http://umsh86.tistory.com/entry/jenkins-%EC%84%A4%EC%B9%98-%EB%B0%8F-%ED%99%98%EA%B2%BD%EC%84%A4%EC%A0%95

\* 목표 (이번 글에서는 1.2번만 정리하겠다)

 1. jenkins.war 파일 다운로드 및 설치   
 2. 계정 관련 설정  
 3. 플러그인 설치(git, gradle 연동)  
 4. Publish Over FTP 플러그인을 사용한 서버에 자동 배포

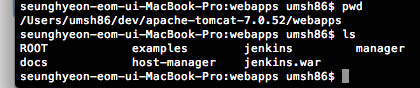
jenkins에 관심이 많았으나.. 사용해 볼 기회가 없어서 이제야 설치해보고 정리한다.

(사실 예전에 설치"만" 해봤었던 적은 있다..)

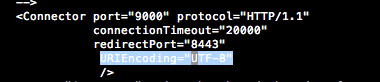
1. jenkins.war 파일 다운로드 및 설치

우선 설치는 mac osx에 tomcat을 설치하고, war파일을 배포하는 방식으로 했다.

<http://jenkins-ci.org/> 에서 war 파일을 받아서 tomcat의 webapps 에 넣는다.



보통 8080 port는 개발시 많이 사용하므로 /apache-tomcat-7.0.52/conf/server.xml 에서 port정보들을 변경해준다. 그리고 URIEncoding 설정을 해준다.(안해주면 나중에 jenkins에서 경고를 띄워준다.)



tomcat 시작한다 : /apache-tomcat-7.0.52/bin/startup.sh

2. 계정 관련 설정  
서버가 시작되면 http://localhost:9000/jenkins로 접속한다.(servert.xml에서 port 9000으로 변경했다.)

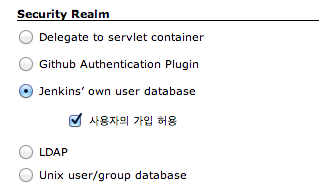
왼쪽 메뉴에서 Jenkins 관리 -> Configure Global Security선택



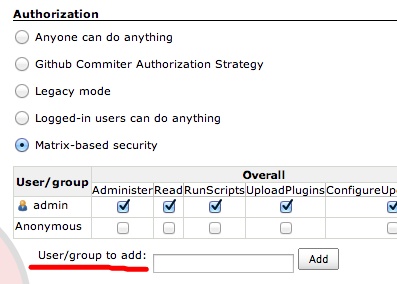
- enable security에 체크



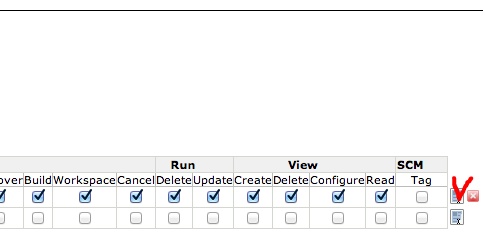
- 사용자의 가입을 허용



- Matrix-based security를 선택 한후, User/group to add 에 admin을 입력하고 Add 버튼 클릭.



- 그러면 matrix에 admin이 추가되는데, 맨 오른쪽에 버튼을 클릭 하면 모든 권한이 체크 된다.



- 적용 후 세이브하고 나와서, 오른쪽 위에 있는 가입버튼을 눌러서 admin 계정으로 가입한다.

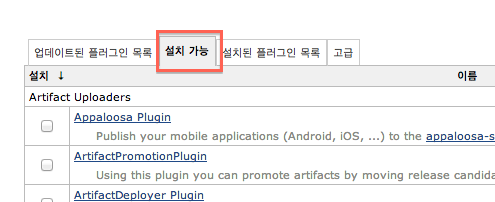
<http://umsh86.tistory.com/entry/jenkins-%EC%84%A4%EC%B9%98-%EB%B0%8F-%ED%99%98%EA%B2%BD%EC%84%A4%EC%A0%95-2>

 목표 ([지난 글](http://umsh86.tistory.com/entry/jenkins-%EC%84%A4%EC%B9%98-%EB%B0%8F-%ED%99%98%EA%B2%BD%EC%84%A4%EC%A0%95)에 이어서 이번 글에서는 3.4번에 대해 정리해보자.)

 1. jenkins.war 파일 다운로드 및 설치   
 2. 계정 관련 설정  
 3. 플러그인 설치(git, gradle 연동)  
 4. Publish Over FTP 플러그인을 사용한 서버에 자동 배포

1. Jenkins관리 -> 플러그인 관리 -> 설치가능 -> 검색하는 부분에 GIt Plugin, Github Plugin, Github OauthPlugin, Gralde Plugin, Publish over FTP 이렇게 5개를 검색해서 설치 후 Jenkins를 재시작 해준다.

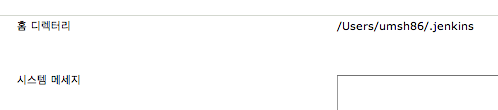




2. 다시 Jenkins관리 -> 시스템 설정으로 들어간다.



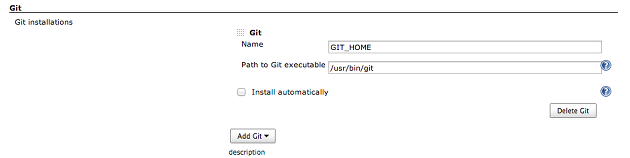
오? 홈디렉토리가 ~/.jenkins 이다. 실제로 테스트해보니, 배포했던 Tomcat/webapps에 폴더를 삭제하고, 재설치를 해도 기존의 내용들이 남아있다. ~/.jenkins의 폴더를 삭제하고, jenkins를 재설치하니(war를 통한 배포) 완전히 초기화되었다.



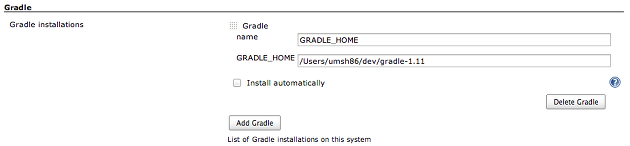
JDK 설정을 해준다.



git 설정도 해주고.. git이 어디에 설치했는지 기억이 안나서 whereis git 명령어로 찾았다(참고로 맥). git을 계속 써야하는데.. 쩝...



gradle 설정...



