Jenkins:

Continuous integration with Jenkins on Amazon EC2

What is continuous integration

Example: Joe and Jane working on a Project. They each individually implement few classes and code them up and unit tested and made sure everything working as expected. Thinking that program is going to be robust.

Once they are done unit testing integrate them. Everything breaks. (code fails to compile or bugs). This situation is going to be known as integration hell

Integration hell is extremely risky for a project.

Difficult to determine how long will take to resolve the integration problems. May vastly exceed budget and schedule.

To avoid these situations Continuous Integration program came up (Originated from eXtreme Programming(XP). To avoid integration hell, Integrate continuously throughout the entire project.

Mitigates risks associated with integration software.

Rather than developed and waiting to Integrate weeks and months integrate early and integrate often (i.e. on every change).

Every time developer pushes the new change to the Repository, project code checked out and build and compiled and all of the testes should be run.

Continuous Integration server

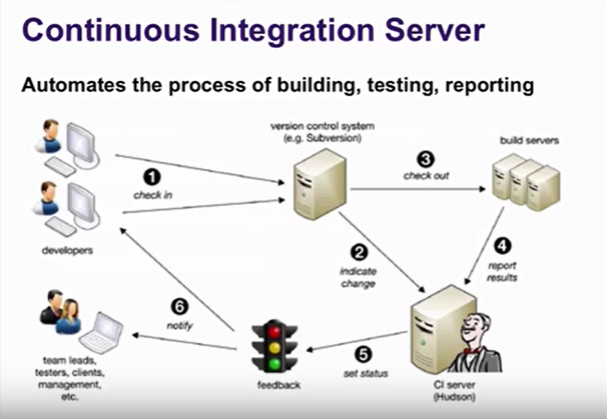
Automates the process of building testing and reporting.

1. Developer pushes the changes into Repository (Ex: GitHub)

2. CI server will be notified (either CI server pull the repository or Repository call the CI server to notify the changes).

3. When the CI server sees the changes has been pushed to the repository it will clone the repository on the server or may actually instruct the build servers to check out the code and build the project. If the build fails send a notification to the Project Team. If the build is successful it will run all the project tests and if any test fails again send notifications to the team

4. Finally CI server generates various reports. (ex: Line coverage of Project tests)



Benefits of the CI server:

Developers might forget to run the tests. (Don't break the build)

It might take too long to run the tests

We might need to test the code in various environments.

Different architectures (32-bit, 64-bit, ARM PowerPC)

Different platforms (Windows, Linux, Mac, Solaris)

CI server can use them (other platforms) as build slaves to run the changes on different platforms. When the CI server detects the change in Repository CI server will instruct the build slaves to check out the latest version of the code to build and test it.

CI server can provide some reports useful insights to the team.

Can track the metrics like line coverage (percentage of lines executed by program tests.)

Can run all sorts of utilities on our code. (Check style and find bugs)

Can deploy automatically (deploy a web project to the test or staging server automatically)

Popular CI servers: Jenkins, Hudson, CruiseControl, TeamCity etc

Tasks:

* Setup an Amazon EC2 instance
* Install Jenkins
* Configure GitHub to notify Jenkins of Changes
* Configure Jenkins
* Clone Repository when notification received
* Build the Project
* Run the Project tests
* Generate Line Coverage Reports.

Step 1: Create EC2 instance (Ubuntu Server 13.10)

Switch to Terminal window

$ ssh -i ~/Downloads/Wordpress-Linux.pem Ubuntu@ Hostname

$ chmod 600 ~/Downloads/Wordpress-Linux.pem

$ exit

make it command ssh -i ~/Downloads/Wordpress-Linux.pem Ubuntu@ Hostname shorter

$ vim ~/.ssh/config

Inside the editor

Host ec2

Hostname ec2-50-17-10-64.compute-1.amazonaws.com

User Ubuntu

IdentityFile ~/.ssh/ Wordpress-Linux.pem

$ mv ~/Downloads/Wordpress-Linux.pem ~/.ssh

$ ssh ec2 (Shorter form of the ssh -i ~/Downloads/Wordpress-Linux.pem Ubuntu@ Hostname)

Fixing Locales in Ubuntu 13.04 on Amazon EC2

$ sudo apt-get install language-pack-en

### Installing Jenkins

### Add the Repository for the Jenkins package. In order to do that we need to install the public key for the Jenkins Repository to tell Ubuntu that this is a trusted Repository.

