<http://bcho.tistory.com/790>

## [maven nexus 설치](http://bcho.tistory.com/790)

[ALM/Build Automation (빌드 자동화)](http://bcho.tistory.com/category/ALM/Build%20Automation%20%28%EB%B9%8C%EB%93%9C%20%EC%9E%90%EB%8F%99%ED%99%94%29) | 2013.09.05 23:47 | Posted by 조대협

**Nexus 설치 및 Maven 연동**

조대협

Nexus는 maven에서 사용할 수 있는 가장 널리 사용되는 무료 repository 중의 하나이다. www.sonatype.com 에서 다운로드 받아서 설치할 수 있다.

Local에 nexus를 설치하게 되면, 외부로 부터 dependency를 끌어 오는 수고를 덜고, local nexus를 proxy (cache)로 사용함으로써 빠르게 라이브러리들을 끌어 올 수 도 있고, 반대로 개발팀내에서 사용하는 공통 라이브러리들을 local nexus에 배포해서 팀간에 공유할 수 있다.

또한 사용자 계정 지정을 통해서 repository에 대한 접근 정책을 정의할 수 도 있다.

Nexus는 repository의 용도와 목적에 따라서 몇 가지로 나눌 수 있는데, 대표적으로 다음과 같은 종류 들이 있다.

①   Snapshots : 빌드등 수시로 릴리즈 되는 바이너리를 배포 하는 장소

②   Releases : 정식 릴리즈를 통해서 배포되는 바이너리를 저장하는 저장소

③   3rd party : 벤더등에서 배포하는 (Oracle,IBM등) 바이너리를 저장해놓는 장소로 특정 솔루션등을 사용할때, 딸려 오는 라이브러리등을 여기에 놓고 사용한다

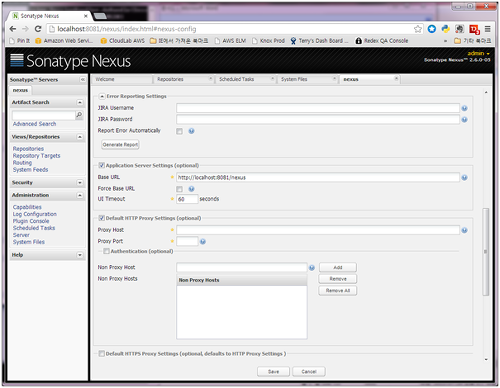
④   Proxy Repository : 원격에 원본 repository가 있는 경우, Local에 캐쉬 용도로 사용한다.

⑤   Virtual Repository : Repository Group은 몇 개의 repository를 하나의 repository로 묶어서 단일 접근 URL을 제공한다.

여기서는 가장 널리 사용하는 local repository로 설정 하는 시나리오와 함께, 외부 repository에 대한 proxy 시나리오로 사용하는 설정을 소개한다.

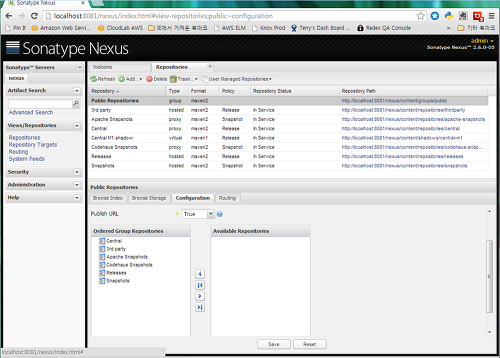
## 설치

http://www.sonatype.com 에서 nexus 무료 버전을 다운로드 받아서 설치한다. 초기 디폴트 로그인 계정과 비밀번호는 "admin/admin123"이다.



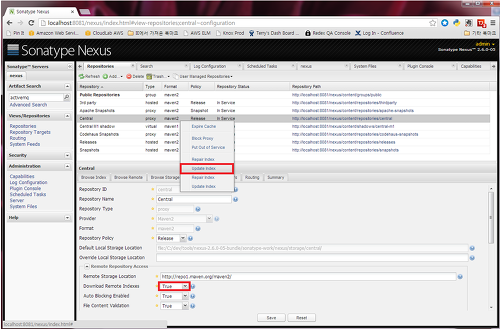
Public Repositories라는 repository 그룹에

local repository (Releases와 Snapshots, 3rd party) 와 proxy repository를 포함시킨다.

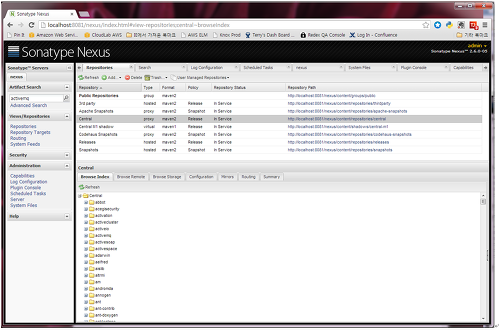


다음으로 Proxy Repository는 Remote Repository의 내용들에 대한 라이브러리 목록(Index)를 Local에 Caching할 수 있도록 되어 있다. 이렇게 하면, nexus의 proxy repository에 실제 바이너리가 내려와 있지 않더라도 목록이 미리 내려와 있기 때문에, nexus의 search 기능을 통해서 검색이 가능하다.

"Maven Central Repository"인 "Central" repository 에 설정을 해보자 "Central repository"를 선택한 후, 메뉴에서 "Download Remote Indexes" 라는 Option을 "True"로 변경한다. 다음 SAVE 버튼으로 저장한 후, 상단 테이블 메뉴에서 "Central" repository를 선택한 후 오른쪽 버튼을 눌러서 팝업 메뉴에서 "update index"를 실행하면, 원격 maven repository에서 라이브러리 목록을 읽어서 업데이트가 된다.



업데이트가 끝나면 "Browse Index" 메뉴에서 라이브러리 목록이 새롭게 업데이트 되어 있는 것을 확인할 수 있다.



## Maven에서 local nexus를 Proxy (Cache) repository로 설정하기

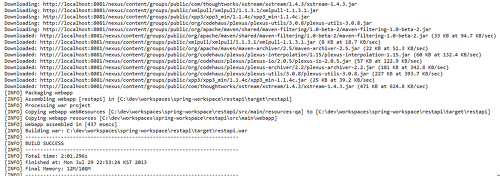
Nexus 설정이 끝났으면 다음으로 maven에 nexus를 rmirroring repository 설정해보자.

$MAVEN\_HOME/.m2/setting.xml

파일에서 <mirrors> section에 아래 내용을 추가하자

|  |
| --- |
| <mirror>        <id>nexus</id>        <mirrorOf>\*</mirrorOf>        <name>Local nexus repository.</name>        <url>http://localhost:8081/nexus/content/groups/public/</url>      </mirror>    </mirrors> |

설정이 끝난후, maven 빌드를 수행하면 maven script가 원격지가 아닌 local에 있는 nexus repository를 통해서 라이브러리를 다운로드 하는 것을 확인할 수 있다.



또한 nexus console을 통해서 "Browse Storage" 메뉴를 통해서 "Central" repository 의 storage 를 보면 빌드에 사용되었던 모든 라이브러리들이 local nexus에 다운로드 받아져 있음을 확인할 수 있다.

* nexus 고급 기능 소개 - <http://bcho.tistory.com/775>

## [maven repository인 nexus pro에 대한 고급 기능 소개](http://bcho.tistory.com/775)

[ALM/Build Automation (빌드 자동화)](http://bcho.tistory.com/category/ALM/Build%20Automation%20%28%EB%B9%8C%EB%93%9C%20%EC%9E%90%EB%8F%99%ED%99%94%29) | 2013.07.31 23:46 | Posted by 조대협

**nexus pro에 대한 고급 기능소개**

**조대협 (bwcho75@지메일)**

nexus는 maven repository로 매우 유명한 솔루션이다. 오픈 소스 버전은 maven을 사용하는 경우에는 거의 필수적으로 사용이 된다고 해도 과언이 아니다.

nexus의 상용 버전인 nexus pro의 경우 CLM (Component Life-cycle Management) 개념을 도입하여, 접근제어나 컴포넌트에 대한 security 나 license risk등을 관리 통제할 수 있다.

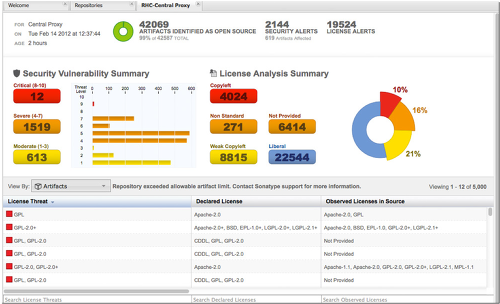
이 글에서는 nexus pro에 대한 몇 가지 고급 기능에 대해서 살펴보고, 이를 통해서 컴포넌트(라이브러리)의 관리가 단순한 중앙 집중형 공유만이 아닌 일종의 life cycle 개념이 있다는 것을 이해하도록 해보자

Nexus의 상용 버전인 Pro 버전에는 단순한 공유나 maven repository cache 용도뿐만이 아니라 조금 더 복잡한 기능의 라이브러리 관리 기능을 제공한다. Nexus에서는 이러한 개념을 CLM (Component Life-cycle Management)이라고 하는데, 라이브러리에 대한 접근 제어나 정책 관리등이 이에 해당한다. 몇가지 대표적인 오픈소스 정책 관리 기능을 살펴보자.

근래에 소프트웨어 개발 패러다임은 오픈소스를 이용한 소프트웨어 개발이 많다. 많은 오픈 소스 라이브러리를 사용하게 되는데, 문제는 각 오픈소스 컴포넌트들의 라이센스 정책이 다르다는 것이다. GPL,Apache,MIT,BSD 등 여러가지 라이센스 정책이 있는데, 라이센스 정책에 따라서, 어떤 오픈 소스는 사용에는 제약이 없지만 배포시 소스코드를 변경해야 하거나, 유료로 비용을 지불해야 하는 경우도 있고, 2.0 버전에서는 멀쩡하게 아무 제약없이 사용할 수 있었던 컴포넌트들이 3.0 버전으로 업그레이드 되면서 제약이 생기는 경우가 있다. (오라클이 인수한 MySQL의 경우 제품에 번들해서 재 배포할 경우 일정의 비용을 지불해야 한다.)  이러한 이유 때문에, 오픈소스 컴포넌트에 대한 라이센스 체크는 점점 필수적인 요건이 되어가는데 문제는 하나의 서비스나 소프트웨어 제품을 개발하는데, 수백개 이상의 라이브러리가 사용된다는 것이고, 각 버전마다 일일이 라이센스를 체크한다는 것은 보통일이 아니다. Nexus는 이런 오픈소스 라이브러리 정책을 repository 차원에서 관리해준다.

## 오픈소스 라이센스 정책 관리

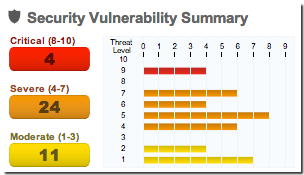
Nexus의 proxy repository를 선택하고, Analysis 라는 버튼을 누르면, 현재 repository에 저장되어 있는 오픈 소스에 대한 라이센스 정책을 분석하여 다음과 같이 보여준다. 이 라이센스 정책에 대한 DB는 nexus 판매사인 sonatype으로 부터 제공된다.



각 라이브러리가 어떤 라이센스 정책을 사용하는지, 그리고 각 라이센스 정책이 문제가 없는 지등을 찾아준다.