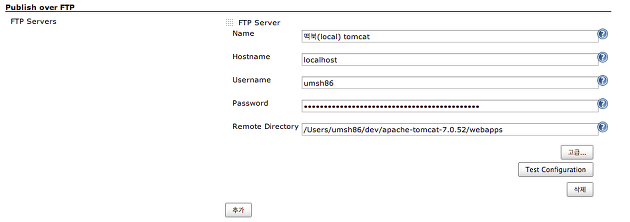
jenkins로케이션 설정...



git 관련 추가 설정.. 이것도 기억이 안나서 git config --list 명령어로 찾았다..



요 부분이 실제로 빌드가 종료되면 FTP를 통해 war가 배포되는 부분이다. 여기까지 설정 후 save~

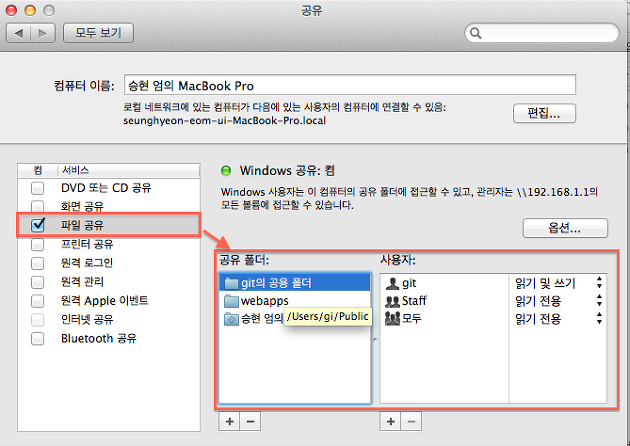


이 부분을 테스트 하기 위해서, FTP서버가 필요했는데.. 따로 테스트할 수 있는 서버가 없어서 우선 local에다가 자동배포 되도록 테스트했다.(가격이 저렴한 가상서버호스팅이라도 하나 해야겠다.. 예전엔 하나 해서 혼자 이것저것했었는데 흠..)

또, 까먹을수 있으니 mac ftp 공유를 설정하는 방법도 같이 정리해놔야겠다.

우선 시스템환경설정->공유에서 관련 설정들을 해주고..





- 터미널에서 다음 명령어를 통해 FTP데몬을 작동시킬 수 있다.

FTP 시작 : sudo launchctl load -w /System/Library/LaunchDaemons/ftp.plist

FTP 중지 : sudo launchctl unload /System/Library/LaunchDaemons/ftp.plist

- 시스템 시작시, 자동으로 서버를 시작할지, 아니면 중지할지 설정하는 방법

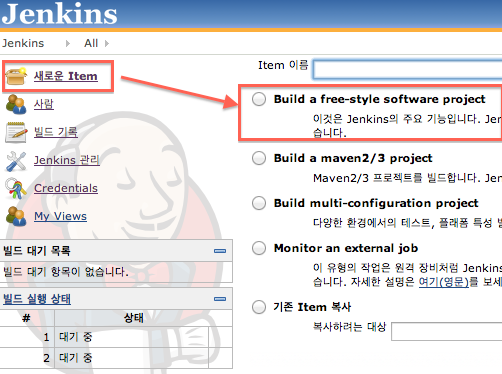
시작 : sudo launchctl start com.apple.ftpd

중지 : sudo launchctl stop com.apple.ftpd

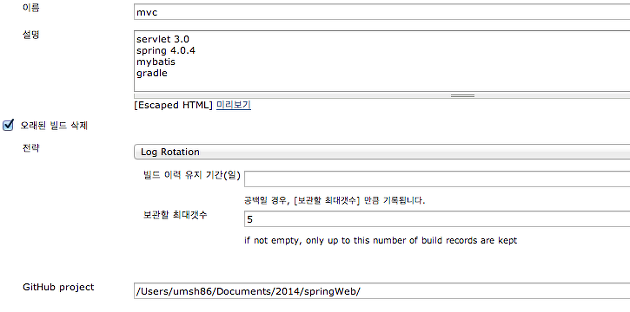
이렇게 하면 환경설정은 끝났다.

그럼 실제로 item(예전에는 job이라고 했던거 같은데 아닌가..?)을 등록하고, 빌드가 정상적으로 되어 배포가 되는지 확인해보자.

새로운 Item -> Item 이름을 적고, gradle project이므로  Build a free-style software project를 선택해준다.

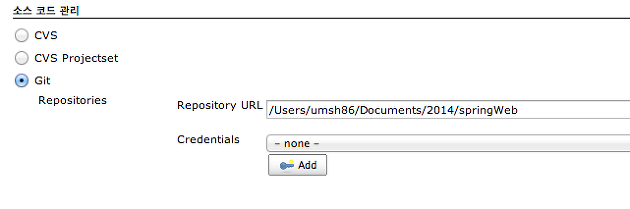


간단한 설명을 쓰고.. 오래된 빌드 삭제에 체크, 보관할 최대갯수를 설정해주고.. github도 우선은 로컬으로 설정을..

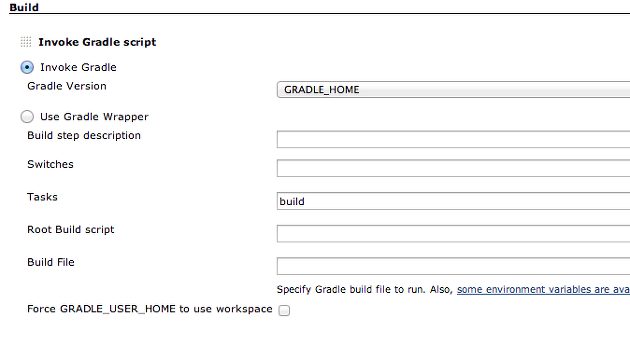


소스코드 관리는 git으로 ..(위의 GItHub project와 아래의 Git Repository URL을 같게 해도 되는지 잘모르겠다..