$ wget -q -O - <http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key> | sudo apt-key add -

Below command to tell Ubuntu where the package Repository is

$ echo "deb <http://pkg.jenkins-ci.org/debian> binary/" | sudo tee -a /etc/apt/sources.list.d/jenkins.list

$ sudo apt-get update

$ sudo apt-get install jenkins

Once Jenkins is installed we can confirm that Jenkins is running by typing the below command

$ ps -ef | grep jenkins

$ sudo apt-get install 'apache2'

apache2 modules:

$ sudo a2enmod proxy

$ sudo a2enmod proxy\_http

Configure the apache2 actually proxy the requests from port 80 to 8080

$ sudo vim /etc/apache2/sites-available/jenkins.conf

<VirtualHost \*:80>

ServerName ec2-50-17-10-64.compute1.amazonaws.com

ProxyRequests Off

<Proxy \*>

Order deny,allow

Allow from all

</Proxy>

ProxyPreserveHost on

ProxyPass / <http://localhost:8080/>

</VirtualHost>

$ sudo a2ensite jenkins (Enabling site jenkins)

$ sudo service apache2 reload (service apache2 reload)

Installing Jenkins on Ubuntu server link:

https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+on+Ubuntu

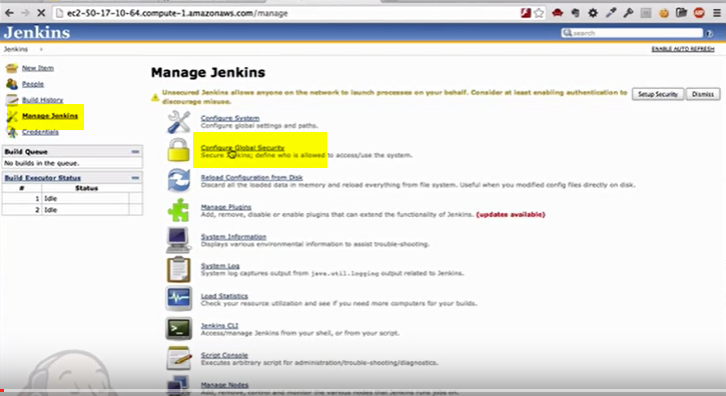
Configure Jenkins:

By default, Jenkins doesn't security enabled.

Launch the site Jenkins.

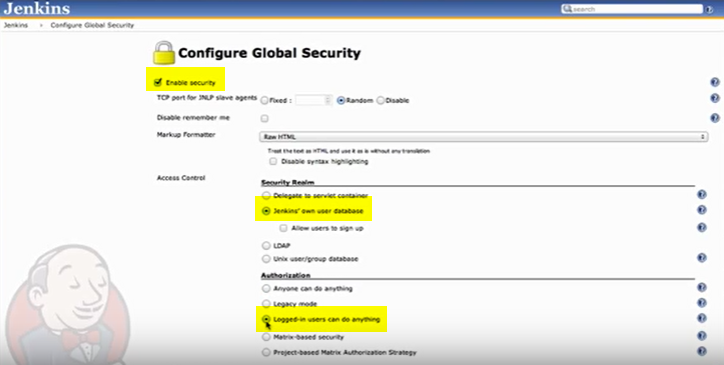
1. Click on Manage Jenkins

2. Configure Global Security



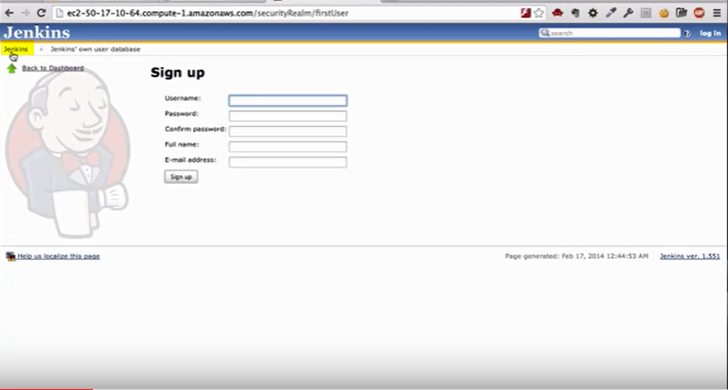
3. Enable Security (Please look at the below screenshot for the details)

and click on save button.



