Trivera Tech®
TECHNOLOGY TRAINING

Continuous Integration With Jenkins

Introducing Continuous Integration and Jenkins
Installing and Running Jenkins
Jenkins Job
Jenkins Plugins
SonarQube Plugin
Advanced Jenkins
Best Practices for Jenkins

Day 1: Sessions

- 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

Trivera Tech®
TECHNOLOGY TRAINING

Introducing Continuous Integration and Jenkins

Introducing Continuous Integration and Jenkins

Installing and Running Jenkins

Jenkins Job

Jenkins Plugins

SonarQube Plugin

Advanced Jenkins

Best Practices for Jenkins

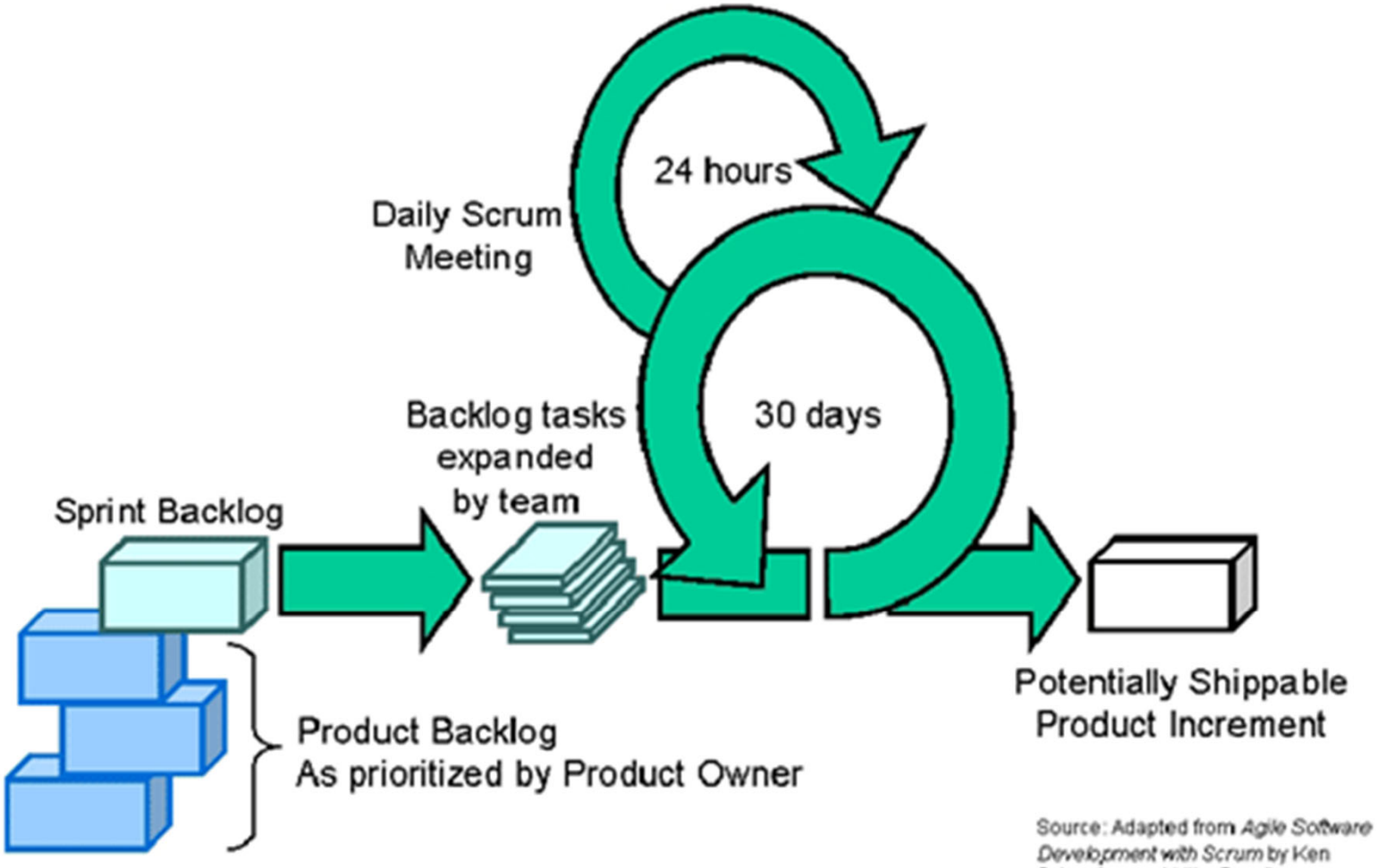
Topics in this Session

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

Agile Development

- Agile development describes 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

Topics in this Session

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

CI and CD in Agile Development

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

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 Fowler)
- Goal is to merge and test the code **continuously** 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 (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

- Single source repository for all developers
- Build automation
- Every change to VCS should make a new build
- Keep the builds fast and trackable
- Make the builds self-testing
- Test the builds in production-like environment
- Keep all verified releases in artifacts repository and available to everyone
- Publish coding metrics

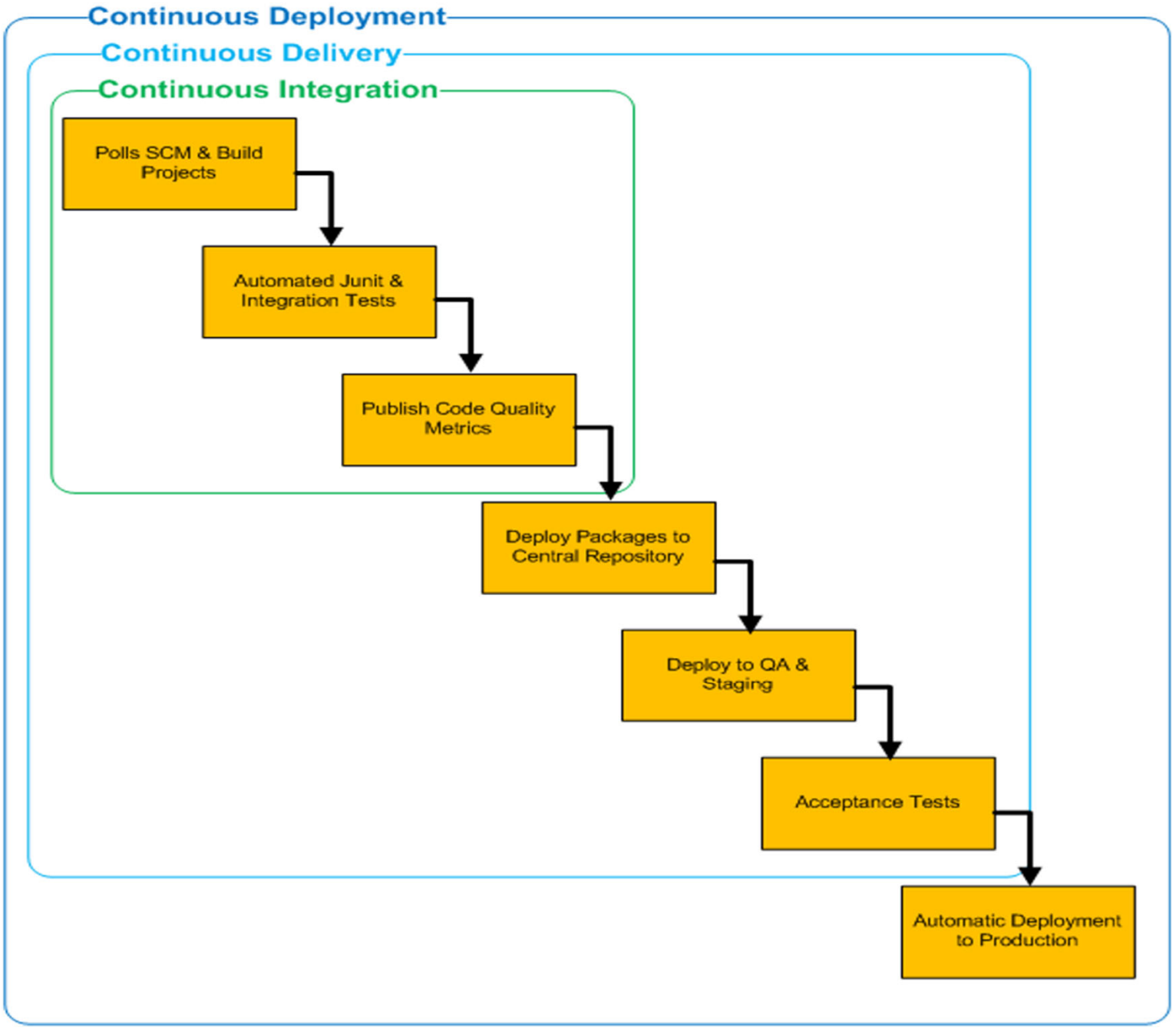
Continuous Delivery

- Continuous Delivery is a **natural extension** of Continuous Integration
  - Every change to the system has passed all the relevant automated tests and is 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 scheduling of the software releases
- Continuous Delivery is a core principle of **DevOps**

Continuous Deployment

- Continuous Development adds **automatic deployment to end users** in the Continuous Delivery process
- Continuous Deployment automatically deploys every successful build directly into production
  - Deploying the build 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 cycles as opposed to arbitrary deployments

Continuous Integration, Delivery and Deployment



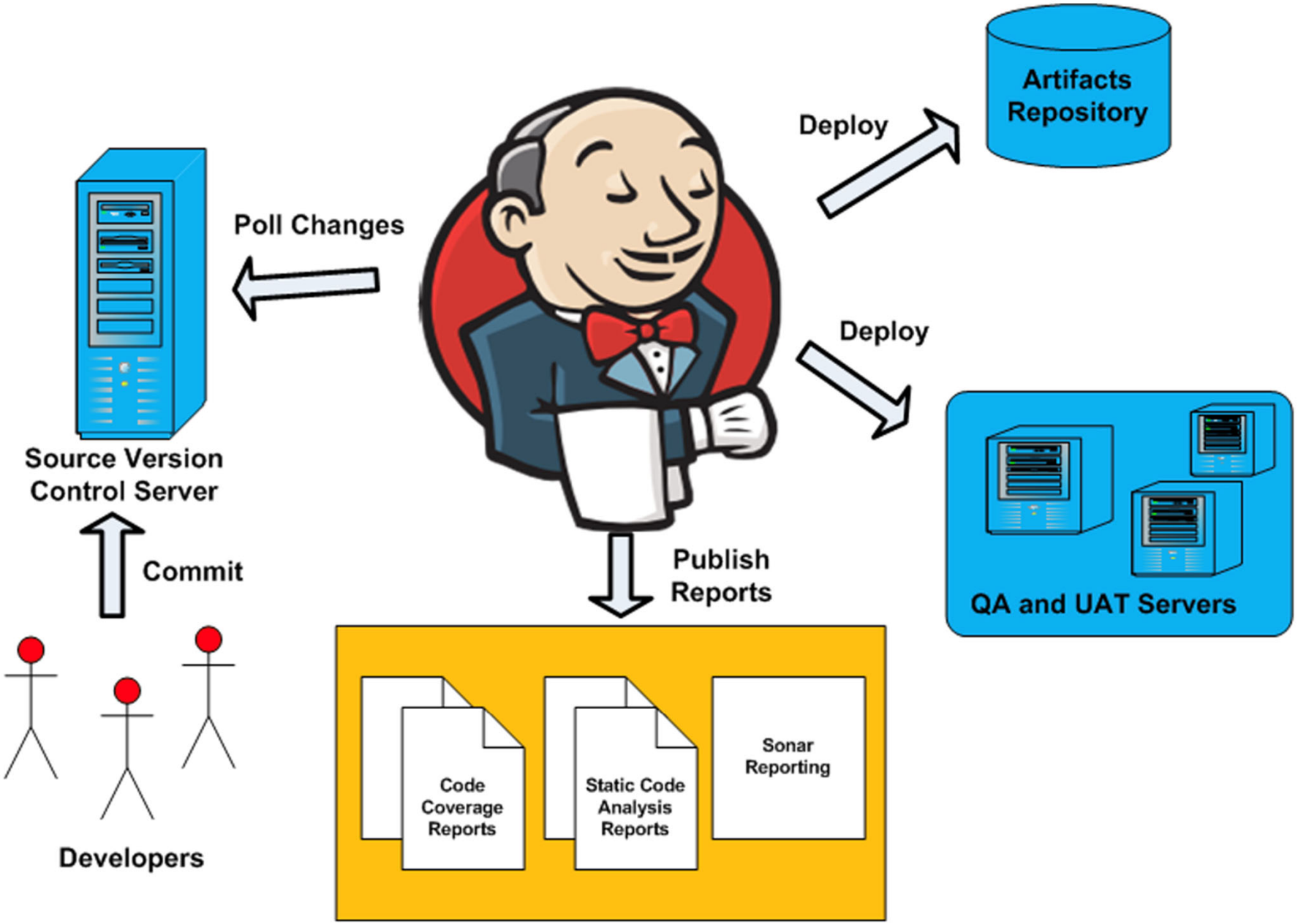
Topics in this Session

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

Jenkins

- **Continuous Integration (CI) Server**
- Open Source, free and written in Java
- Large and dynamic community with massive adoption
- Easy to install on many different platforms
- Easy to use with a user-friendly interface
- Significant amounts of resources and tutorials available
- More than 1000 plugins
- New plugins released every week
- Extensible architecture – easy to extend and customize
- Distributed builds
- Many more features...

How Jenkins Fits in CI and CD



Jenkins - History

- Jenkins is derived from **Hudson**
- Hudson was first released by **Kohsuke Kawaguchi** of Sun Microsystems in 2005
- Initially it was only used within Sun but by 2010 Hudson captured 70% of CI market share
- Oracle bought Sun Microsystems in 2010
- Due to naming and an open-source dispute, original Hudson team forked Jenkins forked from Hudson as a new project
- Oracle continued the development of original Hudson
- Majority of Hudson users **migrated to Jenkins** within few months of the initial Jenkins release

Topics in this Session

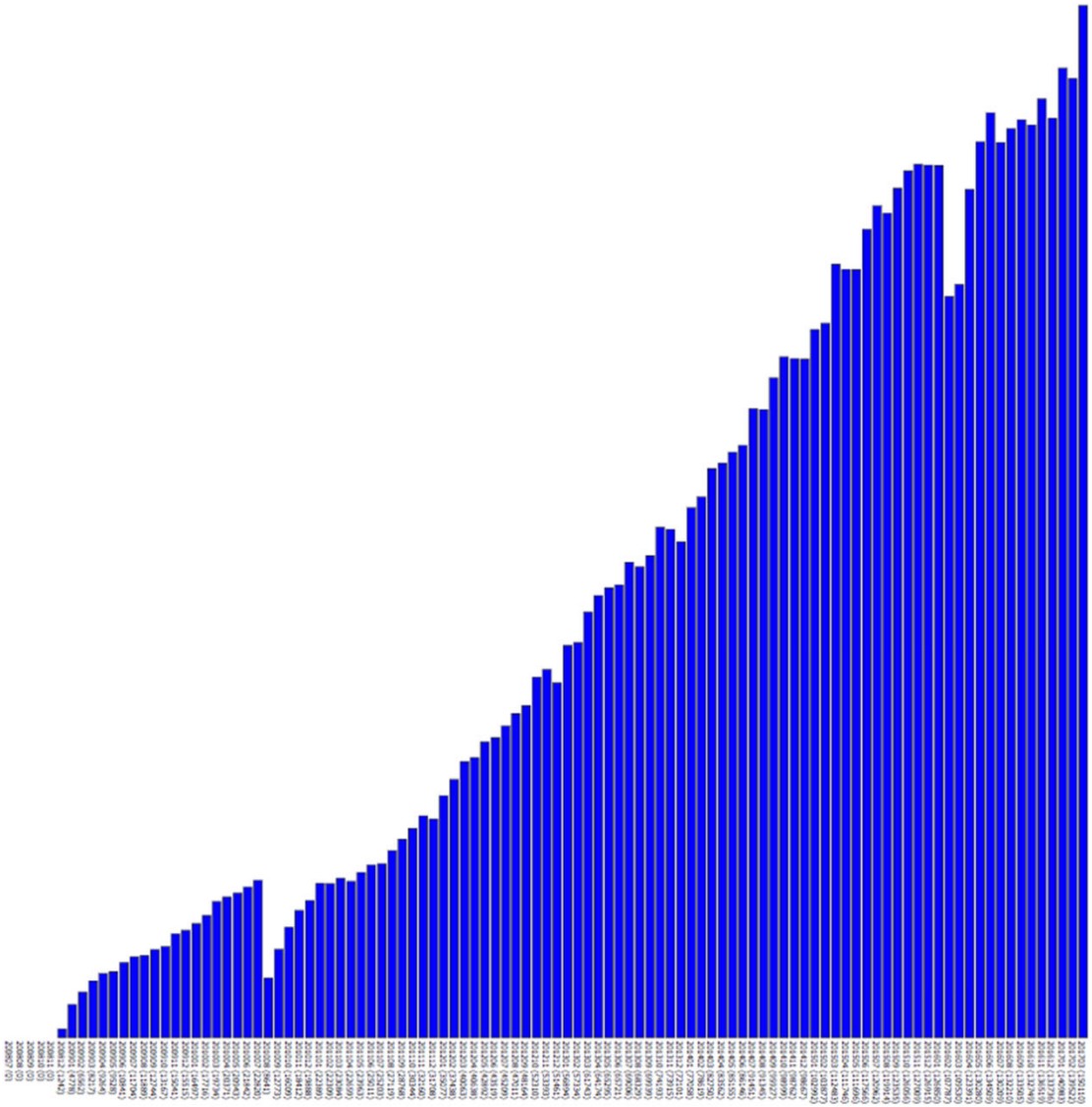
- Agile Development
- Continuous Integration versus
- Continuous Delivery versus
- Continuous Deployment
- History of Jenkins
- **State of the Jenkins Community**

State of the Jenkins Community

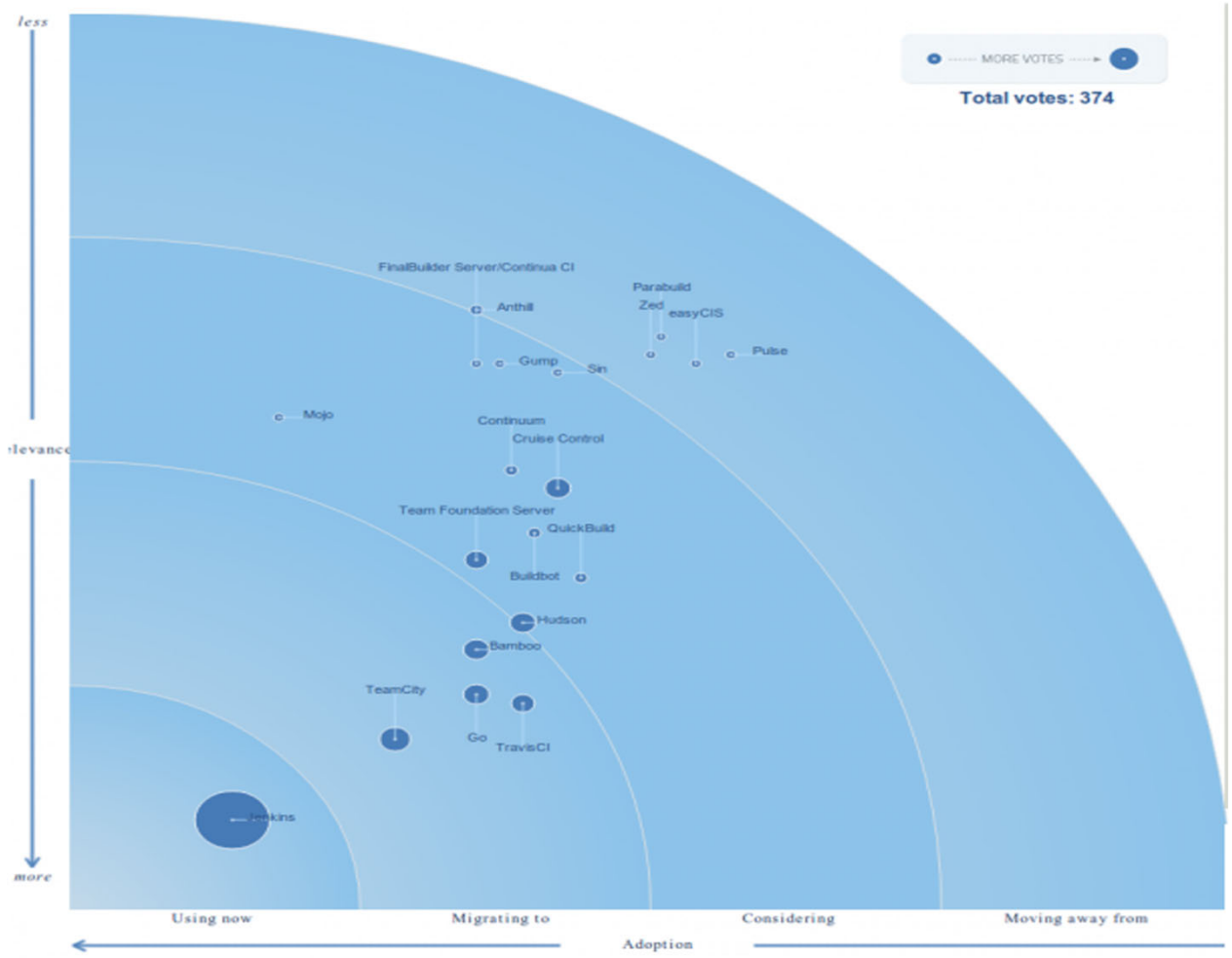
- Largest installed base of any open-source continuous integration and delivery platform
- More than **100K active users** in open-source Jenkins CI project
- Community contributed with more than **1000 plugins**
- Over 1000+ public repositories on GitHub and strong commit activity
- Quick feedback with 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
- 04/2007-04/2017



InfoQ CI Survery 2014

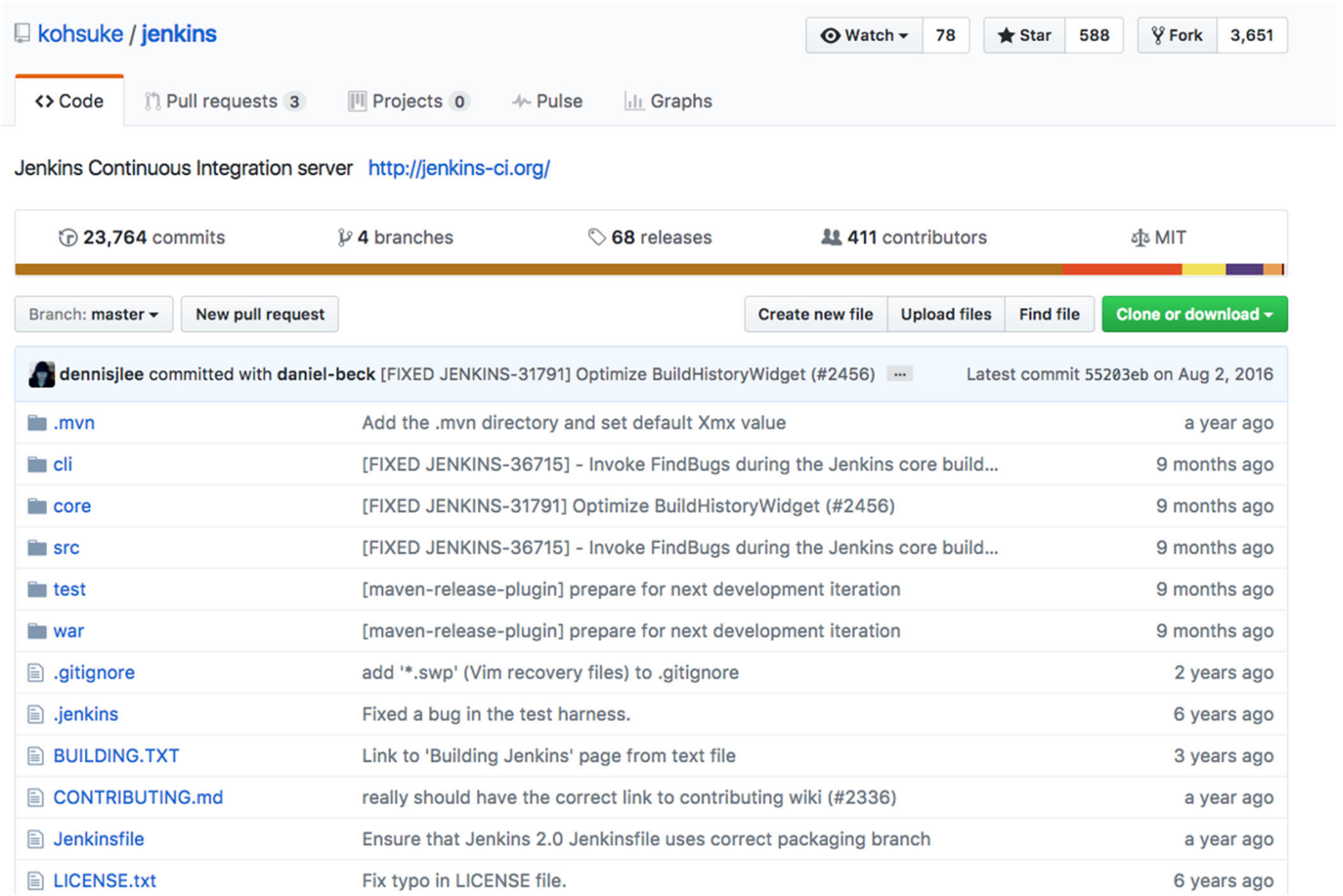


- **Ref: https://jenkins-ci.org/blog/2014/04/11/infoq-ci-survey-2014**

Continuous integration is elusive

- As above, Jenkins is king, however...
- Survey says
  - 14% deploy on an hourly basis
  - 34% deploy once a day
  - 21% deploy weekly and 31% deploy less often than weekly
- (See https://www.infoq.com/news/2017/03/agile-king-ci-ilusive-goal)

Jenkins Very Active GitHub Repository



Trivera Tech®
TECHNOLOGY TRAINING

Installing and Running Jenkins

Introducing Continuous Integration and Jenkins

Installing and Running Jenkins

Jenkins Job

Jenkins Plugins

SonarQube Plugin

Advanced Jenkins

Best Practices for Jenkins

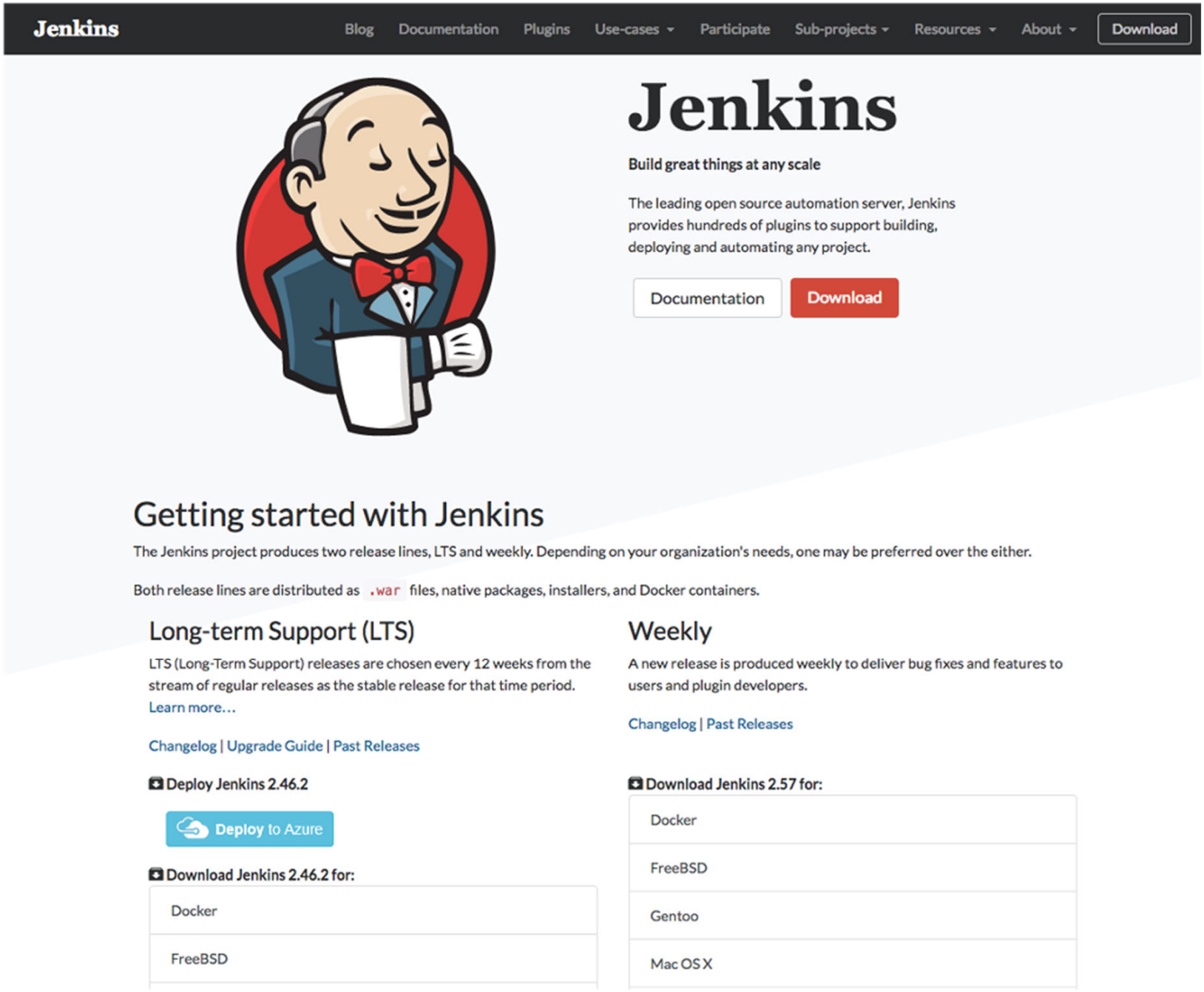
Topics in this Session

- **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

Installing 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/



Running Jenkins from the jar File

- Download the WAR as instructed in the labs
- Open command prompt and execute the following command
- `java -jar jenkins.war`
- It uses Jetty to run Jenkins