아직 git, gradle에 대해서 잘 모르는 상태라..)



build는 gradle로 할 거고.. (task도 저렇게 해주는게 맞는건가..-\_-;)



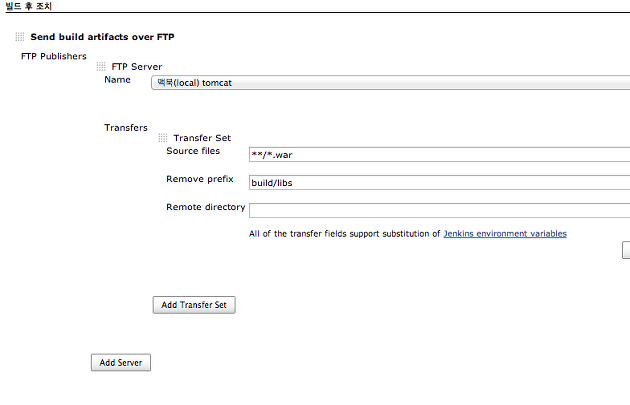
빌드 후에 조치(서버로 배포)와 관련된 설정이다.

FTP서버는 아까 등록했던 맥북 local로 해주고 Sources files는 \*\*/\*.war로 해준다.

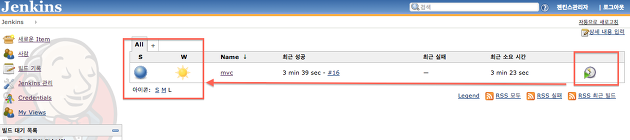
실제로 \*\*/\*.war는 build/libs/\*.war에 떨궈진다.

그래서 Remove prefix설정을 통해 build/libs는 삭제하고, 실제 \*.war만 복사해서 옮기도록 설정한다.

여기까지 설정하고 save~

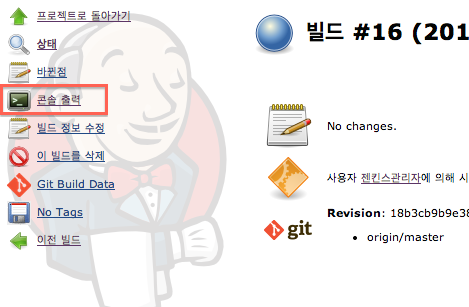


메인으로 돌아와서 빌드버튼을 누르고, 빌드가 정상적으로 끝나고 나면 실제 배포하기로 했었던곳에 정상적으로 war파일이 떨궈져있는지 확인해보면 끝~



혹시 오류가 나거나 진행상태를 보고 싶으면, Item Name을 클릭해서 (여기서는 mvc) 아래와 같이 차례대로 들어가면 콘솔로 빌드 내용을 볼 수 있다.





**Jenkins에서 자바 웹 프로젝트 자동 배포**

[](http://jaydee88.github.io/images/jenkins_logo.png)Jenkins logo

**들어가기**

개발자로서 커리어를 처음 시작하고 모 은행 SI프로젝트에 가서 처음 주어졌던 일이 허드슨(지금은 이름이 바껴 Jenkins) 배포 관리였다. 현업이나 PL/PM들이 주로 아침에 개발서버와 스테이징 서버를 통해서 변경된 기능에 대해서 확인하고 이슈를 확인하는 식이였다. 문제는 개발자들이 야근하면서 로컬에서 작성한 코드들을 커밋하고 퇴근하면서 다른 컴포넌트의 의존성 문제로 허드슨의 빌드가 깨져서 개발서버 인스턴스가 부팅이 되질 않아 아침에 확인을 못하는 문제가 생겼다. 그래서 나같은 신입 개발자들이 주로 밤 늦게까지 빌드가 정상적으로 동작하는 지 확인하고 문제가 생기는 경우 직접 수정하거나 개발자한테 연락해서 수정을 하는 식으로, 그야말로 몸빵(?)을 했던 추억이 있다. 그 이후로는 CI툴은 거의 써보질 못했고 대부분 형상관리 시스템이나 개발 프레임워크에서 지원하는 식으로 배포에 대한 이슈가 없었다.

최근에 부산에 몇몇 업체에 개발프로젝트 일을 하거나 주위의 이야기를 들어보면 자동 배포는 개나 줘버려라는 식으로 개발자 로컬 PC에서 파일 단위로 FTP를 통해서 개발서버에 배포하는 경우가 대부분이였다. 일단 개발자의 수가 작은 프로젝트라서 배포에 대한 고민이 크질 않거나 프로젝트마다 개발 환경이 달라져서 CI구성을 하기 힘든 경우가 많아지니 CI나 자동배포를 포기하게 되는 원인이질 않을까 추측해 본다.

내 생각에는 혼자 개발하지 않는 이상, 개발 프로젝트에서는 꼭 CI툴을 이용해서 자동배포하는 환경이 꼭 있어야 본다. 첫번째는 비용(개발자가 하나하나씩 배포할 경우에 시간과 노력)적인 문제이고 두번째는 서버에서 빌드된 파일들의 관리 문제이다. 자바 프로젝트의 경우 개발자 로컬 PC에서 컴파일한 클래스파일을 서버에 올리는 문제인데 개발서버가 리눅스고 자바 환경이 Open JDK를 사용하는 경우 크게 문제가 없을 거 같은데 유닉스 환경에서 벤더사의 JDK를 사용할 경우에는 문제가 달라 진다. 벤더 사의 JDK로 컴파일된 바이트 코드가 런타임 환경과 다를 경우 성능 이슈나 나중에 알 수 없는 오류를 접할 가능성이 있다. 그러므로 꼭 자바 웹 프로젝트일 경우 CI툴을 사용해서 자동배포를 꼭 하자.

**준비물(환경)**

얼마전에 회사 프로젝트에 배포를 수정한거라 기억이 가물가물하다.

자바 웹 프로젝트이고 프로젝트 루트에 SRC와 WEB(WAS의 도큐먼트 루트와 매칭)이라는 디렉토리가 있고, SRC에 있는 자바 파일을 컴파일되어 개발 서버의 WEB-INF/classes 디렉토리로 배포되어야 한다. 형상관리는 Git을 사용하고 저장소는 Github 유료 플랜으로 비공개 저장소를 사용한다. 개발서버는 데비안 리눅스 버전이고 Open JDK그리고 WAS는 Apache Tomcat이다. Jenkins는 물리적으로 다른 서버에 설치되어 있다. Jenkins 설치 정보는 구글링해 보면 많이 나오니 참고하면 된다.

