.
7. Observe how the procurement rule prevents the build from succeeding, because retrieval of a component is blocked by procurement.

### Conclusion

Procurement is a useful tool, if you are operating in an environment that needs to qualify every single component before it can be used in development or for QA or release builds. Using Procurement you can create explicit white and blacklists of acceptable components and prevent the inclusion of other components.

## 24.5 Process Improvements

### 24.5.1 Grouping Repositories

Once you have established Nexus and set up your build, provisioning system and other tools to connect to Nexus, you can take advantage of Nexus repository groups. The best practice to expose Nexus is to get users to connect to the Public Repositories group as configured in the settings.xml as documented in Section 24.3.1.

When all clients are connecting to Nexus via a group, you can easily provide additional repository content to all users by adding new repositories to the group.

For example imagine a group in your organization is starting to use components provided by the JBoss release repository available at <https://repository.jboss.org/nexus/content/repositories/releases/>. The developers are already accessing Nexus via the public group. All you have to do is to create a new proxy repository for the JBoss release repository and add it to the public group and all developers, CI servers and other tools will have access to the additional components.

Want to add the Grails repositories? No problem - proxy them and add them to the group. Proxy Clojars? No problem. How about a repository of a business partner or supplier, that is protected by user credentials? No problem - the same approach applies.

Another advantage of groups is that you can mix release and snapshot repositories and easily expose all the components via one easy access point.

Besides using the default public group, you can create additional groups that expose other contexts. An example would be to create a group for all approved components including release, snapshots and approved components provisioned via procurement as detailed in Section 24.4.3.

#### Conclusion

Using groups allows you to expose multiple repositories, mix snapshot and release components and easily administrate it all on the Nexus server. This allows you to provide further components to your developers or other users, without requiring a change on these client system, tremendously simplifying the administration effort.

## 24.5.2 Staging A Release With Nexus

When was the last time you did a software release to a production system? Did it involve a QA sign-off? What was the process you used to re-deploy, if QA found a problem at the last minute? Developers often find themselves limited by the amount of time it takes to respond and create incremental builds during a release.

The Nexus Staging Suite changes this by providing workflow support for binary software components. If you need to create a release component and deploy it to a hosted repository, you can use the Staging Suite to post a release, which can be tested, promoted, or discarded, before it is committed to a release repository.

### In this example, you will...

- Configure a project to publish its build output component to Nexus
- Deploy a release and view the deployed component in a temporary staging repository
- Promote or discard the contents of this temporary staging repository

### Let's get started using the provided scripts:

1. This example assumes that you have successfully deployed the simple-project as documented in Section [24.3.1](#).
2. Inspect the pre-configured *Example Release Profile* staging profile by selecting it from the list available after selecting *Staging Profiles* in the the *Build Promotion* menu in the left hand navigation
3. Notice that the version of the simple-project in the pom.xml ends with -SNAPSHOT. This means that it is in development.
4. Change the version of the simple project to release version by removing the -SNAPSHOT in a text editor or run the command

```
$./build -f simple-project/pom.xml versions:set -DnewVersion=1.0.0
```

5. Publish the release to the Nexus Staging suite with

```
$./build -f simple-project/pom.xml clean deploy
```

6. To view the staging repository, click on *Staging Repositories* in the *Build Promotion* menu and you should see a single staging repository as shown in this illustration.

7. Click on *Close* to close the repository and make it available via the public group.
8. Experiment with Staging, at this point you can:
  - a. Click on *Drop* to discard the contents of the repository and staging another release.
  - b. Click on *Release* to publish the contents of the repository to the Release repository.
9. Once you release the staging repository, you will be able to find the release components in the *Releases* hosted repository

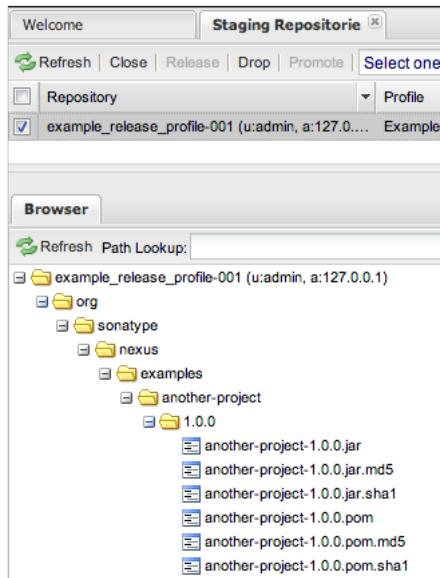


Figure 24.7: Closing A Staging Repository In Nexus User Interface

The individual transactions triggered by closing, dropping, promoting or releasing a staging repository can be enriched with email notifications as well as staging rule inspections of the components.

