Continuous Integration and delivery for Java based web applications using Jenkins-Gradle-Artifactory

Sunil Dalal (@sunieldalal)
Full Stack Developer / Architect

What is 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. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly."

- Martin Fowler, ThoughtWorks Chief Scientist

What is Continuous Delivery?

Continuous Delivery is the natural extension of Continuous Integration: an approach in which teams ensure that every change to the system is releasable, and that we can release any version at the push of a button. Continuous Delivery aims to make releases boring, so we can deliver frequently and get fast feedback on what users care about.

http://www.thoughtworks.com/continuous-delivery

It’s best thing happened in my developer life! CI and CD gives a great confidence to developer and makes them more productive. CI/CD is a must for any successful software product.

- My View

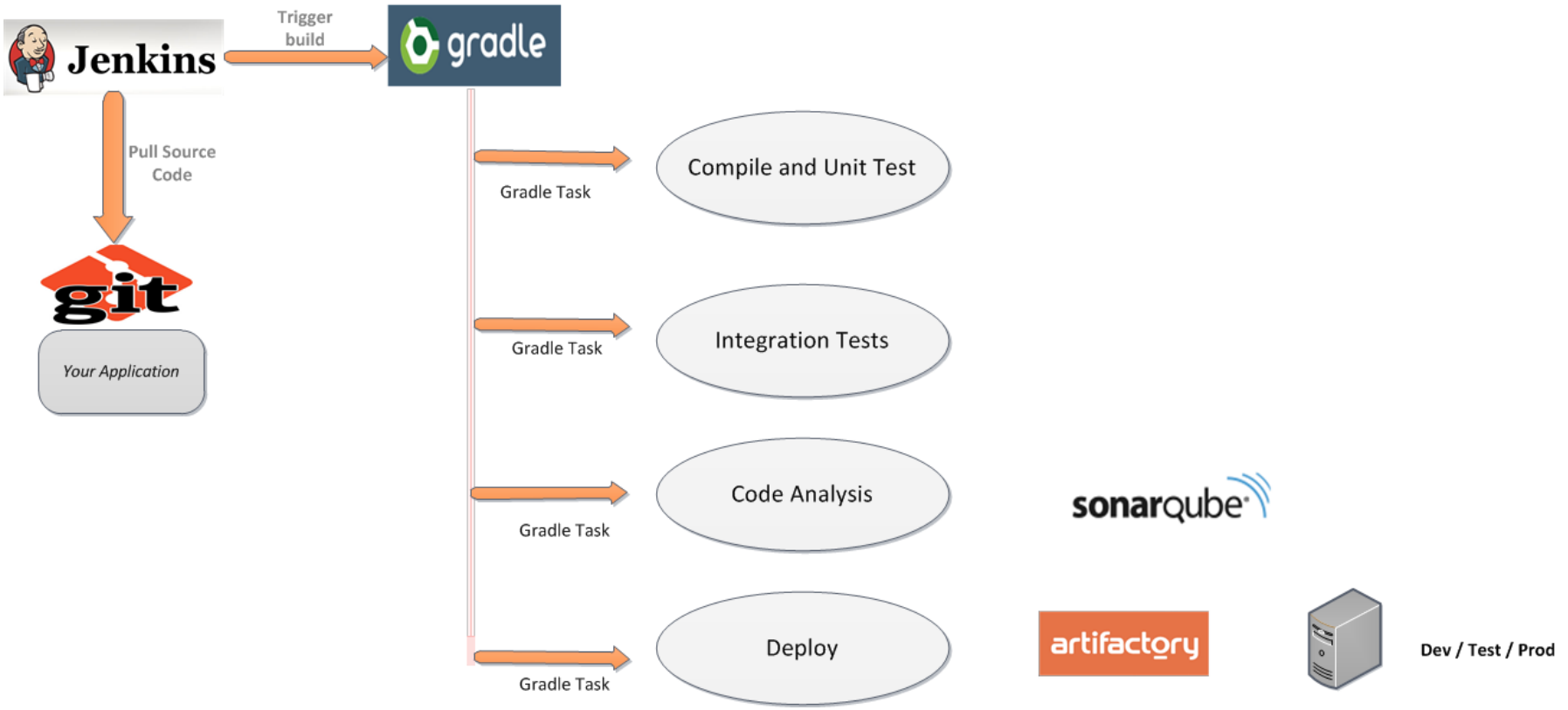
Before Continuous Integration / Delivery?