Installing Jenkins in a Servlet Container

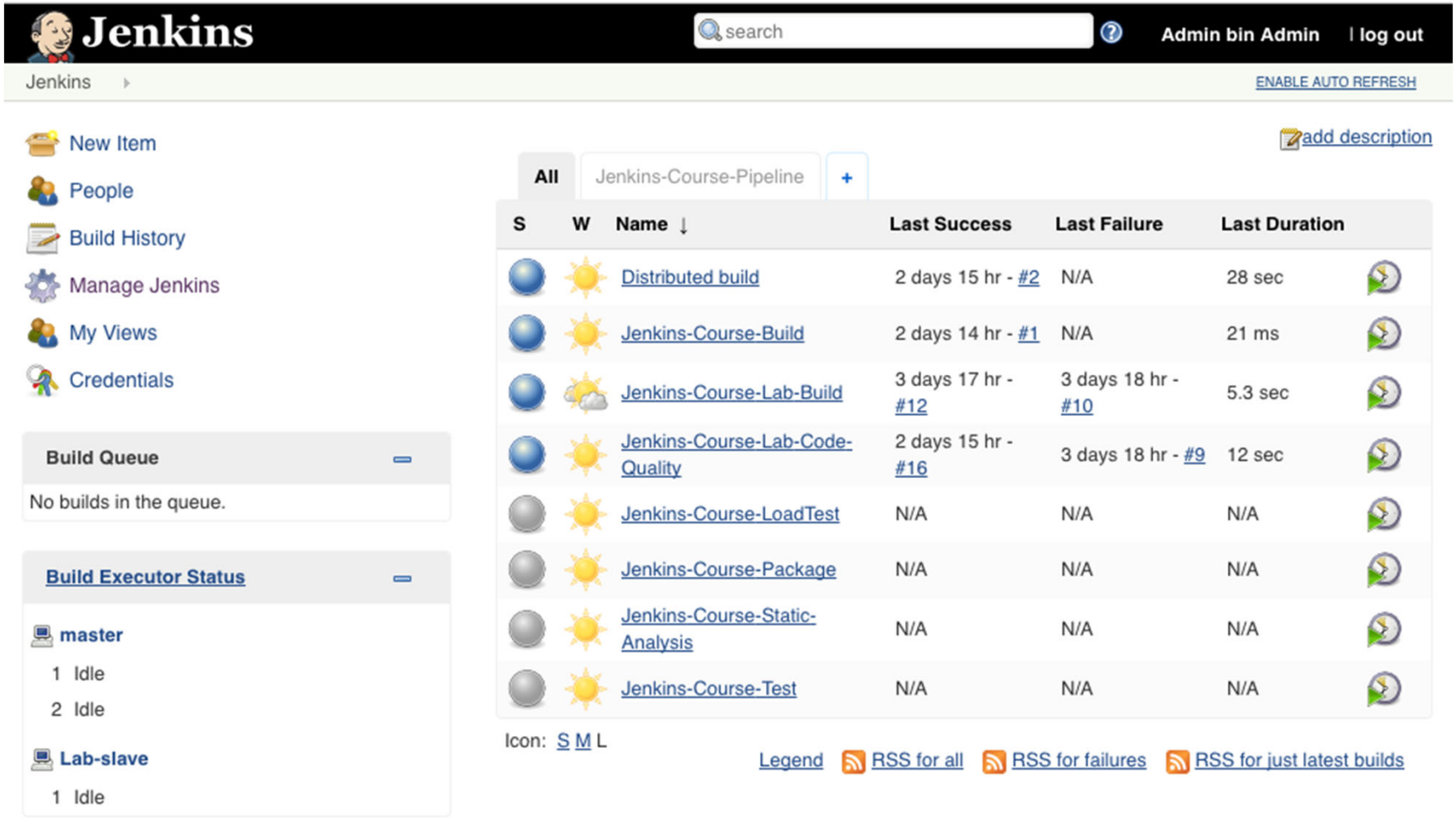
- **Tomcat and Jetty** are the most popular containers for Jenkins
  - Install Tomcat at appropriate folder
  - Simply copy jenkins.war to $TOMCAT_HOME/webapps folder
- You can install Jenkins on any other Servlet Containers
- However some containers may required minor configuration changes to work with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

Installing Jenkins using Platform Specific Installer

- 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

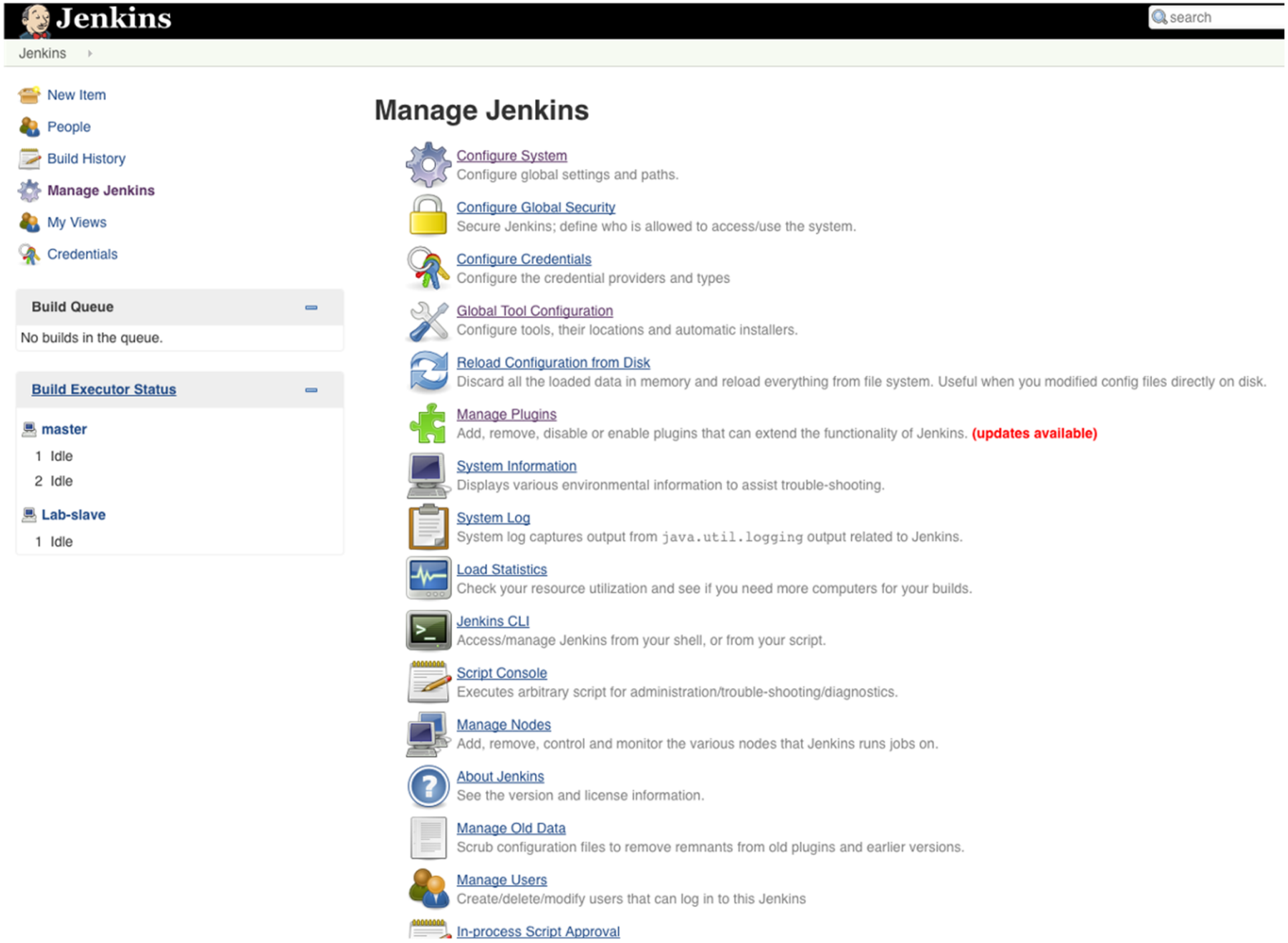
- Verify Jenkins by pointing your browser to http://localhost:8080
- The screenshot below shows all labs completed



Manage Jenkins

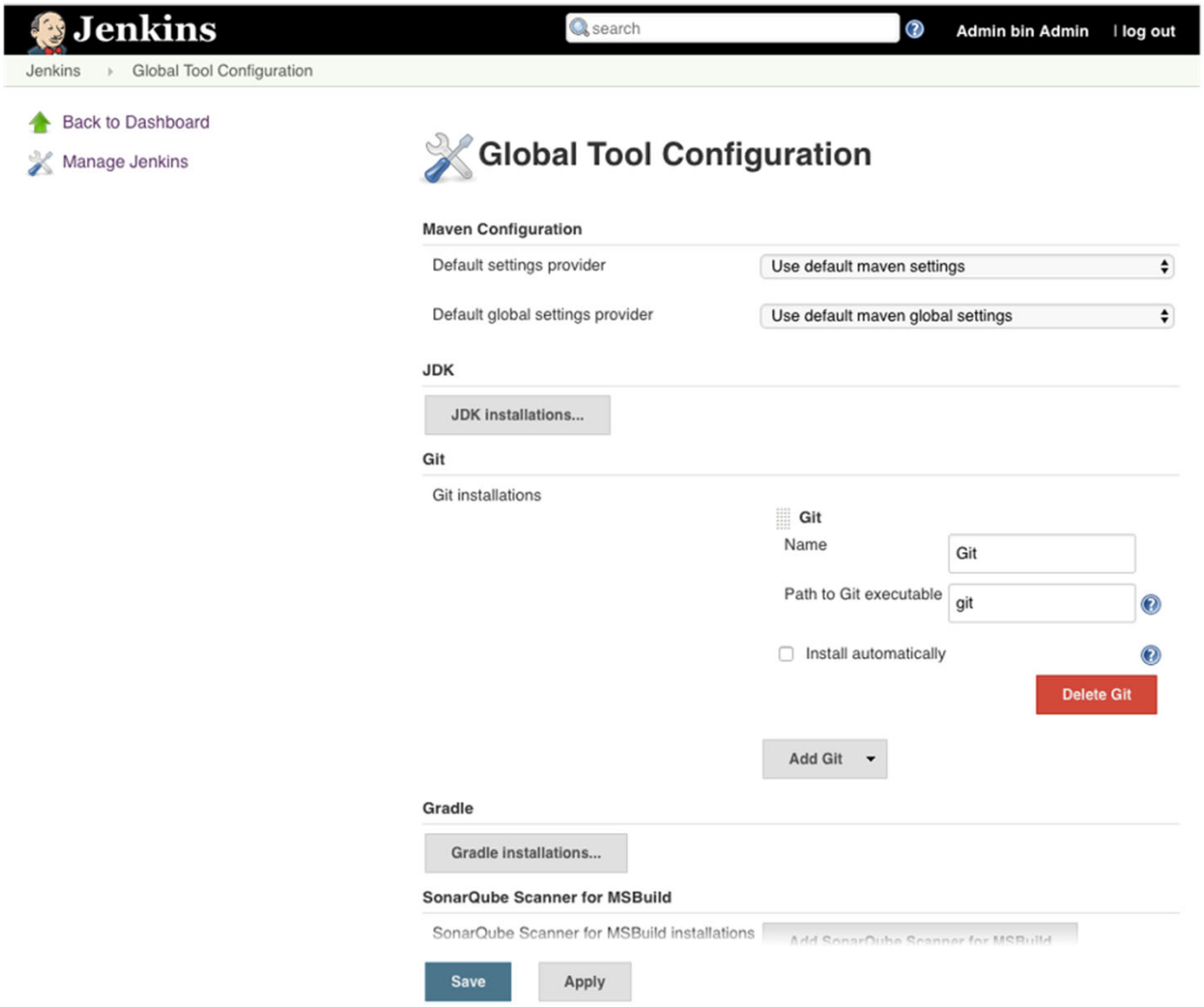
- **All in one configuration dashboard** for managing Jenkins
- Configure JDKS, Ant, Maven, Security, Email and Version Controls
- Install new plugins and update any existing plugins
  - Will review Gradle and Git plugins in this section
- Configure parallel and distributed builds
- Reload configuration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

Manage Jenkins Screen



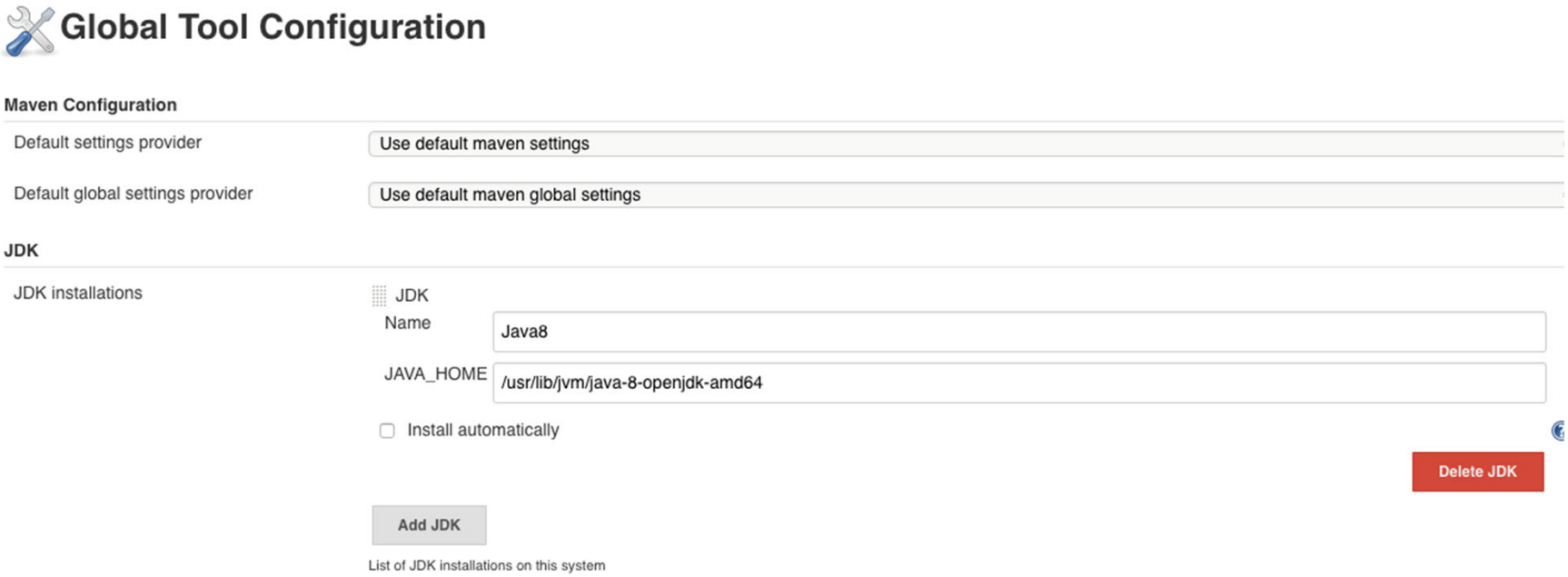
Configure System

- The two important tabs are “Configure System” and “Global Tool Configuration”.
- There you can set up JDK, Maven, Ant, Git and Gradle.

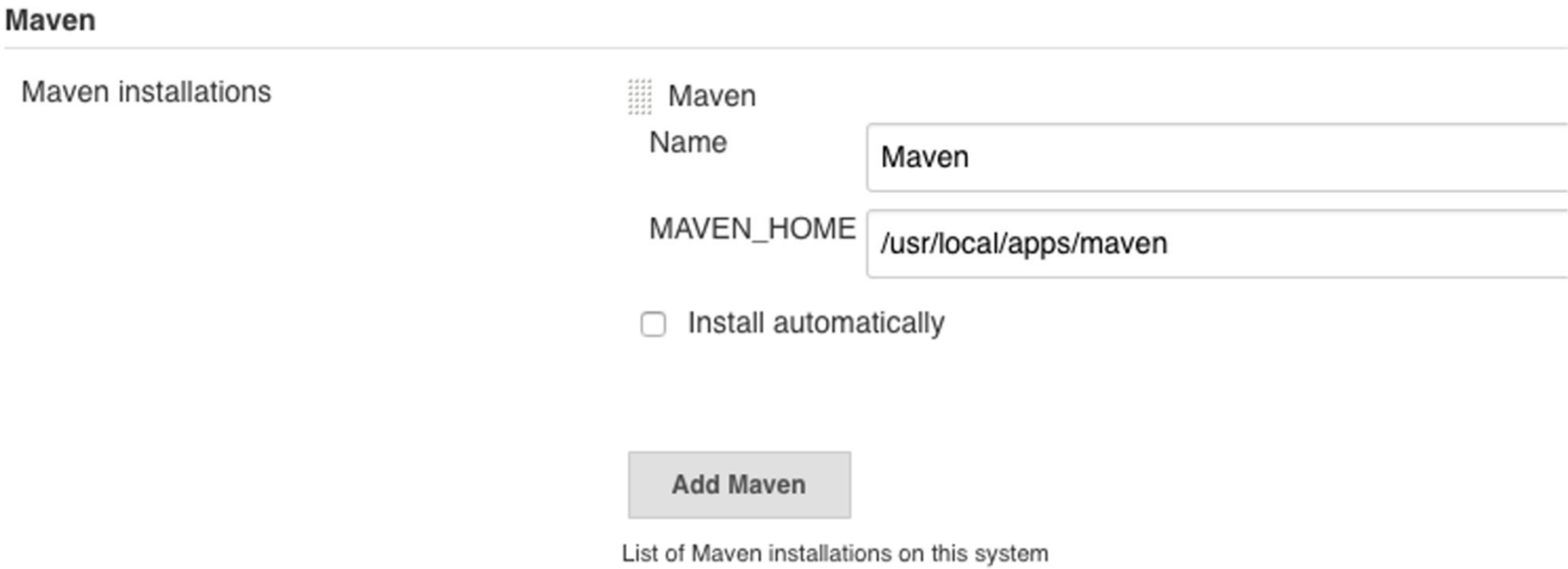


Configure JDK

- Provide JDK name and JAVA_HOME
  - You can have multiple JDKs listed here

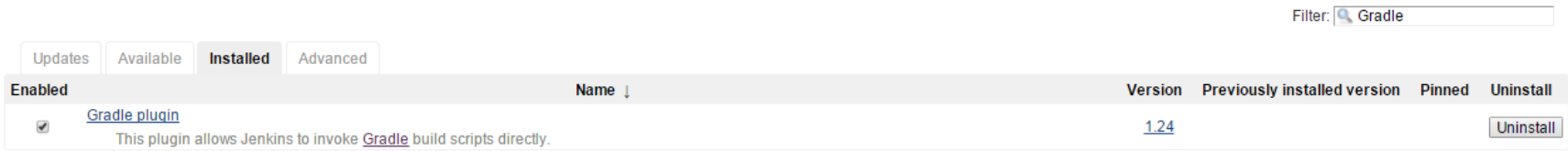


Configure Maven



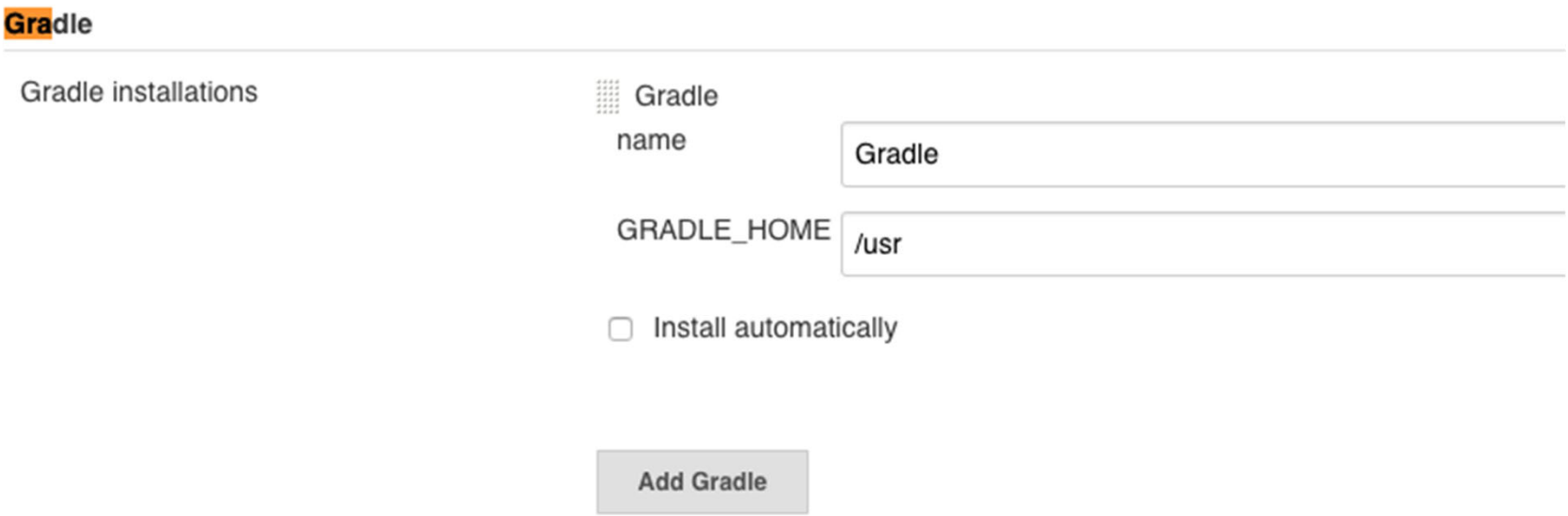
Gradle Jenkins Plugin

- 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



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



Upgrade Jenkins

- 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, just in case (Jenkins keeps your data)

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

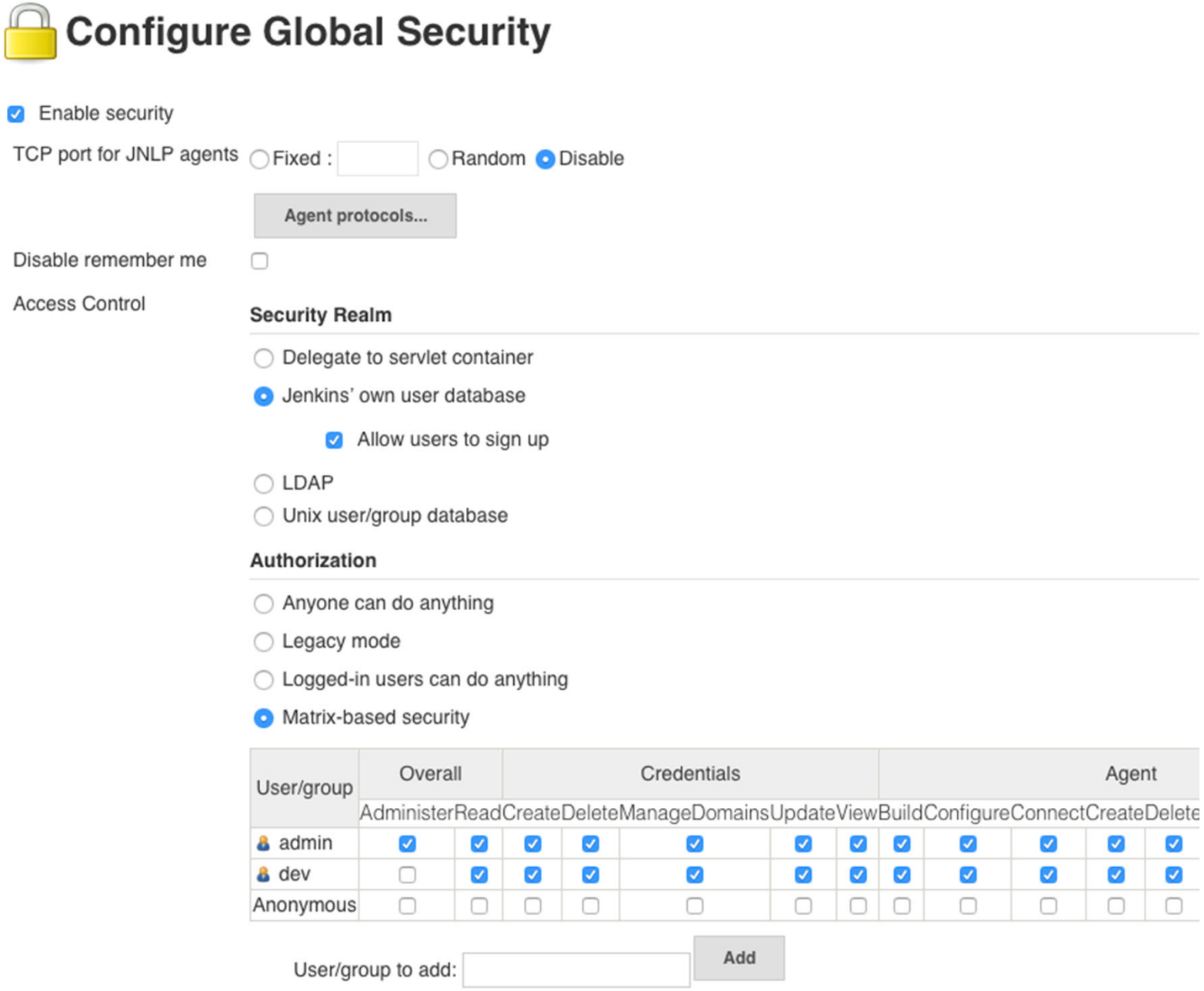
Topics in this Session

- 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

Setup Security

- 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



Security Realms

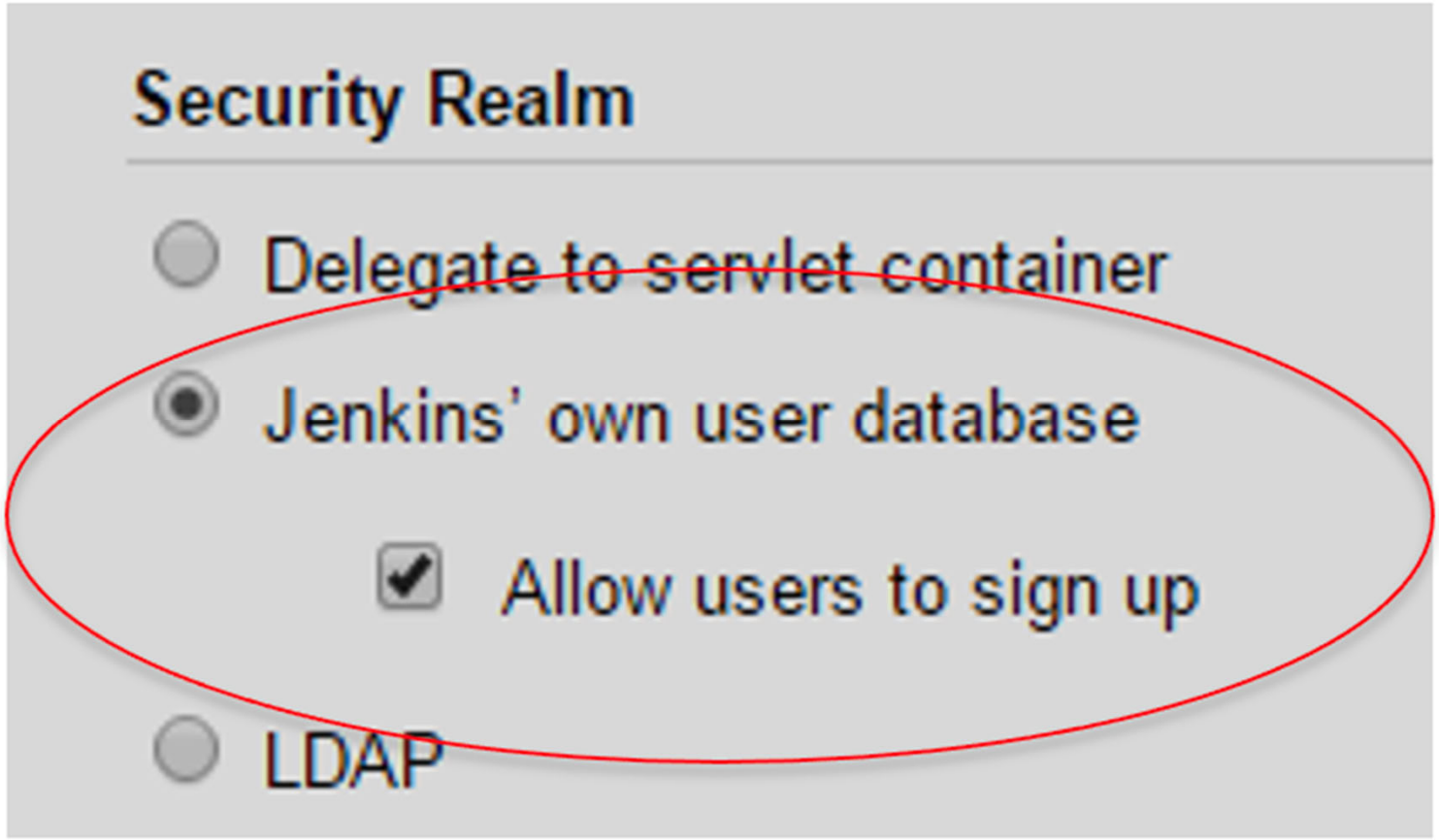
- Identify Jenkins users and establish user authentication methods
- Few options for user authentication methods
  - Jenkin's Built-in User Database
  - Using an Active Directory
  - Using Unix User and Groups
  - Delegating to the Servlet Container