#### Alternatively using your own Apache Maven setup:

1. Follow the steps described above with the modification of setting the new version with

```
$ cd maven/simple-project
$ mvn versions:set -DnewVersion=1.0.0
```

2. And publishing to the Nexus Staging suite with

```
$ mvn clean deploy
```

### Conclusion

Staging gives you a standard interface for controlling and managing releases. A collection of related release components can be staged for qualification and testing as a single atomic unit. These staged release repositories can be discarded or released pending testing and evaluation.

#### 24.5.3 Hosting Project Web Sites

Nexus Professional and Open Source can be used as a publishing destination for project websites. You don't have to worry about configuring another web server or configuring your builds to distribute the project site using a different protocol. Simply point your Maven project at Nexus and deploy the project site.

With Nexus as a project's site hosting solution, there's no need to ask IT to provision extra web servers just to host project documentation. Keep your development infrastructure consolidated and deploy project sites to the same server that serves your project's components.

You can use this feature internally, but it is even better suited if you are providing an API or components for integration. You can host full project web sites with JavaDoc and any other desired documentation right with the components you provide to your partners and customers.

**In this example, you will...**

- Create a Hosted repository with the Maven Site provider
- Configure your project to publish a web site to Nexus Professional

**Let's get started using the provided scripts:**

1. Create a hosted repository with the *Site* format and a *Repository ID* called *site* → [Read more...](#)
2. Deploy the simple-project component and site to Nexus

```
$./build -f simple-project/pom.xml clean deploy site-deploy
```

3. Browse the generate site on Nexus at <http://localhost:8081/nexus/content/sites/site/>
4. Optionally configure your own Maven project to deploy a site to Nexus → [Read more...](#)
5. And publish it to Nexus → [Read more...](#)

#### Alternatively using your own Apache Maven setup:

1. Follow the steps described above with the modification of deploying the site with

```
$ cd maven/simple-project
$ mvn clean deploy site-deploy
```

#### Conclusion

If your projects need to publish HTML reports or a project web site, Nexus provides a consolidated target for publishing project-related content.

#### 24.5.4 Process and Security Improvements With Maven Settings Management And User Token

The Maven settings.xml file plays a key role for retrieving as well as deploying components to Nexus. It contains <server> sections that typically contain the username and password for accessing Nexus in clear text. Especially with single sign on (SSO) solutions used for Nexus authentication, this is not desirable. In addition security policies often mean that the file regularly needs to be updated.

The User Token feature of Nexus Professional allows you to replace the SSO username and password with Nexus specific tokens that are autogenerated and managed by Nexus.

Furthermore the Nexus Maven Settings Management allows you to manage Maven Settings. Once you have developed a Maven Settings template, developers can connect to Nexus Professional using the Nexus M2Settings Maven plugin, which will take responsibility for downloading a Maven Settings file from Nexus and replacing the existing Maven Settings on a local workstation. It can be configured to automatically place your user tokens in the settings.xml file.

**In this example, you will...**

- Explore the configuration of a Maven Settings template in Nexus Professional
- Activate and access your user token

**Let's get started**

1. Log into Nexus as administrator and access the Maven Settings administration via the item in the Enterprise menu

2. Press the Add button, provide a name and edit the new settings file

3. Add the server section

```
<servers>
 <server>
 <id>nexus</id>
 <!-- User-token: ${userToken} -->
 <username>${userToken.nameCode}</username>
 <password>${userToken.passCode}</password>
 </server>
</servers>
```

4. Read more about potential configuration and usage in Manage Maven Settings Templates → [Read more...](#)

5. Downloading the settings template requires Nexus running via https and can then be performed with

```
mvn org.sonatype.plugins:nexus-m2settings-maven-plugin:1.4:download -Dsecure=false
```

and following the prompts

6. Note that the *secure* option is set to *false* for your evaluation. The plugin would otherwise abort for using the insecure http protocol once you provide your evaluation Nexus url of `http://localhost:8081/nexus`. For a production usage we recommend using the secure https protocol for your Nexus deployments.

