

# CI,CD AND JENKINS

# JENKINS – AN INTRODUCTION



# DISCLAIMERS

- Sun, Sun Microsystems, Solaris, Java and JavaServer Pages are trademarks or registered trademarks of Oracle Corporation. UNIX is a registered trademark in the United States and other countries, exclusively licensed through 'The Open Group'. Microsoft, Windows, WindowsNT, and Win32 are registered trademarks of Microsoft Corporation. Linux is a registered trademark of Linus Torvalds.
- "Apache Tomcat" and "Tomcat" are trademarks of the Apache Software Foundation. Use of these trademarks is subject to the terms of section 6 of **Apache License, Version 2.0** (current).
- All other product names mentioned herein and throughout the entire web site are trademarks of their respective owners.

# COURSE TOPICS

- **Module 01**
  - An Overview of Jenkins
- **Module 02**
  - Getting Started with Jenkins
- **Module 03**
  - An overview of Plugins
- **Module 04**
  - Introduction to Jenkins Jobs
- **Module 05**
  - Build using Jenkins
- **Module 06**
  - Parameterized Build
- **Module 07**
  - Automation of Testing and Static code Analysis
- **Module 08**
  - Managing Jenkins Nodes and Masters
- **Module 09**
  - Administering Jenkins
- **Module 10**
  - Tying it all together

# OBJECTIVES

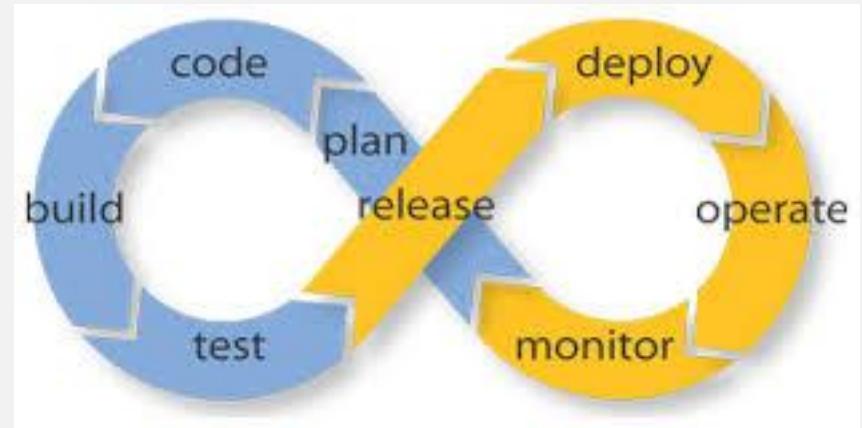
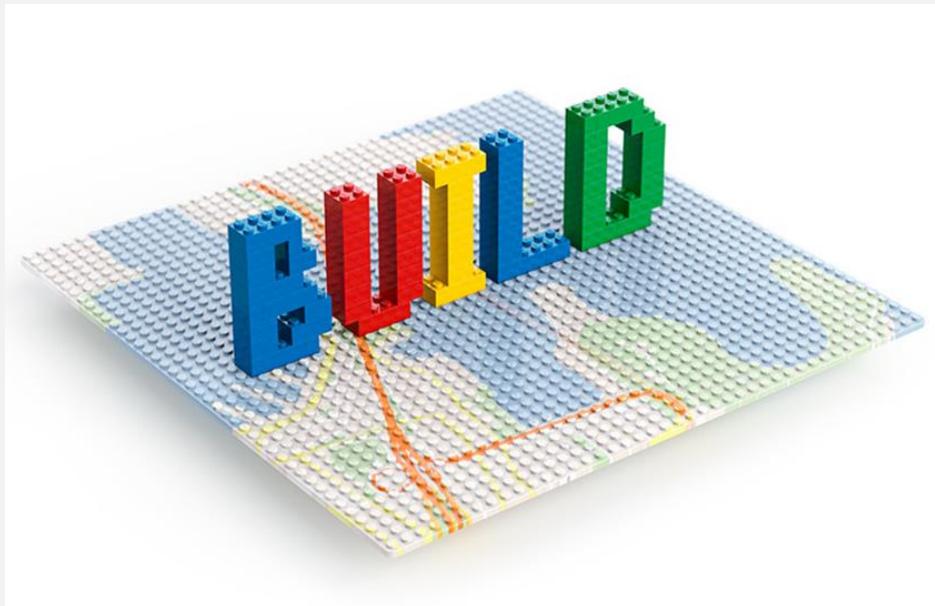
- At the end of this course, the learner should
  - Understand basics of CI and how Jenkins helps CI
  - Creating and managing builds using Jenkins
  - See how to integrate build tools with Jenkins
  - Automation of testing and static code analysis
  - Basic Administration of Jenkins
  - Perform simple deployments using Jenkins

# MODULE 01

## AN OVERVIEW OF JENKINS



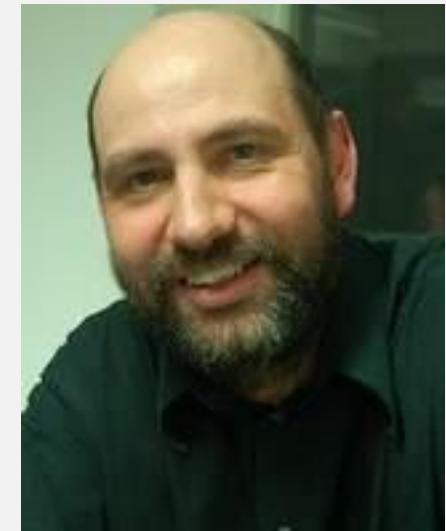
# CONTINUOUS INTEGRATION



What is Continuous Integration ?

# CONTINUOUS INTEGRATION

- In software engineering, continuous integration (CI) implements **continuous processes** of applying quality control - small pieces of effort, applied frequently.
- Continuous integration aims to improve the quality of software, and to reduce the time taken to deliver it,
- This is done by replacing the traditional practice of applying quality control **after** completing all development.



# WHAT IS CI?

- **What it is**
  - a development **methodology**
  - of **daily** developer integrations
  - verified by **automated** builds

## →**What it is not**

- »Nightly builds
- »Building using IDEs/command line
- »Developer branches



**Continual Compilation != Continuous Integration**

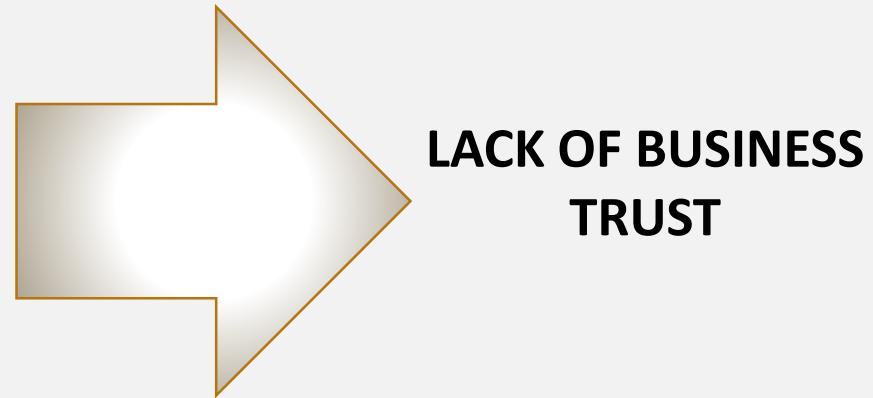
# DEVELOPMENT WITHOUT CI



LOTS OF BUGS!



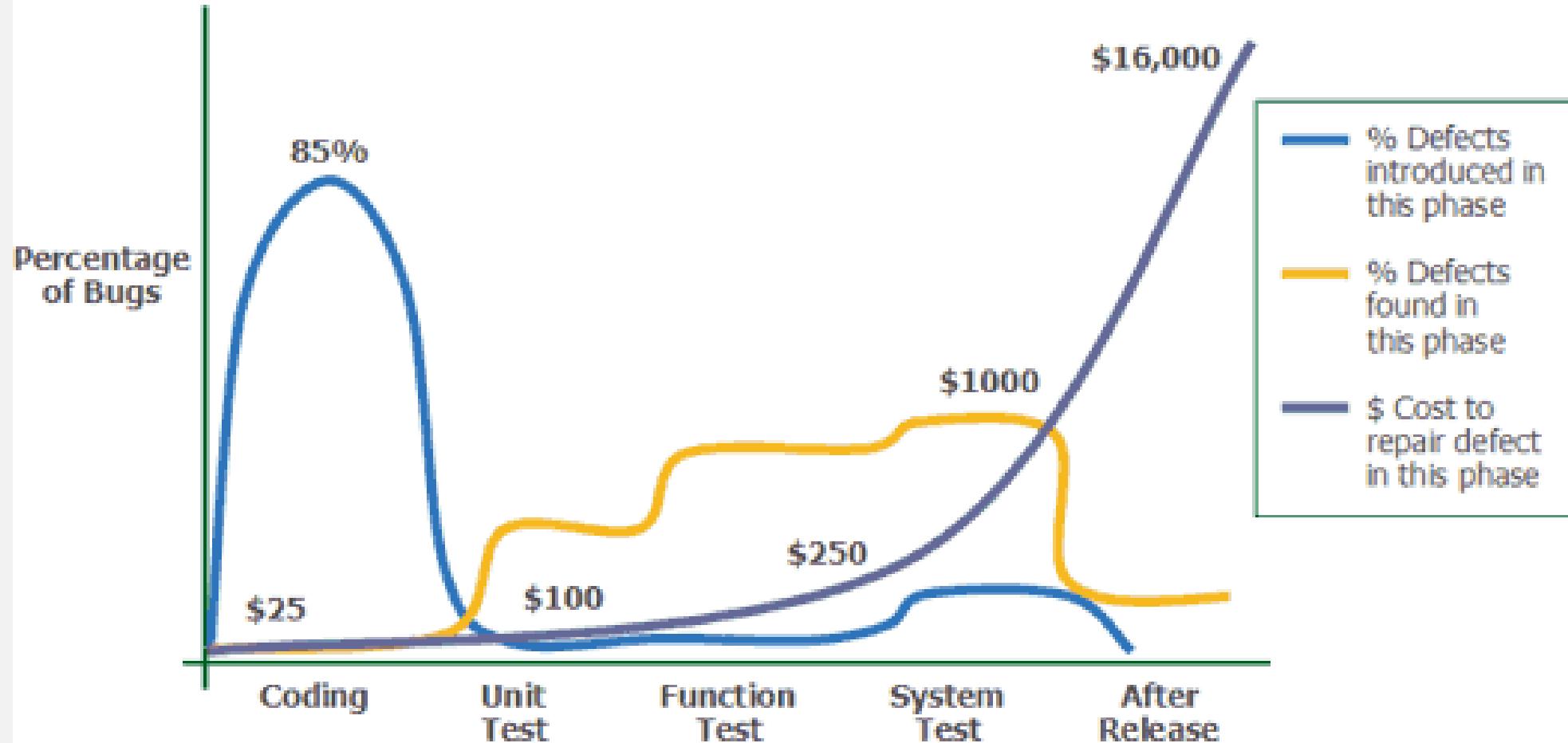
LATE TESTING



# Version X.Y.Z

Infrequent Releases

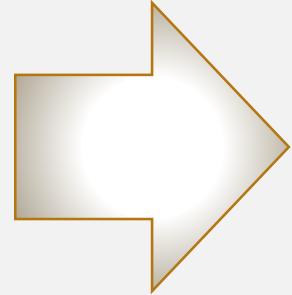
# DEVELOPMENT WITHOUT CI



Source: [http://www.agitar.com/solutions/why\\_unit\\_testing.html](http://www.agitar.com/solutions/why_unit_testing.html):

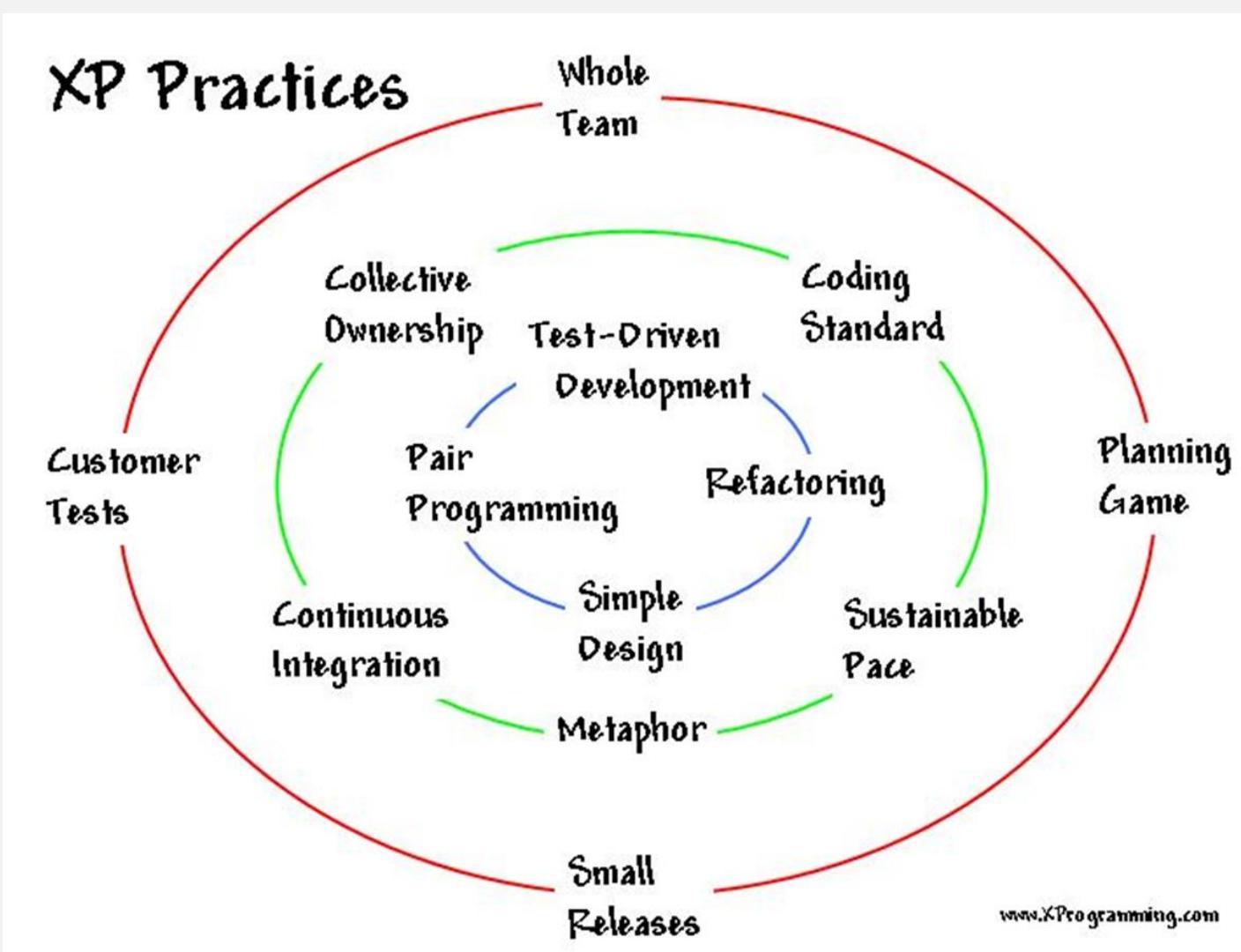
# DEVELOPMENT WITHOUT CI

- Poor Quality code
- Lack of team cohesion
- Harder to fix issues
- Poor Project Visibility
- Lack of deployable software
- High Maintenance Costs
- Project Delays

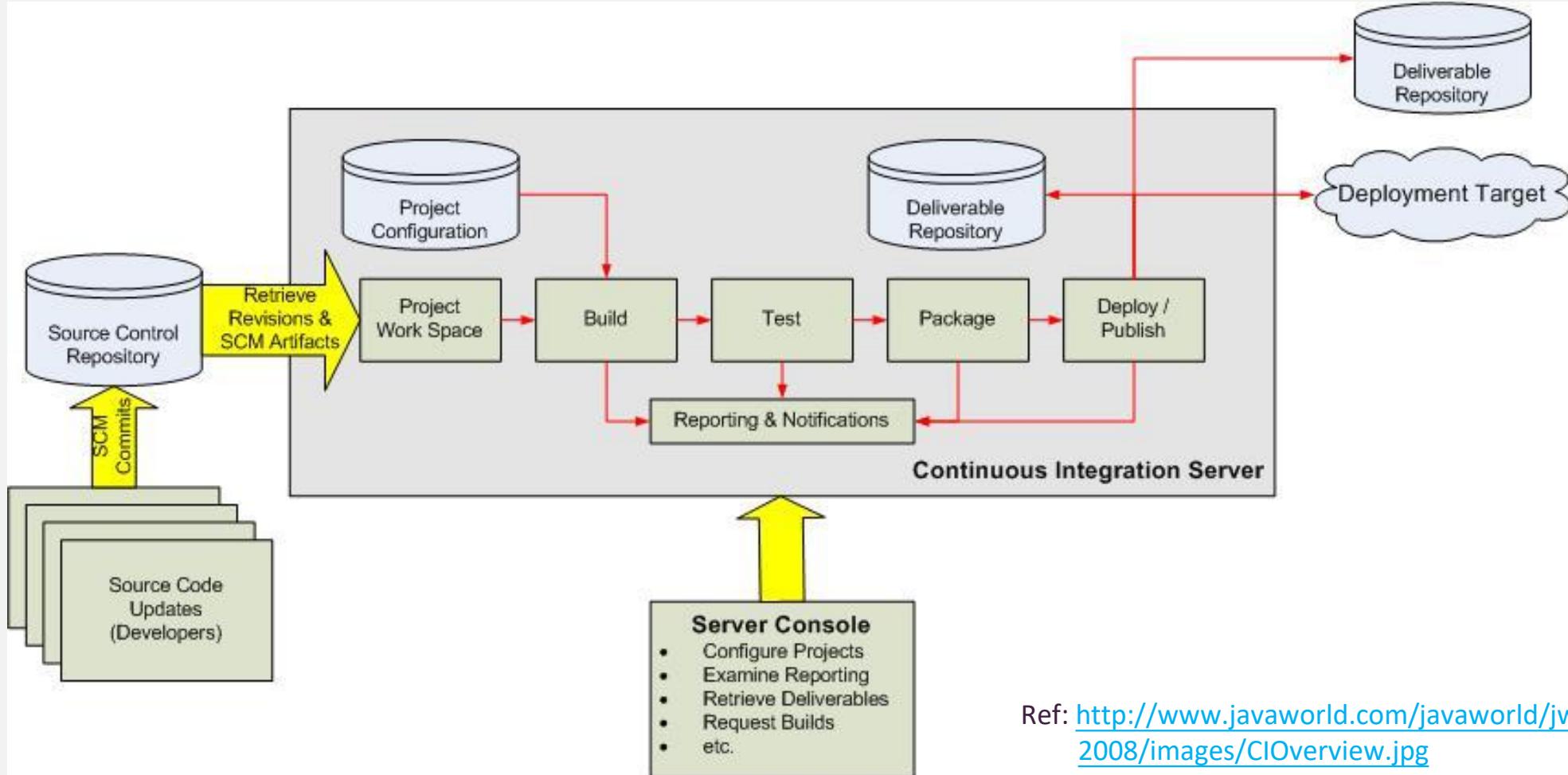


**Unhappy Customers!**

# CI – THE ORIGINS

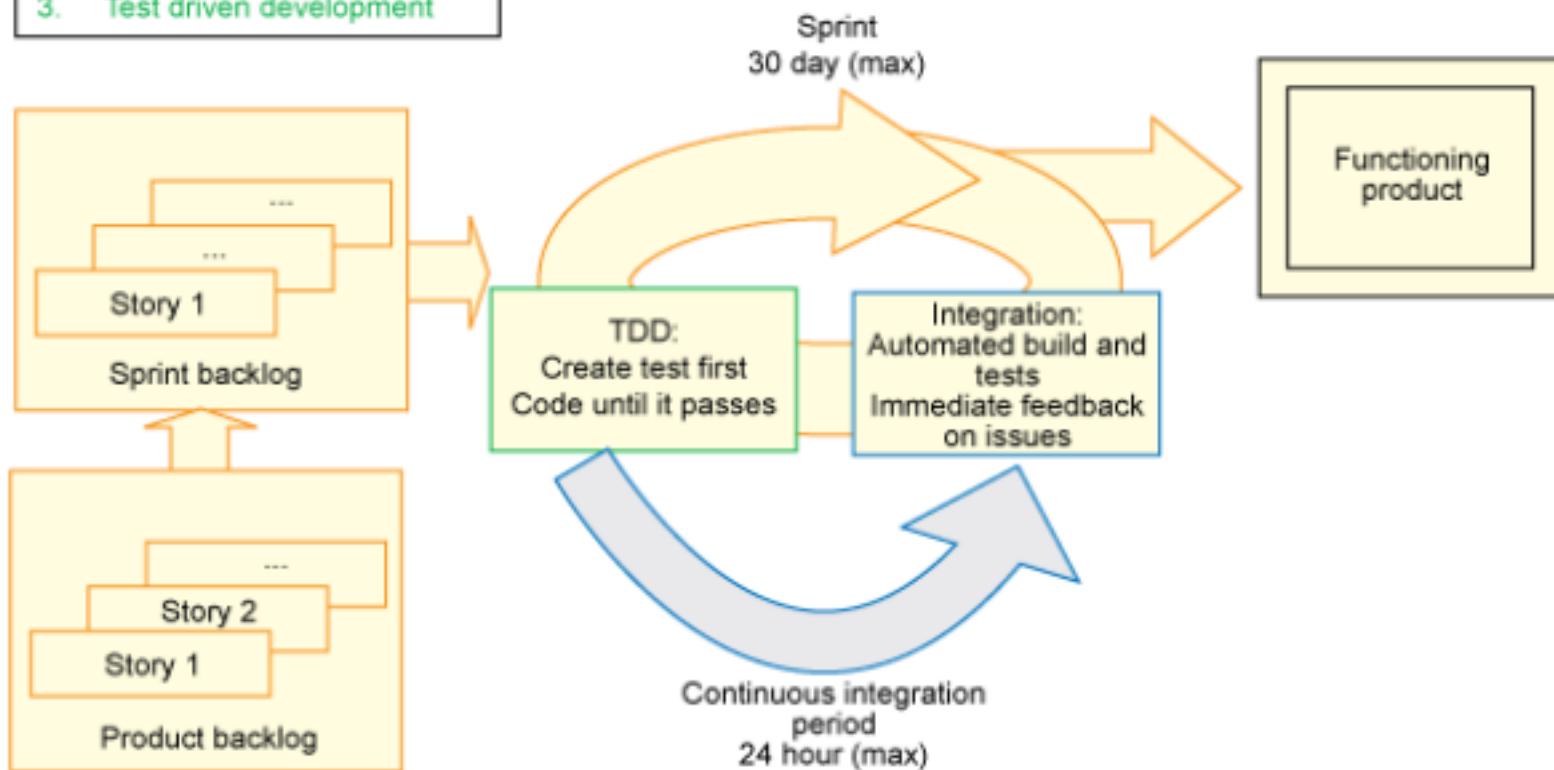


# CONTINUOUS INTEGRATION OVERVIEW



# A SAMPLE VIEW OF AGILE, CI AND TDD

Breaking down the process areas  
1. Agile  
2. Continuous integration  
3. Test driven development

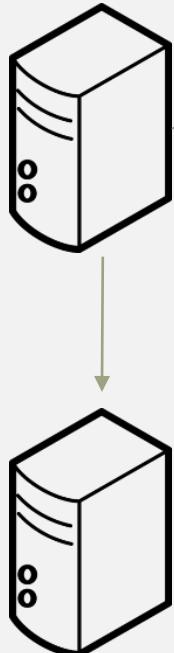


REF:

<http://www.ibm.com/developerworks/rational/library/continuous-integration-agile-development/continuous-integration-agile-development-pdf.pdf>

# DEVELOPMENT WITH CI

Dedicated Build Server



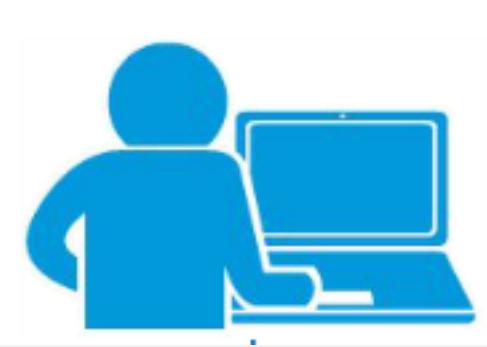
Test  
Servers

Automated Build  
Automated Tests  
Automated Code  
Metrics

Automated  
Releases



Code Repository



Regular Commits



Fewer Bugs

# DEVELOPMENT WITH CI

- Better
  - Build better quality software
  - That is **tested early and often**
  - And **adheres to best practices and coding standards**
- Faster
  - Tested in parallel, not in end
  - No integration points
  - Builds are part of daily routine
- Cheaper
  - Identify defects earlier
  - Fix when least costly
  - Repeatable testing

## CONTINUOUS INTEGRATION BENEFITS

- Project Management
  - Detect system development problems earlier
  - Reduce risks of cost, schedule, and budget
- Code Quality
  - Measurable and visible code quality
  - Continuous automatic regression unit test

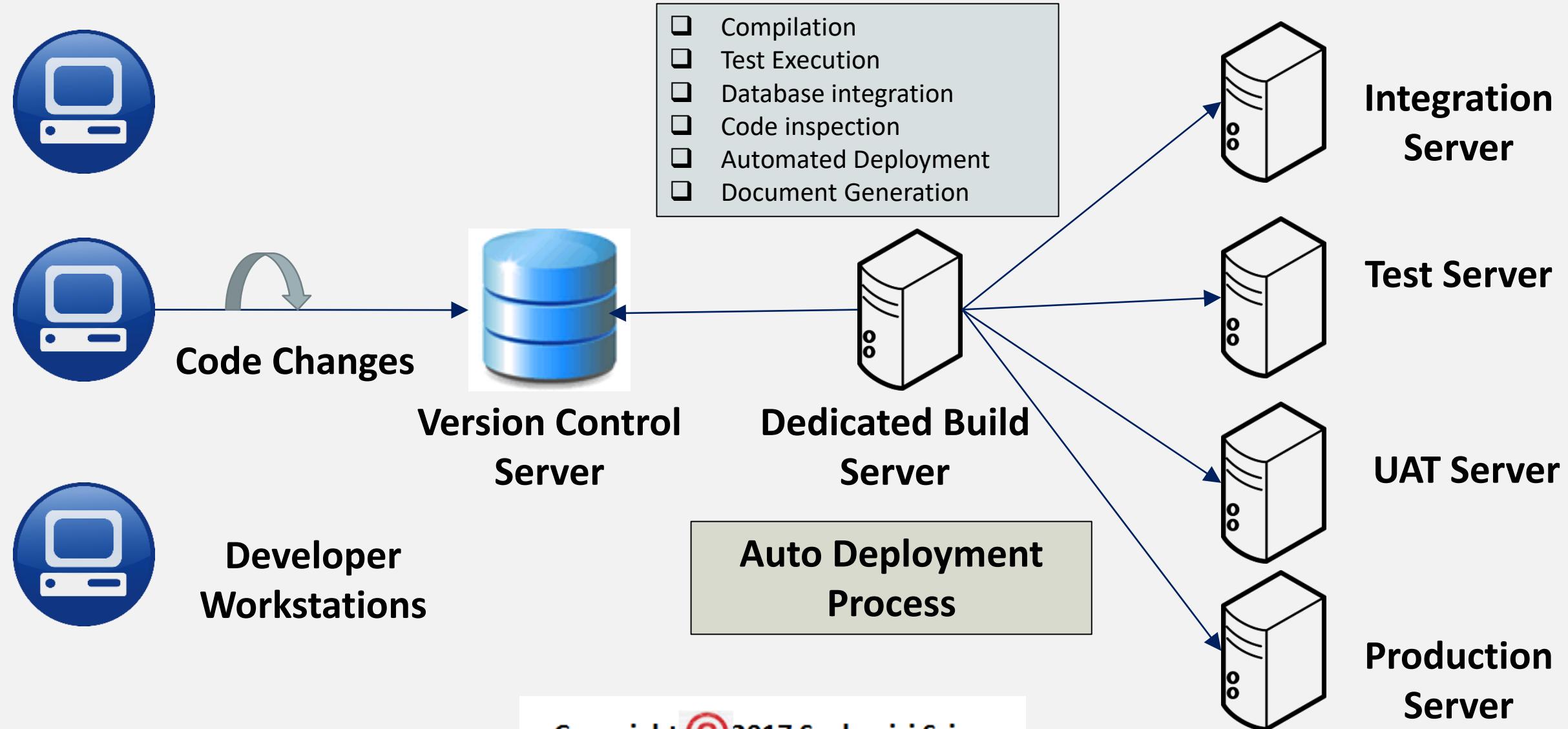
# Build Components



Compilation != Build

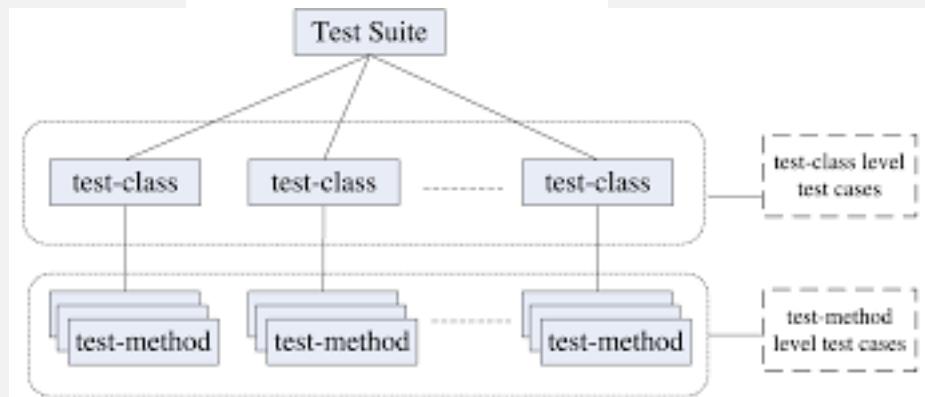
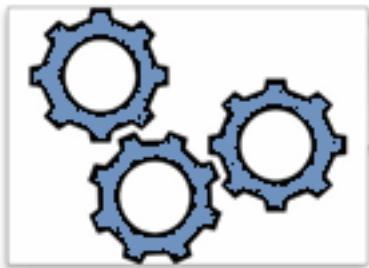
- Compilation
- Test Execution
- Database integration
- Code inspection
- Automated Deployment
- Document Generation

# CONTINUOUS INTEGRATION BASICS



# CI STARTER KIT

## Automated Build process



## Automated Test Suite



## Source Code Repository



## Continuous Build Server

# CI - IMPORTANT POINTS

- Build
  - Build scripts should be able to run from command line, not IDE
  - Should build consistently on all machines
  - Should build from source on all machines
- Deployment
  - Should be Automated
  - Should deploy to
    - Dev
    - Integration
    - Staging
    - Prod

## BEST PRACTICE OF CI

- Single Source Repository.
- Automate the Build and Test
- Everyone Commits Every Day
- Keep the Build Fast
- Everyone can see what's happening
- Automate Deployment (Optional)

# WHEN, HOW OF BUILDS

- When
  - At **every** check-in
  - Every time a **dependency** changes
- How
  - Use a **single** build script
  - That can run from the **cmdline**
  - Do **not** depend on an IDE
  - Continuously
    - Use a dedicated CI **server**, not cron
    - Trigger on **all** daily check-ins
  - **Not** [only] at midnight
  - Provide Continual feedbacks
  - Easily accessible builds
  - No or minimal developer effort

# THE GOLDEN CARDINAL RULES

- Commit Early, Commit often
- Never commit broken code
- Fix build failures immediately
- Fail Fast
- Act on Metrics
- Build in every target Environment
- Create Artifacts from every build



# Jenkins



# Hudson



# HUDSON

- Written in Java
- Runs inside servlet container e.g. Apache Tomcat
- Primarily written and developed by Kohsuke Kawaguchi
- Supports multiple SCM tools e.g. CVS, SVN, Git, Perforce, Clearcase
- Can execute multiple project types, including Ant and Maven
- Now released under Eclipse Public License
- Split into Hudson and Jenkins in 2011



# JENKINS

- Written in Java
- Runs inside servlet container e.g. Apache Tomcat
- Primarily written and developed by Kohsuke Kawaguchi
- Open Source
- Split from Hudson
- Large Number of Plugins
- Easy to install
  - WAR
  - Native Packages
- Easily configurable from browser



# WHY JENKINS

- Easy install, easy upgrade, easy configuration
- Distributed builds
- Monitoring external jobs
- No limit to the number of jobs, number of slave nodes
- Plugin architecture: Support for various version control systems, authentication methods, notification, workflow building, and many more features can be added.
- Jenkins provides API to its functionalities
- Can be scripted itself

## JENKINS SUPPORTED PLATFORMS



**and others!!!!**

# WHAT CAN YOU DO WITH JENKINS

- Get source code from repository
  - CVS (build-in)
  - SVN (build-in)
  - GIT (requires Git)
  - ClearCase (requires ClearCase)
  - Mercurial, PVCS, VSS, ...
- Trigger a build
  - Java
    - Maven (build-in), Ant, Gradle
  - .Net
    - MSBuild, PowerShell
  - Shell script
    - Python, Ruby, Groovy

# WHAT CAN YOU DO WITH JENKINS

- Automatically build and test
- Generate report
  - Static Code Analysis
    - Checkstyle, PMD, Findbugs, Compiler Warning
  - Test Report & Code Coverage
    - JUnit, TestNG, Cobertura, Clover
  - Open Tasks
- Notify
  - E-mail
  - Twitter
  - Jabber
  - IRC
  - RSS
  - Google calendar
  - ...