Jenkins Built-In User Database

- 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

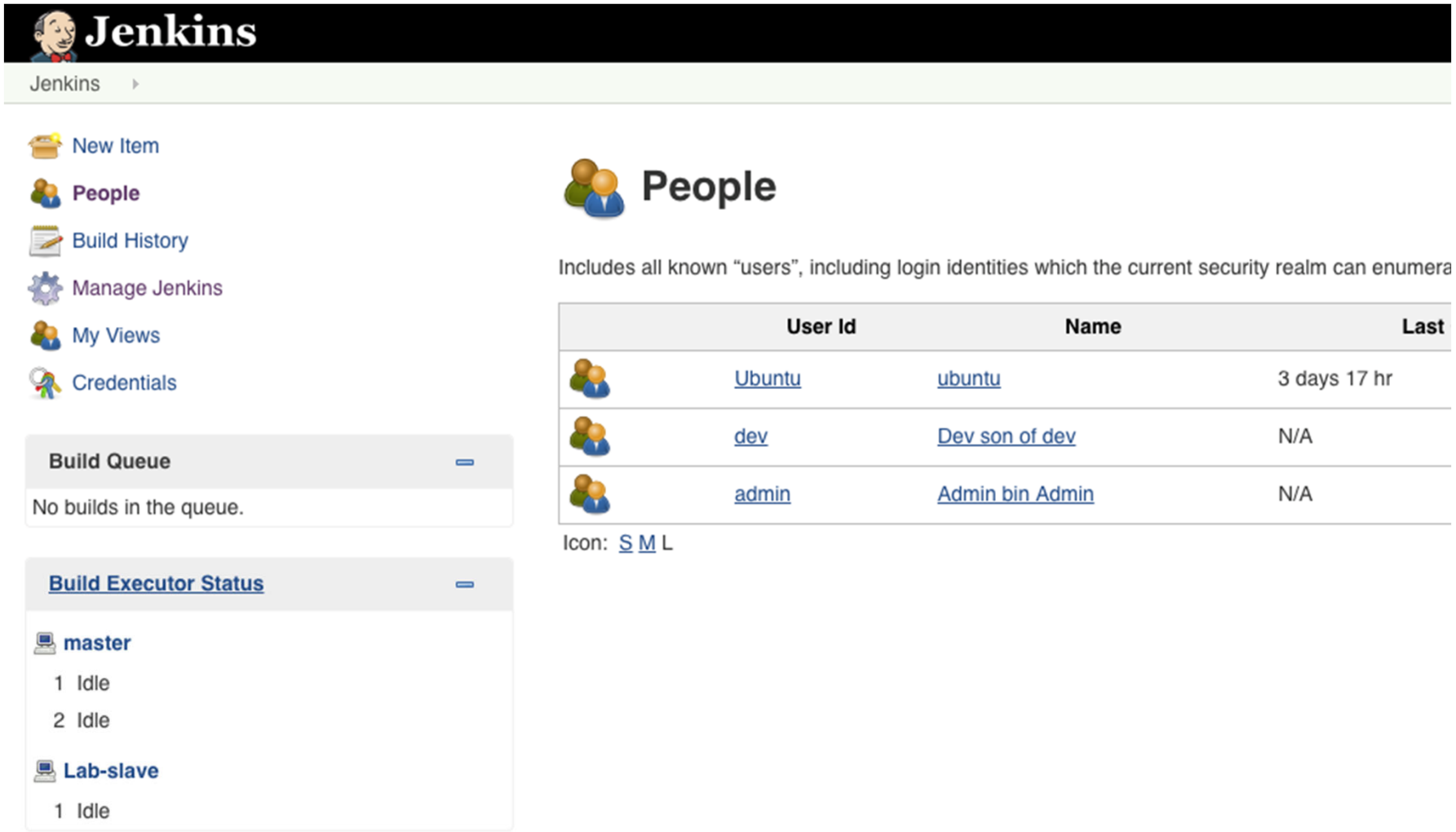
How to enable Jenkins Database

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



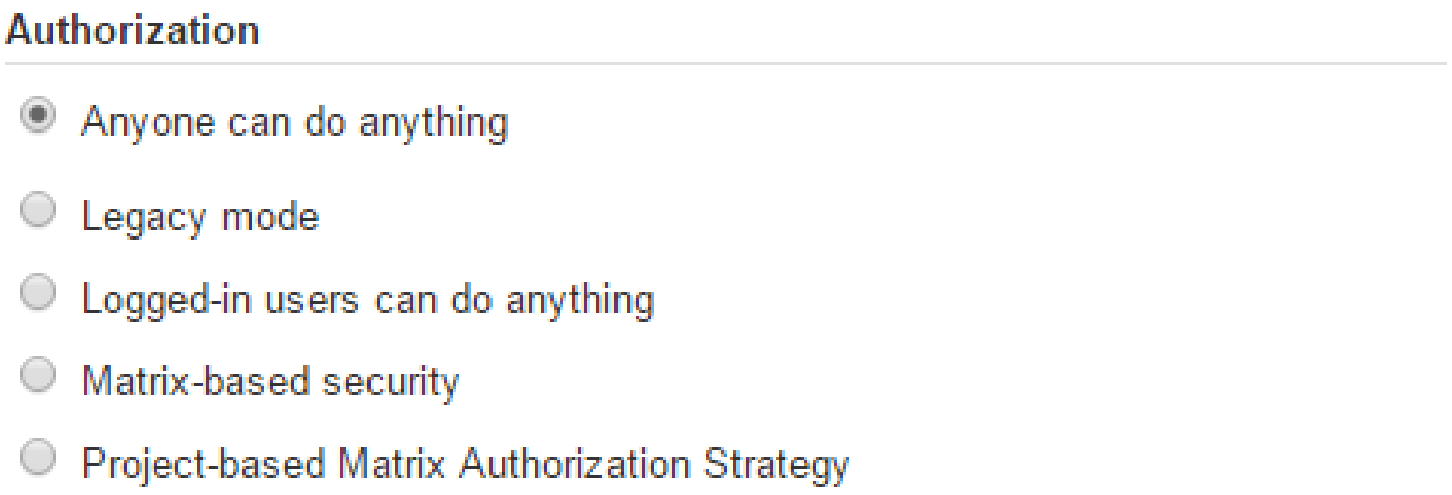
View and Manage Users

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



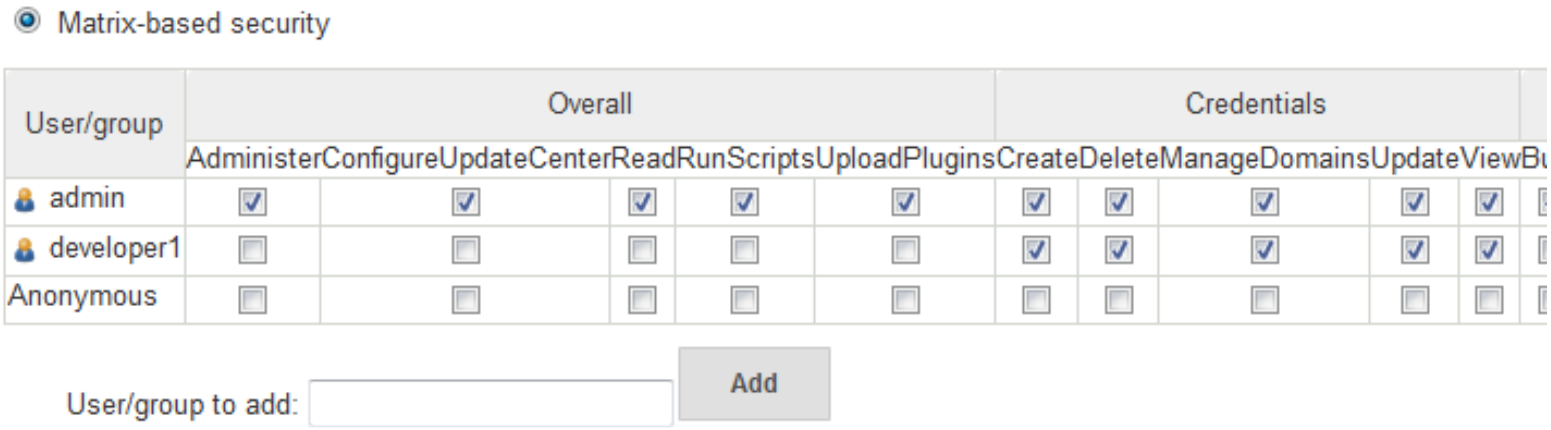
Authorization (In Global Security Configuration)

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



Matrix-based Security

- 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



Common Permissions

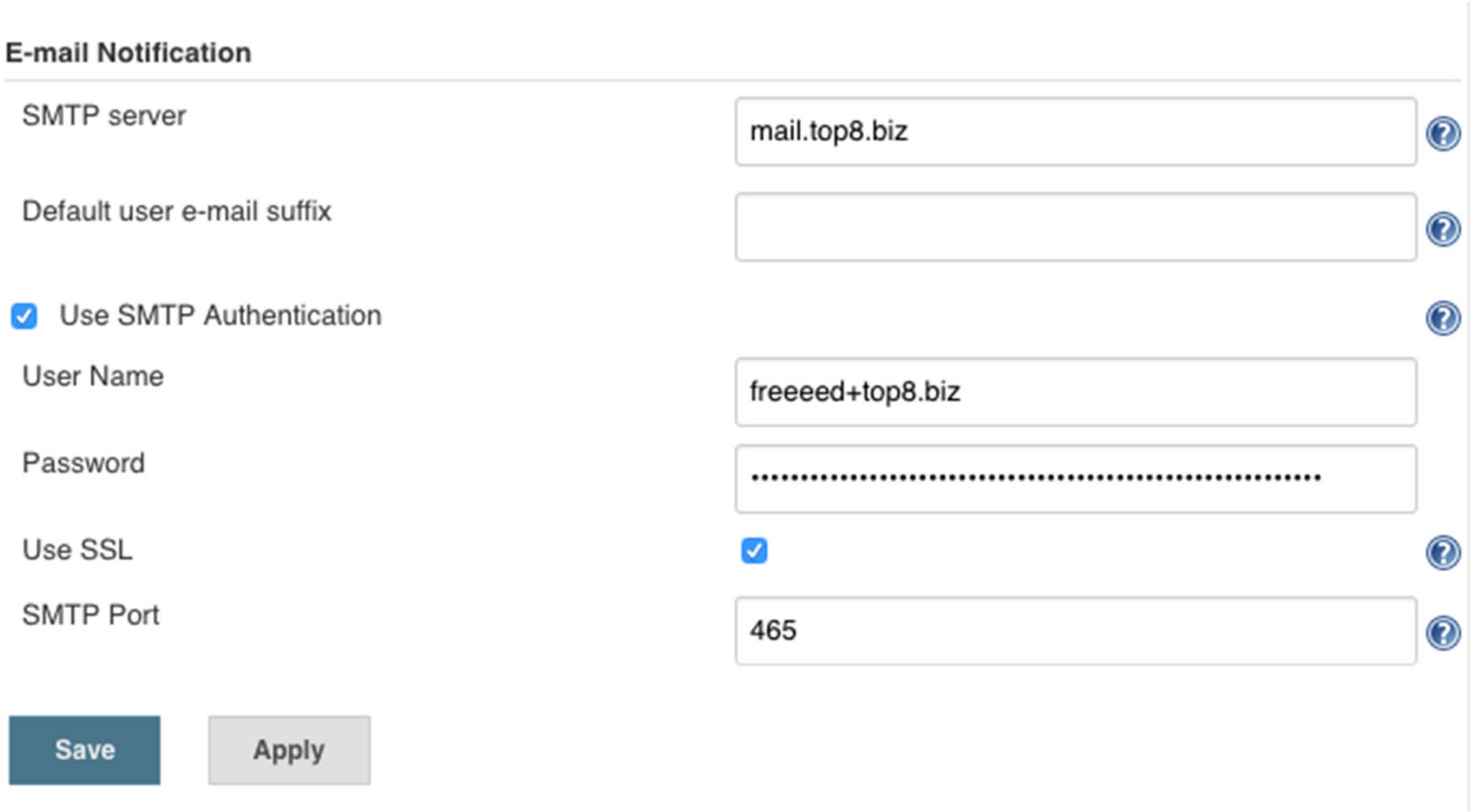
- Overall
  - Administer access
  - Run groovy scripts
  - Upload plugins or configure update center
- Job
  - Create, Delete and Configure Jobs
  - Start a new build or Cancel a running build
- View
  - Create, Delete and Configure views
- SCM
  - Create a new tag in source code repository for any build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

Topics in this Session

- 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
  - Provide other required parameters



Version Control

- 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

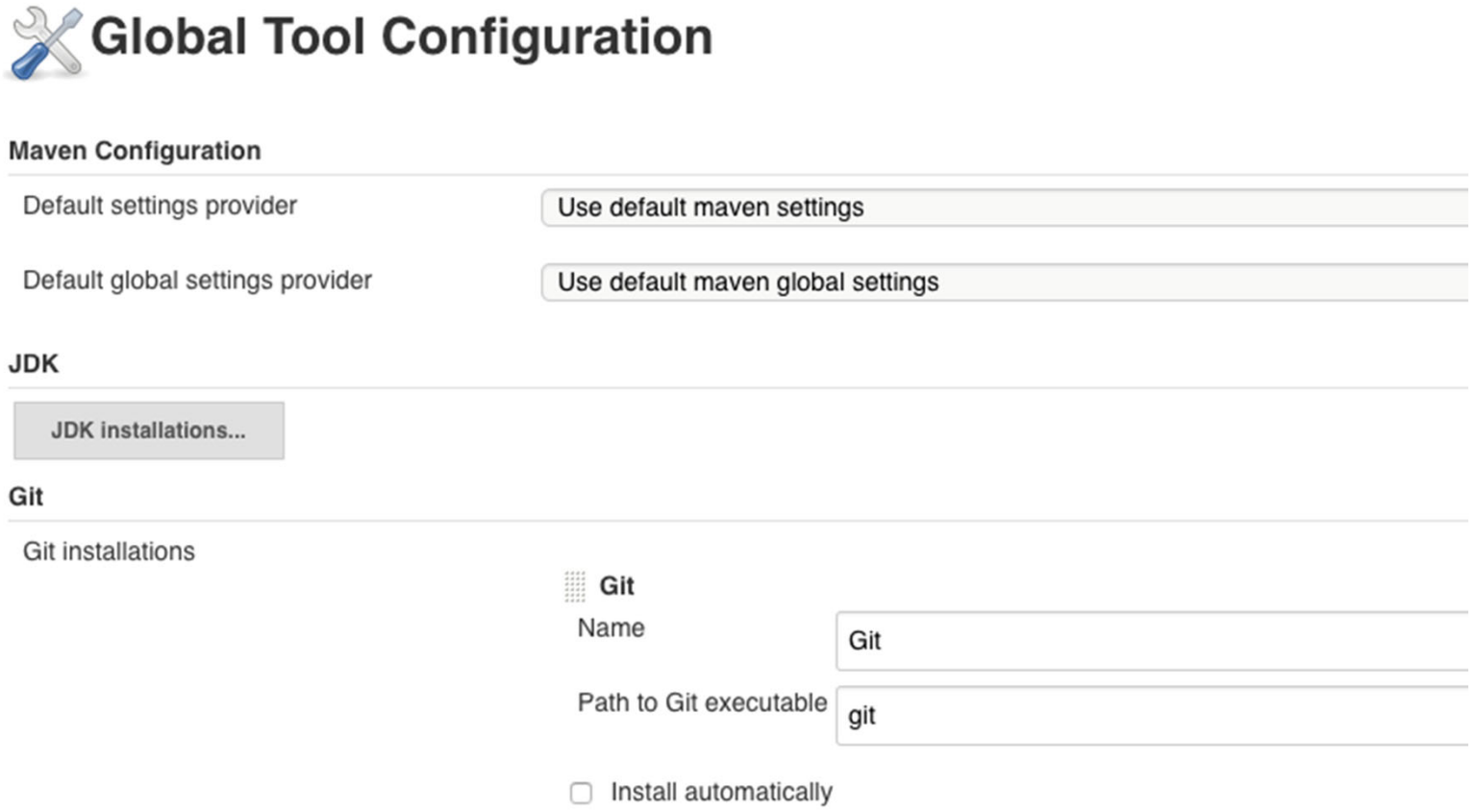
- Subversion is part of Jenkins installation and so no special configuration required
- Simply provide Subversion repository URL in your job definition
  - Jenkins will automatically verify the URL and shows error if it is not valid
  - If repository URL requires authentication then Jenkins will prompt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

Working with Git

- 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

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



Topics in this Session

- 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
- Lab 1: Install and Configure Jenkins

Topics in this Session

- 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

Trivera Tech®
TECHNOLOGY TRAINING

Jenkins Job

Introducing Continuous Integration and Jenkins
Installing and Running Jenkins
Jenkins Job
Jenkins Plugins
SonarQube Plugin
Advanced Jenkins
Best Practices for Jenkins

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

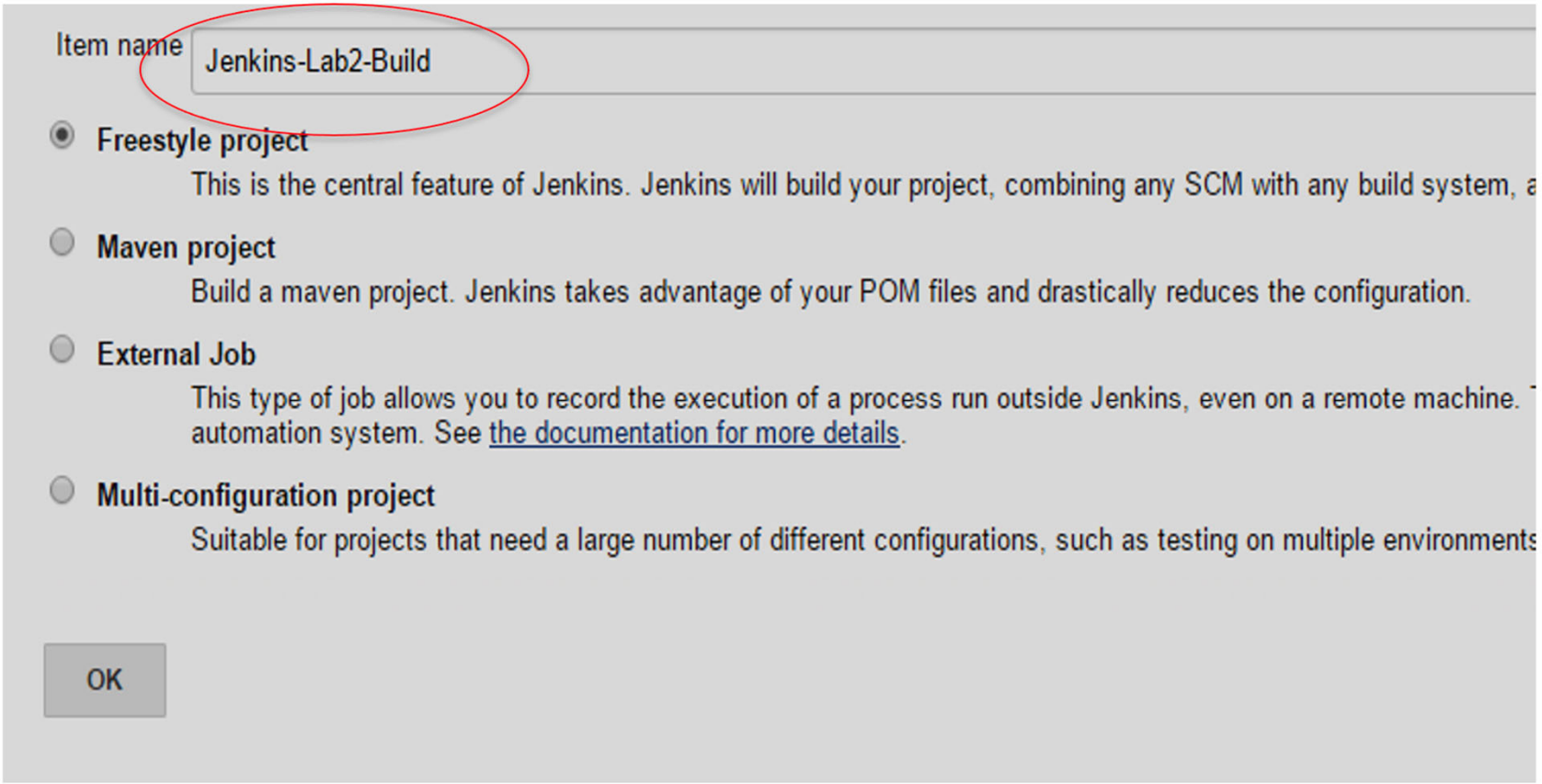
Jenkins Build Jobs

- 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 Build Jobs

- 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



Jenkins List Views

- 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

Topics in this Session

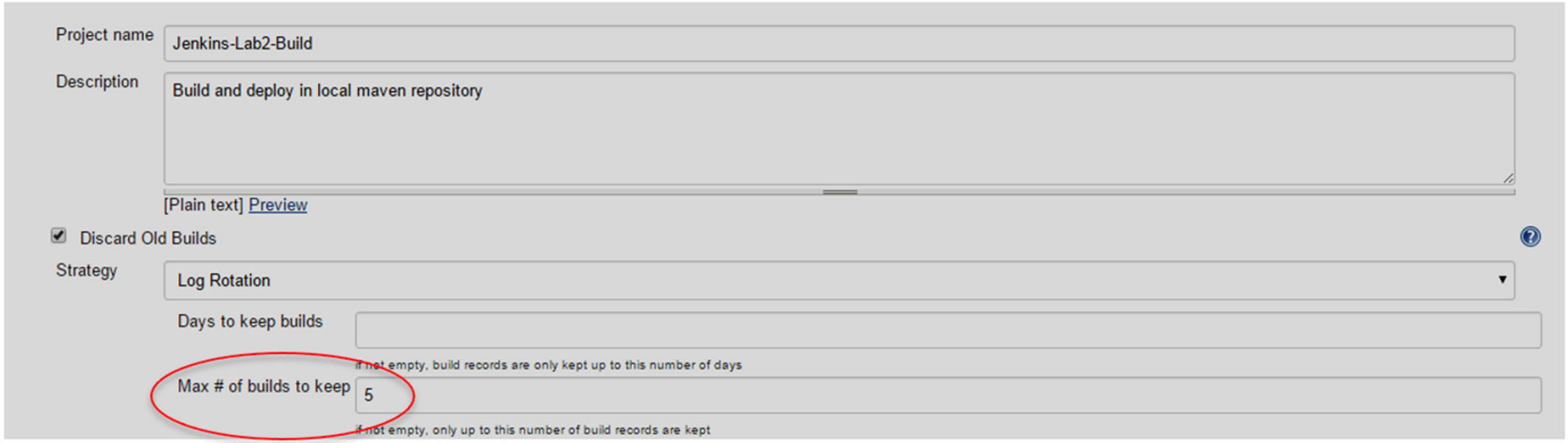
- Create a job
- **Configure a Job**
  - Configure a Freestyle Job
  - Configure a Maven Job
  - Copy from Existing Jobs
- Trigger a Build Job
  - Run a job manually
  - Run a job on a regular schedule
  - Run a job when source code is checked into version control
  - Run a Job After Another Job Finished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