We did disable to ability to sign up. Fortunately, Jenkins allows us to sign in first user.

Click on the Jenkins link on the Top left most corner and then signup.

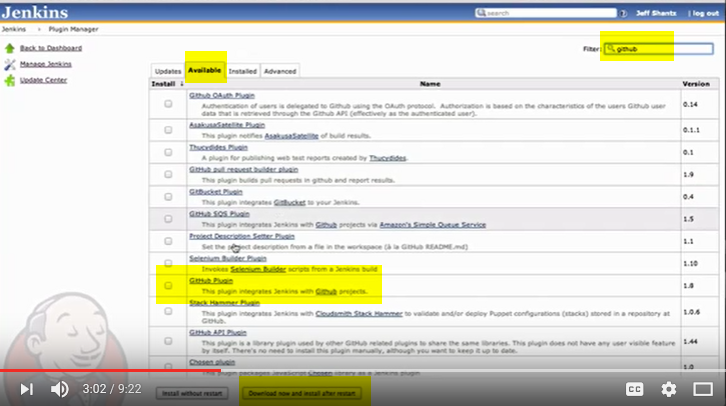


Want to setup global configurations for Jenkins.

Before we do that go to the manage Jenkins and click on mange plugins.

Click on all the checkboxes and click on Install without restart.

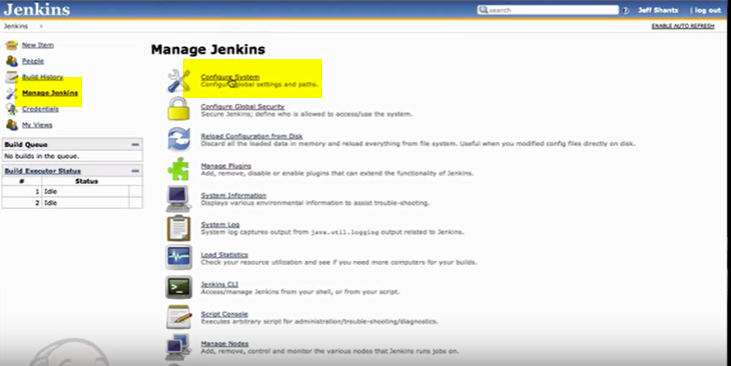
Then go back to manage Plugins. Follow the below screenshot



then restart the Jenkins.

Once it is restarted you can create the accounts for other users by clicking on Manage Jenkins and then manage users tab.

Then go back to manage Jenkins and click on Configure System.



Switch back Ec2 instance

Installing Java/Maven/Git

$ sudo add-apt-repository ppa:webupd8team/java

$ sudo apt-get update

$ sudo apt-get install oracle-java7-installer maven git-core

Verify and make sure softwares are installed

$ javac -version

find the java home by typing the below command

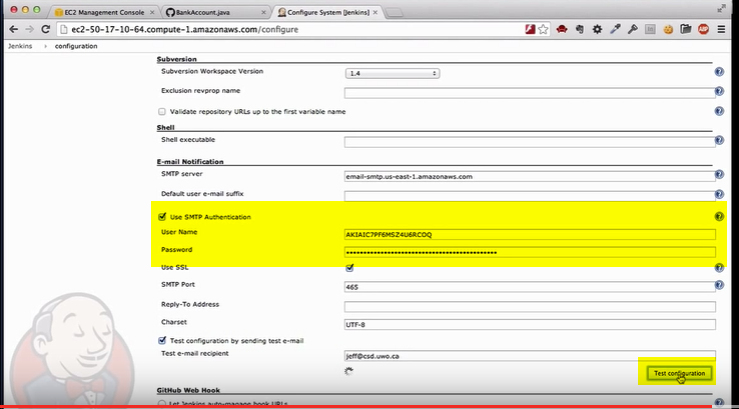
$ readlink -f /usr/bin/javac

/usr/lib/jvm/java-7-oracle/bin/javac (copy everyting before bin and paste in Jenkins site)

$ mvn -version

$ readlink -f /usr/bin/mvn

/usr/share/maven/bin/mvn (copy everyting before bin and paste in Jenkins site)



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A **Project Object Model** or **POM** is the fundamental unit of work in **Maven**. It is an XML file that contains information about the project and configuration details used by **Maven** to build the project. It contains default values for most projects.

