

# Continuous Integration With Jenkins



- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins Intro
  - Lab 1 : Install and Configure Jenkins
- A Jenkins Job
  - Lab 2 : Build and Configure your first Jenkins job
- Jenkins Plug-ins
  - Lab 3 : Build Jenkins Job to generate code quality reports

# Day 2: Sessions

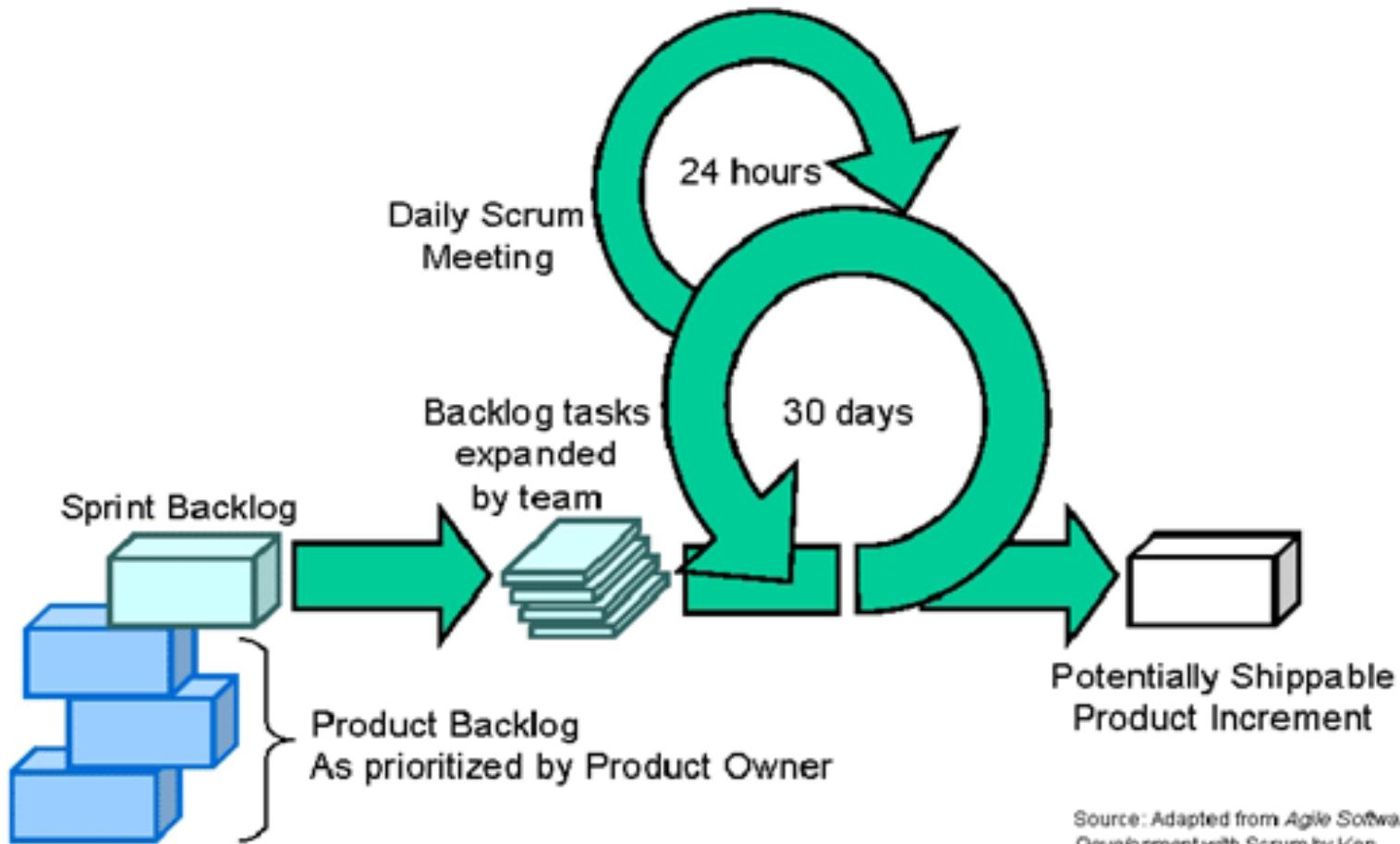
- Any remaining Materials from Day 1
- SonarQube Plugin
  - Lab 4 : Generate first SonarQube report using Jenkins
- Advanced Jenkins
  - Lab 5 : Configure distributed builds using Master/Slave configuration
  - Lab 6 : Build your first pipeline with Jenkins
- Best Practices for Jenkins
- References
- Q & A

# Introducing Continuous Integration and Jenkins

- Agile Development
  - Continuous Integration versus Continuous Delivery versus Continuous Deployment
  - History of Jenkins
  - State of the Jenkins Community

- Agile development to describe methodologies for incremental software development
  - Empowers people to collaborate and make team decisions for continuous planning, continuous testing and continuous integration and delivery
  - Scrum, XP, Kanban, Lean, FDD are some of the most popular Agile development methodologies
- Manifesto for Agile Software Development
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan

# Agile Development Process Overview



Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

# Agile Development Practices

- Product Backlog and Spring Backlog
- Sprints and Daily Sync
- Burn down and velocity charts
- Sprint review and retrospective
- Simple design and Regular refactoring
- Pair Programming
- Test Driven Development
- Automation is the key
- Continuous Integration (CI)
- Continuous Delivery (CD)
- Definition of Done
- Common “war-room” style work area

- Continuous Integration and Continuous Delivery become an essential ingredients for teams doing iterative and incremental software delivery in Agile Development
  - Developers share the common source code repository
  - Dedicated Continuous Integration environment
  - All code must pass unit tests
  - Integrate often
  - Regression tests run often
  - Code matrices are published
  - Every change to the system is releasable to production
  - Automation is the key

- Agile Development

- **Continuous Integration versus  
Continuous Delivery versus  
Continuous Deployment**

- History of Jenkins
- State of the Jenkins Community

# Continuous Integration

- Continuous Integration is a software development practice where members of a team **integrate their work frequently**, usually each person integrates at least daily - leading to **multiple integrations per day**. Each integration is **verified** by an automated build (including test) to detect integration errors as quickly as possible (Martin Flower)
- Goal is to merge and test the code **continuous** to catch issues early by **automating integration process**
  - Your project must have a reliable, repeatable, and automated build process involving no human intervention
- Continuous Integration Server (aka Build Server Jenkins) is responsible for performing the integration tasks
- Concepts of unit testing, static analysis, failing fast and automated testing are core to Continuous Integration

# Continuous Integrations Practices

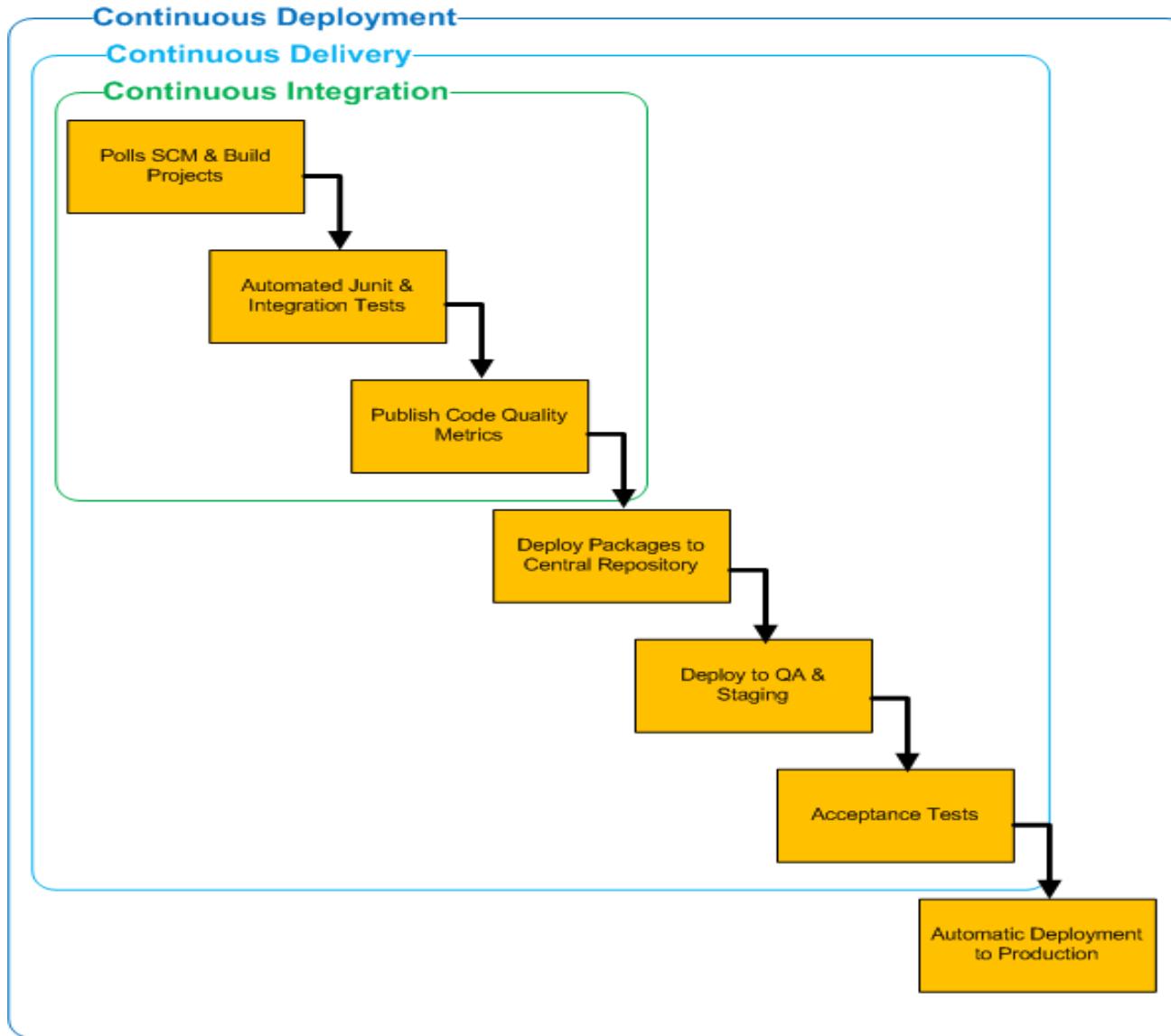
- Have a single source repository for all developers
- Automate the build
- Every change to SVN should make a new build
- Keep the build fast and trackable
- Make your build self-testing
- Test the build in production-like environment
- Keep all verified releases in artifacts repository and available to everyone
- Publish coding metrics

- Continuous Delivery is a **natural extension** of Continuous Integration
  - Every change to the system has passed all the relevant automated tests and it's ready to deploy in production
  - Team can release any version at the push of a button
  - But the deployment to production is **not automatic**
- The goal of CD is to put business owners in the control of making the software releases
- Continuous Delivery is an absolute requirements of **DevOps practices**

# Continuous Deployment

- Continuous Development is adding **automatic deployment to end users** in the Continuous Delivery process
- Continuous Deployment is the practice of automatically deploying every successful build directly into production
  - Deploying the code to production as soon it passes the automated and UAT tests
- Continuous Deployment is not appropriate for many business scenarios
  - Business Owners prefer more predictable release cycle and not making code release to production every week

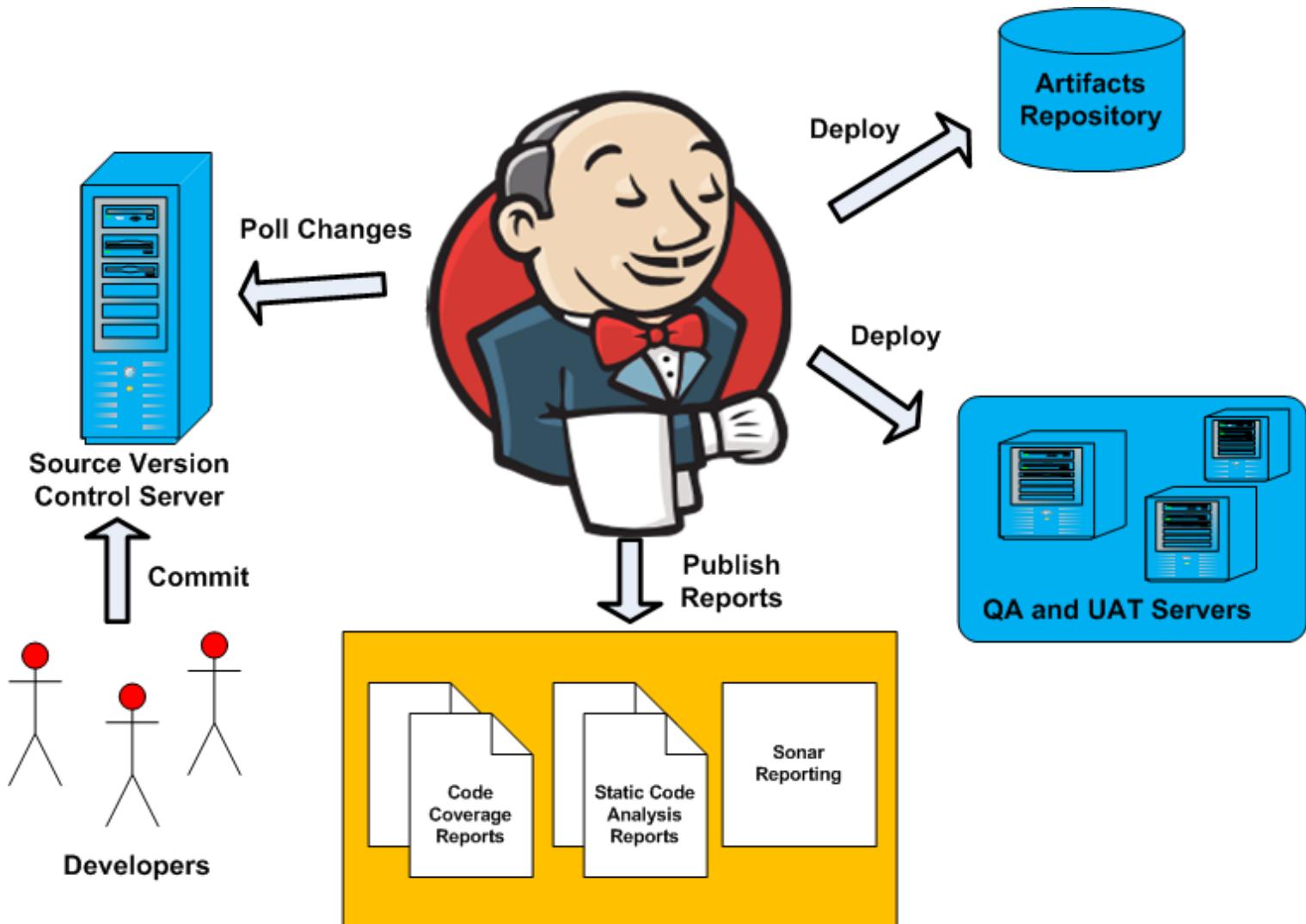
# Continuous Integration, Delivery and Deployment



- Agile Development
- Continuous Integration versus  
Continuous Delivery versus  
Continuous Deployment
- **Jenkins and History of Jenkins**
- State of the Jenkins Community

- Awesome Continuous Integration (CI) Server
- Open Source, Free and Written in Java
- Large and Dynamic community with massive adoption
- Easy to Install in many different platforms
- Easy to Use and friendly user interface
- Lot of resources and tutorials available
- More than 1000 plugins and new Plugins coming every week
- Extensible architecture – easy to extend and customize
- Distributed Builds
- Many more things...

# How Jenkins Fits in CI and CD



- Jenkins formerly known as [Hudson](#)
- Hudson was first released by [Kohsuke Kawaguchi](#) of Sun Microsystems in 2005
- Initially it was only used within Sun but by 2010 the Hudson captured 70% of CI market share
- Oracle bought Sun Microsystems in 2010
- Due to Naming and open source dispute, Original Hudson team created new project Jenkins forked from Hudson
- Oracle continued the development of Original Hudson
- Majority of Hudson users [migrated to Jenkins](#) within few months of initial Jenkins phase

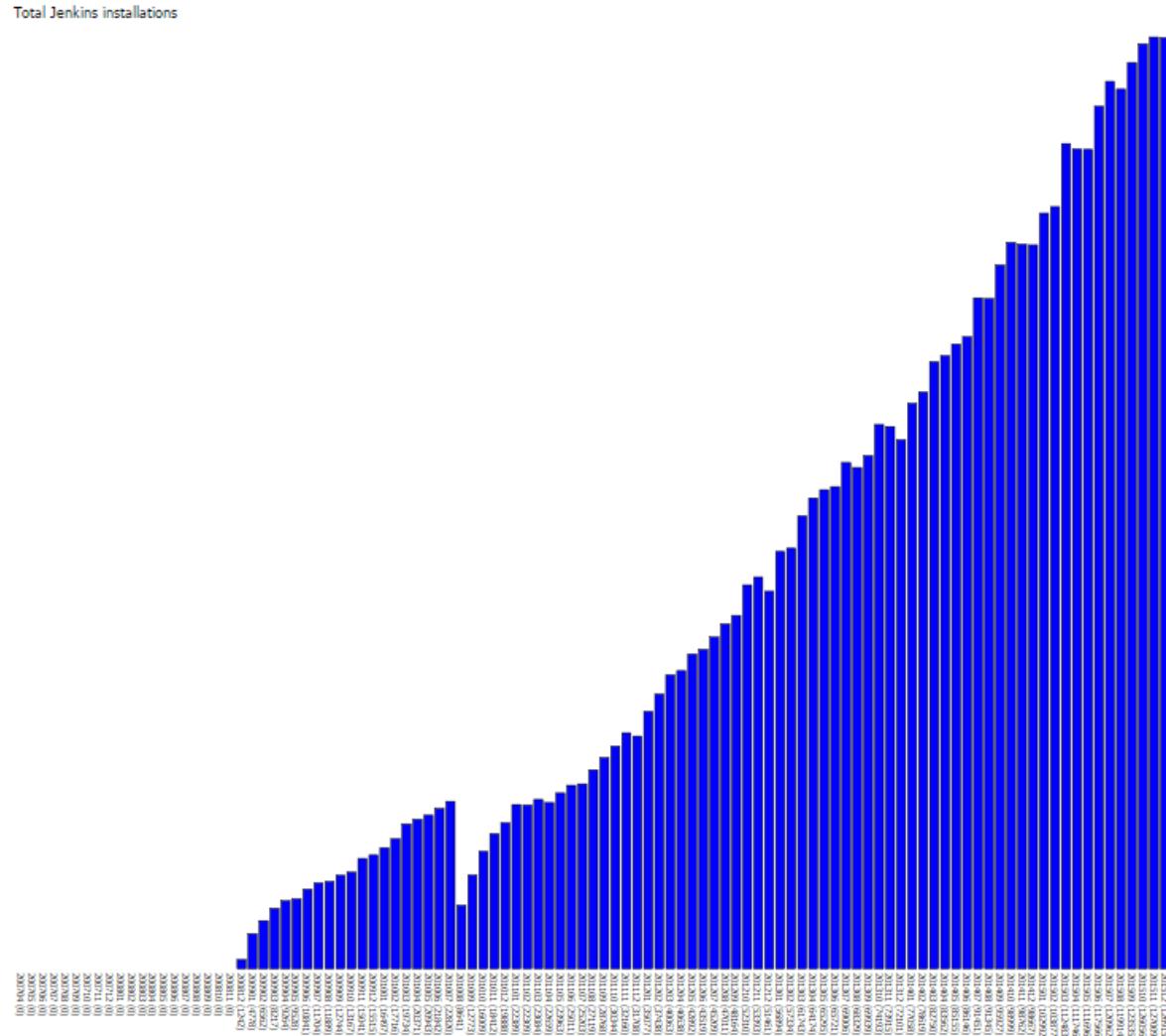
- Agile Development
- Continuous Integration versus  
Continuous Delivery versus  
Continuous Deployment
- History of Jenkins
- **State of the Jenkins Community**

# State of the Jenkins Community

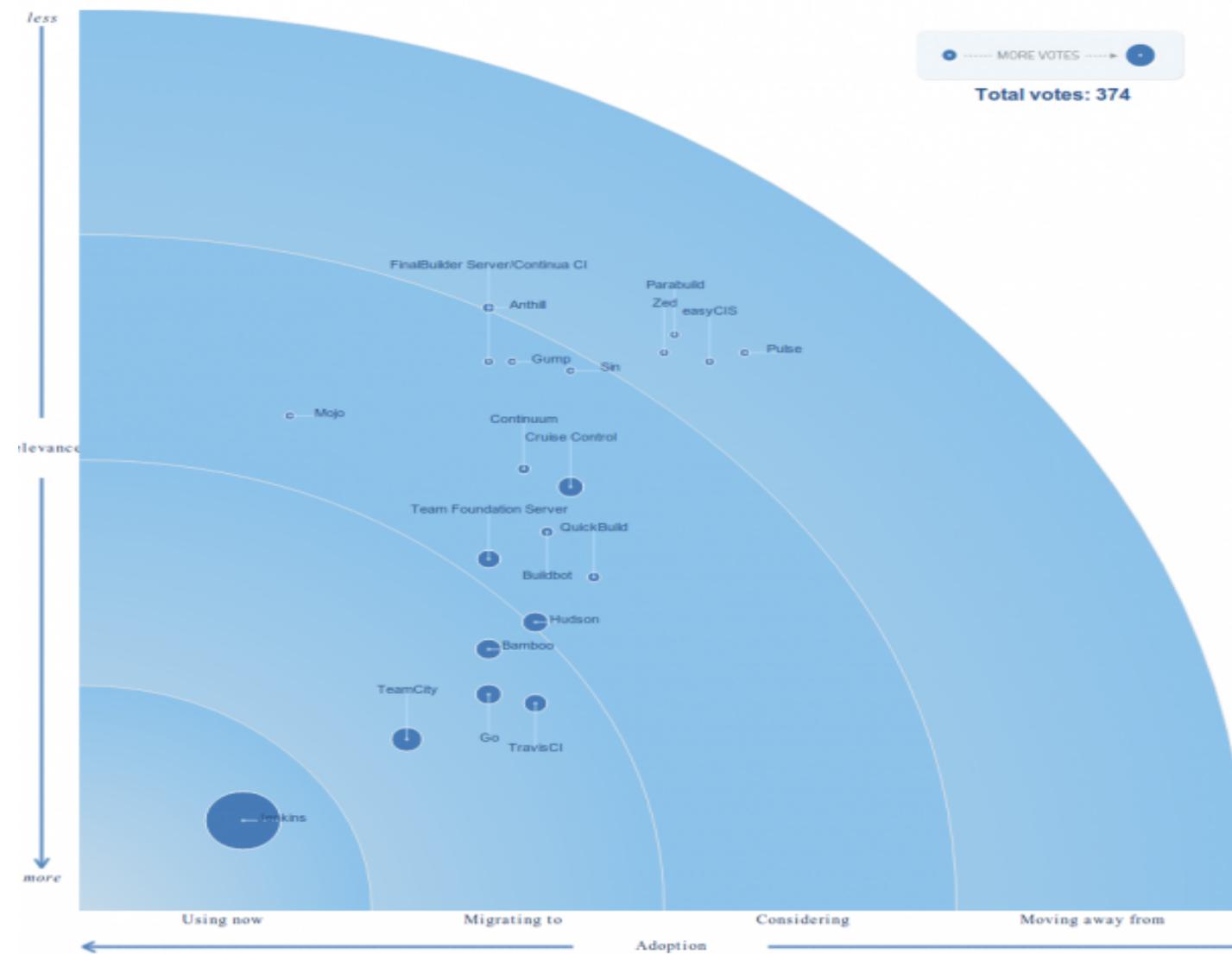
- Largest install base of any open source continuous integration and delivery platform
- More than **100K active users** in open source Jenkins CI project
- Community contributed with more than **1000 plugins**
- Over 1000+ public repositories on GitHub and strong commit activity
- Quick feedback with addressing bugs and issues
- Get answer on any questions from Jenkins user mailing list and Stackoverflow
  - Chances are other people have had your question and may have a solution
- Learn more about Jenkins at <http://jenkins-ci.org/>

# Total Jenkins Installations

More stats can be found at - <http://stats.jenkins-ci.org/jenkins-stats/svg/svgs.html>



# InfoQ CI Survey 2014



Ref: <https://jenkins-ci.org/blog/2014/04/11/infoq-ci-survey-2014/>

# Jenkins Very Active GitHub Repository

jenkinsci / jenkins  
forked from kohsuke/hudson

Watch 599 Star 4,671 Fork 2,203

Jenkins Continuous Integration Server <http://jenkins-ci.org/>

21,636 commits 85 branches 386 releases 364 contributors

Branch: master / jenkins / +

This branch is 1163 commits ahead of kohsuke:master. #1883 Compare

oleg-nenashev Noting fix of JENKINS-31098 in #1882 Latest commit e3b3638 9 hours ago

cli Use of class Base64 in a static way 5 days ago

core Merge pull request #1861 from rodrigc/JENKINS-15792 15 hours ago

plugins [maven-release-plugin] prepare for next development iteration 6 days ago

src/site Resurrect the maven site changes 3 years ago

test Merge pull request #1882 from pjanouse/JENKINS-31098 15 hours ago

war [maven-release-plugin] prepare for next development iteration 6 days ago

.gitignore add '\*.swp' (Vim recovery files) to .gitignore 7 days ago

.jenkins Fixed a bug in the test harness. 5 years ago

BUILDING.TXT Link to 'Building Jenkins' page from text file a year ago

Code Pull requests 105

Pulse Graphs

HTTPS clone URL <https://github.com>

You can clone with **HTTPS** or **Subversion**.