- Manual Deployments.
- Manual Restarts.
- No Automated Test infrastructure. Only manual testing!
- Less confidence! Long turnaround time.

Why Continuous Integration / Delivery?

- Immediate feedback, faster turn around time.
- Enforces discipline of frequent automated testing.
- Better coding practices - Frequent code check-in pushes developers to create modular, less complex code.
- Every commit can result in a release!

Continuous Integration / Delivery Workflow



CI / CD step by step

- Developers **check out** code into their local machines.
- Developer **commit** changes to the source code repository.
- The CI server monitors the source code repository and starts build process.
- The CI server **builds** the system and runs unit and integration tests.
- The CI server releases deployable artifacts for **testing**.
- The CI server assigns a build label to the version of the code it just built.
- The CI server informs the team of the **successful build**.
- If the build or tests fail, the CI server **alerts** the team.
- The team fix the issue at the earliest opportunity.
- Continue to continually integrate and test throughout the project
- On Approval, CI server deploys to test environments.
- CI Server runs functional / performance tests on Test environment.
- CI Server stage / Deploys to Production Environments (Can set up email based approval).

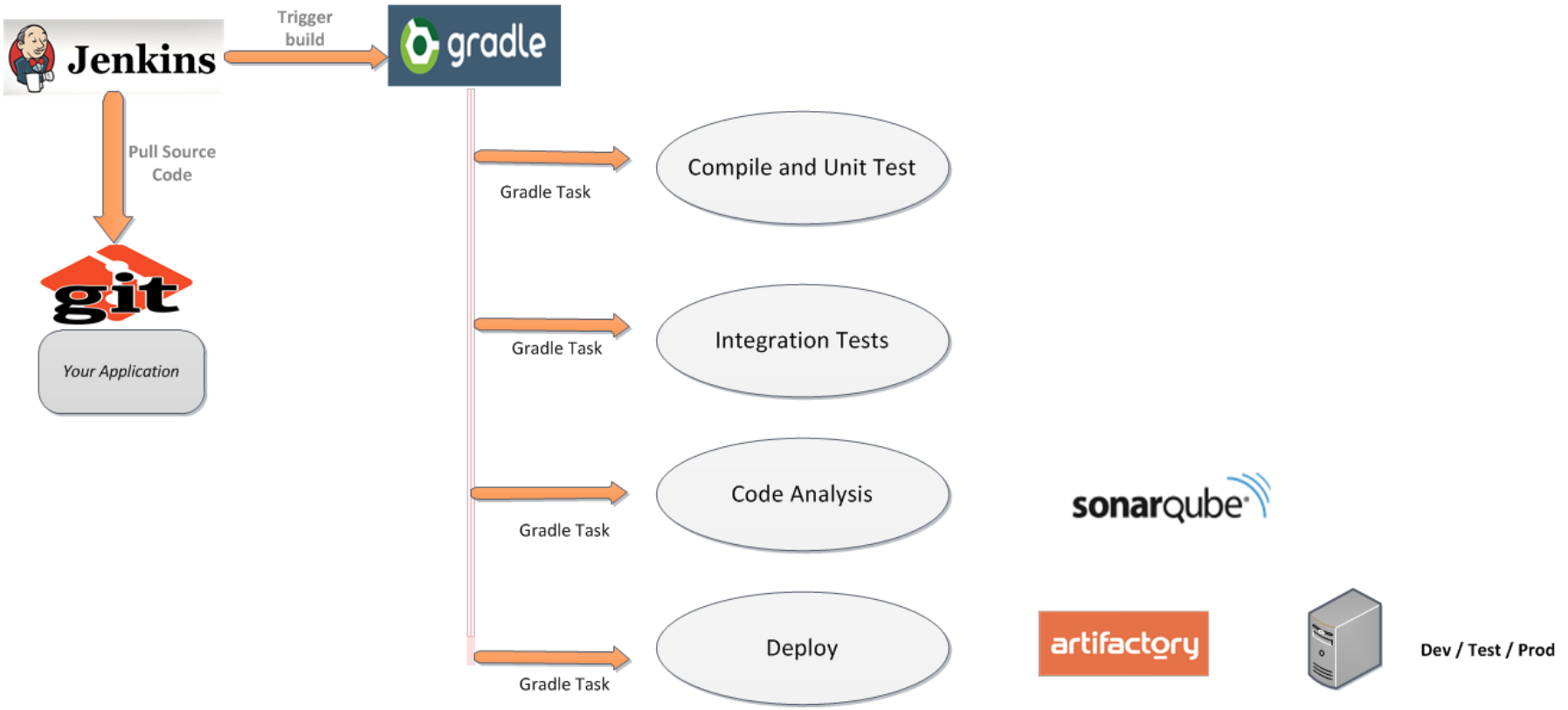
Reference: https://www.thoughtworks.com/continuous-integration