# WHAT CAN YOU DO WITH JENKINS

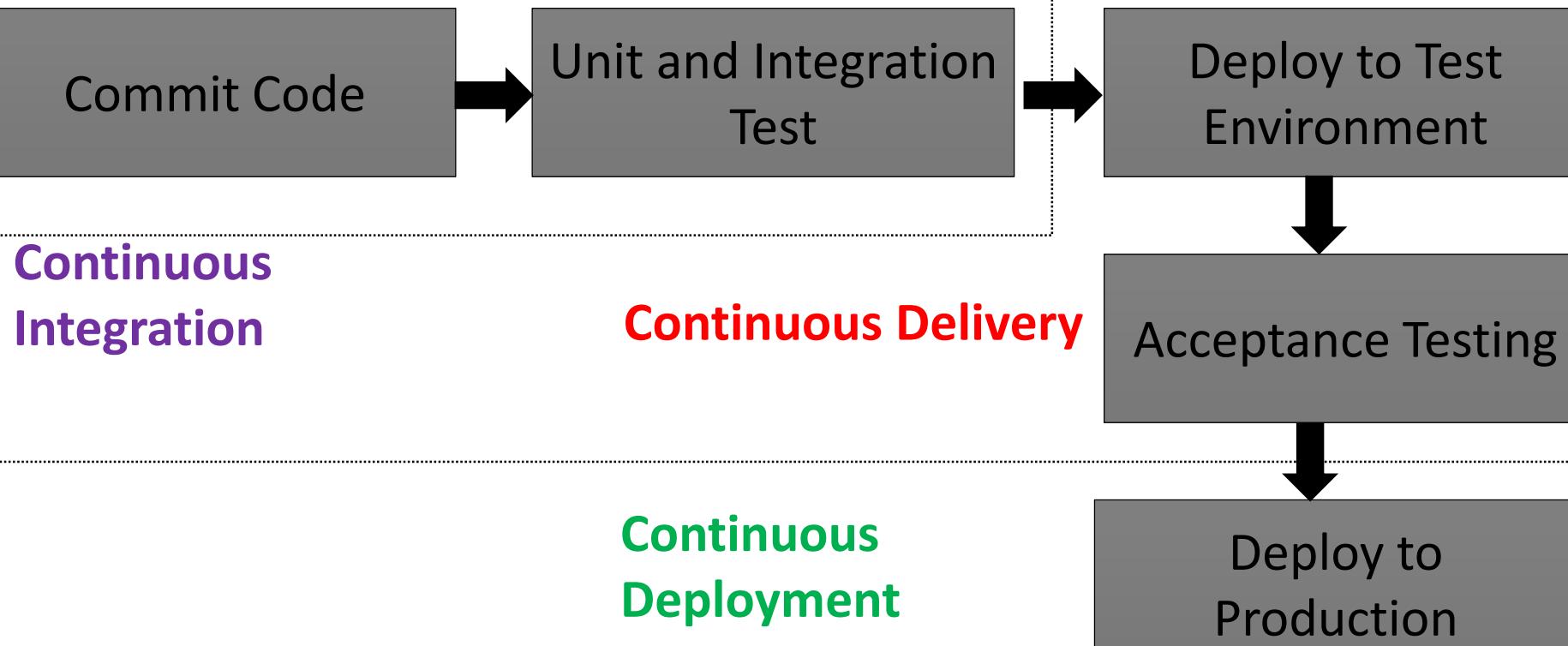
- Monitor executions of externally run jobs
- Distributed build
  - Master/Slave
- Deploy artifacts
  - Tomcat
  - JBoss
  - Glassfish
  - WebSphere
  - FTP
  - SSH

If you don't see a plugin for  
your task, write one yourself !  
**It is easy !**

## NON JAVA PROJECTS

- Yes, You can.
- Some supported non-Java project types include
  - ❑ .Net
  - ❑ Ruby
  - ❑ PHP
  - ❑ Drupal
  - ❑ Perl
  - ❑ C++
  - ❑ Node.js
  - ❑ Python
  - ❑ Android
  - ❑ Scala

# SIMPLIFIED CI/CD PIPELINE



# BENEFITS

- Accelerated Time to Market:
  - Deliver the business value to customers more quickly.
  - Stay a step ahead of the competition
- Building the Right Product:
  - Frequent releases let the application development teams obtain user feedback more quickly.
  - Work on only the useful features.
- Improved Productivity and Efficiency:
  - Significant time savings for developers, testers, operations engineers, etc. through automation.
- Reliable Releases:
  - Release process is more reliable.
  - Deployment process and scripts are tested repeatedly before deployment to production
  - Number of code changes in each release decreases.
    - Finding and fixing any problems is easier
- Improved Product Quality
- Improved Customer Satisfaction: A higher level of customer satisfaction is achieved.



## BUT WHY?

- Deploy Software Manually
  - Tense teams
  - Late nights
  - Roll backs
  - Re-verification
  - Incorrect Documentation
- Deploy to production like environment only when all development is complete.
  - More chances of things going wrong during final integration testing
  - Much of documentation produced for installation is incorrect anyways.
- And other anti patterns.....

# CONTINUOUS DELIVERY VS DEPLOYMENT

- Continuous deployment is continually pushing to production. This is not always a right thing to do
  - User Training
  - Marketing of products
  - Compatibility/3<sup>rd</sup> party integrations...
- Continuous Delivery on other hand is to keep all artifacts ready and deploy manually/automatically with scripts.
  - Variations: Automatically deploy to test/QA and manually deploy to Production via clicking buttons
  - Manual in this context is to manually invoke scripts, not manually deploying by hand.

# CONTINUOUS DELIVERY

- Simplest form is scripting of deployment to servers
  - Usually does not handle: Software installations or reboots
  - In such cases, use Chef/Puppet/Vagrant.....

## Scripting Process

**Database Updates**

- Liquibase – XML Based changes
- Part of SCM itself!!!
- Can be run as
  - Script
  - Ant
  - Maven

**Roll Back Changes**

- Not just Liquibase DB changes.
- Liquibase cannot handle all changes
- Take a snapshot and revert back

**Smoke Tests**

## AGENDA FOR NEXT MODULE

- REVISIT CI/CD
- Preparing your Jenkins Environment
- Installing Jenkins
- Start up Jenkins
- **SSH INTRO**
- Set up Java, Maven and Ant with Jenkins
- Create a first Jenkins job
- Building Java docs with Jenkins



Q&A

# Questions???

## FOOD FOR THOUGHT!

- in CI, all components should be written in same language. True or false?
- Can Database integration be skipped in the CI Process?
- If you had to choose Hudson over Jenkins, what reasons would you give your manager?

## MODULE 02

# GETTING STARTED WITH JENKINS



## PLANNING FOR JENKINS

- Decide on what OS your CI build servers will be.
- You can change it later on – but all CI build server should ideally be on same platform
- Document your CI servers and security constraints
- Document existing Build tools used in the organization.
- **Above all, clearly define your build and deploy process.**
- These are not cast in stone. If they do not work for you, you should actively look at process improvements.

# JENKINS – INVOLVE ALL!



Software  
Developers



Project Managers



System Admins



Quality  
Consultants

# INSTALLATION PRE-NOTES

- Ensure that Java 7 or 8 are setup.
  - Ensure system variable JAVA\_HOME is set and present on path.
- Create an Environment variable JENKINS\_HOME.
  - This is the place where all Jenkins works (including workspaces are stored).
  - In production environment this should be on a directory with reasonably large amount of space.
  - This will also be the place to install JENKINS too.

# INSTALLING JENKINS

- Can be installed either as
  - Web Archiver (WAR)
  - Native Package
- Two versions
  - Release
  - Long Term Support (LTS) Release
- LTS versions are some point releases away from “latest and greatest” versions.
- They are more stable
- Always use LTS versions for production environments.

# INSTALLING JENKINS

The screenshot shows the Jenkins website at <https://jenkins.io>. The top navigation bar includes links for Most Visited, Virtualbox Pxe Enabled..., Setting up a Caching ..., YAMLint - The YAML ..., Designing Puppet – Ro..., Simple Puppet Modul..., and Moon. The main menu has sections for Jenkins, Downloads (selected), Blog, Documentation, Plugins, Use-cases, Participate, Sub-projects, Resources, and Search.

**LTS Release**

LTS (Long-Term Support) releases are chosen every 12 weeks from the stream of regular releases as the stable release for that time period.

[1.651.2 .war](#)

[Changelog | Past Releases](#)

**Weekly Release**

A new release is produced weekly to deliver bug fixes and features to users and plugin developers.

[2.8 .war](#)

[Changelog | Past Releases](#)



[Download Jenkins](#)

# INSTALLATION AS NATIVE PACKAGE

- The default is to install c:\program files (x86)\Jenkins
  - CHANGE THIS to use JENKINS\_HOME and ensure that JENKINS\_HOME does not contain spaces.
  - This is true for **all** native packagers
- The native package comes bundled with a JRE which is used.
- All configuration is stored in file Jenkins.xml.
  - Internally, you can see it invoke java on **Jenkins.war** on port 8080
  - The engine used is winstone
  - Log files are
    - Jenkins.out.log
    - Jenkins.err.log
    - Jenkins.wrapper.log

## NATIVE PACKAGE – CAN I RUN AS A WAR FILE?

- Yes. All Jenkins.exe does is to invoke java on Jenkins.war.
- Create a batch file that contains this:
  - `java -Xrs -Xmx1024m -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -jar jenkins.war" --httpPort=8080`
- That will work too! However
  - You lose benefits of running as a service with auto start/stop from service control panel.
  - Ugly – one closable terminal will continue to be displayed (which can be hidden though)

# INSTALLING AS PACKAGE



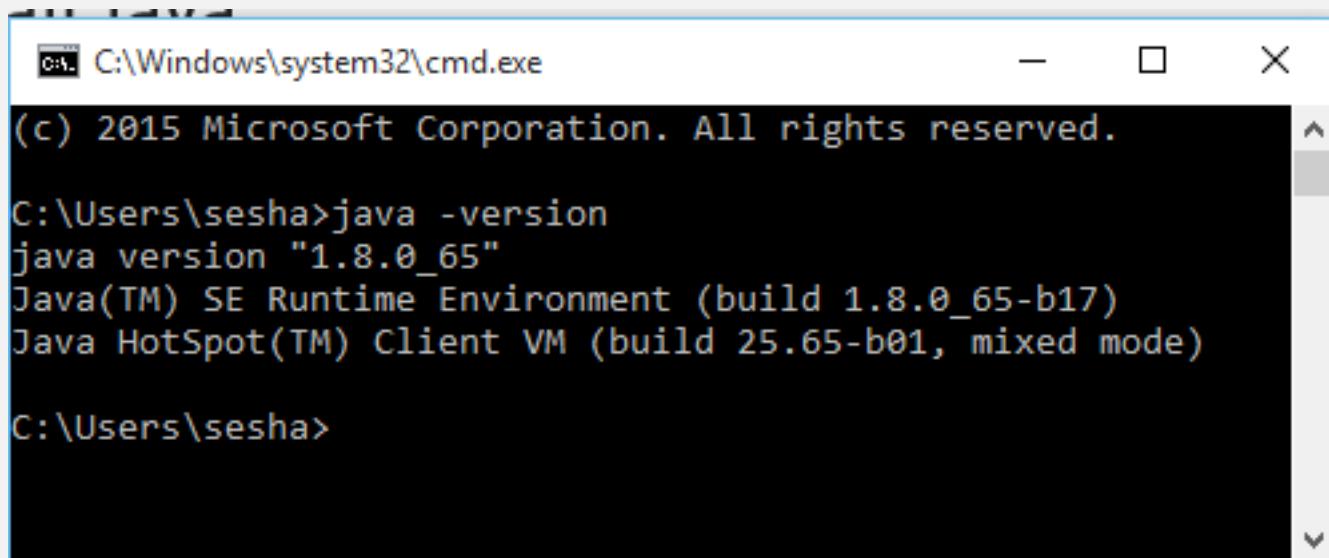
**Apache Tomcat 6+**  
**(We recommend 7 or 8)**



**Oracle JDK 7/8**

# INSTALL JAVA

- Download Java from java.com.
- Double click on installer to start.
- Verify correctness of install by typing below from a command line
  - `java -version`



A screenshot of a Windows Command Prompt window titled "cmd.exe". The window shows the following text:  
(c) 2015 Microsoft Corporation. All rights reserved.  
C:\Users\seshas>java -version  
java version "1.8.0\_65"  
Java(TM) SE Runtime Environment (build 1.8.0\_65-b17)  
Java HotSpot(TM) Client VM (build 25.65-b01, mixed mode)  
C:\Users\seshas>



- On windows, it is always recommended to install to a directory that has no spaces in its name.
- `JAVA_HOME` must be set and `JAVA` should be in `PATH`

# INSTALLING JAVA

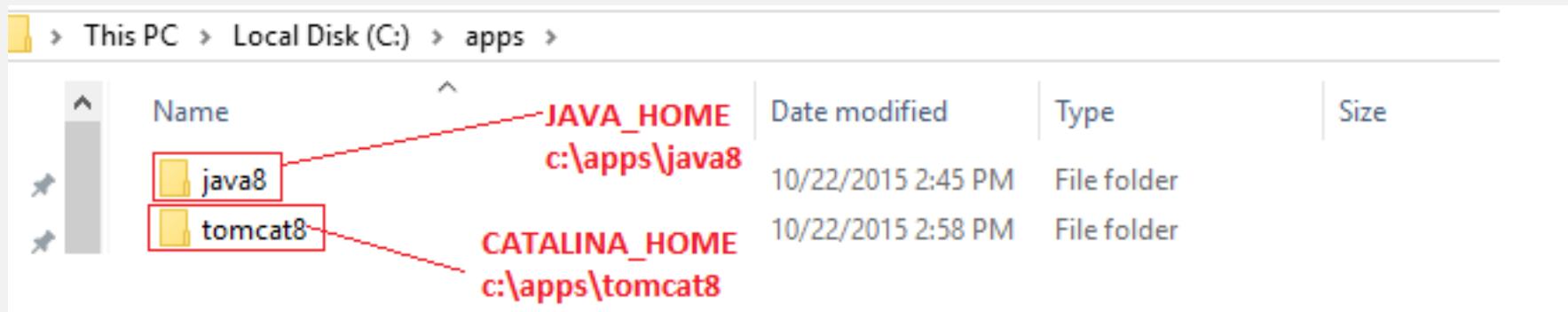
- Choose Correct installer. (JDK or JRE)
- If choosing JDK, ensure Public JRE is also installed and that the JRE and JDK are installed separately.
- JDK 8: 1.8.0\_66+ is preferred.

Name	Date modified	Type	Size
jdk7	11/25/2015 10:45 ...	File folder	
jdk8	11/25/2015 4:52 PM	File folder	
jre7	11/25/2015 10:45 ...	File folder	
jre8	11/25/2015 4:52 PM	File folder	

# INSTALL TOMCAT

- There are multiple ways to install Tomcat.
  - Binary installers
  - Zip installers
  - Install Tomcat as a service
- Refer here: <https://tomcat.apache.org/tomcat-8.0-doc/setup.html#Windows> for setup of Tomcat 8 on Windows
- At end of install of Java and tomcat, the following system variables should be set to point to correct location
  - JAVA\_HOME: the location where java is installed
  - CATALINA\_HOME: The location where TOMCAT is installed.

# INSTALL TOMCAT



## INSTALL AS A WEB WAR

- Download WAR file to %CATALINA\_HOME%\webapps directory folder. (if using Tomcat)
- Tomcat should automatically expand Jenkins.war to Jenkins folder and auto-deploy the application.
- Navigate to <http://localhost:8080/jenkins>

## INSTALL AS A WEB WAR



If not using Tomcat or other container, the application can be run using the following command:

```
java –jar jenkins.war --httpPort=8080
```

## INSTALLATION NOTES

- Installing as a Win installer creates context under root i.e. <http://localhost:8080>, while installing as war creates Jenkins context as <http://localhost:8080/jenkins>
- Configuration under installer can be found under jenkins.xml while in war file, it is standard server.xml/web.xml file.
- Typically in production, build servers are usually on UNIX and UNIX variants mainly because of
  - Better SSH/SSL Support
  - Better support for upgrade to latest versions of Jenkins as required.
- In all cases, default port is 8080 which can be changed.

# CHANGE DEFAULT BINDING FOR JENKINS WAR

- it is a preferred, not mandatory practice.
- On tomcat, there are multiple ways to do this.
- **Method 1:**
  - Unzip the WAR file for Jenkins directly under ROOT folder
- **Method 2:**
  - Change the context for default servlet and change context for Jenkins to /

# INSTALL MAVEN

- Maven is a build tool.
- To install,
  - Download the ZIP/GZ file
  - Unzip the same
    - unzip apache-maven-3.3.3-bin.zip (Windows)
    - tar xzvf apache-maven-3.3.3-bin.tar.gz (Linux)
  - Set MAVEN\_HOME environmental variable to where Maven has been extracted.
  - Add \$M2\_HOME/bin (%M2\_HOME%\bin) to the PATH variable.



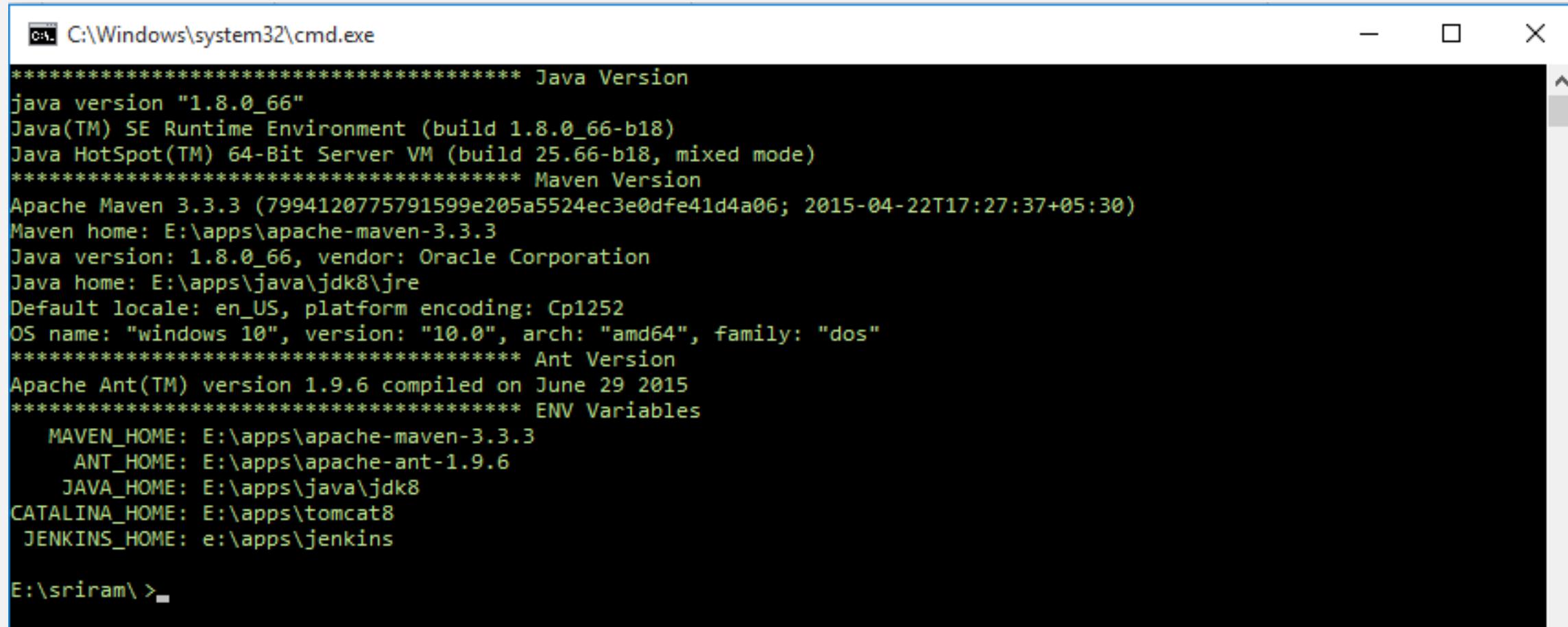
# INSTALL ANT

- ANT is a build tool.
- To install,
  - Download the ZIP/GZ file
  - Unzip the same
    - unzip apache-ant-1.9.6-bin.zip (Windows)
    - tar xzvf apache-ant-1.9.6-bin.tar.gz (Linux)
  - Set ANT\_HOME environmental variable to where Ant has been extracted.
  - Add \$ANT\_HOME/bin (%ANT\_HOME%\bin) to the PATH variable.



# VERIFICATION

- If all went fine, you can verify the correct installation path (Custom script) as below:



```
C:\Windows\system32\cmd.exe
*****
Java Version
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b18, mixed mode)
*****
Maven Version
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30)
Maven home: E:\apps\apache-maven-3.3.3
Java version: 1.8.0_66, vendor: Oracle Corporation
Java home: E:\apps\java\jdk8\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"
*****
Ant Version
Apache Ant(TM) version 1.9.6 compiled on June 29 2015
*****
ENV Variables
MAVEN_HOME: E:\apps\apache-maven-3.3.3
ANT_HOME: E:\apps\apache-ant-1.9.6
JAVA_HOME: E:\apps\java\jdk8
CATALINA_HOME: E:\apps\tomcat8
JENKINS_HOME: e:\apps\jenkins

E:\sriram\ >
```

# INSTALLATION ON UBUNTU

- Install OpenJDK 7
- Install Tomcat 7
- Download WAR file to \$CATALINA\_HOME/webapps
- Restart tomcat 7

**OR SIMPLY INSTALL NATIVE IMAGES**

# INSTALLATION ON UBUNTU

- **Require**



**Open JDK 8 /Open JRE 8**



**Prefer not to use GNU Compiler  
for Java**

# INSTALLATION ON UBUNTU

- To install, you first need to install Jenkins to the list of package source list.
- This is done by first adding the key ring and source to package list

```
 wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -
 sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list.d/jenkins.list'
```

- Next we need to update apt to refresh with the changes

```
 sudo apt-get update
```

- Finally, we install Jenkins

```
 sudo apt-get install jenkins
```

# INSTALLATION ON UBUNTU

- **Updation**
  - Just perform an apt-get update and installation
- **Removal**
  - Just perform an apt-remove (or) apt-purge followed by apt-get autoremove

# INSTALLATION ON UBUNTU

- Jenkins will start on auto-startup.
  - See /etc/init.d/Jenkins for additional details.
- User jenkins is created to run this service
- Log file can be found here: /var/log/jenkins/jenkins.log
- Configuration file can be found here:
  - /etc/default/jenkins
- Default port listening is always 8080. Edit above file (line than contains HTTP\_PORT) to change the number
- You could install either Apache, NGINX or simply edit iptables to redirect traffic on port 80 to 8080 or use Apache/NGINX as a proxy.

# DEMO & LAB

Q&A

# Questions???

## AGENDA FOR NEXT MODULE

- Introduction to plugins
- Adding Plugins to Jenkins
- Managing Plugins
- Upgrading Plugins
- Plugins in Action
- Commonly used plugins
  - Git Plugin
  - Parameter Plugin
  - HTML Publisher
  - Copy Artifact
  - extended choice parameters



## FOOD FOR THOUGHT !!!

- Why do you think Jenkins provides 2 ways to install itself and provide multiple ways to run itself?
- If you were following us closely, by default, Jenkins allows anyone full access. Why do you think this is?

## **MODULE 03**

### **AN OVERVIEW OF PLUGINS**



## JENKINS - PLUGINS

- The Jenkins plugin architecture aims
  - To Handle **different types of activities** (e.g. compilation, testing, etc..)
  - With **different technologies** and
  - **Different sources and targets**

# JENKINS – JOBS AND PLUGINS

- Jenkins does it work using core and additional plugins.
- The default plugins include

Plugin	Description
ANT	Used for running ant tasks from supplied build.xml and build.properties files
Credentials	Allows Storing of credentials in Jenkins
CVS	Integrates Jenkins with CVS version control system using a modified version of the Netbeans cvsclient.
External Job Monitor	Adds the ability to monitor the result of externally executed jobs
Javadoc	Publishes Javadoc
Junit	Publishes Junit Test Results (Not run Junit test cases)

# JENKINS – JOBS AND PLUGINS

- Default plugins list (contd)

Plugin	Description
LDAP	LDAP Authentication Support
Mailer	Supports Email sending. Not usually used as is but with additional plugins.
SSH Credentials	Supports saving and using of SSH credentials.
Subversion	Supports SVN as a SCM Tool
Maven integration	This plug-in provides deep integration of Jenkins and Maven. <b>This functionality used to be part of the Jenkins core.</b> Now it is a plug-in that is installed by default, but can be disabled.
PAM Authentication	Adds Unix Pluggable Authentication Module (PAM) support to Jenkins

# JENKINS – JOBS AND PLUGINS

- Other Built in plugins include:
  - Matrix Authorization Strategy Plugin
  - Matrix Project Plugin
  - OWASP Markup Formatter Plugin
  - Script Security Plugin
  - SSH Slaves Plugin
  - Windows Slaves Plugin
  - Translation Assistance

# JENKINS – JOBS AND PLUGINS

Compile



Reports



Notification

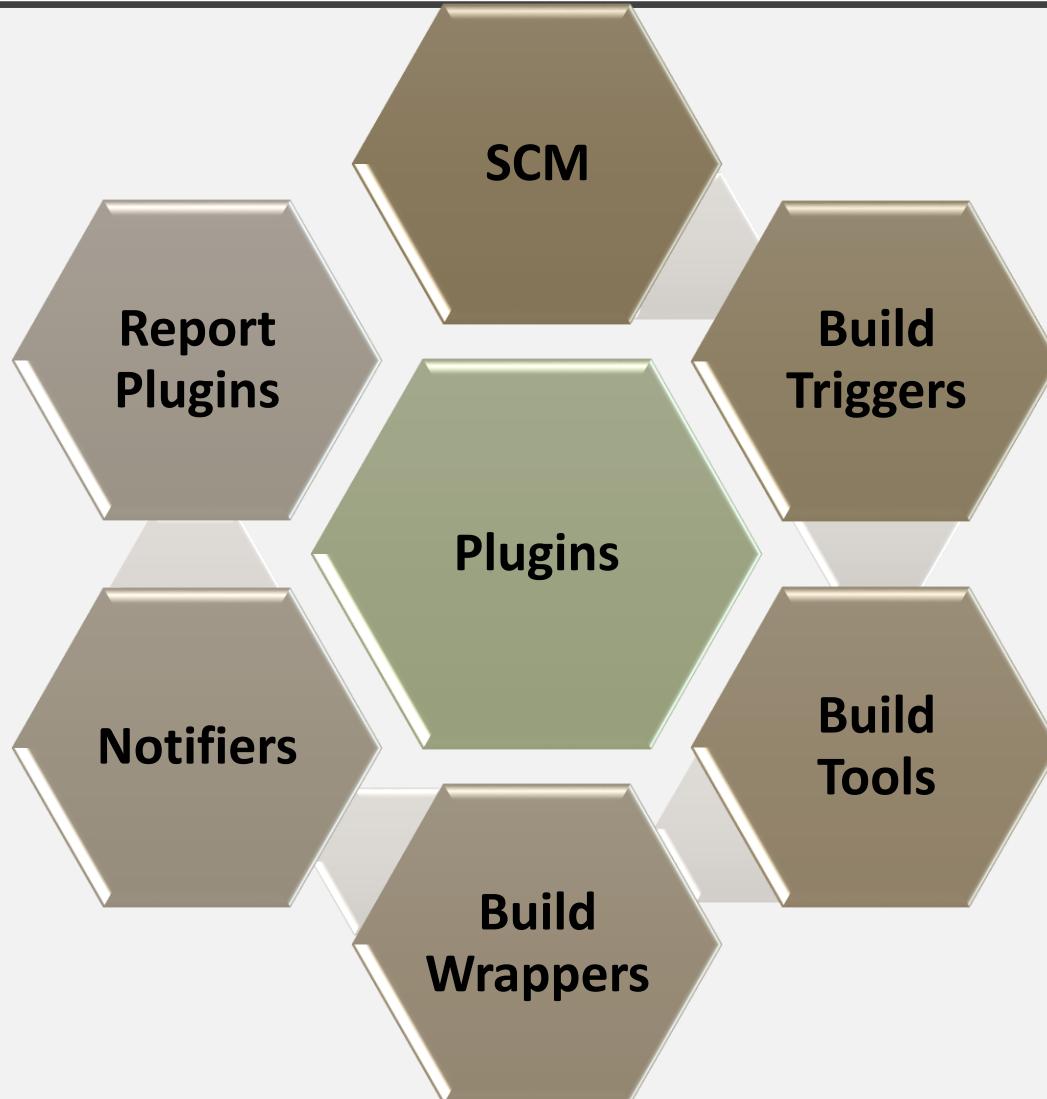


Testing



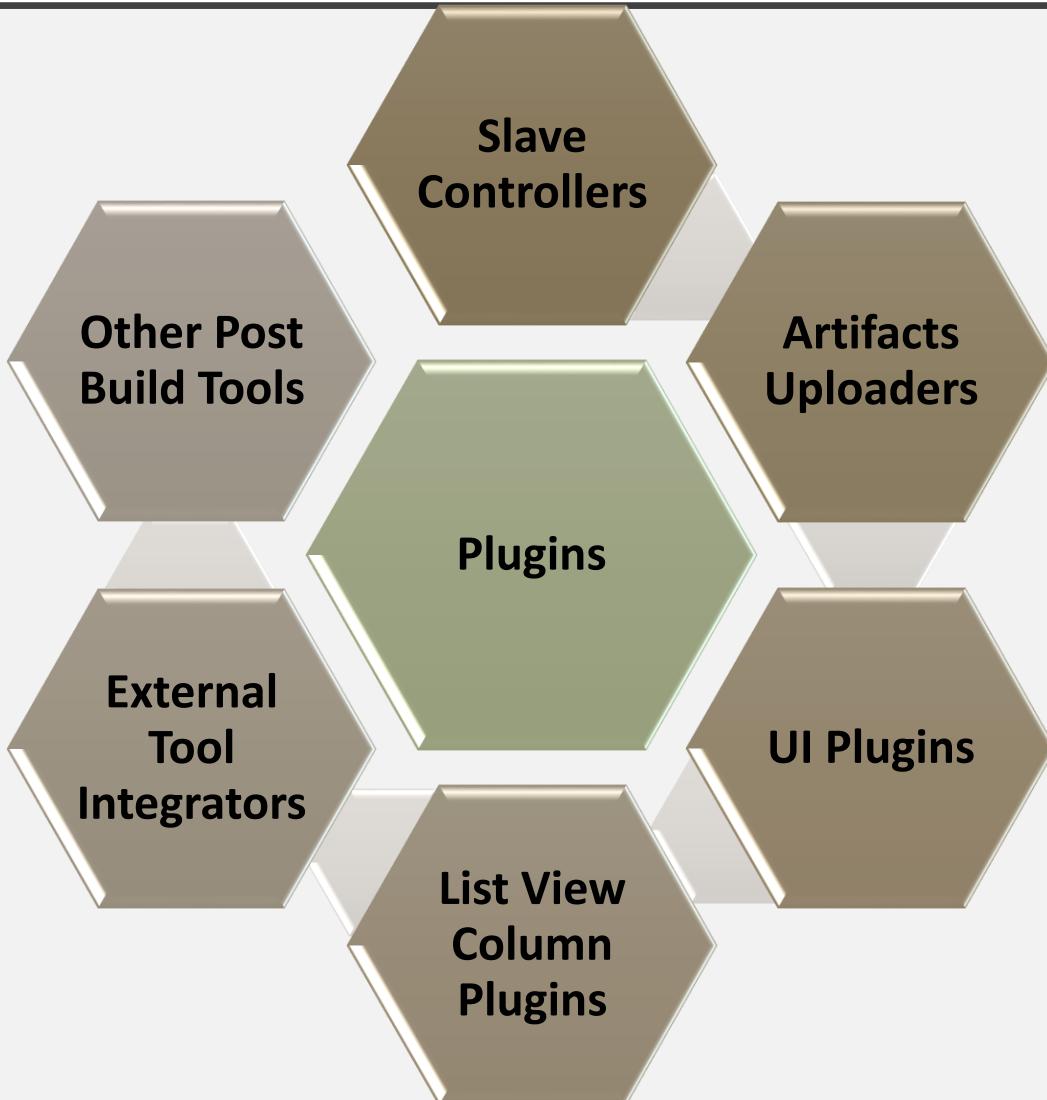
Deployments

# JENKINS - PLUGINS



REF: <https://wiki.jenkins-ci.org/display/JENKINS/Plugins>

# JENKINS - PLUGINS



REF: <https://wiki.jenkins-ci.org/display/JENKINS/Plugins>

# JENKINS - PLUGINS



On installation of Jenkins for the first time, especially if not using LTS version (Pre Jenkins 2), do the following as one of the 1<sup>st</sup> things:

- Navigate to Manager Jenkins->Manage Plugins
- Click on Update Tab
- Select all and choose to update.