Configuring 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

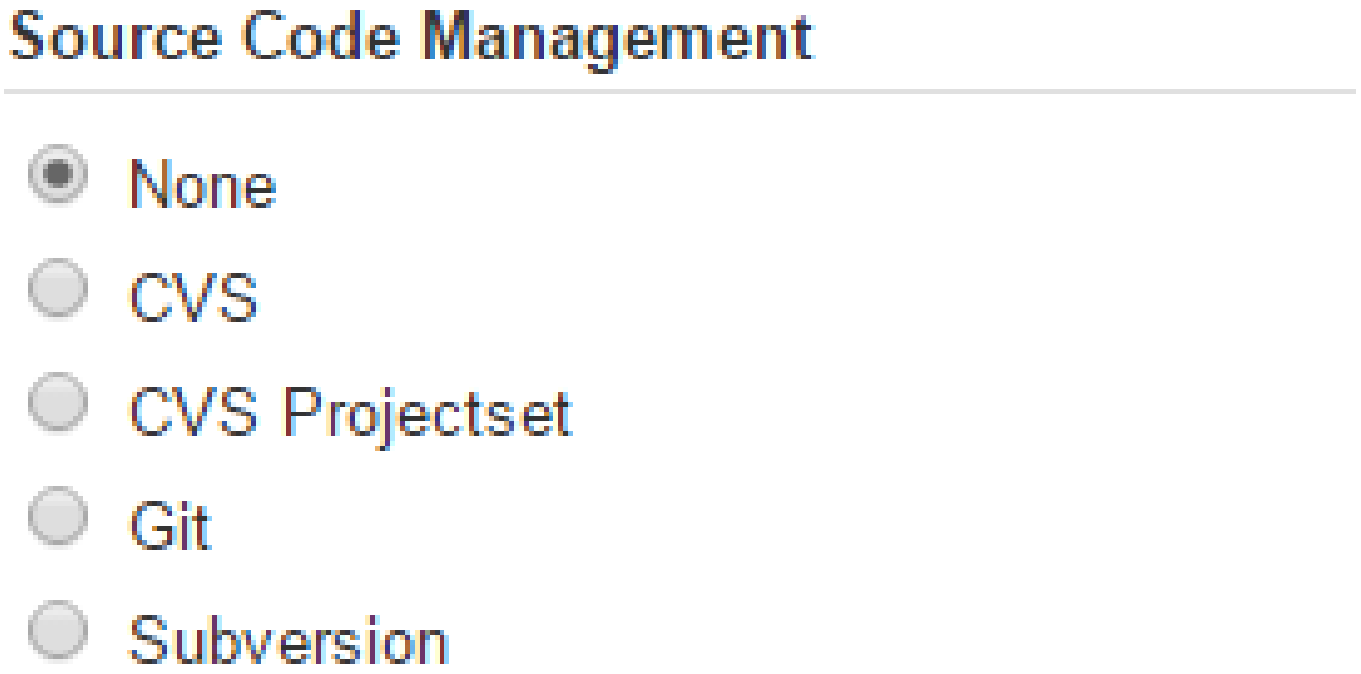
General Options – Name and Old Builds

- 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



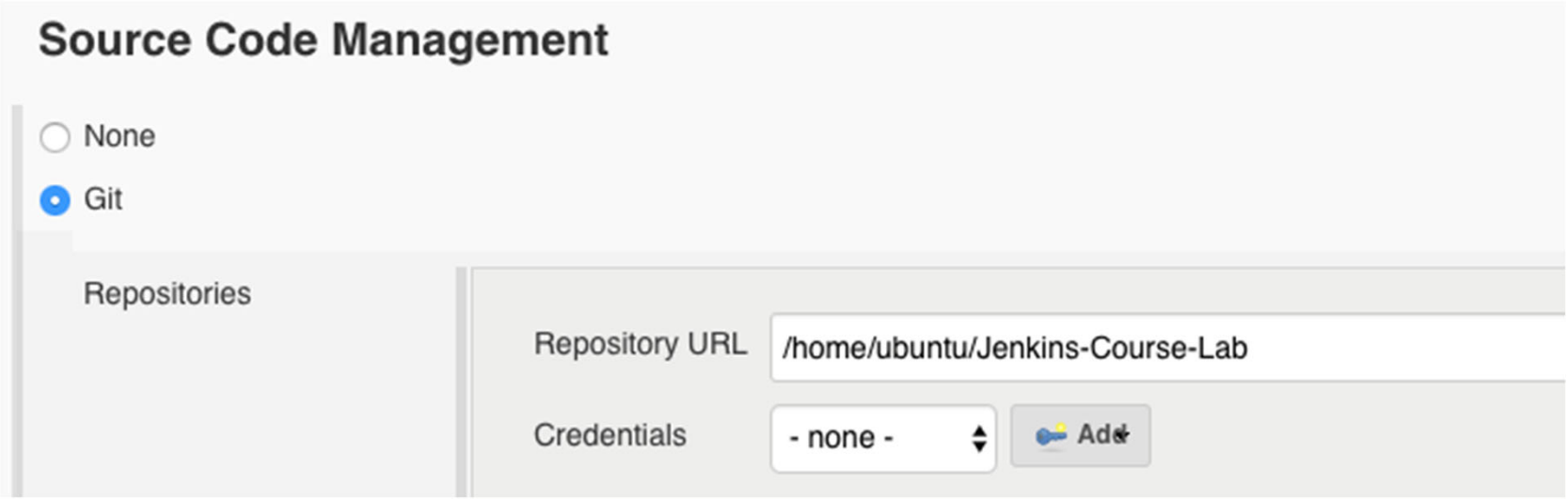
Source Code Management

- 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

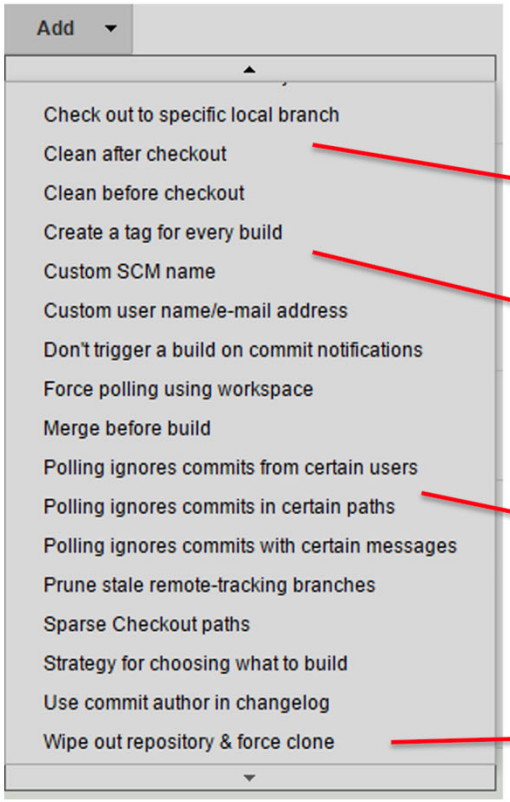


Working with Git

- In most cases, you only need to provide URL of the Git repository
- Please note : for our labs, we point to a local Git repo



Git – Additional Functionalities



Cleaning any untracked files after or before checkout

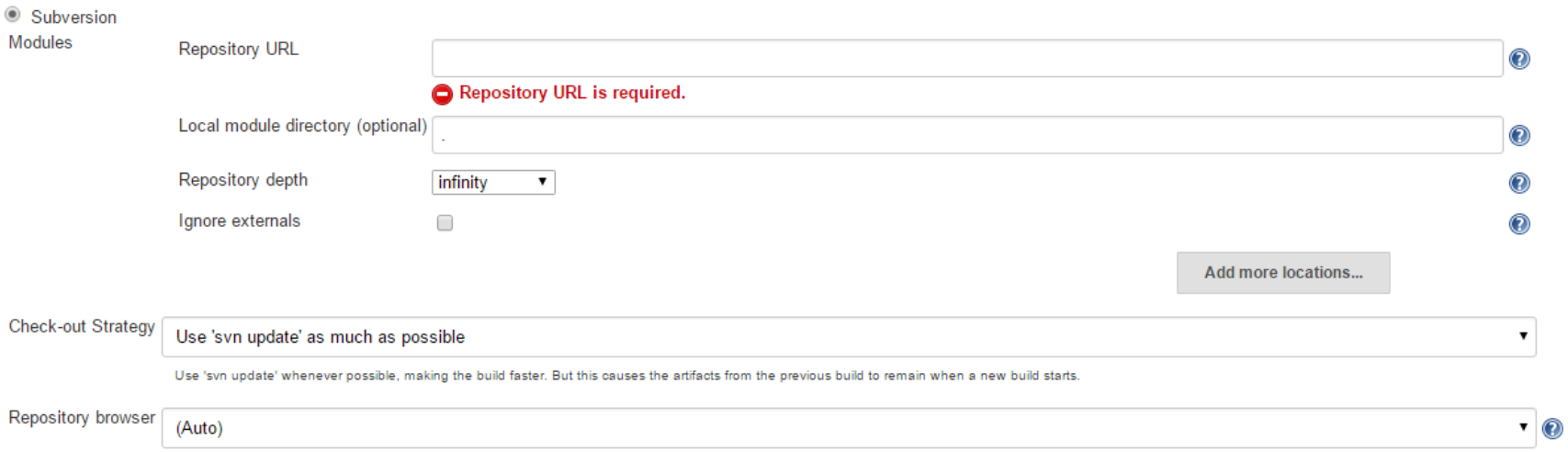
Configure to create new tag for every successful build

Specify to ignore certain users, paths or messages

Option for deleting and rebuilding workspace for every build

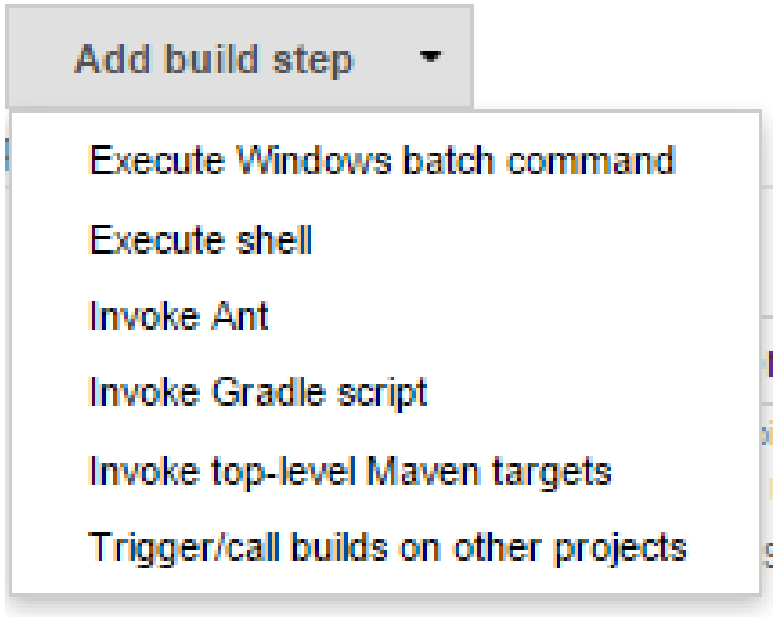
Working with Subversion

- Simply provide Subversion repository URL (similar to Git)
- Jenkins will automatically verify the URL and prompts for authentication if credentials are required
- Jenkins offers a 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



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

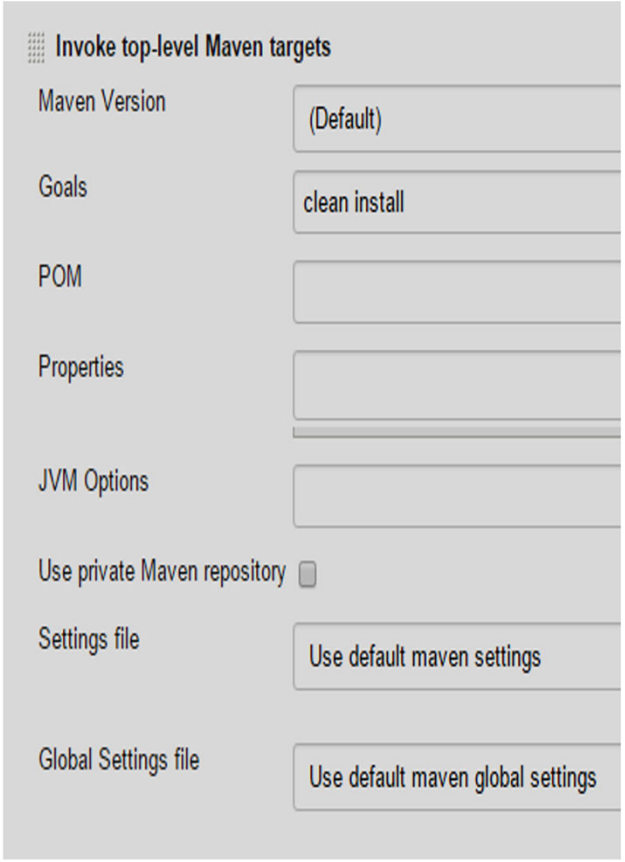
- There are two ways to run the Jenkins jobs with Gradle
  - Using Invoke Gradle through Gradle installation
  - Using Gradle Wrapper without Gradle installation
- In both options, you are only listing gradle tasks for the most Jenkins jobs

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

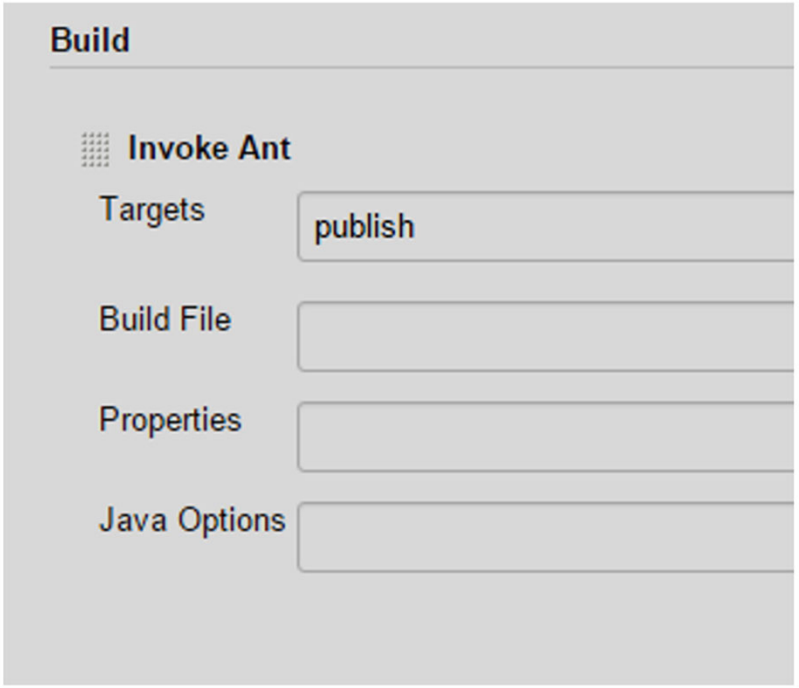


Ant Build Steps

- 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 an Ant script
- Java Options can be used to specify Java memory limits



Execute shell

- 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

Execute Windows Batch Command

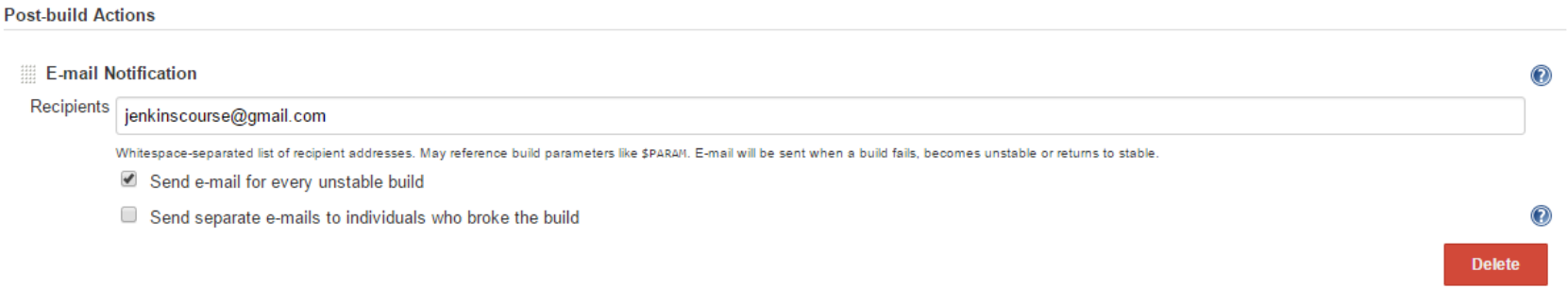
- Run Windows commands or batch scripts. It will be run in the workspace as the current directory
- The Batch command is running inside a cmd
  - No need to specifically start a new one
  - Just "call" your BAT file

Post Build Actions

- 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

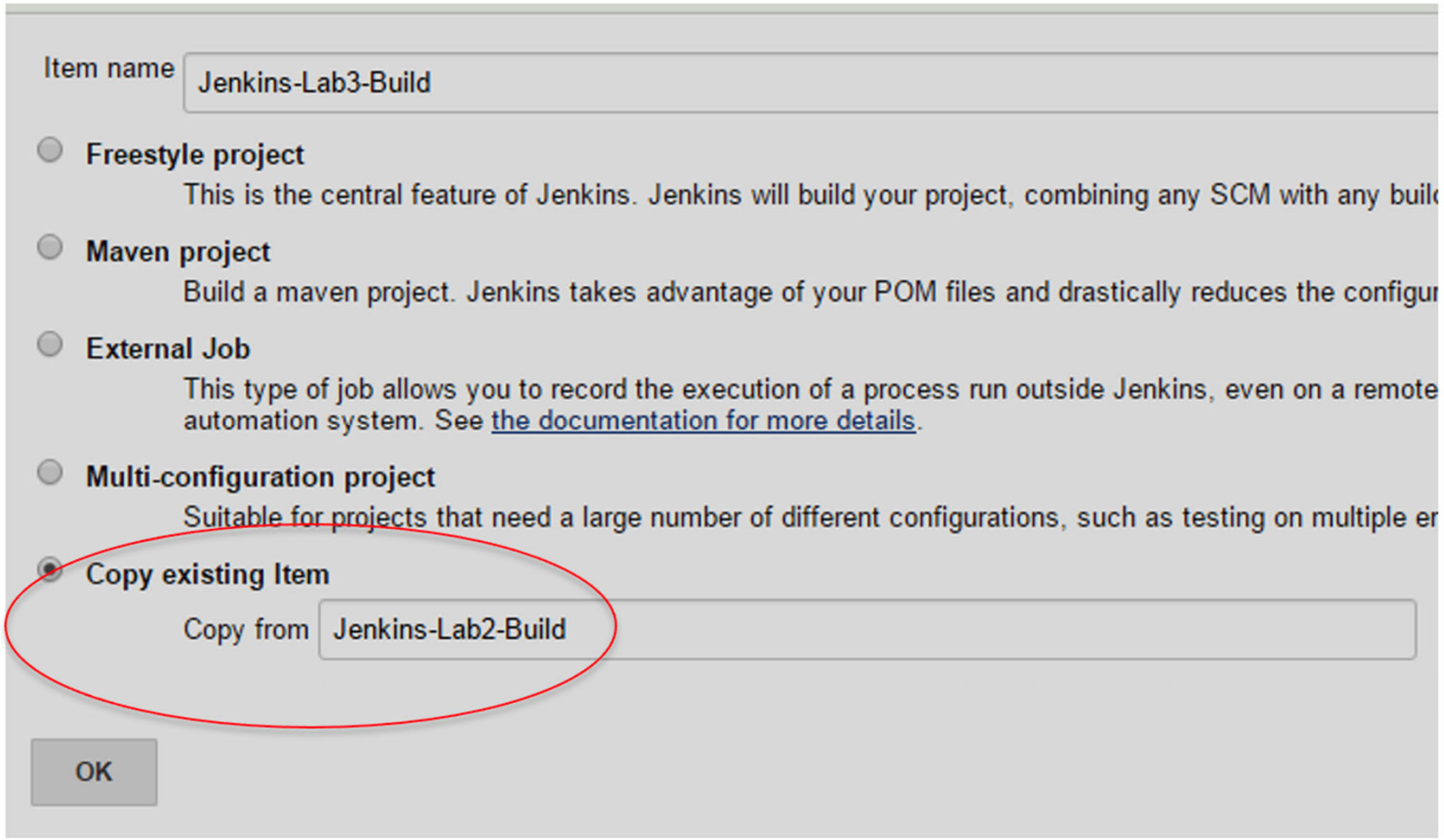


Configure Maven Job

- 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 a 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

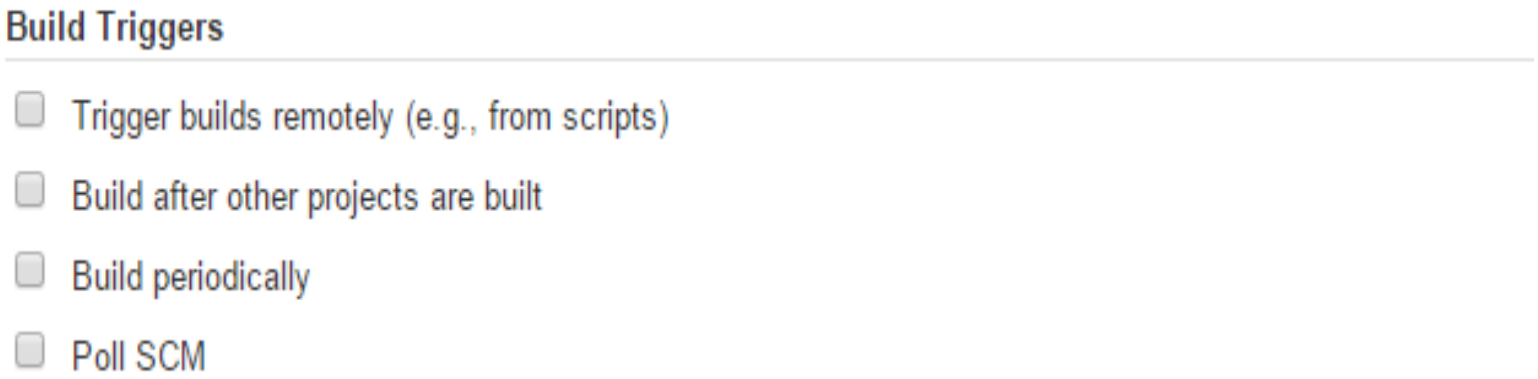


Topics in this Session

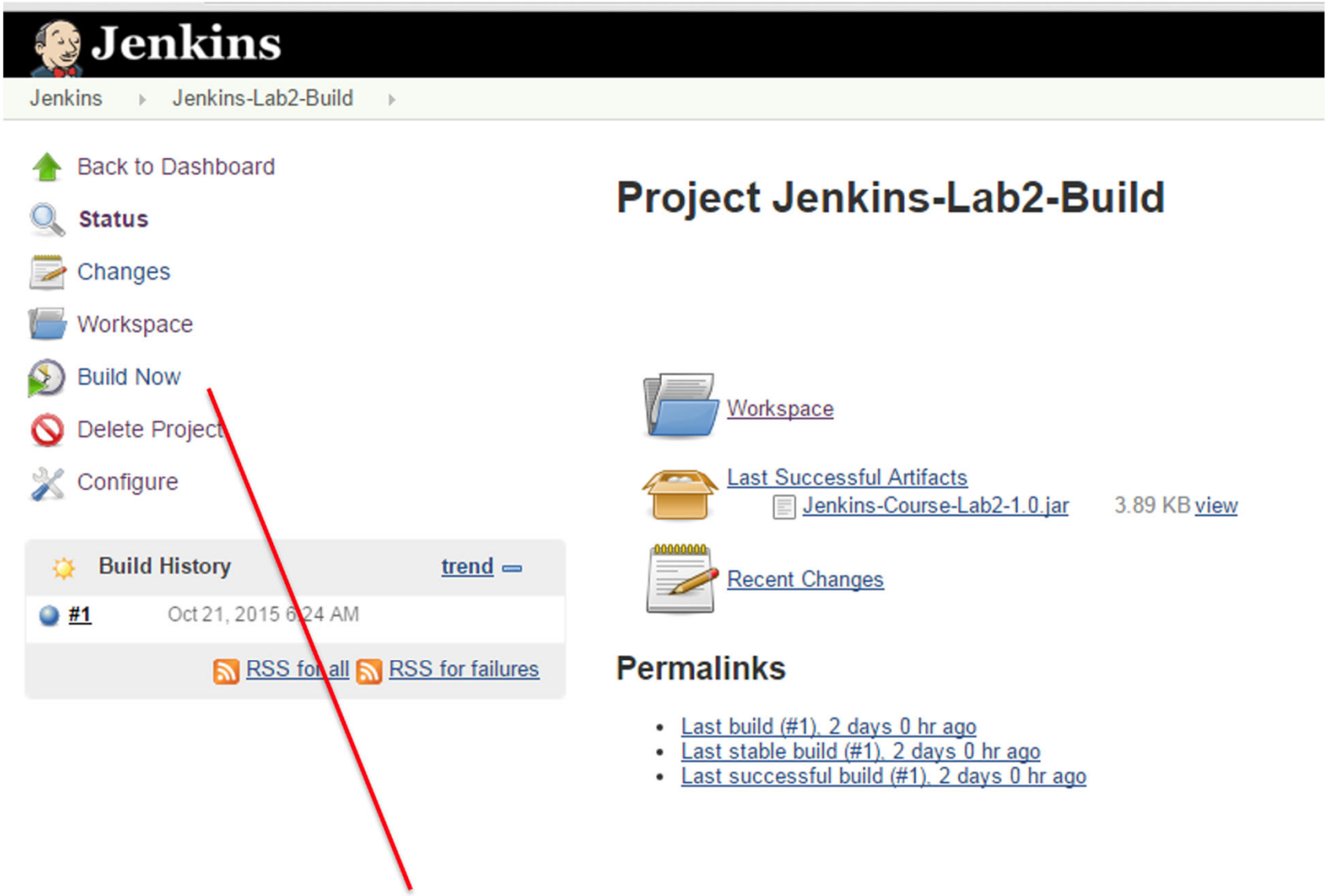
- Create a job
- Configure a Job
  - Configure a Freestyle Job
  - Configure a Maven Job
  - Copy from Existing Jobs
- **Trigger a Build Job**
  - Run a job manually
  - Run a job on a regular schedule
  - Run a job when source code is checked into version control
  - Run a Job After Another Job Finished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

Run Job Manually

- 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



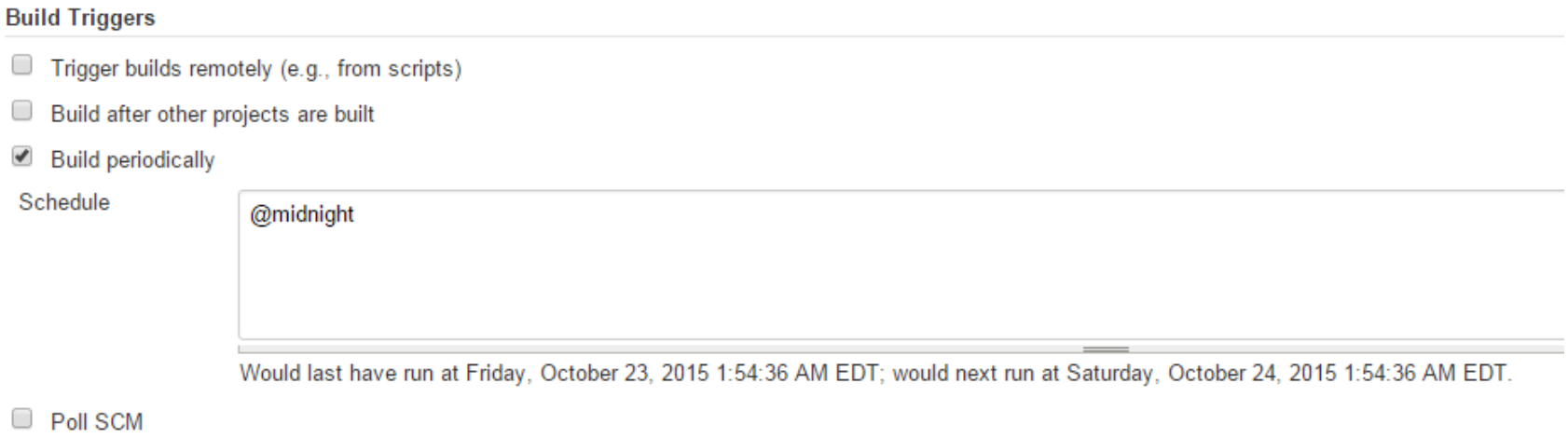
Build Now



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



Cron Style Syntax

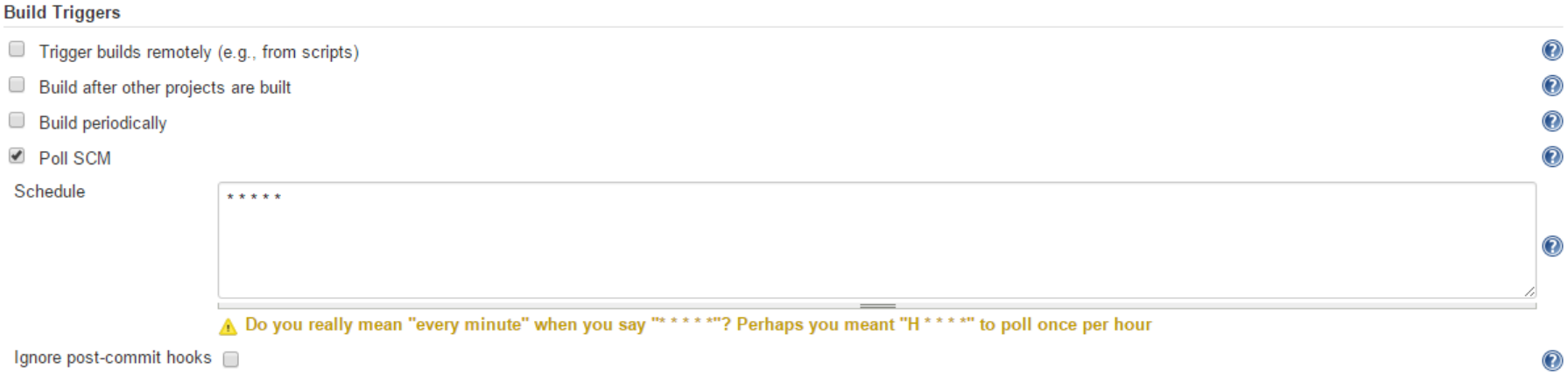
- You can use @yearly, @annually, @monthly, @weekly, @daily, @midnight, @hourly to schedule the jobs
- OR You can use cron style syntax in the following format
  - MINUTE HOUR DOM MONTH DOW
  - MINUTE – minutes within the hour (0-59)
  - HOUR – hour of the day (0-23)
  - DOM – day of the month (1-31)
  - 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 1st of every month
  - Note : Jenkins encourages the use of 'H' to balance load across different times of day to avoid sudden spikes of activities