**2/21/2017 (James Lee)**

JAVA\_HOME environment variable points to the installation path for the Java Development kit

It is required by other applications which use java.

Text Direction: Install Java

Section 1, Lecture 5

#### **Configure Java\_Home on Windows:**

* Right click My Computer and select Properties.
* On the Advanced tab, select Environment Variables, and then edit JAVA\_HOME to point to where the JDK software is located, for example, C:\Program Files\Java\jdk1.6.0\_02

#### **Configure Java\_Home on Linux:**

* Login to your account and open the startup script file which is usually ~/.bash\_profile  file (or can be .bashrc depending on your envrionment settings)

$ vi ~/.bash\_profile

* In the startup script, set JAVA\_HOME  and PATH
  + Korn and bash shells:
    1. export JAVA\_HOME=jdk-install-dir
    2. export PATH=$JAVA\_HOME/bin:$PATH
  + Bourne shell:
    1. JAVA\_HOME=jdk-install-dir
    2. export JAVA\_HOME
    3. PATH=$JAVA\_HOME/bin
    4. export PATH
  + C shell:
    1. setenv JAVA\_HOME jdk-install-dir
    2. setenv PATH $JAVA\_HOME/bin:$PATH
    3. export PATH=$JAVA\_HOME/bin:$PATH

jdk-install-dir  is the JDK installation director, which should be something similar to /usr/java/jdk1.5.0\_07/bin/java

* Type the following command to activate the new path settings immediately:

$ source ~/.bash\_profile

* Verify new settings:  
  $ echo $JAVA\_HOME<br>$ echo $PATH

Jenkins use master and Slave Architecture to manage distributed builds.

Master: Schedule the build jobs, Dispatch builds to the slaves for the actual job execution and Monitor the slaves and record the build results. Can also execute build jobs directly.

Slave: Small java program listens to the master. Execute build jobs dispatched by the master. It is always possible to configure the particular job on slave machine.

Job or Project: These two terms are used interchangeably. They all refer to runnable tasks that are controlled/monitored by Jenkins.

Slave/Node:

Slaves are computers that are set up to build projects for a Master.

Jenkins runs a separate program called “slave agent” on slaves.

When slaves are registered to a master, a master starts distributing loads to slaves

Node is used to refer to all machines that are part of Jenkins grid, slaves and master.

Executor:

Executor is a separate stream of builds to be run on a node in parallel.

A node can have one or more executors.

Build: is a result of one of the projects.

Plugin: is a piece of software that extends the core functionality of the core Jenkins server.

Created and build a simple Jenkins build job.

Install Git and GitHub

Install and Configure Maven

What does Maven Do?

Maven describes how the software is built

Maven describes the project dependencies.

Java build Tools:

Maven, Gradle and Apache Ant

Text Direction: Create our First Maven-based Jenkins Project

Section 2, Lecture 15

#### **Command to clone the repo:**

git clone https://github.com/jleetutorial/maven-project.git

#### **Git repo webpage:**

https://github.com/jleetutorial/maven-project

#### **URL of the Git repo (the one that Jenkins needs):**

https://github.com/jleetutorial/maven-project.git

#### **Maven Project type**

In this lecture you might NOT see the Maven Project  type available to choose from when you click the New Item  button.

In order to have the Maven Project  type, you will need to install the Maven Project Plugin.

https://wiki.jenkins-ci.org/display/JENKINS/Maven+Project+Plugin If you are building a complicated maven project, definitely pick the maven project type here, this is a project template that Jenkins has reserved specifically for Maven-based projects.

Jenkins understands the pom dot XML file format and would take advantage of it which can greatly reduce the manual configuration.

But considering our maven project is a very simple one and the free style project would suffice, so here we just  go with the free style project.