배포 시나리오는 Git Branch애 개발자가 개발 커밋을 푸시하면 Github 트리거가 발생해서 Jenkins의 웹훅을 호출하고, Jenkins의 스케줄러잡에서 해당 브랜치가 설정되어 있으면 연결된 Git Branch를 체크아웃 받아서 Gradle로 빌드하고 자바 클래스 파일을 War로 묶어서 개발서버에 배포 한다. 그리고 Jenkins 작업 마지막에 원격으로 개발서버에 접속해서 셸스크립트를 실행해서 개발서버 work directory에서 해당 브랜치의 파일들을 pulling해서 WAS의 document ROOT로 복사하는 식이다.

사실 하고 나면 간단한 내용(실제로도 별 내용이 없다)인데 초보자일 경우 아마 환경적인 부분에서 삽질(X) 헤매는 경우가 있을거라 본다. 이런 부분들이나 지적사항은 댓글 남겨 주면 고맙겠다.

**Jenkins 설정**

1) 젠킨스 Github 플러그인 설치. “Jenkkins 관리”-“플러그인 관리” 화면서 Git/Github 연동 플러그인을 설치 한다. 아래는 실제 설치된 목록이다.

|  |
| --- |
| Delivery Pipeline Plugin  Visualisation of Delivery/Build Pipelines, renders pipelines based on upstream/downstream jobs. Full screen view for information radiators.  0.9.5 0.9.4    GIT client plugin  Shared library plugin for other Git related Jenkins plugins.  1.18.0 1.17.1    GIT plugin  This plugin allows use of Git as a build SCM. A recent Git runtime is required (1.7.9 minimum, 1.8.x recommended). Plugin is only tested on official git client. Use exotic installations at your own risks.  2.4.0 2.3.5    GitHub API Plugin  This plugin is a library plugin used by other GitHub related plugins to share the same libraries. This plugin does not have any user visible feature by itself. There's no need to install this plugin manually, although you want to keep it up to date.  1.69 1.68    GitHub plugin  This plugin integrates Jenkins with Github projects.  1.12.0 1.11.3    Groovy  This plugin adds the ability to directly execute Groovy code.  1.27 1.25    Maven Integration plugin  Jenkins plugin for building Maven 2/3 jobs via a special project type.  2.11 2.10    SSH Slaves plugin  This plugin allows you to manage slaves running on \\*nix machines over SSH.  1.10 1.9 |

2) 새로운 스케줄러 잡 등록  
젠킨스 첫 화면에서 “새로운 Item”을 클릭해서 새로운 스케줄 잡을 등록하자. 사용하는 Jenkins가 한글화되어 있어서 영어 단어와 다를 수 있으니 유의하길 바란다.  
그리고 스케줄러 잡에는 왠만하면 공백을 넣지 않도록 한다. 잡 이름이 시스템상의 경로가 될 수 있기에 편의상 공백없는 이름을 추천 한다.

* 기본 정보 “Github project”에서 Git Repository 주로 입력
* 소스코드 관리 항목에서 Git을 선택하고 “Repository URL”과 젠킨스에서 사용할 GitHub 계정을 추가한다. “Branches to build”에서 실제 빌드한 브랜치를 적어 준다. GitFlow를 사용해서 Devel이나 Release브랜치를 기입 한다.
* Build항목에서 Invoke Gradle 항목을 선택한다 실행 권한 문제로 Jenkins에서 제공하는 Gradle를 사용했다. “Tasks”는 컴파일 빼곤 다른 게 없어 “build”를 입력하고 “ROOT build script”에서 Gradle 스크립트과 위치한 경로를 기입 한다.

레드햇 리눅스에 yum으로 설치된 젠킨스일 경우 아래 경우에는 gradle스크립트를 위치해 놓는다.

|  |
| --- |
| /var/lib/jenkins/jobs/(Jenkins 스케줄러 이름)/workspace |

아주 간단한 Gradle 스크립트다. 빌드에서 class파일을 build/bin으로 타겟디렉토리를 지정 한다. 자바 웹 프로젝트이기 때문에 서블릿 jar파일을 빌드 클래스패스에 추가 했다.

|  |
| --- |
| apply plugin: 'java' apply plugin: 'war' apply plugin: 'distribution'  sourceSets {  main {  output.classesDir = 'build/bin'  output.resourcesDir = 'build/bin'  java {  srcDir 'src'  }  resources {  srcDir 'src'  }  }  } // In this section you declare where to find the dependencies of your project repositories {  // Use 'maven central' for resolving your dependencies.  // You can declare any Maven/Ivy/file repository here.  mavenCentral() }  // In this section you declare the dependencies for your production and test code dependencies {  // The production code uses the SLF4J logging API at compile time  compile 'org.slf4j:slf4j-api:1.7.5'  providedCompile 'javax.servlet:servlet-api:2.5'  compile fileTree(dir: 'lib', include: '\*.jar')  } |

* 다시 Jenkins 스케줄러 화면에서 “빌드 후 조치” 섹션에서 “Send build artifacts over SSH”항목에 개발 서버에 배포할 파일, 여기서는 빌드한 클래스 파일은 build/bin밑에 war파일을 sftp방식으로 배포할려고 한다. 배포한 파일을 ssh로 셸스크립트를 이용해서 archive 해제해서 웹루트에 복사 한다. 항복별로 보면 “Transfers”-“Transfer Set Source files”에서 "\*\*/\*.war",입력했고 exec command 에서 셸스크립트의 경로를 지정했다. 참 SSH Server는 미리 Jenkins 전역 설정 화면에서 미리 추가해 놓아야 한다. 나 같은 경우 Jenkins->개발서버로 SSH로 접속시에 공개키로 ssh접속이 가능토록 미리 설정해 놓아서 따로 인증 정보를 추가할 필요가 없다. 이런 방식으로 하길 권한다.

|  |
| --- |
| /home/tomcat/deploy/sh/runapp.sh; |

나머지 정보는 환경에 맞도록 입력하고 저장 한다. Jenkins에서 “Build Now”를 선택해서 해당 잡의 Console 로그 화면에서 Git까지 체크아웃하고 Gradle까지 실행되었다면 거의 다 한셈이다.