7. Find out more about the usage in Download Settings from Nexus → [Read more...](#)

8. Activate User Token in the configuration in the *Security* menu *User Token* administration by checking the *Enabled* box and pressing the *Save* button

9. Access your *User Profile* in the drop down of your user name in the top right hand corner of the Nexus user interface

10. Use the drop down in the *Profile* panel to access *User Token*
11. In the *User Token* screen press *Access User Token*, provide your username and password again and inspect the tokens in the pop up dialog

### Conclusion

The distribution of settings.xml is an crucial part of the roll-out of Nexus usage. With the help of the the Nexus M2Settings Maven Plugin and the server side settings template it is possible to automate initial distribution as well as updates to the used settings.xml files. The User Token feature allows you to avoid having SSO credentials expose in your file system at all and replaces them with

## 24.6 .NET Integration

### 24.6.1 Consume .NET Components From NuGet Gallery

The NuGet project provides a package and dependency management solution for .NET developers. It is integrated directly into Visual Studio and makes it easy to add, remove and update libraries and tools in Visual Studio and on the command line for projects that use the .NET Framework. Nexus can act as a proxy between your developer's Visual Studio instances and the public NuGet Gallery.

When you configure Nexus Professional to act as a proxy for NuGet Gallery you gain a more reliable build that depends on locally cached copies of the components you depend on. If NuGet Gallery has availability problems, your developers can continue to be productive. Caching components locally will also result in a faster response for developers downloading .NET dependencies.

#### In this example, you will...

- Configure your Visual Studio instance to download NuGet packages from your local Nexus server
- Consume components from NuGet Gallery via Nexus

#### Let's get started

Your Nexus Professional Trial instance has been preconfigured with the following NuGet repositories:

- A Proxy Repository for NuGet Gallery
- A Hosted Repository for your internal .NET components
- A Group which combines both the NuGet Gallery Proxy and the Hosted NuGet Repository

Repository	Type	Quality	Format
NuGet Group	group	<b>ANALYZE</b>	nuget
Public Repositories	group	<b>ANALYZE</b>	maven2
3rd party	hosted	<b>ANALYZE</b>	maven2
Apache Snapshots	proxy	<b>ANALYZE</b>	maven2
Central	proxy	<b>ANALYZE</b>	maven2
Central M1 shadow	virtual	<b>ANALYZE</b>	maven1
Codehaus Snapshots	proxy	<b>ANALYZE</b>	maven2
Local NuGet Packages	hosted	<b>ANALYZE</b>	nuget
NuGet.org	proxy	<b>ANALYZE</b>	nuget
Releases	hosted	<b>ANALYZE</b>	maven2

Figure 24.8: NuGet Repositories In Repository List

To consume .NET components from Nexus Professional you will need to install the NuGet feature in Visual Studio as referenced in Section 24.1.6 and configure it appropriately:

1. Open Nexus Professional, click on Repositories in the left-hand navigation menu and locate the *NuGet Group* repository group. This is the aggregating group from which Visual Studio should download packages. Click on this repository group in the list of repositories.
2. Select the NuGet tab below the list of repositories with the NuGet Group selected and copy the URL in the *Package Source* field to your clipboard. The value should be <http://localhost:8081/nexus/service/local/nuget-group/>
3. Now in Visual Studio, right-click on a Visual Studio project and select *Add Library Reference*
4. In the *Add Library Package Reference* click on the *Settings* button in the lower left-hand corner.
5. This will bring up an *Options* button. Remove the initial NuGet repository location and replace it with a reference to your Nexus instance. Clicking *Add* to add the reference to your Nexus Instance.

6. Click on *OK* to return to the *Add Library Package Reference* dialog.
7. Select the *Online* item in the left-hand side of the dialog, at this point Visual Studio will interrogate your Nexus instance for a list of NuGet packages.
8. You can now locate the package you need and install it.
9. To verify that the NuGet package components are being served from Nexus you can return to the Nexus web interface and browse the local storage of your NuGet proxy repository.

---

**Note**

Watch [this video](#) of the steps being performed in Visual Studio.

---

The above instructions were created using Visual Studio 10 Web Developer Express. Your configuration steps may vary if you are using a different version of Visual Studio.

**Conclusion**

When your developers are consuming OSS .NET components through a Nexus proxy of NuGet gallery your builds will become more stable and reliable over time. Every component will be only downloaded to Nexus once and every following download will enjoy the performance and reliability of a local download from the Nexus cache.

### 24.6.2 Publish And Share .NET Components With NuGet

Nexus Professional can improve collaboration and control while speeding .NET development. NuGet defines a packaging standard that organizations can use to share code.

If your organization needs to share .NET components you can publish these components to a hosted NuGet repository on Nexus Professional. This makes it easy for projects within your organization to start publishing and consuming NuGet packages using Nexus as a central hub for collaboration.