## Security 체크

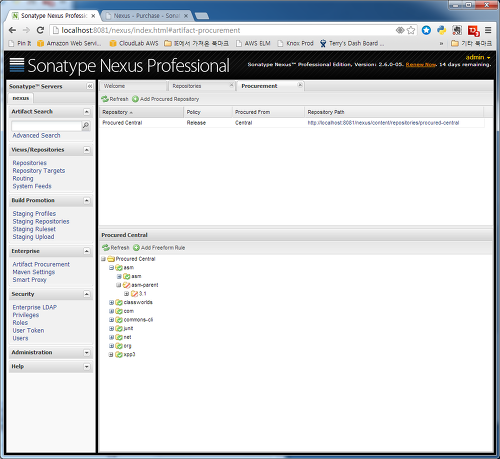
또한, 보안에 위험이 있는 라이브러리등을 찾아서 보안 위험도등을 표시해준다. 2013년 기준으로 struts2에 대해서 아주 큰 보안 위험이 발생된 일이 있었다. 이렇게 중앙의 repository에서 회사내에서 사용하는 라이브러리에 대한 보안 위험성을 감지해서 중앙에서 통제하게 되면, 개별 개발자에 대한 수고도 덜 수 있을 뿐더러, 보안 위험에 대해서 피해갈 수 있는 장점을 가질 수 있다.



이렇게 체크만 할뿐만 아니라, 이렇게 검출된 문제 있는 라이브러리들을 접근하지 못하게 막을 수 있다.

## Procurement

nexus pro에는 “artifact procurement” 라는 기능이 있는데, 이 기능을 사용하면 proxy repository를 만들고, 여기에 속해 있는 라이브러리에 대해서 white list 또는 black list 방식으로 접근을 제어할 수 있다. 아래 그림은 asm-parent 라이브러리에 대해서 모두 접근을 제어 하는 설정을 적용한 예이다.

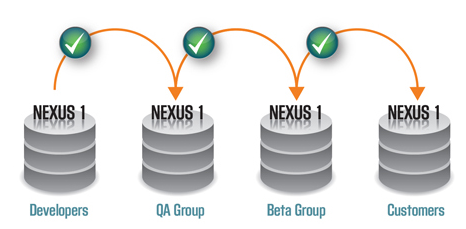


**Staging**

또 다른 재미 있는 기능중에 하나는 staging 개념을 지원한다는 것이다.

즉 개발자가 컴포넌트나 라이브러리를 개발하여 nexus에 배포하면 다른 개발자나 사용자들이 바로 그 라이브러리를 사용하게 하는 것이 아니라, 일종의 워크플로우를 통해서 릴리즈 절차가 끝나면 일발 개발자들이 사용할 수 있도록 프로세스를 조정할 수 있다.

이를 위해서 staging repository라는 개념을 제공하는데, 설명하자면 다음과 같다.



개발자가 컴포넌트를 개발하고, 개발이 끝나면 staging repository로 배포를 진행한다. 배포된 repository는 QA 그룹 엔지니어만 접근이 가능하다. QA 엔지니어는 컴포넌트를 받아서 테스트를 진행하고, 문제가 없으면, 해당 컴포넌트를 베타 테스트 단계로 넘긴다.

베타 테스트 단계에 있는 컴포넌트는 베타 테스트 사용자에게만 접근이 허용되며, 테스트가 끝나면 일반 repository로 이동되어 일반 개발자도 접근이 가능하게 해준다.

이 워크플로우에서 단계별로 넘어갈때, 각 단계 이동별로 정책을 정할 수 있다. 예를 들어 앞서 설명한 security level이 낮은 경우 reject을 하거나, open source 라이센스가 문제가 있는 경우에, reject을 하는 중의 policy를 정의할 수 있다

< 출처 : <http://www.sonatype.com/take-a-tour/nexus-pro-tour-start> >

## [Nexus Repository Manager 구축하기 ( for maven )](http://ncanis.tistory.com/357)

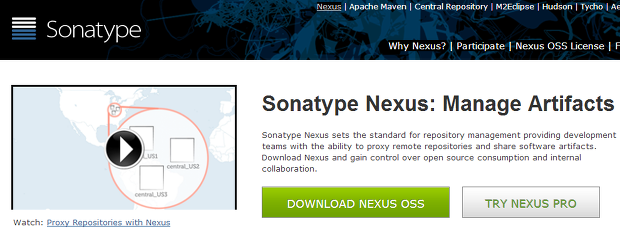
[자바 일반/TIP](http://ncanis.tistory.com/category/%EC%9E%90%EB%B0%94%20%EC%9D%BC%EB%B0%98/TIP)2013.09.23 16:31

* 뷰어
* [댓글로](http://ncanis.tistory.com/357#ycmt)
* [이전글](http://ncanis.tistory.com/358)
* [다음글](http://ncanis.tistory.com/351)

<http://ncanis.tistory.com>

**Nexus Repository Manager 구축하기 ( for maven )**

**1.**[**http://www.sonatype.org/**](http://www.sonatype.org/)**에서 최신 버전 설치**

[](http://www.sonatype.org/nexus/go)

**2. 다운로드 받은 것을 풀고 설치한다.**

D:\Tool\nexus-2.6.2-01\bin\jsw\windows-x86-64 (64비트 os 인경우) 에서

install-nexus.bat 실행 (windows service에 등록되어 구동된다.)

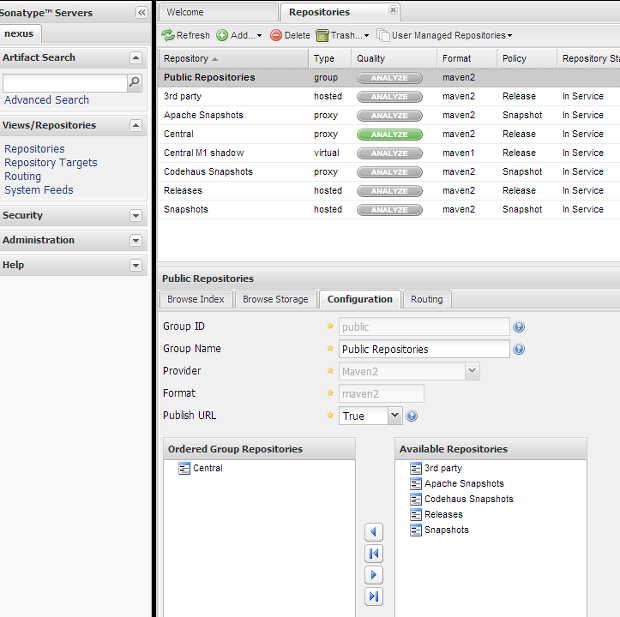
**3. nexus manager web에 접속**

<http://localhost:8081/nexus/index.html#welcome>

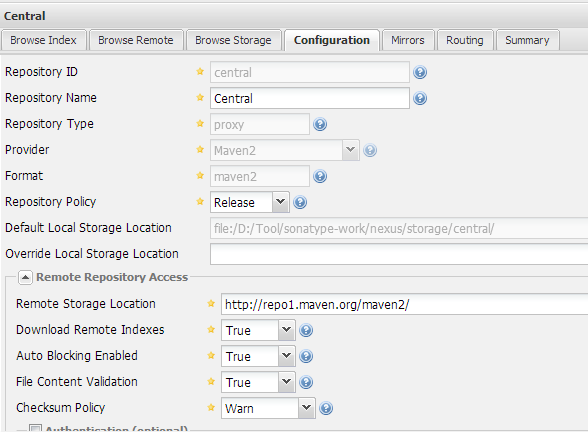
(초기 계정 admin/admin123)

**4. Repository 설정**

   -  이용가능한 Repositories에 Cetral를 제외하고 모두 등록하고 저장한다.

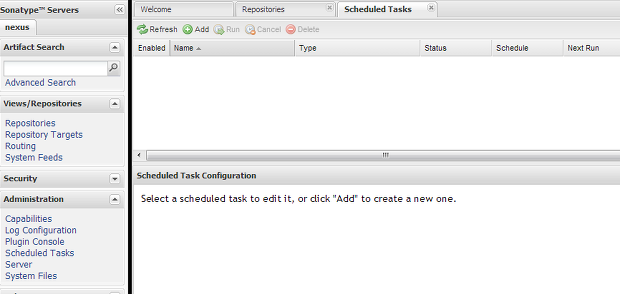


   - Download Remote Indexes를 True로 설정한다. (기본은 리모트에 접속해서 받아오는걸로 되어있다. 이렇게 하면 느리므로, 다운로드 받아올수 있게 설정한다.)

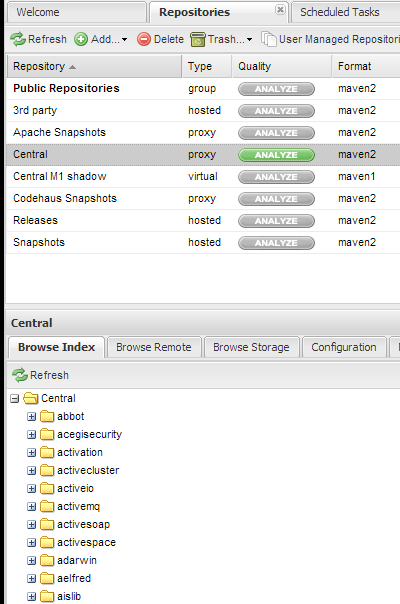


     이 동기화 부분은 좀 느린데, 작성 진행 상황은 아래 위치에서 확인할 수 있다.

    Administration/Scheduled Tasks



     완료가 되면 Repositories/Central 의 Browse Index 를 누르면 제대로 등록된것을 확인할 수 있다.



**5. 이제 Eclispe m2e로 이렇게 셋팅된 Repository를 이용해 라이브러리를 추가해 보자**

- 중앙 Repository와 써드파티 저장소 2개를 추가했다.  
- 작업중인 프로젝트 배포는 Releases 와 Snapshots 주소르 업로드 하도록 설정한다.

<pom.xml>

<repositories>

    <repository>

        <id>**central**</id>

        <url>http://127.0.0.1:8081/nexus/content/groups/public</url>

        <releases><enabled>true</enabled></releases>

        <snapshots><enabled>true</enabled></snapshots>

    </repository>

<repository>

        <id>**thirdparty**</id>

        <url>http://127.0.0.1:8081/nexus/content/repositories/thirdparty</url>

        <releases><enabled>true</enabled></releases>

        <snapshots><enabled>true</enabled></snapshots>

    </repository>

</repositories>

<pluginRepositories>

    <pluginRepository>

        <id>**central**</id>

        <url>http://127.0.0.1:8081/nexus/content/groups/public</url>

        <releases><enabled>true</enabled></releases>

        <snapshots><enabled>true</enabled></snapshots>

    </pluginRepository>

</pluginRepositories>

<distributionManagement>

    <!-- use the following if you're not using a snapshot version. -->

    <**repository**>

      <id>Releases</id>

      <url>http://localhost:8081/nexus/content/repositories/releases/</url>

    </**repository**>

    <!-- use the following if you ARE using a snapshot version. -->

    <**snapshotRepository**>

      <id>Snapshots</id>

      <url>http://localhost:8081/nexus/content/repositories/snapshots/</url>

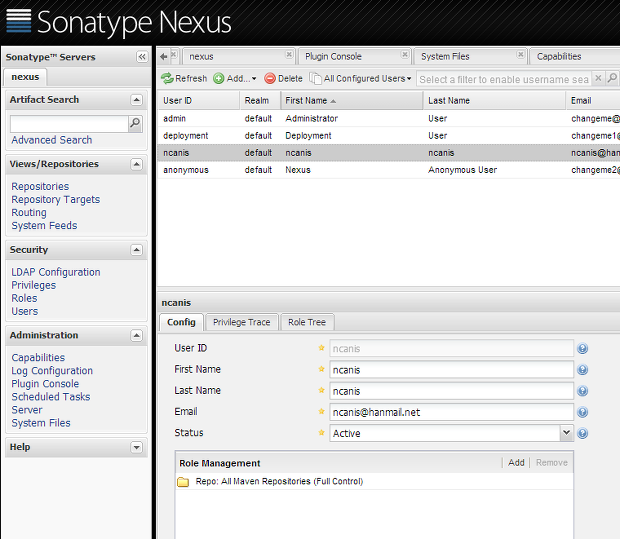
    </**snapshotRepository**>

  </distributionManagement>

- releases, snapshots는 작업중인 프로젝트의 빌드버전을 deploy 해서 nexus에 업로드는 하는것이기 때문에 권한 설정이 필요하다.

  nexus/Security/Users로 가서 이렇게 사용할 계정을 발급하도록 한다. (여기서는 ncanis/ncanis로 발급하였다.)