This will update the core plugins shipped with Jenkins as default.

# JENKINS – SCM PLUGINS

- Accurev
- Bazaar
- BitKeeper
- ClearCase
- Darcs
- Dimensions
- Git
- Harvest
- MKS Integrity
- PVCS
- StarTeam
- Subversion
- Team Foundation Server
- Visual SourceSafe

# JENKINS – BUILD TOOLS

- Ant
- Maven
- MSBuild
- Cmake
- Gradle
- Grails
- Scons
- Groovy



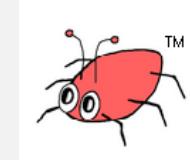
# JENKINS – TEST FRAMEWORK

- Junit
- Nunit
- MSTest
- Selenium
- Fitnesse



# JENKINS – CODE QUALITY ANALYZERS

- Checkstyle
- CodeScanner
- DRY
- Crap4j
- Findbugs
- PMD
- Fortify
- Sonar
- FXCop
- Emma
- Cobertura
- Clover
- GCC/GCOV



# JENKINS – PLUGIN INTERFACE

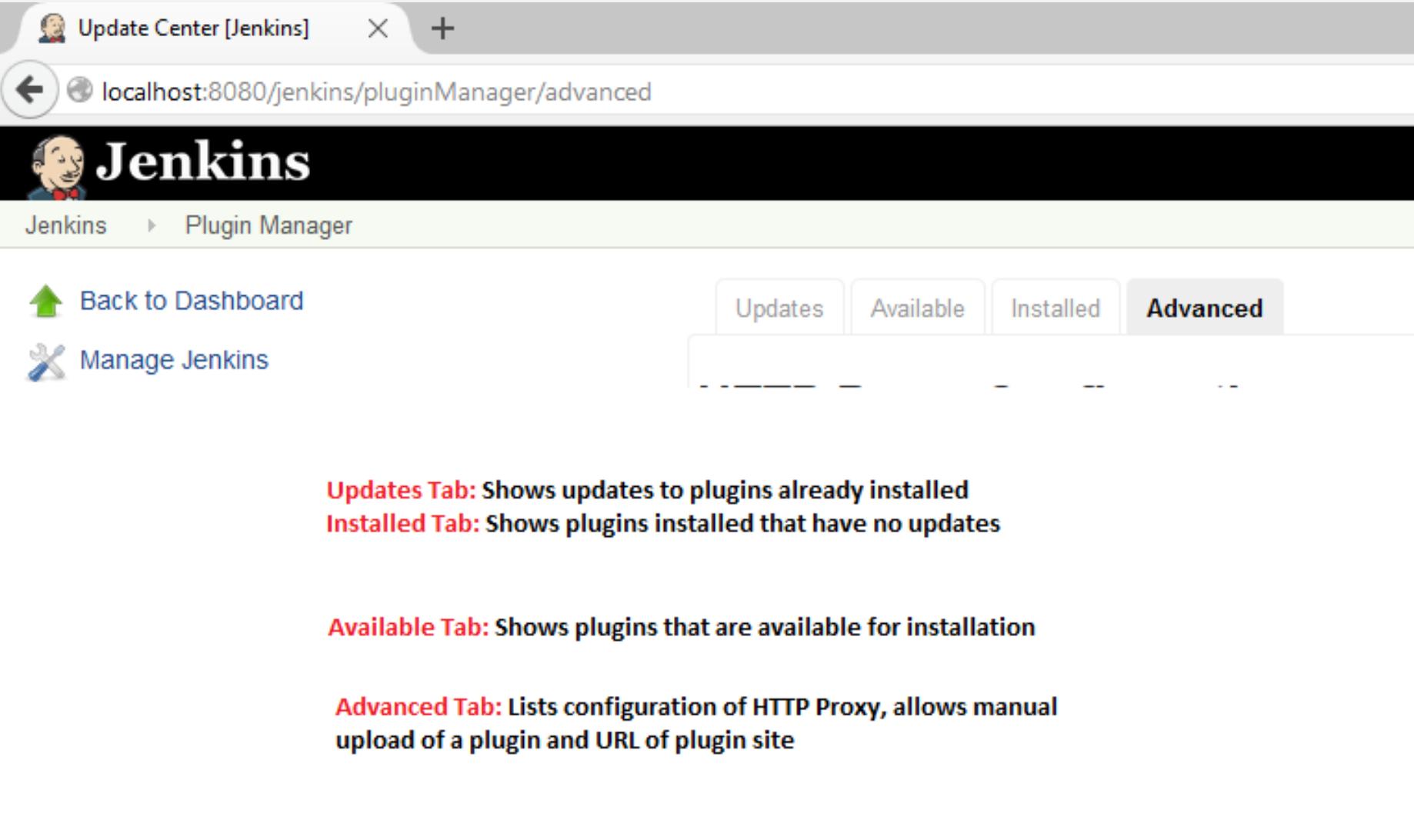
The Jenkins Plug In Interface Starting Point

**Manage Jenkins**

- [Configure System](#)  
Configure global settings and paths.
- [Configure Global Security](#)  
Secure Jenkins; define who is allowed to access/use the system.
- [Reload Configuration from Disk](#)  
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config.
- [Manage Plugins](#)  
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

On installation of Jenkins for the first time, there is no security. Before exposing Jenkins to external world, secure it so that unauthorized users cannot add/remove plugins.

# JENKINS - PLUGINS



The screenshot shows the Jenkins Plugin Manager interface. At the top, there's a header bar with the Jenkins logo and the text "Update Center [Jenkins]". Below it is a browser-style address bar with the URL "localhost:8080/jenkins/pluginManager/advanced". The main title is "Jenkins" with a subtitle "Plugin Manager". A navigation bar below the title includes "Back to Dashboard" and "Manage Jenkins". On the right, there are four tabs: "Updates", "Available", "Installed", and "Advanced", with "Advanced" being the active tab. Below the tabs, there are several dashed horizontal lines. To the left of these lines, there is descriptive text for each tab:

- Updates Tab:** Shows updates to plugins already installed
- Installed Tab:** Shows plugins installed that have no updates
- Available Tab:** Shows plugins that are available for installation
- Advanced Tab:** Lists configuration of HTTP Proxy, allows manual upload of a plugin and URL of plugin site

## JENKINS - PLUGINS

- Plugins that are bundled with Jenkins can be pinned.
- When Jenkins is upgraded, bundled plugins are also usually updated.
- If however, a bundled plugin, is manually updated, it gets pinned to the current version of Jenkins
  - Update of Jenkins will not update this plugin
  - Tech Aside: File **\$JENKINS\_HOME/plugins/plugin\_name.jpi.pinned** is created in case of pinned plugin
  - Unpinning is the fancy name of removing the above file and allowing for updates to happen.

# COMMON PLUGINS

- **ANT**
- **Maven**
- **JavaDoc**
- HTML Publisher
- GIT
- **Credentials**
- Parameter (and Extended Parameter/choice)
- Artifact



## Legend:

Built in plugins are highlighted in **blue, bold**.  
All other plugins need to be manually installed.

## JENKINS – ANT PLUGIN

- Supports Ant Builds for applications.
- Part of core Build tools plugin set.
- If no target specified, it will default to default target specified in build.xml (unless another build file is specified)

# JENKINS – ANT PLUGIN

**Invoke Ant**

Ant Version	Default	?
Targets		▼ ?
Build File		▼ ?
Properties		▼ ?
Java Options		▼ ?

**Delete**

Add build step ▾

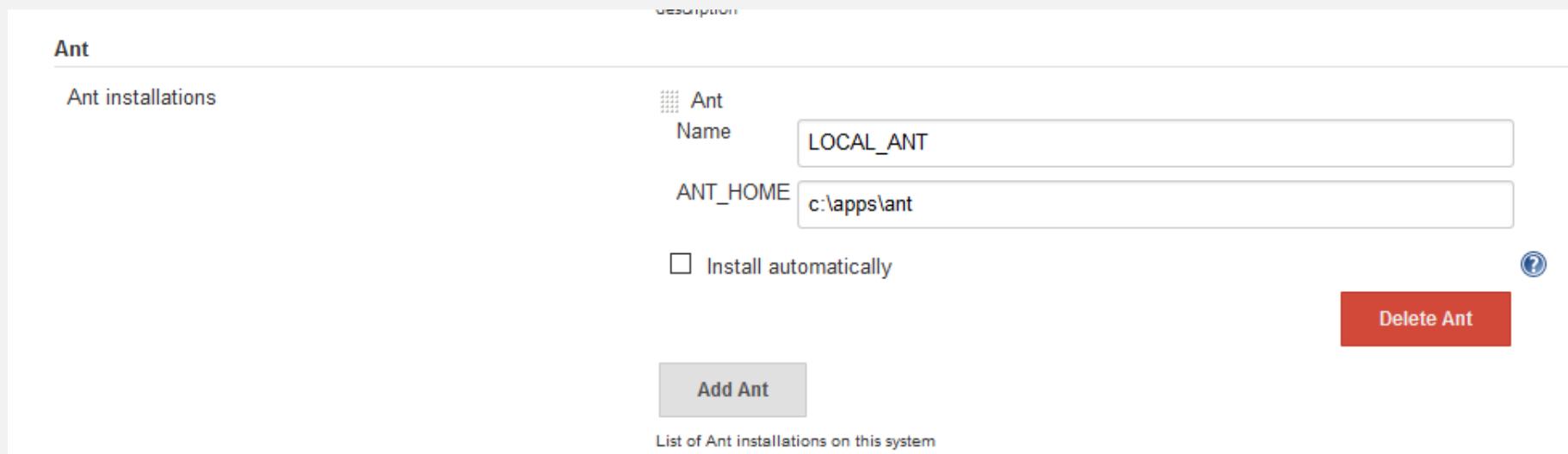


# JENKINS – ANT PLUGIN

- 2 ways to configure the same.
- **Method 1:**
  - Specify full path to where ANT is located (listed by ANT\_HOME environment variable)
  - Assumes that ANT has already been installed.

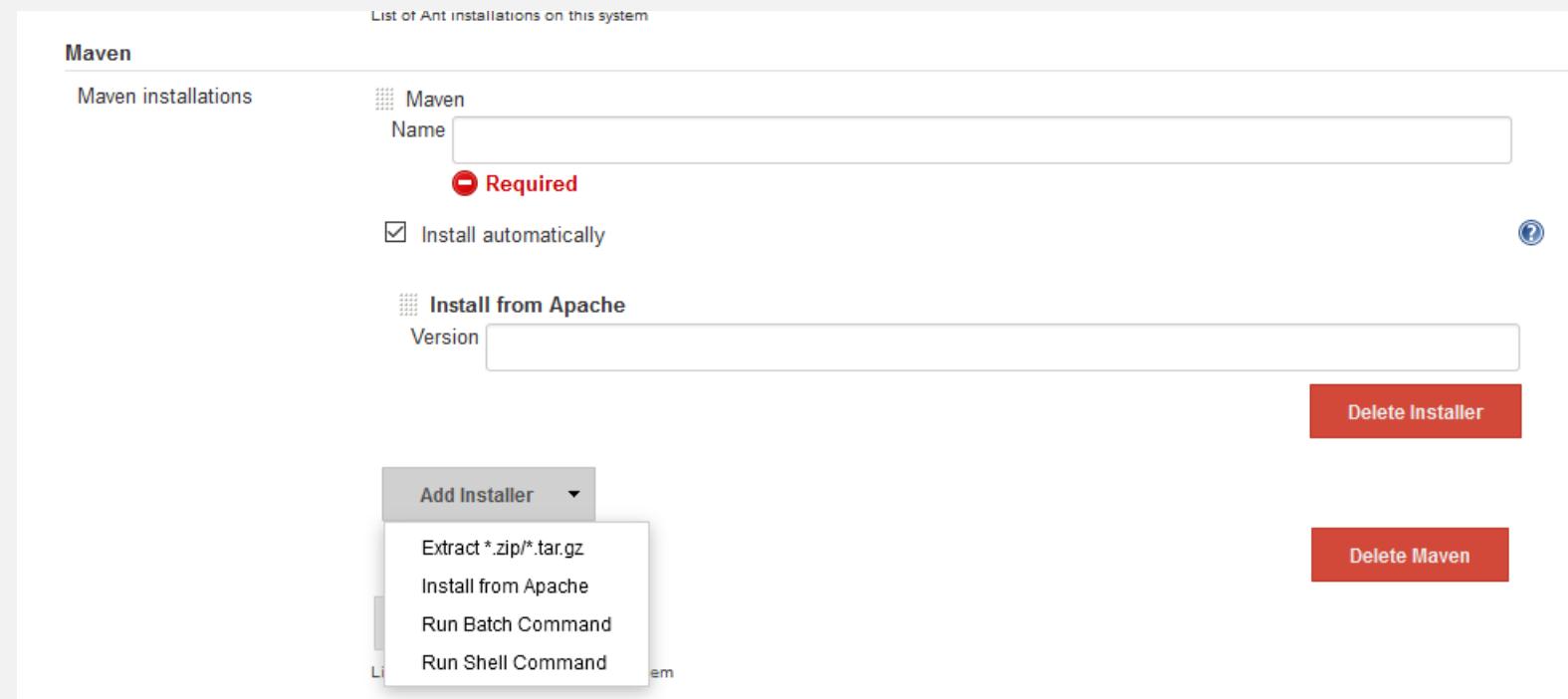
# JENKINS – ANT PLUGIN

- **Method 2: (Requires Internet Connection)**
  - Select Install Automatically. Jenkins will automatically download and install latest version of ANT Suitable for your systems
  - This method is preferred especially for Slave Installations



# JENKINS – A NOTE ON AUTO INSTALL

- Install Automatically allows you to have custom installers or scripts.
- This is used usually to install across many machines.
- Default is to download from Apache (version needs to be specified)
- Else you can extract an already downloaded file (Click on down arrow in add installer button)



# JENKINS – MAVEN PLUGIN

- Built in Plugin.
- Was part of core, now separated as a plugin
- 2 ways to configure the same.
- **Method 1:**
  - Provide the path specified in MAVEN\_HOME environment variable.
  - This is the most common way to do so and assumes that you have maven already setup.

# JENKINS – MAVEN PLUGIN

- **Method 2 of configuration: (requires Internet Connection)**
  - Select **Install Automatically**. Jenkins will automatically download and install latest version of Maven Suitable for your systems

**Maven**

Maven installations

 Maven	Name	LOCAL_MAVEN
MAVEN_HOME	c:\apps\maven	
<input type="checkbox"/> Install automatically		?
		Delete Maven

Add Maven

List of Maven installations on this system

**Maven Project Configuration**

Global MAVEN\_OPTS

Local Maven Repository

Default (~/.m2/repository)

## JENKINS – MAVEN PLUGIN

- In addition to other environment variables, the maven plugin makes the below variables available to your job/scripts.
  - POM\_DISPLAYNAME
  - POM\_VERSION
  - POM\_GROUPID
  - POM\_ARTIFACTID
  - POM\_PACKAGING
    - e.g. echo Group/ArtifactID is %POM\_GROUPID%/%POM\_ARTIFACTID%
- **Do not use** this for release management. There are separate M2 Release plugins for this purpose.

# JENKINS – JAVADOC PLUGIN

- Publishes your Javadoc if created/modified by your build
- For **Maven Projects**,
  - Goal to run: javadoc:javadoc
  - If Javadoc is being built as part of the build flow, it will automatically get published on web page
- For **Freestyle projects**,
  - Triggered via post-build Actions->Publish JavaDoc

Post-build Actions

 Publish Javadoc

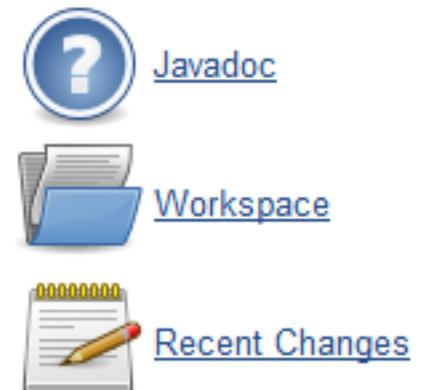
Javadoc directory:

Directory relative to the root of the workspace, such as 'myproject/build/javadoc'

Retain Javadoc for each successful build

[?](#) [Delete](#)

## Project gitcode



# JENKINS – HTML PUBLISHER

- Publishes and archives HTML reports
  - Your build needs to generate HTML files
  - Not specific to Javadoc only. E.g. PMD reports can be generated as HTML files.
- If not already installed, click on Manage Jenkins->Manage Plugins,
  - Navigate to available tab and install this plugin.

## Post-build Actions

### Publish HTML reports

#### Reports

HTML directory to archive

Index page[s]

index.html

Report title

HTML Report

Keep past HTML reports

Always link to last build

Allow missing report

Add

## JENKINS – GITHUB

- Non-trivial plugin that allows you to connect Jenkins to a SCM provider (in this case GITHUB)
- It allows you to
  - Create hyperlinks between Jenkins projects and GitHub
  - Trigger a job when a push to the repository happens by groking HTTP POSTs from post-receive hook and optionally auto-managing the hook setup.
  - Report build status result back to github as Commit Status

## JENKINS – GITHUB

- In github you create a web hook and an URL that will handle post receives. This is the manual way of doing it.
- Instead, first create a personal token in github.
  - Go to GitHub, log in, go to **Settings, Personal access tokens**, click on **Generate new token**.
  - Check **repo:status**
  - Click on Generate Token and copy it.
- In Jenkins Management, Add a new GITHUB Server config.
- You can either go to advanced->Manage Additional Github actions->Convert Login and Password to token (or)
- Generate a token in github and paste it back in Jenkins
- Ensure that the manage URLs is checked on. That's it.

# JENKINS - GITHUB

**GitHub Plugin Configuration**

Servers configs with credentials to manage GitHub integrations

GitHub Server Config	
Manage hooks	<input checked="" type="checkbox"/>
Credentials	<input type="text" value="some text"/> <input type="button" value="Add"/>
<input type="checkbox"/> Custom GitHub API URL	
GitHub client cache size (MB) <input type="text" value="20"/>	
Credentials verified for user SeshagiriSriram, rate limit: 4970	
<input type="button" value="Verify credentials"/>	
<input type="button" value="Delete GitHub Server Config"/>	
<input type="button" value="Add GitHub Server Config"/>	
List of GitHub Servers to manage hooks, set commit statuses etc.	
<input type="button" value="Re-register hooks for all jobs"/>	
Override Hook URL	<input type="checkbox"/> Specify another hook url for GitHub configuration
Additional actions	<input type="button" value="Manage additional GitHub actions"/>

# JENKINS - GITHUB

The screenshot shows a Jenkins GitHub integration interface. At the top, there is a navigation bar with tabs: Overview (selected), Yours, Active, Stale, and All branches. To the right of the tabs is a search bar labeled "Search branches...". Below the navigation bar, there is a section titled "Default branch". Under this section, there are two entries: "master" (Updated 8 minutes ago by SeshagiriSriram) and "default" (which is currently selected, indicated by a red "X" icon). There is also a "Change default branch" button.

Overview    Yours    Active    Stale    All branches

Search branches...

Default branch

master Updated 8 minutes ago by SeshagiriSriram    X    Default    Change default branch

Default branch

master Updated 2 minutes ago by SeshagiriSriram    ✓    Default    Change default branch

## Jenkins Status in GitHub

## JENKINS - GITHUB

- Security is a concern. The Hook URL is a public URL – so security is a big concern.
  - In Automatic mode, we specify an arbitrary URL where GitHub can do a post.
  - We then set up a reverse proxy and keep Jenkins inside a firewall and have reverse proxy forward traffic from GitHub to internal Jenkins address
- On Windows, these are issues:
  - Jenkins will use the the SSH key of the user it is running as, which is located in the %USERPROFILE%\.ssh folder. Therefore, Jenkins has to run as the user that has the SSH key configured.
  - Jenkins does not support passphrases for SSH keys. Therefore, if you set one while running the initial Github configuration, rerun it and don't set one.
-

## JENKINS – BUILD WITH PARAMETERS PLUGINS

- Allows users to provide parameters for a build in a URL.
- Users are prompted to enter and requested for parameters before triggering the job
- Exposes [\\$JENKINS/job/\\$JOB/parambuild](#) url to trigger the build with parameter.
- Useful if you want to create a list of jobs with parameters to trigger ahead of time, and execute it at some future date (e.g. a deployment plan).

# JENKINS – PARAMETER PASSING NOTES

- By default, any build can be parameterized by clicking the “this build is parameterized” check box
- Users are usually prompted to enter the parameter value before running the job.

The screenshot shows the Jenkins project configuration interface. On the left, there's a sidebar with tabs for 'Advanced Project' and 'Source Code Management'. Under 'Source Code Management', 'None' is selected, while 'CVS' is an option. At the bottom of this sidebar is a 'Save' button. In the main area, a checkbox labeled 'This build is parameterized' is checked. A dropdown menu titled 'Add Parameter' is open, listing various parameter types: Boolean Parameter, CVS Symbolic Name Parameter, Choice Parameter, Credentials Parameter, File Parameter, List Subversion tags (and more), Password Parameter, Run Parameter, String Parameter, and Text Parameter. The 'Boolean Parameter' item has a tooltip: 'the project is re-enabled.' To the right of the dropdown is a 'Advanced...' button.

## JENKINS – CREDENTIALS PLUGIN

- Provides a convenient way to organize credentials across domains and
- Maintain passwords secrets just once.
- Used by multiple other plugins indirectly when you are required to provide credentials.
- There are multiple credentials plugins including:
  - Plain Credentials
  - SSH Credentials
  - Google Oauth
  - Dockers Common

# JENKINS – CREDENTIALS PLUGIN

- The kinds of credentials
  - User name with password
  - SSH user name with private key
  - Certificate (PKCS#12 key)
- Scope of credential
  - Global
    - Available to the object on which the credential is associated **and all** objects that are children of that object. Use global-scoped credentials at job level
  - System
    - Only available to the object on which the credential is associated.
    - E.g. Uses: email authentication, slave connection, etc
    - Restricts where the credential can be used, thereby providing a higher degree of confidentiality to the credential.

## JENKINS – COPY ARTIFACT

- Allows you to copy artifacts built from other jobs into current workspace
- Very useful when using Jenkins for Continuous Deployment/Delivery

# JENKINS – COPY ARTIFACT

Build

Copy artifacts from another project

Project name: TEST ?

Which build: Latest successful build ▼ ?

Stable build only

Artifacts to copy: ?

Artifacts not to copy: ?

Target directory: ?

Parameter filters: ?

Flatten directories    Optional    Fingerprint Artifacts ?

Advanced...

## JENKINS – COPY ARTIFACT

- On installing this plugin, A new build step: “Copy artifacts from other projects” is now made available for free style build jobs.
- The default of Jenkins is to copy the directory structure
  - Use Flatten Directories option to override this

# DEMO & LAB

Q&A

# Questions???

# AGENDA FOR NEXT MODULE

- Introduction
- Jenkins Build Jobs
  - Creating a Freestyle Build Job
  - Configuring Source Code Management
  - Introduction to Build Triggers
- Introduction to Build Steps and flows
- Pre and Post-Build Actions
- Running Your New Build Job
- Working with Maven Build Jobs
- Maven Build flow



## FOOD FOR THOUGHT!!

- Why do we need separate plugins for Javadoc and HTML Publishing? Extending this further, explore if the JUNIT Plugin actually runs Unit test. If not, why not?

# **MODULE 04**

## **AN OVERVIEW OF JENKINS JOBS**



# ANATOMY OF A JENKINS JOBS

- A job consists of one or more pre-build steps.
- Examples of pre-build steps are:
  - A script to check if a directory exists and If not, create one.
  - Successful build of another project
- Jobs are started by triggers
- Examples of triggers are
  - Build after another project
  - Build periodically
  - Poll SCM



A pre-build step can be considered as preliminary steps before “main” job runs.

Jobs may be triggered manually

# ANATOMY OF A JENKINS JOBS

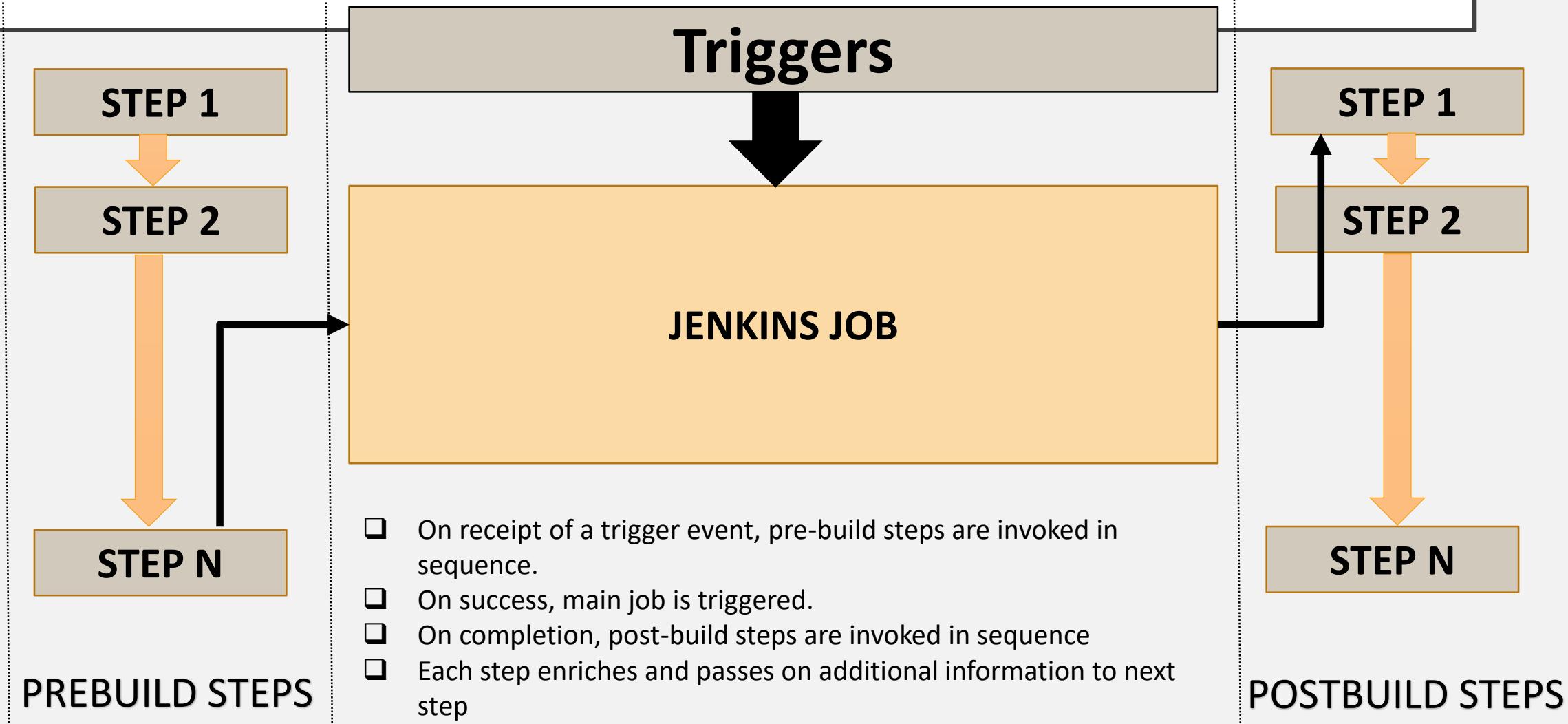
- A job is a sequence of build steps.
- By default, these are the build steps
  - Run Shell Command
  - Run Windows Shell Command (BAT/CMD files)
  - Run ANT targets
  - Invoke Top Level Maven projects
- Similar to pre-build steps, we have post-build actions also.
- Some of these include
  - Publish Artifacts
  - Build other projects
  - Publish Junit Test Results
  - Publish Javadoc
  - E-Mail Notification



What is listed here are the default steps and post-build actions.

Additional plugins will add to the list of available steps and post-build actions

# ANATOMY OF A JENKINS JOB



# JENKINS JOBS

- As Post Completion tasks, Jobs can start other jobs either on
  - Success – Most common actions steps...
  - Failure
  - Always
- The Chaining of jobs is a [pipeline](#). Typical use case
  - Perform unit tests only if compilation succeeds
  - Perform integration tests only if unit tests succeeds
  - Document (Javadoc) if code compile works (irrespective of test status)
  - Publish test results always (whether failed or passed)
  - Deploy only if all tests are successful

# SAMPLE PIPELINE



Ref:  
Jenkins: The definitive  
Guide

# TRIGGERS

- Manual build
  - Click manually on “Build Now” link to start the build
- On Schedule
  - Uses UNIX Cron notation
    - Minutes
    - Hours
    - Day of Month
    - Month
    - Day of Week
- When code changes in SCM
- Polling SCM
- Triggered on other builds

# TRIGGERS

## Build Triggers

- Trigger builds remotely (e.g., from scripts) (?)
- Build after other projects are built (?)
- Build periodically (?)
- Build when a change is pushed to GitHub
- Poll SCM (?)

**You can always run the job manually at any time.**

# THE CRON FORMAT

- Jenkins uses UNIX Cron notation
  - Minutes (0-59)
    - \* represent every value, hence \* \* \* \* \* means once every minute
    - / allows you to skip e.g. \*/5 means every 5 minutes
  - Hours (0-23)
    - Commas represent list of value, e.g. 2,3 means at 2 and 3 AM in morning.
  - Day of Month (1-31)
  - Month (1-12)
  - Day of Week (0-7) 0 and 7 are Sunday.
    - Ranges are allowed – 1-5 means between Monday and Friday
- Other shorthands:
  - @yearly, @annually, @monthly, @weekly, @daily, @midnight, and @hourly

# SCHEDULING

- **REMEMBER:** There are only 5 fields separate by a space.
- SCM Polling Pattern is usually: \* \* \* \* \*
- Periodic tasks put load on system.
- The H Symbol is used to distribute load more evenly.
- E.g. 0 0 \* \* \* runs once a day. So does H H \* \* \* (second form is preferred).
- Multiple schedules can be specified one per line.
- Lines beginning with # are comments.
- Empty lines are ignored.



# SCHEDULING

- The H symbol can be used with ranges also.
  - E.g. H H(0-7) \* \* \* means sometime between 12:00 AM and 7:59 AM
- This symbol does not work reliably with Day field.
  - Reason: Each month has a different number of days and the H provides a hash only between 1 and 28
- **Examples:**

## SCHEDULING

```
# every fifteen minutes (perhaps at :07, :22, :37, :52)
H/15 * * * *
# every ten minutes in the first half of every hour (three times, perhaps at :04, :14, :24)
H(0-29)/10 * * * *
# once every two hours every weekday (perhaps at 10:38 AM, 12:38 PM, 2:38 PM, 4:38 PM)
H 9-16/2 * * 1-5
# once a day on the 1st and 15th of every month except December
H H 1,15 1-11 *
```

## TRIGGERING VIA POLLING

- Better than scheduled jobs
- Applicable only for small projects with smaller number of builds
  - SCM/Network saturation
  - **Not applicable** for CVS with very large number of files
    - Move to GIT/Subversion
    - (or) poll every 30 minutes

## TRIGGER VIA SCM CHANGES

- **READ YOUR CHOSEN SCM MANUAL FIRST.**
- The core idea is that the SCM supports hooks on post-commit.
- Jenkins supports builds via URL. E.g.
  - <http://SERVER/jenkins/job/PROJECTNAME/build>
  - These can be invoked via wget or curl.
  - E.g. wget <http://localhost:8080/jenkins/job/somejob/build>
- Now your SCM post-commit hook script just invokes the wget/curl on Jenkins.

# TRIGGER VIA SCM CHANGES

- For Subversion
  - Add Post-Commit Hook
  - Create post-commit file in \$REPOSITORY/hooks directory
- See link: <https://wiki.jenkins-ci.org/display/JENKINS/Subversion+Plugin> for additional details on various possibilities for committing.