Clone in Desktop Download ZIP

# Installing and Running Jenkins

- **Installing Jenkins**
  - Installing Jenkins from the jar File
  - Installing Jenkins in a Servlet Container
  - Installing Jenkins using Platform Specific Installer
- Setup Security
- Email and Version Control
- Master/Slave Configurations
- Lab 1 : Install and Configure Jenkins

- Easy to install on different operating systems
- Also available as installer or native package
- Couple of options for installing Jenkins
  - Run as standalone application by launching with java -jar
  - Deployed on Servlet Container
  - Use platform specific package or installer
- Java is the only requirement for installing Jenkins
  - Install latest Java
  - Set JAVA\_HOME environment variable

# Download Jenkins - <https://jenkins-ci.org/>

An extensible open source continuous integration server

BLOG

CONNECT

BUG TRACKER

WIKI

CI

DONATE



## Meet Jenkins

Find out what Jenkins is and get started.



## Use Jenkins

See how to get more out of your Jenkins.



## Customize Jenkins

Choose from 1092 plugins to customize Jenkins exactly to your needs.



## Extend Jenkins

Learn how to build Jenkins or extend Jenkins by writing plugins.

## Download Jenkins

Release

Long-Term Support Release

### Java Web Archive (.war)

Latest and greatest (1.631)

[changelog](#) | [past releases](#)

[upgrading from Hudson?](#)

### Native packages



Windows



Ubuntu/Debian



Red Hat/Fedora/CentOS



Mac OS X



openSUSE



FreeBSD



OpenBSD



Gentoo

### Docker

`docker pull jenkinsci/jenkins`

# Running Jenkins from the jar File

- Download latest and greatest WAR from jenkins-ci.org
- Copy jenkins.war into the appropriate folder
- Open command prompt and execute the following command
  - java -jar jenkins.war
- It uses Jetty to run Jenkins

```
c:\JenkinsCourse\Jenkins>java -jar jenkins.war
Running from: C:\JenkinsCourse\Jenkins\jenkins.war
webroot: $user.home/.jenkins
Oct 05, 2015 10:39:13 AM winstoney.Logger logInternal
INFO: Beginning extraction from war file
Oct 05, 2015 10:39:46 AM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: jetty-winstone-2.8
Oct 05, 2015 10:40:06 AM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: No JSP support is available due to missing javax.servlet.jsp API.
```

# Installing Jenkins in a Servlet Container

- Tomcat and Jetty are most popular containers for Jenkins
  - Install Tomcat at appropriate folder
  - Simply copy jenkins.war to \$TOMCAT\_HOME/webapps folder
- You can install Jenkins on any other Servlet Containers
- However some containers may required minor configuration changes to work with Jenkins

- Installer and Native packages are available for popular Operating Systems such as Windows, Mac OS X and major Linux distributions
- Windows Installer comes as ZIP file containing MSI package for Jenkins
  - Unzip the ZIP file
  - Run jenkins.x.x.msi installer
  - Installer will create Windows service to start and stop Jenkins
- MSI installer comes with a bundled JRE

# Verify Jenkins on the browser

- Verify Jenkins by pointing your browser to <http://localhost:8080>

The screenshot shows the Jenkins web interface running at <http://localhost:8080>. The browser's address bar displays "localhost:8080". The main page features a large "Jenkins" logo with a cartoon character. On the left, a sidebar menu includes links for "New Item", "People", "Build History", "Manage Jenkins", and "Credentials". The central area has a large "Welcome to Jenkins!" message and a call-to-action button that says "Please [create new jobs](#) to get started.". Below this are two collapsed sections: "Build Queue" (which shows "No builds in the queue.") and "Build Executor Status" (which shows "1 Idle" and "2 Idle").

- All in one configuration dashboard for managing Jenkins
- Configure JDKS, Ant, Maven, Security, Email and Version Controls
- Install new plugins and update any existing plugins
  - Will review Gradle and Git plugins in this section
- Configure parallel and distributed builds
- Reload configuration from disk (xml files)
- Lists Java System properties and system environment variables
- View Jenkins logs and statistics in real time

# Manage Jenkins Screen

-  New Item
-  People
-  Build History
-  Manage Jenkins
-  Credentials
-  My Views

**Build Queue** -

No builds in the queue.

**Build Executor Status** -

1 Idle  
2 Idle

## Manage Jenkins

New version of Jenkins (1.633) is available for [download \(changelog\)](#).

[Or Upgrade Automatically](#)

### [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. ([updates available](#))

### [System Information](#)

Displays various environmental information to assist trouble-shooting.

### [System Log](#)

System log captures output from java.util.logging output related to Jenkins.

# Configure System

By Clicking on Configure System link, Jenkins will redirect you to another configuration page where you can configure global settings such as JDK, Maven, Ant, Git and Gradle.

## Manage Jenkins

⚠ New version of Jenkins (1.647) is available for [download \(changelog\)](#). Or Upgrade Automatically

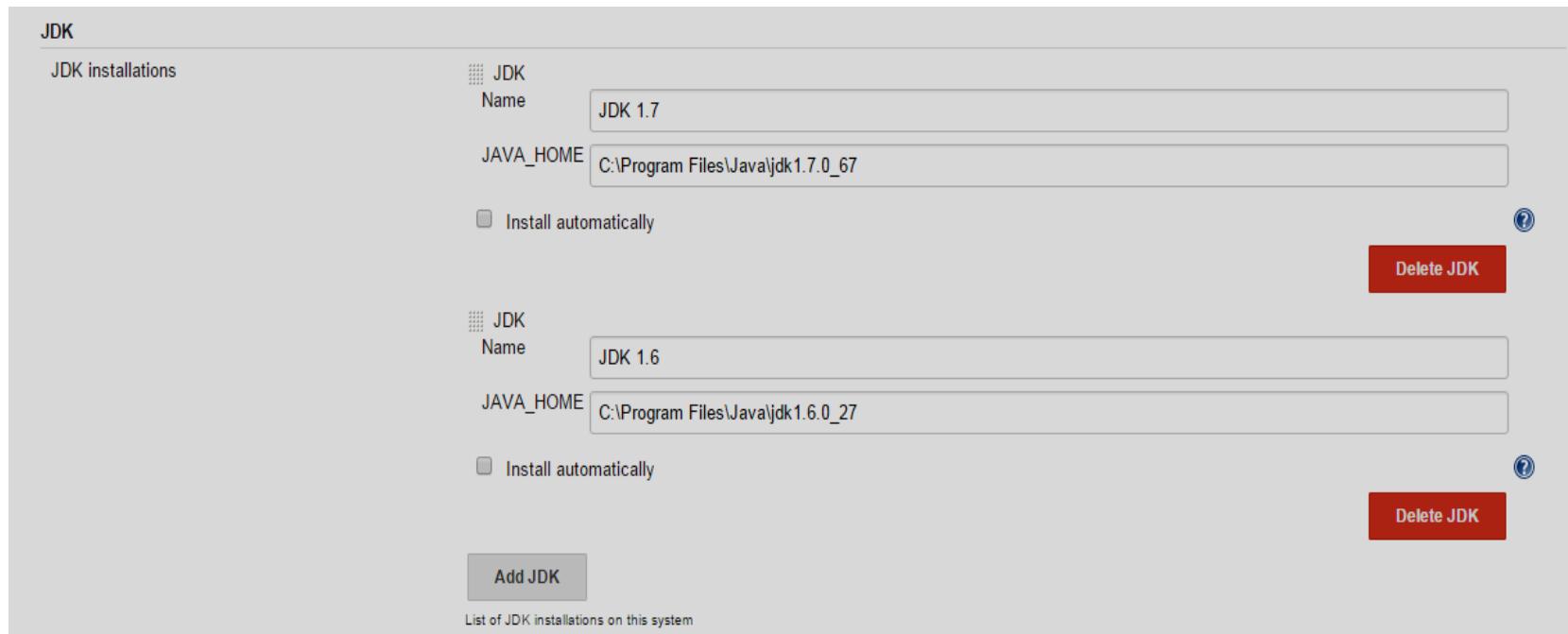
⚠ Slave to master security subsystem is currently off. [Please read the documentation](#) and consider turning it on.

The screenshot shows the Jenkins 'Manage Jenkins' page. At the top, there are two notifications: one about a new version available for download and another about the slave-to-master security subsystem being off. Below these are five management options, each with an icon and a brief description:

- Configure System** (Icon: wrench and screwdriver): Configure global settings and paths. This option is circled in red.
- Configure Global Security** (Icon: padlock): Secure Jenkins; define who is allowed to access/use the system.
- Reload Configuration from Disk** (Icon: circular arrow): Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.
- Manage Plugins** (Icon: puzzle piece): Add, remove, disable or enable plugins that can extend the functionality of Jenkins. **(updates available)**
- System Information** (Icon: computer monitor): Displays various environmental information to assist trouble-shooting.

# Configure JDK

- Just provide JDK name and path to Java home folder
  - You can have multiple JDKs listed here



# Configure Maven

Specify Maven installation folder or install from website

The screenshot shows the Maven configuration interface. At the top, under 'Maven installations', there is a single entry for 'Maven 3.0' with the path 'C:\apps\apache-maven-3.0.4'. A red arrow points from the text 'Specify Maven installation folder or install from website' to the 'MAVEN\_HOME' field. Below this, there is a checkbox for 'Install automatically' and a 'Delete Maven' button. An 'Add Maven' button is also present. In the 'Maven Project Configuration' section, there are fields for 'Global MAVEN\_OPTS' and 'Local Maven Repository'. The 'Local Maven Repository' field contains the default value 'Default (~/.m2/repository)'. A red arrow points from the text 'Set MAVEN\_OPTS such as memory size or different location for Local Maven Repository' to the 'Local Maven Repository' field.

Set MAVEN\_OPTS such as memory size or different location for Local Maven Repository

- Install Gradle plugin using Manage Plugins link
  - Click on Manage Jenkins button on Jenkins vertical navigation
  - Then click on Manage Plugins link
  - Next click the Available tab and search for Gradle
  - Select the Gradle plugin and click on Install without restart button
- After the installation, you will able to see Gradle plugin under Installed tab

Enabled	Name ↓	Version	Previously installed version	Pinned	Uninstall
<input checked="" type="checkbox"/>	<a href="#">Gradle plugin</a> This plugin allows Jenkins to invoke <a href="#">Gradle</a> build scripts directly.	1.24			<a href="#">Uninstall</a>

# Gradle Configuration Section

- The next thing is to configure Gradle to use in Jenkins job
  - Either you can install Gradle locally or you can have it installed automatically by Jenkins plugin
- Click on Manage Jenkins link on the dashboard and then click on Configure System link
- Navigate to Gradle section on the page and click on Add Gradle button



- Upgrading Jenkins is pretty easy
  - Just replace your local jenkins.war with new version and restart Jenkins
  - Can upgrade Jenkins installation directly from web interface
- **Don't forget to backup Jenkins before upgrading...**

# Restart Jenkins using Web Interface

- If you need to restart Jenkins, go to `/safeRestart` or `/restart`
  - This option will not work if you are installing Jenkins with `java -jar`
- **safeRestart** will restart Jenkins after the current builds have completed  
<http://localhost:8080/safeRestart>
- **restart** will force a restart without waiting for jobs to complete  
<http://localhost:8080/restart>

- Installing Jenkins
  - Installing Jenkins from the jar File
  - Installing Jenkins in a Servlet Container
  - Installing Jenkins using Platform Specific Installer

## • **Setup Security**

- Email and Version Control
- Master/Slave Configurations
- Lab 1 : Install and Configure Jenkins

- No Security enabled by default
- Highly recommended to have Security enabled for your Jenkins configuration
- Two Important Security Features
  - **Security Realms**
    - Determines users and their passwords
    - What groups the users belong to
  - **Authorization Strategy**
    - Who has access to what
    - what users can do once they are logged in

# Security Configuration

 **Configure Global Security**

**Enable security**

TCP port for JNLP slave agents  Fixed :   Random  Disable

Disable remember me

Access Control

**Security Realm**

- Delegate to servlet container
- Jenkins' own user database
- LDAP

**Authorization**

- Anyone can do anything
- Legacy mode
- Logged-in users can do anything
- Matrix-based security
- Project-based Matrix Authorization Strategy

Markup Formatter

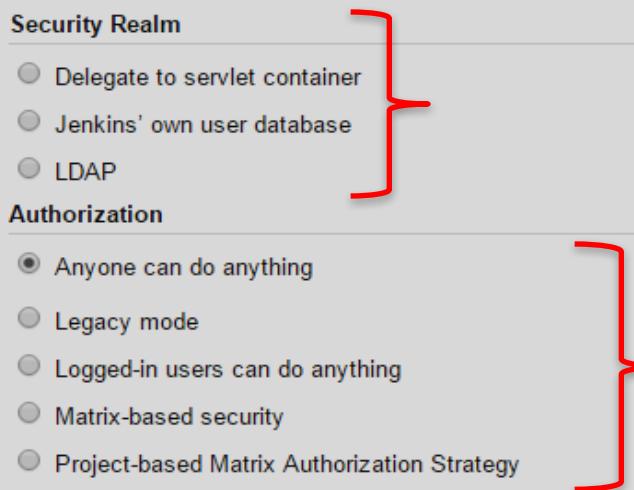
Plain text

Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

Prevent Cross Site Request Forgery exploits

Use browser for metadata download

**Save** **Apply**

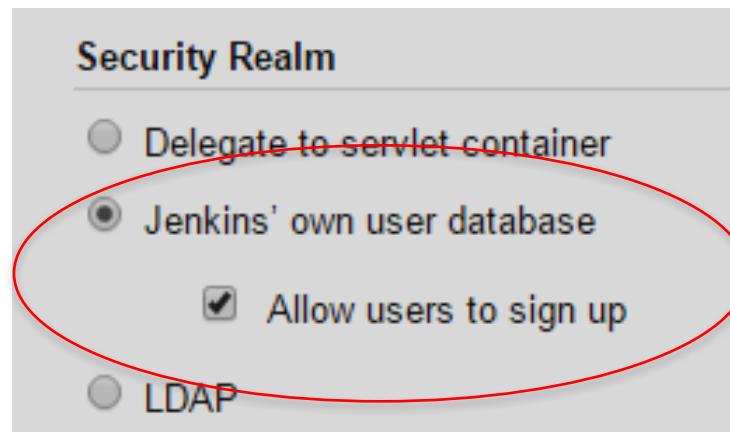


- Identify Jenkins users and establish user authentication methods
- Few options for user authentication methods
  - Jenkin's Built-in User Database
  - Using an Active Directory
  - Using Unix User and Groups
  - Delegating to the Servlet Container

- Simplest and most popular authentication method for Jenkin's users
  - Very little configuration is required
- Jenkins maintains its own user database
- Users can sign up for their own accounts
- Administrator can decide what these users are allowed to do
- Adds all SCM users automatically to this internal database

# How to enable Jenkins Database

- Click Enable Security
- Click on Jenkins' own user database option
- Then click on Allows users to sign up



# View and Manage Users

By clicking on People item on dashboard, you can see list of all users

The screenshot shows the Jenkins dashboard with the 'People' item selected from the left sidebar. The main content area is titled 'People' and contains a table listing two users: 'admin' and 'developer1'. The table includes columns for User Id, Name, and Last Activity. Below the table, there is a note about including known users and a link to commit message history. On the left, there are sections for 'Build Queue' (empty) and 'Build Executor Status' (two idle executors).

User Id	Name	Last Activity
<a href="#">admin</a>	<a href="#">Administrator</a>	N/A
<a href="#">developer1</a>	<a href="#">Developer1</a>	N/A

By clicking Manage Users link in Jenkins configuration page, you can configure users

The screenshot shows the Jenkins configuration page with the 'Manage Jenkins' item selected from the left sidebar. The main content area is titled 'Users' and displays a table of users. It includes a note about auto-created users and a link to a full user list. The table lists the same two users: 'admin' and 'developer1'.

User Id	Name	Action
<a href="#">admin</a>	<a href="#">Administrator</a>	
<a href="#">developer1</a>	<a href="#">Developer1</a>	

- Identify what users are allowed to do once authenticated
- Offers different strategies for most precise control
- Matrix-based Security is a more sophisticated approach

## Authorization

- Anyone can do anything
- Legacy mode
- Logged-in users can do anything
- Matrix-based security
- Project-based Matrix Authorization Strategy

- Role-based approach - Different users will be created with different access
- First to create an administrator user with all access
- Give the Anonymous user only Read access
- Add other users and grant them necessary access

Matrix-based security

User/group	Overall										Credentials									
	Administer	Configure	UpdateCenter	ReadRunScripts	UploadPlugins	Create	Delete	ManageDomains	UpdateViewBu	ViewBu	ViewBu									
admin	<input checked="" type="checkbox"/>																			
developer1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>						
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>																		

User/group to add:

# Common Permissions

- Overall
  - Administer access
  - Run groovy scripts
  - Upload plugins or configure update center
- Job
  - Create, Delete and Configure Jobs
  - Start a new build or Cancel a running build
- View
  - Create, Delete and Configure views
- SCM
  - Create a new tag in source code repository for any build
- Slave
  - Create, Delete and Configure slaves
  - Connect or Disconnect slaves

- Installing Jenkins
  - Installing Jenkins from the jar File
  - Installing Jenkins in a Servlet Container
  - Installing Jenkins using Platform Specific Installer
- Setup Security
- **Email and Version Control**
- Master/Slave Configurations
- Lab 1 : Install and Configure Jenkins

# Mail Server Configuration

- Email notification is fundamental notification technique
- Mailer Plugin allows you to configure **email notifications**
- Simple email setup
  - Provide SMTP server
  - Just add organization domain name
    - If this field is set to @myCompany.org, then user foo will by default get the e-mail address foo@myCompany.org



# Advanced Email Configuration

- Mailer plugin provides advanced email configuration such as SMTP authentication and SSL
- Can configure Gmail service with user name and password as your email server

E-mail Notification

SMTP server	smtp.gmail.com	?
Default user e-mail suffix		?
<input checked="" type="checkbox"/> Use SMTP Authentication		?
User Name	jenkinscourse@gmail.com	?
Password	.....	?
Use SSL	<input checked="" type="checkbox"/>	?
SMTP Port	465	?

- In-built support for CVS and Subversion
  - Requires no special configuration for both of them
- Plugins are available for other version control like
  - Git
  - ClearCase
  - PVCS
  - Dimensions
  - StarTeam
- Will review **Subversion and Git** configuration in this session

# Working with Subversion

- Subversion is part of Jenkins installation and so no special configuration required
- Simply provide Subversion repository URL in your job definition
  - Jenkins will automatically verify the URL and shows error if it is not valid
  - If repository URL requires authentication then Jenkins will prompt for credentials

- First Install Git on your build server
  - Include Git executable in your system path
- Second install Git Plugin using Jenkins Plugin Manager
  - This plugin allows use of Git as a build SCM

The screenshot shows the Jenkins Plugin Manager interface. A sidebar on the left lists several plugins with checkboxes. The 'GIT plugin' checkbox is checked, while others like 'REPO plugin', 'Google Git Notes Publisher Plugin', 'Embeddable Build Status Plugin', and 'JGit plugin' are unchecked. The main panel displays detailed information about the selected plugin.

**REPO plugin**  
 This plugin adds Repo ([\[http://code.google.com/p/git-repo/\]](http://code.google.com/p/git-repo/)) as an SCM provider in Jenkins.