Tools Used

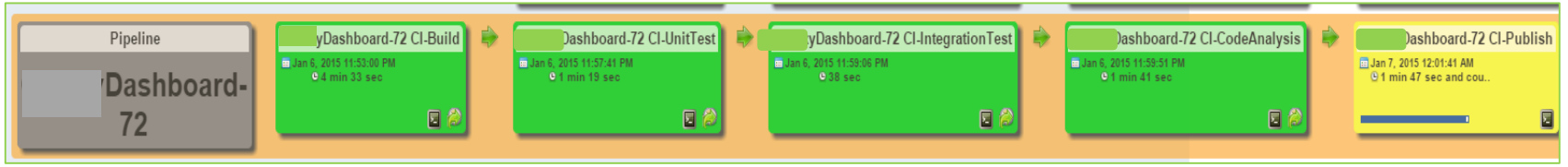
Tools	Options
Source Code Repository	Github, Perforce
Continuous Integration Server	Jenkins
Binary Repository Manager	Artifactory
Code Analysis	Sonar Qube
Code Analysis Tools(Java)	Checkstyle, PMD, Jacoco, FindBugs, JDepend
Build Tools (Java)	Gradle

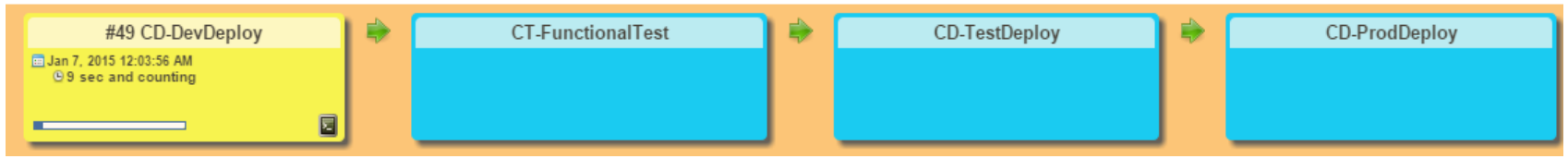
Achieving CI / CD using Gradle / Jenkins / Artifactory

CI/CD Workflow



Jenkins Workflow



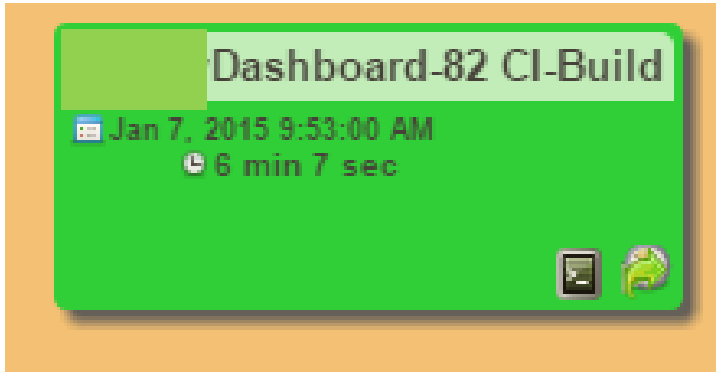


Jenkins Workflow

Each Jenkins job executes one of the commands available via Gradle

gradle compile
gradle build
gradle test

Jenkins Build Step



Jenkins Build Step

Executes gradle build via Jenkins job

```
//  Logging Support
compile 'org.slf4j:slf4j-api:1.7.7'
runtime 'org.slf4j:slf4j-log4j12:1.7.7'

// Servlet Api compile time support
providedCompile  'javax.servlet:javax.servlet-api:3.0.1'

// Spring WEB MVC Support
compile 'org.springframework:spring-core:4.0.2.RELEASE'
compile 'org.springframework:spring-webmvc:4.0.2.RELEASE'

// Spring Data JPA Support
compile 'org.springframework.data:spring-data-jpa:1.3.0.RELEASE'
compile "org.springframework:spring-tx:4.0.2.RELEASE"
compile "org.springframework:spring-orm:4.0.2.RELEASE"
compile "org.springframework:spring-aop:4.0.2.RELEASE"
```