## Contents of post Commit file

```
REPOS="$1"
REV="$2"
UUID=`svnlook uuid $REPOS`  

/usr/bin/wget \  

--header "Content-  

Type:text/plain; charset=UTF-8" \  

--post-data "`svnlook changed --revision $REV  

$REPOS`" \  

--output-document "-" \  

--timeout=2 \  

http://server/subversion/${UUID}/notifyCommit?  

rev=$REV
```

## TRIGGER VIA SCM CHANGES

- In case Jenkins security is enabled, you will need to trigger build remotely (from scripts) and
- Pass an authentication token, which is hardcoded. E.g
  - <http://SERVER/jenkins/job/PROJECTNAME/build?token=MYSECRETTOKEN>
  - (or) pass user name/password as in:
    - <http://user:password@SERVER/jenkins/job/PROJECTNAME/build>
  - The 1<sup>st</sup> method will work when any user is allowed to do anything.

# TRIGGERING VIA SCM CHANGES

- For GITHUB, the process is slightly more complex.
- **Step #1:**
  - Create a robot user say githubuser
  - Create a github user for the above
  - Alternatively, you can use deploy keys as listed [here](#)
  - Verify: ssh [git@github.com](mailto:git@github.com) should return authenticated.
- **Step #2:**
  - In Jenkins, install both git and github plugins.
- **Step #3:**
  - Add the https github URL to specify github project.

GitHub project

<https://github.com/SeshagiriSriram>HelloWorld/>



# TRIGGERING VIA SCM CHANGES

- Under Source Code Management, use GIT. For private repos, use SSH notation style

The screenshot shows the Jenkins interface for managing repositories. A radio button labeled 'Git' is selected. Below it, there's a section for 'Repositories'. A single repository is listed with the 'Repository URL' set to <https://github.com/SeshagiriSriram>HelloWorld>. The 'Credentials' field contains 'SeshagiriSriram/\*\*\*\*\*' with a dropdown arrow and an 'Add' button next to it. On the right side of the screen, there are three buttons: 'Advanced...', 'Add Repository', and 'Delete Repository'.

- Under build triggers, enable Build when changes made to github.
- Step #4:**
  - Enable a public URL on your end e.g. <http://yourdomain.com/github-webhook> and
- Step #5:**
  - In github, enable the posthook by passing the user name/password .e.g
    - <http://myjenkinsuser:myjenkinspassword@yourdomain.com/github-webhook>.

# GITHUB

- Now when a commit happens to the repository, git hub will post to the public URL.
- The Github plugin is now aware of posts to this URL and will consider this as a trigger to perform a build from github

# TYPES OF JOBS

- Free Style
  - You can do pretty much anything in this, including running Maven projects
- Maven projects
  - Requires POM.xml
  - Executes maven lifecycle goals
- Monitor an External job
- Multi-Configuration jobs
  - When you want to build for a number of different configurations but commands are similar
  - Also called a matrix job

# TYPES OF JOBS

- Multi Job Project

## MultiJob Project RELOAD

More than one job runs in more than one phase. Triggers can be controlled on how each job/phase is to be triggered.

S	W	Job	Last Success	Duration
		<a href="#">RELOAD</a>	59 sec	15 sec
		<i>test</i>		
		<a href="#">gitcode</a>	52 sec	N/A 6 sec
		<a href="#">Test</a>	52 sec	N/A 0.3 sec

# MAVEN PROJECTS

- This is pretty much straightforward. One interesting thing
- This can build the module if it depends on other modules and if the SNAPSHOT changed, will build this module also.
  - E.g Module A depends on B 1.0.1
  - Now say module B's is also built in same Jenkins and the snapshot version changed to 1.0.2-SNAPSHOT.
  - Then if the build trigger “Build on Snapshot Dependency build” is enabled, Module A will automatically be built.
- This is what CI is all about. Integration building now is done as soon as inter dependent modules are built.

# MAVEN STYLE JOBS

**Build**

Root POM pom.xml 

Goals and options 

Specifies the goals to execute, such as "clean install" or "deploy". This field can also accept any other command line options to Maven, such as "-e" or "-DskipTests=true".

MAVEN\_OPTS  ▾

Incremental build - only build changed modules 

Disable automatic artifact archiving 

Disable automatic site documentation artifact archiving 

Disable automatic fingerprinting of consumed and produced artifacts 

**Some of the options deal with artifact management.**

# MAVEN STYLE JOBS

- Build modules in parallel ?
  - Use private Maven repository ?
  - Resolve Dependencies during Pom parsing ?
  - Run Headless ?
  - Process Plugins during Pom parsing ?
  - Use custom workspace ?
- Maven Validation Level DEFAULT ▼
- Settings file Use default maven settings ▼ ?
- Global Settings file Use default maven global settings ▼ ?

## Post Steps

- Run only if build succeeds  Run only if build succeeds or is unstable  Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Add post-build step ▾

# MAVEN JOBS

- Interesting Options to consider:
  - Incremental build
    - Only build modules that have changed to speed up the build cycle
  - Build modules in parallel
    - Same reasoning as above. Anything to speed up build cycle is good.
  - Resolve Dependencies during POM
    - Use with care. The first time a local Maven Repository is being built, this option can slow down build cycles.

## FREE STYLE JOBS

- Can be used to build adhoc projects, including maven
- Can be Triggered from
  - Other scripts
  - After other projects are built
  - Built Periodically
  - Can Poll SCM
  - Build on SCM changes (see earlier notes on setting up URLs in GITHUB)

# BUILD FLOW PROJECTS

- If the Build flow plugin is enabled, you can also create flows of jobs based on DSL.

## Flow

Flow run needs a workspace

Define build flow using flow DSL

```
1 build( "flow1" )
2 build( "flow2" )
3
```

- Parameters can be passed e.g.
  - `b = build("job1", param1:"mycode", param2:"bar")`
  - `build("job2",param1:b.build.number)`

# BUILD FLOW PROJECTS

- Environment variables can be obtained using this expression:
  - `def revision = b.environment.get("GIT_REVISION")`
- Predefined variables:

Variable	Meaning
build	The current flow execution.
out	The flow build console
env	The flow environment, as a Map
params	Triggered parameters
upstream	the upstream job, assuming the flow has been triggered as a downstream job for another job

## BUILD FLOW EXAMPLES

```
// output values for display
out.println 'My Parameters:'
out.println params
out.println 'Build Object Properties:'
build.properties.each { out.println "$it.key -> $it.value" } // Similar to Java Code

// use it in the flow
build("job1", parent_param1: params["param1"])
build("job2", parent_workspace:build.workspace)
```

# BUILD FLOW

- Use the Guard/Rescue syntax to handle possible errors
- This is similar to Try/catch in java
- Errors can be ignored

```
guard {  
    build("some job")  
}  
rescue {  
    build ("cleanup")  
}
```

```
ignore(FAILURE){  
    build("send_me_mail")  
}
```

```
retry(3){  
    build("send_me_mail")  
}
```

## BUILD FLOW

- Groups can be named for future use.

```
join = parallel ([  
    first: { build("job1") },  
    second: { build("job2") },  
    third: { build("job3") }  
])  
  
// now, use results from parallel execution  
build("job4",  
    param1: join.first.result.name,  

```

# BUILD FLOW

- Parallel flows can be defined

```
parallel(  
    { build("job1")},  
    { build("job2")},  
    { build("job3")},  
)  
build("job4") // 1, 2 and 3 are in ||
```

# BUILD FLOW

- Parallel branches can sequentially chain jobs
  - In this example, Jobs 1, 2 and 3 are executed in sequence
  - Jobs 4, 5,6 are executed in sequence
  - However group (Job 1, 2,3) is executed in parallel with group (Job 4, 5,6)

```
parallel(  
  { build("job1")  
   build("job2")  
   build("job3")  
 },  
  { build("job4")  
   build("job5")  
   build("job6")  
 }  
)
```

# BUILD PIPELINES

- Completion of one job can trigger other job.
  - This is built into Jenkins

The screenshot shows two configurations for Jenkins build triggers:

**Build other projects** (Left):  
A checked checkbox labeled "Build other projects". Below it is a text area explaining that builds of other projects are triggered once a build is successfully completed, with examples like "abc, def". It also notes that this is useful for splitting long build processes into stages. A "Projects to build" input field contains "performance-tests". Below the input field are three radio button options:

- Trigger only if build succeeds
- Trigger even if the build is unstable
- Trigger even if the build fails

**Build Triggers** (Right):  
A checked checkbox labeled "Build after other projects are built". Below it is a text area explaining that a trigger is set up so that when other projects finish building, a new build is scheduled for this project, useful for running extensive tests. A "Projects names" input field contains "acceptance-tests". Below the input field is a note: "Multiple projects can be specified like 'abc, def'".

A large orange "OR" symbol is centered between the two screenshots.

# BUILD PIPELINES

- Use the Jenkins **Build Pipeline** Plugin
- Used as a post Build action from Project A.
- Default is to run manually
- To visualize, create a view based on build pipeline

# BUILD PIPELINES

## Build Pipeline Plugin -> Manually Execute Downstream Project



The Build Pipeline View plugin uses the relationship between upstream and downstream projects to map out a build pipeline. As default, the downstream project requires a manual trigger when the upstream job is completed. Specify the Downstream triggered projects in the "Downstream Project Name" field. Multiple projects can be specified by using comma, like "abc, def".

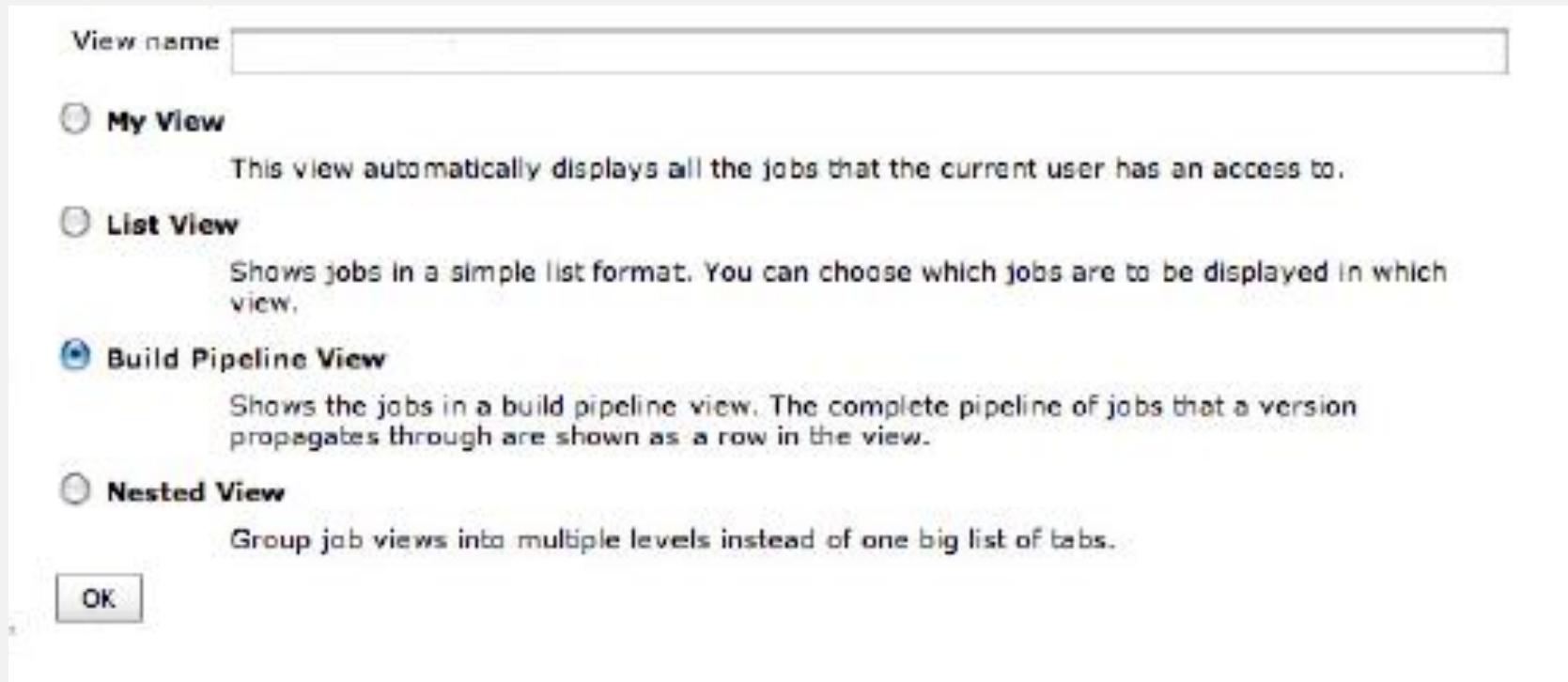
(from [Build Pipeline Plugin](#))

Downstream Project Names



# BUILD PIPELINES

- You can also create views called Build Pipeline views.
- To configure, navigate to URL of view: `http://<myserver>/view/<view_name>/configure`
- Just need to specify the starting job.



# BUILD PIPELINES

- Used **Parametrized Trigger** Plugin
- Can re-use parameters

The screenshot shows the configuration for a build pipeline trigger. It includes sections for 'Build Triggers' and 'Predefined parameters'.

**Build Triggers:**

- Trigger parameterized build on other projects**:
  - Projects to build**: A text input field containing 'X'.
  - Trigger when build is**: A dropdown menu set to 'Stable'.
  - Trigger build without parameters**: A checkbox that is unchecked.

**Current build parameters**: A section with a red 'Delete' button.

**Predefined parameters**: A section showing a parameter named 'NEW\_BUILD\_PARAMS=somevalue'.

# BUILD PIPELINES

- Downstream Buildview Plugin
  - This is used to visualize the flow of the jobs
  - Can be used instead of Build Pipeline Views (It's a matter of preference here)

The screenshot shows a Jenkins interface titled "Downstream build view". It displays a hierarchical tree of downstream builds for a parent build. The tree structure is as follows:

- acceptance-tests build number 62 (Thu Apr 19 10:08:06 EEST 2012 - SUCCESS)
  - deploy-testing build number 60 (Thu Apr 19 10:08:18 EEST 2012 - SUCCESS)
    - performance-tests build number 37 (Thu Apr 19 10:08:29 EEST 2012 - SUCCESS)
      - deploy-staging build number 55 (Thu Apr 19 10:08:38 EEST 2012 - SUCCESS)
        - deploy-production build number 20 (Thu Apr 19 10:09:54 EEST 2012 - SUCCESS)

# DEMO & LAB

Q &A

# Questions???

## AGENDA FOR NEXT MODULE

- Introduction
- Parameterized Builds
- Distributed builds
- Setting Email Notification
- Enabling Security on Jenkins
- Different Levels of Authentication
- Types of Access
- Administration of the Access.



## FOOD FOR THOUGHT!!

- What is the difference between a job and a project ? (Trick Question)
- A build failure is always fatal. Do you agree with this statement?
- What is the build flow/pipeline for your projects? Is it the same across the entire organization? If not, why?

# MODULE 05

## PARAMETRIZATION OF BUILDS AND NOTIFICATION



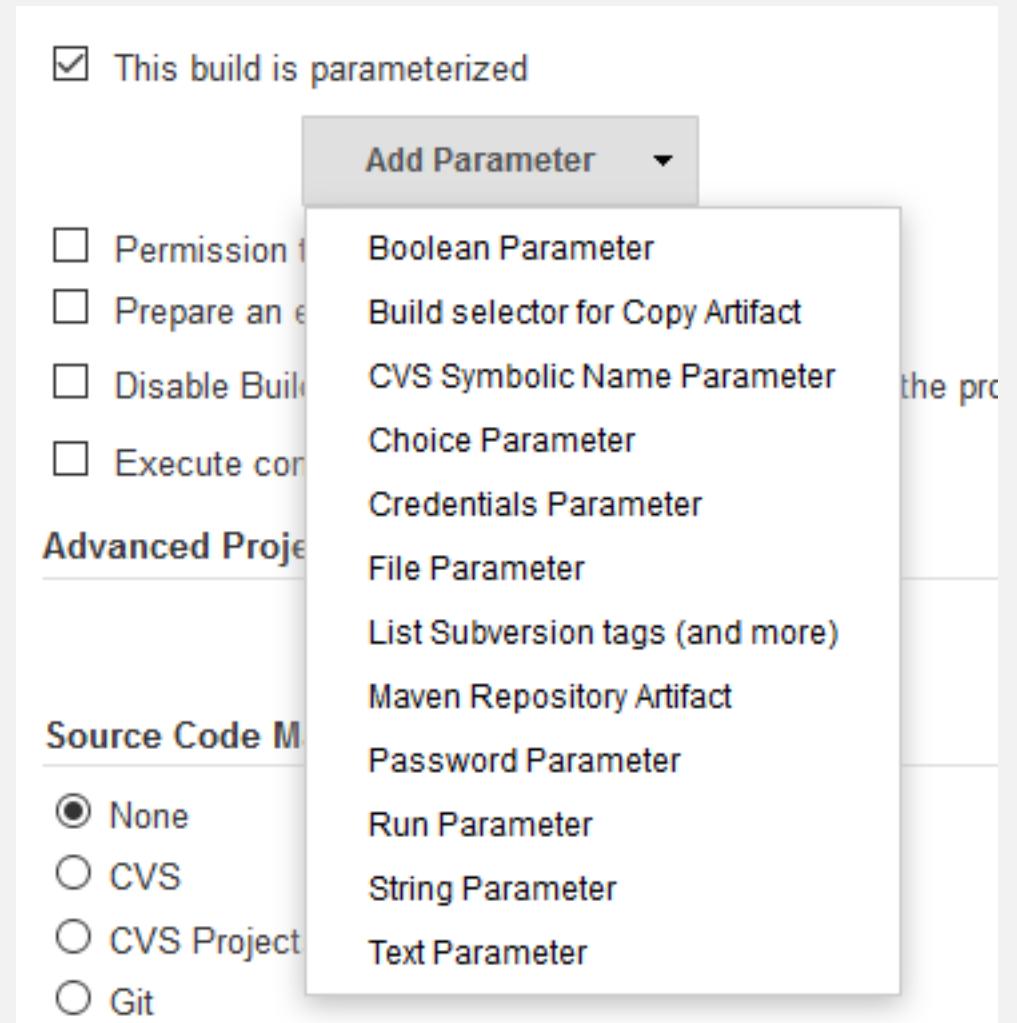
# PARAMETERIZATION

- Jenkins allows parameters to be set for builds
- This can be set external via SHELL variables which are made
- Alternatively use the Build with Parameters plugin
- Also you can specify additional parameters in job properties
- When configuring a job, check the “This build is parameterized” check box to enable parameterization



# TYPES OF PARAMETERS

- Simple Types
  - Boolean
  - Choice – Enter a drop down of choices
  - String
  - Text (Same as String, but allows new lines)
  - Password (Same as String, but will be masked)
- Complex Types
  - File Parameter – Allows for files to be uploaded to specified locations
  - Run Parameters – Access past builds
  - Build selector for Copy Artifact (Requires Copy Artifact Plugin)



# PARAMETRIZATION EXAMPLE

This build is parameterized ?

**Choice Parameter** ?

Name	ENVIRONMENT	<span style="color: blue;">?</span>
Choices	DEV TEST PROD	<span style="color: blue;">?</span>
Description	<input type="text"/>	<span style="color: blue;">?</span>
<a href="#">[Plain text]</a> <a href="#">Preview</a>		<span style="color: blue;">?</span>

Delete

**String Parameter** ?

Name	STRINGPARAM	<span style="color: blue;">?</span>
Default Value	XX	<span style="color: blue;">?</span>
Description	<input type="text"/>	<span style="color: blue;">?</span>
<a href="#">[Plain text]</a> <a href="#">Preview</a>		<span style="color: blue;">?</span>

# PARAMETRIZATION EXAMPLE

## Project X

This build requires parameters:

ENVIRONMENT

STRINGPARAM XX

**Build**

**Click on Build to start the build**



## Console Output

```
Started by user anonymous
Building in workspace e:\apps\jenkins\jobs\X\workspace
[workspace] $ cmd /c call E:\apps\tomcat8\temp\hudson2049588060178015153.bat
ENVIRONMENT: DEV
STRINGPARAM: XX
Finished: SUCCESS
```

## PARAMETRIZATION NOTES

- As best practices, please keep parameter names consistently.
- It is preferred to keep the name in all **UPPERCASE**.
- The difference between TEXT and STRING Parameter is this:
  - TEXT Parameter allows for new lines in entering the value – STRING Does not.
  - Credentials parameter requires the credential plugin to be enabled

# PARAMETERS

- Alternatively, parameters can be passed as parameters from an upstream job (or)
- Using the **build with parameters** plugin.
- A Parameterized build is accessed by POSTING to
  - `http[s]://<yourserver>/job/$JOB/buildwithParameters?PARAMETER=value`
  - E.g.
    - <http://localhost:8080/Jenkins/job/testjob/buildwithParameters?XYZ=test&ABC=test2>
- In above example, the job testjob is run with 2 parameters XYZ and ABC with values of test and test2.
- All parameters must be URL safe and encoded.
  - E.g. `&MyRunParam=testjob%2399`" for test-job#99
- Run Parameters are always passed as job#buildNumber e.g. test-job#999 indicates build 999 for test-job
- A parameter of delay=0sec starts the job immediately

## NOTES



Most of Jenkins functionality is exposed via REST API. To see full API list, navigate to:

`http://<yourserver>/api`

e.g. `http://localhost:8080/jenkins/api`

## FURTHER PARAMETERS EXAMPLE

- We will create a local credential and have our job print the user name/password
- **DO NOT** mark the job as parametrized.
  - Since we are using Credentials parameter, we will instead use the Credentials Binding Plugin **instead**
- Add a credentials (user name and password) and bind to variable MY\_CREDENTIALS.
- Add code to execute awk script (Shown only for demo purposes – not production quality)
  - Full setup is given next

# PARAMETERS EXAMPLES

Use secret text(s) or file(s) ?

## Bindings

### Username and password (conjoined)

Variable

MYVARIABLE

Credentials

Specific credentials

Parameter expression

someuser/\*\*\*\*\*



Delete

Add

## Build

### Execute Windows batch command

Command

```
@echo off  
echo %MYVARIABLE% | c:\cygwin64\bin\gawk -F: '{print "User name:", $1}'  
echo %MYVARIABLE% | c:\cygwin64\bin\gawk -F: '{print "Password:", $2}'
```

## PARAMETERS EXAMPLES

- Execute the job
- The console output shows correct output



### Console Output

```
Started by user admin
[EnvInject] - Loading node environment variables.
Building in workspace C:\jenkins\workspace\CheckCredentials
[CheckCredentials] $ cmd /c call C:\Windows\TEMP\hudson753280693791804471.bat
User name: someuser
Password: hacker12##
Started calculate disk usage of build
Finished Calculation of disk usage of build in 0 seconds
Started calculate disk usage of workspace
Finished Calculation of disk usage of workspace in 0 seconds
Finished: SUCCESS
```

## PARAMETERS EXAMPLE

- In the above example, the credentials themselves are not visible.
- However, these are stored as UserName:Password (separated by a colon)
- That is what the script does: simply display the user name and password.
- This could be extended to pass the password safely to other scripts/applications that need it.

# DEFAULT ENVIRONMENT VARIABLES SET

Variable	Meaning
BUILD_NUMBER	Current Build Number e.g. 189
BUILD_ID	Similar to BUILD_NUMBER for builds created with Jenkins 1.597+
BUILD_DISPLAY_NAME	Display Name of current build. Default is #BUILD_NUMBER e.g. #189
JOB_NAME	Name of project of this build e.g. test or MYUNIT/test
BUILD_TAG	String of format \${JOB_NAME}-\${BUILD_NUMBER} e.g. test-189
NODE_NAME	Master if being built on master, else name of slave
NODE_LABELS	Whitespace-separated list of labels that the node is assigned.
WORKSPACE	Absolute path of directory used for build
JENKINS_HOME	Directory on Master node used to store data. Defaults to Home Directory/.Jenkins
JENKINS_URL	Full URL of Jenkins
BUILD_URL	Full URL of current build. E.g. http://server:port/jenkins/job/foo/15/
JOB_URL	Full URL of current job. E.g. http://server:port/jenkins/job/foo

# SETTING E-MAIL NOTIFICATION

- By default, Jenkins supports e-mail notification. This is enabled in the post-build section

Build

Add build step ▾

Post-build Actions

E-mail Notification

Recipients sriram@test.org, xxx@x.org

Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

Send e-mail for every unstable build

Send separate e-mails to individuals who broke the build

including support for attachment of files, build logs

# SETTING E-MAIL NOTIFICATION

- Under Configure System menu option, there are also options to set the SMTP server and reply-to address.

E-mail Notification

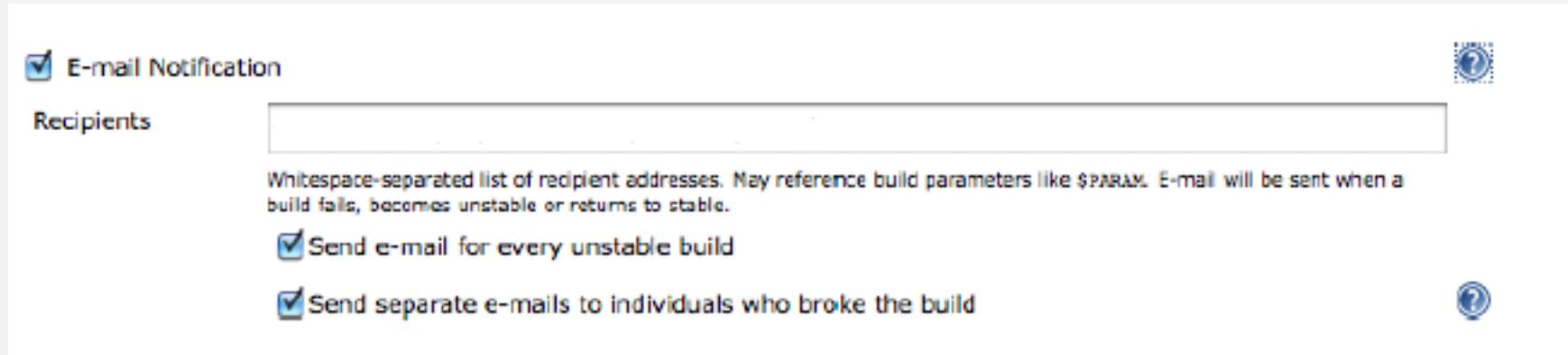
SMTP server	<input type="text"/>	
Default user e-mail suffix	<input type="text"/>	
<input type="checkbox"/> Use SMTP Authentication	<input type="checkbox"/>	
Use SSL	<input type="checkbox"/>	
SMTP Port	<input type="text"/>	
Reply-To Address	<input type="text"/>	
Charset	<input type="text" value="UTF-8"/>	
<input type="checkbox"/> Test configuration by sending test e-mail		

## SETTING E-MAIL NOTIFICATION

- The SMTP Server needs to be secured.
- To do so, create a system level credential which is used by Jenkins
- The reply-to address should be a valid email address which is essentially a no-reply email.

## E-MAIL NOTIFICATIONS

- Email notifications are sent as post build actions.
- Email addresses are separated by commas.
- It is preferable to send email individually to person whose commit broke the build.
  - Jenkins security has to be enabled for same.



# E-MAIL NOTIFICATIONS

- Notifications are sent
  - Whenever a build fails (e.g. compilation error).
  - When the build becomes unstable for the first time (e.g. if there are unit test failures).
    - Unless you configure it to do so, Jenkins will not send emails for every unstable build, but only for the first one.
  - When a previously failing or unstable build succeeds, to let everyone know that the problem has been resolved.

# E-MAIL NOTIFICATIONS

- Usually the default template is good enough. If not,
- Install the Email Template Management plugin

The screenshot shows the Jenkins interface for managing email notification templates. The top navigation bar has tabs for 'Bringing Order to Your Je...', 'edureka\_jenkins - Dropbox', 'Jenkins', and 'Configuring the Mail Serv...'. The main title is 'localhost:8080/emailexttemplates/addTemplate'. The left sidebar includes links for 'Manage Jenkins', 'Editable Email Templates', and 'Add New Template'. A 'Build Executor Status' panel shows '1 Idle' and '2 Idle'. The right side is the 'the template' configuration form:

the template	
ID	emailext-template-1446203495155
Name	[Empty]
Description	[Empty]
Disable Extended Email Publisher	<input type="checkbox"/>
<small>Allows the user to disable the publisher, while maintaining the settings</small>	

At the bottom, there is a footer with the text 'Copyright © 2017 Seshagiri Sriram'.

# E-MAIL NOTIFICATIONS

- A complete list of tokens and support can be viewed by clicking on the help icon against Content Token Reference.

## Content Token Reference



Some special tokens are:

- \$DEFAULT SUBJECT
  - This is the default email subject that is configured in Jenkins system configuration page.
- \$DEFAULT CONTENT
  - This is the default email content that is configured in Jenkins system configuration page.
- \$BUILD\_ID
  - Displays the build ID of the current build.
- \$BUILD\_NUMBER
  - Displays the number of the current build.

# E-MAIL TRIGGERS

- You can also configure the triggers when email is sent out. The following are some triggers:

Trigger	Meaning
Failure	Any time the build fails
Still Failing	Any Successive build failures
Unstable	Any time a build is unstable
Still Unstable	Any successive unstable builds
Success	Any Successful build
Fixed	When a build changes from Failure or Unstable to successful
Before Build	Send before every build begins

Q&A

# Questions???

## AGENDA FOR NEXT MODULE

- Recap
- Automating Your Unit and Integration Tests
- Configuring Test Reports in Jenkins
- Displaying Test Results
- Ignoring Tests
- Code Coverage
- Automated Acceptance Tests
- Automated Performance Tests with JMeter



## FOOD FOR THOUGHT!!

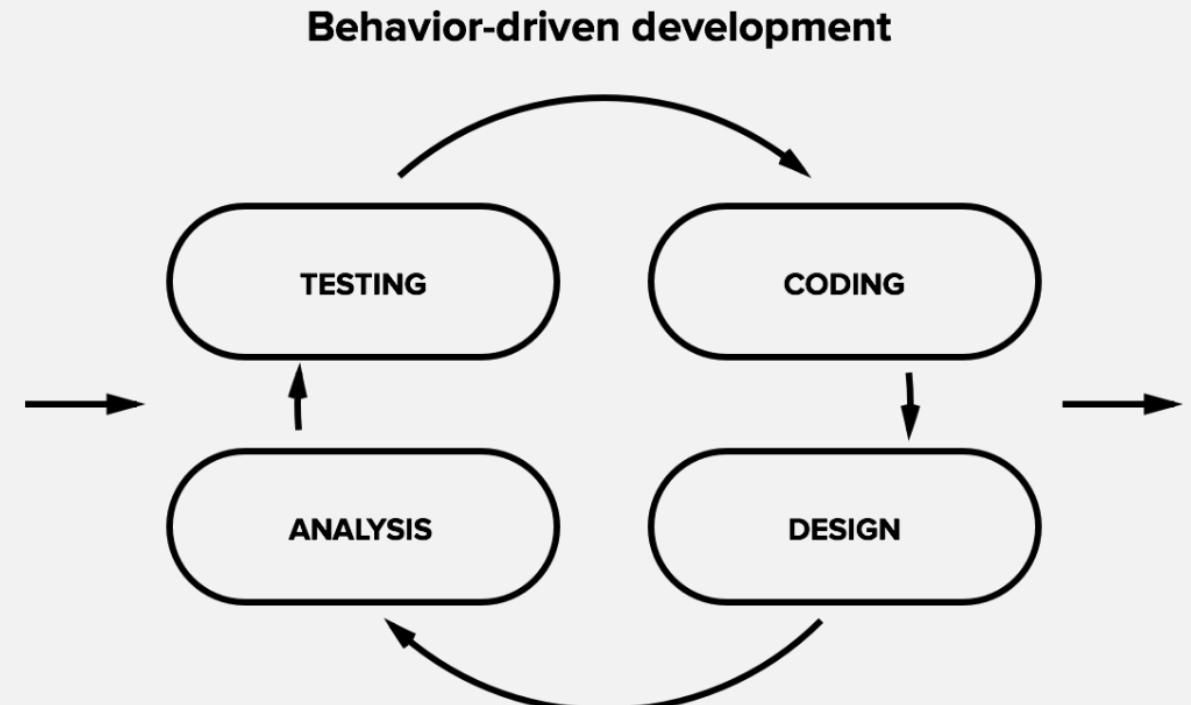
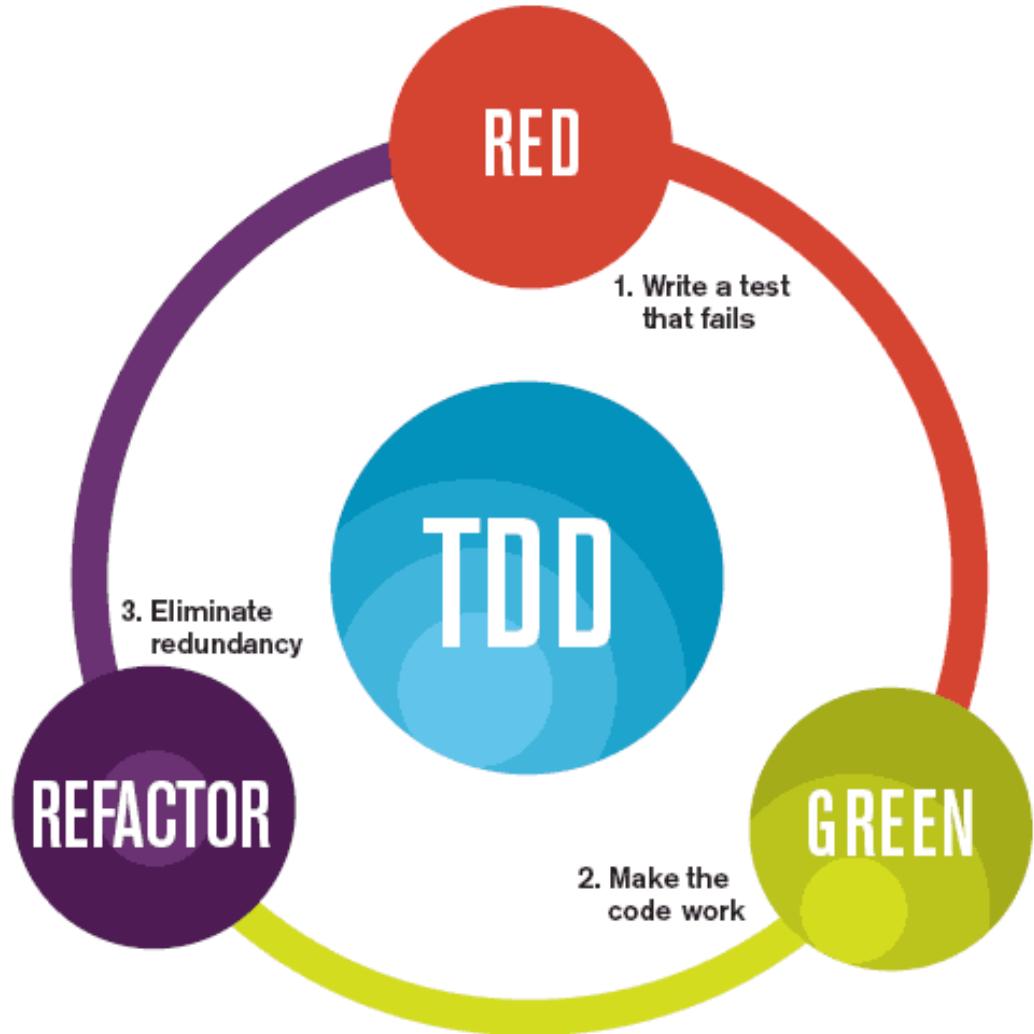
- How would you validate if the parameters passed are indeed correct from business point of view?
- What plugins are available for security?