Run a job when source code is checked into SCM

- Polling the SCM is the best strategy for continuous integration builds
  - Build your jobs as soon 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

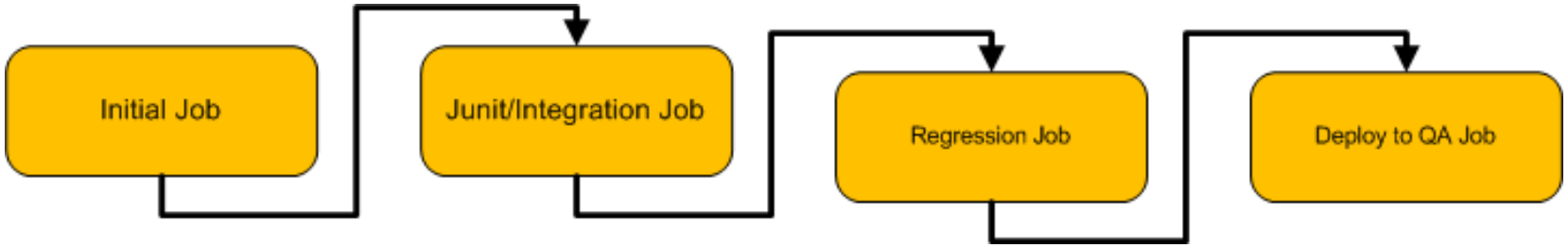
SCM Polling Interval

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



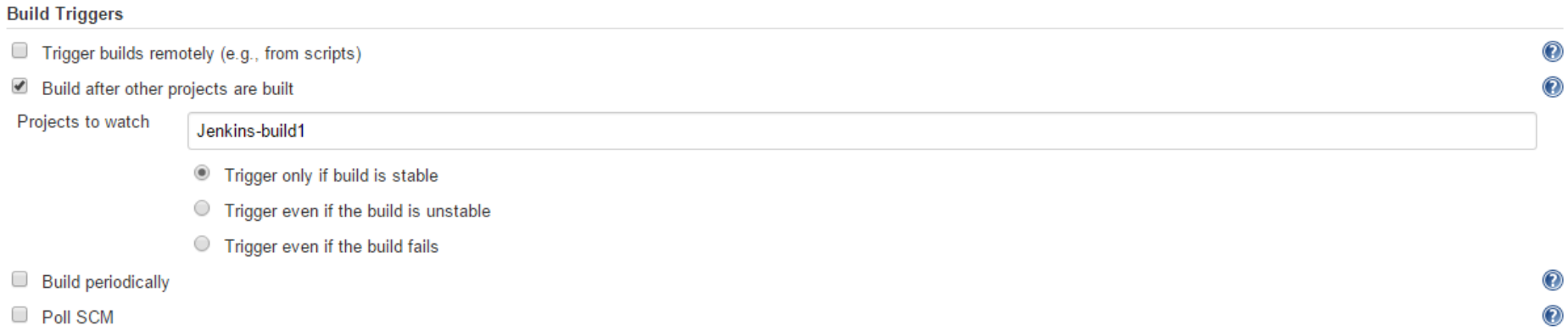
Run a Job After Another Job Finished

- 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



Run a Job After Another Job Finished - Configuration

- 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



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

Trivera Tech®
TECHNOLOGY TRAINING

Jenkins Plugins

Introducing Continuous Integration and Jenkins
Installing and Running Jenkins
Jenkins Job
Jenkins Plugins
SonarQube Plugin
Advanced Jenkins
Best Practices for Jenkins

Topics in this Session

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

Jenkins Plugins

- 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 1,000 plugins
  - Will review a few popular plugins especially the ones that are used for building quality code

Code Coverage

- 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 can find areas of code that have not been tested by automated tests
- Jenkins provides excellent support in generating code coverage metrics
  - Jenkins supports many of the 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

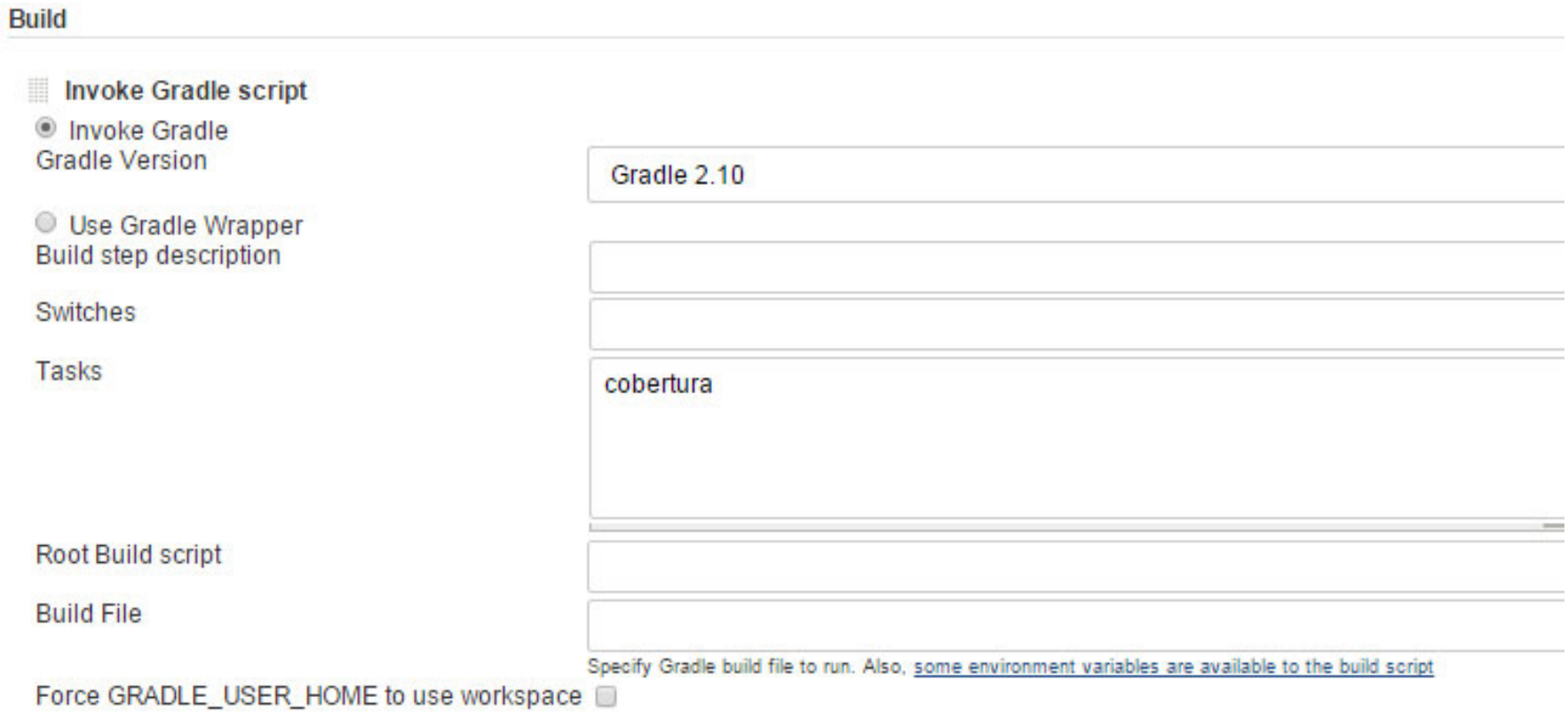
- First, install Cobertura plugin using Manage Plugins screen and restart Jenkins
- Apply Cobertura plugin in Gradle or Maven build script
  - Update your project's Maven POM file to include Cobertura plugin and then run `mvn cobertura:cobertura`
  - Update build.gradle file to apply Cobertura plugin 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']

 }
```

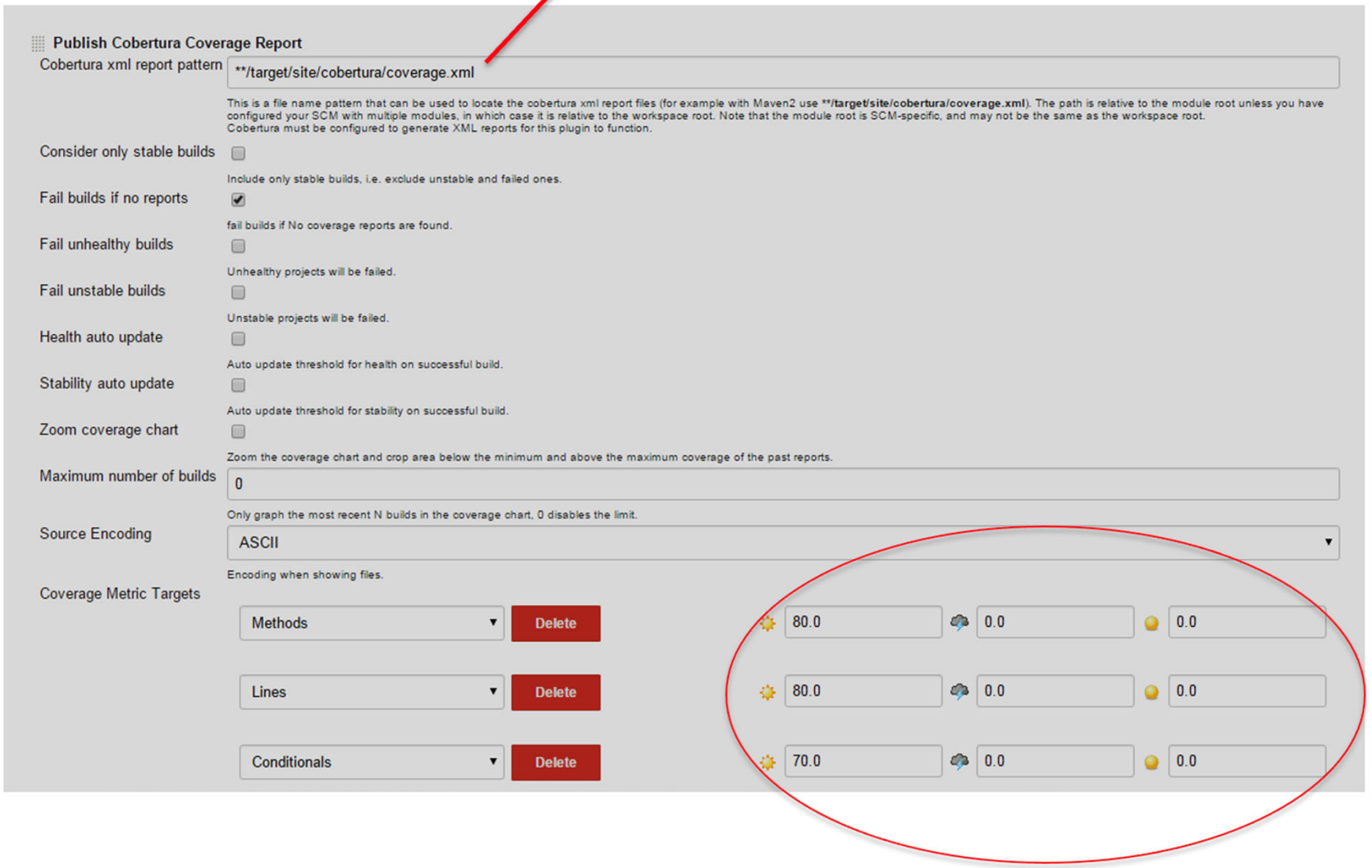
Configure Jenkins Job to generate Cobertura XML

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



Adding Cobertura in Post Build Actions

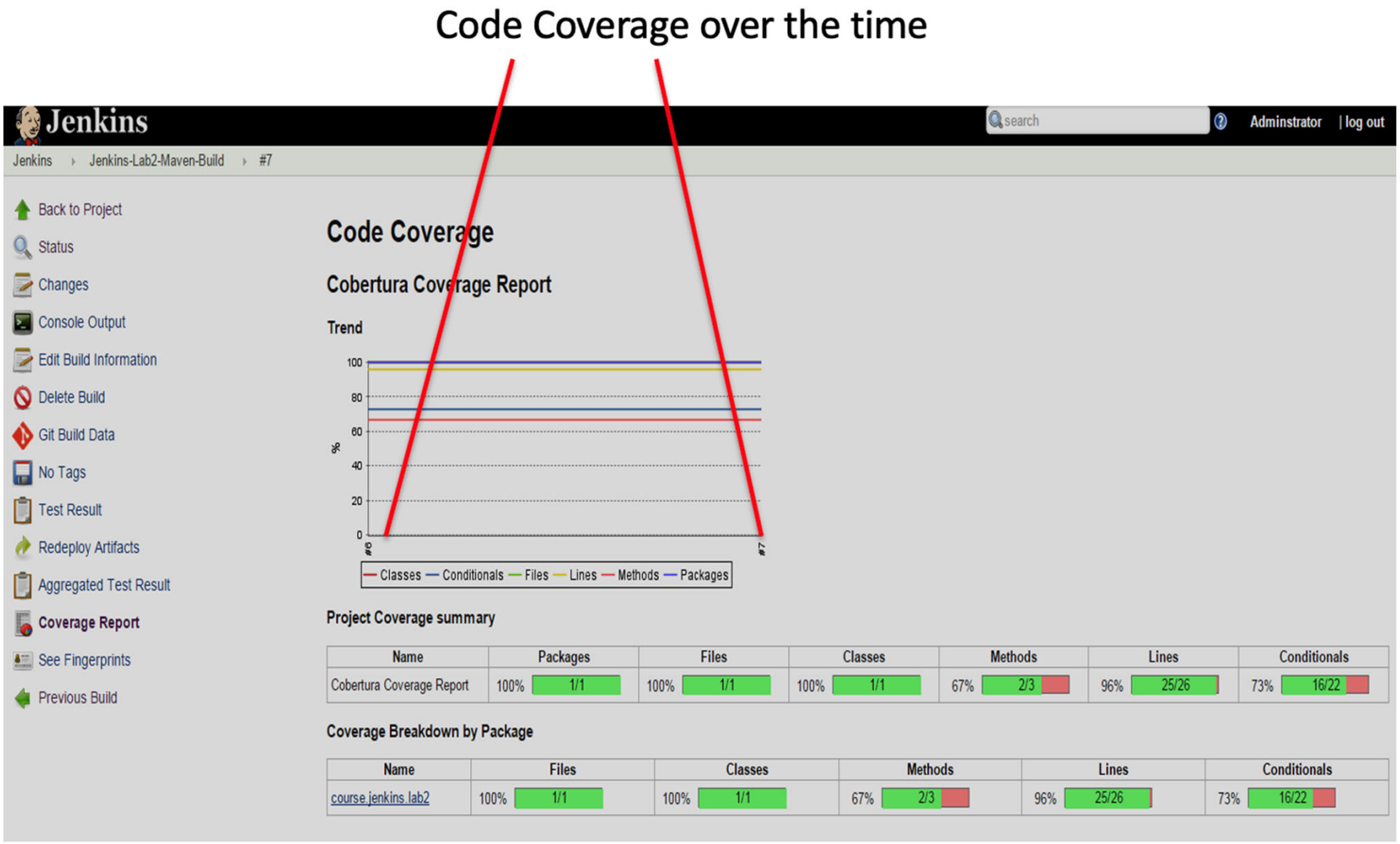
Specify path to coverage.xml file



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



Topics in this Session

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

Static Code Analysis

- 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 a 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

```
1 //required step
2
3 apply plugin: 'findbugs'
```

FindBugs – Optional Configuration

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

```
1  //Optional configuration
2
3    findbugs
4
5  {
6
7      toolVersion = "2.0.1"
8
9      ignoreFailures = true
10
11     effort = "max"
12
13   reportLevel = "low"
14
15  }
```

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

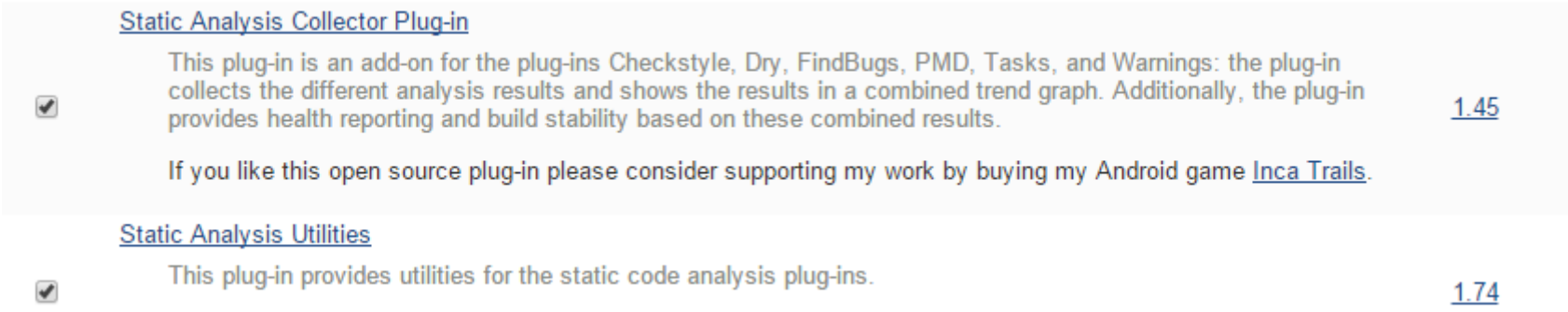
```
1 apply plugin: 'checkstyle'
2
3   checkstyle {
4
5     ignoreFailures = true
6
7   }
```

Configuring PMD in build.gradle

```
1  //Only required step to enable PMD
2
3    apply plugin: 'pmd'
4
5    /*
6
7    Other properties and ruleSets can be set under pmd task
8
9    */
10
11   pmd {
12
13   ignoreFailures = true
14
15  ruleSets = ["java-basic", "java-braces"]
16
17  }
```

Install Static Code Analysis Plugins in Jenkins

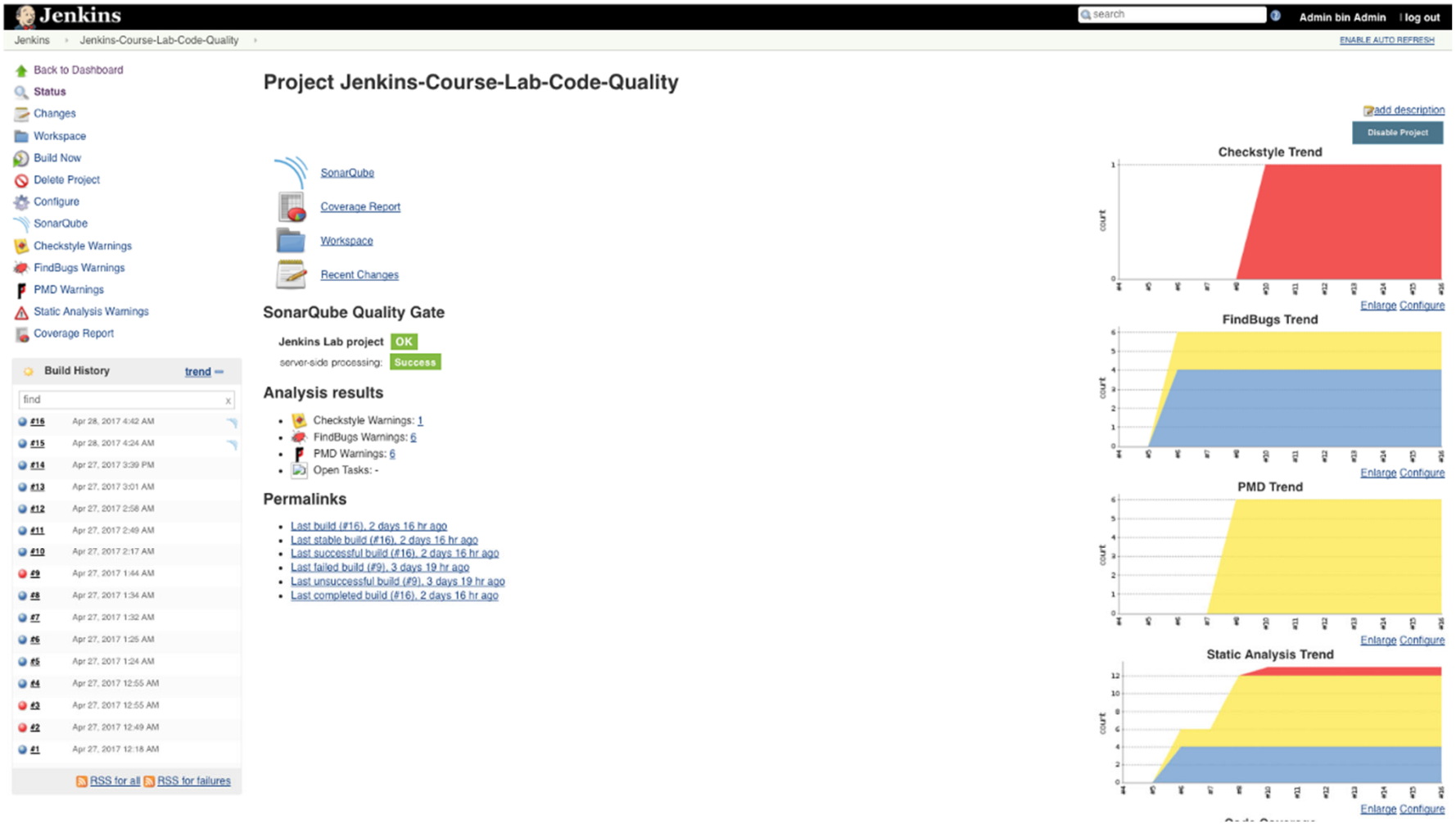
- Install static code analysis plugins through plugin manager
  - The plugins related static code analysis is available under Build Reports category
  - You can install Findbugs, PMD and Checkstyle plugins individually
  - Then you can also install Analysis Collector Plugin and Static Analysis Utilities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



Generating Static Code Analysis in Jenkins build

- Gradle should include the following tasks to generate static code analysis reports as part of Jenkins build section of a job:
  - You can 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



Topics in this Session

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

Performance Reporting

- 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

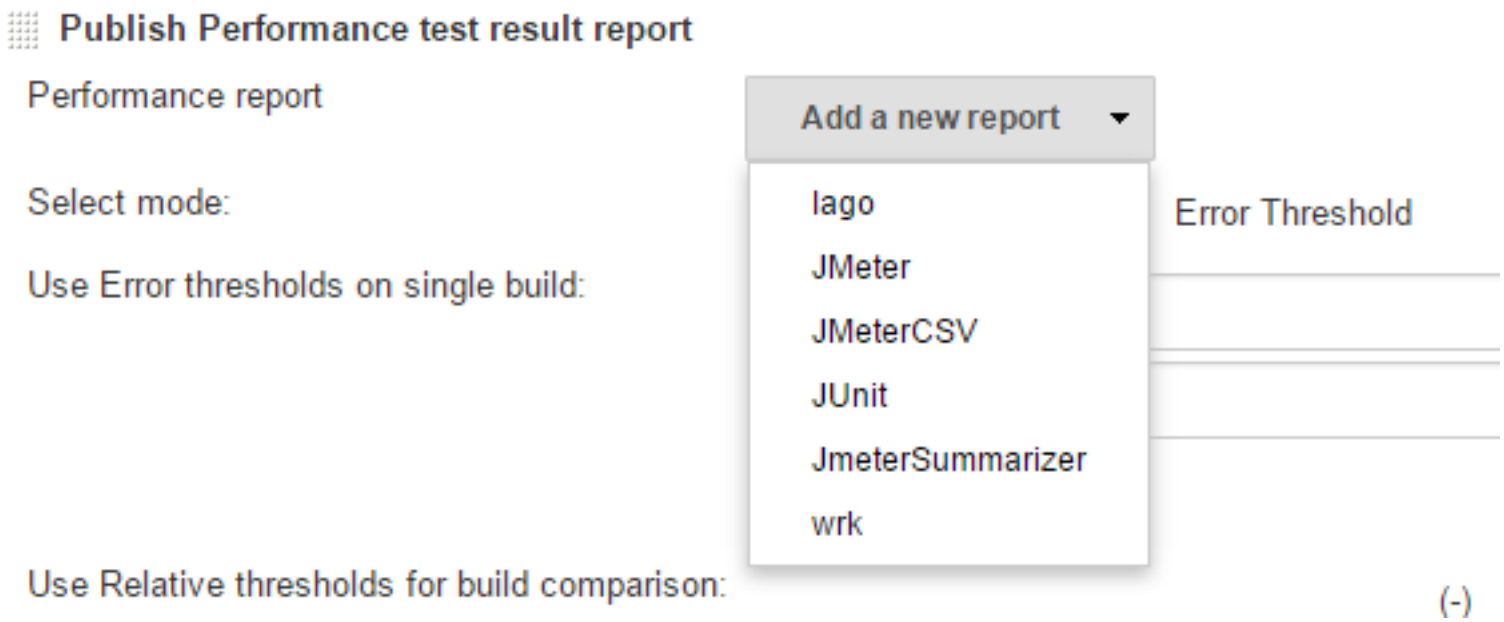
- The Gradle JMeter plugin is a quick way to integrate JMeter into the build.gradle

```
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



Topics in this Session

- 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** keeps 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 a live job status monitor
- **Workspace Cleanup Plugin** deletes your workspace before or after builds
- **Disk-usage Plugin** keeps track of your disk space usage

Topics in this Session

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

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

Trivera Tech®
TECHNOLOGY TRAINING

SonarQube Plugin

Introducing Continuous Integration and Jenkins
Installing and Running Jenkins
Jenkins Job
Jenkins Plugins
SonarQube Plugin
Advanced Jenkins
Best Practices for Jenkins

Topics in this Session

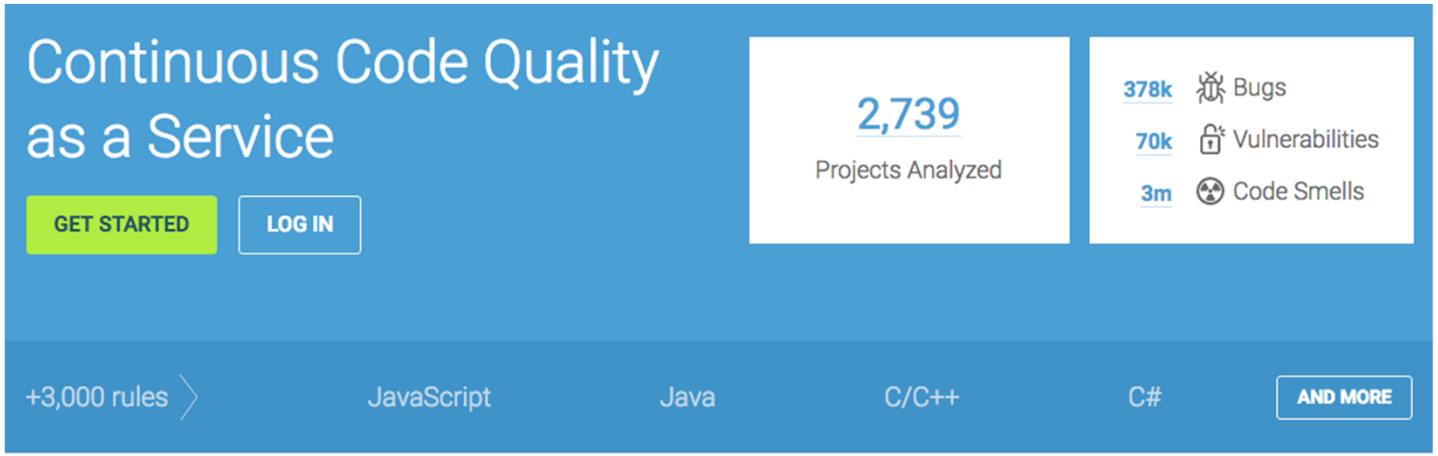
- **Install SonarQube**
- Integrate Jenkins with SonarQube
- Locate and Open Generated Report
- Review the Report's Organization
- Lab 4 – Generate first SonarQube report using Jenkins