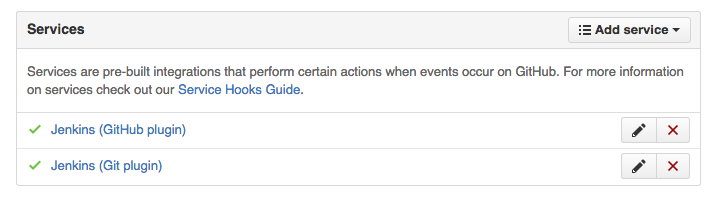
이젠 빌드후에 개발서버에서 실행된 셸스크립트 파일을 간단하게 만들어 봤다. 위에서 설정한 runapp.sh의 내용이다. Git에서 프로젝트 소스를 체크아웃해서 워크디렉토리를 통해 pulling 받은 소스를 웹루트에 복사하는 스크립트이다.

|  |
| --- |
| #!/bin/sh cd home/tomcat/webapps/app/ROOT; jar -xf /home/tomcat/webapps/app/ROOT/build/libs/workspace.war rm -rf /home/tomcat/webapps/app/ROOT/build/; # 워킹 디렉토리에서 Git pulling cd /home/tomcat/deploy/(프로젝트 체크아웃 디렉토리)/web; git checkout develop; git pull; # 웹루트로 카피 cp -rf /home/tomcat/deploy/(프로젝트 체크아웃 디렉토리)/web/\* /home/tomcat/webapps/app/ROOT/; # tomcat 컨테이너 재기동 /home/tomcat/tomcat7/bin/shutdown.sh; /home/tomcat/tomcat7/bin/startup.sh; # 슬랙 웹훅으로 빌드 성공 메시지를 전송하면 더 좋다.  curl -X POST --data-urlencode 'payload={"channel": "#server\_room", "username": "bot", "text": ":ok\_hand::ok\_hand:deploy complete.", "icon\_emoji": ":ok\_hand:"}' (웹훅 URL) > /dev/null 2>&1 |

다시 Jenkins로 돌아와서 “Build Now”로 빌드를 실행해서 결과를 확인 한다.

**Github 설정**

브랜치에 push이벤트가 발생하면 위 Jenkins가 빌드 잡을 실행하도록 서비스 훅 설정이 필요하다. 해당 리파지터리 “Settings”메뉴에서 “Webhooks & Services”메뉴를 선택하고 아래 “Service” 항목에서 Jenkins 정보를 추가 하자.

[](http://jaydee88.github.io/images/github_conf.png)Github hook service

Github설정까지 끝났으면 브랜치에 소스를 push해서 Jenkins스케줄러 잡이 동작하는 지 확인 해본다. push시에 Jenkins가 동작하지 않는다면 웹 훅 설정이 잘 못된 것이므로 다시 확인해 본다.

사실 내용에 나온 Gradle도 잘 모르는 상태고 Jenkins 설정도 오랫만에 보는거라 삽질을 좀(사실 꽤) 했다. 정리한 내용이 다음 삽질자(?)에게 도움이 되었으면 좋겠다.

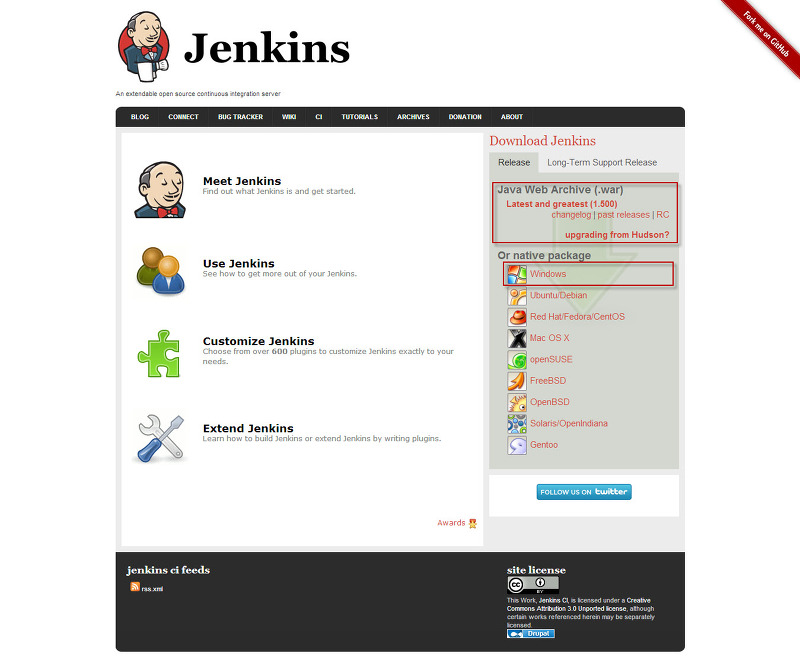
http://jawscompany.tistory.com/6

Jenkins : 젠킨스

많이 접해 보지 못한 이름입니다. 허드슨(Hudson)과 비슷한 자동배포 웹 애플리케이션입니다. 허드슨 개발자들이 나와서 제작한 툴로서, 허드슨 보다 버전이 낮지만 강력한 기능을 자랑합니다. 허드슨은 많이들 사용하고 있고 지금은 3.0.0 버전대로 올라가 있습니다. 하지만 버전이 너무 높다 보니 플러그인과의 호환성이 다소 떨어지고 있으며, 필자가 허드슨을 사용하기 위해 세팅을 해 보았지만 CVS에서 파일은 들고오되 후속처리인 Ant에서 에러가 발생하여 부득이하게 Jenkins으로 갈아탔었지만, 허드슨보다 호환성이 좋고, 후속처리인 Ant부분에서도 완벽하게 동작합니다.

**1. 설치방법**

젠킨스는 현재 1.500 버전이 최신버전이며, 다운 받는 곳은 공식 홈페이지인 젠킨스([https://jenkins-ci.org](https://jenkins-ci.org/))에서 받을 수 있습니다.