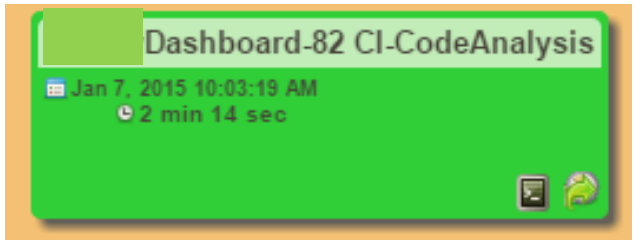
Jenkins Unit Test Step

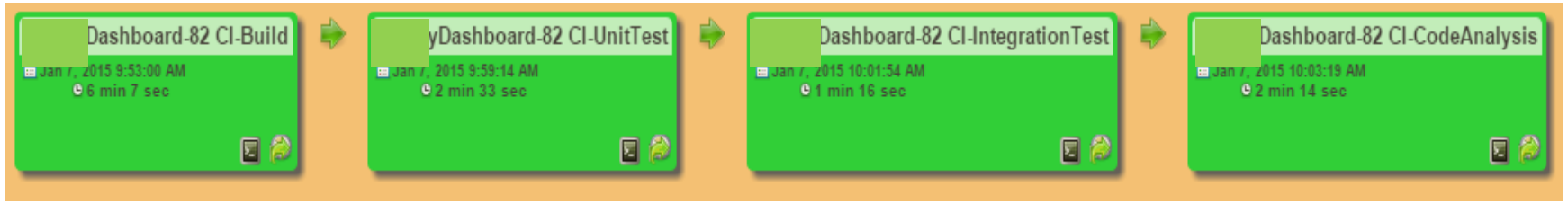
Jenkins Build Step

Executes gradle test command via Jenkins job

```
        testCompile 'org.springframework:spring-test:4.0.2.RELEASE'
        testCompile 'junit:junit:4.11'
        testCompile "org.mockito:mockito-core:1.9.5"
        testCompile "org.hamcrest:hamcrest-library:1.3"
        // Spring Test Support
        testCompile 'org.springframework:spring-test:4.0.2.RELEASE'
}

test {
        // Force execution of tests. Will help in Jenkins scenario as gradle supports incremental builds and avoids rerunning tests
        // This line will force execution of tests
        outputs.upToDateWhen { false }

        //  The tests that are executed by the test task run in a separate, isolated JVM process.
        testLogging {
            // Show that tests are run in the command-line output We can pass arguments to determine
            // which test events we want to see in the
            // command-line output.
            events 'started', 'passed'

            // Show logging events for test methods.
            minGranularity = 3

            // All valid values for the stackTrace output.
            stackTraceFilters 'groovy', 'entry_point', 'truncate'
            // Show System.out and System.err output
            // from the tests.
            showStandardStreams = true

            // Set exception format to full instead of default value 'short'.
            testLogging.exceptionFormat 'full'

            // Configure options for DEBUG log level.
            debug {
                events 'started'
            }
```

Jenkins Code Analysis Step



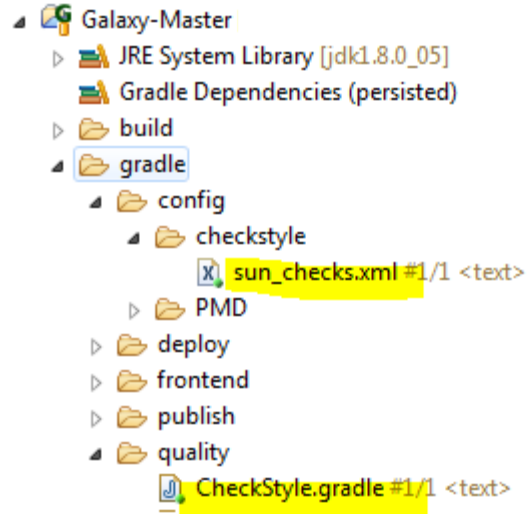


Code Analysis Tools

- Checkstyle
  - ✓Ideal for projects that want to enforce a coding standard. Discovers poor design, duplicated code and bug patterns.
- FindBugs
  - ✓Helps in discovering potential bugs, performance issues and bad coding practices.
- Jdepend
  - ✓Helps in measuring design quality metrics like maintainability, reusability, extensibility.
- PMD
  - ✓Finds unused, overly complex and inefficient code.
- JaCoCo
  - ✓Used for Code Coverage. On the fly byte code instrumentation.