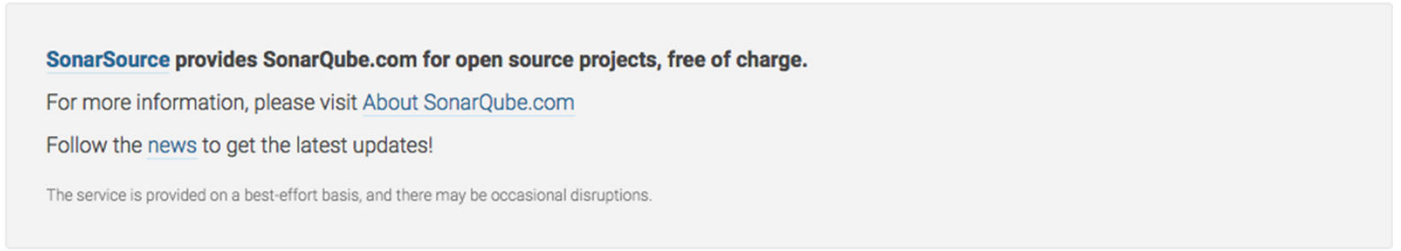
Introducing SonarQube

- SonarQube is a free and open source **code quality platform**
- More than 50 plugins are available to extend SonarQube with CI and development tools
- It is not just for Java: More than 20 programming languages are 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 easy to install and configure
- Follow our labs instructions





Configure 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

Topics in this Session

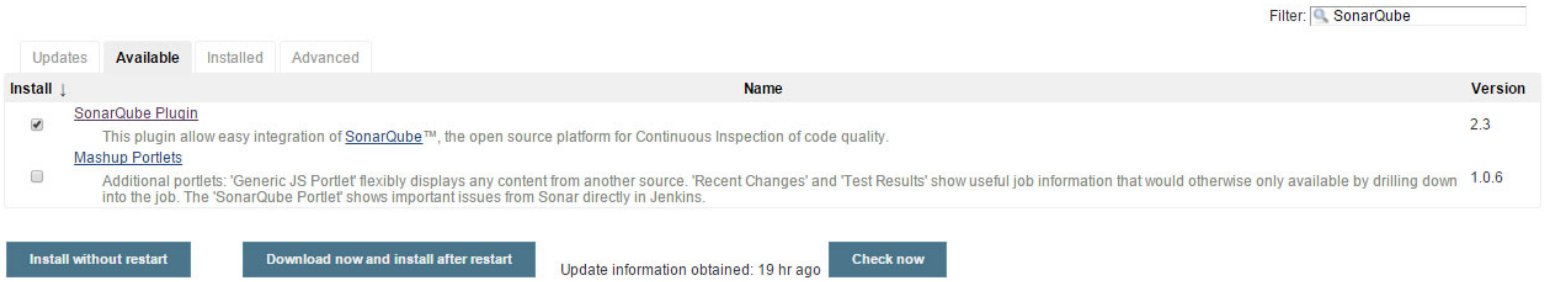
- Install SonarQube
- **Integrate Jenkins with SonarQube**
- Locate and Open Generated Report
- Review the Report's Organization
- Lab 4 – Generate first SonarQube report using Jenkins

Integrate SonarQube with Jenkins

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

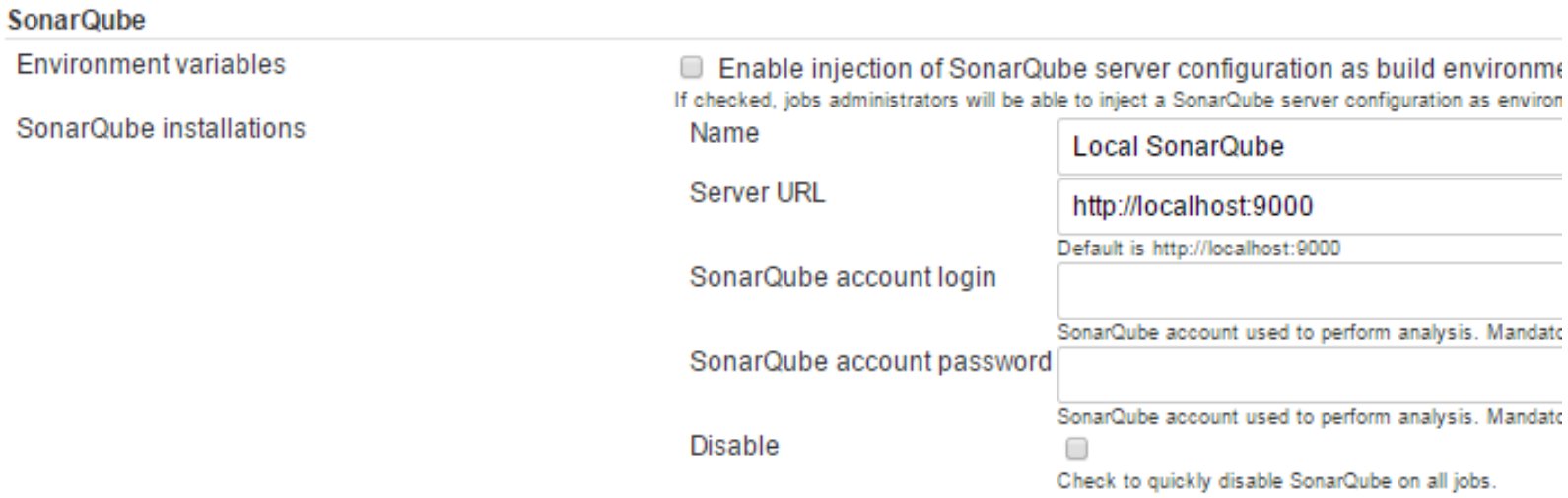
Install SonarQube Jenkins Plugin

- 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



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

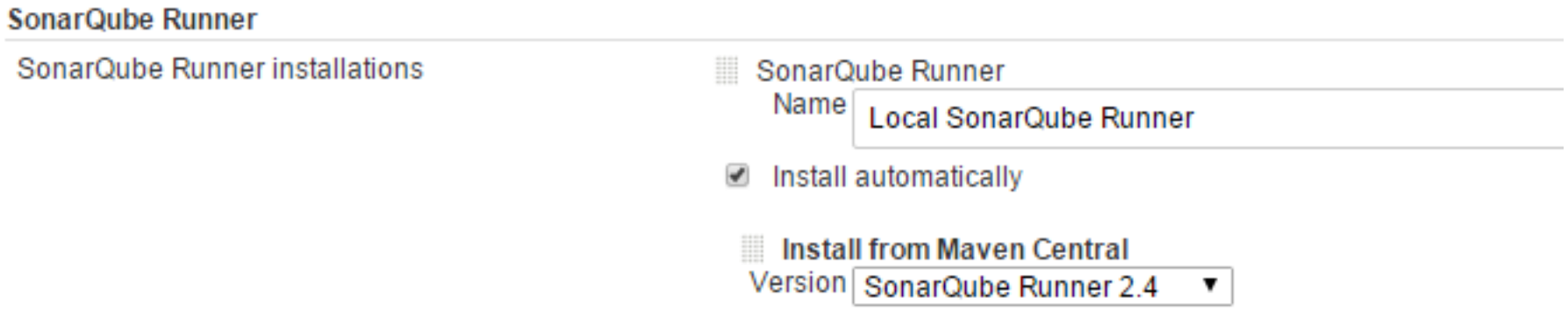


SonarQube Scanner for Jenkins

- SonarQube Scanner is the preferred option to analyze a project with SonarQube for non-Maven based projects
- Requires a 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

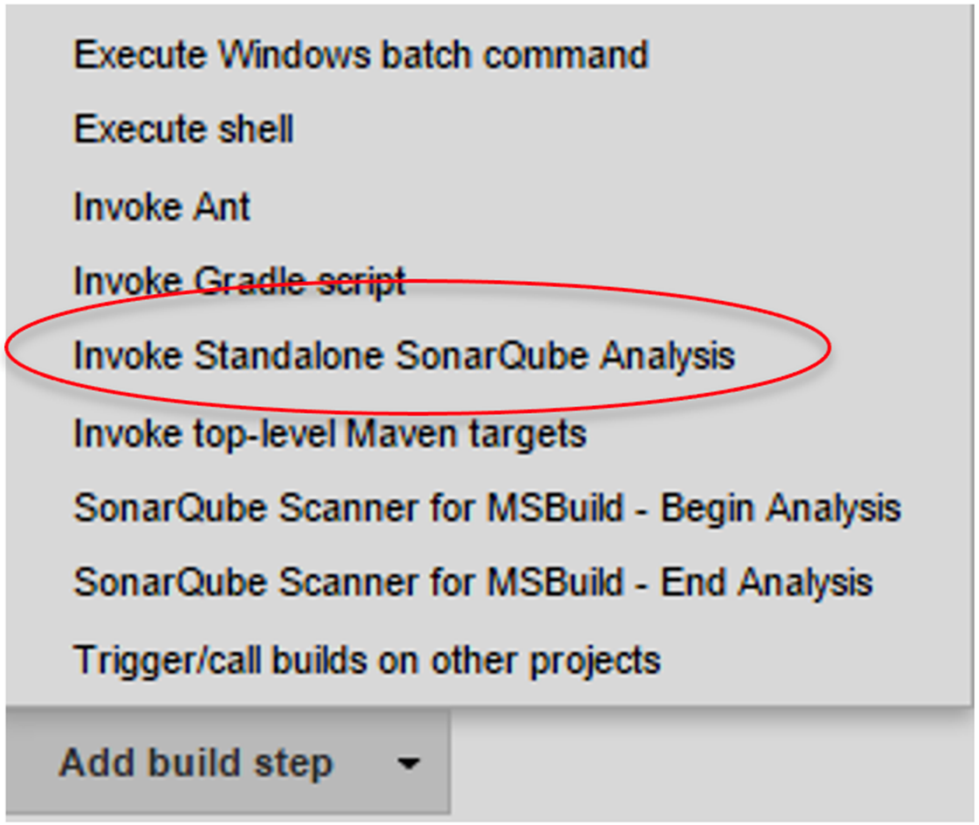


Enabling SonarQube analysis in a build job

- 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

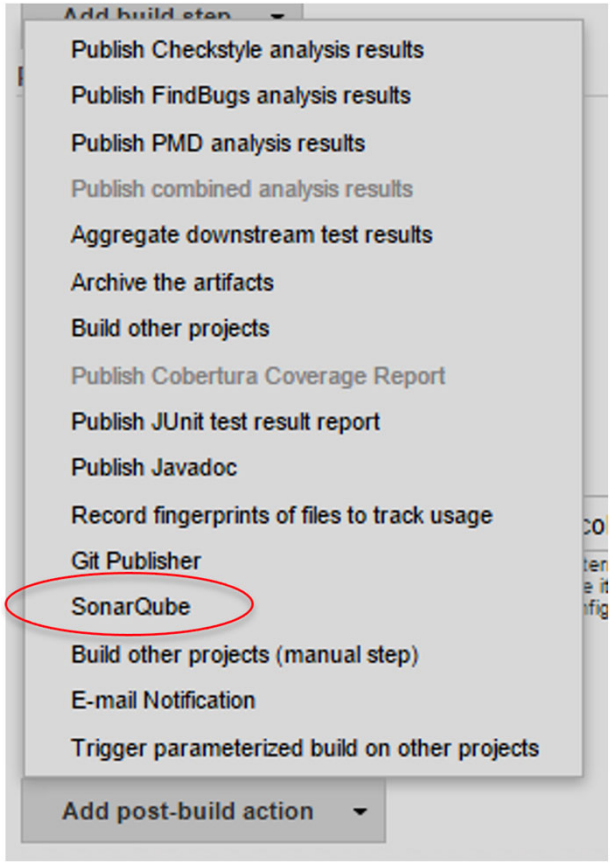
Build step to trigger the SonarQube analysis

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



- 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"

        }

    }
```

Topics in this Session

- 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

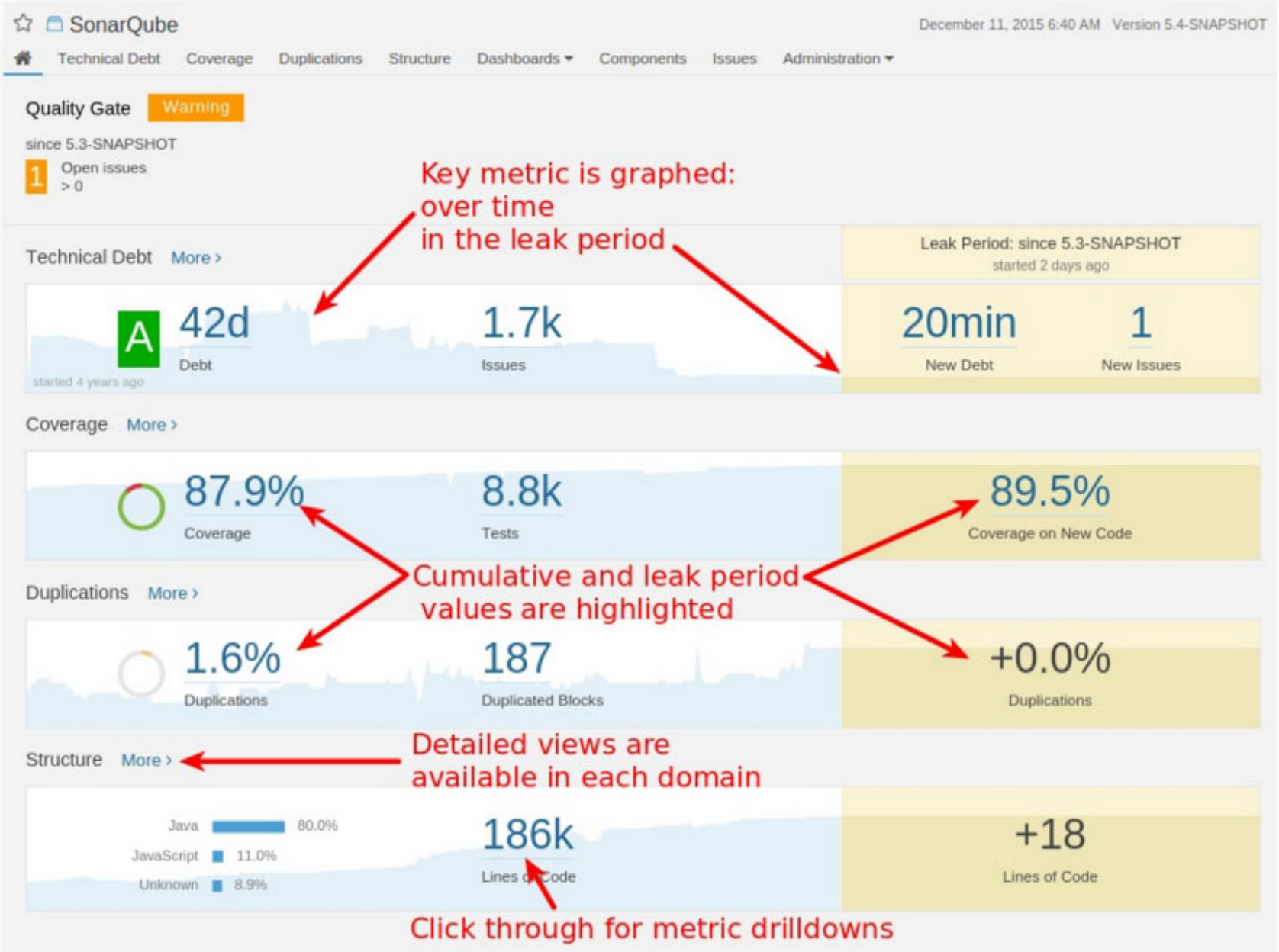
Topics in this Session

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

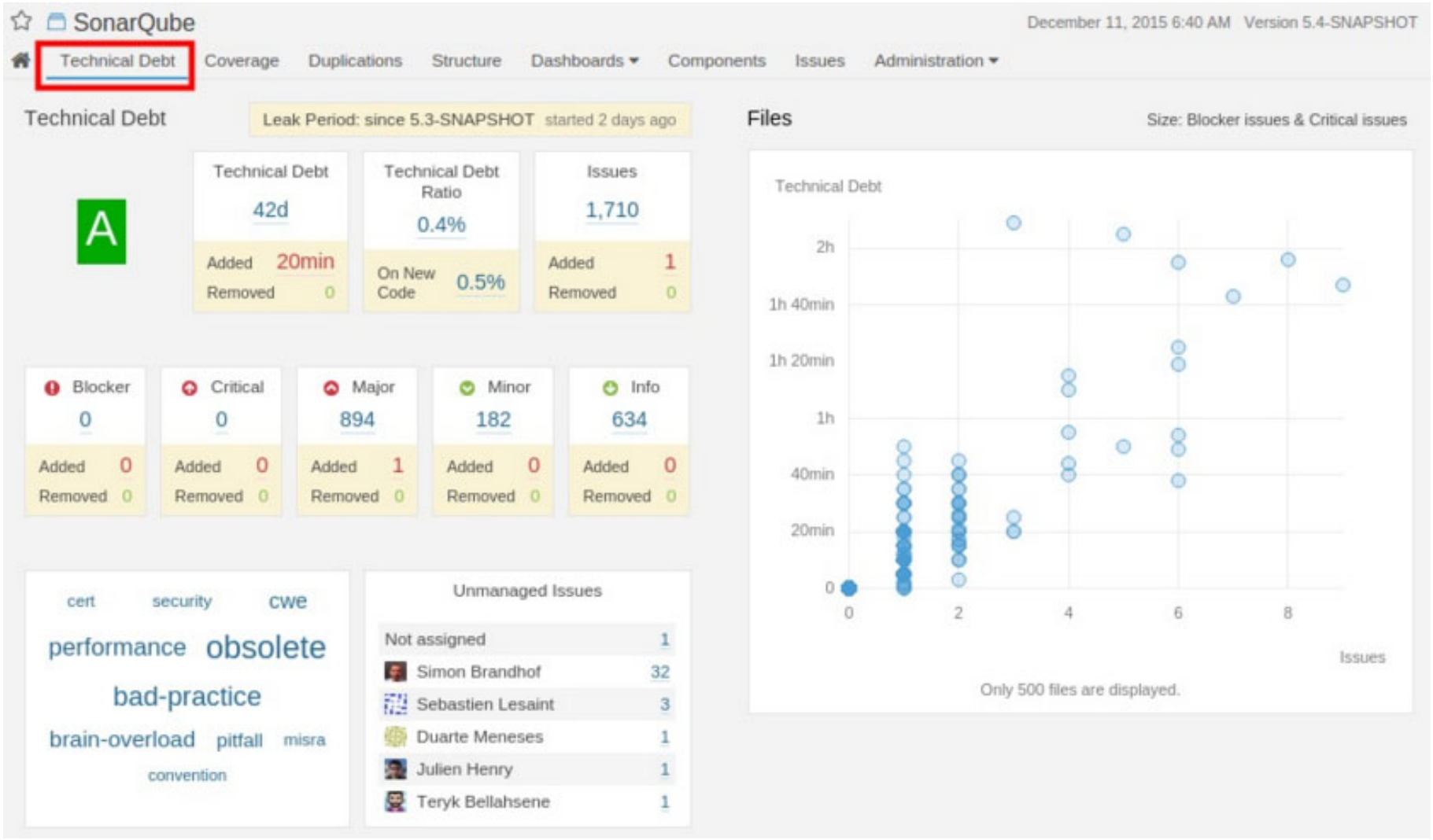
Review the Report's Organization

- The complete Sonar report is organized in four 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

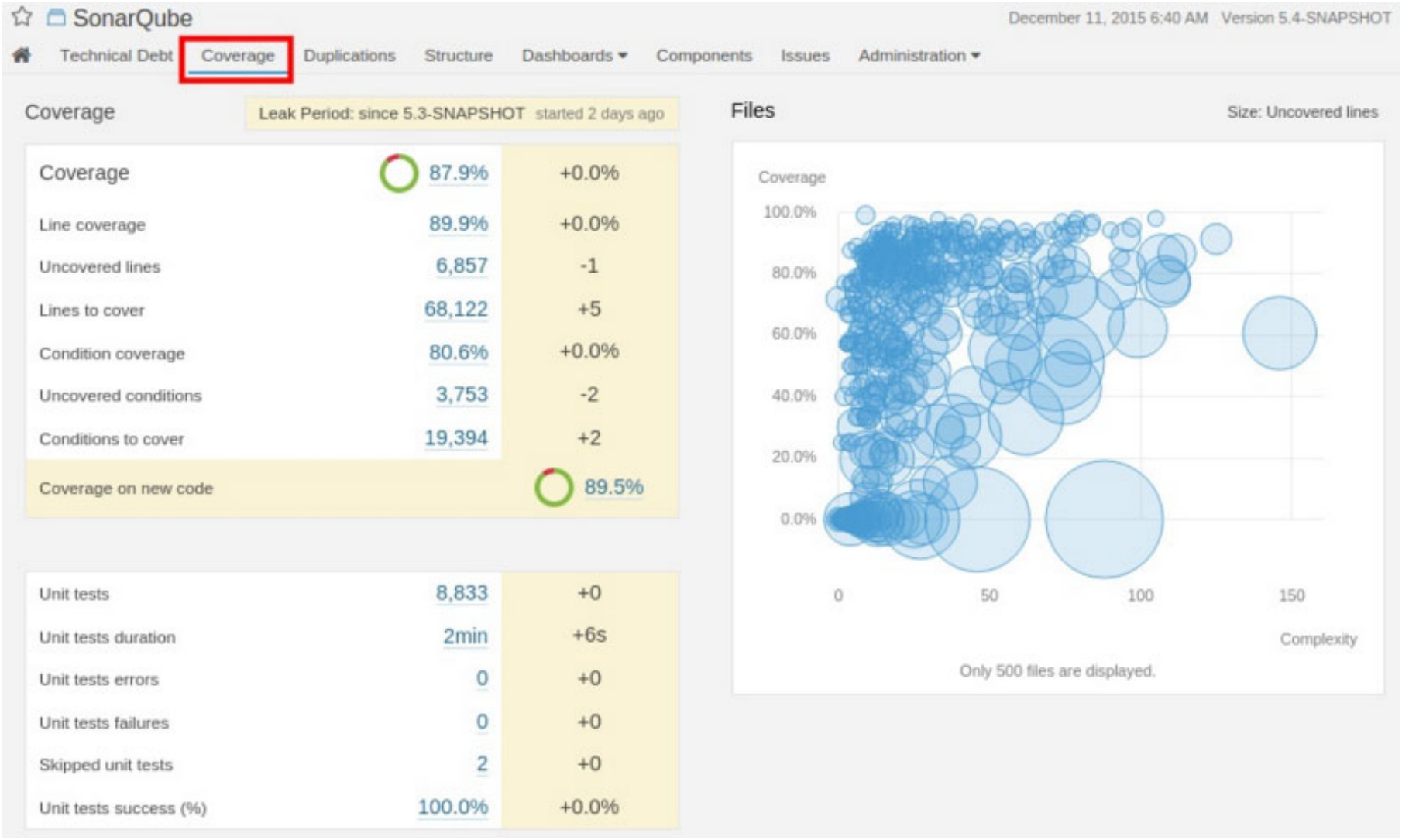
Sonar Report – Project Home Page



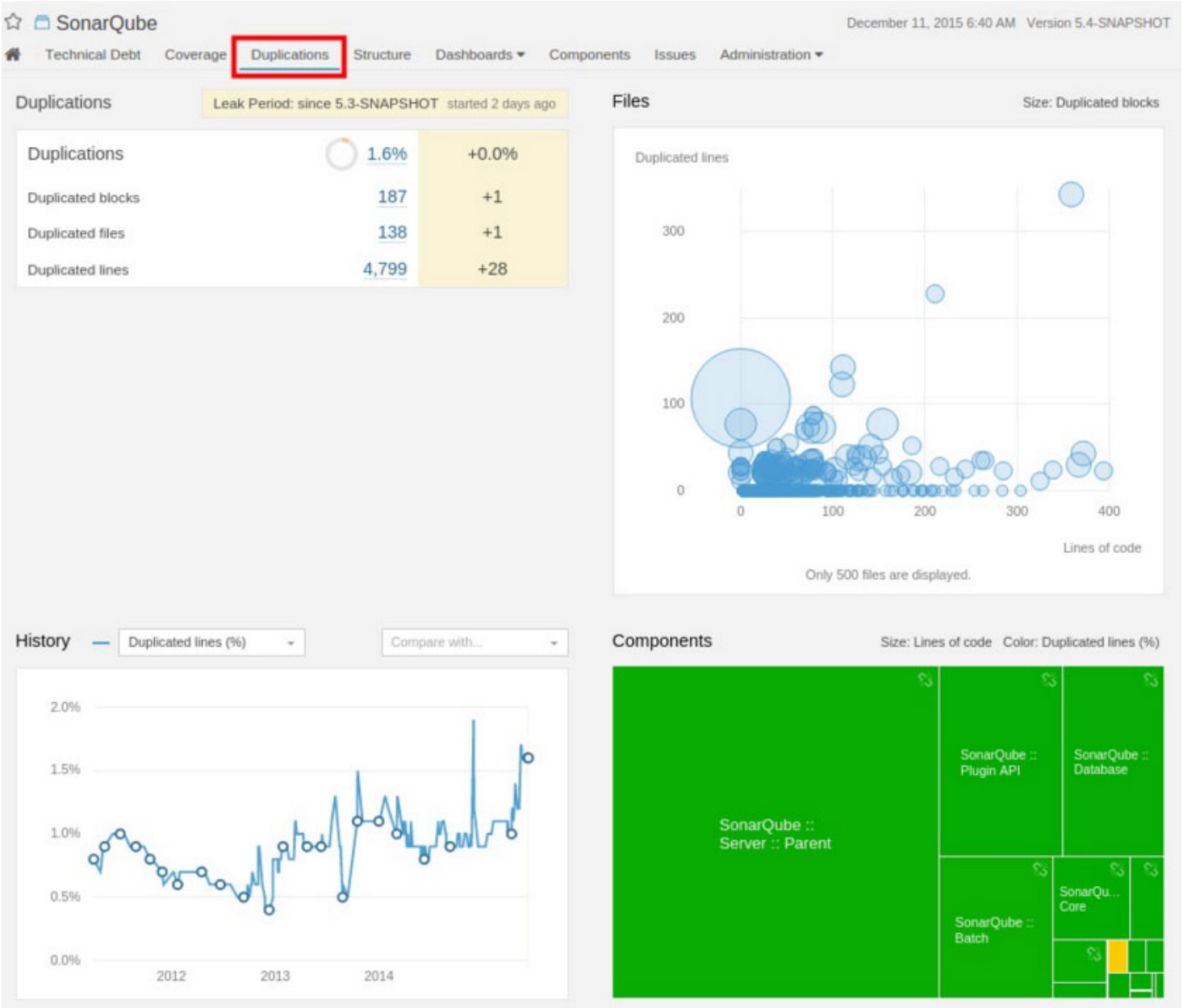
Sonar Report - Technical Debt and Issues



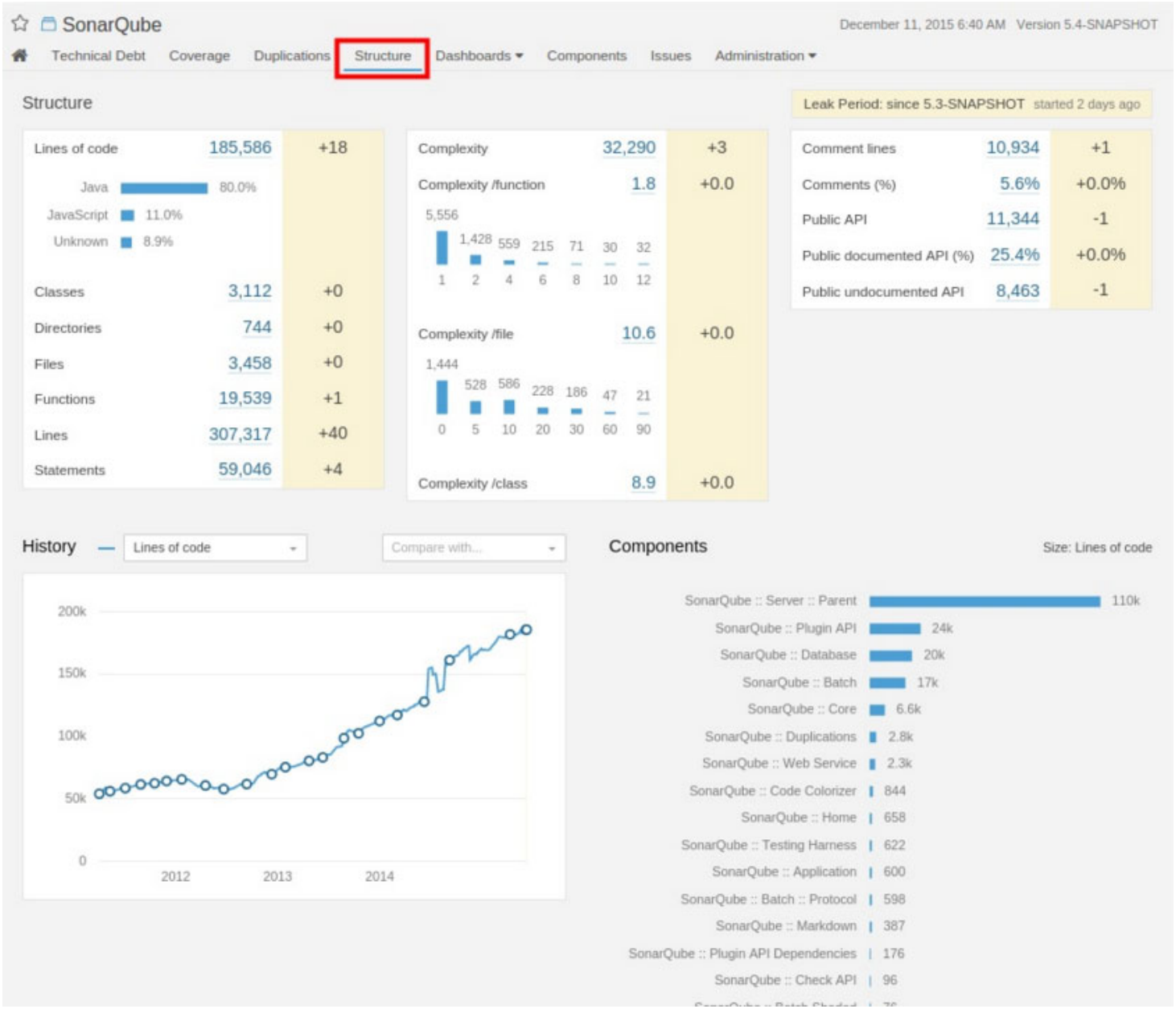
Sonar Report – Code Coverage



Sonar Report - Duplications



Sonar Report – Structure and Complexity



Topics in this Session

- 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

Trivera Tech®
TECHNOLOGY TRAINING

Advanced Jenkins

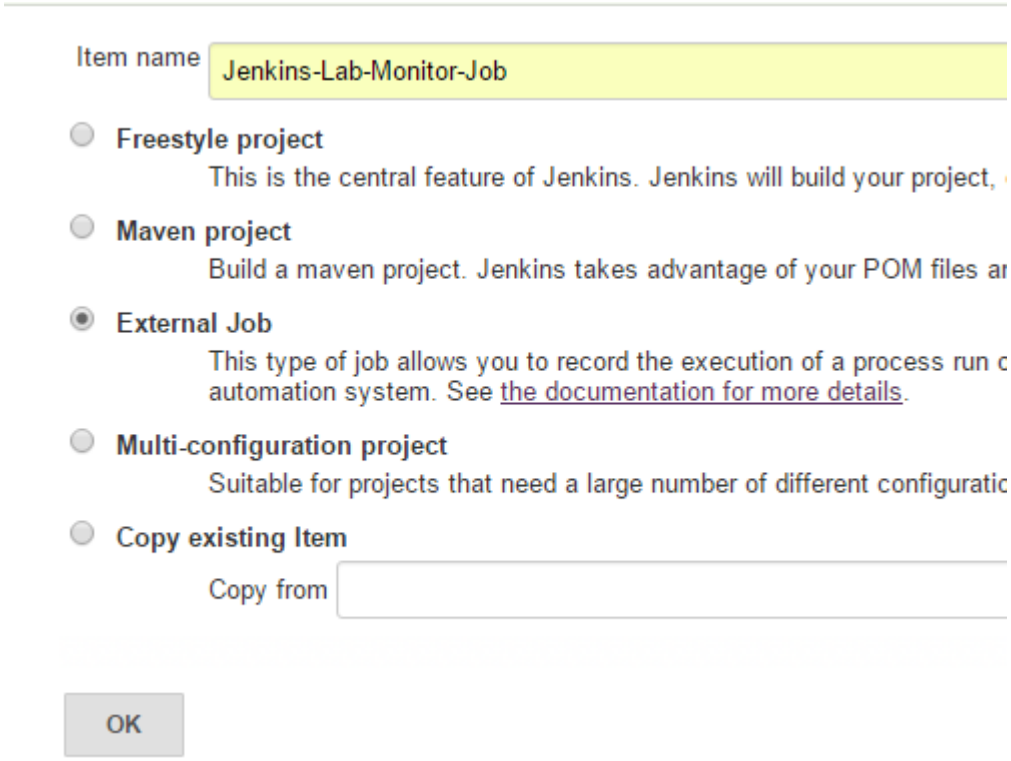
Introducing Continuous Integration and Jenkins
Installing and Running Jenkins
Jenkins Job
Jenkins Plugins
SonarQube Plugin
Advanced Jenkins
Best Practices for Jenkins

Topics in this Session

- **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

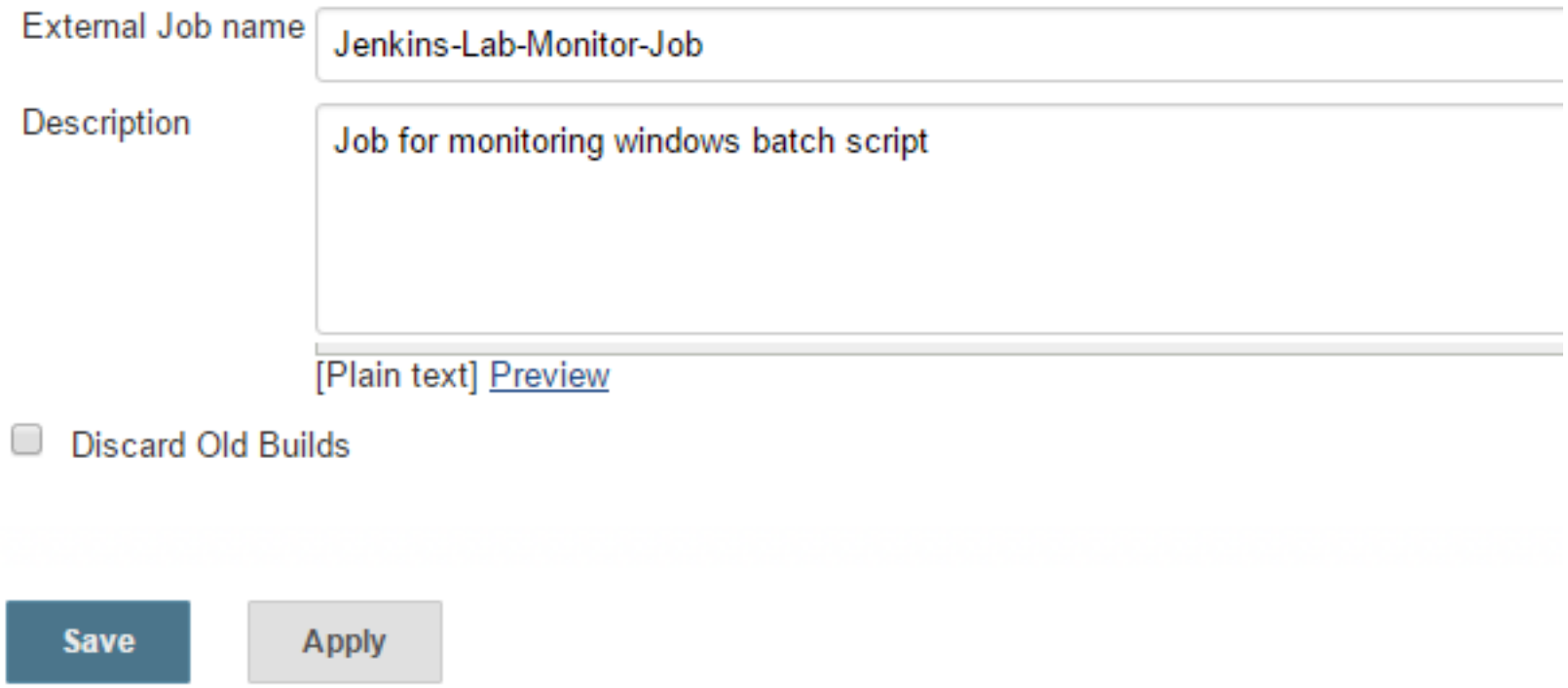
Monitor External Jobs

- 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



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