# **MODULE 06**

## **AUTOMATED TESTING AND STATIC CODE ANALYSIS**

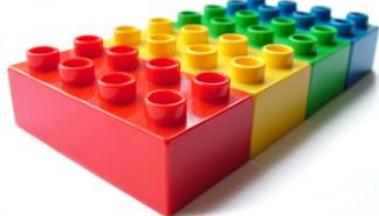


# WRITING TEST CASES

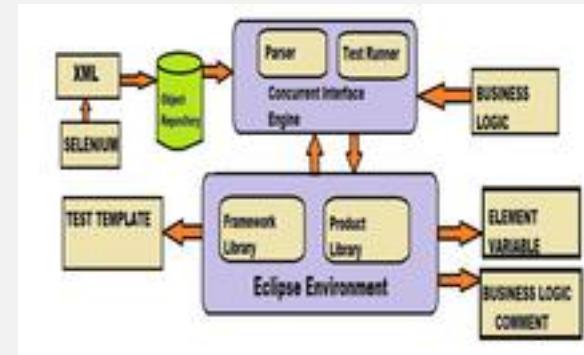


# JENKINS AND TESTING

Unit Tests



bbv Software Services AG  
www.bbv.ch



Integration Tests

Functionally Testing an App



Functional Tests

Web Tests

Performance Tests

# JENKINS AND TESTING



Developers

Communicate  
(via)

- easyb
- fitnesse
- jbehave
- rspec
- Cucumber
- .....



Stakeholders



Reporting should be as far as possible in non-technical terms.

## TESTING FACILITIES

JUnit



easyb

N unit



Ruby Test::Unit

CPPUnit



FitNesse

# JENKINS AUTOMATED TESTING

- Almost all xUnit frameworks are similar.
- You will need to specify exactly which XUNIT component (e.g. Junit, testing, etc. you are using)

# CONFIGURING TEST REPORTS

- For maven style jobs, just make sure you are running a goal (test or verify) that will generate reports.



The screenshot shows the Jenkins configuration page for a job named 'MavenTest'. The URL is 'localhost:8080/job/MavenTest/configure'. The 'Build' section is visible, showing the 'Root POM' set to 'pom.xml' and the 'Goals and options' set to 'test'. The page includes standard Jenkins navigation and status indicators at the top.

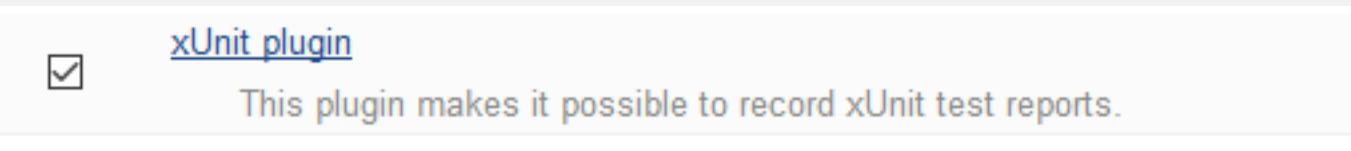
# CONFIGURING TEST REPORTS

- For free style projects, build should run the tests and publish the same yourself.

The screenshot shows the Jenkins configuration interface for a project. Under the 'Build' section, there is a step titled 'Invoke top-level Maven targets' with 'Maven Version' set to 'LOCAL\_MAVEN' and 'Goals' set to 'test'. Below this, there is an 'Advanced...' button and a 'Delete' button. A 'Add build step' dropdown menu is visible. Under the 'Post-build Actions' section, there is a step titled 'Publish JUnit test result report' with 'Test report XMLs' set to an empty field. A note below the field states: 'Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/\*.xml'. Basedir of the fileset is the workspace root.' There is also a question mark icon next to the note.

## SUPPORT FOR NON-JAVA

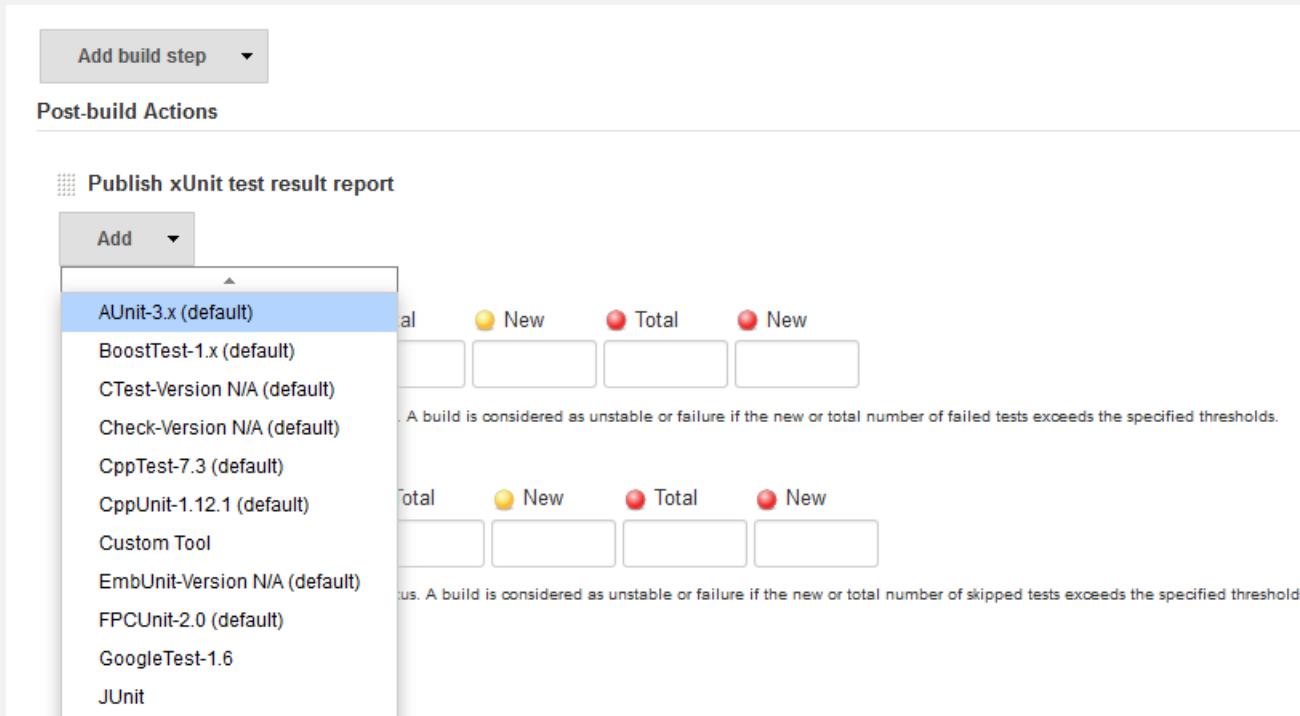
- xUnit plugin should be enabled.



-

# SUPPORT FOR NON-JAVA

- In addition, you will need to select the type of xUnit as shown below.



## SKIPPING TESTS

- A JUnit test can be marked not to be used by using the @Ignore annotation
- For Maven, all tests can be skipped by setting skipTests to true

```
@Ignore("TEST")
@Test
public void mTest(){.....}
```

```
mvn package –  
DskipTests=true
```

## SKIPPING TESTS

- In TestNG, skips can be ignored by setting enabled to false

```
@Test(enabled=false)
Public void mTest(){ ....}
```

- In TestNG, dependencies between tests can also be set so that a test is run only after other tests have run,

```
@Test(dependsOnMethod
      s={"test1"})
Public void mTest(){ ....}
```

# CONFIGURING REPORTS

- A freestyle report can also generate JUNIT reports via an ant script.
- Depending on how ANT JUNIT is configured, the JUNIT Report may directly be output to HTML (or) retained in XML file.
  - See here: <https://ant.apache.org/manual/Tasks/junit.html> for full details on configuring JUNIT and Ant

# POST BUILD ACTIONS

- Reporting on Test Results
  - JUNIT XML Reports

**Post-build Actions**

 **Publish JUnit test result report** 

Jenkins understands the JUnit test report XML format (which is also used by TestNG). When this option is configured, Jenkins can provide useful information about test results, such as historical test result trends, a web UI for viewing test reports, tracking failures, and so on.

To use this feature, first set up your build to run tests, then specify the path to JUnit XML files in the [Ant glob syntax](#), such as `**/build/test-reports/*.xml`. Be sure not to include any non-report files into this pattern. You can specify multiple patterns of files separated by commas.

Once there are a few builds running with test results, you should start seeing something like [this](#).

(from [JUnit Plugin](#))

**Test report XMLs**

[Fileset 'includes'](#) setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/\*.xml'. Basedir of the fileset is [the workspace root](#).

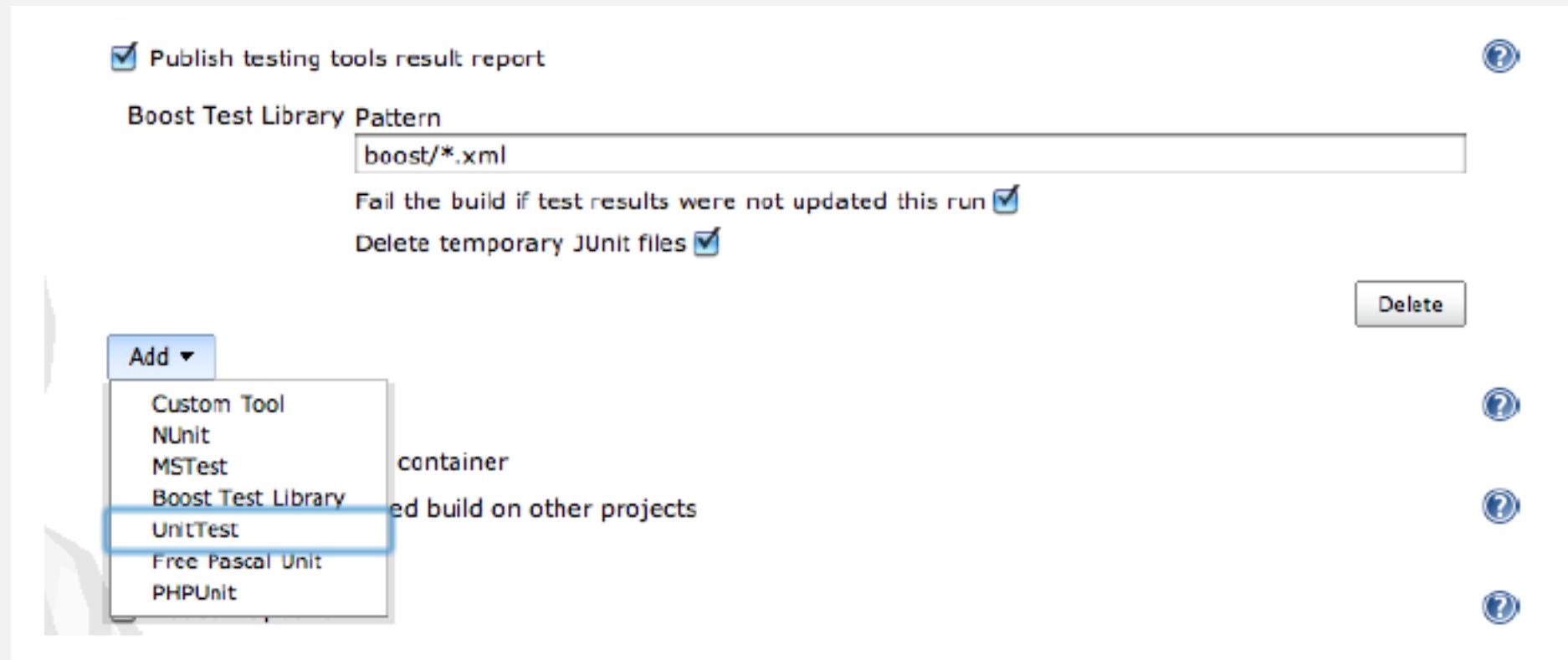
Retain long standard output/error 

**Health report amplification factor**   

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

# POST BUILD ACTIONS

- Reporting on Test Results
  - XUNIT XML Reports



## POST BUILD ACTIONS

- Reporting on Test Results
  - If your ant script or any other script reports using HTML directly, then the HTML Publisher plugin can be used in post-build action to publish reports directly to home page of the job.

## A WORKING EXAMPLE WITH JUNIT

- For our example, we will consider using a free style job with JUNIT 3 and ANT.
- The main running is done via a <junit> task. E.g.
  - <junit printSummary="yes" haltonerror="false" haltonfailure="false" fork="true" dir=".">>
- The formatter type is set to xml and an output directory is specified.
- Within the Junit task, we invoke batchtest with a todir specified. E.g.
  - <batchtest skipNonTests="false" todir="\${maven.test.reports}">
- Complete ant script fragment available on resources section of this course
- As post-build action, publish JUNIT Reports.

## A WORKING EXAMPLE WITH JUNIT

- For our example, we will consider using a free style job with JUNIT 3 and Maven
- The build step just invokes a top level test or verify goals. This is done by executing a script that calls mvn test or mvn verify.
- In post Build step, call publish Junit reports (As in previous slide)

# A WORKING EXAMPLE WITH JUNIT

**Build**

Invoke top-level Maven targets

Maven Version: Maven 2.2.1

Goals: verify

Advanced... Delete

Add build step ▾

**Post-build Actions**

Publish Javadoc

Archive the artifacts

Aggregate downstream test results

Publish JUnit test result report

Test report XMLs: \*\*/target/surefire-reports/\*.xml

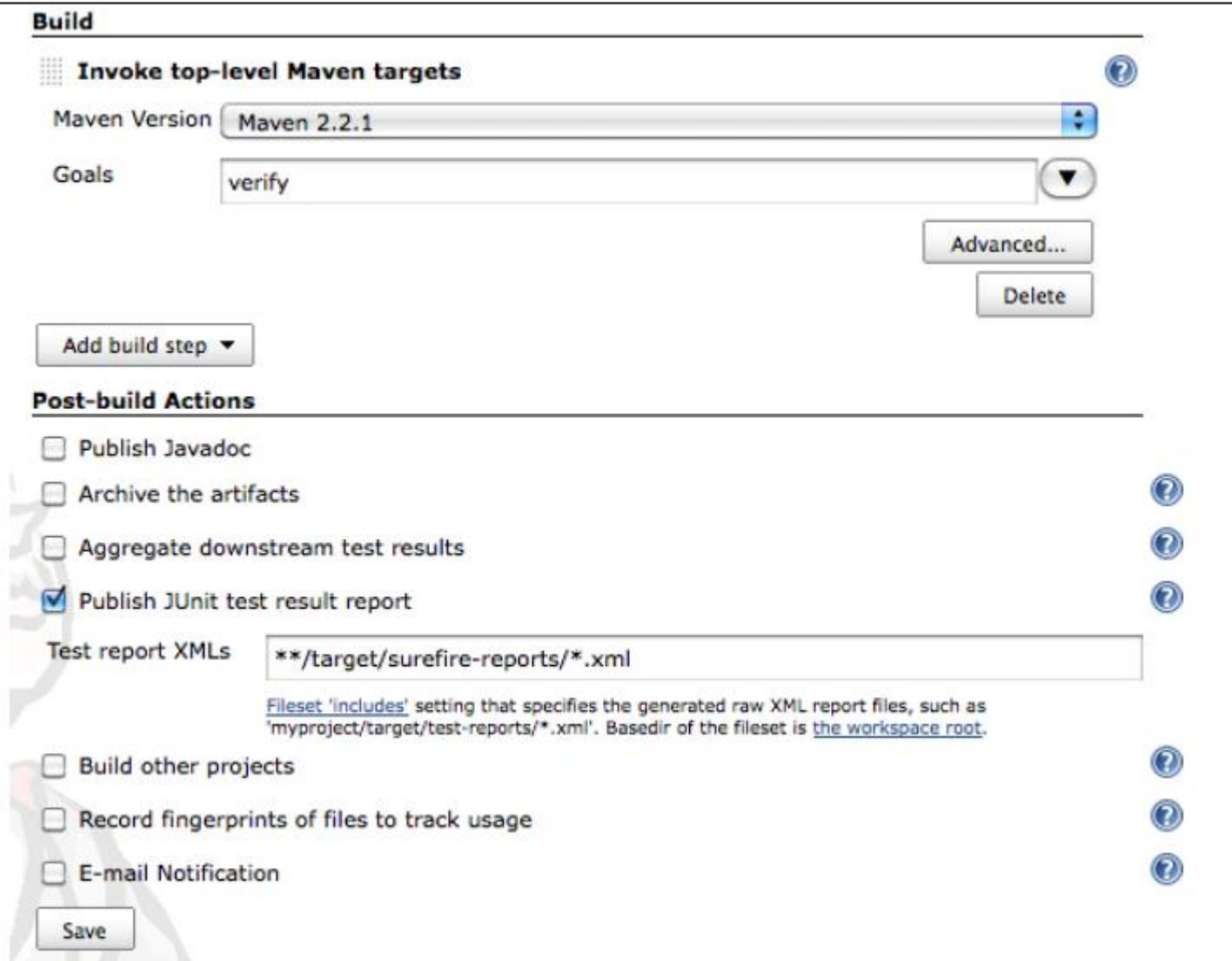
Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/\*.xml'. Basedir of the fileset is the workspace root.

Build other projects

Record fingerprints of files to track usage

E-mail Notification

Save



## A WORKING EXAMPLE WITH JUNIT

- For our example, we will consider using a maven job with JUNIT 3 and Maven
- All you need to do is to invoke either test or verify. Reports **are automatically published.**

## DISPLAY OF REPORTS

- Jenkins understands Multi modules in Maven and for Maven projects, shows results/modules
- For Free Style projects, display will be per package (Even if you invoke high level maven targets)
- At each level, you can drill down further
- Test Execution times are also reported. This is also an important metric for quality.

# DISPLAYING TEST RESULTS

- Jenkins differentiates between failed and unstable builds.
  - Unstable builds reflects on quality.
    - Talk to your Quality Assurance Team on metrics for your organization
- Reports are published to the home page of the project

The screenshot shows the Jenkins interface for the 'GameofLife' Maven project. At the top, there is a header bar with a back arrow, a globe icon, and the URL 'localhost:8080/job/GameofLife/'. Below the header is a black navigation bar with the Jenkins logo and the text 'Jenkins'. Underneath the navigation bar, the page title is 'Maven project GameofLife'. On the left side, there is a sidebar with several links: 'Back to Dashboard' (with a green arrow icon), 'Status' (with a magnifying glass icon), 'Changes' (with a document icon), 'Workspace' (with a folder icon), 'Build Now' (with a circular progress icon), and 'Delete Maven project' (with a red circle and slash icon). At the bottom right of the main content area, there is a 'JUnit' logo and a link 'View JUnit Reports'.

# DISPLAY OF REPORTS

## Test Result

0 failures , 1 skipped

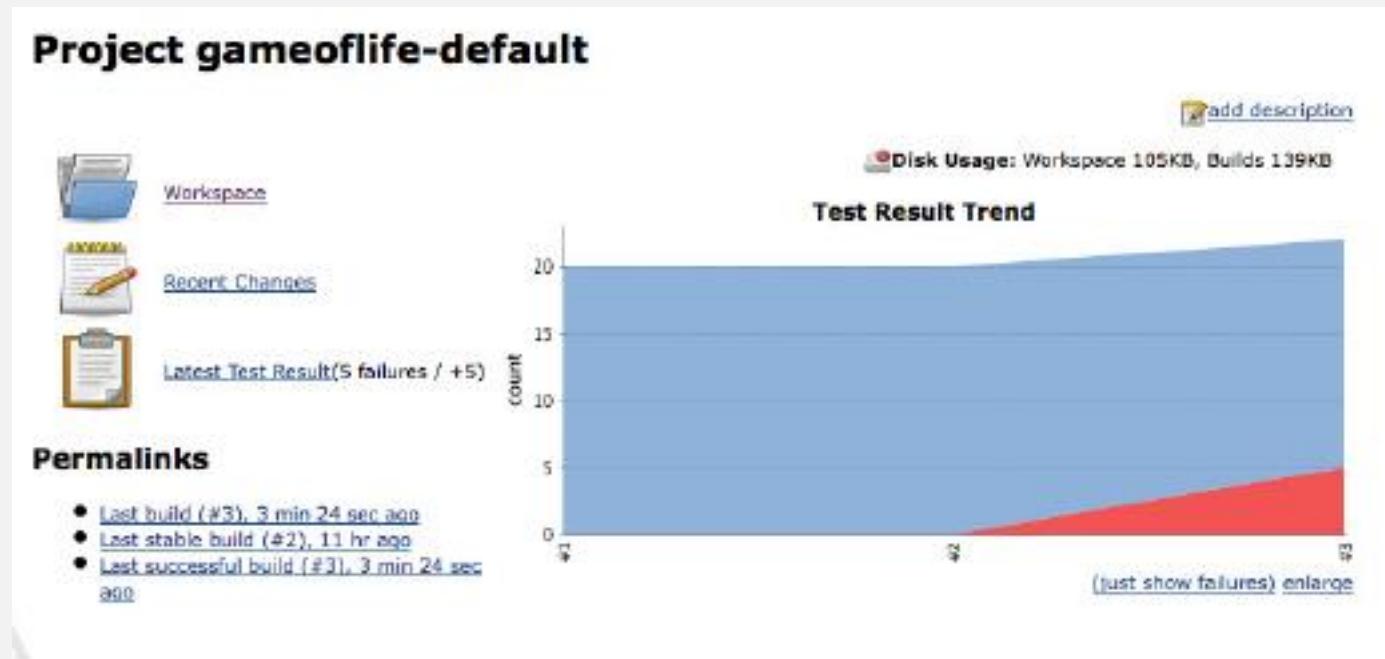
Module	Fail	(diff)	Total	(diff)
<a href="#">com.wakaleo.gameoflife:gameoflife-core</a>	0		51	+51
<a href="#">com.wakaleo.gameoflife:gameoflife-web</a>	0		7	+7

The screenshot shows a Jenkins test result page. At the top, there is a header bar with a back arrow, a forward arrow, a search icon, and a Jenkins skip tests button. Below the header, the Jenkins logo is on the left, and the breadcrumb navigation path is displayed: Jenkins > testCode > gameoflife-web > #3 > Test Results > com.wakaleo.gameoflife.webtests.controllers. A red oval highlights this breadcrumb path. To the right of the breadcrumb, a note says "Each now is a breadcrumb allowing you to drill down!". The main title is "Test Result : com.wakaleo.gameoflife.webtests.controllers". Below the title, it says "0 failures". The "All Tests" section contains a table with three rows:

Class	Duration	Fail	(diff)	Skip	(d)
<a href="#">WhenCreatingANewGame</a>	0.8 sec	0		0	
<a href="#">WhenDisplayingTheHomePage</a>	0 ms	0		0	
<a href="#">WhenSpawningANewGeneration</a>	35 ms	0		0	

# DISPLAYING TEST RESULTS

- All reports are displayed on project Home Page
- Jenkins also shows trend reporting on test case failures



# DISPLAYING TEST RESULTS

- For Maven style jobs, the results are initially shown per module

The screenshot shows a Jenkins test result page for a 'Test Result' job. The left sidebar contains links for Back to Project, Status, Changes, Console Output, History, Executed Males, Test Result (which is selected), Redeploy Artifacts, See Fingerprints, Previous Build, and Next Build. The main content area has a title 'Test Result' and a summary bar indicating 5 failures (+5) out of 22 tests (+2), which took 31 ms. A link to 'add description' is also present. Below this is a section titled 'All Failed Tests' with a table:

Test Name	Duration	Age
>>> com.civwithhudson.gameoflife.domain.UniverseTest.aLiveCellWithTwoNeighboursWillLiveInTheNextGeneration	0.0050	1
>>> com.civwithhudson.gameoflife.domain.UniverseTest.aLiveCellWithFourNeighboursAboveWillDieInTheNextGeneration	0.0010	1
>>> com.civwithhudson.gameoflife.domain.UniverseTest.aLiveCellWithFourNeighboursBelowWillDieInTheNextGeneration	0.0010	1
>>> com.civwithhudson.gameoflife.domain.UniverseTest.aDeadCellWithThreeNeighboursWillLiveInTheNextGeneration	0.0010	1
>>> com.civwithhudson.gameoflife.domain.UniverseTest.aUniverseCanHaveManySuccessiveGenerations	0.0020	1

Below this is another section titled 'All Tests' with a table:

Package	Duration	Fail	(diff)	Skip	(diff)	Total	(diff)
com.civwithhudson.gameoflife.domain	31 ms	5	+5	0		22	+2

# DISPLAYING TEST RESULTS

- Click on each test failure to see exact reason why the test failed.

## Regression

Failing for the past 1 build (Since #13 )

Took 1 ms.

Add description

### Error Message

Expected: is "...\\n...\\n...\\n"  
got: "...\\n\*.\*\\n\*.\*\\n"

### Stacktrace

```
java.lang.AssertionError  
Expected: is "...\\n...\\n...\\n"  
got: "...\\n*.*\\n*.*\\n"  
  
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:21)  
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:8)  
at com.wakaleo.gameoflife.domain.GameOfLifeTest.aDeadCellWithNoNeighboursShouldRemainDeadInTheNextGeneration(GameOfLifeTest.java:28)  
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)  
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)  
at java.lang.reflect.Method.invoke(Method.java:597)  
at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:44)  
at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)  
at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:41)  
at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:20)  
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:76)  
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:50)  
at org.junit.runners.ParentRunner$3.run(ParentRunner.java:193)  
at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:52)  
at org.junit.runners.ParentRunner.run(ParentRunner.java:111)
```

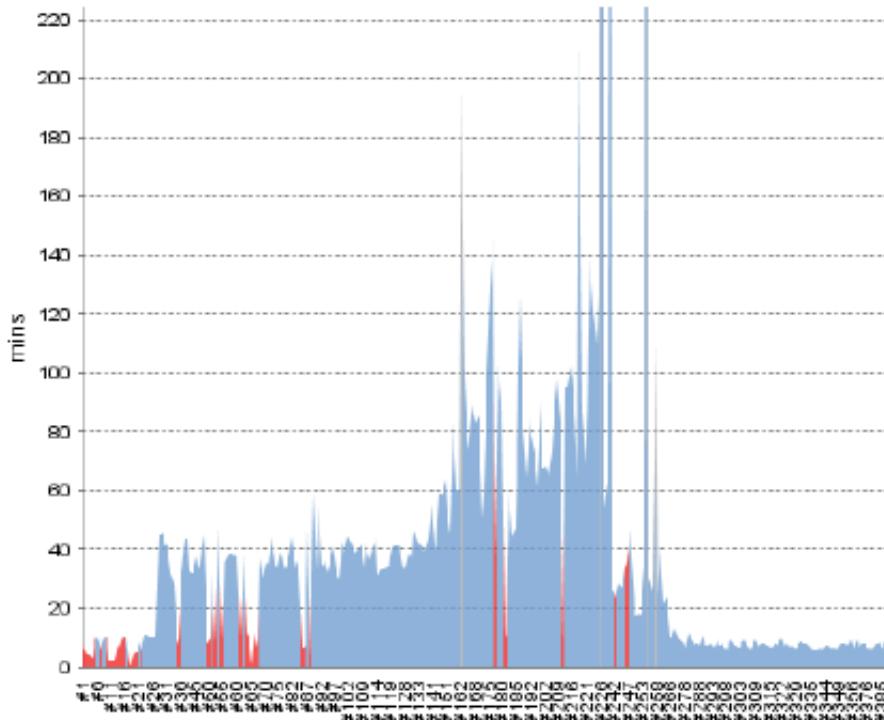
## DISPLAYING TEST RESULTS

- Measuring time for test execution is also important.
- Unit tests should complete real fast.
- Jenkins shows build trend times
  - Click Trend link of Build History (on left of screen)

# DISPLAYING TEST RESULTS

## Build Time Trend

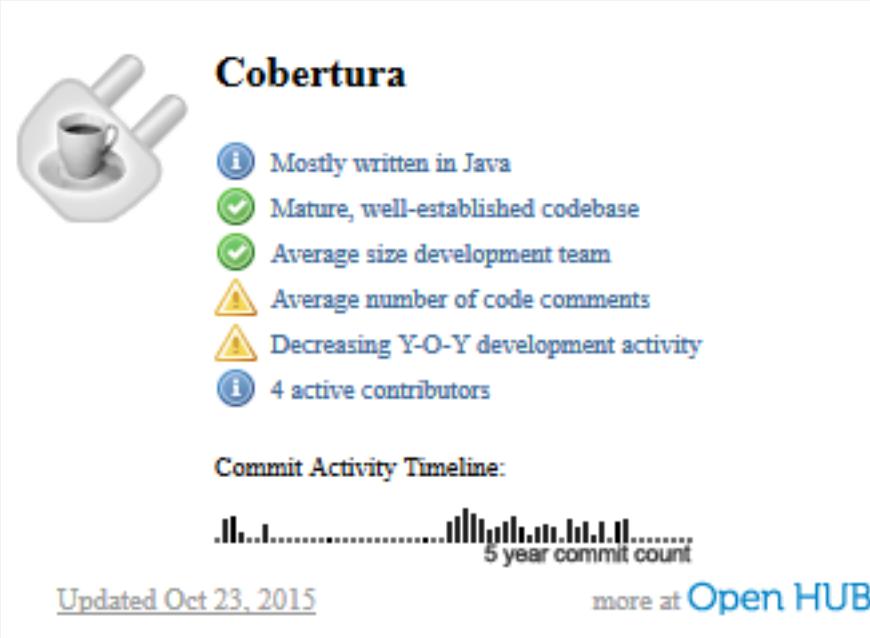
Build	Duration
#392	8 min 55 sec
#389	8 min 44 sec
#387	7 min 23 sec
#386	6 min 6 sec
#385	8 min 8 sec
#383	7 min 41 sec
#382	5 min 56 sec
#380	6 min 9 sec
#377	5 min 38 sec
#376	7 min 42 sec
#375	7 min 32 sec
#372	6 min 12 sec
#358	9 min 10 sec
#357	5 min 52 sec
#356	8 min 37 sec
#355	6 min 52 sec
#354	7 min 22 sec
#351	7 min 46 sec
#350	7 min 44 sec



# CODE COVERAGE

- Important Metric in software Development
- CPU and Memory Intensive
  - Run as separate job after unit and integration Testing

Ref:  
[https://www.openhub.net/p/cobertura/widgets/project\\_factoids\\_stats](https://www.openhub.net/p/cobertura/widgets/project_factoids_stats)



**Cobertura**

Mostly written in Java  
Mature, well-established codebase  
Average size development team  
Average number of code comments  
Decreasing Y-O-Y development activity  
4 active contributors

Commit Activity Timeline:



5 year commit count

Updated Oct 23, 2015 more at [Open HUB](#)



# CODE COVERAGE

- Cobertura integrates well with
  - Ant
  - Maven
  - Jenkins
- The coverage data has to be generated by you.
- How do they work?
  - Compiled class is instrumented (Cobertura stores this information in cobertura.ser)
  - Run tests against instrumented code (Data on how many lines were executed).
  - Format data to XML / HTML

# CODE COVERAGE

- 3 types of coverage
  - Statement coverage – percentage of statements tested.
  - Branch coverage – percentage of branches tested.
  - Basic path coverage – percentage of basic paths tested
- Cobertura supports all 3.