.war 파일로 되어 있는 젠킨스와 여러 플랫폼에서 설치할 수 있는 패키지 형태로 되어 있는 젠킨스가 있습니다. 이 글에서는 war 파일이 아닌 windows 설치 패키지로 받아 사용하겠습니다.

jenkins-1.500.zip 파일을 압축 푸시면, setup.exe와 jenkins-1.500.msi 두 가지의 실행 파일이 있는데 익숙한 setup.exe를 더블클릭하여 next 버튼을 차례차례 눌려서 설치를 완료합니다.

**2. 젠킨스 웹 애플리케이션에 접속**

젠킨스를 설치하셨다면 해당 웹에 접속을 해야되는데, war 같은 경우에는 was에서 설정한 도메인과 포트로 접속을 하지만 설치파일로 하셨다면 어떻게 접속을 해야할까요? 설정하는 부분도 없었잖아요?

젠킨스 설치파일은 로컬아이피(127.0.0.1 or localhost)와 포트(8080)을 사용하여 접속이 가능합니다. 만약 다른 PC에 설치하였고 같은 네트워크에 속해 있다면 해당 IP주소를 입력하여 접속합니다.

**3. 젠킨스 설정 및 플러그인 설치**

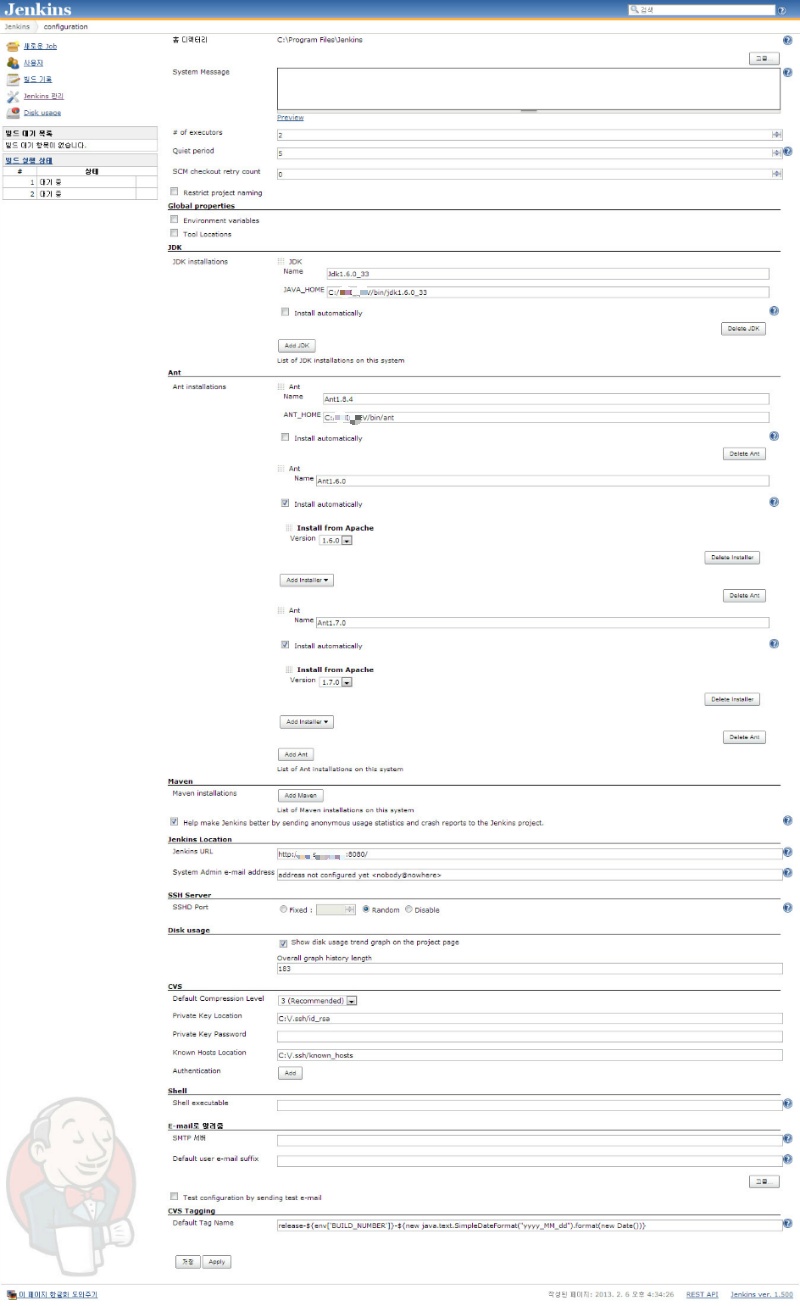
설치파일 또는 war로 설치하게 되면 젠킨스에서 추천하는 플러그인이 모두 설치가 되어지는데, 여기서 그냥 써도 상관은 없지만 가끔 낮은 버전으로 인해 CVS에서 파일을 들고 오지 못한다거나 그런 현상이 가끔 있습니다. 또한, 해당 플러그인에 의존성이 있는 플러그인은 높은 버전으로 인해 동작이 되지 않을 가능성도 있으니 주의하시고 업데이트 또는 설치를 완료하십시오.



Jekins 관리를 클릭하여 관리 페이지로 이동합니다.



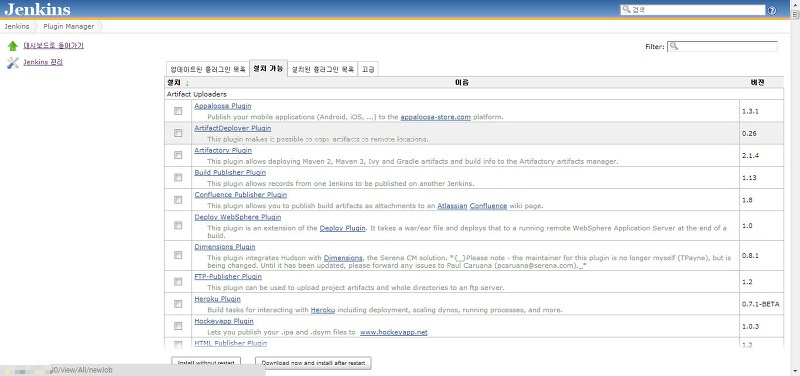
젠킨스 관리에는 크게 시스템 관리, 보안 설정, 플러그인 관리가 있으며, 각각 서비스에 도움이 되는 관리화면이 준비되어 있습니다.