Code Analysis Tools - Checkstyle

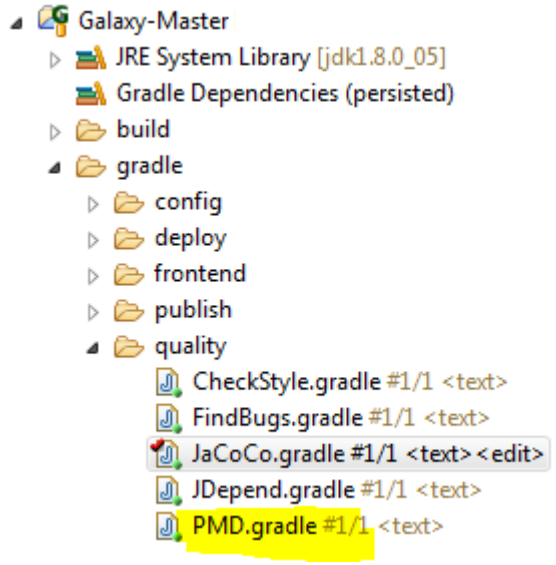
✓Checkstyle is highly configurable and can be made to support almost any coding standard.
✓Integrated by adding checstyle Gradle plugin in the project.




```
apply plugin: "checkstyle"
ext.checkstyleConfigDir = "$configDir/checkstyle"

checkstyle {
    // https://github.com/checkstyle/checkstyle/blob/master/sun_checks.xml
    configFile = new File(checkstyleConfigDir, 'sun_checks.xml')
    ignoreFailures = true
    showViolations = false
}
```

Code Analysis Tools - PMD

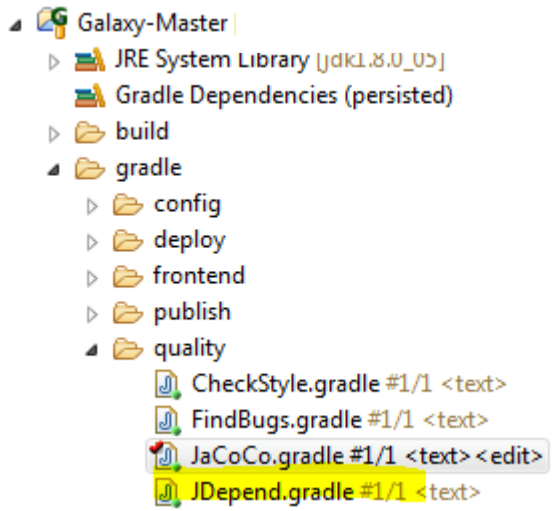
- ✓ PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, XML, XSL.
- ✓ Integrated by adding PMD Gradle plugin in the project.




```
1  apply plugin: 'pmd'
2
3  pmd {
4      ignoreFailures = true
5      ruleSets = ["java-basic", "java-braces"]
6  }
7
```

Code Analysis Tools - JDepend

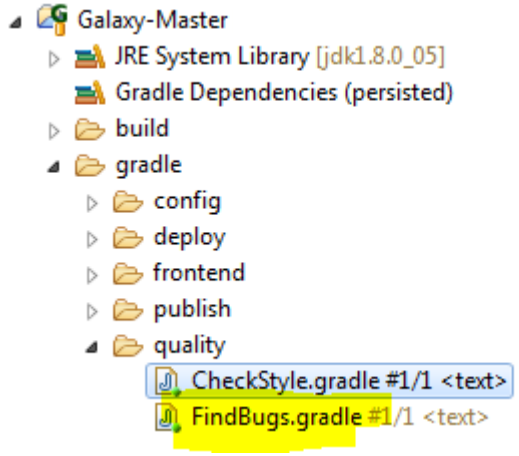
- ✓ JDepend traverses Java class file directories and generates design quality metrics for each Java package. JDepend allows you to automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to manage package dependencies effectively.
- ✓ Integrated by adding Jdepend Gradle plugin in the project.




```
1  apply plugin: 'jdepend'
2  ext.jdependConfigDir = "$configDir/jdepend"
3
4  jdepend {
5      ignoreFailures = true
6  }
7  tasks.withType(JDepend) {
8      reports {
9          text.enabled = false
10         xml.enabled = true
11     }
12 }
```