# INTEGRATE COBERTURA TO MAVEN

- Add **Cobertura-maven-plugin** to build section
- Invoke mvn **cobertura:cobertura**



- Define specific profiles for generating metrics.
- Maven plugin runs only in test phase, not in integration-test.

```
<project>
...
<build>
<plugins>
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>cobertura-maven-plugin</artifactId>
<version>2.5.1</version>
<configuration>
<formats>
<format>html</format></formats>
</configuration>
</plugin>
...
</plugins>
<build>
...
</project>
```

## INTEGRATE COBERTURA WITH ANT

- Tell Ant where Cobertura is installed.
- Set Classpath for ANT to pick up Cobertura'a Library
- Compile files to separate class directory (So that instrumented classes are not packaged)
- Run JUNIT test cases including instrumented classes
- Run Report against same.

# COBERTURA REPORTING

**Build**

Root POM pom.xml

Goals and options clean cobertura:cobertura -Pmetrics

**Advanced...**

Publish Cobertura Coverage Report

Cobertura xml report pattern `**/target/site/cobertura/coverage.xml`

This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use `**/target/site/cobertura/coverage.xml`). The path is relative to the module root unless you have configured your SCM with multiple modules, in which case it is relative to the workspace root. Note that the module root is SCM-specific, and may not be the same as the workspace root.  
Cobertura must be configured to generate XML reports for this plugin to function.

Consider only stable builds

Include only stable builds, i.e. exclude unstable and failed ones.

**Coverage Metric Targets**

Metric	Condition	Value	Unstable Condition	Value
Conditionals	Sunny	98	Cloudy	75
Lines	Sunny	98	Cloudy	75
Methods	Sunny	100	Cloudy	80
Packages	Sunny	100	Cloudy	95

**Add**

Configure health reporting thresholds.  
For the row, leave blank to use the default value (i.e. 80).  
For the and rows, leave blank to use the default values (i.e. 0).

**Build Step**

**Post Build**

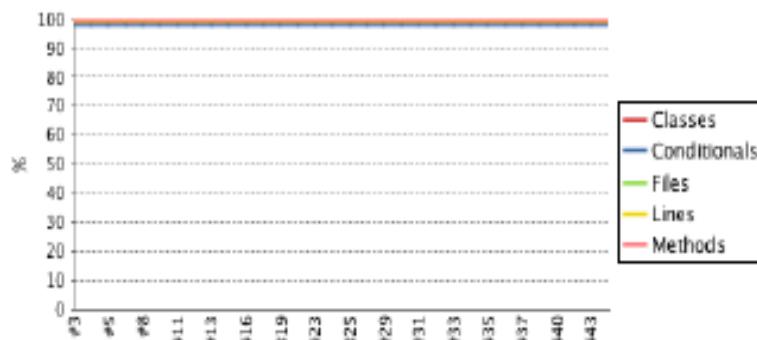
# INTERPRETING METRICS

## Code Coverage

[Cobertura Coverage Report >](#)

**com.wakaleo.gameoflife.domain**

Trend



Package Coverage summary

Name	Classes	Conditionals	Files	Lines	Methods					
com.wakaleo.gameoflife.domain	100%	5/5	98%	53/54	100%	5/5	100%	108/108	100%	36/36

Coverage Breakdown by File

Name	Classes	Conditionals	Lines	Methods
Grid.java	100%	1/1	100%	30/30
Cell.java	100%	1/1	75%	3/4
GridWriter.java	100%	1/1	100%	4/4
Universe.java	100%	1/1	100%	12/12
GridReader.java	100%	1/1	100%	4/4

- Reports can be drilled down.
- Lines fully covered are shown in green, red otherwise
- Yellow lines means not all conditions were tested.

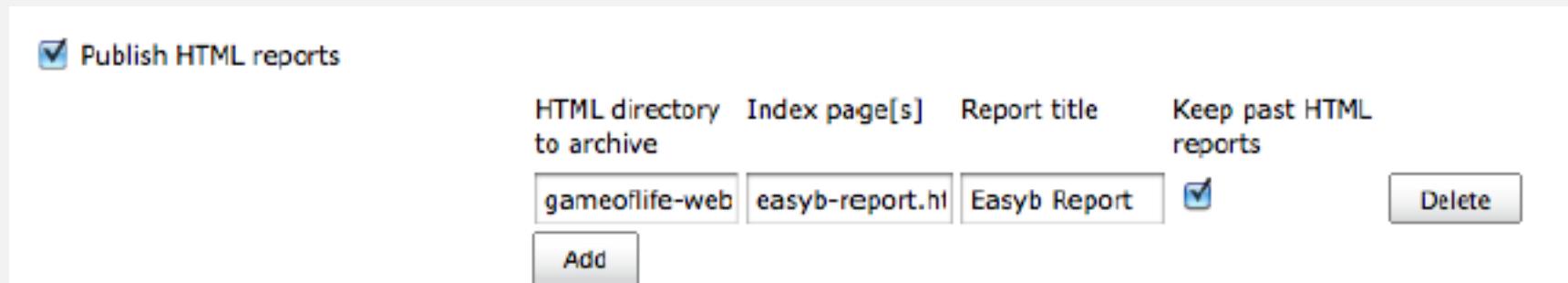
Ref: Jenkins the Definitive Guide

# AUTOMATED ACCEPTANCE TESTS

- Usually run last in pipeline. Why?
  - Developers do not usually care for these tests.
  - It is more for business persons as a proof rather than extensive testing
- Usually done using JUNIT, but more often BDD tools are increasingly used.
- BDD Pending Tests
  - Not yet done by Development team.
  - If Automated early on, you get info on what
    - Features implemented
    - Do not work
    - Have not been started
- Run as separate tasks/jobs

# AUTOMATED ACCEPTANCE TEST

- Example of a tool: Easyb
- Steps
  - Install HTML Publisher plugin
  - In post-Build actions, invoke publish HTML Reports



- Once done, link to report will be published in home page or build with a link to the report file.
- You can also use the DocLinks plugin to publish documents in other (non-HTML) format.

# PERFORMANCE TESTING



# PERFORMANCE TESTING

- Never Directly Load Tests on same CI server master
- Jmeter runs as SWING Application (and even as a proxy to prepare initial scripts)
- **No official JMETER Plugin for Jenkins**
- Use Ant Plugin for Jmeter.
  - Run JMETER task as an ANT Task
  - If running via Maven, use Maven-Ant Integration for doing the same
    - Run during integration-test phase.
- Install Performance Plugin which can understand JMETER Reports
- Set Job to run once every midnight.
- In Post-build action, publish performance test results.

## TIPS AND NOTES

- Unit Tests should usually finish within a minute
- Integration and functional tests should complete within 10 minutes.
- If this is not the case.
  - Consider adding additional HW power to your CI server.
    - Just move over to cloud
  - Run fewer integration/functional tests
    - Run more JUNIT individual test cases.
    - Only when all are fine, run integration/functional tests
  - Run tests in parallel
    - Split functional tests into subsets and run on different slaves and aggregate test results
    - TestNG/newer versions of Junit allow tests to be run in parallel

Q&A

# Questions???

## AGENDA FOR NEXT MODULE

- Managing Masters and Slaves
- Introduction to Distributed Builds



## FOOD FOR THOUGHT!!

- How would you share information between different builds?
- The test cases can easily number in 1000's. How do you ensure that test cases are run in parallel? Extending this, how would you distribute cases across nodes for faster execution?
- Take the same example as above, but let's give it a twist. Your company wants to test its web application on multiple browsers? How would you do the same? (HINT: Selenium, Distributed Builds)

## **MODULE 08**

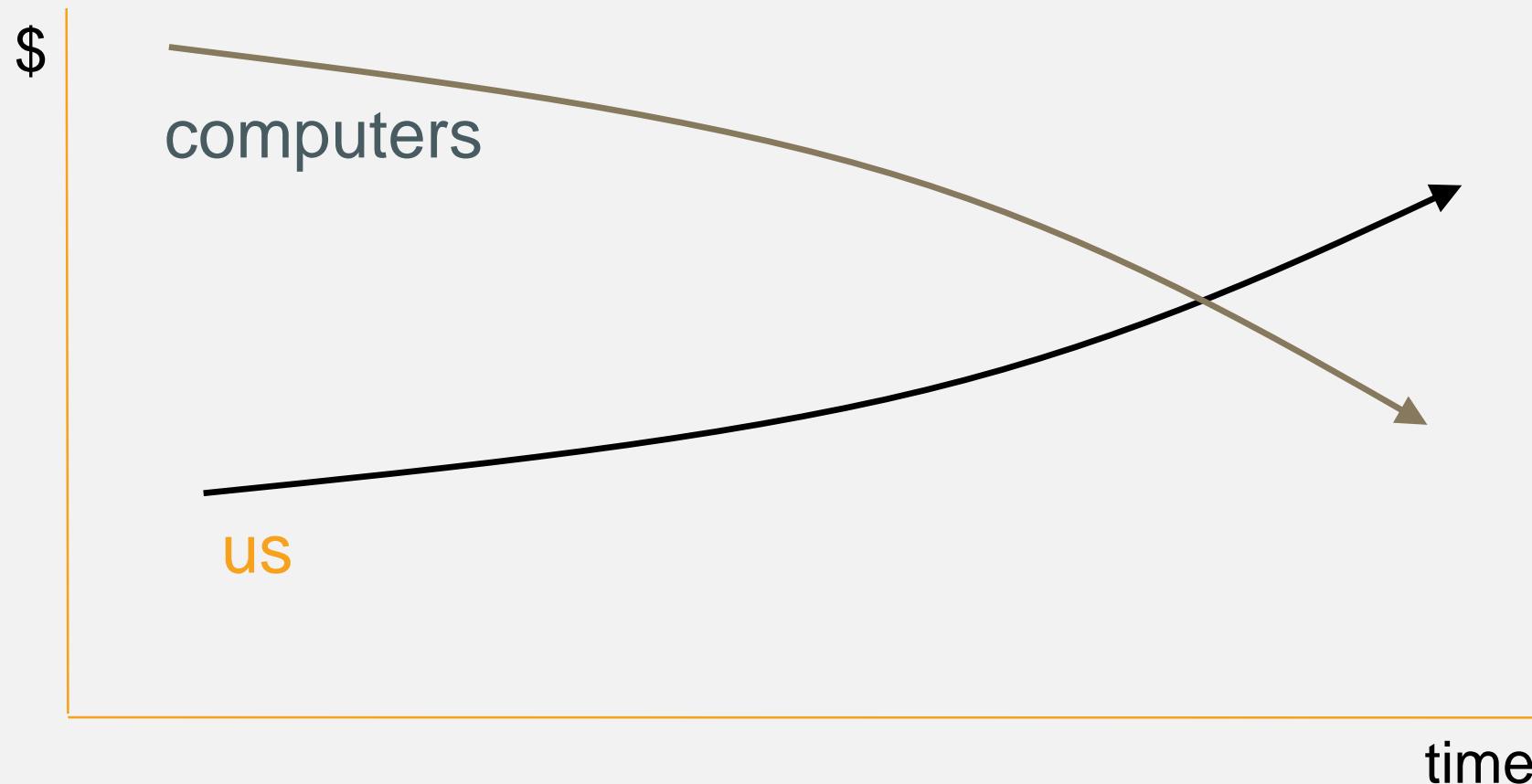
# **MASTERS, SLAVES AND DISTRIBUTED BUILDS**



# RISE OF CONTINUOUS INTEGRATION

- Offload from people, push to computers

**Ref:** Presentation in JavaOne by Kohsuke Kawaguchi/Jesse Glick



## CI NEEDS

- CPU Intensive
  - Worth spending money on a very good CI infrastructure
- Start with laptops and Desktops
  - Ever hated it when eclipse is slow to start with?
  - Ever wondered why no one wants to build (not just compile)?
- CI Servers do much more
  - Build
  - Test and more frequently
  - Static Code analysis
  - Reporting/Notification.....
  - Deployment.....

## WHY DISTRIBUTED BUILDS?

- **Why?**

- You use different environments
- You need to isolate environments
- Lot of stuff to do.

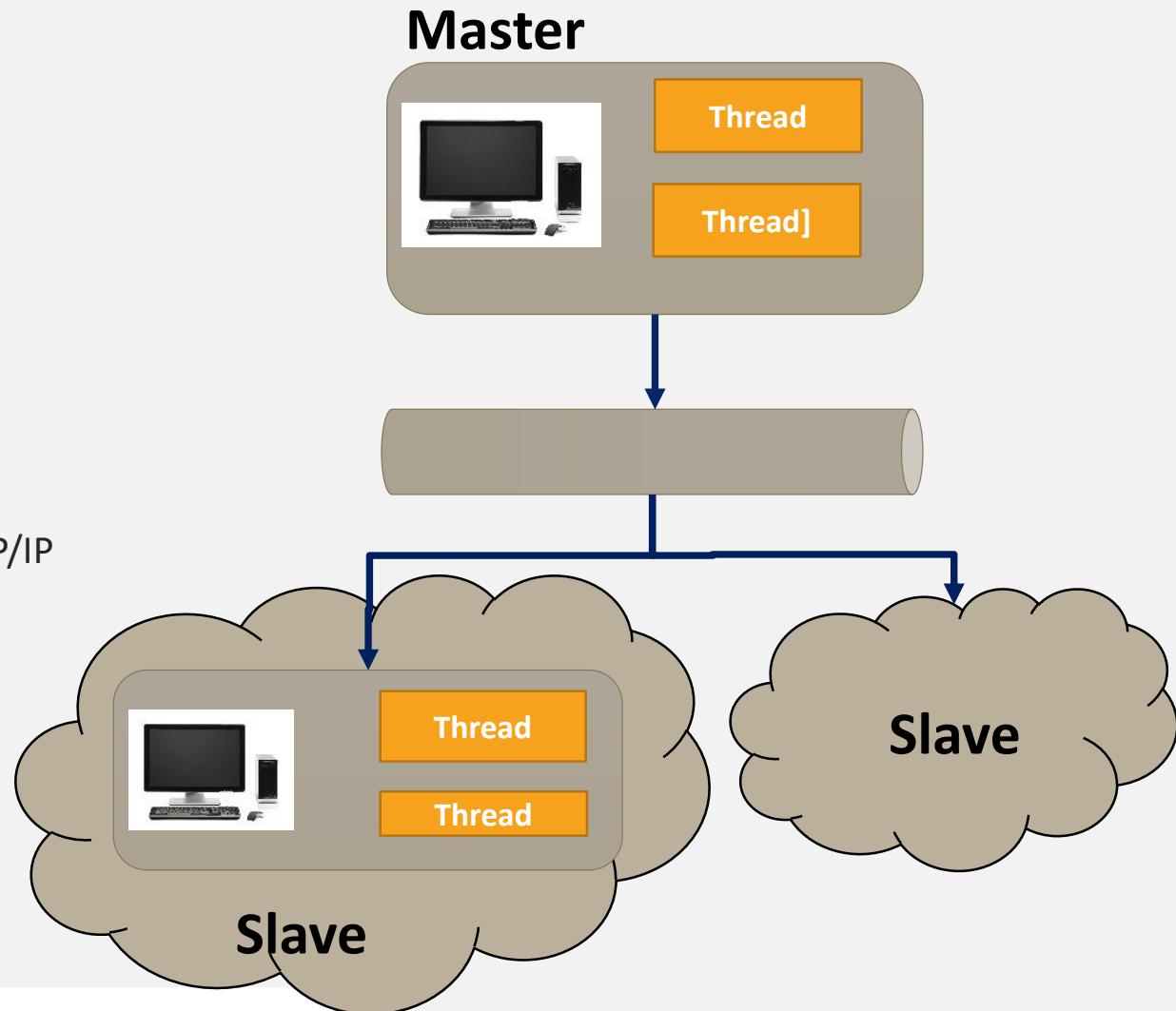
One computer simply  
**cannot**  
take this load

# CI AND VIRTUALIZATION

- Virtualization **does not** solve
  - Software installation problem
  - Node Failure issues
  - Remote Maintenance
  - .....

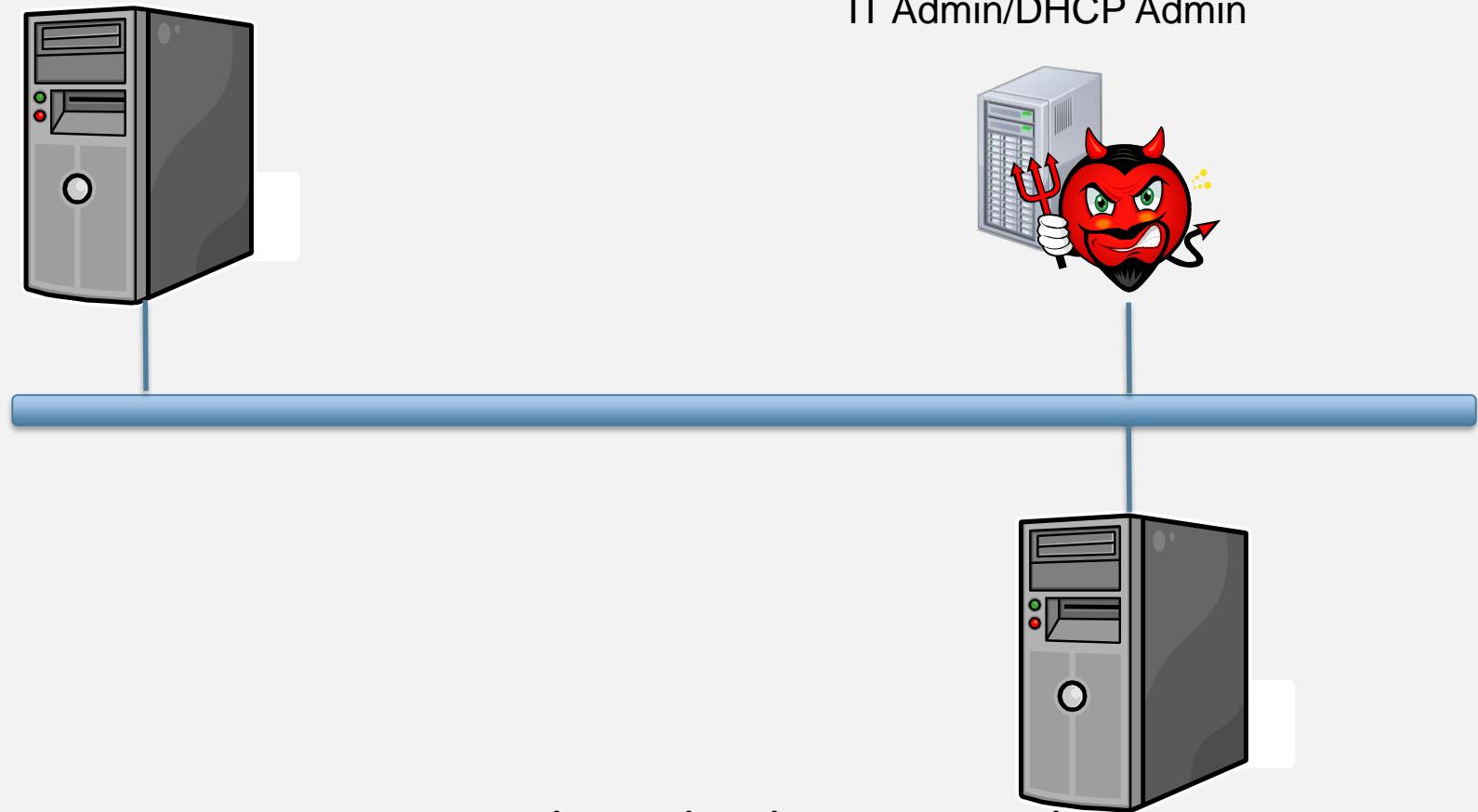
# HOW JENKINS/HUDSON SOLVES THIS

- **Master:**
  - A full-fledged jenkins server running on a machine.
  - Can build on own
  - (or) requests a slave to do the build.
  - Provides the user view: Dashboard.
- **Slave:**
  - Agent running on same machine (or) different machine
  - Can live anywhere as long as it's accessible to master via TCP/IP
- **Executor:**
  - Thread that does actual build.
  - Number of Executors = number of cores of CPU
- **Label:**
  - A String provided by slave to advertise its capabilities



# AUTOMATIC INSTALLATION OF SLAVES

- Hudson + PXE plugin
  - ISO images of OS



- Slaves
  - Power on, hit F12/F8
  - PC boots from network (PXE)
  - Choose OS from menu
  - Installs non-interactively

**Ref:** Pictures adapted from presentation in JavaOne by Kohsuke Kawaguchi/Jesse Glick

# AUTOMATIC INSTALLATION OF SLAVES

The screenshot shows the Jenkins Network Slave Installation Management configuration page. The URL in the browser is `localhost:8080/pxe/configure`. The left sidebar includes links for Back to Dashboard, Status, Console, and Configure. The main content area has sections for General (with a checked checkbox for "Respond to all requests"), Super-user account information (Root Username and Root Password fields), and Boot Configurations. A dropdown menu under "Add Boot Configuration" lists various options: CentOS, Chainboot from another TFTP server, OpenSolaris, Oracle VM Server, Parted Magic, RedHat/Fedora, Ubuntu, and VMWare ESXi.

# AUTOMATED INSTALLATIONS OF SLAVES

- Supports Open Solaris, Ubuntu, CentOS, Fedora
  - Trivial with most Linux
  - Partial Windows, too
    - Prefer to use \*nix

# SLAVE INSTALLATION

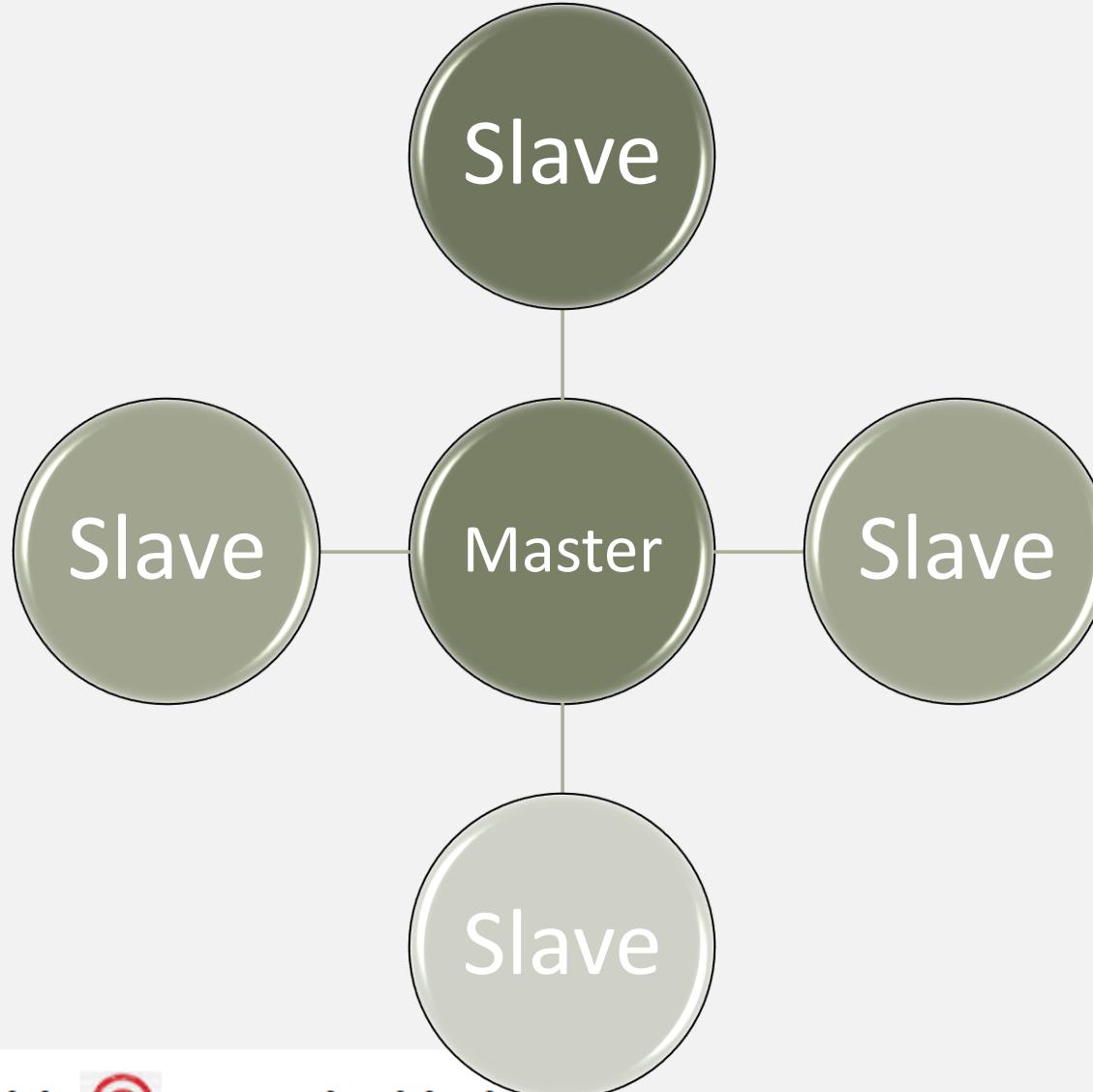
- This is not the only way!
  - You could always create Docker Images
    - Do require SSHD, Java, GCC, maven, ant and other stuff required for your build
    - Do require a user account for Jenkins to connect
    - Use Chef/Puppet to push out image to target box and start the same
    - Configure the Docker Plugin in Jenkins to connect to the slave.
    - Docker is now the cloud provider for Jenkins.

## SLAVE TYPES

- Managed Slaves
  - The Slave Agent of the Managed Slaves are in the control of Master.
  - If the Slave Agent dies, Jenkins restarts it when its services are required
- Unmanaged Slaves
  - Typically started with JNLP invocation
  - If Slave dies, it has to be manually started.

# MASTER SLAVE

- Master
  - Serves HTTP requests
  - Stores all important info
- Slaves
  - 170KB single JAR
  - Assumed to be unreliable
  - Scale to at least 100
- Link
  - Single bi-di byte stream
  - No other requirements

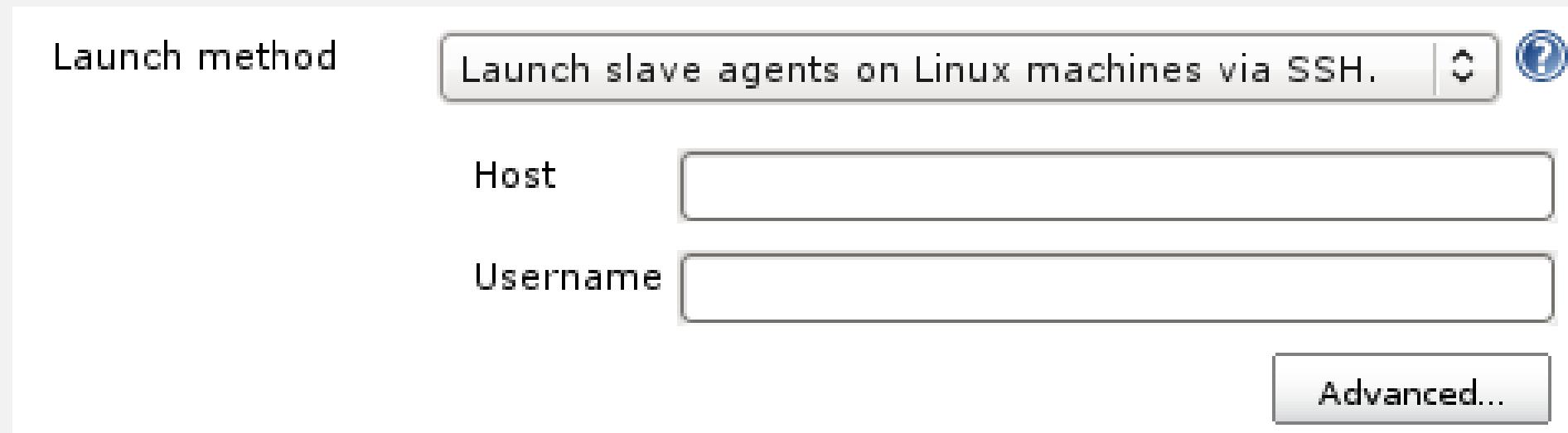


# MASTER SLAVE COMMUNICATION

- Master-Slave communication
  - Bi-Directional
  - TCP/IP Socket Communication
- When the Slave Agent was invoked by Jenkins, a secret key is passed as parameter
  - Java -jar slave.jar -credential <secretKey> -url <Master URL>
- Slave agent tries to establish a HTTP connection to Master using key in header
- Jenkins responds with the port number on which the SlaveAgentListener is listening for Master-Slave communication.
- Slave Agent requests a Secure socket connection with that port number and credentials.
- If accepted, a secure communication channel is established between master and Slave

# SLAVE CONFIGURATION

- For Unix slaves, via SSH
  - Only need SSH and JRE on slaves
  - We just need a host name
- For Windows, it is via DCOM
  - Need admin user name/Password
  - Works even with Unix Masters



# ADDING SLAVES

- Click on Manage Jenkins->Manage Nodes->New Node.
- Pay attention to labels.

The screenshot shows the Jenkins 'New Node' configuration page. The URL in the browser is `localhost:8080/computer/createlnodem`. The page has a sidebar with links like 'Back to Dashboard', 'Manage Jenkins', 'New Node', and 'Configure'. Below the sidebar, there are sections for 'Build Queue' (empty) and 'Build Executor Status' (listing 1-7 Idle). The main form fields are:

- Name: NODE1
- Description: (empty)
- # of executors: 1
- Remote root directory: (highlighted in red with error message: "Remote directory is mandatory")
- Labels: (empty)
- Usage: Utilize this node as much as possible
- Launch method: A dropdown menu showing:
  - Launch slave agents on Unix machines via SSH (selected)
  - Launch slave agents via Java Web Start
  - Launch slave via execution of command on the Master
  - Let Jenkins control this Windows slave as a Windows service
  - Mock Slave Launcher

## STARTING SLAVES- THE QUICK AND DIRTY MODE

- Open Browser on slave machine and point to master url (<http://yourjenkinserverurl>)
- Go to Manage Jenkins->Manage Nodes
  - Click on newly created slave
  - Click on Launch to launch agent from Browser (Java plugin for browser should be enabled – not recommended)
- Instead of browser launch,
  - javaws [http://yourjenkinsserverurl/computer/Jenkins Slave/slave-agent.jnlp](http://yourjenkinsserverurl/computer/Jenkins%20Slave/slave-agent.jnlp) (or)
  - java -jar slave.jar –jnlpUrl [http://yourjenkinsserverurl/computer/Jenkins Slave/slave-agent.jnlp](http://yourjenkinsserverurl/computer/Jenkins%20Slave/slave-agent.jnlp)

# STARTING SLAVES

- Ways to Start slaves:
  - The master starts the slave agents via ssh
  - Starting the slave agent manually using Java Web Start
  - Installing the slave agent as a Window service
  - Starting the slave agent directly from the command line on the slave machine from the command line

# STARTING SLAVES – METHOD 1

- **The Master Starts the Slave Agent Using SSH**
  - Best suited on Unix (and Linux and any \*ix variants)
  - Support for SSHD
  - Jenkins has own SSH Client
  - Slave created as Dumb slave

The screenshot shows the Jenkins web interface at [localhost:8080/jenkins/computer/new](http://localhost:8080/jenkins/computer/new). The left sidebar includes links for Back to Dashboard, Manage Jenkins, New Node (which is highlighted in blue), and Configure. The main content area is titled 'Node name' with the value 'BUILD\_SLAVE'. Below this, there are two radio button options: 'Dumb Slave' (selected) and 'Mock Slave'. The 'Dumb Slave' description states: 'Adds a plain, dumb slave to Jenkins. This is called "dumb" because Jenkins doesn't provide higher level of integration with these slaves, such as dynamic provisioning. Select this type if no other slave types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.' The 'Mock Slave' description states: 'A slave using a new temporary directory for its filesystem root and running on localhost.' At the bottom right of the configuration form is an 'OK' button.