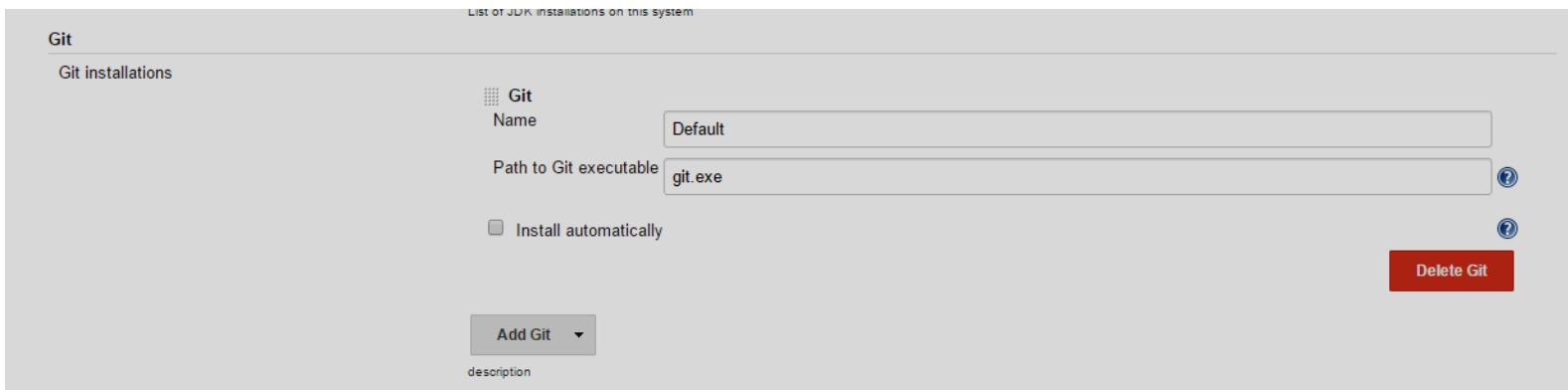
**Google Git Notes Publisher Plugin**  
 This plugin provides automatic recording of Jenkins build actions to Git Notes.

**GIT plugin**  
 This plugin allows use of [Git](#) as a build SCM. A recent Git runtime is required (1.7.9 minimum, 1.8.x installations at your own risks).

**Embeddable Build Status Plugin**  
 This plugin allows Jenkins to expose the current status of your build as an image in a fixed URL. You can see the current state of the job (last build) or for a specific build.

# Git Configuration Section

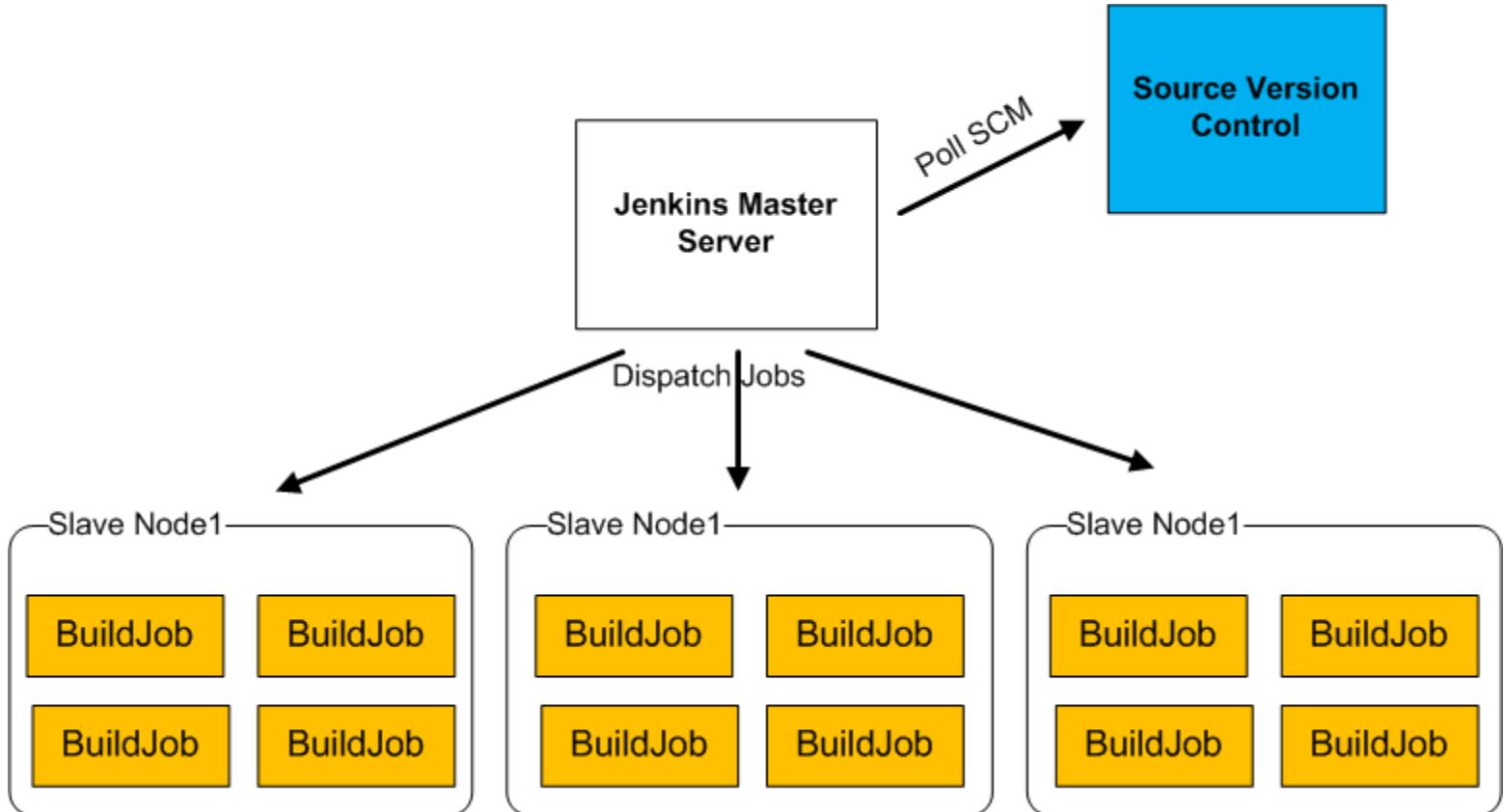
- After the successful installation of Git Plugin, you will see Git section in Configure System page
- You can add newer and older versions of Git executables in this section



- Installing Jenkins
  - Installing Jenkins from the jar File
  - Installing Jenkins in a Servlet Container
  - Installing Jenkins using Platform Specific Installer
- Setup Security
- Email and Version Control
- **Master/Slave Configurations**
- Lab 1 : Install and Configure Jenkins

- Jenkins supports powerful feature for running build jobs across a large number of machines using Master/Slave configuration
  - One Master with one or more Slave agents
- **Main Jenkins server is the Master**
  - Master is the basic installation of Jenkins
  - Master is responsible for scheduling and dispatching build jobs to the slaves, and presenting build results
  - But the same time, Master can also execute build jobs itself
- **Slaves are small executable that runs on remote machines**
  - Slaves are responsible for executing build jobs dispatched by the master
  - No need to install full Jenkins on a slave
  - Slaves instance can be started in a number of different ways depending on the operating system

# Master/Slaves Architecture



- Installing Jenkins
  - Installing Jenkins from the jar File
  - Installing Jenkins in a Servlet Container
  - Installing Jenkins using Platform Specific Installer
- Setup Security
- Email and Version Control
- Master/Slave Configurations
- **Lab 1 : Install and Configure Jenkins**

# Lab 1 : Install and Configure Jenkins

- In this lab, you will learn how to install and configure Jenkins. This lab is prerequisite for the remaining labs in this training.
  - Install and Run Jenkins as standalone application
  - Enable Security to access and use Jenkins
  - Setup Java, Maven and Email configuration
  - Configure Gradle
  - Configure Git Version Control

# Jenkins Job

- **Create a job**
- Configure a Job
  - Configure a Freestyle Job
  - Configure a Maven Job
  - Copy from Existing Jobs
- Trigger a Build Job
  - Run a job manually
  - Run a job on a regular schedule
  - Run a job when source code is checked into version control
  - Run a Job After Another Job Finished
- Lab 2 – Build and Configure your first Jenkins job

- A job defines a **sequence of repetitive tasks** for Jenkins to perform
- A job could be any of these
  - Compiling project
  - Testing project
  - Packaging project
  - Deploying project to different environment
  - Running code quality metrics

- Jenkins supports different types of build jobs
  - **Freestyle Job** : general purpose jobs for combining any SCM with any build system. Most commonly used with Gradle builds
  - **Maven Job** : jobs specifically for Maven project to take advantage of POM files
  - **External Job** : record or monitor the execution of a process on local or remote servers
  - **Multi-configuration Job** : run the same job in different configurations
- Jenkins also allows you to **copy from existing jobs** – very useful feature for quickly creating new jobs
- Freestyle with Gradle and Maven Jobs are the most common build jobs

# First Build Job

Item name

Jenkins-Lab2-Build

Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, a

Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

External Job

This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. automation system. See [the documentation for more details](#).

Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments

OK

- Jenkins has **built-in standard List Views** where you can group jobs in different views
- The additional View plugins are available for grouping and categorizing the jobs

Grouping all build jobs

Grouping all deployment related jobs

The screenshot shows a Jenkins List View interface. At the top, there are four tabs: 'All' (disabled), 'Build Jobs' (selected and highlighted in blue), 'Deployment Jobs', and 'Regression Jobs'. Below the tabs is a table with three rows of build jobs. The columns are labeled 'S' (Status), 'W' (Last Success), 'Name', and 'Last Success'. The first job is 'Jenkins-build1', the second is 'Jenkins-Lab2-Build' (last success 2 days 2 hr - #1), and the third is 'Jenkins-Lab3-Build'. At the bottom left, there is an icon section labeled 'Icon: S M L'.

S	W	Name	Last Success
●	☀	<a href="#">Jenkins-build1</a>	N/A
●	☀	<a href="#">Jenkins-Lab2-Build</a>	2 days 2 hr - #1
●	☀	<a href="#">Jenkins-Lab3-Build</a>	N/A

Icon: [S](#) [M](#) [L](#)

Grouping all regression related jobs

- Create a job
- **Configure a Job**
  - Configure a Freestyle Job
  - Configure a Maven Job
  - Copy from Existing Jobs
- Trigger a Build Job
  - Run a job manually
  - Run a job on a regular schedule
  - Run a job when source code is checked into version control
  - Run a Job After Another Job Finished
- Lab 2 – Build and Configure your first Jenkins job

- Any Freestyle or Maven job usually consists the following elements
  - Start with general job properties such as job name and description
  - Configure Version Control System such as Git or Subversion to obtain the project source code
  - Define build steps using Ant, Gradle, Maven, shell script, batch file or calling other Jenkins job
  - Include post-build actions for archiving artifacts or email notification on build status
- Version Control System and Post-build actions are optional steps

# General Options – Name and Old Builds

- Name and Description
  - Give the proper name and describe the purpose of the job
- Discard Old Builds
  - Limits the number of builds to keep
  - Important setting for managing the disk space otherwise it will store all previous build artifacts

Project name

Description   
[\[Plain text\]](#) [Preview](#)

Discard Old Builds ?

Strategy

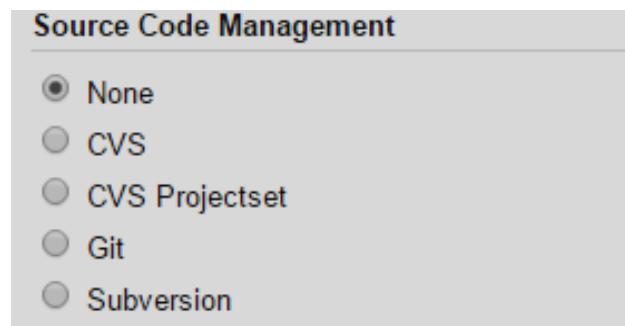
Days to keep builds

Max # of builds to keep  5

If not empty, build records are only kept up to this number of days

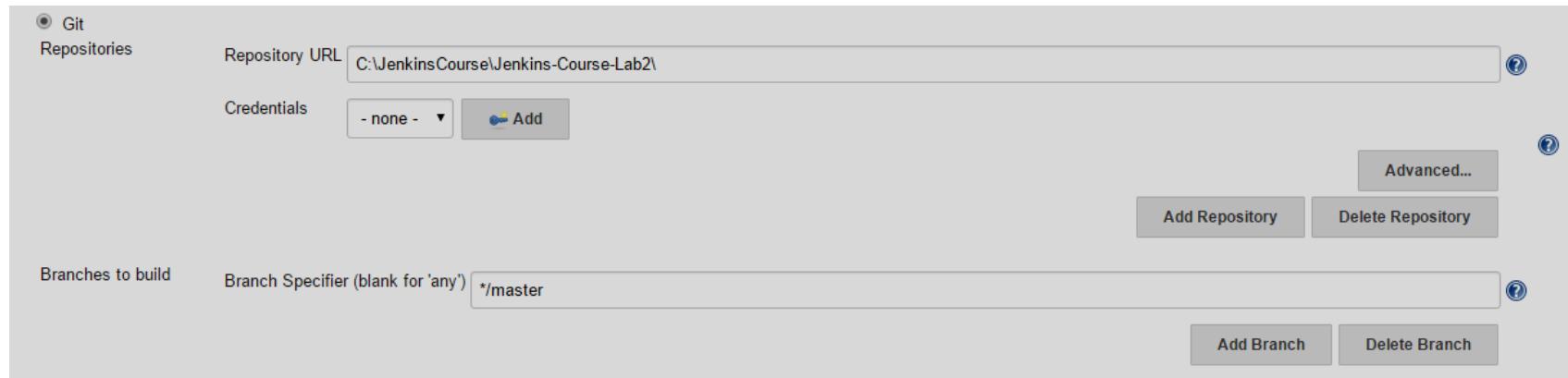
If not empty, only up to this number of build records are kept

- Jenkins can integrate with pretty much any version control system
  - Subversion and CVS are out the box
  - We installed Git plugin in earlier session
  - Once you configure particular version control system, Jenkins check out and builds the latest version of source code as continuous monitoring or regular interval

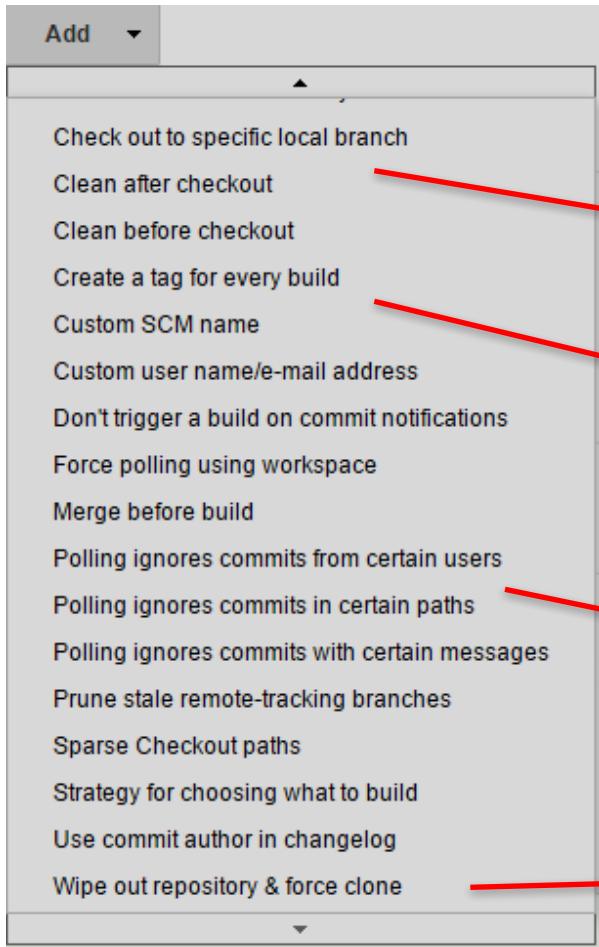


# Working with Git

- In most cases, you only need to provide **URL of the Git repository**
- You can have the specific branch name that should be built if not the default master branch
  - If blank or populated \* then it will build all branches



# Git – Additional Functionalities



Cleaning any untracked files after or before checkout

Configure to create new tag for every successful build

Specify to ignore certain users, paths or messages

Option for deleting and rebuilding workspace for every build

# Working with Subversion

- Simply provide **Subversion repository URL** (similar to Git)
- Jenkins will automatically verify the URL and prompts for authentication if credentials are required
- Jenkins offers few choices for selecting check-out Strategy
  - From fastest option (svn update as much as possible) to slowest and safest option of checking out fresh copy all the time

The screenshot shows the Jenkins configuration interface for a Subversion repository. The 'Subversion Modules' section is selected. The 'Repository URL' field is highlighted in red with the error message 'Repository URL is required.' Below it, the 'Local module directory (optional)' field and 'Repository depth' dropdown (set to 'infinity') are shown. The 'Ignore externals' checkbox is unchecked. A 'Check-out Strategy' dropdown is set to 'Use 'svn update' as much as possible'. A tooltip explains: 'Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.' The 'Repository browser' dropdown is set to '(Auto)'. A 'Add more locations...' button is visible.

Subversion Modules

Repository URL

Repository URL is required.

Local module directory (optional)

Repository depth

infinity

Ignore externals

Add more locations...

Check-out Strategy

Use 'svn update' as much as possible

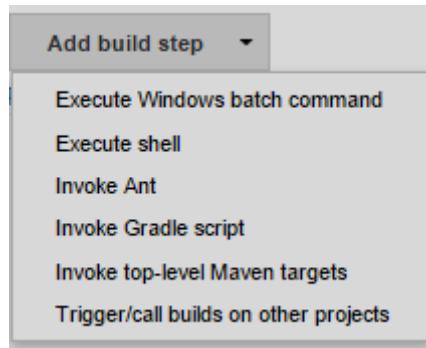
Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Repository browser

(Auto)

# Build Steps and Tools

- Define the steps for building your project
  - Some jobs may need more than one build steps or others may not need any build steps
- Built-in support for following build tools
  - Invoke Maven targets
  - Invoke Ant scripts
  - Running Windows batch commands or OS specific shell scripts
- Other build tools such as Groovy, Gradle, Ruby, Python, Rake can be integrated with installing additional plugins



# Gradle Build Steps

- There are two ways to run the Jenkins jobs with Gradle
  - Using Invoke Gradle through Gradle installation
  - Using Gradle Wrapper without Gradle installation
- In both options, you are only listing **gradle tasks** for the most Jenkins jobs

The screenshot shows the Jenkins 'Build' configuration page. On the left, there's a sidebar with various options: 'Invoke Gradle script' (unchecked), 'Invoke Gradle' (checked), 'Gradle Version' (set to 'Gradle 2.10'), 'Use Gradle Wrapper' (checked), 'Build step description' (empty), 'Switches' (empty), 'Tasks' (highlighted with a red circle), 'Root Build script' (empty), 'Build File' (empty), and 'Force GRADLE\_USER\_HOME to use workspace' (unchecked). The main area has several input fields for 'Tasks': 'clean build' is listed in the first field, and there are three more empty fields below it. Each field has a help icon (a question mark inside a blue circle) and a delete icon (a red square with a white minus sign).

# Maven Build Steps

- Maven build steps are also simple and easy to configure
- You only need to **enter Maven goal** that you want to run



# Maven Build Steps – Advanced Properties

- POM field is for overriding the default pom.xml file location
- Properties field is used to pass properties into Maven build process (like –D with maven goal)
- JVM options for configuring more memory with maven build
- You can specify different maven repository or settings file

Invoke top-level Maven targets	
Maven Version	(Default)
Goals	clean install
POM	
Properties	
JVM Options	
Use private Maven repository	<input type="checkbox"/>
Settings file	Use default maven settings
Global Settings file	Use default maven global settings

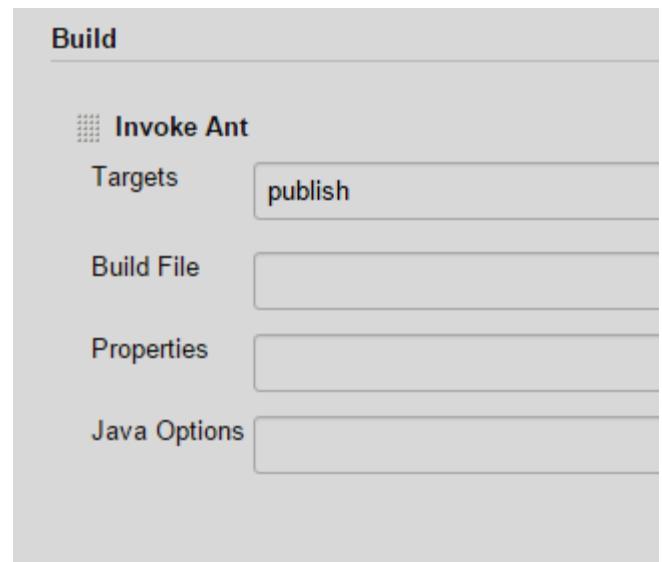
# Ant Build Steps

- Configuring as Ant build step is as simple as Maven build steps
- You just need to provide **the name of Ant target** that you want to run
- If it is default main target in build.xml, then you are not required to provide target name in the build step

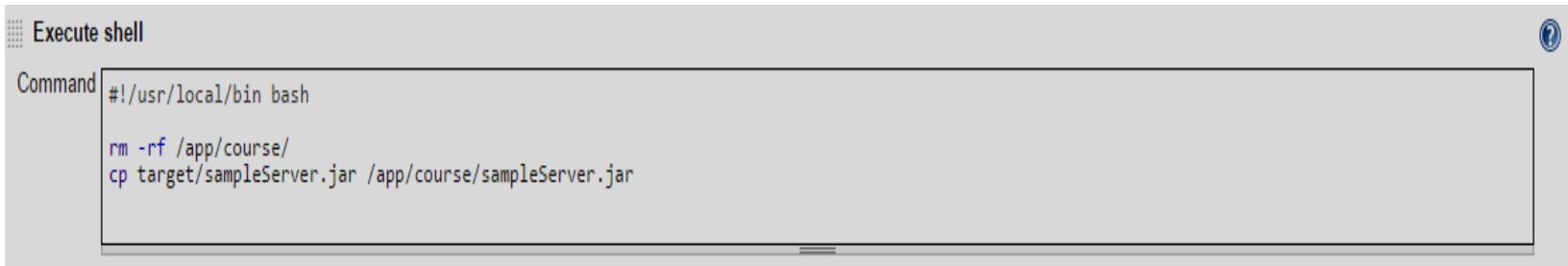


# Ant Build Steps – Advanced Properties

- Build File option can be used to override the default build file (build.xml in root directory)
- Properties option is used to pass the properties to Ant script
- Java Options can be used to specify Java memory limits



- You can also execute shell script local or remote to run some commands such as copying the artifacts to server folder
- If the shell script does not have header like #! then the default shell configuration will be used
- You can **use SSH** for run the commands on the remote server

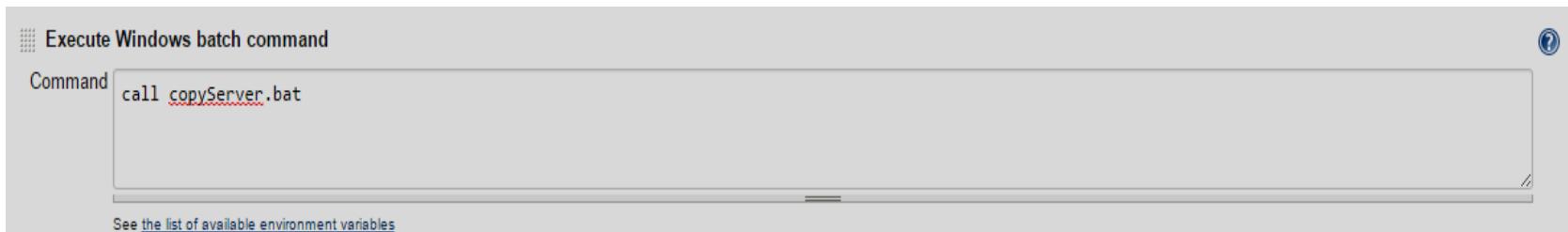


The screenshot shows a software interface titled "Execute shell". In the "Command" field, the following shell script is entered:

```
#!/usr/local/bin bash
rm -rf /app/course/
cp target/sampleServer.jar /app/course/sampleServer.jar
```