시스템 설정을 하기 전에 플러그인을 업데이트 또는 설치를 하는 것을 추천드립니다. 플러그인 버전에 따라 설정화면이 달라지므로, 두번 설정하는 것보다 한번에 설정하는 것이 여러모로 편하겠죠?



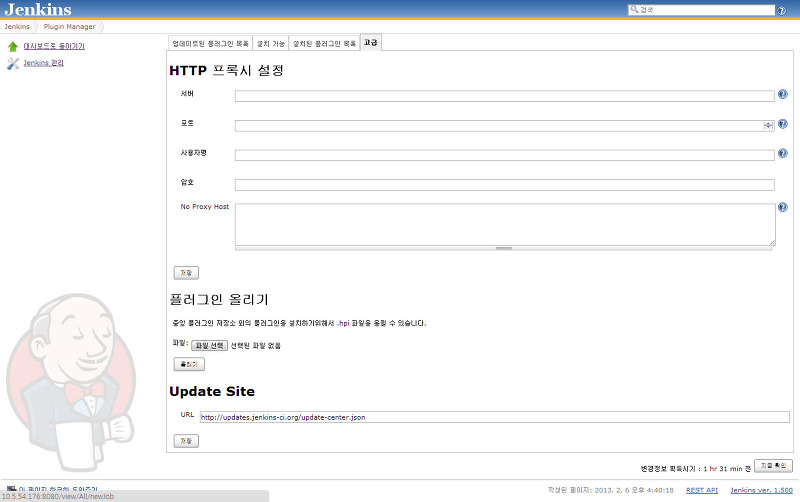
업데이트가 있다면 업데이트 목록이 출력됩니다. 1.500버전을 설치하게 되면 4가지 정도 업데이트가 이루어집니다.



CVS를 비롯한 배포에 필요한 플러그인을 설치할 수 있습니다. Filter 기능이 있지만, 컨트롤 F로 직접 찾아서 하는 것이 편하겠습니다.



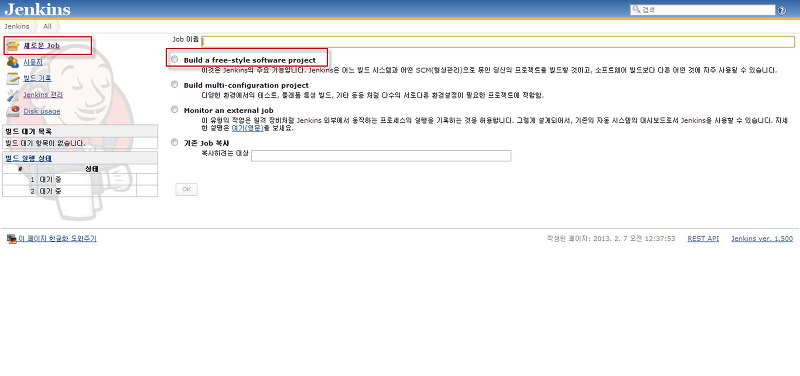
상기 화면은 현재 설치된 플러그인 목록을 나타냅니다. 만약 사용하지 않는다면 체크를 해제하시면 되고, 지우고 싶다면 Uninstall, 고급 탭에서 직접 플러그인을 업로드하여 설치하고 최신 버전으로 업데이트 하였다면 다운그레이드가 가능해집니다.



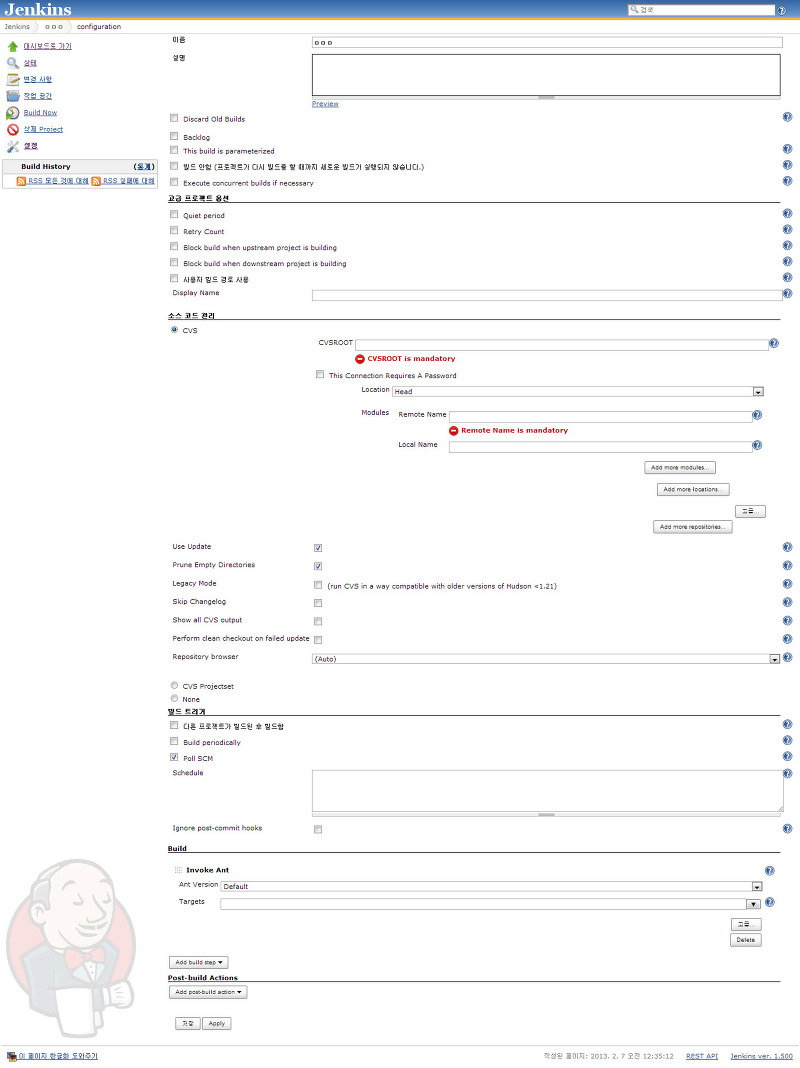
고급 탭은 네트워크가 불안정할 때 등, 우회를 하여 접속을 해야할 경우에 설정하는 부분이고, 플러그인 올리기에서는 사용자가 직접 플러그인 파일을 올려 설치가 가능합니다. 이때에는 버전에 상관없이 올릴 수 있으나, 의존도에 따라 정확하게 동작이 구동되지 않을 가능성이 있습니다.