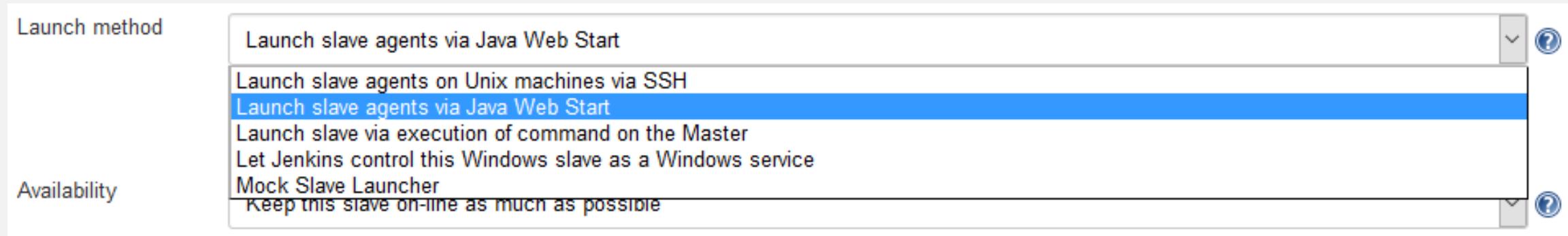
# STARTING SLAVES – METHOD 1

- The Master Starts the Slave Agent Using SSH

Name	BUILD_SLAVE	(?)
Description		(?)
# of executors	1	(?)
Remote root directory	/var/home/jenkins/workspace	(?)
Labels	BUILD_SLAVE	(?)
Usage	Utilize this node as much as possible	(?)
Launch method	Launch slave agents on Unix machines via SSH	(?)
Host		
Credentials	<input type="button" value="▼"/> <input type="button" value="Add"/>	(?)
		<input type="button" value="Advanced..."/>
Availability	Keep this slave on-line as much as possible	(?)

## STARTING SLAVES – METHOD 2

- **Starting the Slave Agent Manually Using Java Web Start**
  - Started from Slave machine
  - When Master cannot reach Slave directly due to firewall.
  - Same as above except launch method is set to launch via Web Start:



## STARTING SLAVES – METHOD 2

- Starting the Slave Agent Manually Using Java Web Start
  - Log on to slave machine
  - Start via browser (or) from command line

## STARTING SLAVES – METHOD 2



### Slave BUILD\_SLAVE

Connect slave to Jenkins one of these ways:

-  Launch Launch agent from browser on slave
- Run from slave command line:

```
java -jar slave.jar -jnlpUrl http://localhost:8080/jenkins/computer/BUILD_SLAVE/slave-agent.jnlp -secret  
b12444f0b6bcb9726e70d1b217c259839ba27eba5a0c404c40f679aba59ac3e7
```

Created by [The Great Admin](#)

### Projects tied to BUILD\_SLAVE

## STARTING SLAVES – METHOD 2

- **Starting the Slave Agent Manually Using Java Web Start**
  - Javaws is a low and effective way to communicate.
  - Communication can be blocked due to firewall
    - In such cases, ensure that TCP port for JNLP slaves is set to fixed,
    - Open Firewall port on specified port number

# STARTING SLAVES – METHOD 3

- Installing a Jenkins Slave as a Windows Service

The screenshot shows the Jenkins slave configuration interface. The 'Launch method' dropdown is set to 'Let Jenkins control this Windows slave as a Windows service'. A note below explains that this method uses DCOM and may have subtle problems, suggesting instead to use Java Web Start. The configuration includes fields for 'Administrator user name', 'Password', 'Host', and 'Run service as' (set to 'Use Local System User'). An 'Advanced...' button is at the bottom right.

Launch method

Let Jenkins control this Windows slave as a Windows service

This launch method relies on DCOM and is often associated with [subtle problems](#). Consider using **Launch slave agents using Java Web Start** instead, which also permits installation as a Windows service but is generally considered more reliable.

Administrator user name

Password

Host

Run service as

Use Local System User

Advanced...

## STARTING SLAVES – METHOD 4

- Starting the Slave Agent in headless mode
  - Useful when no UI is available.
  - Requires slave.jar found under: JENKINS\_HOME/war/WEB-INF
  - Run:

```
java -jar slave.jar -jnlpUrl http://server/jenkins/computer/windows-slave-1/slave-agent.jnlp
```

- If authentication is required, pass –auth username:password

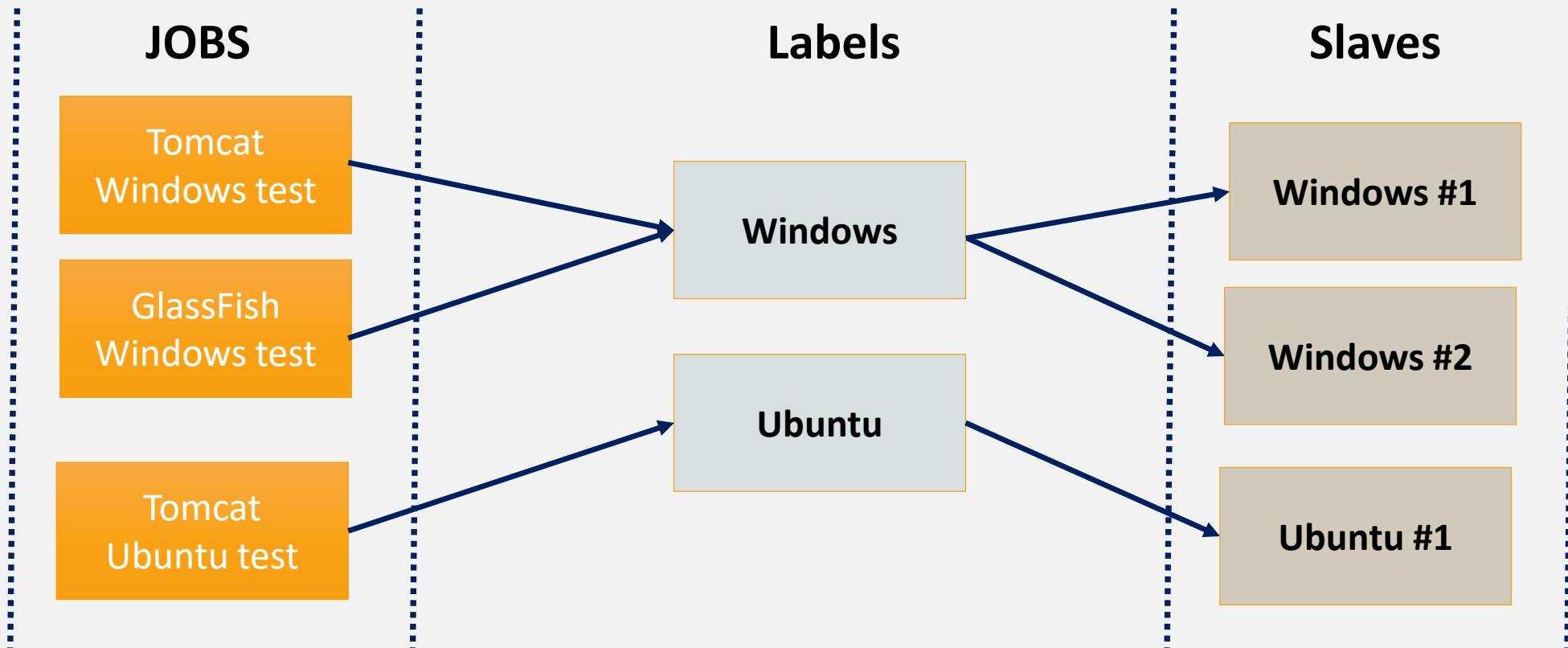
# SLAVE CONFIGURATION

- Not Covered, but possible
  - WMI interface
- Why Not covered?
  - Too many subtle issues (ref: <https://wiki.jenkins-ci.org/display/JENKINS/Windows+slaves+fail+to+start+via+DCOM>)
  - No UI
  - Limited functionality
  - Complexity of opening Windows firewall.

**LONG AND SHORT MORAL**  
Stick to homogeneous environments.

# LABELS

- A Label is a group of slaves
- Tie jobs to labels



# STICKY BUILDS

- Consistency
- Jobs to be on same slave for
  - Faster checkouts
  - Consistent Results
  - Minimize disk consumption
- In project configuration, you can specify restrictions based on labels.
  - (Logically) Available only when at least slave node is present

# STICKY BUILDS

- Expression is based on expression language
- To specify just one node, just specify the node name.
  - This is not a recommended best practice except when the number of nodes are small.

localhost:8080/job/TEST/configure

Jenkins ▶ TEST ▶ configuration

Restrict build execution causes

Sidebar Links

Prepare an environment for the run

Disable Build (No new builds will be executed until the project is re-enabled.)

Execute concurrent builds if necessary

Restrict where this project can be run

Label Expression

If you want to always run this project on a specific node/slave, just specify its name. This works well when you have a small number of nodes.  
As the size of the cluster grows, it becomes useful not to tie projects to specific slaves, as it hurts resource utilization when slaves may come and go. For such situation, assign labels to slaves to classify their capabilities and characteristics, and specify a boolean expression over those labels to decide where to run.

# SLAVE UTILIZATION STRATEGIES

- “Utilize this slave as much as possible”
  - most commonly used option
- “Leave this machine for tied jobs only”
  - Only jobs restricted to this node run on this node. Typically, load and performance tests jobs are tied to a very specific node and that node is tied only to these jobs.
- “Take this slave on-line when on demand and off-line when idle”
  - Only when load goes beyond a certain limit, will these nodes be used. Think of them as spare nodes.
  - In demand delay – delay in minutes for jobs to wait in queue before it is used.
  - Idle delay – time in minutes slave is idle before becoming off-line

# SLAVES IN THE CLOUD

- AMAZON EC2
  - Pay as you go
  - Instances launch in high speed
  - Programmable API
  - SLA from Amazon (See the fine print in the contract)
  - Data is still inside your firewall
  - Code checkout or build artifacts are relatively slower (or faster, depending on existing infra in place)
- Jenkins EC2 Plugin
  - Based on typical
  - Provisions EC2 slaves on demand
  - Picks right AMI on demand
  - Starts slave agents
  - Shutdowns unused instances

# JENKINS IN THE CLOUD

- Jenkins powered by Bitnami
  - Sold by: BitRock Inc.
  - Pre-configured, ready to run image for running Jenkins on Amazon EC2.
  - Essentially running master in cloud also.
  - See: <https://aws.amazon.com/marketplace/pp/B008AT8BYK>
- Other Alternatives
  - Eucalyptus
  - CloudBees DEV@Cloud

# SLAVE MANAGEMENT

- Keeping slaves consistent is a good thing
  - Particularly hard on heterogeneous environment
- General system administration tasks
  - Network configuration
  - Package installations for native tools
  - Tools like Chef, Puppet or cfEngine help.. !!!!
- Install build tools in the cluster
  - Prepare tools on one file system
  - rsync to everywhere
- Consider Docker for image management and separate Docker/Chef processes in place for keeping slaves in sync.

# DEMO & LAB

Q&A

# Questions???

## AGENDA FOR NEXT MODULE

- Recap
- Jenkins Maintenance and Administration
- Jenkins Updates
- Backup of Jenkins
- CLI interface to Jenkins
- Best Practices



## FOOD FOR THOUGHT!!

- How would you share information between different builds?
- The test cases can easily number in 1000's. How do you ensure that test cases are run in parallel? Extending this, how would you distribute cases across nodes for faster execution?
- Take the same example as above, but let's give it a twist. Your company wants to test its web application on multiple browsers? How would you do the same? (HINT: Selenium, Distributed Builds)

# **MODULE 09**

## **MANAGING JENKINS**



# ADMINISTERING JENKINS

- All of Jenkins administration can be found in the Manage Jenkins page. Since this is a general UI based administration, we will not cover this in any great depth other than pointing out main components.

## Manage Jenkins

 [Configure System](#)  
Configure global settings and paths.

 [Configure Global Security](#)  
Secure Jenkins; define who is allowed to access/use the system.

 [Manage Plugins](#)  
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

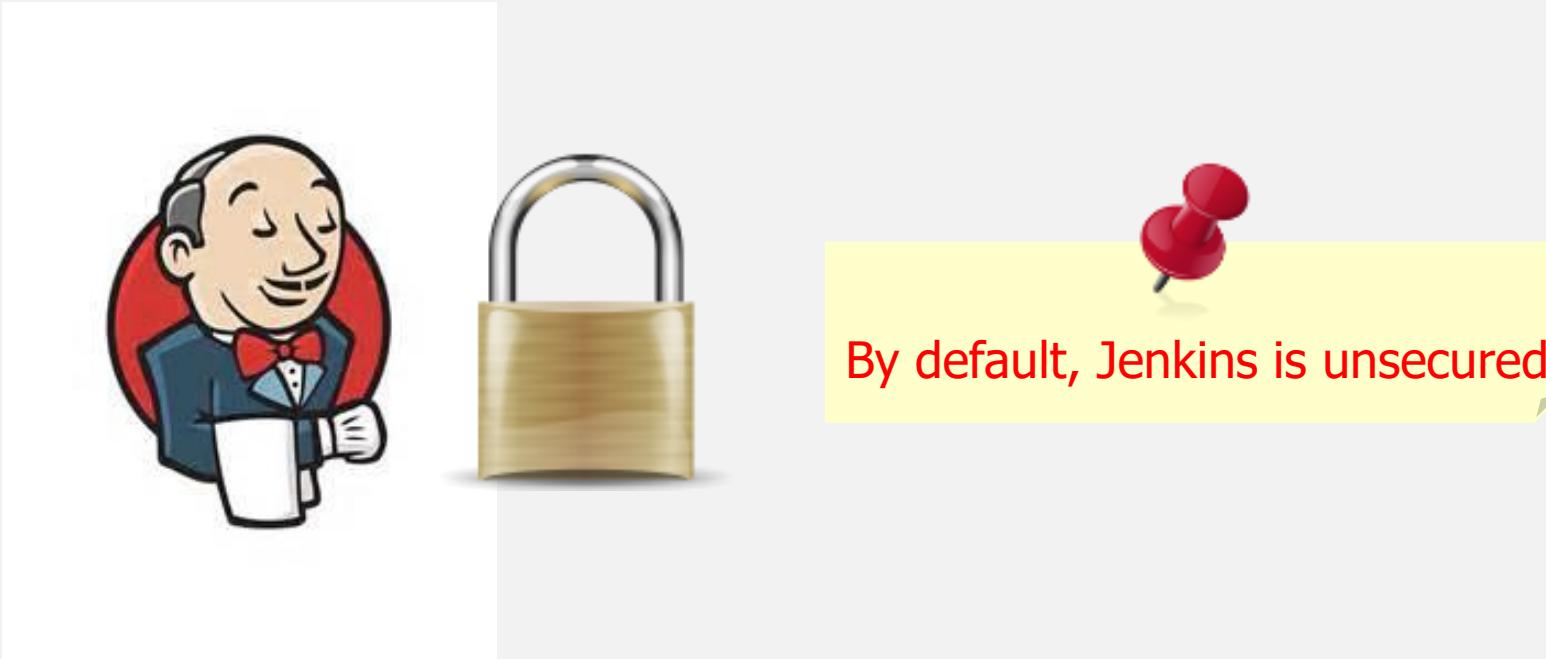
 [System Information](#)  
Displays various environmental information to assist trouble-shooting.

 [System Log](#)  
System log captures output from `java.util.logging` output related to Jenkins.

 [Load Statistics](#)  
Check your resource utilization and see if you need more computers for your builds.

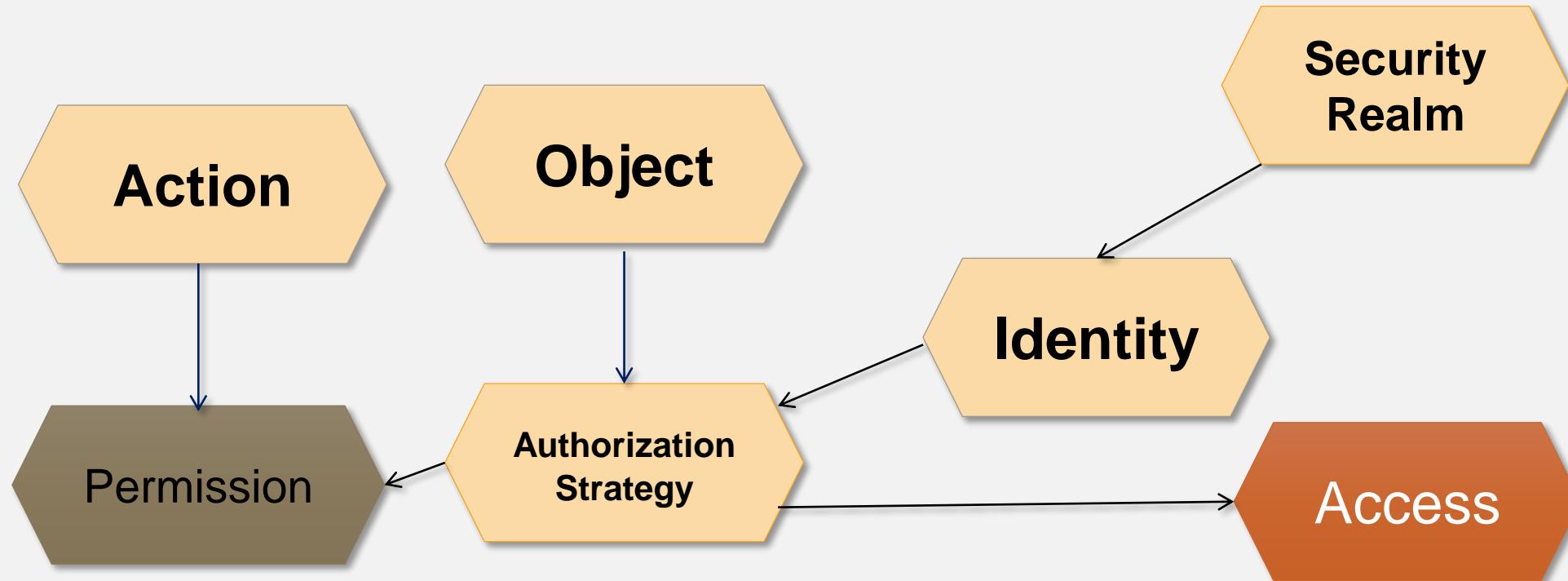
 [Manage Nodes](#)  
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

# JENKINS SECURITY



# JENKINS SECURITY ARCHITECTURE

- Security Realm provides user identity
- Authorization Strategy provides user's permissions for each object.
- Actions can require a specific permission to be performed.



# JENKINS SECURITY

- Server Security
  - OS Dependent
    - Linux
    - Windows
  - Servlet Container
    - Tomcat
    - Jetty
    - JBOSS
    - Winstone (java –jar jenkins.war)
    - etc.

# JENKINS SECURITY

- Server Security Checklist
  - Server patches and hotfixes
  - Configure server firewall to prevent unauthorized access
  - Lock down remote server access
    - Remote Desktop Access
    - SSHD shutdown
  - Run server not as Local System, but with very restricted access
  - Expose application server from behind Apache HTTPD or NGINX
    - Port Redirection
    - URL Re-writes

# JENKINS SECURITY REALMS

## WHAT IS A SECURITY REALM

- Core Jenkins extension point for Authentication
- Responsible for validating user identity
- Can only select one.
- Default for clean install:
  - None

## WHAT IS AVAILABLE

- Core
  - None
  - Unix PAM
  - Internal DB
  - Legacy Container
- Open Source Plugins
  - Active Directory
  - CAS v1
  - MySQL DB
  - CollabNet
  - Crowd
  - OpenID SSO
  - .....

# JENKINS AUTHORIZATION

## WHAT IS A SECURITY REALM

- Core Jenkins extension point for Authorization
- Responsible for deciding the permissions available to users.
- Can only select one.
- Default for clean install:
  - Unsecured

## WHAT IS AVAILABLE

- Core
  - Global Matrix
  - Project Matrix
  - Logged in user can do anything
  - Legacy Authorization
- Open Source Plugins
  - CollabNet
  - Role strategy
  - SourceForge Enterprise Edition
- CloudBees' Plugins
  - RBAC

## JENKINS PERMISSION

- The fine-grained activities that can be secured within Jenkins
- Some permissions aggregate others, e.g. Global Admin implies all other standard permissions
- Plugins can define their own permissions for their own actions

# JENKINS PERMISSION

- Overall
  - »Administer
  - »Read
- Slave
  - »Configure
  - »Delete
- Job
  - »Create
  - »Delete
  - »Configure
  - »Read
  - »Build
  - »Workspace
- View
  - »Create
  - »Delete
  - »Configure

## WHAT IS AVAILABLE BY DEFAULT?

# JENKINS AUTHENTICATION PLUGINS

- Key Features
  - Support Signup
  - Group Support
  - Support Group Lookup
  - Logout
- Based on what plugin supports which of the above features, authorization strategy is to be decided



Not all plugins support all features. Hence security should be one of first considerations when installing Jenkins

# JENKINS ACTIVE DIRECTORY

- Authenticates the username and the password through Active Directory
- Actually multiple implementations under the hood and one is chosen based on your environment



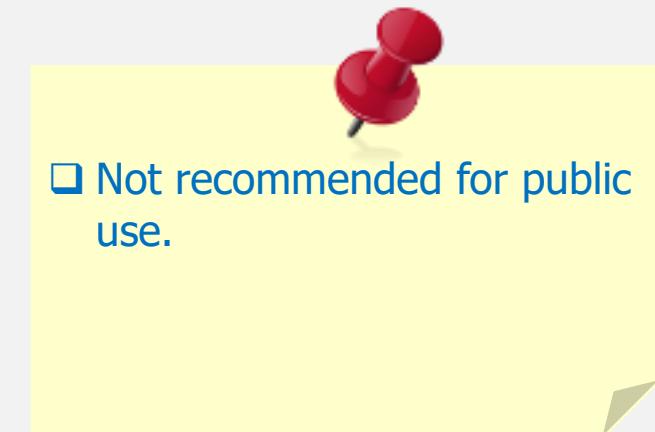
Feature Supported
Signup
Group Details
Group Lookup
Sign out

 A red pushpin is pinned to a yellow sticky note. The note contains two blue checkmark icons followed by text: "Jenkins does not require to be on Windows." and "Almost always DNS has to be configured correctly."

# JENKINS OWN DATABASE

- Authenticate against internal user name/password database

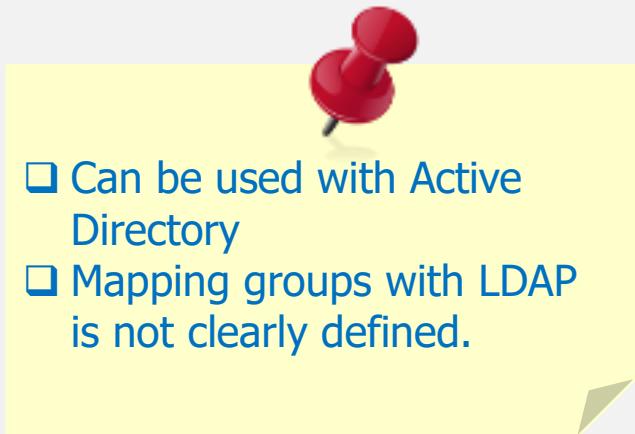
Feature Supported
Signup
Group Details
Group Lookup
Sign out



# JENKINS LDAP

- Authenticates the username and the password through LDAP
- Every LDAP server is different
  - Very flexible
  - Harder to configure than some of the other providers

Feature Supported
Signup
Group Details *
Group Lookup *
Sign out



# OTHER AUTHENTICATION PROVIDERS

- OpenID
- Unix PAM
- .....



Feature Supported	Active Directory	Atlassian Crowd	Jenkins Own DB	LDAP	OpenID	UNIX PAM
Signup			✓			
Group Details	✓	✓		✓	✓	✓
Group Lookup	✓			✓		✓
Sign out	✓	✓	✓	✓	✓	✓

# JENKINS MATRIX STRATEGY

- Each row represents one user/group
  - If Authentication plugin does not support groups, then each user has to be represented on another row
- Each Column is a permission
- Supports
  - Per project Configuration (Project Based Matrix)
  - Per object Configuration
  - Local group Definition (via Roles Strategy Plugins)
  - Delegate Management

# JENKINS SECURITY

- Managed via Configure Global Security

The screenshot shows the Jenkins 'Configure Global Security' configuration page. At the top, there is a 'Disable remember me' checkbox. Below it, under 'Access Control', the 'Security Realm' is set to 'Jenkins' own user database'. There is also an unchecked checkbox for 'Allow users to sign up'. Under 'Authorization', the 'Project-based Matrix Authorization Strategy' is selected. A matrix table below defines permissions for 'User/group' (admin, Anonymous) across various Jenkins roles (Overall, Credentials, etc.).

User/group	Overall	Credentials	...								
Admin	Checkmark	Checkmark	Checkmark	Checkmark	Checkmark	Checkmark	Checkmark	Checkmark	Checkmark	Checkmark	Checkmark
Anonymous	Blank	Blank	Blank	Blank	Blank	Blank	Blank	Blank	Blank	Blank	Blank

At the bottom, there are 'Save' and 'Apply' buttons.

## TIPS

- When first installing Jenkins (Pre Jenkins 2), do the following:
  - Allow any user to do anything
  - Select Allowing of signup and Jenkins own user database
  - Create an user called admin who has full rights.
  - Log out and log back in again as admin
  - Ensure you have matrix strategy enabled and admin has full rights.
  - No reset Anonymous to no access
- In Jenkins 2, at time of install, Jenkins defaults to using own user database and lets any logged in user do anything.

# UPGRADING JENKINS – WAR FILES

- The quick and dirty way:
  - Just take a physical backup (Just in case....)
  - Replace the WAR File.
    - Either hot or cold deploy.
- If you start Jenkins with
  - Java –jar Jenkins.war....
- There will be a link to upgrade jenkins

# UPGRADING JENKINS

- If you run Jenkins as a standalone war (java –jar jenins.war), the Manage Jenkins page will have a button.
- This is supposed to work on Windows too when installed as a service.
- For Debian packages, the below commands will work.
  - aptitude update
  - aptitude install jenkins
- The safest way is to enable scripting for updates.

## UPDATE JENKINS.WAR - \*X

- cd /tmp  
rm -f jenkins.war.backup  
cp /usr/local/jboss/server/default/deploy/jenkins.war /tmp/jenkins.war.backup  
rm -f jenkins.war  
wget http://mirrors.jenkins-ci.org/war/latest/jenkins.war  
nohup /usr/local/bin/copywar.sh > /tmp/copywar.out 2>&1 &

### Contents of copywar.sh

```
#!/bin/bash  
sleep 20  
cp /tmp/jenkins.war /usr/local/jboss/server/default/deploy
```



Adjust paths as required  
in your local application.

# UPDATE JENKINS AS WINDOWS SERVICE

```
@rem --[ This code block detects if the script is being running with admin PRIVILEGES If it isn't it pauses and then quits]----  
-  
@echo OFF  
AT > NUL  
IF %ERRORLEVEL% EQU 0 (  
    ECHO Administrator PRIVILEGES Detected!  
) ELSE (  
    echo.  
    echo ##### ERROR: ADMINISTRATOR PRIVILEGES REQUIRED #####  
    echo This script must be run as administrator to work properly!  
    echo If you're seeing this after clicking on a start menu icon,  
    echo then right click on the shortcut and select "Run As Administrator".  
    echo #####  
    echo.  
    PAUSE  
    EXIT /B 1  
)
```

## UPDATE JENKINS AS WINDOWS SERVICE (CONT.)

```
@echo off
REM === Some modifications need to be made for your setup options ===
set deployLoc="C:\Program Files\Jenkins"
set jenkinsHome="C:\Program Files\Jenkins"
set jenkinsURL="http://mirrors.jenkins-ci.org/war/latest/jenkins.war"
set WGET="C:\Program Files\GnuWin32\bin\wget.exe"
Echo === Stopping Current Jenkins Service ===
sc stop Jenkins
Echo === Sleeping to wait for file cleanup ===
ping -n 4 127.0.0.1 > NUL
Echo === clean files ===
copy /Y %deployLoc%\jenkins.war %deployLoc%\jenkins.war.bak"
del %deployLoc%\jenkins.war
Echo === make room to explode new war file ===
RD /s /q %jenkinsHome%\war
Echo === get new files ===
%WGET% -P %deployLoc% --no-check-certificate %jenkinsURL%
Echo *** Starting new upgraded Jenkins
sc start Jenkins
Echo *** Sleeping to wait for service startup
ping -n 4 127.0.0.1 > NUL
```

# UPGRADE JENKINS ON WINDOWS

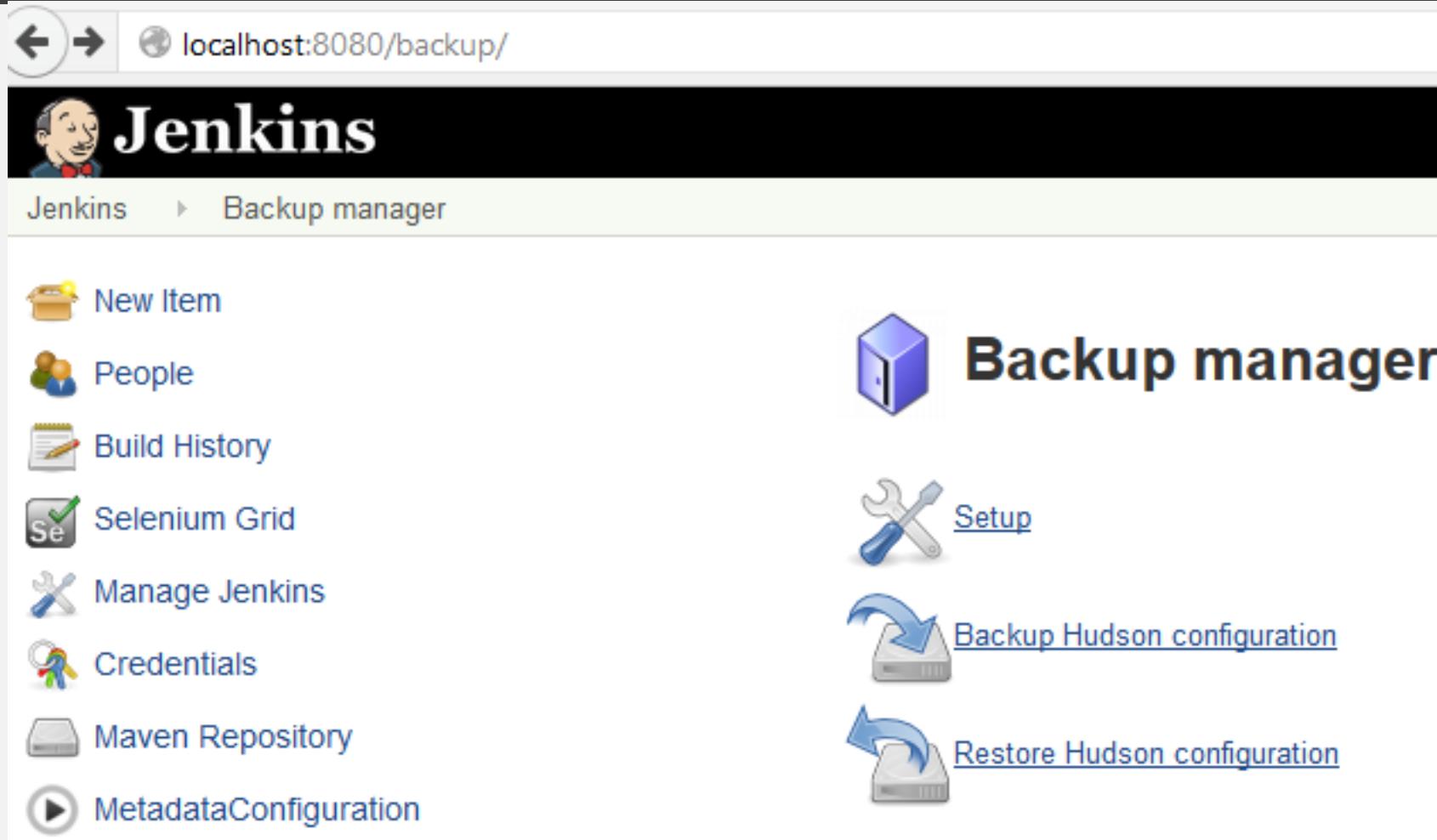
- If installed as a service on Windows and you do not have enough prowess in Windows Scripting,
  - Take a backup of JENKINS\_HOME
  - Use Jenkins CLI to export all jobs
  - Uninstall as service/program
  - Delete JENKINS\_HOME
  - Re-install Jenkins and re-set security
  - Grep difference between old and new set of plugins and install missing plugins
  - Re-import all jobs back using CLI

# BACKUP JENKINS

- Usually only required on Master.
- Slaves backup is usually a simple matter of disk backup (Use any backup s/w that does not lock up files/directories or require exclusive locks)
- Install the backup plugin.
  - The backup manager visible in Manage Jenkins page
- Click on Backup Manager



# BACKUP AND RESTORE JENKINS



The screenshot shows the Jenkins interface at [localhost:8080/backup/](http://localhost:8080/backup/). The left sidebar lists various Jenkins management options: New Item, People, Build History, Selenium Grid, Manage Jenkins, Credentials, Maven Repository, and Metadata Configuration. On the right, the "Backup manager" section is displayed, featuring a blue cube icon and the title "Backup manager". Below this are two more sections: "Setup" with a wrench and screwdriver icon, and "Backup Hudson configuration" and "Restore Hudson configuration" with icons showing a drive being inserted or removed.