```
jenkins-core-*.jar </br>
    remoting-*.jar </br>
    ant-*.jar </br>
    commons-io-*.jar </br>
    commons-lang-*.jar </br>
    jna-posix-*.jar </br>
    xstream-*.jar
```

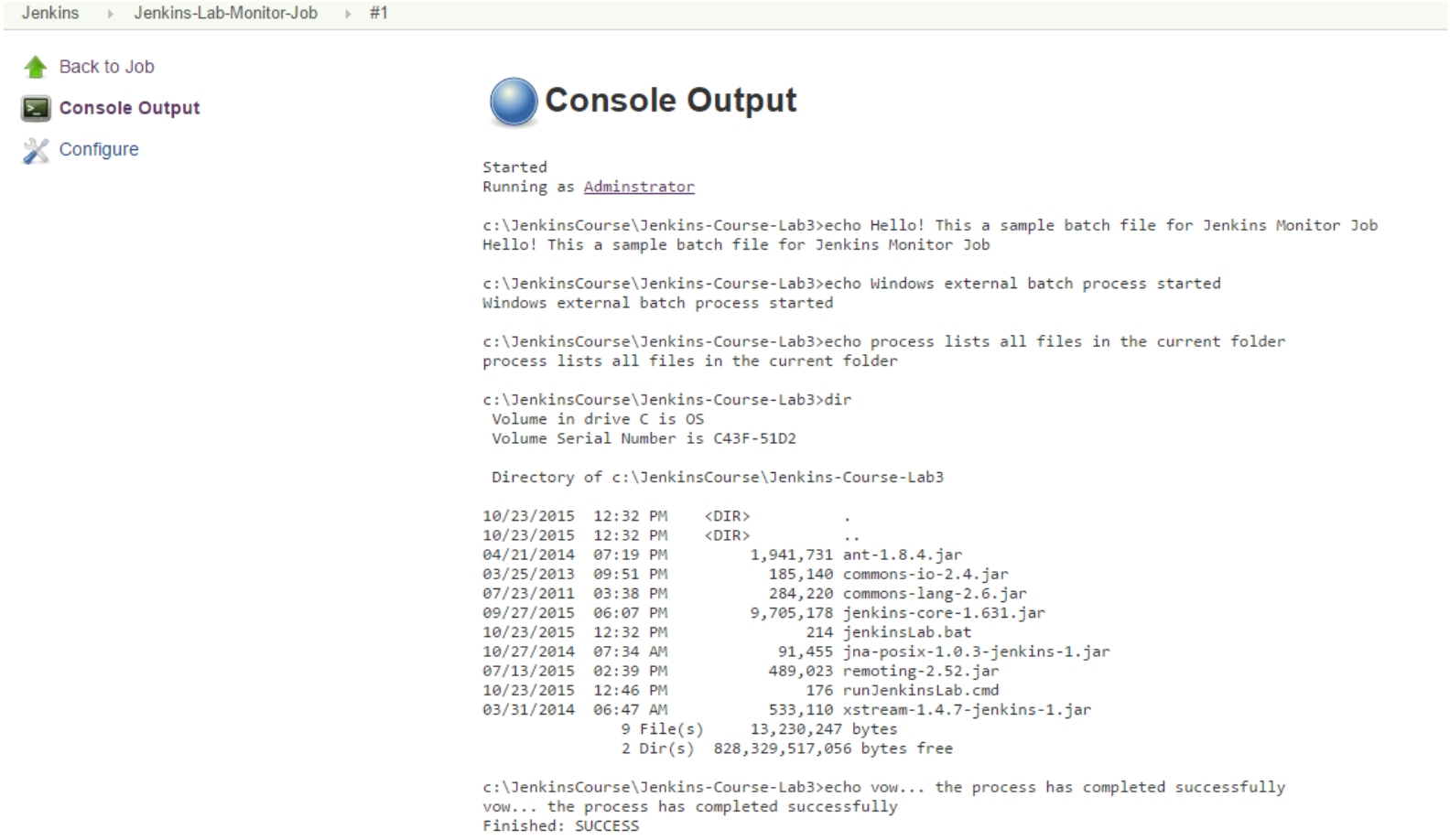
How to Invoke Jenkins External Job

- 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

```
1 set JENKINS_HOME=http://admin:admin@localhost:8080/
2
3  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



Topics in this Session

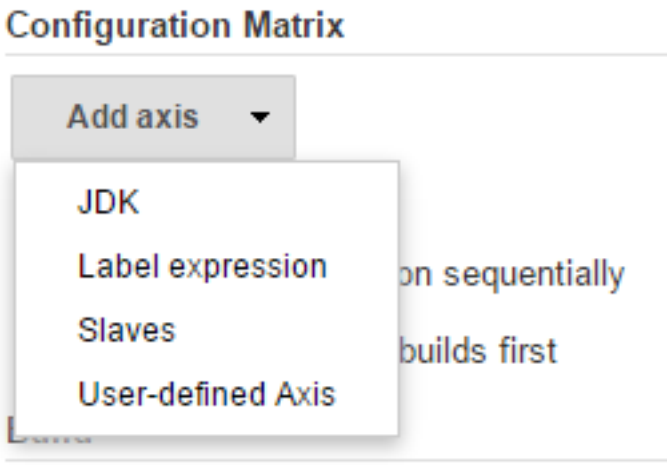
- Monitor External Jobs
- **Matrix or Multi-Configuration Jobs**
- Distributed Builds
- Concept of a Pipeline
- Splitting a Big Job into Smaller Jobs
- File Fingerprint Tracking
- Using Jenkins for non-Java Projects
- Lab 5: Configure distributed builds using Master/Slave configuration
- Lab 6: Build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

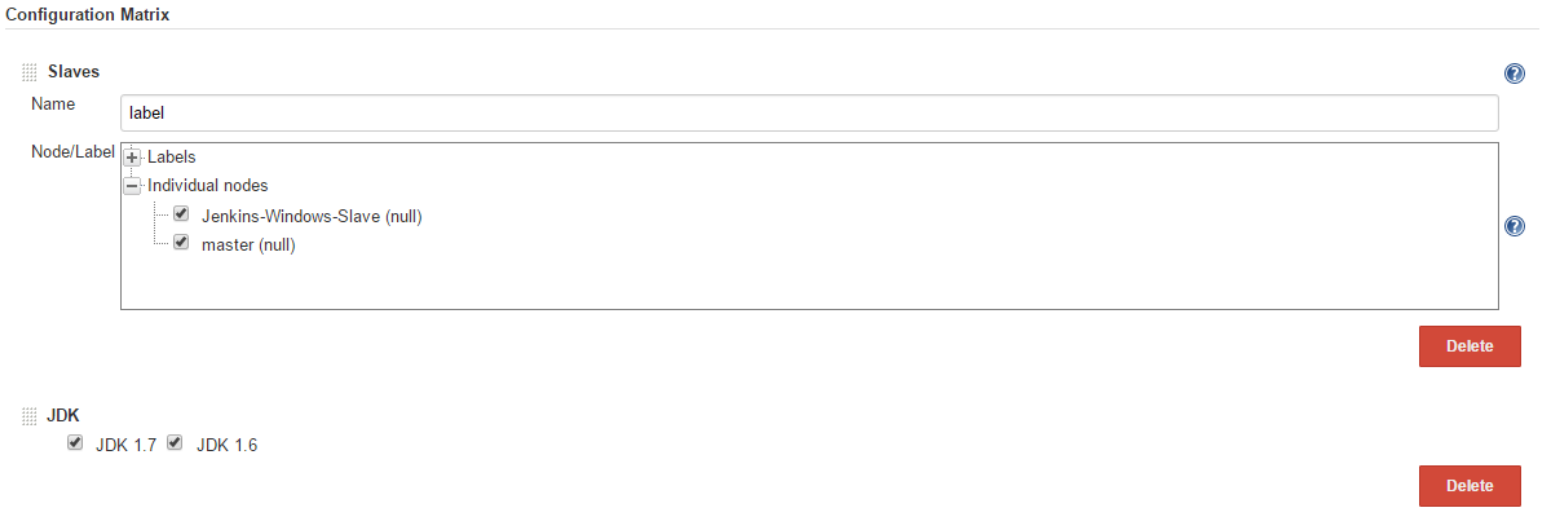
Multi-Configuration Job Configuration

- 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



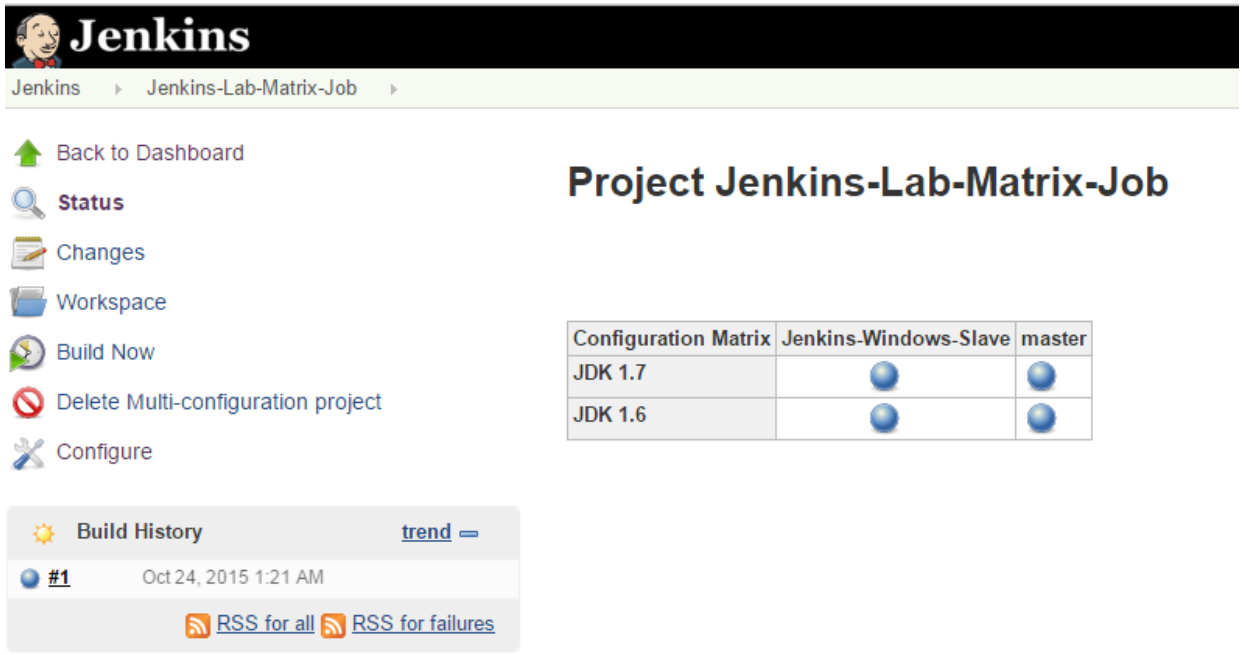
Multi-Configuration Matrix

- 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
    - JKD 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