  권한은 모든 repository에 접속 가능하도록 셋팅했다.(테스트용이므로)



Save 한다.

-위에서 설정한 계정정보를 maven 에도 알려주어야 한다. settings.xml 이 그것이다.

maven repository에 있는 settings.xml에 아래 server tag 부분을 집어 넣는다.

보통 디폴트로 settings.xml 은 C:\Users\ncanis\.m2 폴더 안에 존재하나 없으면 생성해도 된다.

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0

                      http://maven.apache.org/xsd/settings-1.0.0.xsd">

<servers>

    <server>

      <id>Releases</id>

      <username>ncanis</username>

      <password>ncanis</password>

    </server>

    <server>

      <id>Snapshots</id>

      <username>ncanis</username>

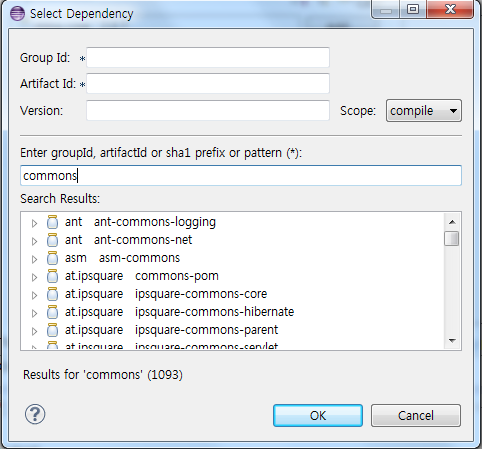
      <password>ncanis</password>

    </server>

  </servers>

</settings>

- Eclipse m2e로 라이브러리를 추가해 본다. commons로 검색해보았는데, 등록되어있는아이디 들이 제대로 나오는것을 확인할 수 있다. 이제 원하는 걸 찾아 OK를 누르면 라이브러리를 사용할 수 있다.



**6. Maven Build 해서 Snapshots/Release nexus 저장소에 배포해보자**

   Eclipse 에서  Run as/Maven Build를 수행한다.

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building ted\_core 1.0-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ ted\_core ---

[debug] execute contextualize

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 2 resources

[INFO]

[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ ted\_core ---

[INFO] Compiling 1 source file to D:\Workspace\dayagam\ted\ted\_core\target\classes

[INFO]

[INFO] --- maven-resources-plugin:2.5:testResources (default-testResources) @ ted\_core ---

[debug] execute contextualize

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] skip non existing resourceDirectory D:\Workspace\dayagam\ted\ted\_core\src\test\resources

[INFO]

[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ ted\_core ---

[INFO] Not compiling test sources

[INFO]

[INFO] --- maven-surefire-plugin:2.10:test (default-test) @ ted\_core ---

[INFO] Tests are skipped.

[INFO]

[INFO] --- maven-jar-plugin:2.3.2:jar (default-jar) @ ted\_core ---

[INFO] Building jar: D:\Workspace\dayagam\ted\ted\_core\target\ted\_core-1.0-SNAPSHOT.jar

[INFO]

[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ ted\_core ---

[INFO] Installing D:\Workspace\dayagam\ted\ted\_core\target\ted\_core-1.0-SNAPSHOT.jar to C:\Users\ncanis\.m2\repository\com\tednet\ted\ted\_core\

1.0-SNAPSHOT\ted\_core-1.0-SNAPSHOT.jar

[INFO] Installing D:\Workspace\dayagam\ted\ted\_core\pom.xml to C:\Users\ncanis\.m2\repository\com\tednet\ted\ted\_core\1.0-SNAPSHOT\ted\_core-1.0

-SNAPSHOT.pom

[INFO]

[INFO] --- maven-deploy-plugin:2.7:deploy (default-deploy) @ ted\_core ---

Downloading: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/1.0-SNAPSHOT/maven-metadata.xml

Downloaded: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/1.0-SNAPSHOT/maven-metadata.xml (774 B at 4.1 KB/sec)

**Uploading**: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/1.0-SNAPSHOT/ted\_core-1.0-20130923.070422-2.jar

**Uploading**: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/1.0-SNAPSHOT/ted\_core-1.0-20130923.070422-2.pom

Uploaded: **http://localhost:8081/nexus/content/repositories/snapshots/**com/tednet/ted/ted\_core/1.0-SNAPSHOT/ted\_core-1.0-20130923.070422-2.pom (3 KB at

 33.1 KB/sec)

**Uploaded**: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/1.0-SNAPSHOT/ted\_core-1.0-20130923.070422-2.jar (175 KB

at 1616.2 KB/sec)

Downloading: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/maven-metadata.xml

Downloaded: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/maven-metadata.xml (288 B at 15.6 KB/sec)

Uploading: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/1.0-SNAPSHOT/maven-metadata.xml

Uploading: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/maven-metadata.xml

Uploaded: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/1.0-SNAPSHOT/maven-metadata.xml (774 B at 16.1 KB/sec)

Uploaded: http://localhost:8081/nexus/content/repositories/snapshots/com/tednet/ted/ted\_core/maven-metadata.xml (288 B at 4.9 KB/sec)

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 5.377s

[INFO] Finished at: Mon Sep 23 16:04:22 KST 2013

[INFO] Final Memory: 19M/181M

[INFO] ------------------------------------------------------------------------

Snapshots URL로 현재 프로젝트가 빌드외서 Deploy된것을 확인할수 있다.

- Nexus에 접속해서 Repositories/Snapshots 선택, Browse Index로 들어가보면 업로드된 파일을 볼수 있다.

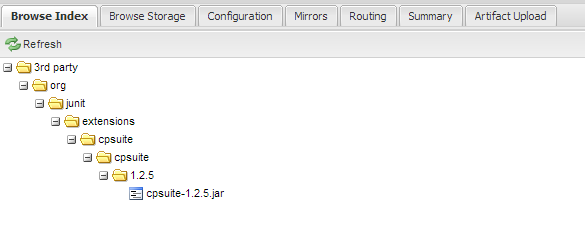
**7. (기타) Maven repository에 존재하지 않은 Thrid party 라이브러리는 어떻게 등록하는가.**

- 이것도 방법이 있다. 물론 Maven 자체에서 로컬로 걸어버리는 수도 있지만, Nexus를 이용하는게 훨씬 간편하고 좋다.

- Nexus/3rd party 리파지토리를 선택한후 Artifact Upload를 선택한다.

- Select Artifact for Upload를 선택한후 다운로드 받은 jar파일을 업로드 한다.

- Upload Artifact 를 실행한다.



여기서는 cpsuite junit extension를 올려보았는데(중앙 저장소에는 원래 없다.) 파일 업로드후 패키지를 분석해서 알아서 groupid, artifact를 생성해서 만들어놓는 것을 Browse Index에서 확인할 수 있다.

NHN,DAUM이 이런 중앙 저장소를 생성해서 사용한다. 다른곳에 문서들이 제대로 작업되어 있는곳이 없어 만들어 보았다.

**8. (기타) Maven 빌드시 Workspace에 있는 프로젝트를 참조하지 못하는경우**

   Eclipse  -> Run As -> Run Configurations... 에서 Profiles 설정의 Resolve Workspace artifacts에 체크하여 빌드한다.

<https://www.lesstif.com/pages/viewpage.action?pageId=20775149>

# 설치 및 설정

## 제품 종류 선택

넥서스는 오픈소스 버전(Nexus OSS)과 상용 버전(Nexus Pro) 두 개의 제품으로 나뉘어 있다. 오픈 소스 버전은 버전 1.9 까지는 AGPL 이었으므로 기업 내부 프로젝트에서 사용할 경우 라이선스 이슈가 발생할 여지가 있었다.

2.x 부터는 Eclipse Public License 로 변경이 되어서 기업 내부의 프로젝트에서 오픈소스 버전을 사용해도 큰 문제는 없다.

다만 상용 버전은 오픈소스보다 더 풍부한 기능을 제공하므로 다음과 같은 기능이 필요하다면 상용 버전 사용도 고려해 볼 만하다.

**고가용성 제공**

Pro 버전은 다수의 인스턴스를 통해 고가용성을 제공하므로 대규모의 팀이나 팀원이 여러 지역에 흩어져 있는 프로젝트라면 네트워크의 장애나 특정 저장소 다운등의 이상 상황에서도 동작할 수 있다.

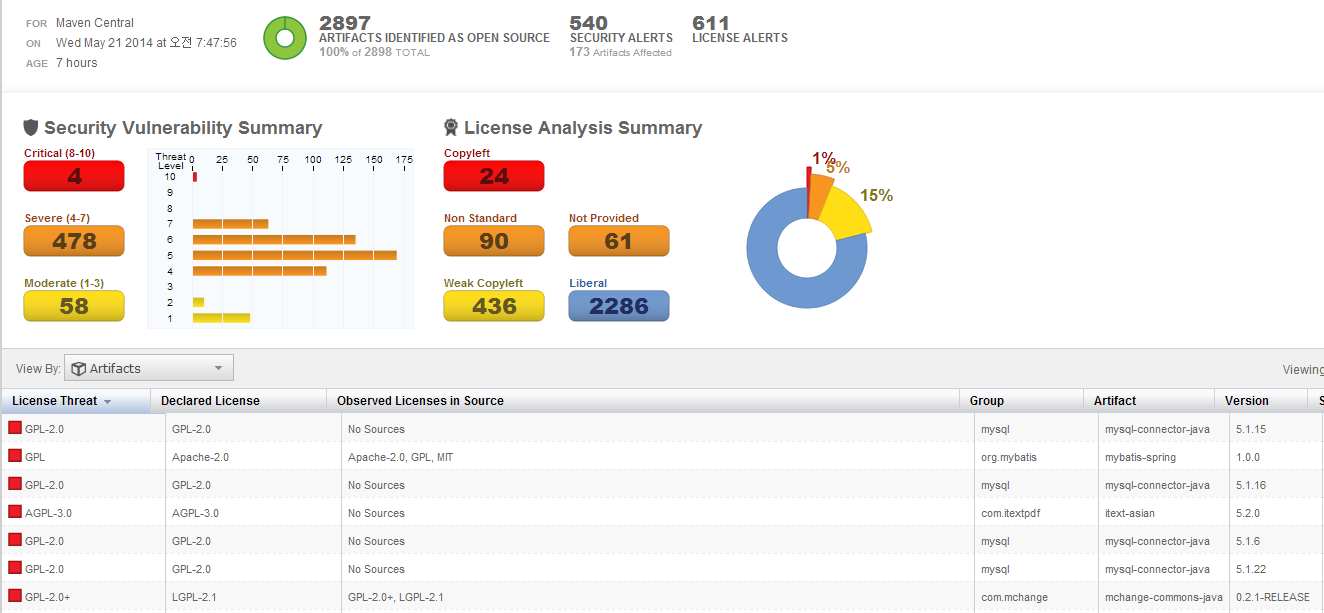
**아티팩트 조달 관리**

회사에서 진행하는 프로젝트이고 상용화 예정이고 소스를 공개하지 않을 것이라면 프로젝트에서 사용하고 있는 외부 아티팩트들의 라이선스가 문제될 만한 것은 없는지 파악해야 한다.