Once NuGet packages are published to your Nexus Professional instance they are automatically be added to the NuGet repository group and your internal packages will be as easy to consume as packages from NuGet Gallery.

**In this example, you will...**

---

- Publish NuGet packages to a Hosted NuGet repository
- Distribute custom .NET components using Nexus Professional

**Let's get started:**

1. Follow the example from Section [24.6](#) to set up proxying of NuGet packages from Nexus
2. Activate the NuGet API Security Realm → [Read more...](#)
3. Create a NuGet Package in Visual Studio → [Read more...](#)
4. Publish a NuGet Package to Nexus Professional → [Read more...](#)

**Conclusion**

Once NuGet packages are published to your Nexus Pro instance and are available via a NuGet repository group, your internal packages will be as easy to consume as packages from NuGet Gallery.

This will greatly improve sharing of components and reuse of development efforts across your teams and allow you to modularize your software.

## 24.7 Security

### 24.7.1 Integration With Enterprise LDAP Solutions

Organizations with large, distributed development teams often have a variety of authentication mechanisms: from multiple LDAP servers with multiple User and Group mappings, to companies with development teams that have been merged during an acquisition. Nexus Professional's Enterprise LDAP support was designed to meet the most complex security requirements and give Nexus administrators the power and flexibility to adapt to any situation.

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

## Let's get started

Read more about [configuring Enterprise LDAP](#) and learn about

- Configuring LDAP Caching and Time out
- Configuring and Testing LDAP Fail over
- Using LDAP User and Group Mapping Templates for Active Directory, POSIX with Dynamic or Static Groups or Generic LDAP Configuration

With Enterprise LDAP support in Nexus Professional you can

- Cache LDAP authentication information
- Use multiple LDAP servers, each with different User and Group mappings
- Use LDAP servers with multiple backup instances and test the ability of Nexus to fail over in the case of an outage
- Augment the roles from LDAP with Nexus specific privileges

### Conclusion

When you need LDAP integration, you will benefit from using Nexus Professional. Nexus Professional can support the largest development efforts with some of the most complex LDAP configurations including multiple servers and support for geographic fail over and does so in production at many users every day.

### 24.7.2 Single Sign On (SSO) Support With Atlassian Crowd

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

**In this example, you will...**

- Install the Atlassian Crowd Nexus plugin
-

- Configure an Atlassian Crowd Authentication and Authorization Realm

### Let's get started

1. Install the Atlassian Crowd Nexus Plugin → [Read more...](#)
2. Configure the Crowd Plugin → [Read more...](#)
3. Add the Crowd Authentication Realm → [Read more...](#)
4. Map Crowd Groups and Roles to Nexus → [Read more...](#)

### Conclusion

If you've consolidated authentication and access control using Atlassian Crowd, take the time to integrate your repository manager with it as well. Nexus Professional's support for Crowd makes this easy.

## 24.8 Enterprise Deployments

### 24.8.1 Scaling Nexus Deployments For Distributed Development

Avoid downtime by deploying Nexus in a highly available configuration! With the Nexus Professional feature Smart Proxy two distributed teams can work with local instances of Nexus that will inform each other of new components as they are published. Smart Proxy is an enhanced proxy setup with push notifications and potential prefetching of components. It allows you to keep proxy keeps repositories on multiple Nexus servers in sync without sacrificing performance.

A team in New York can use a Nexus instance in New York and a team in Sydney can use an instance in Australia. If a component has been deployed, deleted, or changed, the source repository notifies the proxy. Both teams are assured that Nexus will never serve stale content. This simple mechanism makes it possible to build complex distributed networks of Nexus instances relying on this publish/subscribe approach.

**In this example, you will...**

---

- Setup two instances of Nexus Professional
- Configure one instance to proxy the hosted instances of the other instance
- Configure the proxying instance to subscribe to Smart Proxy events

### Let's get started

1. Enable Smart Proxy Publishing → [Read more...](#)
2. Establish Trust between Nexus Instances → [Read more...](#)
3. Configure Smart Proxy → [Read more...](#)

### Conclusion

With Smart Proxy, two or more distributed instances of Nexus can stay up-to-date with the latest published components. If you have distributed development teams, Smart Proxy will allow both teams to access a high-performance proxy that is guaranteed to be up-to-date.

## Appendix A

# Contributing to the Nexus Book

This appendix covers the basics of contributing to the book you are currently reading. This book is an open source project, you can participate in the writing effort if you have an idea for documentation. Sonatype's books are different: they are open writing efforts and we see documentation contributions as having equal value to code contributions. If you are interested in our technology, we'd welcome your contribution.