Code Analysis Tools - FindBugs

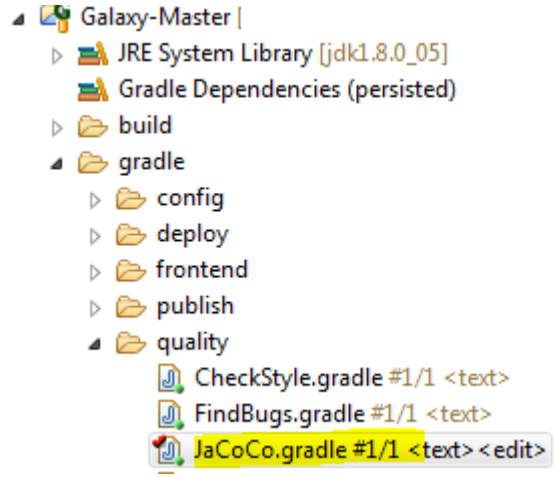
✓uses static analysis to look for bugs in Java code.
✓Integrated by adding FindBugs Gradle plugin in the project.




```
1  apply plugin: 'findbugs'
2
3  findbugs {
4      ignoreFailures = true
5      effort = 'max'
6  }
7
8  tasks.withType(FindBugs) {
9      reports {
10     xml.enabled = false
11     html.enabled = true
12     }
13 }
```

Code Analysis Tools - JaCoCo

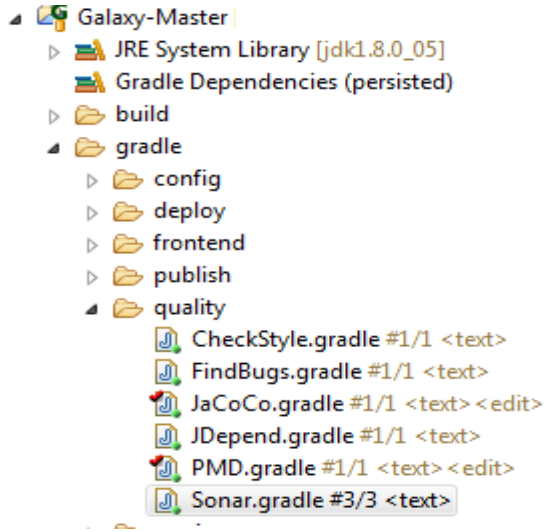
- ✓ Used for Code Coverage. On the fly byte code instrumentation.
- ✓ In a continuous delivery pipeline, where packaging the deliverable is done after executing the code analysis phase, we want to make sure that the source or byte code isn't modified after the compilation process to avoid unexpected behavior in target enironment. That's why, on-the-fly instrumentation should be preferred.
- ✓ Integrated by adding JaCoCo Gradle plugin in the project.



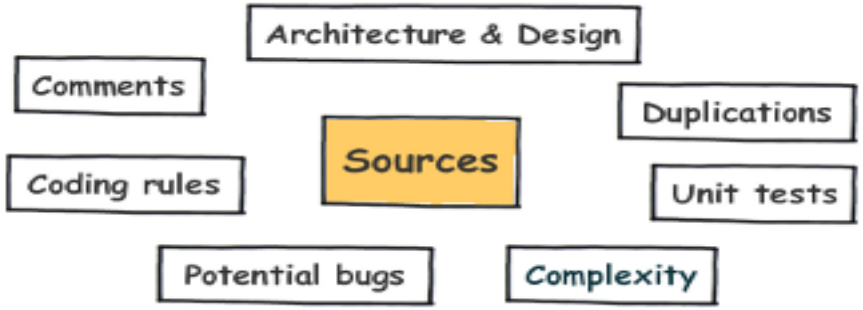

```
1 apply plugin: "jacoco"
2
3 jacoco {
4
5 }
6
```