# Execute Windows Batch Command

- Run Windows commands or batch scripts. It will be run in the workspace as the current directory
- The Batch command is running inside a cmd
  - No need to specifically start a new one
  - Just "call" your BAT file



- Optional steps for collecting or reporting information out of the build or notifying other people or systems
- The following are the out of the box Post Build Actions
  - Publish Javadoc
  - Publish JUnit test result report
  - Archive the artifacts
  - Email Notification about build results
  - Trigger other Jenkins jobs
- You can install additional plugins to add more Post Build Action Items
  - Such as build status notification to Slack channel using Jenkins Slack Plugin

# Post Build – Email Notification

- Click on Email Notification from Post Build Actions and enter email addresses to inform when the build breaks
- Can configure to send separate emails to individuals who's last commits broke the build

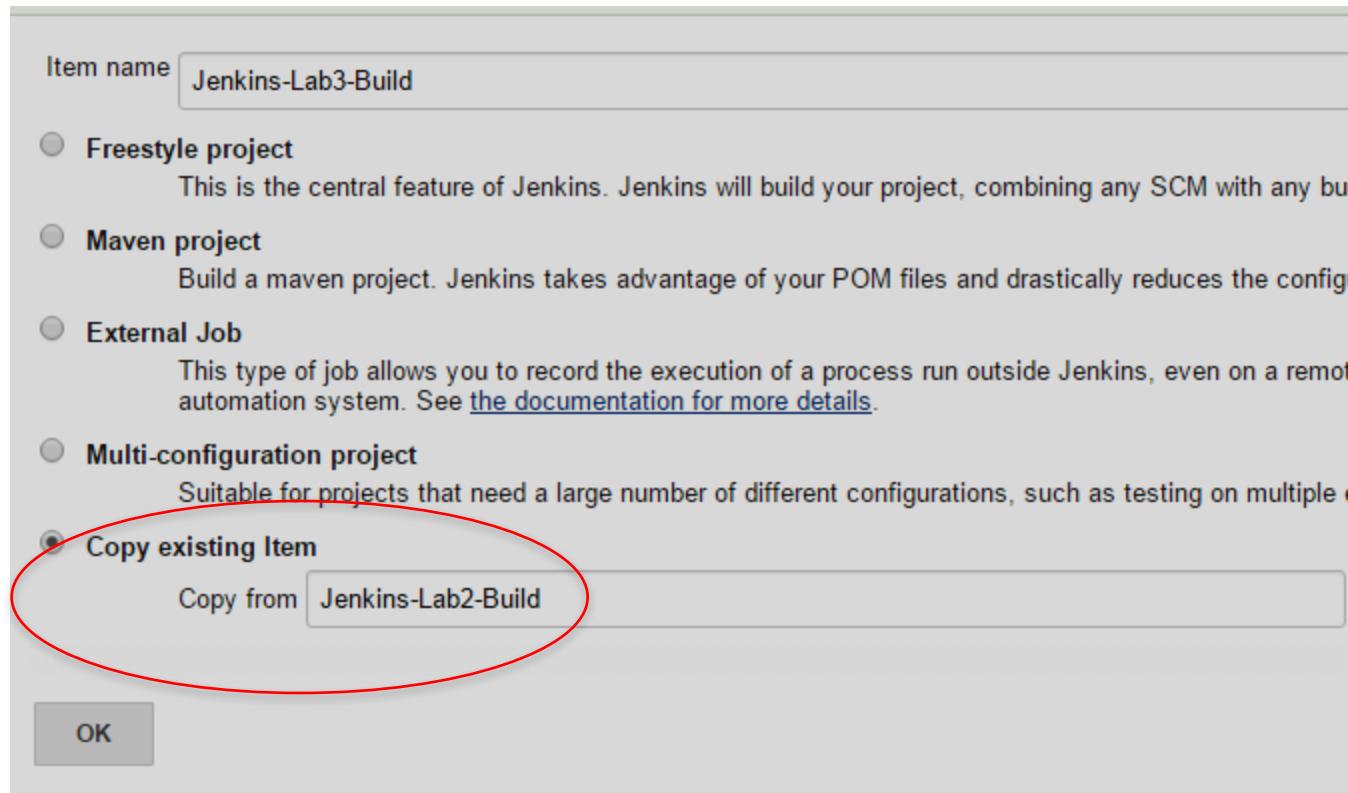
The screenshot shows the Jenkins 'Post-build Actions' configuration page. Under the 'E-mail Notification' section, the 'Recipients' field contains 'jenkinscourse@gmail.com'. A note below the field specifies: 'Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.' Two checkboxes are present: 'Send e-mail for every unstable build' (checked) and 'Send separate e-mails to individuals who broke the build' (unchecked). A red 'Delete' button is located at the bottom right of the configuration block.

# Configure Maven Job

- Jenkins provides excellent built-in Maven support to take advantage of Maven POM file
- The job elements are still the same as FreeStyle build job, but it is much more straight forward with few differences (listed below) for configuring Maven job
- Jenkins builds maven jobs automatically whenever a SNAPSHOT dependency is built
  - Jenkins read dependencies from POM and if any of the SNAPSHOT dependency changes then it automatically trigger to build your job
- Build Section has only one step for invoking a single Maven goal
- Post Build Actions provide extra option for deploying job artifacts to Maven repository such as Nexus or Artifactory

# Copy From Existing Job

- Jenkins provides a very useful way to create new job by just **copying an existing job**
  - Quick option for creating jobs similar to existing jobs



- Create a job
- Configure a Job
  - Configure a Freestyle Job
  - Configure a Maven Job
  - Copy from Existing Jobs
- **Trigger a Build Job**
  - Run a job manually
  - Run a job on a regular schedule
  - Run a job when source code is checked into version control
  - Run a Job After Another Job Finished
- Lab 2 – Build and Configure your first Jenkins job

- Most likely the job will trigger automatically based on some external factors or regular schedule
- You may want to run the job manually in some scenario using human intervention
- Don't configure anything in the build trigger section for Manual jobs

**Build Triggers**

<input type="checkbox"/>	Trigger builds remotely (e.g., from scripts)
<input type="checkbox"/>	Build after other projects are built
<input type="checkbox"/>	Build periodically
<input type="checkbox"/>	Poll SCM

The screenshot shows the Jenkins interface for the project 'Jenkins-Lab2-Build'. On the left, there's a sidebar with links: 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now' (which has a red arrow pointing to it), 'Delete Project', and 'Configure'. Below these are sections for 'Build History' (one entry for build #1 on Oct 21, 2015, at 6:24 AM) and 'RSS feeds' (for all and failures). The main content area is titled 'Project Jenkins-Lab2-Build' and contains links for 'Workspace', 'Last Successful Artifacts' (a file named 'Jenkins-Course-Lab2-1.0.jar' is shown, 3.89 KB), and 'Recent Changes'. At the bottom, there's a 'Permalinks' section with three items: 'Last build (#1), 2 days 0 hr ago', 'Last stable build (#1), 2 days 0 hr ago', and 'Last successful build (#1), 2 days 0 hr ago'.

You can always run any job manually with **Build Now** option

# Run a job on a regular schedule

- Trigger the build job at regular interval such as nightly build
- Some long running jobs are good candidates for scheduled jobs
  - Jobs for generating and publishing code quality metrics and reports on Sonar
- Jenkins uses **cron-style syntax** to schedule the jobs at regular interval

**Build Triggers**

Trigger builds remotely (e.g., from scripts)  
 Build after other projects are built  
 Build periodically

Schedule @midnight

Would last have run at Friday, October 23, 2015 1:54:36 AM EDT; would next run at Saturday, October 24, 2015 1:54:36 AM EDT.

Poll SCM

- You can use @yearly, @annually, @monthly, @weekly, @daily, @midnight, @hourly to schedule the jobs
- OR You can use cron style syntax in the following format

## MINUTE HOUR DOM MONTH DOW

- MINUTE – minutes within the hour (0-59)
  - HOUR – hour of the day (0-23)
  - DOM – day of the month (1-31)
  - MONTH – the month (1-12)
  - DOW – day of the week (0-7) where 0 and 7 are Sunday
  - Examples
    - \* \* \* \* \* represents once a minute
    - H/15 \* \* \* \* represents at every fifteen minutes
    - H 1 \* \* represents once a day on the 1<sup>st</sup> of every month
- Note: Jenkins encourages the use of 'H' to balance load across different times of day to avoid sudden spikes of activities

- **Polling the SCM** is the best strategy for continuous integration builds
  - You want to build your jobs as quickly as there are changes in SCM
  - Manual and Scheduled builds are applicable in few scenarios but all other jobs should be configured with Polling SCM option
- Jenkins polls the version control server at regular interval if any changes have been committed
  - If changes are committed then the Jenkins will build your job
- Polling is very quick for Git and Subversion, however this is not effective solution for CVS
  - Alternatively Jenkins provides the post-commit hook script for triggering you build remotely

- Jenkins uses the **same cron style syntax** to configure polling schedule for SCM

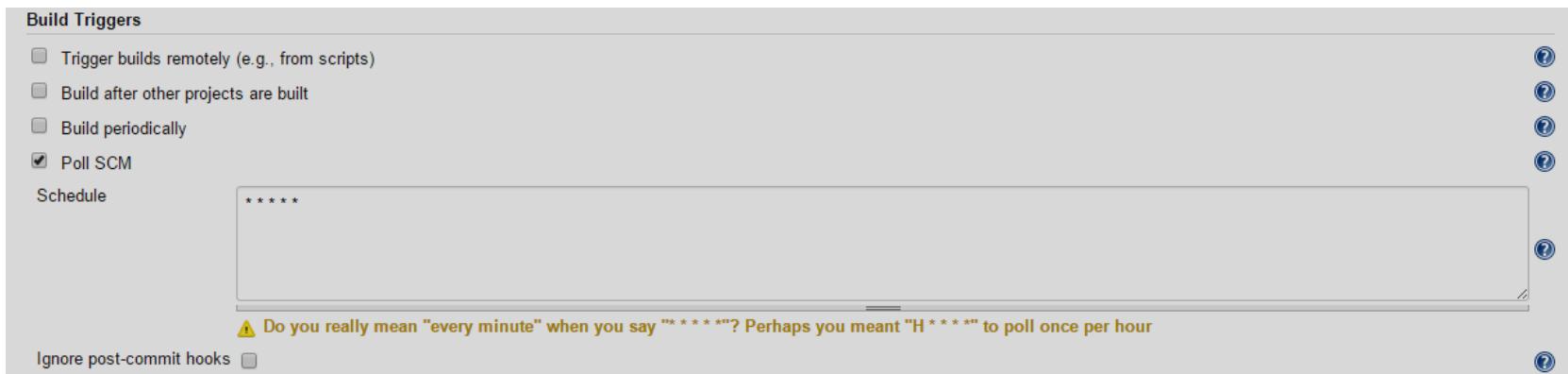
**Build Triggers**

- Trigger builds remotely (e.g., from scripts) ?
- Build after other projects are built ?
- Build periodically ?
- Poll SCM ?

Schedule

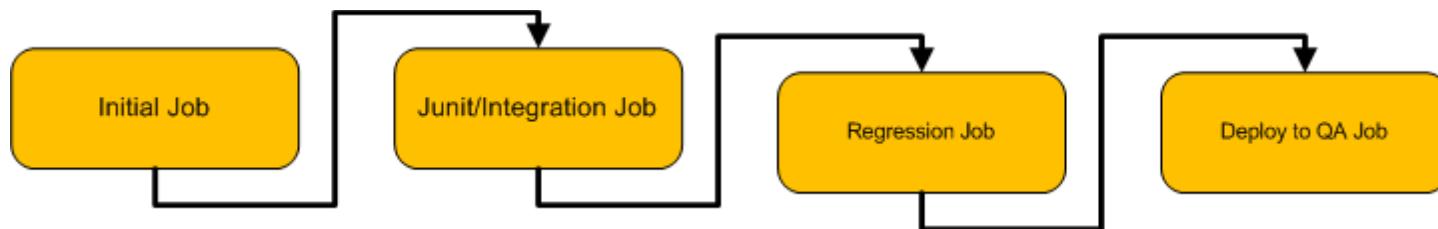
⚠ Do you really mean "every minute" when you say "\*\*\*\*\*"? Perhaps you meant "H \* \* \* \*" to poll once per hour

Ignore post-commit hooks  ?



# Run a Job After Another Job Finished

- Jenkins also provides the option for running your job whenever another jobs finish building
- You can specify one or more preceding build jobs to trigger your new job
- This is useful feature for **setting up a build pipeline**



- When you configure this option for your new job, it will automatically configure the "Build other projects" section in the "Post-build Actions" of the preceding job
  - Basically, this configuration complements the "Build other projects" section in the "Post-build Actions" of an upstream project
- This configuration also provide the option for triggering the new build even if the preceding build is unstable or fails

The screenshot shows the 'Build Triggers' configuration page for a Jenkins job. The 'Build after other projects are built' checkbox is checked. A dropdown menu for 'Projects to watch' contains the entry 'Jenkins-build1'. Below this, under 'Trigger only if build is stable', there are three radio button options: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. At the bottom, there are two unchecked checkboxes: 'Build periodically' and 'Poll SCM'.

# Topics in this Session

- Create a job
- Configure a job
  - Configure a Freestyle job
  - Configure a Maven job
  - Copy from Existing jobs
- Trigger a Build Job
  - Run a job manually
  - Run a job on a regular schedule
  - Run a job when source code is checked into version control
  - Run a job after another job finished
- **Lab 2 – Build and Configure your first Jenkins job**

# Lab 2 – Build and Configure your first Jenkins job

- In this lab you will configure and run your first Jenkins job to build your java project
  - Create new Freestyle job with Gradle build
  - Configure your Freestyle job to poll changes from Git project
  - Configure email notification for failed builds
  - Build and run tests for your java project

# Jenkins Plugins

- **Code Coverage**
  - Static Code Analysis
  - Performance Reporting
  - Change Reporting
  - Lab 3 – Build a Jenkins job to generate code quality reports

- In the previous session, our focus was the continuous build with the new changes from the version control
- In this session, we will focus on **how to use Jenkins to build quality product** using different Jenkins plugins
  - Run tests and generate code coverage
  - Perform static code analysis
  - Run performance tests
- Jenkins has more than 1000 plugins and there is no way we can even list all of those plugins
  - Will review few popular plugins especially the ones are used for building quality code

- Automated tests and code coverage metrics are the important steps in continuous integration and continuous delivery
- Code Coverage is a very good indicator of how thoroughly your tests exercise your code base and finding areas of code that have not been tested by automated tests
- Jenkins provides excellent support in generating code coverage metrics
  - Jenkins supports pretty much all popular code coverage tools
  - [Cobertura](#), [Emma](#), [Clover](#), [JaCoCo](#)
- We will review how to use Cobertura with Jenkins in this session
  - the same steps are used for other code coverage plugins too

# Code Coverage Using Cobertura Plugin

- First, install Cobertura plugin using Manage Plugins screen and restart Jenkins
- Apply Cobertura plugin in Gradle or Maven build script
  - Update your project's Maven POM file to include Cobertura plugin and then run `mvn cobertura:cobertura`
  - Update build.gradle file to apply Cobertura plugin and then run `gradle cobertura` task
- Configure Cobertura Coverage Report as part of “Post Build Actions”

# Adding Cobertura in build.gradle

```
apply plugin: 'cobertura' //required step
buildscript {
    repositories {
        mavenCentral()
    }
    // How to find Cobertura
    dependencies {
        classpath 'net.saliman:gradle-cobertura-
        plugin:2.2.4'
    }
}
//Optional section for Cobertura properties
cobertura {
    coverageFormats = ['html', 'xml']
}
```

# Configure Jenkins Job to generate Cobertura XML

- Next thing is to configure your Jenkins job to produce Cobertura xml coverage files
- You will just add gradle cobertura task to the gradle build step

Build

Invoke Gradle script

Invoke Gradle  
Gradle Version **Gradle 2.10**

Use Gradle Wrapper  
Build step description

Switches

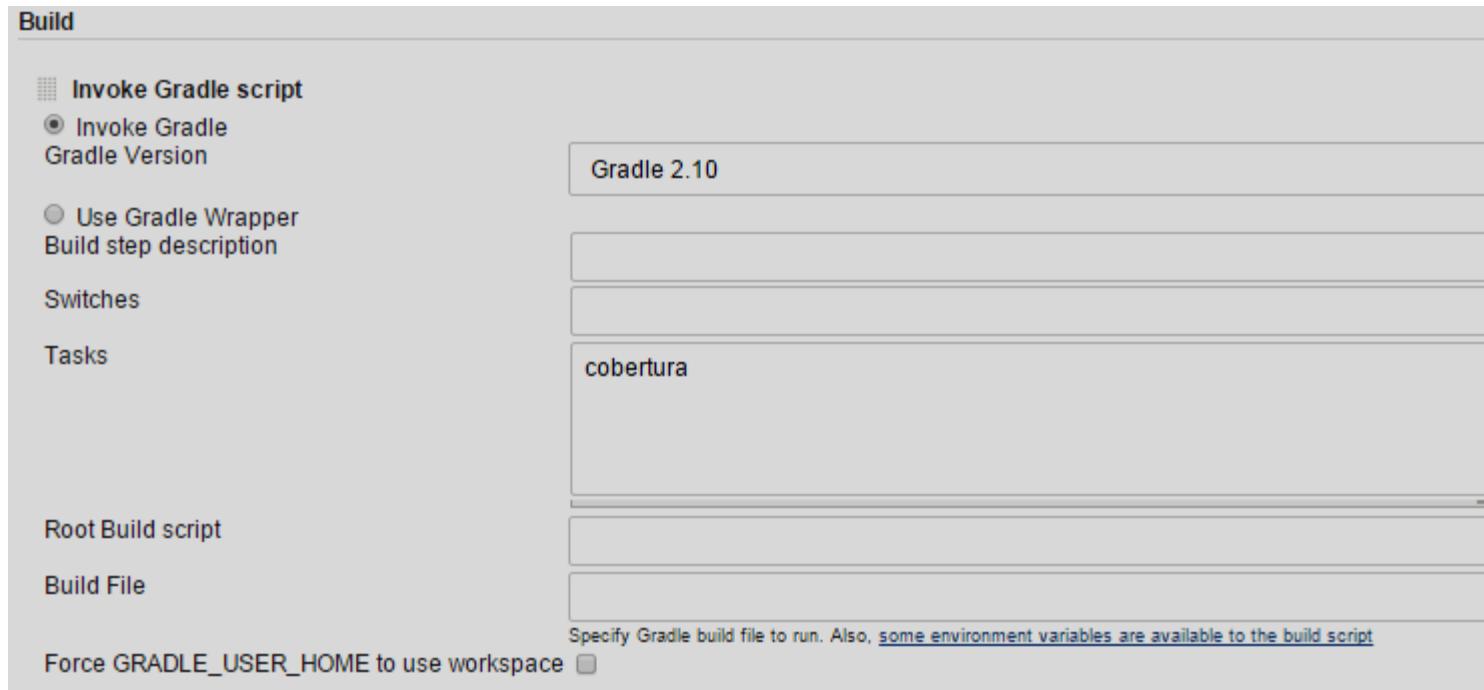
Tasks **cobertura**

Root Build script

Build File

Specify Gradle build file to run. Also, [some environment variables are available to the build script](#)

Force GRADLE\_USER\_HOME to use workspace



# Adding Cobertura in Post Build Actions

Specify path to coverage.xml file

**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

Fail builds if no reports

Fail unhealthy builds

Fail unstable builds

Health auto update

Stability auto update

Zoom coverage chart

Maximum number of builds

Source Encoding

Coverage Metric Targets

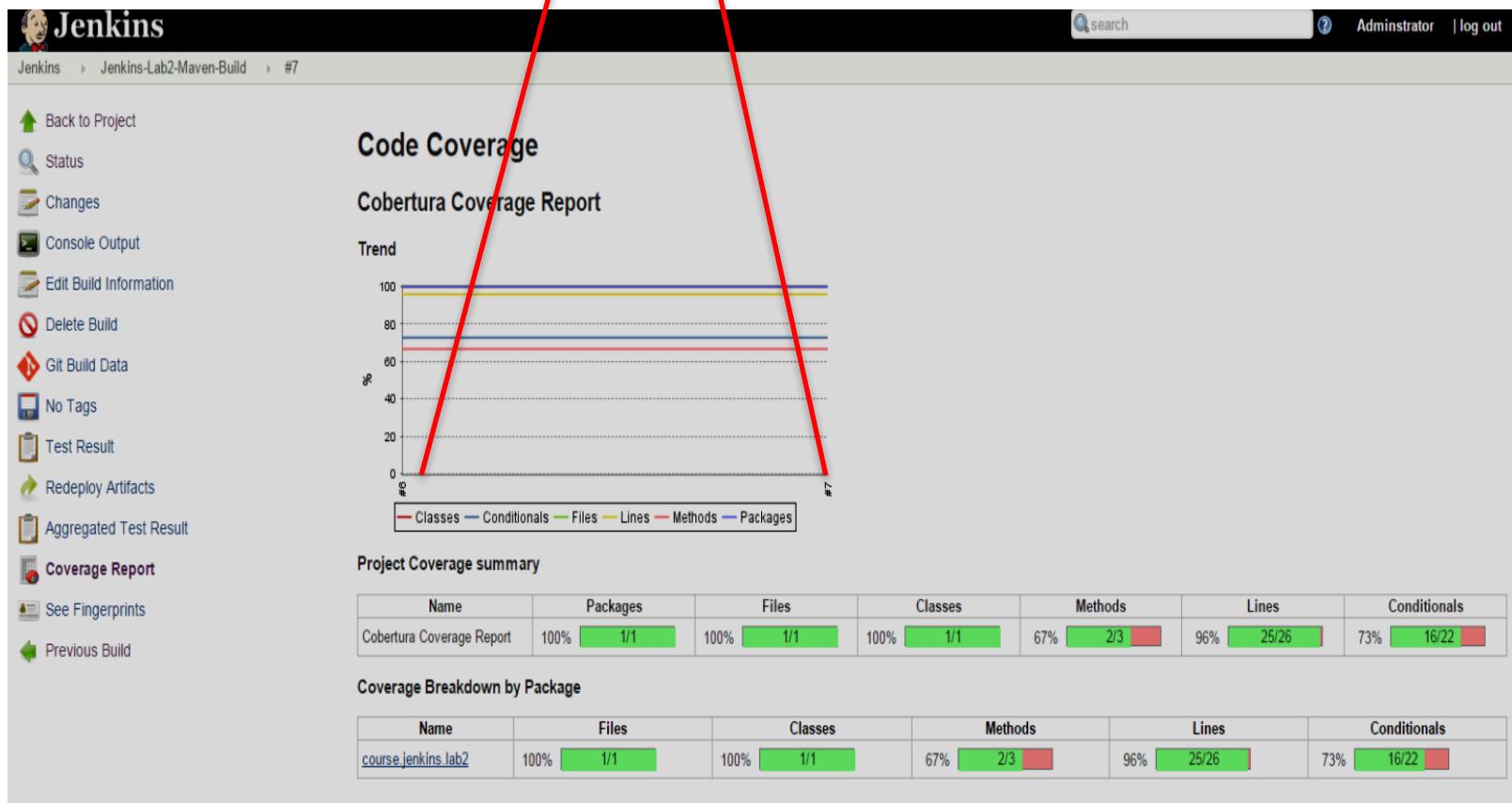
Target Type	Value	Icon
Methods	80.0	sun
Lines	80.0	cloud
Conditionals	70.0	yellow dot