New Item

People

Build History

Selenium Grid

Manage Jenkins

Credentials

Maven Repository

Metadata Configuration

Backup manager

Setup

[Backup Hudson configuration](#)

[Restore Hudson configuration](#)

# BACKUP AND RESTORE JENKINS - SETUP

The screenshot shows the Jenkins 'Backup config files' configuration page. On the left, there is a sidebar with various Jenkins management links. The main page has two sections: 'Backup configuration' and 'Backup content'. In 'Backup configuration', the Hudson root directory is set to C:\Program Files (x86)\Jenkins. The backup directory is empty, the format is set to zip, and the file name template is backup\_@date@.@extension@. There is also a section for custom exclusions and three checkboxes for verbose mode, configuration files only, and no shutdown. In 'Backup content', there are four checkboxes for backing up job workspace, builds history, maven artifacts archives, and fingerprints. A 'Save' button is located at the bottom right.

New Item  
People  
Build History  
Selenium Grid  
Manage Jenkins  
Credentials  
Maven Repository  
Metadata Configuration  
Metadata Search  
Failure Cause Management  
My Views  
Disk Usage  
Job Config History  
JS Widgets  
UI Samples  
Sounds  
Lockable Resources

**Backup config files**

**Backup configuration**

Hudson root directory C:\Program Files (x86)\Jenkins

Backup directory

Format zip

File name template backup\_@date@.@extension@

Custom exclusions

Verbose mode  
 Configuration files (.xml) only  
 No shutdown

**Backup content**

Backup job workspace  
 Backup builds history  
 Backup maven artifacts archives  
 Backup fingerprints

Save

# BACKUP AND RESTORE

- Alternatives to this plugin:
  - The **SCM Sync configuration plugin** allows automatically tracking changes brought to the global configuration of Jenkins and to the jobs configuration into a configuration management system (Subversion, etc.).
  - The **thinBackup** plugin focuses on backing up Jenkins's global configuration and jobs configuration. It also offers a scheduling feature
- Custom Modes
  - Shell Script + CRON jobs
  - Shell Script + CLI + SCM + CRON Jobs
    - CLI is used for XML level exports of jobs, SCM used for baselining this into version control.
- Ultimately, depends on what level of backup and security is defined at your organization end ☺.

# 7 best practices

# BEST PRACTICES

- Practice #1:
  - Use Plugins effectively
- Practice #2:
  - Standardize on Slave
- Practice #3:
  - Don't Build on master
- Practice #4:
  - Use incremental builds
- Practice #5:
  - Integrate with tools
- Practice #6:
  - Break up big jobs
- Practice #7:
  - Stick with Stable Releases

## PRACTICE #1 – USE PLUGINS EFFECTIVELY

- Search for plugins that match your needs.
- Plugins are organized by category and
- Have overlapping functionality
- If you are not using a plugin
  - Disable it.
  - This helps
    - Reduce Memory Usage
    - Reduce clutter on job page

# PRACTICE #1 – USE PLUGINS EFFECTIVELY

- Recommended plugins
  - JobConfigHistory
    - See the difference between job configurations now and then.
    - With authentication enabled, see who changed the job and how.

Jenkins

jenkins » jenkins\_main\_trunk » Job Config History

search abayer | log out

ENABLE AUTO REFRESH

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

[Job Config History](#)

[Git Polling Log](#)

**Job Configuration History**

Date	Operation	User	Show File	Diff	
				File A	File B
2011-07-04_23-18-30	Changed	<a href="#">jiervn</a>	<a href="#">View as XML (RAW)</a>	<input type="radio"/>	<input checked="" type="radio"/>
2011-05-04_20-12-25	Changed	<a href="#">SYSTEM</a>	<a href="#">View as XML (RAW)</a>	<input checked="" type="radio"/>	<input type="radio"/>
2011-05-04_17-59-11	Changed	<a href="#">abayer</a>	<a href="#">View as XML (RAW)</a>	<input type="radio"/>	<input type="radio"/>
2011-05-04_17-59-10	Changed	<a href="#">abayer</a>	<a href="#">View as XML (RAW)</a>	<input type="radio"/>	<input type="radio"/>
2011-03-31_21-03-15	Changed	<a href="#">kohsuke</a>	<a href="#">View as XML (RAW)</a>	<input type="radio"/>	<input type="radio"/>
2011-03-31_21-03-14	Changed	<a href="#">kohsuke</a>	<a href="#">View as XML (RAW)</a>	<input type="radio"/>	<input type="radio"/>
2011-03-19_09-39-49	Changed	<a href="#">olamy</a>	<a href="#">View as XML (RAW)</a>	<input type="radio"/>	<input type="radio"/>
2011-03-19_09-08-38	Changed	<a href="#">olamy</a>	<a href="#">View as XML (RAW)</a>	<input type="radio"/>	<input type="radio"/>
2011-03-19_09-07-10	Changed	<a href="#">olamv</a>	<a href="#">View as XML (RAW)</a>	<input type="radio"/>	<input type="radio"/>

[Show Diffs](#)

**Build History (trend)**

#964 Jul 20, 2011 11:01:45 AM 63MB
#963 Jul 19, 2011 6:31:53 PM 63MB
#962 Jul 19, 2011 11:36:51 AM 63MB
#961 Jul 17, 2011 5:31:34 PM 63MB

## PRACTICE #1 – USE PLUGINS EFFECTIVELY

- Recommended plugins
  - Disk Usage
  - Build Timeout
  - E-Mail Ext
  - Parametrized trigger

## PRACTICE #2: STANDARD SLAVES

- Use Same environment preferably
- Multiple slaves having same roles should have same environment
- Use Tools like Chef, Puppet, Dockers.
  - If you are not already using them, seriously consider using these.
- Have Jenkins install your tools
  - JDK, Maven, Ant ....
  - Script installation of tools inside jobs itself.
- Start slaves from pre-configured images
  - Cloud based
  - Dockers

## PRACTICE #3: DON'T BUILD ON MASTER

- Follows from Practice #2
- Set Executor count to 0 on master
- Easier to add slaves than to keep adding up capacity to master
- Never mix build between masters and slaves – sooner or later, there will be inconsistencies between builds

## PRACTICE #4: INCREMENTAL BUILDS

- If a build takes hrs, CI is not possible.
- If integrating with code review or using pre-tested commit processes, changes should be verified as fast as possible.
- These are complementary to full builds
- QA/Deployment should always be full and clean (built from source) builds
- You can use different jobs for incremental/full builds or have one job based on parameters set
- Maven projects have support for incremental builds
- Not always possible with free style projects
  - Will require Additional work to setup and configure to figure what changes last done.

## PRACTICE #5: TOOL INTEGRATION

- Let Jenkins install tools where possible.
  - Pre-tested commits with Gerrit (Code Review)
- Integrate with JIRA and other bug trackers
  - Issues reported can be created as new tasks/issues for planning
- Integrate with Sonar
  - Very very great tool for Code metrics, coverage, UTR etc.... All in one place
- Integrate with Artifactory
  - Control Staging of Maven/Gradle/IVY release candidates
  - Report on dependencies
- Integrate with Mylyn
  - See build and test results from within Eclipse itself.
- Integrate with Chat/IM
  - Fast communication of results

## PRACTICE #6: REDUCE BLOAT

- Break big jobs into smaller pieces
  - More manageable
  - Don't break it so much that it is impossible to navigate the UI
    - BALANCE!
- Use multiple Jenkins Master
  - Better Performance
  - Easier to navigate
  - Easier to split jobs logically
  - Easier to restart
- Clearly define phases in your development process
  - Logical organization
  - Pass data between jobs using parameterized triggers
  - Just re-run phases that fail – no need to run whole phase!!!

## PRACTICE #7: STABLE RELEASES

- Stick with LTS releases of Jenkins
- Do not upgrade just because an update is available
- 3 months is an acceptable time to schedule upgrades
- Upgrade plugins carefully.
  - Check if these will work with new LTS and no issues are found/re-introduced.
  - If not requiring a bug fix or new feature, don't bother updating plugging
  - Stick with what works!

## BEST PRACTICES FROM JENKINS WIKI

- Do not allow Anonymous access to Jenkins
- Backup Jenkins Home Regularly
- Use “fingerprinting” to manage dependencies
- Prefer to use “Clean build” when possible
- Integrate tightly with JIRA/Bugzilla
- Integrate with Repository Browsing tools like Fisheye
- Always configure jobs to show trend reports and automated testing
- Setup Jenkins on a partition with sufficient disk space
- If a job is unused, archive it and remove it.

## BEST PRACTICES FROM JENKINS WIKI

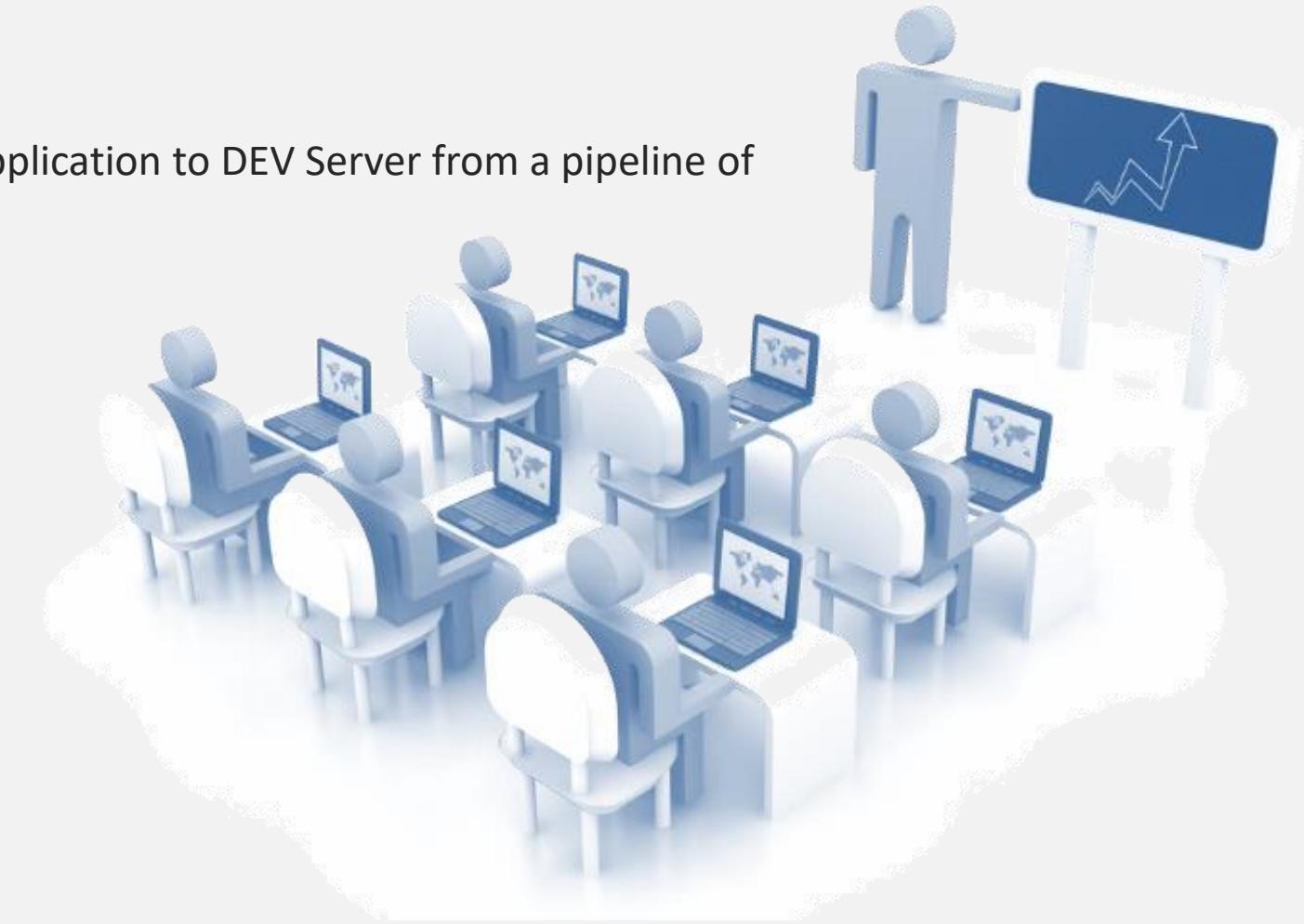
- For each master/development branch, separate projects should be used.
  - Really.. Branching strategy is crucial.
  - Arguments for and against feature branches.
  - Check with QA.. 
- Notification on a build should go to all developers
- Report failures as soon as possible.
- Tag, label or baseline codebase after successful build.
- Update working copy before running goal/target.
- In large systems, never ever build on masters

Q&A

# Questions???

# AGENDA FOR NEXT MODULE

- RECAP of all we taught so far
- Practical on Deploying a standard Web application to DEV Server from a pipeline of
  - Compile
  - Unit Test
  - Deploy
  - Run Acceptance Tests
  - Run Load Tests



## FOOD FOR THOUGHT!!

- How would you share information between different builds?
- The test cases can easily number in 1000's. How do you ensure that test cases are run in parallel? Extending this, how would you distribute cases across nodes for faster execution?
- Take the same example as above, but let's give it a twist. Your company wants to test its web application on multiple browsers? How would you do the same? (HINT: Selenium, Distributed Builds)

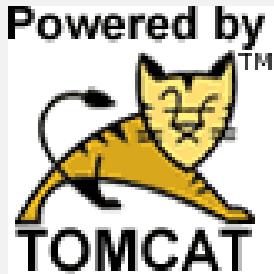
# **MODULE 10**

## **TYING IT ALL TOGETHER - DEMO**



## DEPLOYMENT TO SERVERS

- Jenkins allows you plugins to deploy to



- For other app servers, you **will need to script. Sorry!**
- Two aspects:
  - You can choose to rebuild from source (or)
  - Deploy only artifacts from previous sources
- Method 1 - inefficient, but maintains clean build principle.

## DEPLOYMENT TO SERVERS

- Method 2 is mostly used.
- Use the Copy Artifact Plugin
  - Copy artifact from another build job workspace into current workspace
- Use Build Triggers and Build Promotion Plugin
- Issues:
  - When you build, property files are included into WAR. This is not correct because DEV, QA and Prod will have different files.
  - Either
    - Externalize references to properties into JNDI (or)
    - Keep DEV,QA, PROD configs separate in SCM and as pre-build step, copy appropriate config file and modify artifacts dynamically and re-test.
    - Build from specific SCM Tag/Revision

# A SIMPLE DEPLOYMENT PIPELINE

- Build the default application.
- Run tests and metrics
- Call Deployment Job

Ref: Jenkins The Definitive Guide



# USE COPY ARTIFACT PLUGIN

- Deployment job does the copying of artifacts and use

Ref: Jenkins The Definitive Guide

Build

**Copy artifacts from another project**

Project name: gameoflife-default/com.wakaleo.gameoflife\$gameoflife-web

Which build: Latest successful build

Stable build only

Artifacts to copy: \*\*/\*.war

Target directory:

Flatten directories    Optional

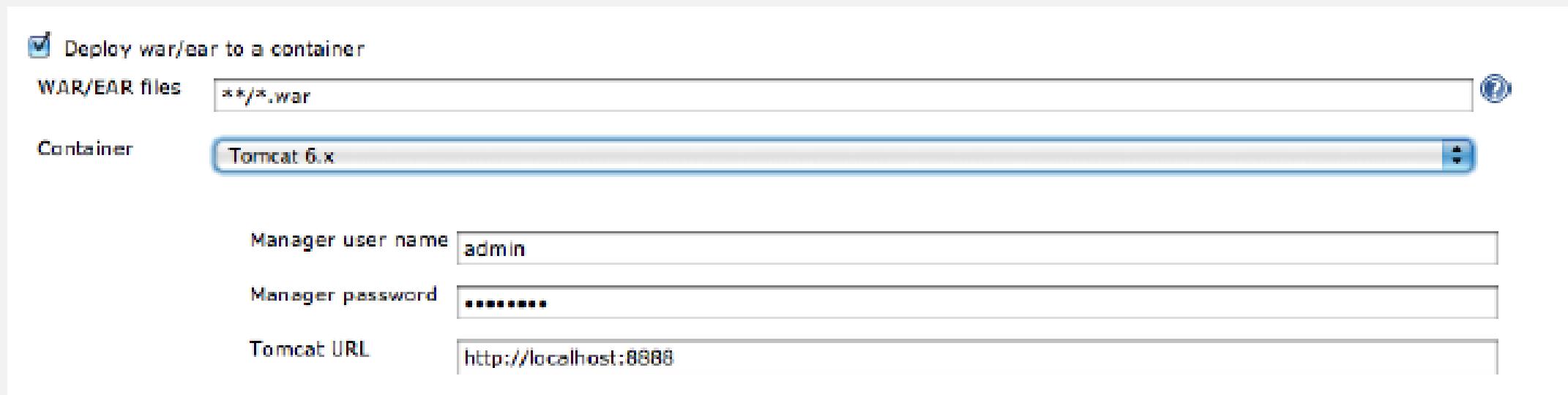
[Delete](#)



# USE DEPLOY PLUGIN

- After Copying, deploy same to server

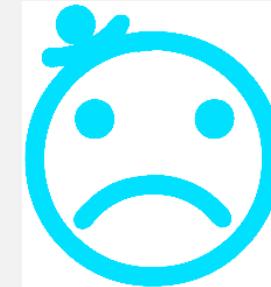
Ref: Jenkins The Definitive Guide



- This is the **Hot Deploy** method

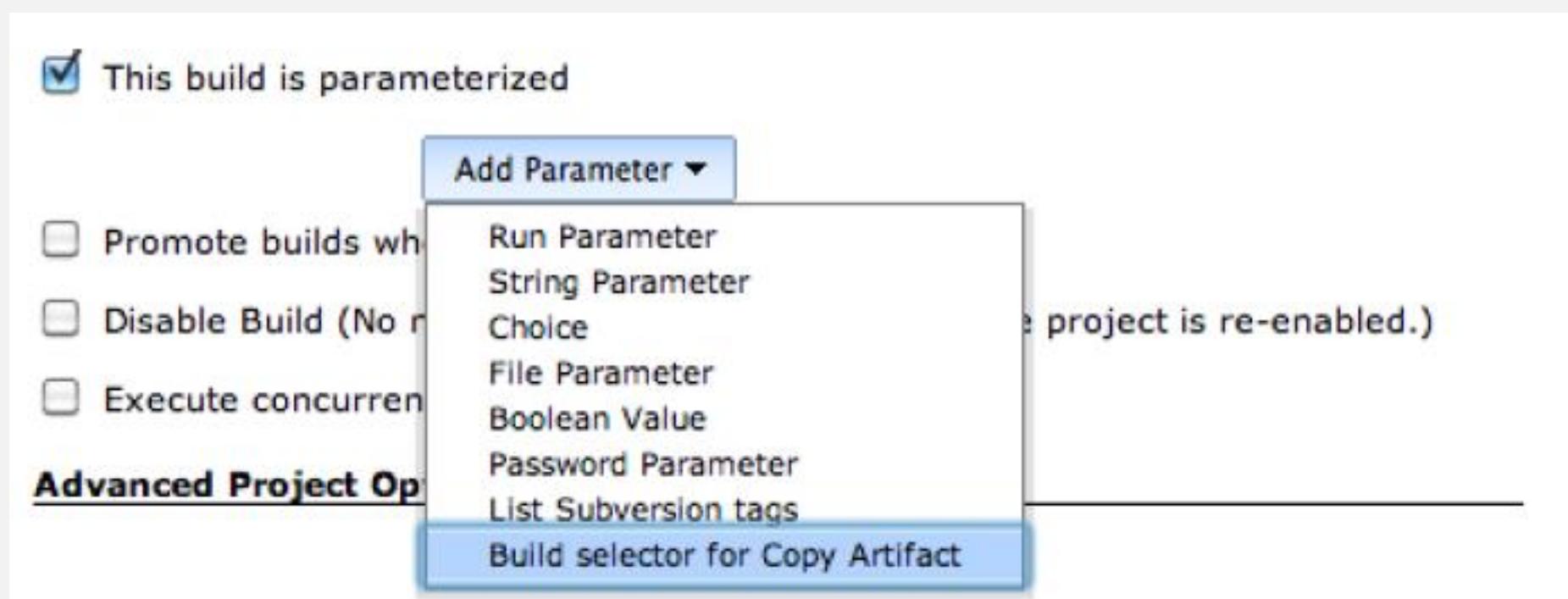
# DEPLOY A SPECIFIC VERSION

- 2 ways to do this:
  - Jenkins hosts all versions
    - Use Copy Artifacts, Deploy and Parametrized Triggers
  - External Artifactory Repository
    - Preferred
    - We use Artifactory (can use Nexus also)
    - Unfortunately, works best with commercial editions



## DEPLOY A SPECIFIC VERSION - JENKINS

- Add a parametrized trigger Plugin step
- Use Special “Build selector for Copy Artifact”



## DEPLOY A SPECIFIC VERSION - JENKINS

- The build is now marked as parameterized
- Add a parameter for copying artifacts.
- Env Variable is called: COPYARTIFACT\_BUILD\_NUMBER\_MY\_BUILD\_JOB
  - *E.g. COPYARTIFACT\_BUILD\_NUMBER\_GAME\_OF\_LIFE\_DEFAULT=34 // selects versions 34*

## DEPLOY A SPECIFIC VERSION - JENKINS

This build is parameterized ?

**Build selector for Copy Artifact** ?

Name  ?

Default Selector  ? 

Stable build only

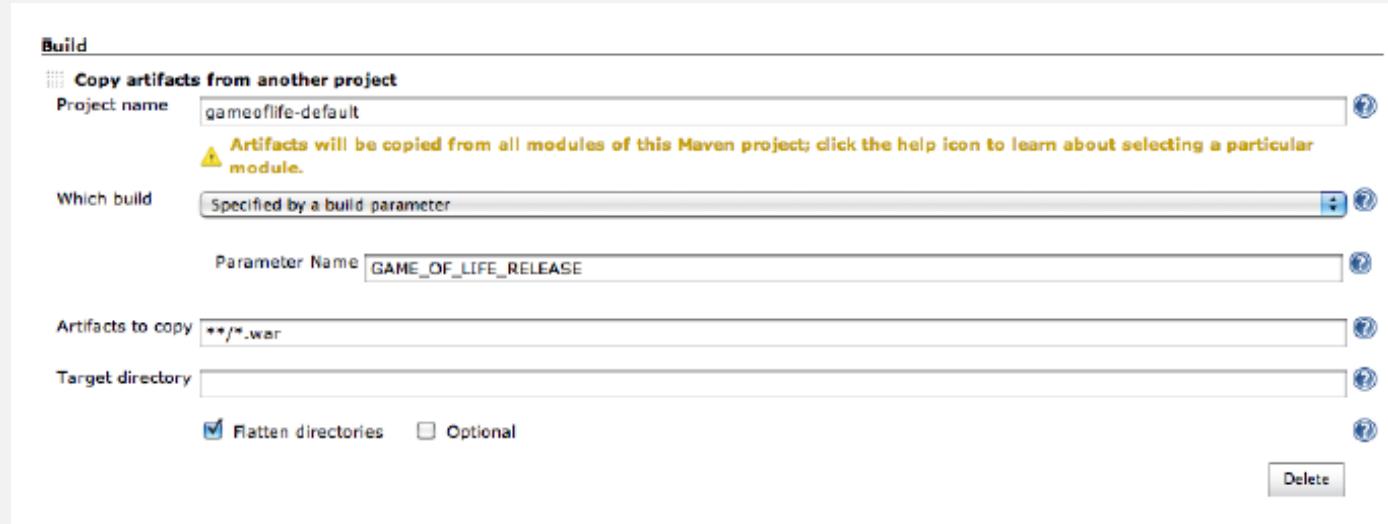
Description  ?

Delete Add Parameter ▾

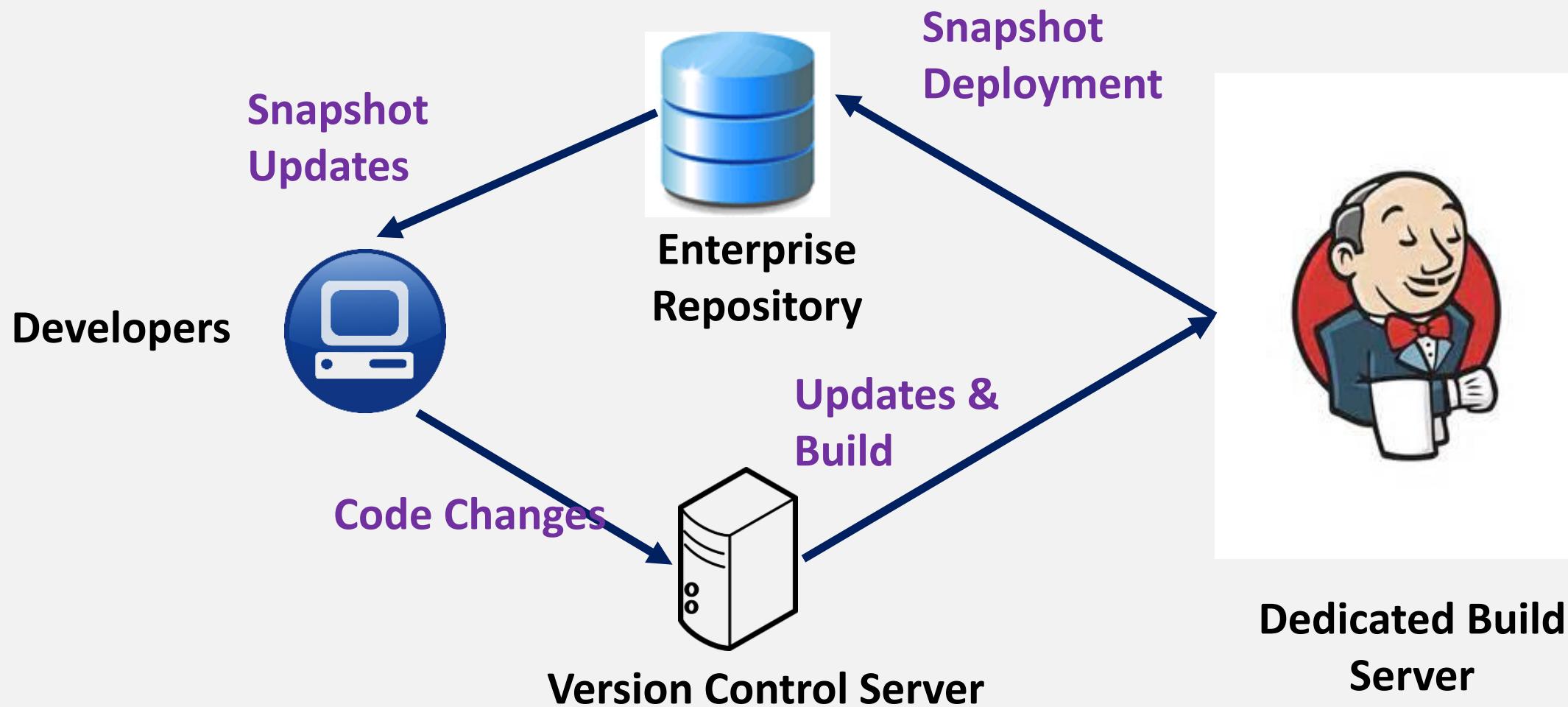
## DEPLOY A SPECIFIC VERSION - JENKINS

- The Build now is configured to this version.
- In the Build section, add a “Copy artifacts from another project” step.
  - Specify the project where the artifact was built and archived
- Use the build specified in the parameter defined earlier.
  - Choose “Specified by a build parameter” in the “Which build” option,
  - Provide the variable name specified earlier in the build selector name field
  - Configure the artifacts to copy as in previous example.
- Deploy as normal

# DEPLOY A SPECIFIC VERSION - JENKINS



# DEPLOY FROM REPOSITORY



## DEPLOY FROM REPOSITORY

- SNAPSHOTs are unstable, RELEASEs are not.
- The exact strategy used to decide when a release version is to be created, and how it is deployed, varies between organization.
  - Formal release at the end of each iteration or sprint, with a **well-defined** version number and corresponding set of release notes that is distributed to QA teams for further testing.
    - When a particular version gets the go-ahead from QA, it can then be deployed into production.
  - Cut a new release whenever a new feature or bug fix is ready to be deployed.
    - If a team is particularly confident in their automated tests and code quality checks, it may even be possible to automate this process completely, generating and releasing a new version either periodically (say every night) or whenever new changes are committed.

## DEPLOY FROM REPOSITORY

- Configure Jenkins to use M2 Release Plugin with Maven.
- Setup a dedicated deployment project
  - Use maven-war-plugin to fetch a version from Repository.
  - Build using package goals
  - Deploy using install goal or via Deploy plugin
- Non Java Projects
  - Usually just copy over files.
  - Use the “Publish over” series of plugins .....
  - Pretty straight forward.

# ADDENDUM

- **Configure M2 Release Plugin: How-to**
- In build environment, check “Maven Release Build”
- **Goals:** release:prepare release:perform

**Build Environment**

---

<input checked="" type="checkbox"/> Maven release build		
Release goals and options	release:prepare release:perform	<a href="#">?</a>
Preselect versioning mode	None	<a href="#">?</a>
Preselect custom SCM comment prefix	<input type="checkbox"/>	<a href="#">?</a>
Preselect append Hudson username	<input type="checkbox"/>	<a href="#">?</a>

# MAVEN RELEASE

## Perform Maven Release

Versioning mode

- Maven will decide the version
- Specify version(s)
- Specify one version for all modules

Release Version

0.0.66

Development version

0.0.67-SNAPSHOT

Append Hudson Build Number



Specify SCM login/password

Specify custom SCM comment prefix



**Schedule Maven Release Build**



The Maven Release plugin is an extremely buggy maven plugin. For example, there are known issues about tagging with certain SCM providers.

# ARTIFACTORY

- Supported via the Artifactory Plugin

The screenshot shows the Jenkins configuration page for the Artifactory plugin. The URL in the browser is `localhost:8080/jenkins/configure`. The page title is "jenkins deploy to artifactory". The main section is titled "Artifactory" and contains "Artifactory servers". There is a checked checkbox for "Use the Credentials Plugin" and an unchecked checkbox for "Artifactory". The "URL" field is set to `http://localhost:8081/artifactory`. Below this, there is a "Default Deployer Credentials" section with a dropdown menu showing "sriram/\*\*\*\*\*" and an "Add" button. A message says "Found Artifactory 4.3.0". There is also a "Test Connection" button. At the bottom right of the section are "Delete" and "Advanced..." buttons.

# ARTIFACTORY - DEPLOYMENT

- For Maven2/3 projects Post-Build action.

Deploy artifacts to Artifactory

Artifactory server

Target releases repository  Different Value

Target snapshot repository  Different Value

Custom staging configuration  Different Value

Items refreshed successfully Refresh

Override default deployer credentials

Deploy even if the build is unstable

Deploy maven artifacts

Include Patterns

Exclude Patterns

Filter excluded artifacts from build info

Deployment properties

Capture and publish build info

Include environment variables

Record implicit project dependencies and build-time dependencies

Allow promotion of non-staged builds

Allow push to Bintray for non-staged builds

# USING CAPISTRANO

- Capistrano is a popular tool used for external deployments.
- Jenkins supports deployment to Tomcat using
  - Normal Artifact Deployer
  - Capitomcat Plugin
  - External Capistrano
- Usage of External Capistrano is recommended for maximum control.

# USING CAPISTRANO

- Capistrano is a Ruby Gem.
- It requires Ruby 1.9.3+
- The following command installs Capistrano
  - `gem install capistrano`
- The following command installs Capistrano Extensions
  - `gem install capistrano-ext`
- Prepare your project for Capistrano by using the capify command
  - Capfile
  - Config/deploy.rb files created

# USING CAPISTRANO

- The deploy.rb contains instructions on how to deploy.
- We will take this with a concrete example

```
require 'capistrano/ext/multistage'
set :applicaton, "mywebapp"
set :scm, :git
set :repository, "git@account.demo.com:/account/repository.git"
set :scm_passphrase, ""
set :stages, ["staging", "production"]
set :default_stage, "staging"
```

At this point you can create 2 files: production.rb and staging.rb where you will have production or staging **specific configuration**

- To deploy, just execute cap production deploy (to deploy to production) or cap deploy (to staging)

# USING CAPISTRANO

- Given this, Jenkins can be used to deploy an artifact via
  - Copy artifact from previous build
  - Copy artifact from Artifactory
  - Call Deploy as a post-build action that executes a cap deploy command



**Questions and Answers.**



**Thanks for listening!!!**

**Seshagiri Sriram**  
**Email:** [SeshagiriSriram@gmail.com](mailto:SeshagiriSriram@gmail.com)