특히 GPL 이나 AGPL 인 아티팩트를 사용할 경우 문제가 될 수 있으므로 이 라이선스를 따르는 아티팩트는 주의해서 사용해야 한다.

하지만 나날이 복잡해지고 수많은 오픈소스를 사용하는 소프트웨어 개발 프로젝트에서 사용하고 있는 아티팩트들의 라이선스를 확인하는 것은 쉬운 일이 아니다. 더군다나 버전이 올라되면서 라이선스가 변경되는 경우도 있다.

Pro 버전은 저장소에 등록된 아티팩트중 기업에서 사용할 경우 문제가 되는게 있는지 확인하고 자세한 정보를 출력하는 기능을 제공하고 있다.



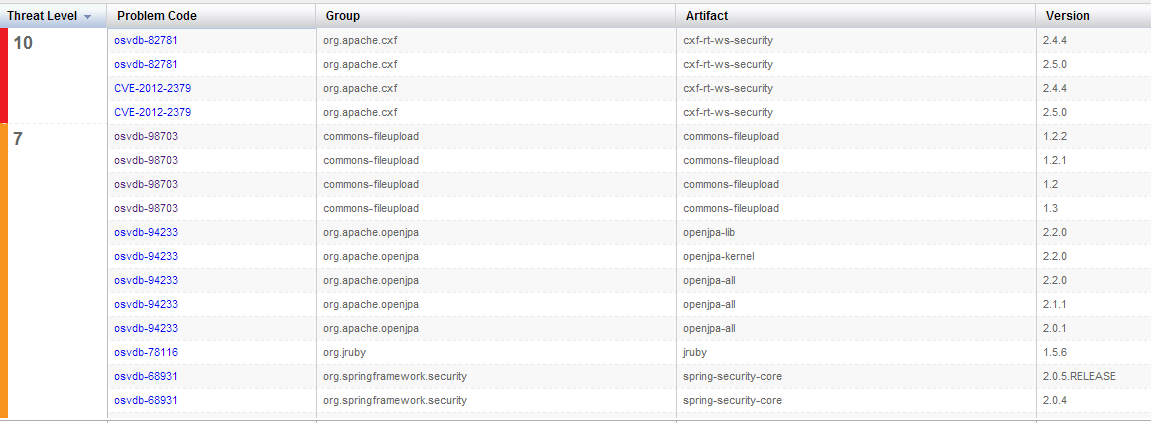
**라이선스 보고서**

**아티팩트 취약점 관리**

개발한 제품은 보안에 신경써서 견고한 제품을 만들었는데 사용한 아티팩트에 취약점이 있다면 제품 자체의 보안성도 저하되게 된다.

하지만 프로젝트에 사용한 수많은 아티팩트가 취약점이 있는지 관리하는 건 보통 일이 아니다.

Pro 버전은 현재 저장소에 등록된 아티팩트중 OSVDB(Open Sourced Vulnerability Database) 나 CVE(Common Vulnerabilities and Exposures) 에 취약점이 등록된 제품을 찾아서 보고해 주므로 제품의 보안 강화에 도움을 줄 수 있다.



**취약점 분석**

**SSO(single Sign On) 지원**

LDAP 이나 atlassian Crowd 등의 솔루션을 통한 SSO 를 지원하므로 계정 및 권한 관리를 손쉽게 할 수 있다.

그외에 NET 플랫폼 지원, OSGi 지원, 기술 지원등의 기능이 제공된다.

<http://www.sonatype.com/nexus/why-nexus-pro/nexus-pro-features>

<http://books.sonatype.com/nexus-book/reference/intro-sect-pro.html>

There are many benefits to using open source software and Nexus OSS is no different. Small teams with simple repository management requirements are happy using the open source version.  
  
It's when organizations are looking to grow their repository manager strategy to incorporate a broader sense of component management across the software development lifecycle, they chose to move to Nexus Pro.

* Do you need your repository to deliver on time, all the time?
* Do you need to know which components in your repository have known security vulnerabilities? How about an analysis of the license obligations associated with each component?
* Do you need to manage development teams across multiple locations?
* Do you want to improve your QA and release process to ensure components meet your requirements before being promoted to the next stage of development?

Then maybe it's time to look at going Pro. Read the top '[10 Reasons to Go Pro](http://s1886.t.en25.com/e/f2?elqSiteID=1886&elqFormName=LNClickthroughTracker&clickthroughURL=http%3A%2F%2Fimg.en25.com%2FWeb%2FSonatypeInc%2F%257Bf44bd2a6-24c4-4289-bf4e-76fbacbcc75e%257D_st_datasheet_10_reasons.pdf&nurtureTrackName=2014%20Nexus&nurtureTrackEmailNumber=3&emailName=Need%20a%20reason%20to%20%27go%20pro%27%3F&linkTitle=10%20Reasons%20to%20Go%20Pro&emailAddress=gsjung@ktnet.com&elq=be17da4c2e74461098d1cee6b6e5dd1a&elqCampaignId=67)' and see how you can start achieving greater value from your repository manager today. Interested in trying Nexus Pro for yourself? [Download our 14 day trial](http://s1886.t.en25.com/e/f2?elqSiteID=1886&elqFormName=LNClickthroughTracker&clickthroughURL=http%3A%2F%2Fwww.sonatype.com%2Fnexus%2Ffree-trial&nurtureTrackName=2014%20Nexus&nurtureTrackEmailNumber=3&emailName=Need%20a%20reason%20to%20%27go%20pro%27%3F&linkTitle=Download%20our%2014%20day%20trial&emailAddress=gsjung@ktnet.com&elq=be17da4c2e74461098d1cee6b6e5dd1a&elqCampaignId=67) now.

프로 버전은 상용이므로 비용이 발생하며 라이선스는 시트(Seat) 라는 단위로 매겨진다.

하나의 시트는 넥서스에 연결하는 클라이언트의 IP 별로 책정되며 개별 시트로 판매하지 않고 10, 20, 50 등 시트를 묶음 단위로 판매하므로 개발자가 많다면 많은 비용이 발생할 수 있는 문제가 있다.

그러므로 작은 시트의 상용 버전을 구매하고 일반적인 빌드와 디플로이시는 오픈 소스 버전을 사용하고 릴리스 준비 단계에서부터 상용 버전을 사용하는 등 비용을 절감하기 위해 두 개를 같이 혼용하는 방법도 있다.

## 설치

이제 어떤 제품을 사용할 지 결정했다면 설치를 진행해 보자. 오픈소스와 상용 버전은 배포하는 URL 만 다르고 설치 과정은 동일하므로 이 책에서는 오픈소스 버전 설치에 대해서만 설명하겠다.

#### 사전 준비 사항

넥서스는 별도의 DB 를 사용하여 정보를 관리하지 않는다. 모든 설정과 데이타는 파일 시스템에 저장하며 효율적인 검색을 위해 오픈 소스 검색 엔진인 Lucene 을 사용하여 인덱싱한다.

구동에는 서블릿 컨테이너가 필요하나 내장하고 있으므로 사전에 설치해야 할 것은 자바 런타임밖에 없다. 버전 2.6 이상부터는 JRE 7u21(JRE 8은 지원하지 않는다) 이상 버전을 요구하므로 사전에 설치하도록 하자.

아마 전 장에서 톰캣을 설치한 독자라면 이미 JRE 도 설치되어 있을 것이다.

<https://support.sonatype.com/entries/24901442-Sonatype-Nexus-System-Requirements>

넥서스는 패키지에 Jetty 라는 경량 WAS 를 내장하고 있으므로 별도의 WAS 가 필요없다. 기본 사용 포트는 8081 이므로 일반 사용자로 구동하는게 바람직하다.

구동할 사용자와 그룹을 생성해 보자.

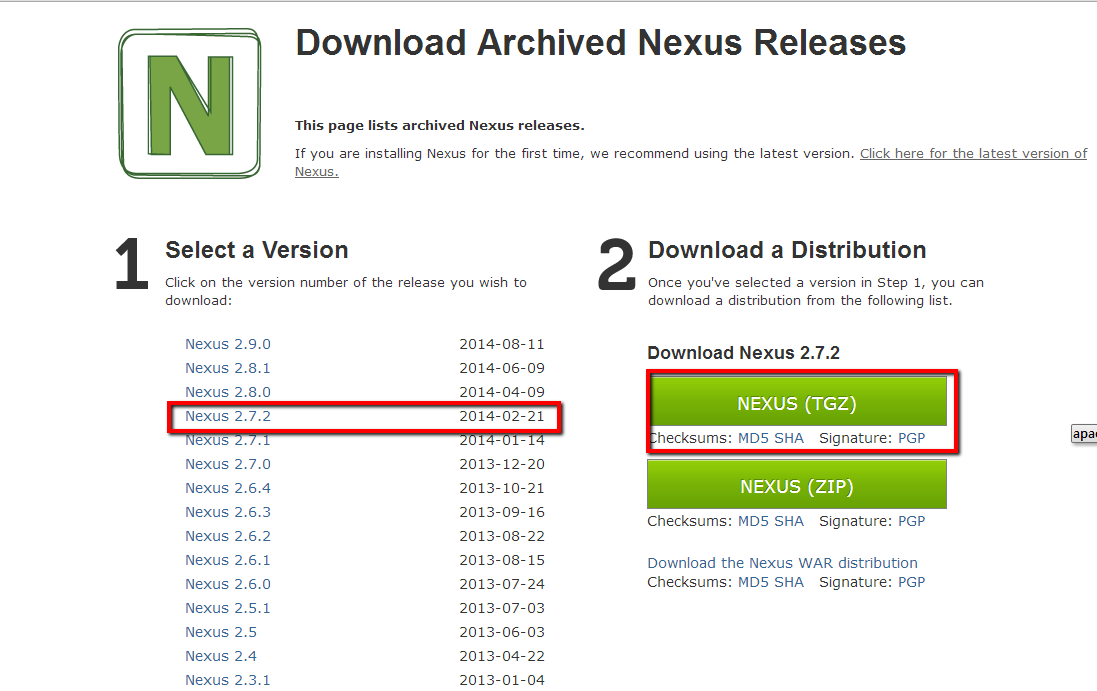
groupadd nexus  
adduser nexus -g nexus

이제 su - nexus 명령어로 계정을 전환한 후에 작업을 진행하면 된다.

#### 다운로드

이제 개발사 웹사이트에서 넥서스를 다운로드 해보자. 메인 페이지는 최신 버전만 링크되어 있으므로 예전 버전을 사용하려면 보관 페이지에 들어가야 한다.

웹브라우저를 열고 <http://www.sonatype.org/nexus/archived> 에 연결한 후에 2.7.2 버전을 클릭하고 .tgz 패키지를 다운받자.



**넥서스 버전 선택**

혹은 넥서스를 설치하려는 리눅스에 로그인한후에 curl 로 다운받을 수도 있다. 다음 명령을 실행하자.

curl -L -O *http://www.sonatype.org/downloads/nexus-2.7.2-bundle.tar.gz*

curl 의 -L 옵션은 컨텐츠가 이동되었을 경우 따라가며 -O 옵션은 저장시 원격지의 파일 이름대로 저장하는 옵션이다.

#### 설치

이제 다운받은 넥서스를 설치해 보자.