Consider the acceptable levels of code coverage

# Code Coverage Report

Build your jenkins job and you will able to see the code coverage report on your build job page

## Code Coverage over the time



- Code Coverage
- **Static Code Analysis**
- Performance Reporting
- Change Reporting
- Lab 3 – Build a Jenkins job to generate code quality reports

- Static Code Analysis tools help to produce quality code by analyzing the source and byte code for possible defects and enforcing coding standards
  - Static Code Analysis tools play the role of automated pair programming with developers
- We will review few popular static code analysis tools with Jenkins
  - [FindBugs](#) is a widely used byte code analyzer tool for scanning your code to detect possible defects
  - [CheckStyle](#) is a source code analyzer tool for enforcing Java coding standards based on your rule set
  - [PMD](#) is another source code analyzer that finds common flaws such as unused objects, duplicate code, empty blocks, unnecessary caches and others based on your rule set

# Configuring FindBugs in build.gradle

- You can apply Findbug plugin in build.gradle
- That's the only required step needed to support Findbugs analysis

```
//required step
apply plugin: 'findbugs'
```

- Additional configuration with findbugs task to specify which bug filter to include or exclude, setting not to fail build and updating effort and reportLevel

```
//Optional configuration
findbugs {
    toolVersion = "2.0.1"
    ignoreFailures = true
    effort = "max"
    reportLevel = "low"
}
```

# Configuring CheckStyle in build.gradle

- The only required step is to apply checkstyle plugin to generate checkstyle analysis with gradle
- Then you can set different properties with checkstyle task as needed

```
apply plugin: 'checkstyle'
```

```
checkstyle {  
    ignoreFailures = true  
}
```

# Configuring PMD in build.gradle

```
//Only required step to enable PMD
apply plugin: 'pmd'

/*
Other properties and ruleSets can be set
under pmd task
*/
pmd {
    ignoreFailures = true
    ruleSets = ["java-basic", "java-braces"]
}
```

- Install static code analysis plugins through plugin manager
  - The plugins related static code analysis is available under Build Reports category
  - You can install Findbugs, PMD and Checkstyle plugins individually
  - Then you can also install Analysis Collector Plugin and Static Analysis Utilities plugins for a combined trend graph

The screenshot shows the Jenkins plugin manager interface. It displays two entries:

- Static Analysis Collector Plug-in**:
  - A checked checkbox icon.
  - Description: "This plug-in is an add-on for the plug-ins Checkstyle, Dry, FindBugs, PMD, Tasks, and Warnings: the plug-in collects the different analysis results and shows the results in a combined trend graph. Additionally, the plug-in provides health reporting and build stability based on these combined results."
  - A link labeled [1.45](#).
- Static Analysis Utilities**:
  - A checked checkbox icon.
  - Description: "This plug-in provides utilities for the static code analysis plug-ins."
  - A link labeled [1.74](#).

- Gradle should include the following tasks to generate static code analysis reports as part of Jenkins build section of a job
  - You can either use `gradle check`
  - You can specify each task separately such as `pmd`, `findbugsMain`, `checkstyle`
- Select the checkbox for generating FindBugs, Checkstyle and PMD reports in build settings
- You can also select Publish combined analysis reports for aggregated results from FindBugs, Checkstyle and PMD

# Static Code Analysis Reports in Build Page

[!\[\]\(e6976676aa10efd5e576f6e1fd69439c\_img.jpg\) Coverage Report](#)[!\[\]\(4acb8c5b11dec3ef08a6617d3e3350fe\_img.jpg\) Workspace](#)[!\[\]\(bd057a9297db5e30e4c1c49a83edfe09\_img.jpg\) Recent Changes](#)[!\[\]\(7c54ca886a803b15c3d9b2957de7ae5a\_img.jpg\) Latest Test Result \(no tests\)](#)

### Analysis results

-  Checkstyle Warnings: [44](#)
-  FindBugs Warnings: 0
-  PMD Warnings: 0
-  Open Tasks: -

[!\[\]\(3b9ccecb1a8c6aa1a3fb9da28594c368\_img.jpg\) Latest Aggregated Test Result \(no tests\)](#)

### Permalinks

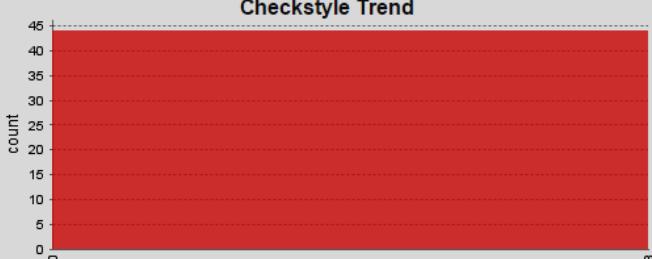
- [Last build \(#13\), 1 min 4 sec ago](#)
- [Last stable build \(#13\), 1 min 4 sec ago](#)
- [Last successful build \(#13\), 1 min 4 sec ago](#)
- [Last failed build \(#12\), 2 min 23 sec ago](#)
- [Last unsuccessful build \(#12\), 2 min 23 sec ago](#)

**FindBugs Trend**



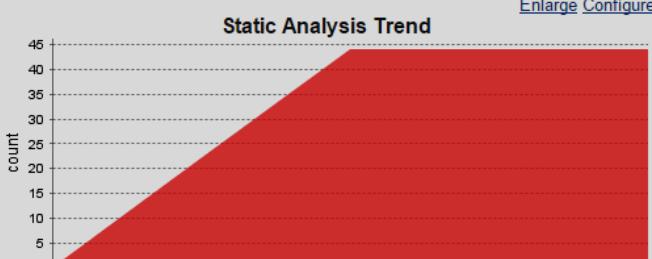
Disable Project

**Checkstyle Trend**



Enlarge Configure

**Static Analysis Trend**



Enlarge Configure

- Code Coverage
- Static Code Analysis
- **Performance Reporting**
- Other Useful Plugins
- Lab 3 – Build a Jenkins job to generate code quality reports

- **Automated Performance Testing** is a quick way to detect any performance issues and verify the performance against the Service Level Agreement (SLA)
- Jenkins is very useful to automate the performance testing process
  - Run the load test with JMeter or Parasoft as part of Jenkins Job
  - Generate the reports using Performance plugin
- JMeter simulates load on your application and measures the response time as the number of simulated users and requests increase
- **The Performance plugin** captures reports from JMeter and generates the trend report of performance and robustness

# Integrating JMeter in build.gradle

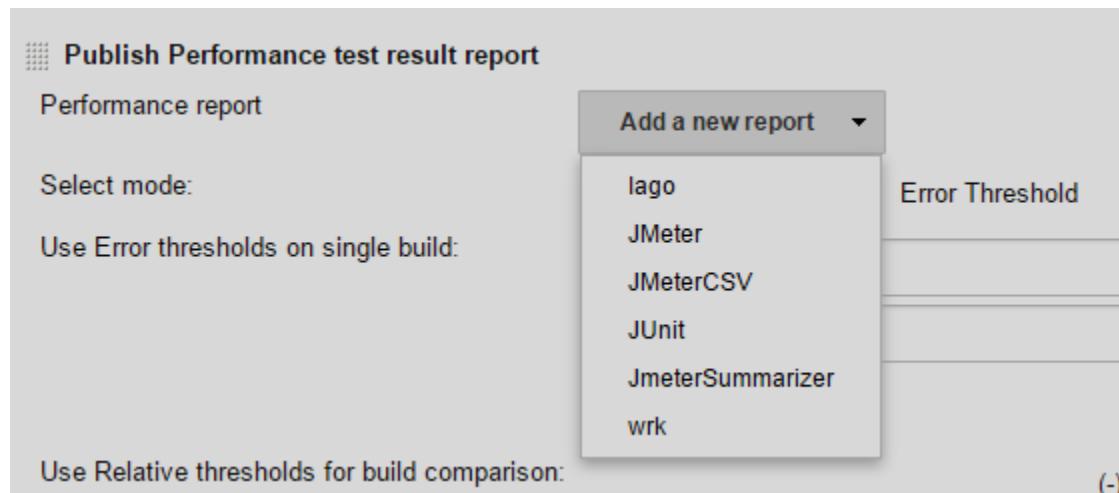
- The Gradle JMeter plugin is a quick way to integrate JMeter into the build.gradle

```
apply plugin: 'jmeter'  
buildscript {  
    dependencies {  
        classpath "com.github.kulya:jmeter-gradle-  
            plugin:1.3.1-2.6"  
    }  
}
```

- Copy your JMeter tests (.jmx) files to your <JavaProject>/src/test/jmeter folder
- Run gradle jmeterRun task in your Jenkins job to run your JMeter tests

# Running reports with Performance Plugin

- After installing Performance Plugin, click on “[Publish test result report](#)” as part of post build action items in your Jenkins job
- Select JMeter from performance report drop down and specify the path to performance report files



- Code Coverage
- Static Code Analysis
- Performance Reporting
- **Other Useful Plugins**
- Lab 3 – Build a Jenkins job to generate code quality reports

# Other Useful Plugins

- [Changes Plugin](#) is used to generate a changelog from all previous builds to the last successful one
- [Job Configuration History Plugin](#) keep track of config changes in each build job including who did it.
- [Email Extension Plugin](#) extends Jenkins built-in email notification functionality
- [Build Timeout Plugin](#) deals with hung builds
- [SonarQube Plugin](#) integrates with Sonar to generate all-in-one quality report on Sonar server
- [Jira Plugin](#) integrates Altassian Jira to Jenkins
- [Copy Artifacts Plugin](#) is used to copy artifacts from another project
- [Build Monitor Plugin](#) is live job status monitor
- [Workspace Cleanup Plugin](#) deletes your workspace before or after builds
- [Disk-usage Plugin](#) keeps track of your disk space usage

- Code Coverage
- Static Code Analysis
- Performance Reporting
- Other Useful Plugins
- **Lab 3 – Build a Jenkins job to generate code quality reports**

- In this lab you will build and run your Jenkins job to generate code quality reports
  - Create new Java Gradle project with Cobertura, Findbugs, Checkstyle and PMD plugins
  - Install new plugins in Jenkins
  - Configure new Freestyle Gradle job with code coverage and static code analysis plugins
  - Run the Jenkins job to generate quality reports

# SonarQube Plugin

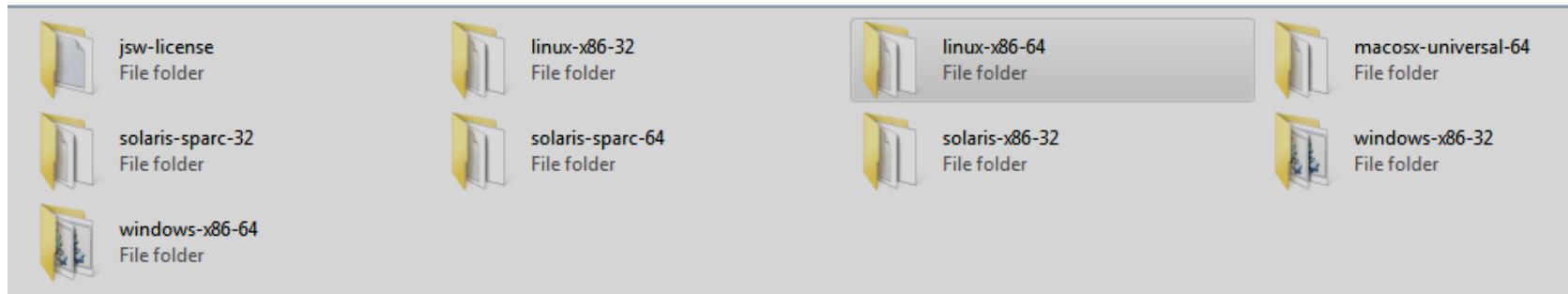
- **Install SonarQube**
- Integrate Jenkins with SonarQube
- Locate and Open Generated Report
- Review the Report's Organization
- Lab 4 – Generate first SonarQube report using Jenkins

# Introducing SonarQube

- SonarQube is free and open source **code quality platform**
- More than 50 plugins available to extend SonarQube with CI and development tools
- It is not just for Java : More than 20 programming languages supported by SonarQube
- Provides in-time code quality snapshots as well as trending of lagging and leading quality indicators
- SonarQube analysis usually is performed **over night or once a day** as it performs a full analysis on the entire code base and sends it to the server, which will process it and save the results to the SonarQube database.
- You can view different SonarQube reports for open source code here at <http://nemo.sonarqube.org/>

# Download and Install SonarQube

- SonarQube is pretty easy to install and configure
- Go to <http://www.sonarqube.org/downloads/> and download the latest SonarQube version as zip file
  - Version 5.3 is the latest stable version
- Then extract the zip files to the appropriate folder based on your operating system
- After extracting, go to the `bin` folder under SonarQube, you will see the folder for each operating system to start the SonarQube



- Only one configuration step is necessary for SonarQube
  - To provide the database details in [sonar.properties](#) file
- The [sonar.properties](#) file is available under conf folder
- Supported databases are MySQL, Oracle, PostgreSQL and Microsoft SQLServer
- The [embedded H2 database](#) is the default database used by SonarQube
  - We will use the default embedded database for this course
  - However, the embedded database is not recommended for production use

# Running SonarQube

- SonarQube installation comes with all the startup scripts for major operating systems
- The operating system specific files are available under `bin` folder under the SonarQube Installation folder
  - Such as `c:\sonarqube\sonarqube-5.3\bin\windows-x86-64`
- Run either `sonar.sh` or `startSonar.bat` to start the SonarQube
- Open your browser with <http://localhost:9000>. You will be able to see the SonarQube dashboard.

The screenshot shows the SonarQube dashboard at `localhost:9000`. The top navigation bar includes links for Dashboards, Issues, Measures, Rules, Quality Profiles, Quality Gates, and More. The main content area displays a welcome message: "Welcome to SonarQube Dashboard. Since you are able to read this, it means that you have successfully started your SonarQube server. Well done! If you have not removed this text, it also means that you have not yet played much with SonarQube. So here are a few pointers for your next step: » Do you now want to run analysis on a project? » Maybe start customizing dashboards? » Or simply browse the complete documentation? » If you have a question or an issue, please visit the Get Support page." To the right, there are two sections: "PROJECTS" and "QG NAME". The "PROJECTS" section shows "No data". The "QG NAME" section shows "No data".

- Install SonarQube
- **Integrate Jenkins with SonarQube**
- Locate and Open Generated Report
- Review the Report's Organization
- Lab 4 – Generate first SonarQube report using Jenkins

# Integrate SonarQube with Jenkins

- Integrating SonarQube with Jenkins procedure contains the following four tasks
  1. Install SonarQube Jenkins plugin
  2. Configure SonarQube installation in Jenkins
  3. Configure SonarQube Scanner/Runner in Jenkins
  4. Enabling SonarQube analysis in a build job
- Let's review each of the steps in next few slides

# Install SonarQube Jenkins Plugin

- Install SonarQube plugin using Manage Plugins link
- Click on Manage Jenkins button on Jenkins vertical navigation
- Then click on Manage Plugins link
- Next click the Available tab and search for SonarQube
- Select the SonarQube plugin and click on Install without restart button
- After the installation, you will able to see SonarQube plugin under Installed tab

The screenshot shows the Jenkins Manage Plugins interface. At the top, there are tabs for 'Updates', 'Available' (which is selected), 'Installed', and 'Advanced'. A search bar at the top right contains the text 'SonarQube'. Below the tabs, there is a table with two rows. The first row is for the 'SonarQube Plugin', which has a checked checkbox and a detailed description: 'This plugin allow easy integration of [SonarQube](#)™, the open source platform for Continuous Inspection of code quality.' The second row is for the 'Mashup Portlets' plugin, which has an unchecked checkbox and a description: 'Additional portlets: 'Generic JS Portlet' flexibly displays any content from another source. 'Recent Changes' and 'Test Results' show useful job information that would otherwise only available by drilling down into the job. The 'SonarQube Portlet' shows important issues from Sonar directly in Jenkins.' At the bottom of the table, there are three buttons: 'Install without restart' (highlighted in blue), 'Download now and install after restart', and 'Check now'. A status message 'Update information obtained: 19 hr ago' is also present.

# Configure SonarQube installation in Jenkins

- Go to Manage Jenkins and choose Configure System
- You will find SonarQube section at the bottom of the page
- Then click on Add SonarQube button to add SonarQube local installation
  - Just need SonarQube server URL

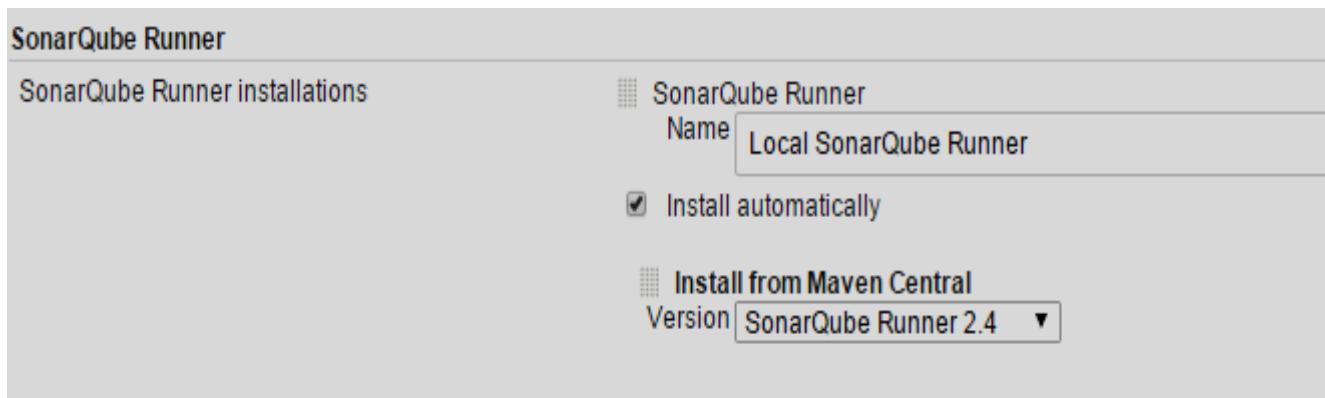
SonarQube

Environment variables	<input type="checkbox"/> Enable injection of SonarQube server configuration as build environment variable. If checked, jobs administrators will be able to inject a SonarQube server configuration as environment variable.										
SonarQube installations	<table border="1"><tr><td>Name</td><td>Local SonarQube</td></tr><tr><td>Server URL</td><td>http://localhost:9000 Default is http://localhost:9000</td></tr><tr><td>SonarQube account login</td><td>SonarQube account used to perform analysis. Mandatory.</td></tr><tr><td>SonarQube account password</td><td>SonarQube account used to perform analysis. Mandatory.</td></tr><tr><td>Disable</td><td><input type="checkbox"/> Check to quickly disable SonarQube on all jobs.</td></tr></table>	Name	Local SonarQube	Server URL	http://localhost:9000 Default is http://localhost:9000	SonarQube account login	SonarQube account used to perform analysis. Mandatory.	SonarQube account password	SonarQube account used to perform analysis. Mandatory.	Disable	<input type="checkbox"/> Check to quickly disable SonarQube on all jobs.
Name	Local SonarQube										
Server URL	http://localhost:9000 Default is http://localhost:9000										
SonarQube account login	SonarQube account used to perform analysis. Mandatory.										
SonarQube account password	SonarQube account used to perform analysis. Mandatory.										
Disable	<input type="checkbox"/> Check to quickly disable SonarQube on all jobs.										

- SonarQube Scanner is the preferred option to analyze a project with SonarQube for non-Maven based projects
- Need simple configuration file, **sonar-project.properties** in the root directory of the project (that is being analyzed) with following properties
  - sonar.projectKey (unique name in SonarQube instance)
  - sonar.projectName (project name displayed in SonarQube UI)
  - sonar.projectVersion (any unique project version, such as 1.0)
  - sonar.sources (relative path for source code, such as src)

# Configure SonarQube Runner in Jenkins

- Go to Manage Jenkins and choose Configure System
- Scroll down to the SonarQube Runner configuration section and click on Add SonarQube Runner
- Just name the runner and install it automatically

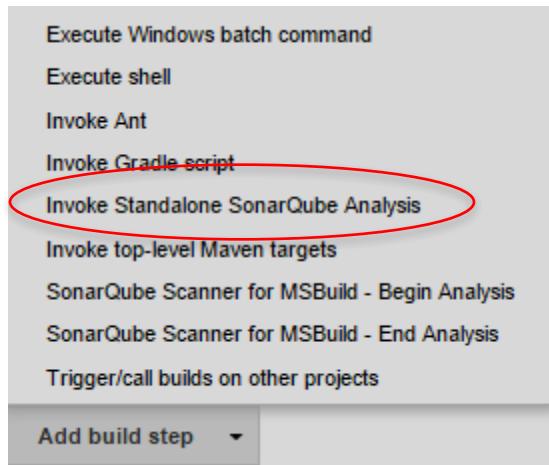


# Enabling SonarQube analysis in a build job

- SonarSource provides different scanners to analyze your source code
  - SonarQube Scanner for Jenkins
  - SonarQube Scanner for Maven
  - SonarQube Scanner for Gradle
  - SonarQube Scanner for Ant
  - Details documentation on each of the Scanners -  
<http://docs.sonarqube.org/display/SONAR/Analyzing+Source+Code>
- The **SonarQube Scanner for Jenkins** provides two ways to enable SonarQube analysis in a Jenkins build job
  - Build step to trigger the SonarQube analysis with the SonarQube Scanner for non-Maven projects
  - Add Post-build action to trigger the SonarQube for Maven based projects

# Build step to trigger the SonarQube analysis

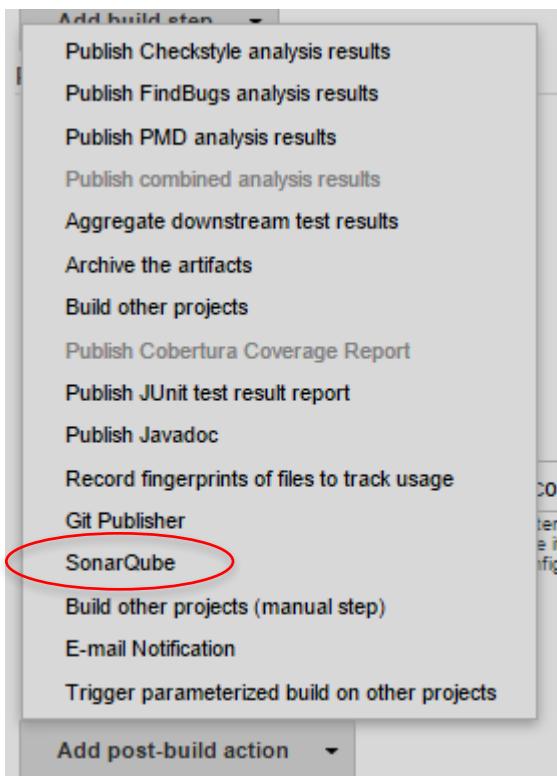
- Create any Jenkins job or update existing job to select Invoke Standalone SonarQube Analysis from Build step



- Pretty much nothing is required in SonarQube Analysis section if you are using default configuration for jdk, sonar-project.properties etc.