**4. 자동배치 작업 생성**

자동으로 컴파일하여 배포를 하기 위해 젠킨스에 새 작업을 등록해야합니다. 작업 등록은 여러개가 가능하고, 한 작업으로 배포를 하여도 되고, 여러 작업에 걸쳐 배포를 하여도 됩니다. 특정 작업이 성공하면 빌드를 하도록 설정할 수도 있습니다.



왼쪽의 메뉴에서 새 작업(New Job)을 클릭하면 프로젝트 이름과 그 프로젝트의 타입을 선택하는 화면이 나타납니다. 프리스타일, 멀티 설정, 외부 장비 모니터링, 기존 프로젝트 복사가 있지만 우리에게 필요한건 프리스타일입니다. 자유롭게 새 작업을 설정하도록 합니다.



프로젝트 이름은 앞서 입력한 이름으로 나타나며, 이 프로젝트가 무엇을 의미하고 있는지 간단한 설명을 포함하여 여러 설정을 할 수 있습니다.

(※ 상기 설정화면은 CVS를 기준으로 설명하고 있으며, 버전은 2.7버전이고, Ant도 앞서 시스템 설정에서 설정하였다는 가정으로 설명합니다.) 소스코드 관리는 CVS, SVN, 등등 많이 있고(플러그인을 설치해야 리스트가 나옵니다.) 여기서는 CVS를 선택하여 상세 설정을 보이게 해서 cvsroot를 설정하고 어떤 모듈을 불러서 자동 배치를 할 지 설정합니다. 구 버전대의 CVS 플러그인을 설치하고 있다면, 패스워드를 직접 cvsroot에 삽입 또는 .cvspass 라는 접속정보에 암호가 있어야 됩니다. (:pserver:user\_id:passwd@host\_name:Module)

Use Update에 체크를 하면, 새로운 파일이 있을 경우에 받아서 배포를 하게 됩니다. 다음으론 빌드 트리거를 설정하게 되는데, 다른 프로젝트가 빌드된 후 빌드를 하면 현재 프로젝트가 빌드되기 전에 꼭 의존성이 필요한 빌드가 있을 경우에 지정하시면 되고, build periodically는 빌드 주기를 설정하는 입력 박드가 생성되고 여기에 cron 문법을 넣으시면 됩니다. Poll SCM은 build periodically와 비슷한 cron으로 이루어 져있는데, 한가지 다른 점이라면 build periodically는 특정 주기마다 매번 동작하는 것이고, Poll SCM은 주기를 돌지만 SCM에 변경이 이루어졌다면 동작하는 것입니다. 써본지가 얼마 되지 않아서 거기서 거기같은 트리거 인것 같네요 ㅎㅎ, 저 같은 경우에는 자동 빌드같은 경우에는 Poll SCM을 사용하고 FTP 전송으로 build periodically를 씁니다. (cron 문법 \*/30 \* \* \* \*  : 매일 30분 단위로 실행)

Build부분에는 Ant, Shell, Windows Command, 등 빌드 후속 처리를 위한 설정을 할 수 있습니다. 시스템 설정에서 설치한 Ant를 사용하기 위해 Invoke Ant를 선택하고 Ant version, target, option을 설정하여 처리합니다.

저장 또는 apply로 설정을 완료하고, Build Now를 하여 빌드를 합니다.

## Jnekins 과 Tomcat 8 연동하여 WAR 파일 배포

<http://hreeman.tistory.com/m/138>

## Jenkins와 SSH를 이용한 배포 연동

http://hreeman.tistory.com/m/136

## Jenkins 와 Gradle을 이용한 Build 연동

<http://hreeman.tistory.com/m/135>

Pipeline script

|  |
| --- |
| node {  def build\_number = "${BUILD\_NUMBER}";  def targetWar = "${WORKSPACE}/${build\_number}/cms.war";  stage('GitPull') { // for display purposes  if(!fileExists(targetWar)){  dir('cms'){  git branch: 'master', credentialsId: 'c499e1d2-0759-4c98-b2e2-57fd2502c22f', url: 'http://admin@st.omnicns.co.kr/scm/onf/cms\_jp.git'  }  dir('omnifit\_common'){  git credentialsId: 'c499e1d2-0759-4c98-b2e2-57fd2502c22f', url: 'http://admin@st.omnicns.co.kr/scm/onf/omnifit\_common\_jp.git'  }  }  }  stage('GradleBuild') {  if(!fileExists(targetWar)) {  dir("${WORKSPACE}/cms/omnifit") {  sh "gradle war -DOMNIFIT\_COMMON=${WORKSPACE}/omnifit\_common/common"  }  sh "mkdir ${WORKSPACE}/${BUILD\_NUMBER}"  sh "cp ${WORKSPACE}/cms/omnifit/build/libs/cms.war ${targetWar}"  }else{  sh 'echo oldversion\_war'  }  }  stage('Publish') {  sh "ssh mindcare\_jp@119.206.205.171 rm -rf /home/mindcare\_jp/tomcat/webapps/\*"  sh "scp ${targetWar} mindcare\_jp@119.206.205.171:/home/mindcare\_jp/tomcat/webapps"  sh "ssh mindcare\_jp@119.206.205.171 /home/mindcare\_jp/tomcat/bin/shutdown.sh"  sleep 2  sh "ssh mindcare\_jp@119.206.205.171 /home/mindcare\_jp/tomcat/bin/shutdown.sh"  sleep 2  sh "ssh mindcare\_jp@119.206.205.171 /home/mindcare\_jp/tomcat/bin/shutdown.sh"  sleep 5  sh "ssh mindcare\_jp@119.206.205.171 /home/mindcare\_jp/tomcat/bin/startup.sh"  }  } |