Source control polling in Jenkins:

SSH-KEYGEN

Text Direction: Source Control Polling in Jenkins

Section 2, Lecture 18

**Checking for existing SSH keys:**

https://help.github.com/articles/checking-for-existing-ssh-keys/

**Generating a new SSH key and adding it to the ssh-agent:**

https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/

**Adding a new SSH key to your GitHub account:**

https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* <https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>

Save the public key in giyhub.(under settings)

Goto the Github and copy the link(choose SSH)

ubuntu@ip-172-31-46-246:~$ git clone [git@github.com:kollasin9/maven-project.git](mailto:git@github.com:kollasin9/maven-project.git)

Goto the maven project repository

ubuntu@ip-172-31-46-246:~/maven-project$ git commit --allow-empty

Text Direction: Other Build Triggers

Section 2, Lecture 20

#### **Jenkins GitHub Plugin:**

https://wiki.jenkins-ci.org/display/JENKINS/GitHub+Plugin

Text Direction: Code Quality and Code Coverage Metrics Report

Section 3, Lecture 22

#### **Git commands:**

Show the working tree status:

git status

Add file contents to the index:

git add <file path>

Record changes to the repository:

git commit -m <commit message>

Record changes to the repository:

git commit -m <commit message>

Update remote refs along with associated objects:

git push

#### **PMD Jenkins plugin:**

https://wiki.jenkins-ci.org/display/JENKINS/PMD+Plugin

#### **Findbugs Jenkins Plugin:**

https://wiki.jenkins-ci.org/display/JENKINS/FindBugs+Plugin

Browse Q&A

Continue

Go to Dashboard

Text Direction: Jenkins' Support for Gradle, Ant and Shell Scripts

Section 3, Lecture 24

#### **Apache Ant:**

http://ant.apache.org/

#### **Ant Targets:**

http://ant.apache.org/manual/targets.html

#### **Gradle:**

https://docs.gradle.org/current/dsl/index.html

Archive build artifacts: Deploy the artifacts to the staging environment then QA’s can do the manual testing before moving it to the production environment.

Archive the WAR artifact from the current Maven project then move that to the staging environment.

Install and configure the Tomcat as the staging environment.

What is Tomcat?

Tomcat is an open source webserver and provides a pure java HTTP web server environment in which java code can run.

How to find the Ubuntu version:

$ lsb\_release –a

Change the Tomcat server port. The reason for the changing the port is Jenkins runs on port 8080. The default port for Tomcat also 8080.

To change the port goto conf/server.xml file and find the Connector at line 69. Change it to 8090.

Script used to start up the Tomcat server is startup.sh or startup.bat (windows)

Tom make it executable

$ ubuntu@ip-172-31-46-246:/opt/tomcat/apache-tomcat-8.5.11/bin$ sudo chmod +x \*

Run the startup script to spin up the Tomcat server.

$ ubuntu@ip-172-31-46-246:/opt/tomcat/apache-tomcat-8.5.11/bin$ ./startup.sh

In order to deploy artifacts to tomcat server Jenkins need to know Tomcat credentials to perform the operation.

Tomcat credentials are available in below file

ubuntu@ip-172-31-46-246:/opt/tomcat/apache-tomcat-8.5.11/conf$ vi tomcat-users.xml

<!--

<role rolename="tomcat"/>

<role rolename="role1"/>

<user username="tomcat" password="<must-be-changed>" roles="tomcat"/>

<user username="both" password="<must-be-changed>" roles="tomcat,role1"/>

<user username="role1" password="<must-be-changed>" roles="role1"/>

-->

Change the above file like below.

<role rolename="manager-script"/>

<role rolename="admin-gui"/>

<user username="tomcat" password="tomcat" roles="manager-script,admin-gui"/>

Next step

$ ubuntu@ip-172-31-46-246:/opt/tomcat/apache-tomcat-8.5.11/bin$ ./shutdown.sh

ubuntu@ip-172-31-46-246:/opt/tomcat/apache-tomcat-8.5.11/bin$ sudo ./startup.sh

Deploy to staging environment:

Install copy artifact and deploy to container plugins.

Deploy our application to staging environment.

Copy artifact plugin allows Jenkin to copy artifact from other jobs.

Deploy to Container plugin Will actually deploy the war files to the Tomcat.