# Post-build action to trigger the SonarQube

- On a new or existing Maven job, go to the Post-build Actions section and click on Add post-build action
- If you select this option for non Maven jobs then your build job will fail



# Gradle SonarQube Plugin

- The SonarQube Gradle plugin is the latest plugin released by SonarSource team for Gradle projects
- It provides the ability to execute the SonarQube analysis via a regular Gradle task
  - Execute `gradle sonarqube` to generate sonar report
- This is what you need to provide in build.gradle

```
apply plugin: 'org.sonarqube'  
sonarqube {  
    properties {  
        property "sonar.projectName", "My Project"  
        property "sonar.projectKey", "My:Project"  
    }  
}
```

- Install SonarQube
- Integrate Jenkins with SonarQube
- **Locate and Open Generated Report**
- Review the Report's Organization
- Lab 4 – Generate first SonarQube report using Jenkins

# Locate and Open Generated Report

- Let's add new Build step to Invoke Standalone SonarQube Analysis in a Jenkins job configuration
- Then Run the build job
- You can verify the SonarQube execution steps in the console and the console has sonar URL to browse the SonarQube report

```
17:51:11.373 INFO - Analysis reports compressed in 78ms, zip size=9 KB
17:51:16.674 INFO - Analysis report uploaded in 5301ms
17:51:16.675 INFO - ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard/index/Jenkins:Lab
17:51:16.675 INFO - Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
17:51:16.675 INFO - More about the report processing at http://localhost:9000/api/ce/task?id=AVJW72iu410IqmoYX02V
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
Total time: 35.637s
Final Memory: 17M/234M
INFO: -----
```

# Locate and Open Generated Report - Continued

- Refresh the dashboard of SonarQube and you will be able to view the recently generated report in SonarQube
- Now click on the project to get more details on code quality metrics

The screenshot shows the SonarQube dashboard with a table titled 'PROJECTS'. The table has columns: QG, NAME, VERSION, LOC, TECHNICAL DEBT, and LAST ANALYSIS. There is one result: 'Jenkins Lab project' (VERSION 1.0, LOC 107, TECHNICAL DEBT 1h 27min, LAST ANALYSIS 17:51). Below the table, there is a legend: 'Size: Lines of code' and 'Color: Coverage'. A dark bar is at the bottom.

QG	NAME	VERSION	LOC	TECHNICAL DEBT	LAST ANALYSIS
✓	Jenkins Lab project	1.0	107	1h 27min	17:51

1 results

PROJECTS

Size: Lines of code Color: Coverage

- Install SonarQube
- Integrate Jenkins with SonarQube
- Locate and Open Generated Report
- **Review the Report's Organization**
- Lab 4 – Generate first SonarQube report using Jenkins

# Review the Report's Organization

- The complete Sonar report is organized in the few sections and widgets to cover the **Seven Axes of Quality**
  - **Potential Bugs** and **Coding Rules** : Are covered under Issues section
  - **Test** : Are covered under code coverage section
  - **Comments** and **Duplications** : Are covered under Duplications section
  - **Architecture / Design** and **Complexity** : Are covered under Structure section
- Let's review all different sections using SonarQube's own project code
  - <https://nemo.sonarqube.org/> is the online instance of SonarQube dedicated to many open source projects

# Sonar Report - Project Home Page

The screenshot shows the SonarQube Project Home Page with several key metrics displayed:

- Quality Gate:** Warning
- Technical Debt:** A (Green), 42d, Debt, started 4 years ago.
- Coverage:** 87.9% (Coverage), 8.8k (Tests).
- Duplications:** 1.6% (Duplications), 187 (Duplicated Blocks).
- Structure:** Java 80.0%, JavaScript 11.0%, Unknown 8.9%.
- Lines of Code:** 186k, +18 (Lines of Code).
- Leak Period:** since 5.3-SNAPSHOT, started 2 days ago.
- New Debt:** 20min, 1 New Issues.

Annotations with red arrows and text:

- Key metric is graphed over time in the leak period (points to the Technical Debt section).
- Cumulative and leak period values are highlighted (points to the Coverage and Duplications sections).
- Detailed views are available in each domain (points to the Structure section).
- Click through for metric drilldowns (points to the Lines of Code section).

# Sonar Report - Technical Debt and Issues

SonarQube      December 11, 2015 6:40 AM Version 5.4-SNAPSHOT

**Technical Debt**      Leak Period: since 5.3-SNAPSHOT started 2 days ago

**Files**      Size: Blocker issues & Critical issues

**Technical Debt**

**A**

<b>Technical Debt</b> <b>42d</b>	<b>Technical Debt Ratio</b> <b>0.4%</b>	<b>Issues</b> <b>1,710</b>
Added <b>20min</b>	On New Code <b>0.5%</b>	Added <b>1</b>
Removed <b>0</b>	Removed <b>0</b>	Removed <b>0</b>

<b>Blocker</b> <b>0</b>	<b>Critical</b> <b>0</b>	<b>Major</b> <b>894</b>	<b>Minor</b> <b>182</b>	<b>Info</b> <b>634</b>
Added <b>0</b>	Added <b>0</b>	Added <b>1</b>	Added <b>0</b>	Added <b>0</b>
Removed <b>0</b>	Removed <b>0</b>	Removed <b>0</b>	Removed <b>0</b>	Removed <b>0</b>

**cert**   **security**   **cwe**

**performance**   **obsolete**

**bad-practice**

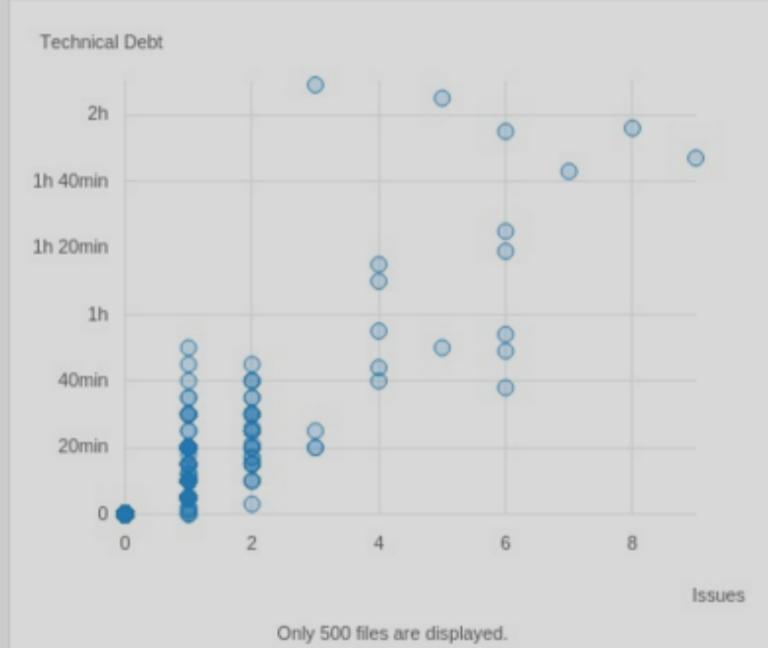
**brain-overload**   **pitfall**   **misra**

**convention**

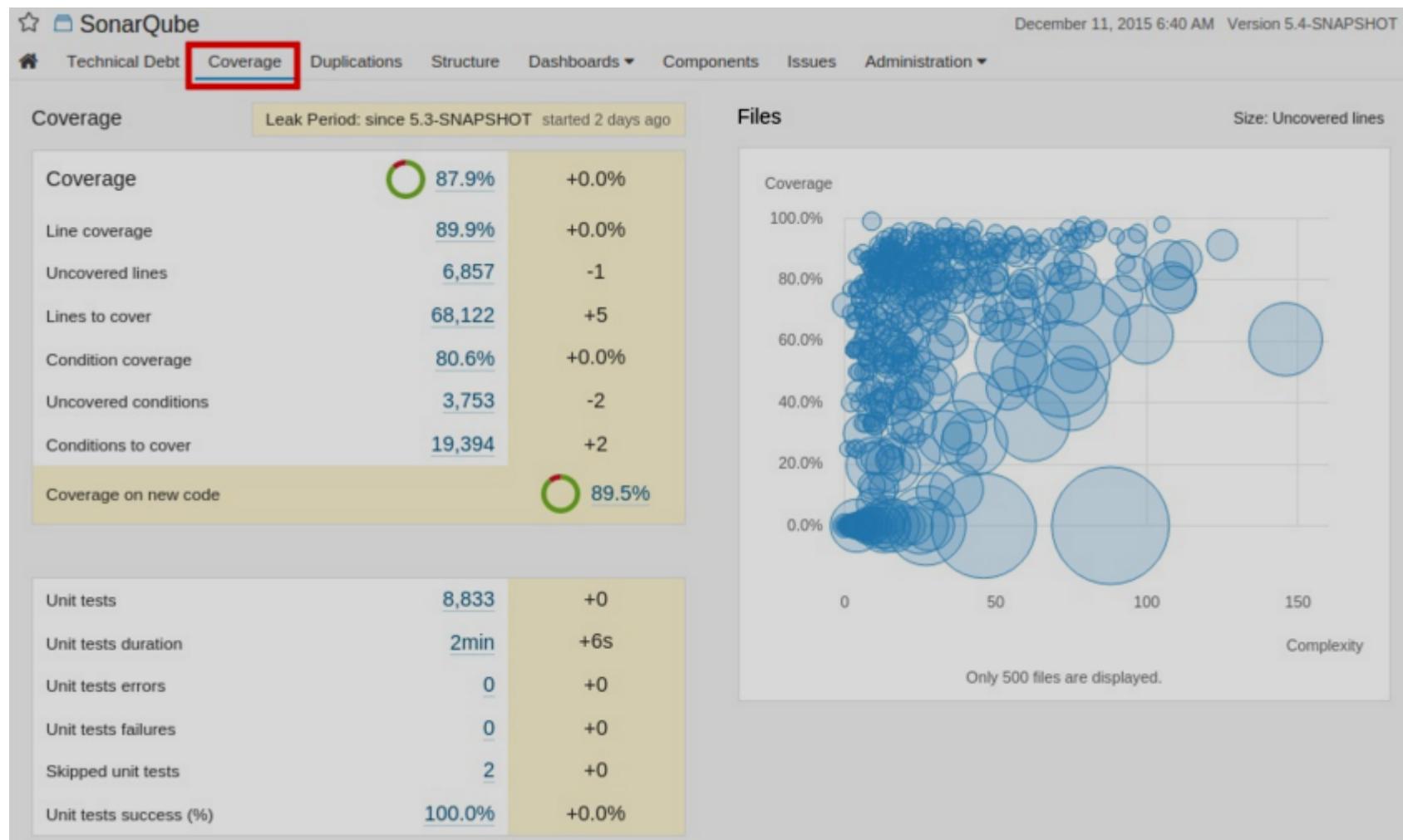
**Unmanaged Issues**

<b>Not assigned</b>	<b>1</b>
Simon Brandhof	32
Sebastien Lesaint	3
Duarte Meneses	1
Julien Henry	1
Teryk Bellahsene	1

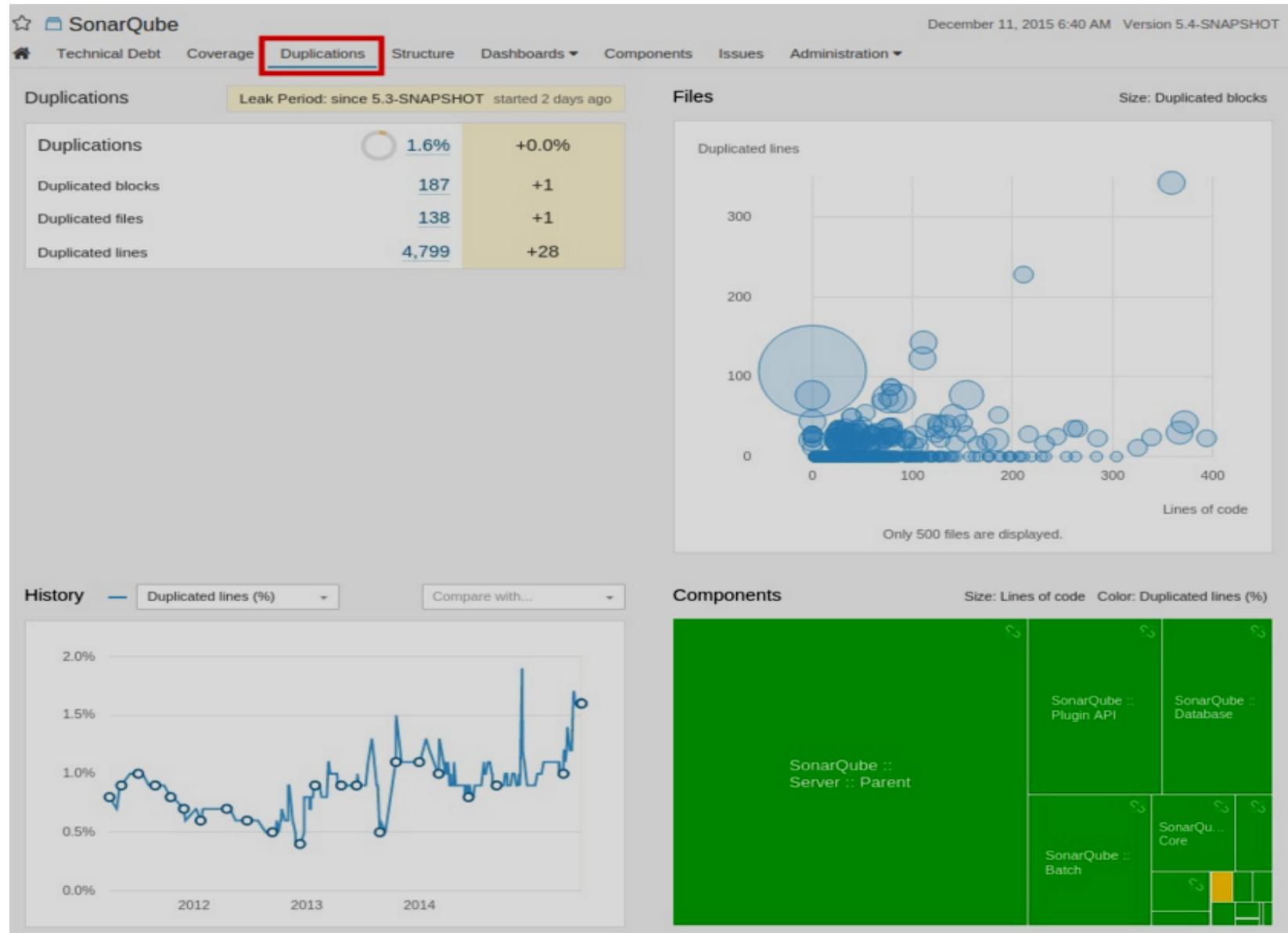
Only 500 files are displayed.



# Sonar Report – Code Coverage



# Sonar Report - Duplications



# Sonar Report – Structure and Complexity

SonarQube      December 11, 2015 6:40 AM Version 5.4-SNAPSHOT

Technical Debt   Coverage   Duplications   **Structure**   Dashboards   Components   Issues   Administration

Leak Period: since 5.3-SNAPSHOT started 2 days ago

### Structure

Lines of code	185,586	+18
Java	80.0%	
JavaScript	11.0%	
Unknown	8.9%	
Classes	3,112	+0
Directories	744	+0
Files	3,458	+0
Functions	19,539	+1
Lines	307,317	+40
Statements	59,046	+4

Complexity	32,290	+3
Complexity /function	1.8	+0.0
5,556	1,428 559 215 71 30 32	1 2 4 6 8 10 12
Complexity /file	10.6	+0.0
1,444	528 586 228 186 47 21	0 5 10 20 30 60 90
Complexity /class	8.9	+0.0

Comment lines	10,934	+1
Comments (%)	5.6%	+0.0%
Public API	11,344	-1
Public documented API (%)	25.4%	+0.0%
Public undocumented API	8,463	-1

### History

Lines of code   Compare with...

### Components

Size: Lines of code

SonarQube :: Server :: Parent	110k
SonarQube :: Plugin API	24k
SonarQube :: Database	20k
SonarQube :: Batch	17k
SonarQube :: Core	6.6k
SonarQube :: Duplications	2.8k
SonarQube :: Web Service	2.3k
SonarQube :: Code Colorizer	844
SonarQube :: Home	658
SonarQube :: Testing Harness	622
SonarQube :: Application	600
SonarQube :: Batch :: Protocol	598
SonarQube :: Markdown	387
SonarQube :: Plugin API Dependencies	176
SonarQube :: Check API	96

TCT

- Install SonarQube
- Integrate Jenkins with SonarQube
- Locate and Open Generated Report
- Review the Report's Organization
- **Lab 4 – Generate first SonarQube report**

# Lab 4 – Generate first SonarQube report

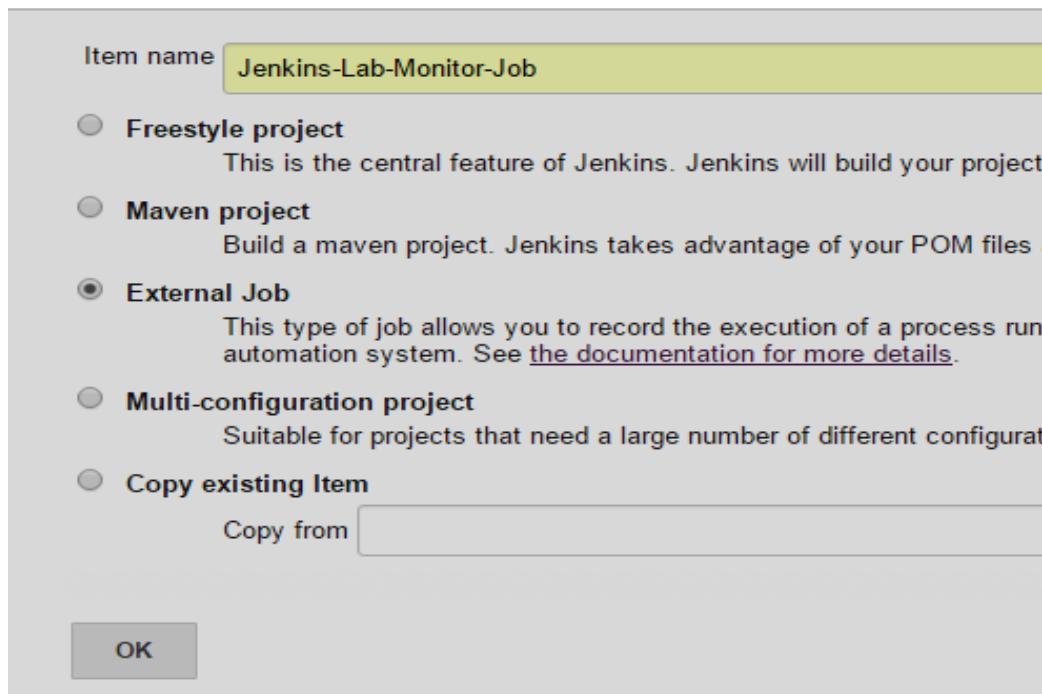
- In this lab, you will install and configure SonarQube and run SonarQube analysis with your Java project
  - Download and Configure SonarQube
  - Start SonarQube Server
  - Install and Configure SonarQube Jenkins plugin
  - Configure SonarQube Runner in Jenkins
  - Enable SonarQube analysis in your Jenkins job
  - Run Jenkins job to generate SonarQube report

# Advanced Jenkins

- **Monitor External Jobs**
  - Matrix or Multi-Configuration Jobs
  - Distributed Builds
  - Concept of a Pipeline
  - Splitting a Big Job into Smaller Jobs
  - File Fingerprint Tracking
  - Using Jenkins for non-Java Projects
  - Lab 5: Configure distributed builds using Master/Slave configuration
  - Lab 6: Build your first pipeline

# Monitor External Jobs

- Jenkins provides a job to **monitor non-interactive execution of processes** such as cron or batch jobs
  - You can monitor local as well as remote processes
  - Your external process sends output to a Jenkins job
- Create a new job and choose **external job** as job type



# Monitor External Jobs - Configuration

- Only two fields are needed to configure this type of job
  - Name of External Monitoring Job
  - Meaningful description

The screenshot shows the configuration page for an external monitoring job in Jenkins. The job name is "Jenkins-Lab-Monitor-Job" and the description is "Job for monitoring windows batch script". There is an unchecked checkbox for "Discard Old Builds". At the bottom are "Save" and "Apply" buttons.