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



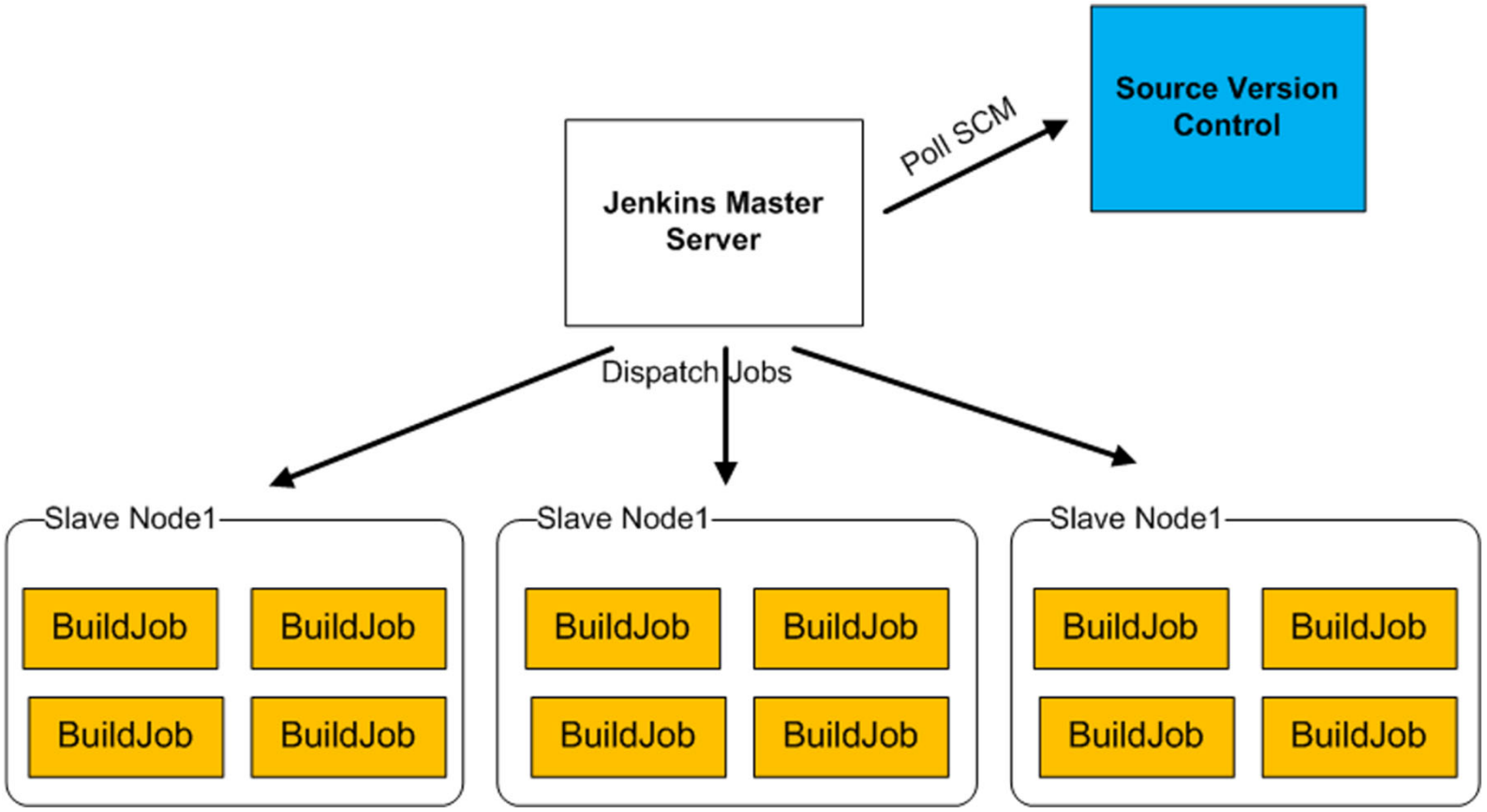
Topics in this Session

- 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

- 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

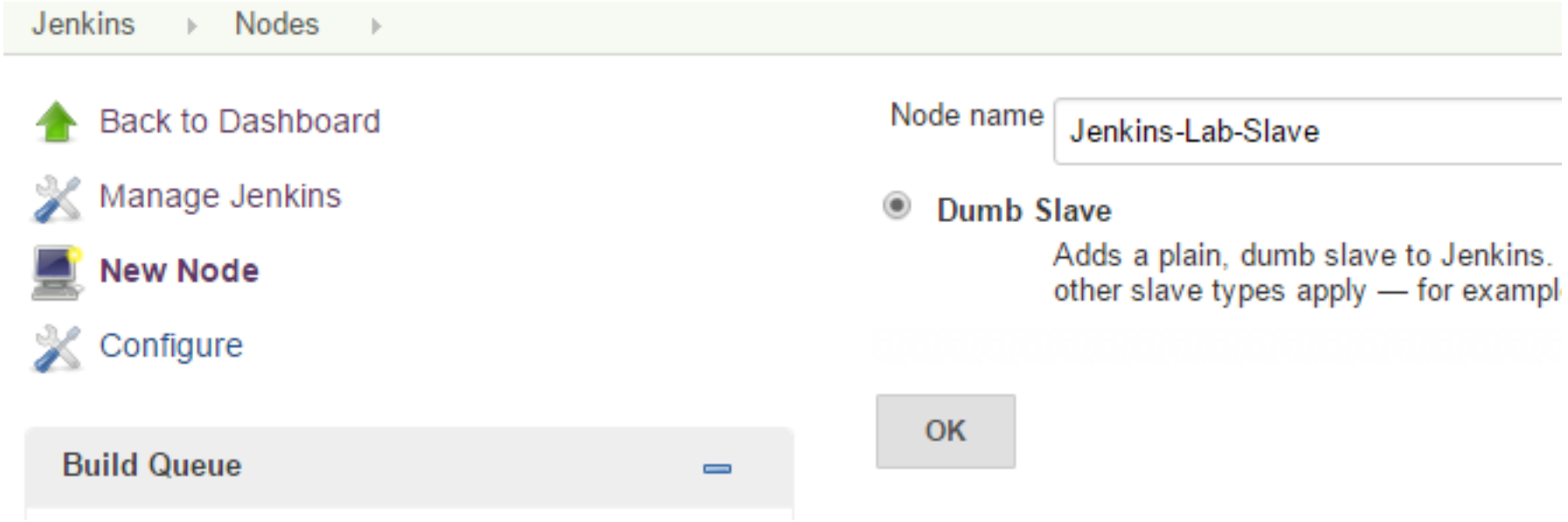


Setup Master Node

- 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

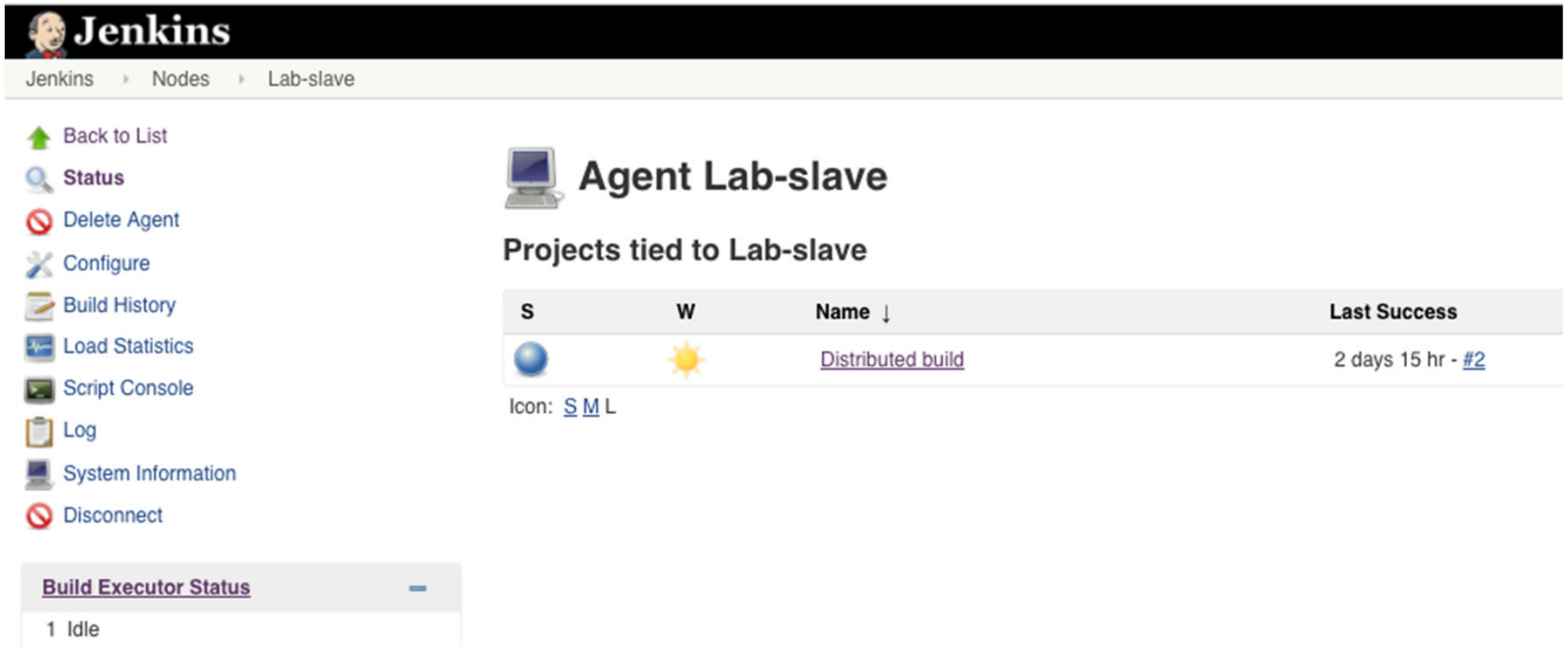


Launching Slave Nodes

- 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

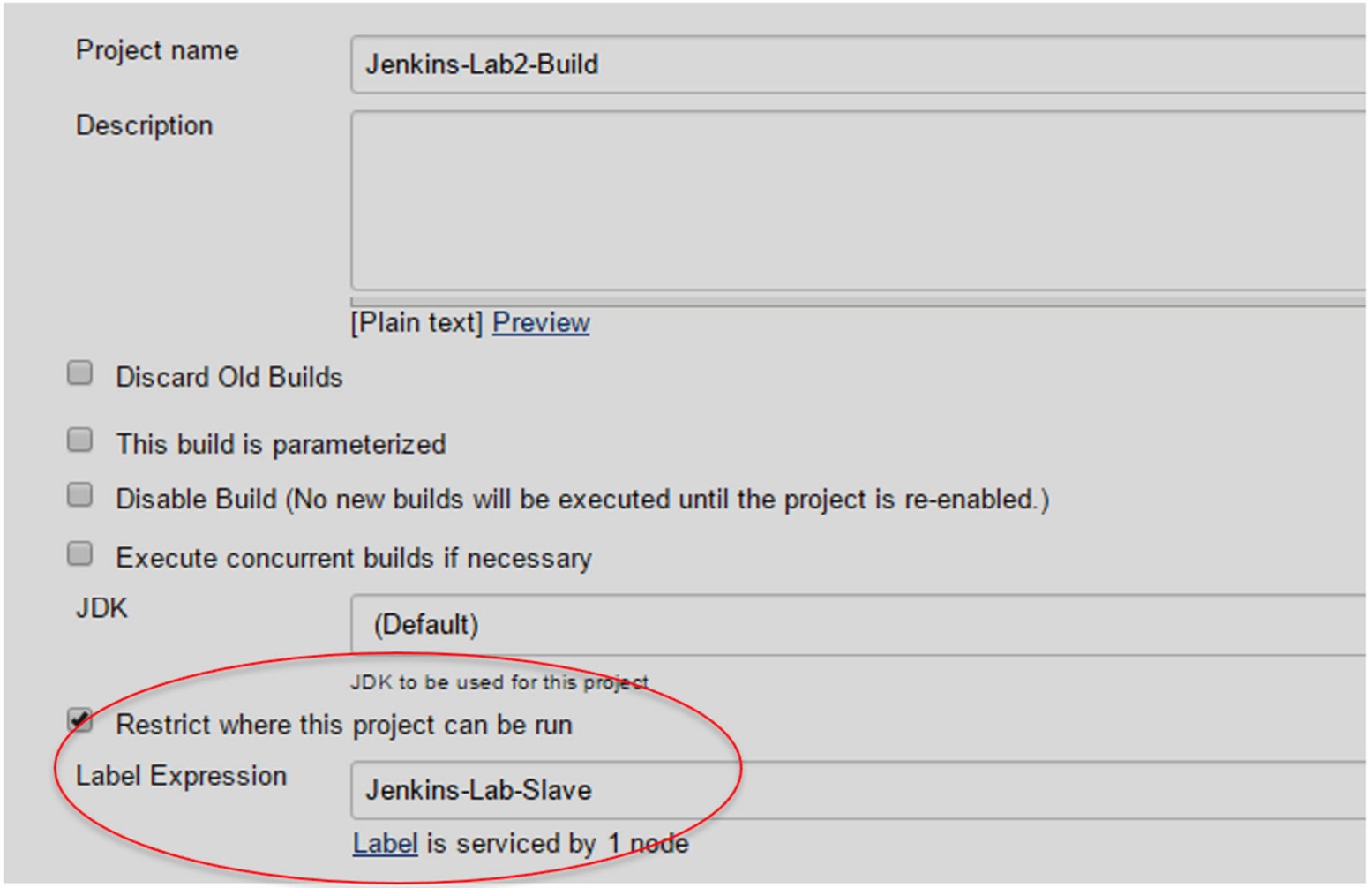
Launch Slave via Java Web Start – Slave Node

- 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 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



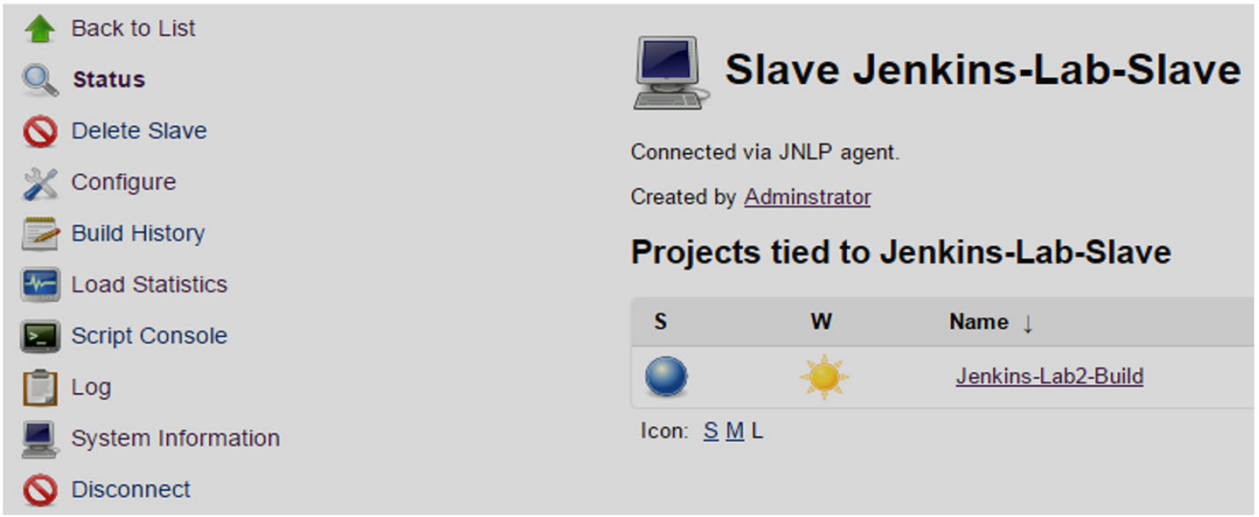
Associate build jobs with Master and Slave nodes

- 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



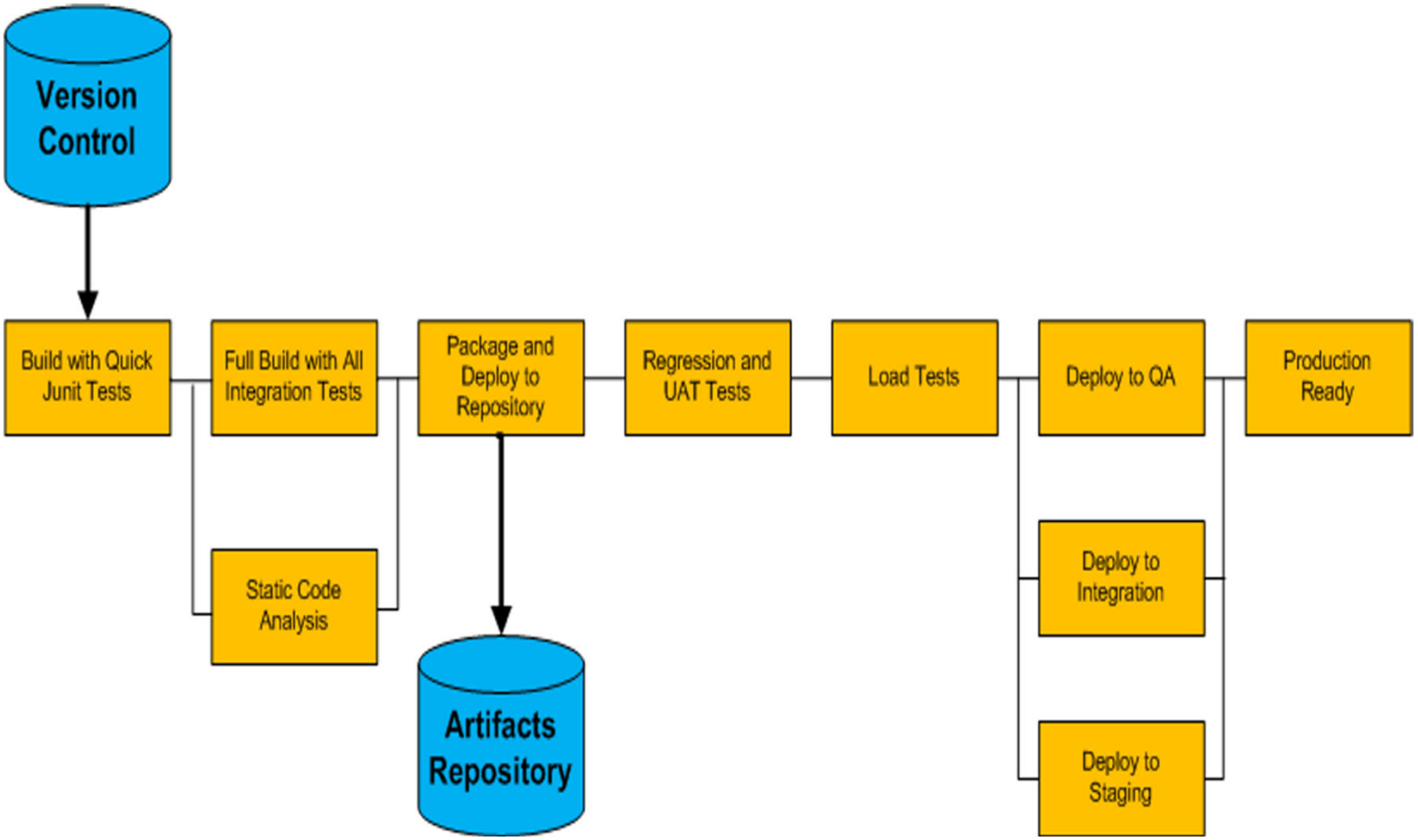
Topics in this Session

- 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

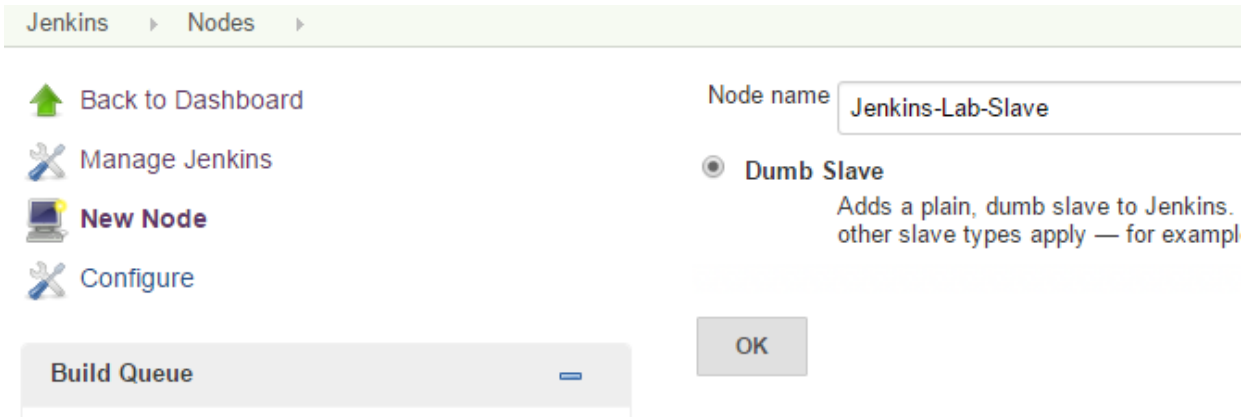
Build Pipelines



Pipeline Practices

- 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

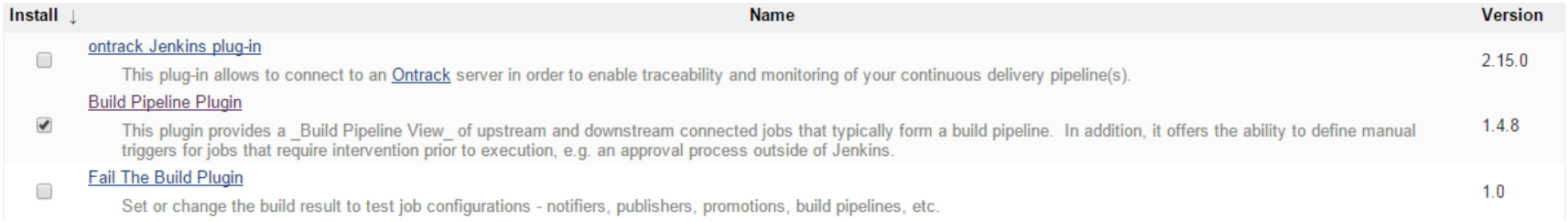
Creating Pipeline with Jenkins



- 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

Jenkins Build Pipeline Plugin

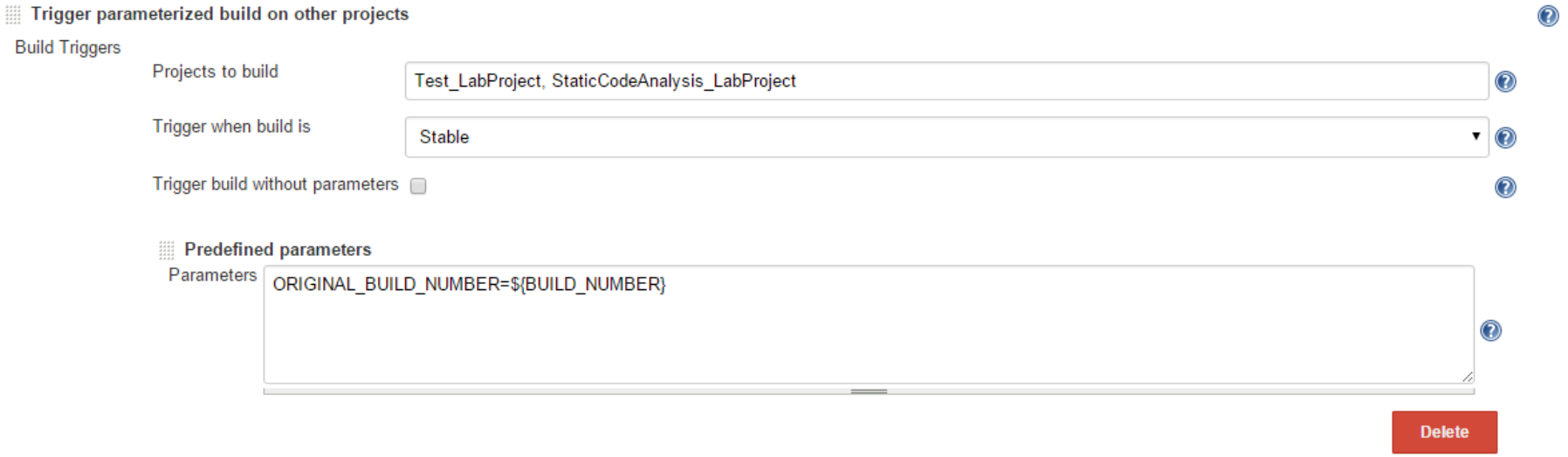
- Jenkins Build Pipeline Plugin provides visualization of the build pipeline view that shows upstream and downstream connected jobs
- Offers ability to define manual triggers for jobs
  - Useful in approval process for deploying in UAT environment



Install ↓	Name	Version
☐	ontrack Jenkins plug-in This plug-in allows to connect to an Ontrack server in order to enable traceability and monitoring of your continuous delivery pipeline(s).	2.15.0
☑	Build Pipeline Plugin This plugin provides a _Build Pipeline View_ 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
☐	Fail The Build Plugin Set or change the build result to test job configurations - notifiers, publishers, promotions, build pipelines, etc.	1.0

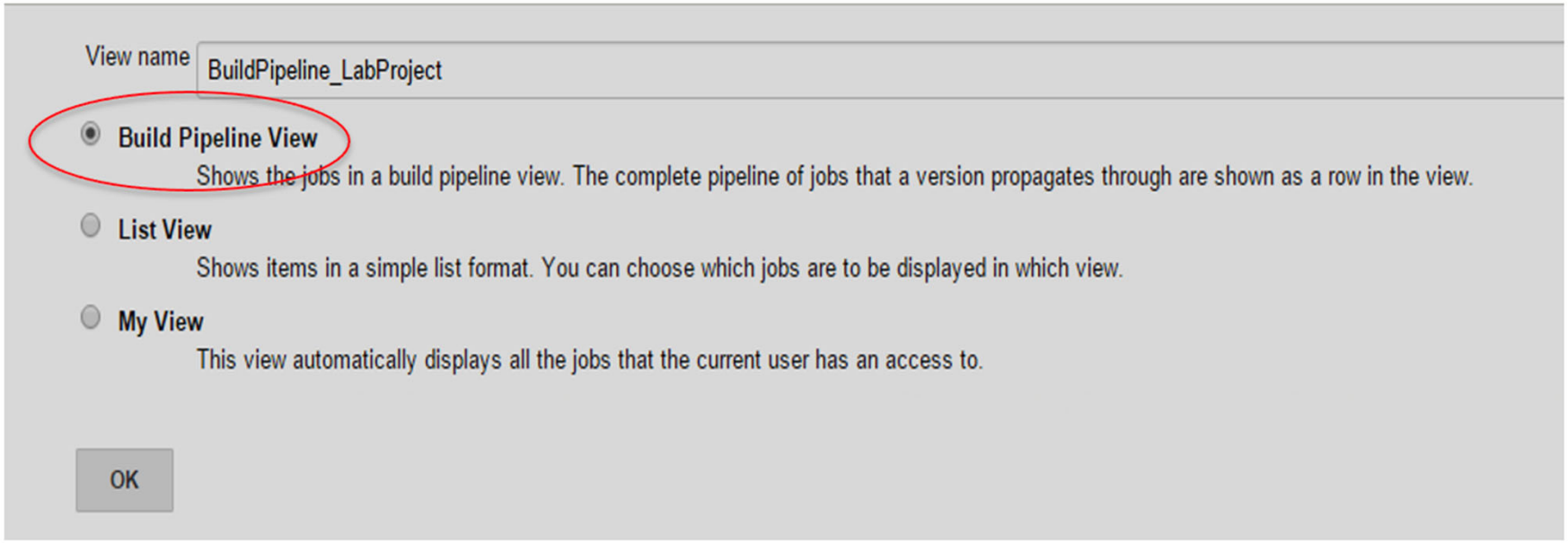
Jenkins Parameterized Trigger Plugin

- Parameterized Trigger is another plugin you need to implement a pipeline in Jenkins
- Build Pipeline plugin has dependencies on the 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



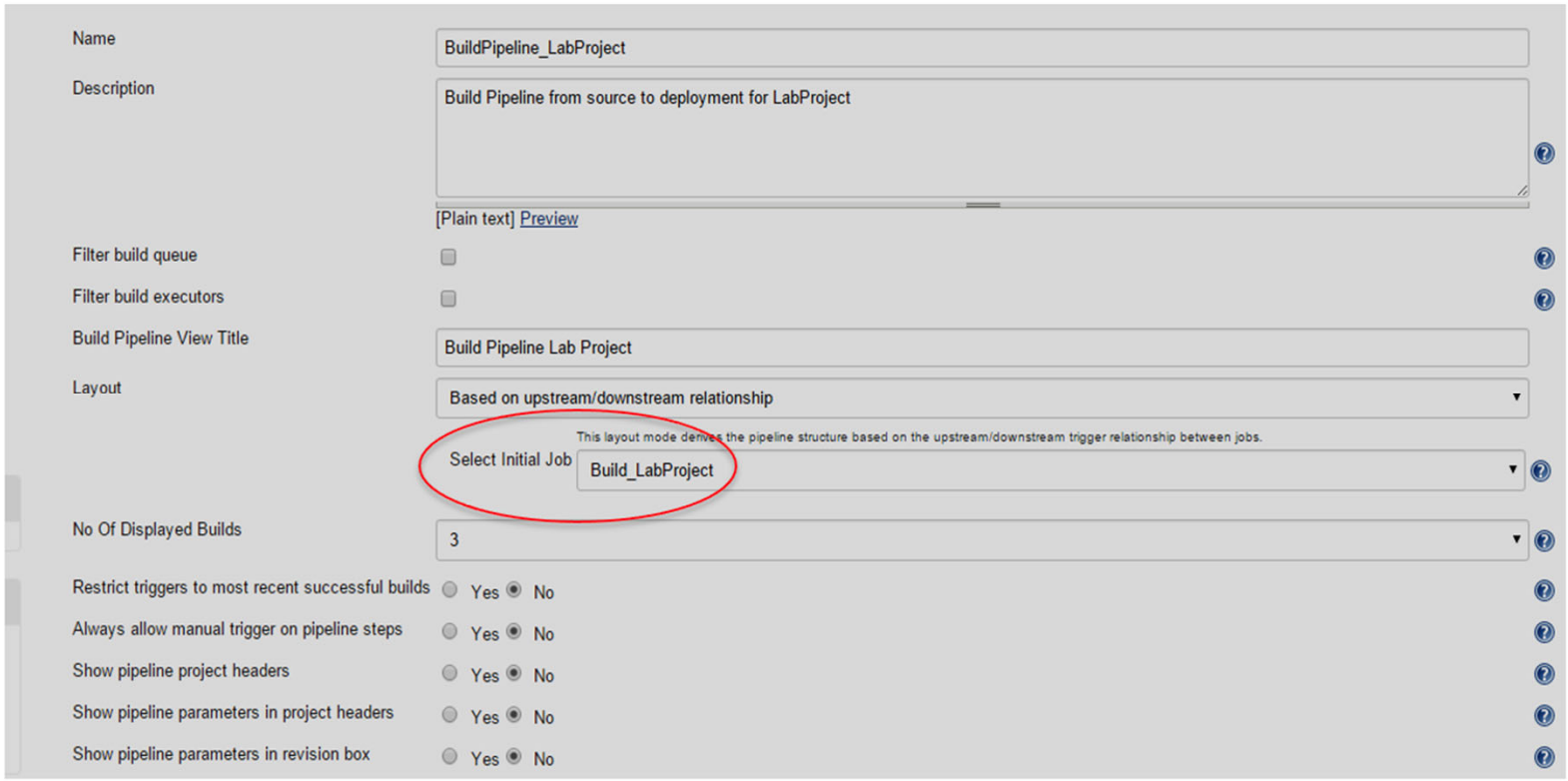
Create New Build Pipeline View

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

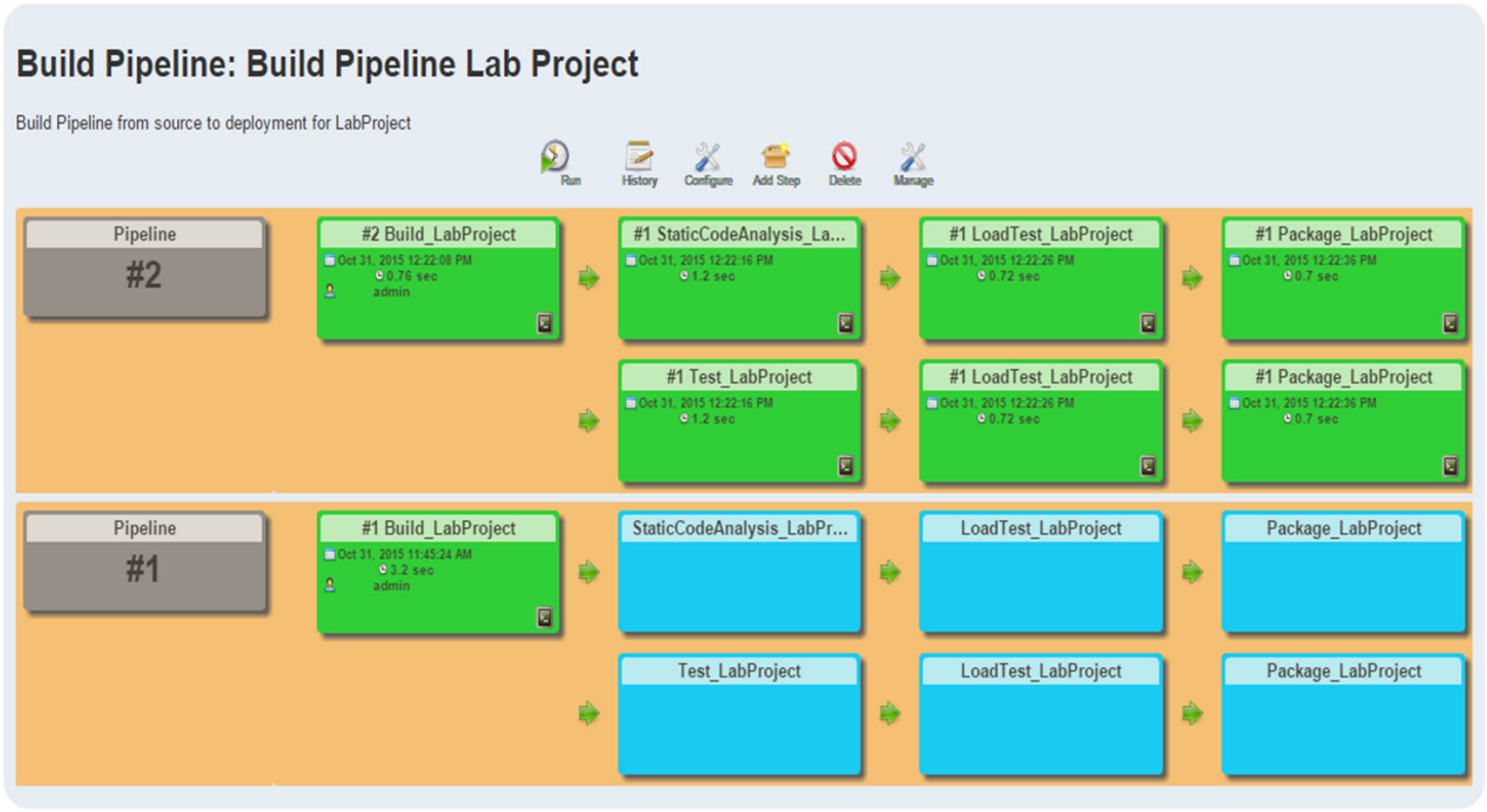


Configure New Build Pipeline View

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



New Pipeline for Lab Project



Topics in this Session

- 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

Archive Workspace from One Job to Next 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 a few plugins for creating smaller jobs
  - Clone Workspace SCM Plugin
  - Build Flow Plugin
  - Build Pipeline Plugin
  - Parameterized Trigger Plugin

Jenkins Clone Workspace SCM 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

Topics in this Session

- 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

File Fingerprint Tracking

- 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

Topics in this Session

- 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

Topics in this Session

- 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

Topics in this Session

- 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

Introducing Continuous Integration and Jenkins

Installing and Running Jenkins

Jenkins Job

Jenkins Plugins

SonarQube Plugin

Advanced Jenkins

Best Practices for Jenkins

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

Culture Change

- 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

Culture Change

- 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

Write unit tests

- 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 performed sparingly 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

Reuse build scripts across IDE and Jenkins

- Use **Groovy and Gradle** to reuse code and scripts across an IDE and Jenkins
- Groovy and Gradle customize, configure, and extend the build process
- Leverage Gradle and Groovy scripts as part of the 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

- A 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

Continuous Integration with Jenkins

- **Q & A**

- **Thank you!**