1. tar 로 압축을 해제한다.

tar zxvf *nexus-2.7.2-bundle.tar.gz*

1. 압축이 해제된 폴더로 이동한다.

cd nexus-2.7.2-03

1. ls 를 해보면 여러 개의 서브 디렉터리가 있는게 보일 것이다. 중요한 디렉터리들은 다음과 같다.
   1. **bin** : 넥서스 실행 스크립트(nexus)가 위치한다.
   2. **conf** : 넥서스 설정 파일(nexus.properties)과 jetty 설정 파일이 위치한다.
   3. **logs** : 넥서스 실행 로그가 저장된다. 로그 파일명은 wrapper.log  이다.
   4. **nexus**: 실제 넥서스 어플리케이션이다.

#### 설정 및 구동

넥서스를 구동하기 전에 먼저 구동 환경을 설정해 보자. 설정할 것은 많지 않지만 중요한 설정이 있다. 설정 파일은 conf/nexus.properties 이므로 이 파일을 선호하는 에디터로 열어 보자.

# 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

설정에서 가장 중요한 부분은 진하게 표시한 세 부분이다. 이제 하나씩 내용을 확인해 보자.

* **application-port=8081** - 넥서스가 사용할 포트이다. 기본 포트는 8081 이다.
* **nexus-webapp-context-path=/nexus**- 넥서스의 웹 어플리케이션 컨텍스트 경로이다. 기본 값은 nexus 이며 다음과 같이 연결해야 넥서스를 사용할 수 있다.   
  http://servername/nexus   
  필자는 nexus.example.com 등의 도메인을 부여하고 바로 <http://nexus.example.com> 으로 연결하는 것을 선호한다. 이럴 경우 컨텍스트 경로를 다음과 같이 nexus 부분을 제거해야 한다.  
  **nexus-webapp-context-path=/**
* **nexus-work=${bundleBasedir}/../sonatype-work/nexus -** 넥서스의 실제 저장소 데이타와 메타 데이타가 저장되는 경로이다.   
  넥서스 프로그램과 데이타를 분리한 이유는 이렇게 하면 프로그램의 갱신과 변경이 쉽고 데이타 백업과 복구가 용이하기 때문이다.  
  기본 설정은 넥서스 설치의 바로 윗 경로에 sonatype-work 이라는 디렉터리를 만들고 여기에 메타 데이타를 저장하는 것이다. 다른 경로에 넥서스 데이타를 저장하고 싶은 경우 이 설정을 변경해 주면 된다.   
  이 폴더는 중요한 폴더이므로 임의로 수정/삭제하면 안 된다.  
  필자는 넥서스의 데이타는 /var/sonatype-work  디렉터리 밑에 저장하고 있다. 이럴 경우 설정을 다음과 같이 변경해야 한다.   
  ***nexus-work=/var/sonatype-work/nexus***

이제 기본 설정은 끝났으니 구동해 보자. 구동은 bin/nexus 스크립트로 가능하며 구동시에는 start 옵션을, 정지시에는 stop 옵션, 재구동시에는 restart 옵션을 주면 된다.

./bin/nexus start

Starting Nexus OSS...

정상 구동 여부는 로그 파일을 통해서도 알수 있다. tail 명령어로 로그 파일을 확인해 보자.

tail -f logs/wrapper.log

8081 포트에 대해 iptables 을 열지 않았으므로 넥서스가 설치된 서버에서만 연결이 가능하다. 정상 구동 여부를 curl 을 사용해서 확인해 보자.

 curl  *http://localhost:8081/index.html*

HTML 이 출력된다면 정상 설정된 것이다.

**nexus-webapp-context-path=/nexus** 를 수정하지 않았다면 index.html 을 nexus/index.html 로 수정해야 정상 동작한다.

#### 아파치 웹서버와 연동

넥서스를 http://nexus.example.com:8081/ 와 같이 사용할 경우 8081 포트를 오픈해야 하고 또 개발자들이 포트를 기억해야 하지만 보통 도메인 이름은 기억해도 넥서스가 8081 포트에 떠 있다는 사실을 기억하기 쉽지 않다.

이런 문제 해결을 위해 아파치 웹서버를 사용하여 http 또는 https 를 통해 넥서스를 서비스 해보자.

전 장의 톰캣 부분에서 설명한 아파치와 연계 부분을 되돌려 보자. mod\_jk 는 톰캣 전용이므로 사용할 수 없으므로 아파치 웹서버와 넥서스를 연동하려면 mod\_proxy 가 유일한 대안이다.

아파치 웹서버의 가상 호스트 설정을 통해 넥서스와 연동해 보자. 다음 가상 호스트 설정을 아파치 설정 파일에 추가하자.

아파치 웹서버 장에서 설명했듯이 기본 경로는 /etc/httpd/conf/httpd.conf 이며 필자는 가상 호스트 파일은 /etc/httpd/conf/httpd-vhost.conf 로 분리해서 관리하고 있다.

<VirtualHost \*:80>

ServerName nexus.example.com  
ProxyRequests Off  
ProxyPreserveHost On  
<Proxy \*>

Order deny,allow  
Allow from all

</Proxy>  
**ProxyPass / http://localhost:8081/  
ProxyPassReverse / http://localhost:8081/**

</VirtualHost>

ProxyPass 와 ProxyPassReverse 항목을 유심히 살펴보자. nexus-webapp-context-path=/nexus 부분을 수정하지 않은 독자라면 이 부분을 다음과 같이 수정해야 한다.

ProxyPass /nexus http://localhost:8081/nexusProxyPassReverse /nexus  http://localhost:8081/nexus

이제 사용자가 *http://nexus.example.com* (수정하지 않은 독자는 http://nexus.example.com/nexus)  으로 연결하면 아파치의 프록시 모듈은 8081 포트에 뜬 넥서스에 연결하여 사용자에게 전달하게 된다.

아파치가 8081 포트에 연결해야 한다는 얘기에 감을 잡은 독자들도 많을 것이다.

짐작대로 8081 포트는 아파치에게 허용된 포트가 아니므로 SELinux 설정을 변경해 주어야 한다.

SELinux 장에서 배운 setsebool 명령어로 httpd\_can\_network\_connect 를 1로 설정하는 방법도 있지만 이럴 경우 아파치 웹서버가 모든 네트워크 포트에 접근 가능하므로 좋은 방법은 아니다.

명확하게 넥서스에만 연결 가능하도록 http\_port\_t 컨텍스트에 8081 포트를 추가해 보자.

semanage port -m -p tcp -t http\_port\_t  8081

이제 service httpd restart 로 아파치 웹서버를 재시작하고 웹브라우저로 연결하면 다음과 같은 화면이 보인다면 정상적으로 설치된 것이다.



**넥서스 초기 화면**

#### SSL 연동

여러 이유로 넥서스에 SSL 로 연결하고 싶을 수가 있다. 아파치 웹서버는 SSL 에서도 가상 호스트가 가능하므로 위 설정을 다음과 같이 변경하면 된다. 이왕이면 mod\_rewrite 를 사용하여 HTTP 로 들어온 클라이언트는 모두 HTTPS 로 전환하도록 하자.

다음은 conf/httpd-vhost.conf 파일이다. HTTPS 가 아닐 경우 https 로 전환하게 rewrite 규칙을 적용했다.

<VirtualHost \*:80>

ServerName nexus.example.com  
RewriteEngine on  
RewriteCond %{HTTPS} !on  
RewriteRule ^(.\*)$ https://%{HTTP\_HOST}$1 [R,L]

</VirtualHost>

이제 conf.d/ssl.conf 파일을 편집해 보자.

<VirtualHost \*:80>

**ServerName nexus.example.com**  
**ErrorLog logs/nexus-ssl\_error\_log**  
**TransferLog logs/nexus-ssl\_access\_log**  
LogLevel warn

SSLEngine on  
SSLProtocol all -SSLv2  
SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW

**SSLCertificateFile /etc/pki/tls/certs/localhost.crt**  
**SSLCertificateKeyFile /etc/pki/tls/private/localhost.key**

<Files ~ "\.(cgi|shtml|phtml|php3?)$">

SSLOptions +StdEnvVars

</Files>

<Directory "/var/www/cgi-bin">

SSLOptions +StdEnvVars

</Directory>

SetEnvIf User-Agent ".\*MSIE.\*" nokeepalive ssl-unclean-shutdown downgrade-1.0 force-response-1.0

**CustomLog logs/nexus-ssl\_request\_log** "%t %h %{SSL\_PROTOCOL}x %{SSL\_CIPHER}x \"%r\" %b"

ProxyRequests Off  
ProxyPreserveHost On  
<Proxy \*>  
Order deny,allow  
Allow from all

**ProxyPass / http://localhost:8081/**  
**ProxyPassReverse / http://localhost:8081/  
RequestHeader set X-Forwarded-Proto "https"**

[</Proxy>](http://localhost:8081/)

</VirtualHost>

독자의 환경에 따라 수정해야 할 부분은 진하게 표시했으니 각자 환경에 맞게 수정하기를 바란다.

마지막 부분의 RequestHeader  부분을 살펴 보자. nexus 는 기본 설정이 http 이며 아파치 웹서버는 https 일 경우 꼭 설정해야 하는 부분이다.

이 설정이 빠지면 넥서스의 모든 링크가 http 이므로 제대로 동작하지 않으므로 https 로 넥서스에 연결하려면 설정하도록 하자.

아파치를 재구동하여 http://nexus.example.com 에 연결하면 자동으로 https://nexus.example.com 으로 전환되는 것을 확인할 수 있을 것이다.

#### 참고 문서

JRE 7 - <http://books.sonatype.com/nexus-book/reference/_nexus_prerequisites.html>

* <http://books.sonatype.com/nexus-book/reference/install-sect-downloading.html>
* <http://books.sonatype.com/nexus-book/reference/_installing_nexus.html>

#### Upgrading Nexus

* <http://books.sonatype.com/nexus-book/reference/install-sect-upgrading.html>

#### Post-Install Checklist

* <http://books.sonatype.com/nexus-book/reference/install-sect-repoman-post-install.html>

reverse proxy 와 사용한다면 다음 헤더 필요

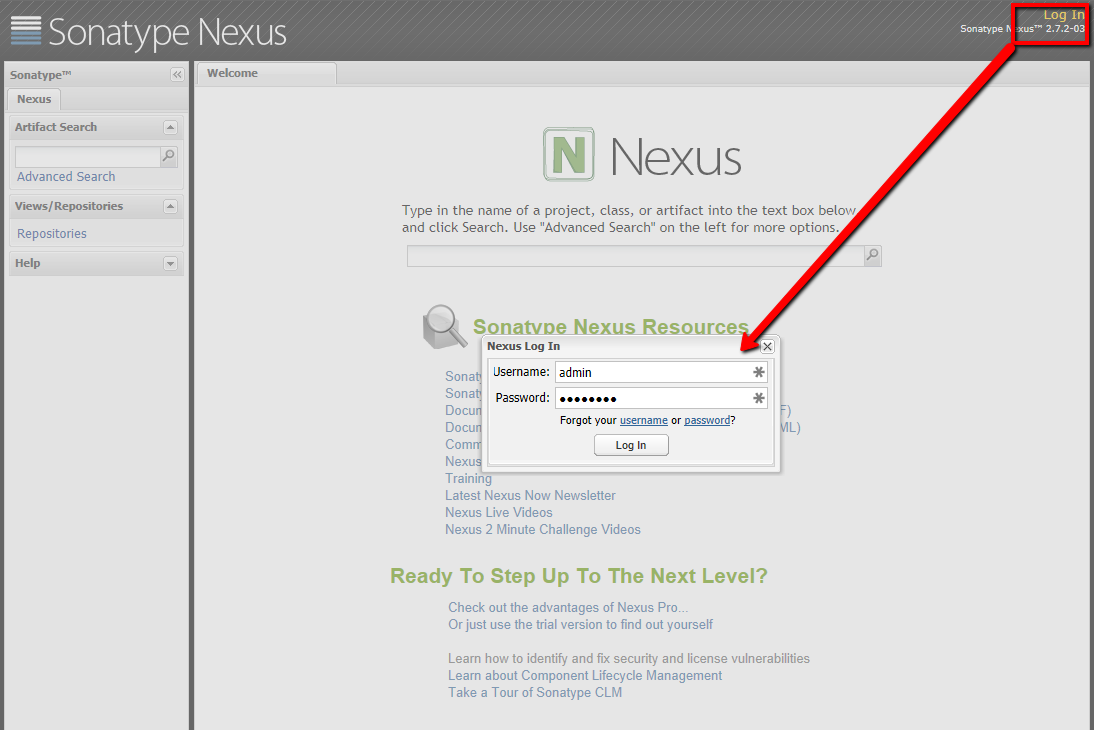
|  |
| --- |
| RequestHeader set X-Forwarded-Proto "https" |

#### 설치후 할 일

넥서스가 정상적으로 설치되었다면 꼭 해줘야할 기본적인 설정이 있다.