External Job name	Jenkins-Lab-Monitor-Job
Description	Job for monitoring windows batch script
<a href="#">[Plain text]</a> <a href="#">Preview</a>	
<input type="checkbox"/> Discard Old Builds	
<a href="#">Save</a>	<a href="#">Apply</a>

# Requirements for External Process

- Extract and Copy the following jars from Jenkins.war (WEB-INF/lib) to the folder where you are executing your external script

jenkins-core-\* .jar  
remoting-\* .jar  
ant-\* .jar  
commons-io-\* .jar  
commons-lang-\* .jar  
jna-posix-\* .jar  
xstream-\* .jar

- JENKINS\_HOME needs to be set to locate the Jenkins server
  - Include username:password@jenkinsUrl if the authentication is required

# How to Invoke Jenkins External Job

- In Windows environment, you can create a .cmd file with the following lines
  - Jenkins-Lab-Monitor-Job is the name of Jenkins external monitoring job
  - jenkinsLab is a simple batch script that just lists directory content
  - Also notice the userId and password as part of Jenkins URL

```
set
JENKINS_HOME=http://admin:admin@localhost:8080/

java -jar jenkins-core-1.631.jar Jenkins-Lab-
Monitor-Job cmd.exe /c jenkinsLab.bat
```

# Output in Jenkins Dashboard

- Once your external process runs, the outcome of the process will be sent to the Jenkins job

Jenkins > Jenkins-Lab-Monitor-Job > #1

Back to Job

Console Output

Configure

## Console Output

```
Started
Running as Administrator

c:\JenkinsCourse\Jenkins-Course-Lab3>echo Hello! This a sample batch file for Jenkins Monitor Job
Hello! This a sample batch file for Jenkins Monitor Job

c:\JenkinsCourse\Jenkins-Course-Lab3>echo Windows external batch process started
Windows external batch process started

c:\JenkinsCourse\Jenkins-Course-Lab3>echo process lists all files in the current folder
process lists all files in the current folder

c:\JenkinsCourse\Jenkins-Course-Lab3>dir
Volume in drive C is OS
Volume Serial Number is C43F-51D2

Directory of c:\JenkinsCourse\Jenkins-Course-Lab3

10/23/2015 12:32 PM <DIR> .
10/23/2015 12:32 PM <DIR> ..
04/21/2014 07:19 PM 1,941,731 ant-1.8.4.jar
03/25/2013 09:51 PM 185,140 commons-io-2.4.jar
07/23/2011 03:38 PM 284,220 commons-lang-2.6.jar
09/27/2015 06:07 PM 9,705,178 jenkins-core-1.631.jar
10/23/2015 12:32 PM 214 jenkinsLab.bat
10/27/2014 07:34 AM 91,455 jna-posix-1.0.3-jenkins-1.jar
07/13/2015 02:39 PM 489,023 remoting-2.52.jar
10/23/2015 12:46 PM 176 runJenkinsLab.cmd
03/31/2014 06:47 AM 533,110 xstream-1.4.7-jenkins-1.jar
9 File(s) 13,230,247 bytes
2 Dir(s) 828,329,517,056 bytes free

c:\JenkinsCourse\Jenkins-Course-Lab3>echo vow... the process has completed successfully
vow... the process has completed successfully
Finished: SUCCESS
```

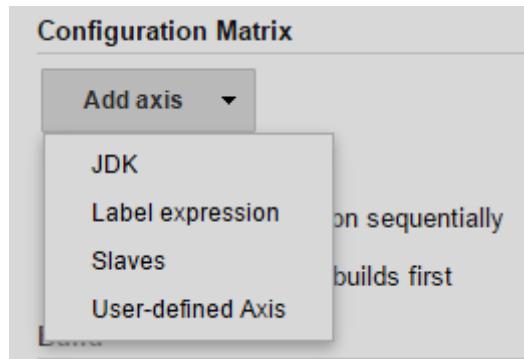
- Monitor External Jobs
- **Matrix or Multi-Configuration Jobs**
- Distributed Builds
- Concept of a Pipeline
- Splitting a Big Job into Smaller Jobs
- File Fingerprint Tracking
- Using Jenkins for non-Java Projects
- Lab 5 : Configure distributed builds using Master/Slave configuration
- Lab 6 : Build your first pipeline

# Matrix or Multi-Configuration Jobs

- Multi-Configuration build jobs are very useful for **running all possible combinations of parameters**
- Possible use cases
  - Build different versions of code for different customers
  - Build your code for different environments
  - Build your test suites in different browsers
- The Configuration Matrix allows you to specify multiple-axis graph of the type of builds to create
- Multi-configuration job is a build which spawns a new build for each configuration
- The multi-configuration build does not end until all the configurations have been built

# Multi-Configuration Job Configuration

- To create a new Matrix job, simply choose Build Multi-Configuration project on the New Job page
- The Matrix job has one important additional element
  - Configuration Matrix for defining different configurations
- You can define a different axis for JDK, Slaves and Label expression as well as provide your own custom parameters for User-defined Axis



# Multi-Configuration Matrix

- Let's configure this job to test for different JDKs and Operating Systems
  - First axis is for Master and Slave nodes
    - Master is Unix machine and Jenkins-Windows-Slave is Windows machine
  - Second Axis is for two different JDKs
    - JDK 1.7 and JDK 1.6
- The build can be configured to run in parallel or sequential mode
- The **combination filter** is another option if you need to exclude some configuration combinations from the build

Configuration Matrix

**Slaves**

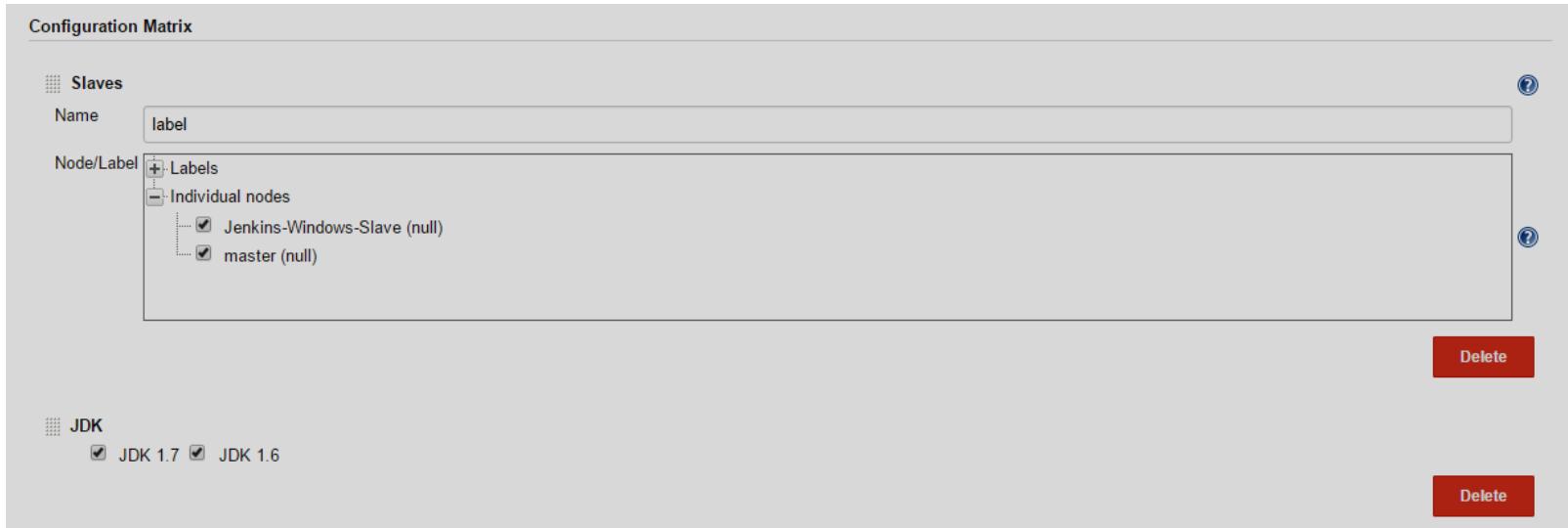
Name	label
Node/Label	+ Labels - Individual nodes - Jenkins-Windows-Slave (null) - master (null)

**JDK**

<input checked="" type="checkbox"/> JDK 1.7	<input checked="" type="checkbox"/> JDK 1.6
---	---

**Delete**

**Delete**



# Run Multi-Configuration Job

- The multi-configuration job is triggered and built like any other job
- The Jenkins-Lab-Matrix-Job runs each of the four combinations separately
- Build status and details can be viewed by clicking on each ball

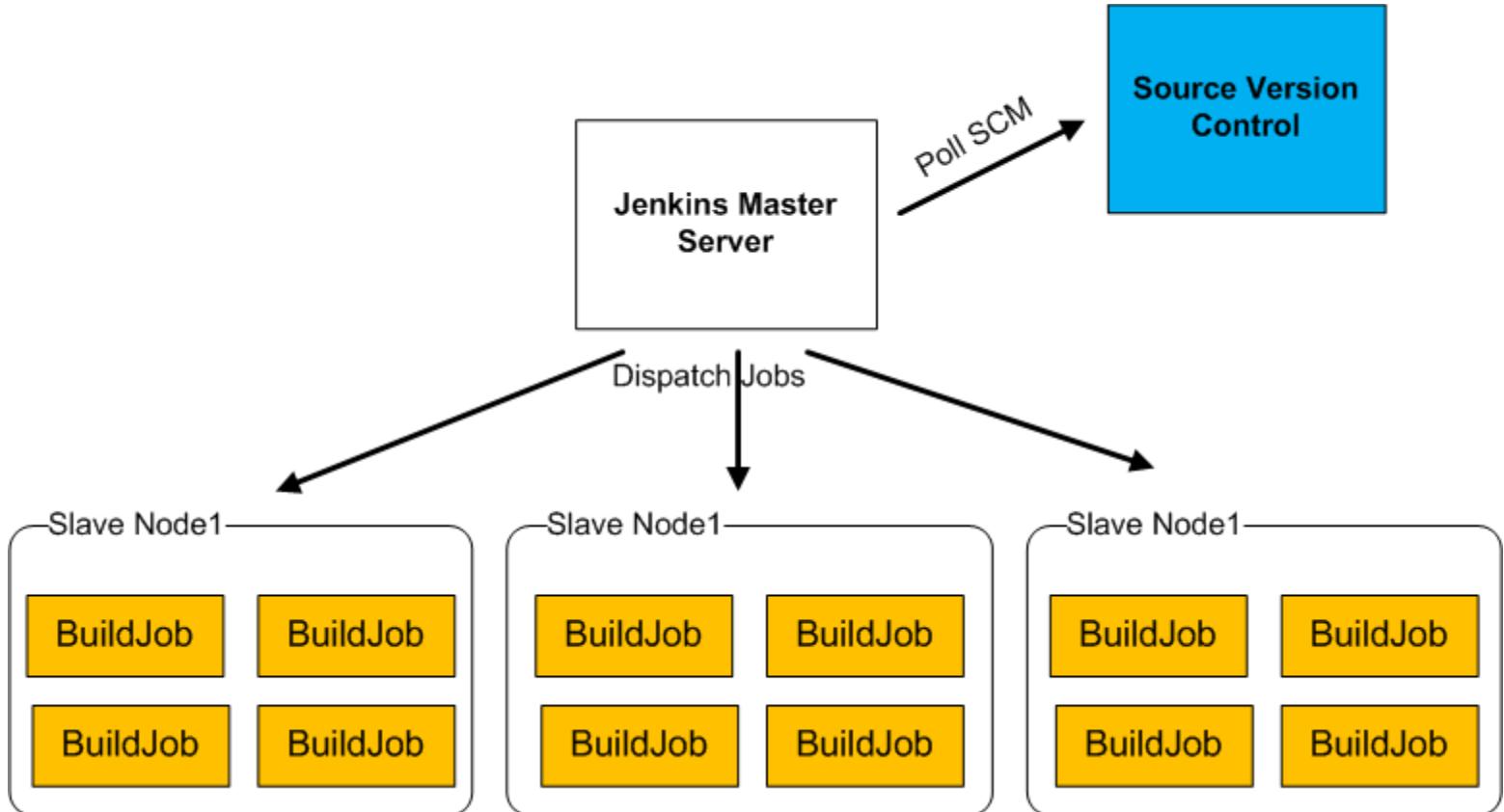
The screenshot shows the Jenkins web interface for a project named "Jenkins-Lab-Matrix-Job". The left sidebar contains links: Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Multi-configuration project, and Configure. Below these are sections for Build History and RSS feeds. The main area displays the title "Project Jenkins-Lab-Matrix-Job" and a table titled "Configuration Matrix" with three columns: Configuration Matrix, Jenkins-Windows-Slave, and master. The table has two rows: one for JDK 1.7 and one for JDK 1.6. Each row contains a blue circular icon representing a build status.

Configuration Matrix	Jenkins-Windows-Slave	master
JDK 1.7	●	●
JDK 1.6	●	●

- Monitor External Jobs
- Matrix or Multi-Configuration Jobs
- **Distributed Builds**
- Concept of a Pipeline
- Splitting a Big Job into Smaller Jobs
- File Fingerprint Tracking
- Using Jenkins for non-Java Projects
- Lab 5: Configure distributed builds using Master/Slave configuration
- Lab 6: Build your first pipeline

- Distributed Builds are the key for a scalable build architecture
- **Master and Slaves configuration** supports distributing the workload of building projects to multiple Slave nodes
  - It can scale to at least 100 remote Slaves
- We did review basics of Master and Slave in the earlier section
- Let's review additional details in this section
  - How to setup Master and Slave nodes
  - How to associate build jobs with Master and Slave nodes
  - How to monitor Master and Slaves nodes

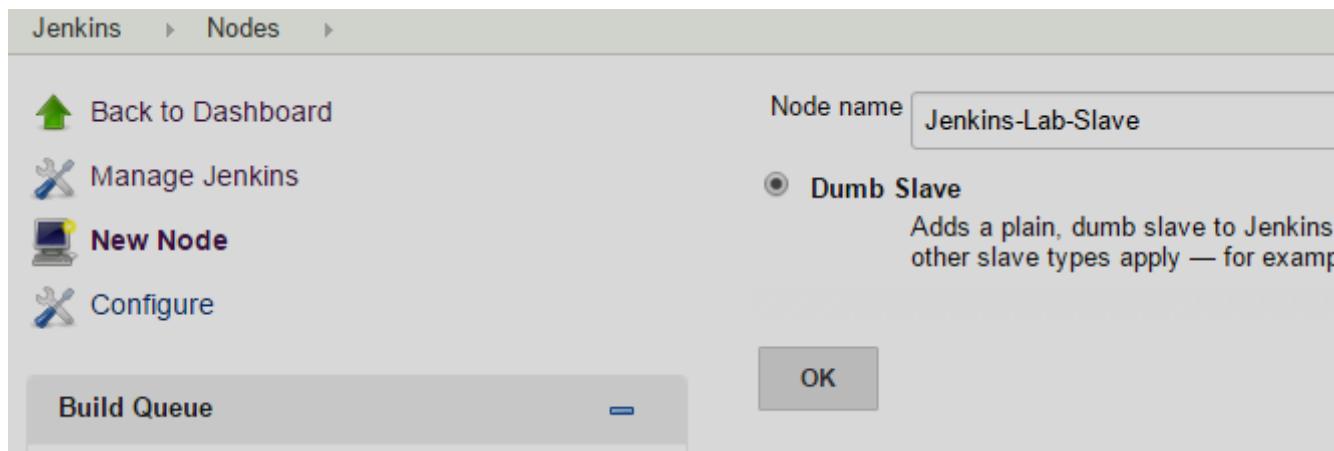
# Master/Slaves Architecture - Review



- Master is just a basic Jenkins installation and only one Jenkins server is needed in the Master/Slaves Configuration
- Master dispatches jobs to Slave nodes and at the same time, it can still build jobs it owns
  - Master distributes jobs to the slaves for the actual execution
  - Monitor the slaves by taking them offline or online as required
  - Record and present the build results

# Setup Slave Nodes

- Go to Manage Jenkins screen and click on Manage Nodes
- Create new Slave node by clicking on the New Node button
- Enter the name of your slave and select Dumb Slave type
- After you click OK button, Jenkins will provide more options for setting up Slave Nodes



# Launching Slave Nodes

- Each Slave runs the Slave agent program without installing full Jenkins on a slave
- The [executable slave.jar](#) is used in creating Slave nodes
- Various options for creating slave agents based on OS
  - Launch Slave agents via SSH
  - Launch Slave agents via Java Web Start
  - Launch Slave agents by writing your own script
  - Launch Slave agents in Headless Mode
- [SSH is the common and preferred method](#) for creating slaves on a Unix machine
  - Only need SSH and host name!!!
- Java Web Start can be used where there are connectivity issues between Master and Slaves nodes due to firewall

# Launch Slave via Java Web Start – Master Node

Two Step Process – First you create new node on the Master machine and then launch JNLP file on the Slave machine directly from browser

- Let's create a Slave node by clicking New Node button
- Select Launch slave agents via JNLP in Launch method field
- Provide remote directory dedicated for a Jenkins slave machine
- Labels field helps to keep slaves together
  - You can use OS and/or geographic location to configure labels such as Windows-Slave, Redhat-Slaves etc
- Finally, click on Save button to save new node

Name	Jenkins-Windows-Slave
Description	Jenkins Slave for Windows Server 1
# of executors	1
Remote root directory	c:\jenkins
Labels	Jenkins-Windows-Slave
Usage	Utilize this node as much as possible
Launch method	Launch slave agents via Java Web Start
Availability	Keep this slave on-line as much as possible
<b>Node Properties</b>	
<input type="checkbox"/> Environment variables	
<input type="checkbox"/> Tool Locations	
<b>Save</b>	

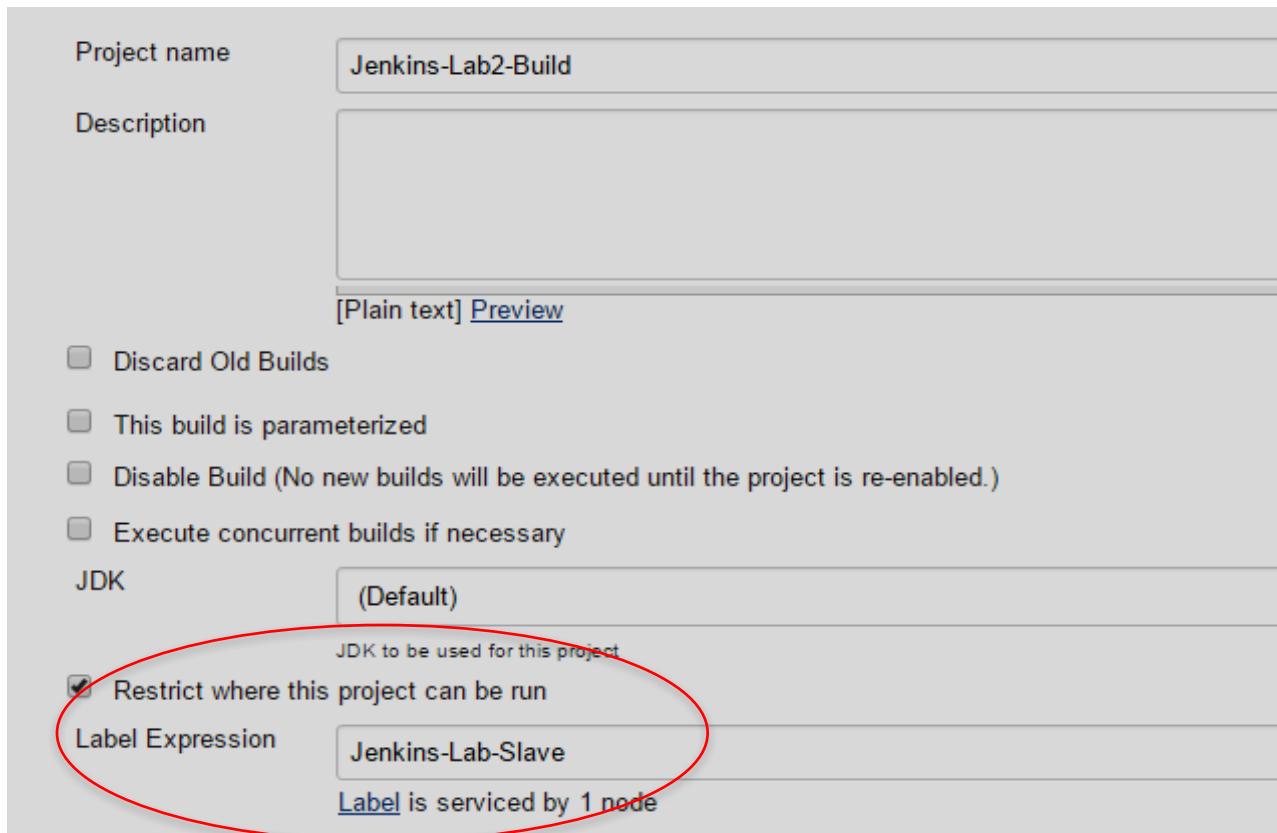
# Launch Slave via Java Web Start – Slave Node

- Now log into the Slave machine
- Open slave node screen in your browser using a URL similar to this
  - http:// [MasterJenkinsServerPath] / [Slave – Job – Name] /
- You will see JNLP launch button for launching a Slave agent on a Slave machine directly from the browser
- After clicking on Launch button, the Slave agent will be connected to the Master node and you will able to see the new Slave agent in Master Jenkins Manage Nodes screen
- A Slave agent can be installed as a Windows service

The screenshot shows the Jenkins web interface for managing a slave node. The URL in the browser is `localhost:8080/computer/Jenkins-Lab-Slave/`. The page title is "Slave Jenkins-Lab-Slave (Jenkins Slave node for Windows Server)". On the left, there's a sidebar with links: Back to List, Status, Delete Slave, Configure, Build History, Load Statistics, and Log. The main content area displays the slave node details. It includes a "Connect slave to Jenkins one of these ways:" section with a "Launch" button (which is highlighted in orange) and a "Run from slave command line:" section with a command-line script. Below that, it says "Created by Administrator". At the bottom, it lists "Projects tied to Jenkins-Lab-Slave" and indicates "None".

# Associate build jobs with Master and Slave nodes

- Once the Slave nodes are configured, you can configure which jobs will run on which node
- Open Job configuration and select Restrict where this project can be run checkbox



# Monitor Slave Nodes

- Jenkins provides monitoring of Slave nodes through Manage Nodes screen
- You can view load statistics, system information, build history and log from each of the Slave nodes
- Jenkins also monitors key **health metrics of slaves**
  - Response time
  - Low disk space and insufficient swap
  - Clock out of sync
- Jenkins automatically takes slaves offline based on the health metrics

Back to List

Status

Delete Slave

Configure

Build History

Load Statistics

Script Console

Log

System Information

Disconnect

 Slave Jenkins-Lab-Slave

Connected via JNLP agent.