### A.1 Contributor License Agreement (CLA)

In order to contribute to the Nexus book, you will first need to fill out a contributor license agreement. This is a legal agreement between you and Sonatype which ensures that your contributions are not covered by any other legal requirements. Sonatype requires contributors to sign this agreement for all major contributions which are 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 JIRA instance. 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 fax it to: (650) 472-9197

---

## A.2 Contributors, Authors, and Editors

As with any open source effort, the contributors to the Nexus book are grouped into a simple hierarchy. Sonatype's writing efforts are loosely structured, but we have found it necessary to define some formal categories for contributors.

### Reviewers

Many individuals have read the book and taken the time to report typos and bugs. Reviewers are always credited in the Foreword of the book and they make an important contribution to the quality of the book.

### Contributors

Contributors are individuals who have contributed one or more sections to the book. Many contributors make a one time contribution to a particular section or collection of sections. Contributors are always credited in the Foreword of the book, and if a contributor sustains a constant level of contribution which adds up to the equivalent of an entire chapter, a contributors name will be added to the list of contributing authors.

### Authors

Authors have made a significant contribution to the book equal to the equivalent of one or more chapters. A long-time contributor can also be transitioned to the status of Author at the discretion of an Editor. Authors are often given editorial control over specific chapters or sections of a book, working with Contributors to review, accept, and refine contributions to defined sections of the book.

### Editors

An Author can also be an Editor. Each book has at least one editor (and ideally no more than two Editors at any time). On Sonatype Open book projects, Editors are the arbiters of content, they review content submissions and make final decisions about content direction.

For each of these levels, the adjective "Active" can be used if a contributor, author, or editor has been active during the previous 12 months. If you have any questions about contributor status, send any enquiries to [book@sonatype.com](mailto:book@sonatype.com)

## A.3 How to Contribute

The source code for the book is hosted on GitHub in the [nexus-book](#) project. Instructions on tools used to author content as well as building the book and more can be found there.

---

## Appendix B

# Copyright

Copyright © 2011 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 rel="license"
```

```
href="http://creativecommons.org/licenses/by-nc-nd/3.0/us/"> >
CC BY-NC-ND 3.0
</div>
```

When downloaded or distributed in a jurisdiction other than the United States of America, this work shall be covered by the appropriate ported version of Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 license for the specific jurisdiction. If the Creative Commons Attribution-Noncommercial-No Derivative Works version 3.0 license is not available for a specific jurisdiction, this work shall be covered under the Creative Commons Attribution-Noncommercial-No Derivative Works version 2.5 license for the jurisdiction in which the work was downloaded or distributed. A comprehensive list of jurisdictions for which a Creative Commons license is available can be found on the Creative Commons International web site at <http://creativecommons.org/international>

If no ported version of the Creative Commons license exists for a particular jurisdiction, this work shall be covered by the generic, unported Creative Commons Attribution-Noncommercial-No Derivative Works version 3.0 license available from <http://creativecommons.org/licenses/by-nc-nd/3.0/>

## C.1 Creative Commons BY-NC-ND 3.0 US License

### Creative Commons Attribution-NonCommercial-NoDerivs 3.0 United States

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

#### 1. Definitions

- a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with one or more other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

- b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
  - c. "Licensor" means the individual, individuals, entity or entities that offers the Work under the terms of this License.
  - d. "Original Author" means the individual, individuals, entity or entities who created the Work.
  - e. "Work" means the copyrightable work of authorship offered under the terms of this License.
  - f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.
  3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:
    - a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works; and,
    - b. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(d) and 4(e).

1. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
  - a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource

Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of a recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. When You distribute, publicly display, publicly perform, or publicly digitally perform the Work, You may not impose any technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by Section 4(c), as requested.

- b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
- c. If You distribute, publicly display, publicly perform, or publicly digitally perform the Work (as defined in Section 1 above) or Collective Works (as defined in Section 1 above), You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear, if a credit for all contributing authors of the Collective Work appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this clause for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

## 2. Representations, Warranties and Disclaimer

---

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR

OFFERS THE WORK AS-IS AND ONLY TO THE EXTENT OF ANY RIGHTS HELD IN THE LICENSED WORK BY THE LICENSOR. THE LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MARKETABILITY, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

1. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
2. Termination
  - a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works (as defined in Section 1 above) from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
  - b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.
3. Miscellaneous
  - a. Each time You distribute or publicly digitally perform the Work (as defined in Section 1 above) or a Collective Work (as defined in Section 1 above), the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
  - b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
  - c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

- d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

## C.2 Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at <http://creativecommons.org/>.