**관리자 암호 변경**

가장 먼저 해야 할 일은 관리자 암호 변경이다. 기본 관리자 계정은 admin 이고 암호는 admin123 이다. 우측 상단의 Log In 을 클릭하고 관리자로 로그인해 보자.

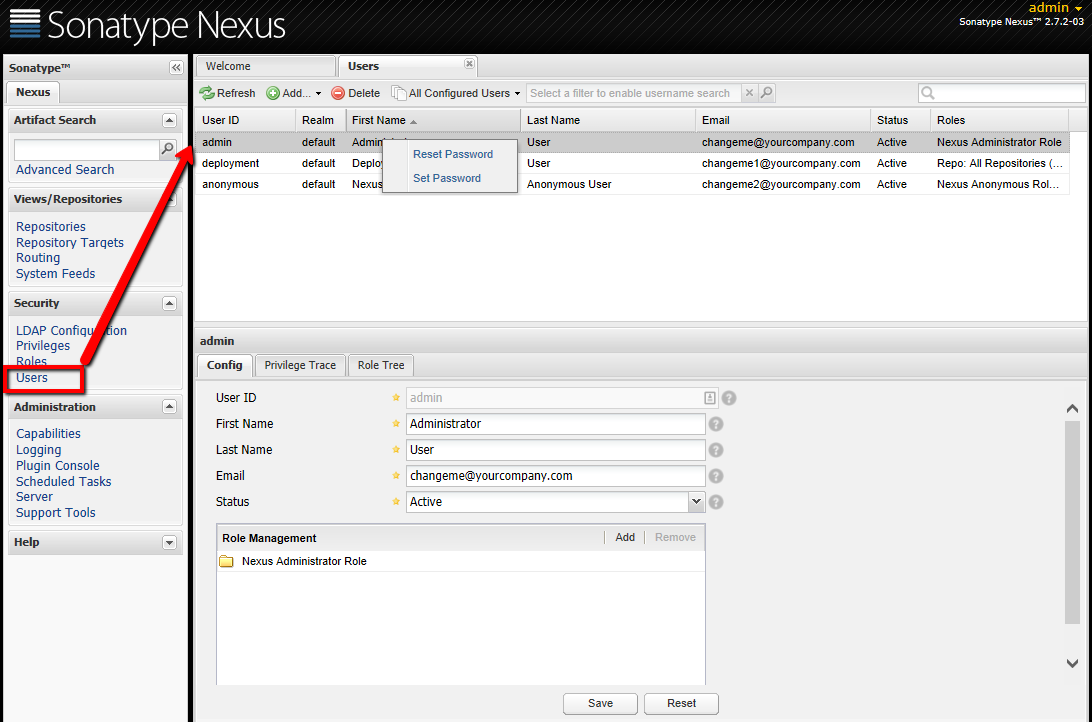


**관리자 로그인**

로그인 했다면 좌측의 메뉴에서 **Security** -> **Users** 를 선택하면 상단에 전체 사용자 목록이 표시된다.

여기에서 admin 을 선택하고 마우스 우측 버튼을 눌러서 팝업 메뉴를 호출하고 메뉴에서 **Set Password** 를 선택하여 암호를  변경하면 된다. **Reset Password** 는 암호를 재설정하고 등록한 이메일로 보내주는 기능이므로 여기에서는 적절하지 않다.

그리고 하단에 표시되는 사용자 정보 화면에서 이름과 이메일 주소도 수정해 주자.



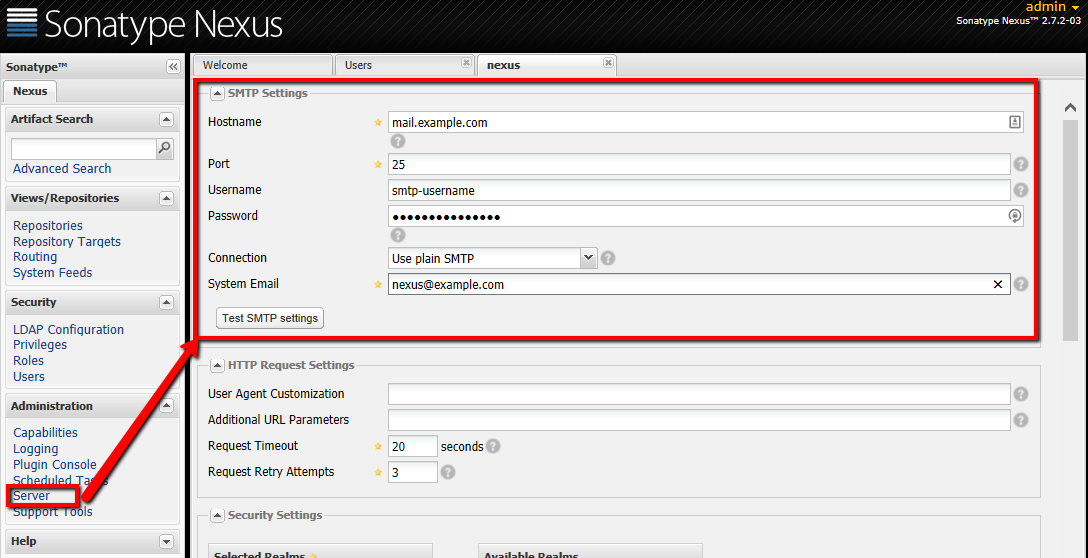
**암호 변경**

**SMTP 서버 설정**

넥서스는 사용자 암호 리셋이 이메일로 전달하므로 이 기능을 사용하려면 이메일 서버를 설정해야 한다. 좌측의 **Administration**-> **Server** 를 선택한후에 SMTP 정보를 설정하자.

localhost 에 SMTP 가 있다면 로그인할 필요가 없으므로 Username/Password 항목은 설정하지 않아도 된다.

설정이 끝났다면 **Test SMTP settings** 를 클릭하여 정상 동작 여부를 확인해 보자.

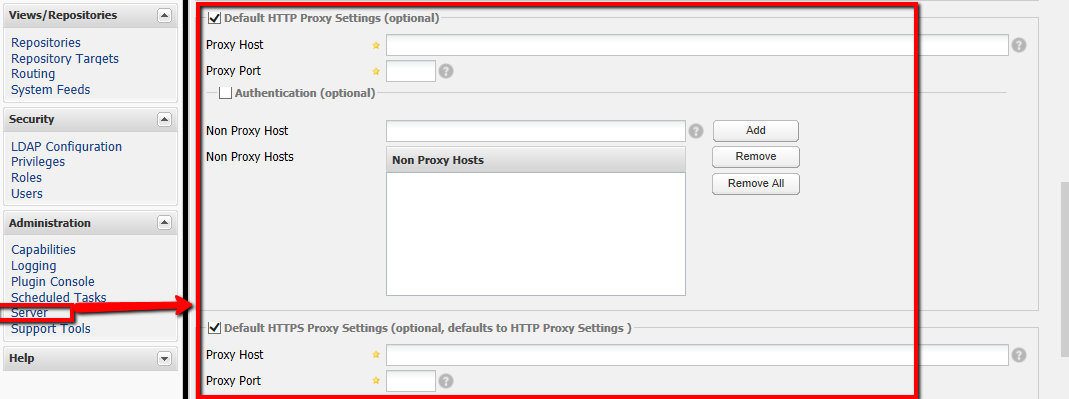


**SMTP 서버 설정**

**HTTP/HTTPS Proxy 설정**

넥서스는 주기적으로 외부에 있는 저장소에 연결하여 변경된 아티팩트를 가져와야 한다. 넥서스가 설치된 서버가 보안 문제때문에 직접 외부에 연결하지 못하고 포워드 프록시를 사용한다면 프록시 설정을 해주어야 외부에서 아티팩트를 가져올 수 있다.

**Server** 메뉴에서 중간 부분에 설정이 있으므로 이런 환경에서 넥서스를 사용한다면 설정해 주자.



**포워드 프록시 설정**

**deploy 계정 암호 변경**

넥서스 설치시 admin 과 함께 deployment 라는 계정이 기본 생성된다. 이 계정은 넥서스에 아티팩트를 올리는데 사용하는 계정으로 기본 암호는 deployment123 이다. 아무나 아티팩트를 올리거나 삭제할 수 있으므로 기본 암호를 변경하자.

## 저장소 관리

저장소 관리는 넥서스의 가장 중요한 기능중 하나이다. 먼저 넥서스가 제공하는 저장소의 종류에 대해 알아보도록 하자.

### 저장소의 종류

#### 프록시 저장소(Proxy Repository)

프록시 저장소는 메이븐 중앙 저장소등 원격지에 있는 저장소를 미러링한다. 넥서스는 기본적으로 3개의 원격지 저장소가 등록되어 있다.

**Apache Snapshots**

아파치 재단의 스냅샷 아티팩트를 미러링한다.

**Codehaus Snapshots**

Codehaus 프로젝트의 스냅샷 아티팩트를 미러링한다.

**Central**

중앙 저장소는 릴리스 아티팩트를 포함하는 저장소로 메이븐 중앙 저장소라고도 한다. 메이븐에 기본 포함되어 있으므로 저장소를 특별하게 지정하지 않으면 이 저장소에 연결하게 된다.

주소는 <http://repo1.maven.org/maven2/> 이다.

#### 호스트 저장소

호스트 저장소는 기업용 사설 저장소이다. 넥서스는 기본적으로 3개의 호스트 저장소가 설정되어 있다.

**3rd Party**

메이븐 중앙 저장소에 없는 아티팩트를 올리기 위한 저장소이다. 기업에서 구매한 상용 라이브러리나 특정 벤더의 JDBC 드라이버등을 이 저장소에 올리고 사용하면 된다.

**Releases**

회사 내부에서 만든 아티팩트를 보관하는 저장소로 릴리스 저장소를 통해 다른 개발자나 프로젝트 팀과 공유 및 협업할 수 있다.

**Snapshots**

회사 내부에서 만든 아티팩트중 릴리스 되기전 개발 단계의 아티팩트를 보관한다.

#### 가상 저장소(Virtual Repository)

가상 저장소는 다른 유형의 저장소의 아답터로 동작한다. 넥서스는 기본적으로 메이븐 1 형식의 저장소를 메이븐 2 형식의 저장소로 변환하는 기능을 제공하고 있다.

#### 그룹 저장소(group Repository)

그룹 저장소는 넥서스가 제공하는 기능으로서 저장소의 종류는 아니다. 다만 위에서 설명한 여러 종류의 저장소를 논리적으로 묶어서 하나의 저장소처럼 사용할 수 있는 기능을 제공한다.