Created by [Administrator](#)

**Projects tied to Jenkins-Lab-Slave**

S	W	Name ↓
		<a href="#">Jenkins-Lab2-Build</a>

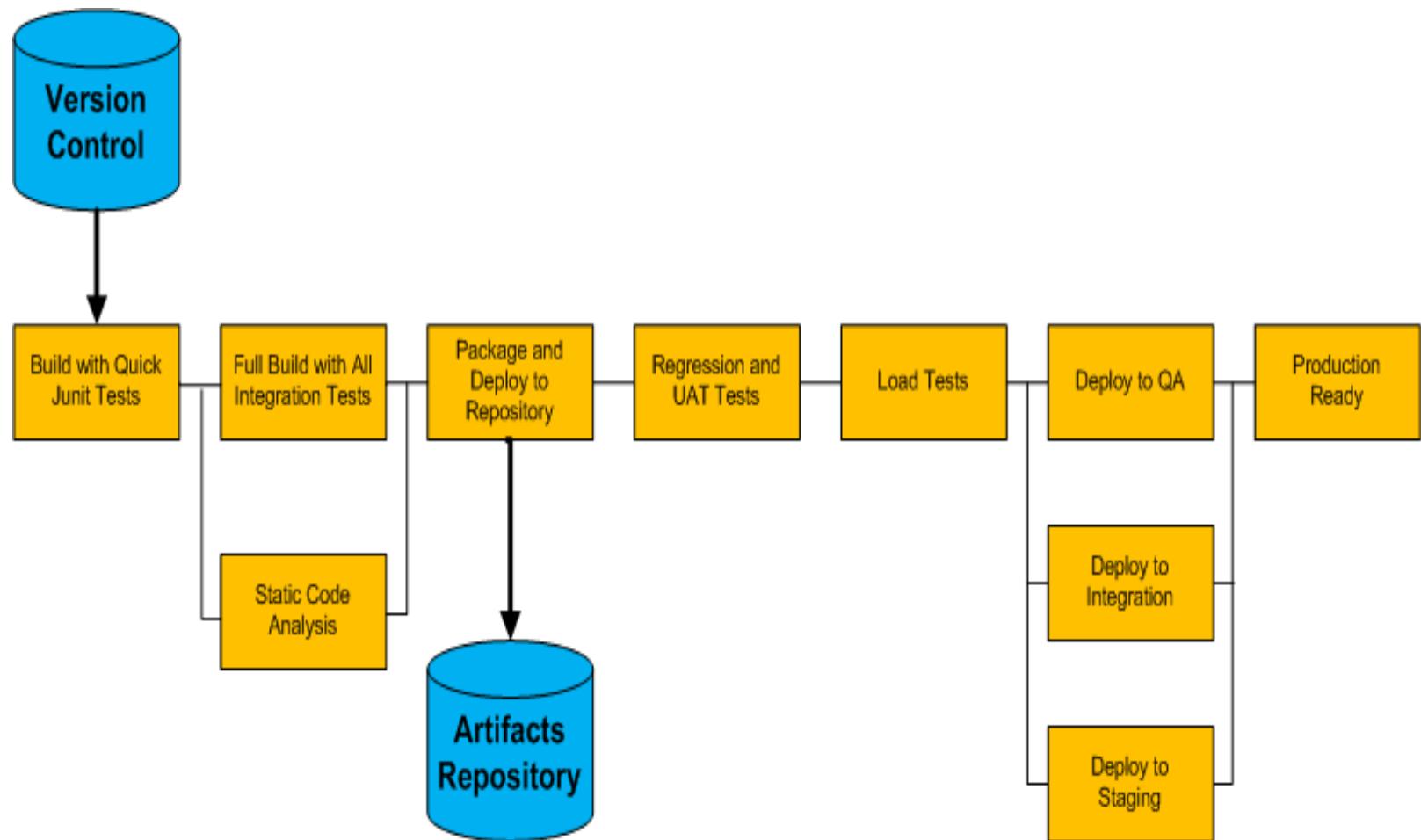
Icon: [S M L](#)

- Monitor External Jobs
- Matrix or Multi-Configuration Jobs
- Distributed Builds
- **Concept of a Pipeline**
  - Splitting a Big Job into Smaller Jobs
  - File Fingerprint Tracking
  - Using Jenkins for non-Java Projects
  - Lab 5: Configure distributed builds using Master/Slave configuration
  - Lab 6: Build your first pipeline

# Concept of Pipeline

- Automated manifestation of the process for getting your source code from version control into hands of your users implemented via the continuous integration server
- The build process is broken down into the following phases
  - Building code from Version Control
  - Running unit tests
  - Performing static code analysis
  - Packing and deploying artifacts to the repository
  - Running regression tests
  - Running performance tests
  - Deploying into different environments
- Each of the above phases can be executed in series or parallel
  - If one phase is successful, it automatically moves on to the next phase

# Build Pipelines



- Repeatable and automated process to ensure that the source code is tested, analyzed and packaged for deployment
- Only build artifacts once and deploy anywhere
- Simplified and same process to support deployments to every environment
  - Lower environments should be production-like
- Have complete visibility from code to deployment and publish matrices
- Stop the pipeline if any part of the pipeline fails

# Creating Pipeline with Jenkins

- Pipeline in Jenkins is the process of automatically starting other jobs after the execution of a job
- Jenkins has a built-in support to build other projects
- Create the following jobs to demonstrate pipeline implementation in Jenkins
  - Build\_LabProject will trigger Test\_LabProject and StaticCodeAnalysis\_LabProject
  - Test\_LabProject and StaticCodeAnalysis\_LabProject will trigger LoadTest\_LabProject
  - LoadTest\_LabProject will trigger Package\_LabProject
- In Post-build action items, you can select one or more builds that you want to trigger after this build is built

The screenshot shows the 'Post-build Actions' configuration for a Jenkins pipeline job. It includes a section for 'Build other projects' where 'Test\_LabProject, StaticCodeAnalysis\_LabProject' are selected. There are three radio button options for triggering: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. A red 'Delete' button is visible at the bottom right.

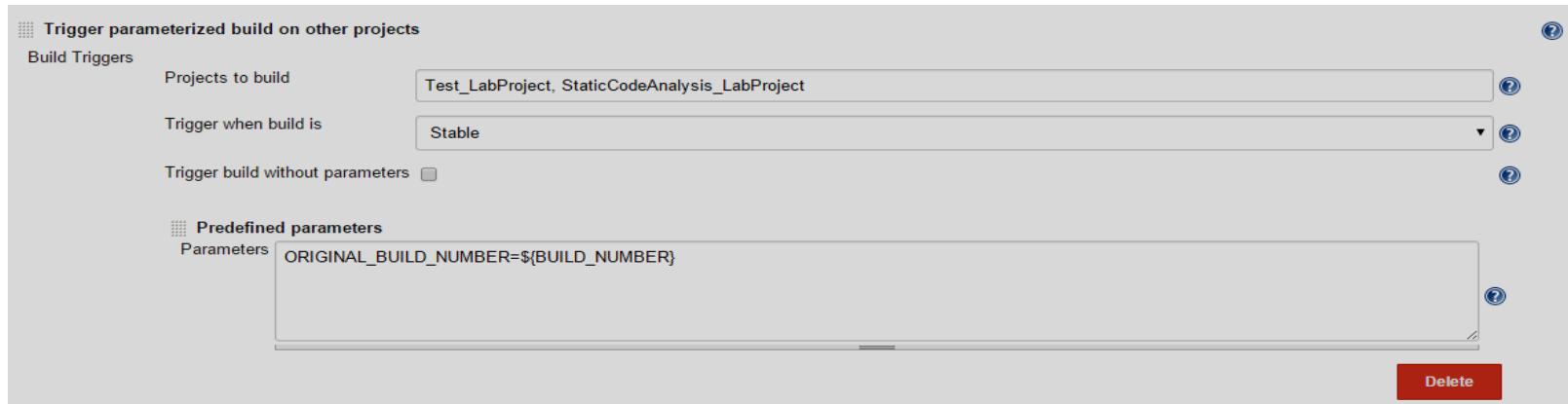
# Jenkins Build Pipeline Plugin

- Jenkins Build Pipeline Plugin provides **visualization of the build pipeline view** that shows upstream and downstream connected jobs
- Offers ability to define manual triggers for jobs
  - Useful in approval process for deploying in UAT environment

Install ↓	Name	Version
<input type="checkbox"/>	<a href="#">ontrack Jenkins plug-in</a>  This plug-in allows to connect to an <u>Ontrack</u> server in order to enable traceability and monitoring of your continuous delivery pipeline(s).	2.15.0
<input checked="" type="checkbox"/>	<a href="#">Build Pipeline Plugin</a>  This plugin provides a <u>Build Pipeline View</u> of upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins.	1.4.8
<input type="checkbox"/>	<a href="#">Fail The Build Plugin</a>  Set or change the build result to test job configurations - notifiers, publishers, promotions, build pipelines, etc.	1.0

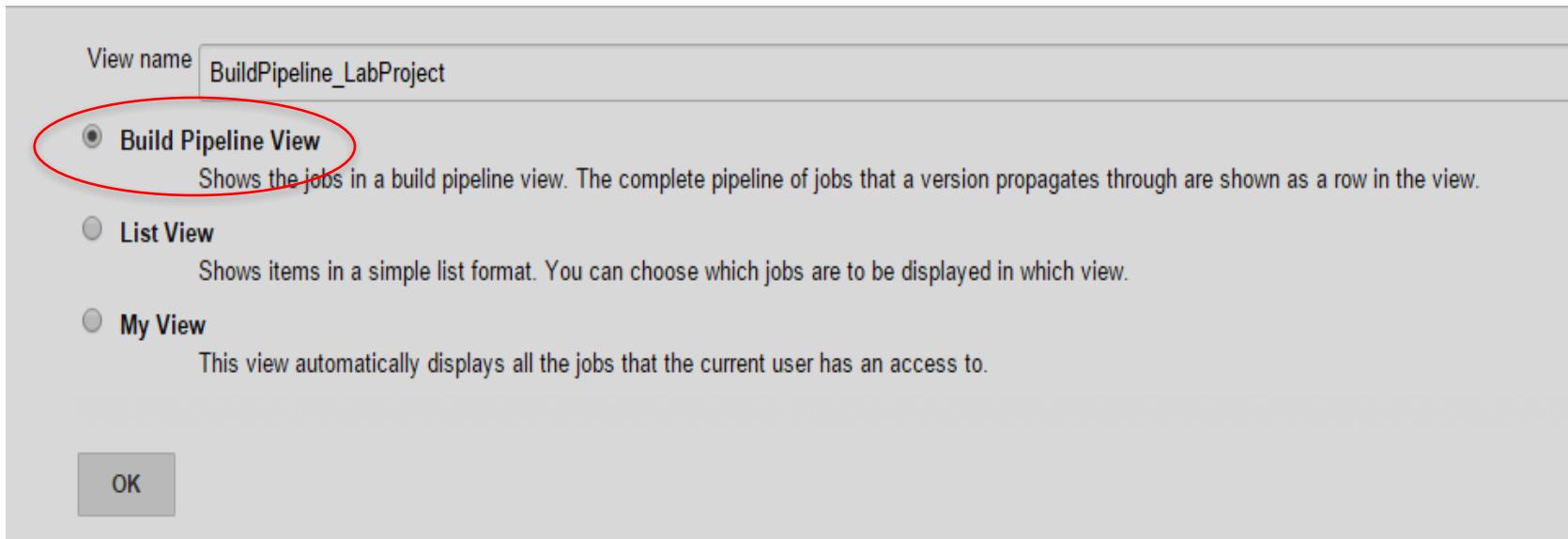
# Jenkins Parameterized Trigger Plugin

- Parameterized Trigger is another plugin you need to implement pipeline in Jenkins
- Build Pipeline plugin has dependencies on Parameterized Trigger plugin
- Parameterized Trigger plugin provides the basic triggering new build functionality and adds other useful features such as [how to specify parameters for the new build](#)



# Create New Build Pipeline View

- Go to the local Jenkins home page and click on + sign to create new view



# Configure New Build Pipeline View

- Only the required parameter is the **name of initial job**
  - Then it will traverse through the downstream jobs to build up entire pipeline

Name: BuildPipeline\_LabProject

Description: Build Pipeline from source to deployment for LabProject

[Plain text] [Preview](#)

Filter build queue:

Filter build executors:

Build Pipeline View Title: Build Pipeline Lab Project

Layout: Based on upstream/downstream relationship

This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs.

Select Initial Job: **Build\_LabProject**

No Of Displayed Builds: 3

Restrict triggers to most recent successful builds:  Yes  No

Always allow manual trigger on pipeline steps:  Yes  No

Show pipeline project headers:  Yes  No

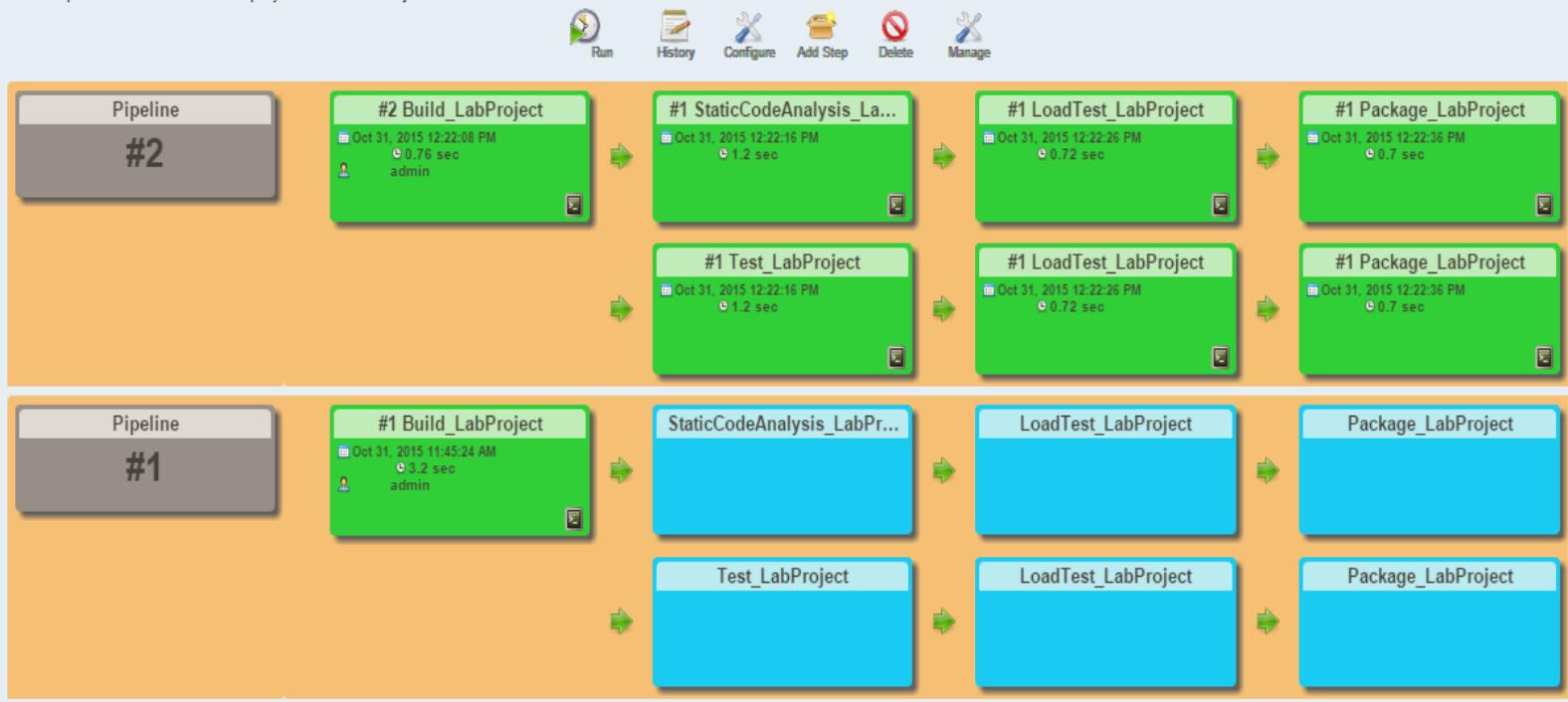
Show pipeline parameters in project headers:  Yes  No

Show pipeline parameters in revision box:  Yes  No

# New Pipeline for Lab Project

## Build Pipeline: Build Pipeline Lab Project

Build Pipeline from source to deployment for LabProject



- Monitor External Jobs
- Matrix or Multi-Configuration Jobs
- Distributed Builds
- Concept of a Pipeline
- **Splitting a Big Job into Smaller Jobs**
- File Fingerprint Tracking
- Using Jenkins for non-Java Projects
- Lab 5: Configure distributed builds using Master/Slave configuration
- Lab 6: Build your first pipeline

# Splitting a Big Job into Smaller Jobs

- The large build may take a few hours for large complicated project with many modules
- You can have many challenges with that type of long running build
  - Losing the value of continuous integration
  - No quick feedback on the committed changes
  - A lot more time wasted, especially painful with failed builds
  - Unmaintainable build configuration
- So the important practice is to split a big job into smaller jobs
  - One of the challenges is how the next jobs get artifacts from the previous jobs

- The earlier job needs to pass workspace and other job artifacts to next jobs
  - The earlier job archives all the job artifacts into a zip file at the end of the build
  - Jenkins takes fingerprint of it and triggers the next job
  - The next job extracts the zip and executes the job using those files from an earlier job
  - This process will repeat for the next job
- Jenkins offers few plugins for creating smaller jobs
  - Clone Workspace SCM Plugin
  - Build Flow Plugin
  - Build Pipeline Plugin
  - Parameterized Trigger Plugin

- **Clone Workspace Plugin** archive the workspace from builds of one project and reuse them as source for another project
- Basically this plugin allows you to divide a large task into smaller ones
  - Assume that the first job is just compiling source code
  - Then the next two jobs are reusing the workspace from the first job and running some other tasks on that
  - You not only save the time to execute the next two jobs but also reduce the total disk space by reusing the workspace

- Monitor External Jobs
- Matrix or Multi-Configuration Jobs
- Distributed Builds
- Concept of a Pipeline
- Splitting a Big Job into Smaller Jobs
- **File Fingerprint Tracking**
- Using Jenkins for non-Java Projects
- Lab 5: Configure distributed builds using Master/Slave configuration
- Lab 6: Build your first pipeline

- Fingerprint Tracking helps Jenkins to discover dependencies among your builds, and tracks your build artifacts
- A “fingerprint” is simply the MD5 checksum of a file
  - Jenkins maintains a database of md5sum which builds off projects using md5sum
- Select Record fingerprints of files to track usage from Post-Build actions in your job configuration and include the files/artifacts that you want to fingerprint

The screenshot shows the 'Post-build Actions' section of a Jenkins job configuration. It displays a single action: 'Record fingerprints of files to track usage'. Below this, there is a field labeled 'Files to fingerprint' containing the value 'target/\*.jar'. In the bottom right corner of the action row, there is a red 'Delete' button.

- Monitor External Jobs
- Matrix or Multi-Configuration Jobs
- Distributed Builds
- Concept of a Pipeline
- Splitting a Big Job into Smaller Jobs
- File Fingerprint Tracking
- **Using Jenkins for non-Java Projects**
- Lab 5: Configure distributed builds using Master/Slave configuration
- Lab 6: Build your first pipeline

# Using Jenkins for non-Java Projects

- Jenkins is written in Java but not limited to being a CI server only for Java-based systems
- Rapidly expanding to many other languages as they do not have viable CI server
- Jenkins extensible plugins architecture and REST API make it portable to many non-Java systems
  - More tools and integrations are being added by the open community to support non-Java technologies
- Jenkins is getting popular in the following:
  - .NET
  - C++
  - Python
  - Ruby
  - PHP

- Monitor External Jobs
- Matrix or Multi-Configuration Jobs
- Distributed Builds
- Concept of a Pipeline
- Splitting a Big Job into Smaller Jobs
- File Fingerprint Tracking
- Using Jenkins for non-Java Projects
- **Lab 5 : Configure distributed builds using Master/Slave configuration**
- Lab 6 : Build your first pipeline

# Lab 5 : Configure distributed builds using Master/Slave configuration

- In this lab, you will learn how to configure distributed builds using Master and Slaves architecture
  - Create new slave node
  - Launch new slave node using Java Web Start
  - Configure existing job to run with slave node
  - Monitor slave nodes through master node

- Monitor External Jobs
- Matrix or Multi-Configuration Jobs
- Distributed Builds
- Concept of a Pipeline
- Splitting a Big Job into Smaller Jobs
- File Fingerprint Tracking
- Using Jenkins for non-Java Projects
- Lab 5 : Configure distributed builds using Master/Slave configuration
- **Lab 6 : Build your first pipeline**

# Lab 6 : Build your first pipeline

- In this lab, you will build your first pipeline with Jenkins
  - Install Build Pipeline plugin
  - Install Parameterized Trigger plugin
  - Configure new jobs to build, test, analyze, and package the code
  - Create new Pipeline view

# Best Practices for Jenkins

- Culture Change
- Attend to broken builds
- Write unit tests
- Re-use build scripts across IDE and Jenkins
- Use your company's templates
- Use Plugins effectively

- Shorter Releases
  - Help team to focus on one project / few priorities at a time
  - Reduce the maintenance of merging and maintaining multiple branches
  - Have production-like environment for UAT testing
- Should be a **single version control** system for all source code
  - Should be able to build everything from a fresh checkout from the version control
  - Everyone should be working from the trunk and minimize the use of branches
- Make it a habit of **committing often** to the version control
  - It is easier to integrate your changes with other's changes
  - The conflicts will be resolved quickly and the team will get quick feedback of their changes
- Use **incremental Builds**
  - Can help you to verify your changes as fast as possible as the incremental builds run fast
  - Run full builds just a few times in a day for QA deployments

# Attend to Broken Builds

- Automate build and integration process to reduce number of failures
  - Single Click or Command should have capability to build entire code base
  - Quick feedback using fast and incremental builds
- Follow the principle of **build once and deploy anywhere**
- Setup **email notifications** to report failed and unstable builds
- **Fixing broken builds is the top priority**
  - Take ownership for addressing failures as soon as possible

- Follow the principles of **Test Pyramids**
  - Unit Tests > Service Tests > User Interface Tests
  - Unit tests should be the largest part of test automation strategy
  - User Interface tests should be as little as possible because they are expensive, time consuming and sometimes partially redundant
  - Service layer tests should not be ignored as it fills the gaps between unit and user interface testing
- Practice **Test Driven Development**
  - Make the goal of **100% test coverage** and write tests first if possible
  - Also helps to implement better design and quality code
- Automate the process for running tests and publishing code coverage reports
  - Leverage SonarQube to measure quality health of your code

- Use **Groovy** and **Gradle** to reuse code and script across IDE and Jenkins
- Groovy and Gradle customize, configure, and extend build process
- Leverage Gradle and Groovy scripts as part of actual build
  - Gradle is specifically focused on multi-project jobs
  - Run scriptler scripts as build steps to reuse them in multiple build jobs easily

# Use Discover Templates

- Use Templates to manage multiple jobs with **similar configuration**
- Use **jobs as templates** for other jobs
  - You can create few abstract templates jobs for building, testing and packaging
  - Then you can create multiple jobs from these templates for different modules or components or branches
- Use **Multi-job builds** to allow reusability of generic jobs across multiple projects
- Use **Templated builders** to simplify common tasks

# Use Plugins Effectively

- Better strategy is to not install new plugins or upgrade existing ones if not necessary
- Quite a few duplicate plugins with similar features exist, so do your homework before installing plugin
- Install one plugin at a time so you can verify your jobs after each install
  - Plugins may have compatibility issues with Jenkins as well as with other plugins
- Uninstall the plugins if you are not using them

# References

- <http://jenkins-ci.org/>
- <https://wiki.jenkins-ci.org/display/JENKINS/Home>
- <https://wiki.jenkins-ci.org/display/JENKINS/Plugins>
- <https://gradle.org/gradle-and-jenkins/>
- Jenkins – The Definitive Guide from O'REILLY

## Q & A

# Continuous Integration with Jenkins



**Thank you!**