기본 탑재된 호스트 저장소(3개)와 프록시 저장소(3개)를 프로젝트에서 모두 사용하는 경우를 생각해 보자. 저장소마다 고유의 URL 이 있으므로 메이븐등의 빌드 툴에서 총 6개의 저장소 URL 을 설정해야 한다.

하지만 그룹 저장소 기능을 사용하여 6개의 저장소를 하나의 저장소로 묶으면 빌드 툴에는 그룹 저장소의 URL만 설정하면 되며 향후 저장소가 추가/변경되어도 메이븐 설정은 변경하지 않아도 되니 매우 편리하게 사용할 수 있다.

넥서스에는 **Public Repositories** 라는 이름의 그룹 저장소가 기본 설정되어 있으며 이 저장소에는 프록시 저장소 하나(메이븐 중앙 저장소), 호스트 저장소 3개 총 4개의 저장소가 포함되어 있다.

[Managing Repositories](http://books.sonatype.com/nexus-book/reference/confignx-sect-manage-repo.html#fig-repo-config)

maven 설정

* <http://books.sonatype.com/nexus-book/reference/maven-sect-single-group.html>

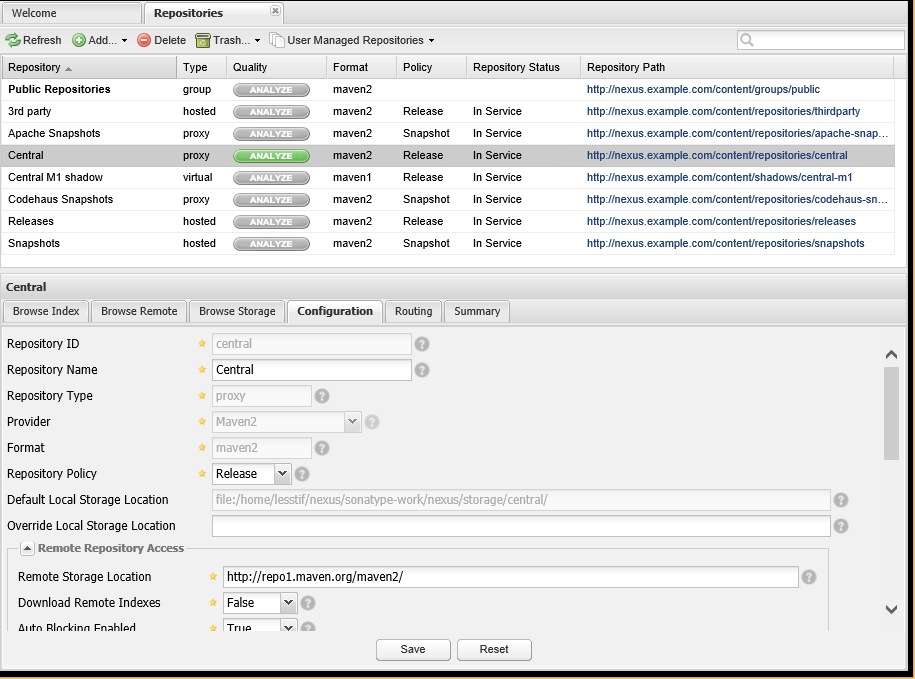
#### REST API

curl -X GET -u [admin:alsdk1](http://adminalsdk1/) <http://nexus.ktnet.com/service/local/users>

* <https://nexus.ktnet.com/nexus-restlet1x-plugin/default/docs/index.html>

### 저장소 설정

이제 관리자로 로그인하여 저장소를 설정해 보자. 저장소 목록은 좌측의 **Views/Repositories** 메뉴에서 **Repositories**를 선택하면 된다. 관리자일 경우 상단의 저장소 목록을 클릭하면 하단에 여러 개의 탭이 표시된다.



**저장소 목록**

**Repository**

저장소의 이름이 표시되는 컬럼이다. 그룹 저장소는 볼드체로 표시된다.

**Type**

 저장소의 종류로 위에서 proxy, 호스트, virtual, group 4가지 종류가 있다.

**Health Check**

 저장소 헬스 체크 기능을 수행하며 자세한 보고서를 얻으려면 상용 버전의 넥서스를 구매해야 한다.

[**Format**](http://books.sonatype.com/nexus-book/reference/rhc.html)

 저장소의 포맷을 의미하며 일반적으로 maven2 이다.

**Policy**

 deployment 정책을 표시하며 Snapshot 또는 Release 두 개중 하나의 정책을 설정할 수 있다.

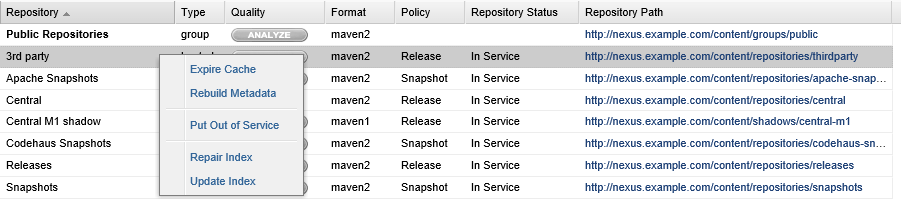
**Repository Status**

저장소의 서비스 상태를 표시한다. In Service 면 정상적으로 서비스되는 상태이다. 그룹 저장소는 논리적 저장소이므로 해당 컬럼을 표시하지 않는다.

**Repository Path**

HTTP로 저장소에 바로 연결할 수 있는 URL 을 의미한다. 이 URL 은 메이븐등의 빌드툴에 저장소 설정할 때 꼭 필요한 중요한 컬럼이다.

상단의 저장소 목록에서 저장소를 선택하고 마우스 우클릭을 하면 팝업 메뉴가 표시되며 여러 가지 관리 기능을 수행할 수 있다.



**Expire Cache**

 저장소의 캐쉬를 만료시킨다.

**Rebuild Metadata**

 호스트 저장소의 메타 데이타를 리빌드한다. 저장소의 스토리지내에서 직접 아티팩트를 변경했을 경우등에 사용하면 된다.

**Block Proxy / Allow Proxy**

 프록시 저장소의 프록시 기능을 허용/거부한다.

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

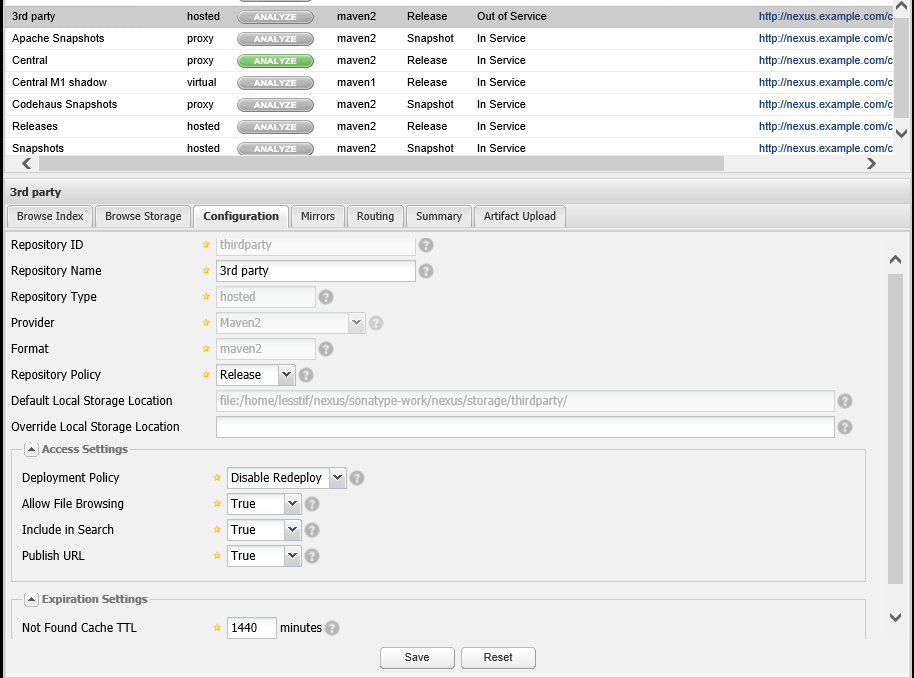
 저장소의 상태를 변경한다. 해당 저장소가 보관된 스토리지를 이동한다거나 할 경우등 운영 작업시 사용할 수 있다.

**Repair Index / Update Index**

 저장소의 인덱스를 복구하거나 갱신한다.

이제 하단의 설정 항목을 살펴보자. 다양한 설정이 있지만 넥서스 사용시 꼭 필요한 항목만 알아 보자. 설정은 저장소가 프록시인지 호스트 인지에 따라 약간 다르다.

먼저 기본 포함된 호스트 저장소인 **3rd party** 를 클릭해 보자



**hosted 저장소 설정**

설정이 불가능한 항목은 회색으로 표시되며 설정할 수 있는 항목은 하얀색으로 표시된다. 중요한 설정 항목은 다음과 같다.

**Repository Name**

장소 목록에 표시할 이름이다.

**Repository Policy**

Release 또는 Snapshot 저장소인지를 설정할 수 있다.

**Deployment Policy**

아티팩트 디플로이 정책을 설정한다. 기본은 ***Disable Redeploy*** 이며 이럴 경우 아티팩트의 버전을 디플로이 했으면 다시 할 수는 없다. 아티팩트에 버그가 있거나 잘못 되어 있으면 버전을 변경하여 다시 디플로이 해야 한다.

***Allow Redeploy*** 로 설정하면 다시 디플로이할 수 있지만 이럴 경우 많은 문제가 발생할 수 있다. 예로 아티팩트 ID가 my-lib 이고 버전이 1.1 인 아티팩트를 디플로이했다고 하자.

1.1 버전이 잘못된걸 알고 수정한 후에 동일 버전으로 재디플로이 하면 이미 1.1 버전을 받아간 개발자의 메이븐은 아티팩트가 변경되지 않았다고 판단하므로 동일한 문제가 발생할 수 있다.

컴포넌트 버전 관리에 맞지 않는 방법이므로 Redeploy 로 설정하는 것은 충분히 영향을 숙지한 후에 설정하는게 좋다.

**Allow File Browsing**

True 일 경우 웹 브라우저로 저장소를 브라우징 할 수 있다.

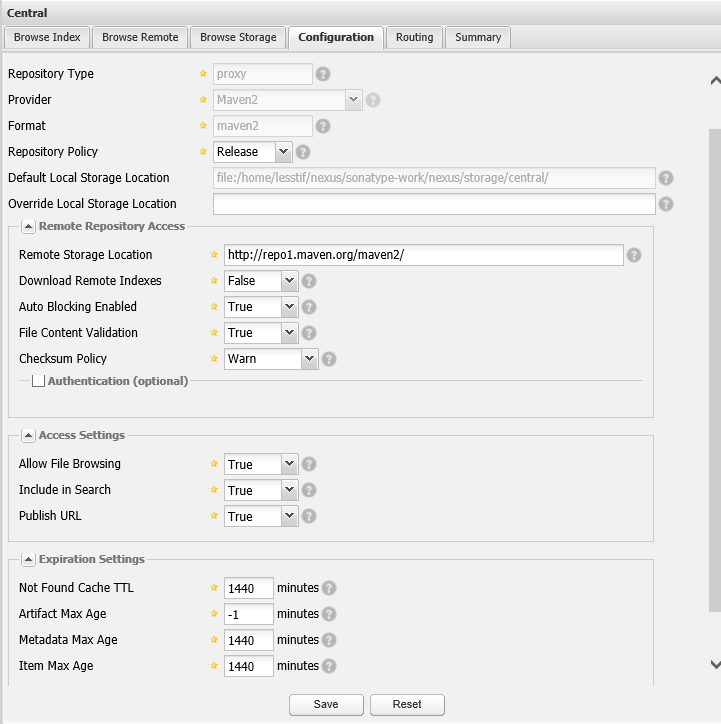
**Include in Search**

True 일 경우 아티팩트 검색시 이 저장소에 있는 아티팩트를 포함시킨다.

**Publish URL**

True 일 경우 이 저장소의 URL 을 퍼블리싱하므로 외부에서 URL 로 접근이 가능하다.

이제 기본 포함된 호스트 저장소인 **Central**을 클릭해 보자



**프록시 저장소 설정**

옵션은 많지만 설정을 변경 해야 하는건 별로 없다. **Remote Repository Access** 설정부터 각 항목의 의미를 알아보자.

**Remote Repository Location**

메이븐 중앙 저장소등 미러링할 원격지 서버의 URL이다. URL 이 변경되지 않은 이상 변경할 일이 없다.

**Download Remote Indexes**

기본은 False 이다. 넥서스를 설치한후에 True 로 변경해 주면 아티팩트들에 대한 인덱스 파일을 다운로드한다.

메이븐 중앙 저장소에는 몇 천개의 아티팩트가 있으므로 모두 다운로드하고 인덱싱하려면 시간이 꽤 오래 걸린다.

True 로 설정하면 인덱스 파일을 받아서 사용하므로 인덱싱이 필요없어서 검색 속도가 빨라지고 검색 결과도 좋아진다.

**Checksum Policy**

저장소에 있는 아티팩트들의 체크섬이 잘못되었을 경우 어떻게 처리할지 설정한다. 기본값은 Warning 이며 Ignore, StrictIfExists, Strict도 설정이 가능하다.

* Ignore - 체크섬이 잘못되도 무시한다.
* Warn - 로그 파일에 경고 메시지를 남긴다.
* StrictIfExists - 체크섬 파일이 있을 경우에만 계산한 체크섬과 저장소의 체크섬이 일치하지 않을 경우 캐싱하지 않는다.
* Strict - 계산한 체크섬과 저장소의 체크섬이 일치하지 않을 경우나 체크섬 파일이 없다면 캐싱하지 않는다.

다음은 캐시 관련 설정이다.

**Not Found Cache TTL**

넥서스가 원격 저장소에서 아티팩트를 못 찾았을 경우 설정한 시간이 지난후에 재시도한다. 기본 값은 24시간이다.

**Artifact Max Age**

원격 저장소에서 새 버전을 다운로드 받기전 아티팩트의 최대 시간을 설정한다. 릴리스 저장소일 경우 -1 이며 스냅샷은 24시간이다.

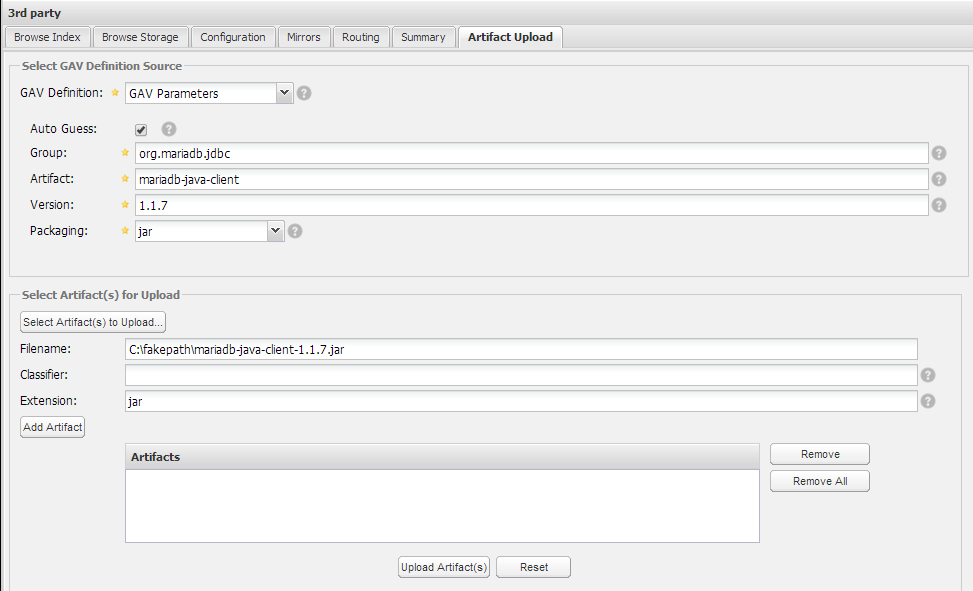
**Metadata Max Age**

넥서스는 원격 저장소에서 설정한 시간이 지나면 메타 데이타를 다운로드 받는다. 기본 설정은 24시간이다.

#### 저장소에 아티팩트 올리기

메이븐을 통해 저장소에 아티팩트를 디플로이할 수 있지만 넥서스 관리자에서도 할 수 있다. 업로드는 호스트 저장소만 가능하다. MariaDB JDBC 를 **3rd party** 저장소에 디플로이해보자.

1. 관리자로 로그인한후에 저장소 목록중 3rd party를 클릭하여 하단에 설정 화면을 띄운다.
2. GAV(Group, Artifact, Version) Definition 탭에서 GAV Parameters 를 선택하고 GAV와 Packaing 을 다음과 같이 설정한다.
   1. Group: org.mariadb.jdbc
   2. Artifact: mariadb-java-client
   3. Version: 1.1.7
   4. Packaging: jar 선택
3. Select Artifact(s) to Upload... 를 클릭하고 다운로드받은 mariadb jar 파일을 선택한다.
4. Add Artifact 버튼을 클릭하여 선택한 아티팩트를 추가한다.
5. 하단의 Upload Artifact(s) 를 클릭하면 아티팩트가 업로드된다.

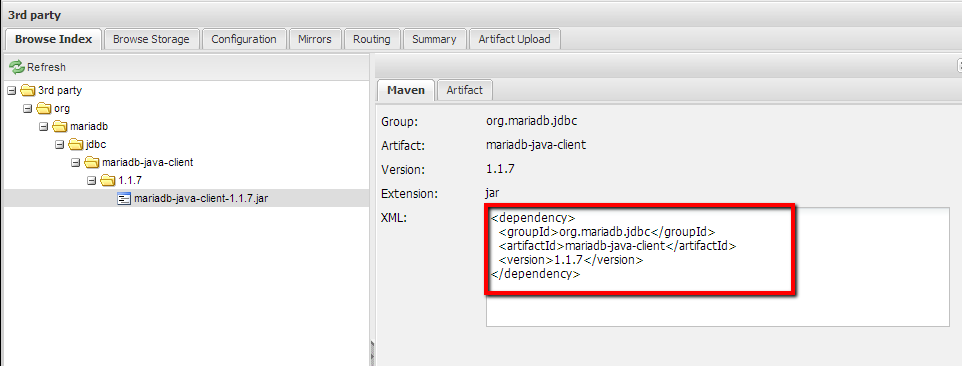


**아티팩트 업로드**

추가한 아티팩트는 **Browse Index** 탭에서 확인해 볼수 있다.

만약 방금 추가한 아티팩트가 보이지 않으면 ***Refresh*** 를 누르면 된다.

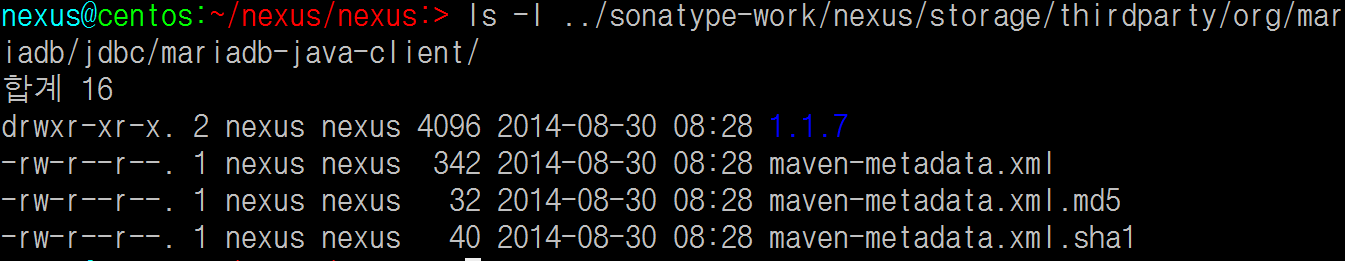
또 아티팩트를 클릭하면 우측에 ***Maven*** 탭에서 메이븐에 연동하기 위한 XML 정보를 볼수 있으며 ***Artifact*** 탭을 누르면 다운로드도 받을수 있고 삭제도 가능하다.



**아티팩트 브라우징**

추가된 아티팩트는 파일 시스템에 저장되므로 ls 명령어를 통해서도 확인해 볼 수 있다. ls 명령어로 저장소의 파일 시스템을 보면 아티팩트 추가를 확인할 수 있다.

ls -l ../sonatype-work/nexus/storage/thirdparty/org/



**파일 시스템 확인**

**아티팩트 추가 시점**

ls -l 로 넥서스 스토리지의 저장소 내 파일 목록을 확인해 본 독자들은 의아한 부분을 발견했을 것이다.  바로 프록시 저장소의 경우 어떤 아티팩트가 없기 때문이다.

넥서스는 중앙 저장소와 주기적으로 아티팩트를 동기화 하지 않는다.

단지 클라이언트가 아티팩트를 요청했을 때 넥서스에 없을 경우 중앙 저장소에 요청 하고 받아온 아티팩트를 스토리지내 저장소에 저장한 후에 클라이언트에게 서비스 한다.

그러므로 초기에는 용량이 작지만 클라이언트들이 다양한 아티팩트를 요청할 수록 스토리지가 사용하는 용량이 점점 커지게 된다.

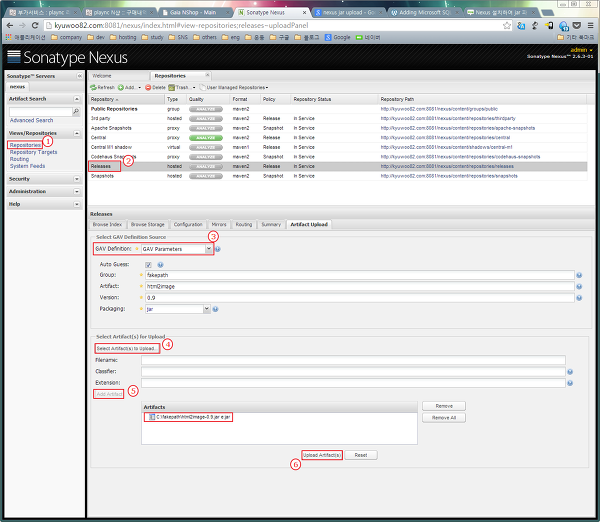
이 때문에 넥서스를 백업할 경우 파일 시스템내 프록시 저장소는 제외해도 문제가 없다.

## [Nexus에 jar파일 올리기](http://lng1982.tistory.com/164)

[tools/others](http://lng1982.tistory.com/category/tools/others) 2013.12.03 12:58

maven 중앙 repository에 jar 파일이 없는 경우 사설 repository에서 다운로드를 제공하려면 해당 jar 파일을 업로드 해야 하는데 업로드 방법을 매번 잊어 먹어 이곳에 정리한다.

순서는 다음과 같이 진행한다.



4번 진행 시 Group 필드에 "fakepath"로 셋팅이 되니 해당 jar 파일의 패키지 경로를 알맞게 넣어주도록 하자. (브라우저에서 보안상의 이유로 중간 path를 fakepath로 바꿈)