SONAR - Architect's friend in managing code quality

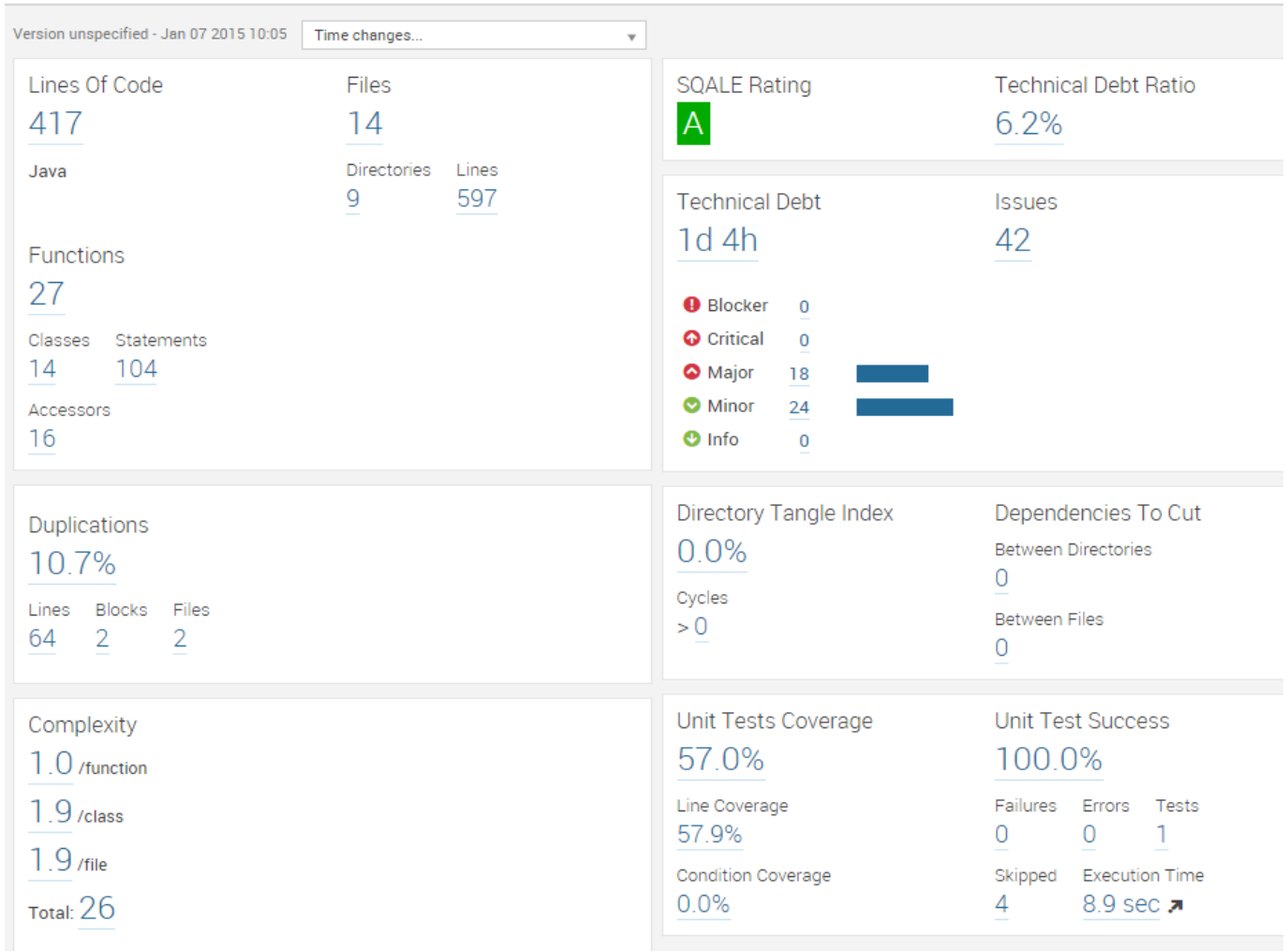
✓SonarQube is an open platform to manage code quality. As such, it covers the 7 axes of code quality.
✓Imports All Code analysis data in SONAR DB for further analysis..



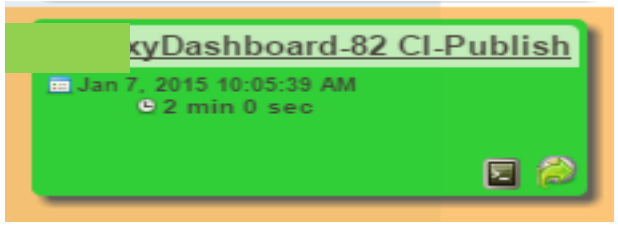

```
apply plugin: 'sonar-runner'

sonarRunner {

    toolVersion = '2.3' // default

    forkOptions {
        maxHeapSize = '512m'
    }
```



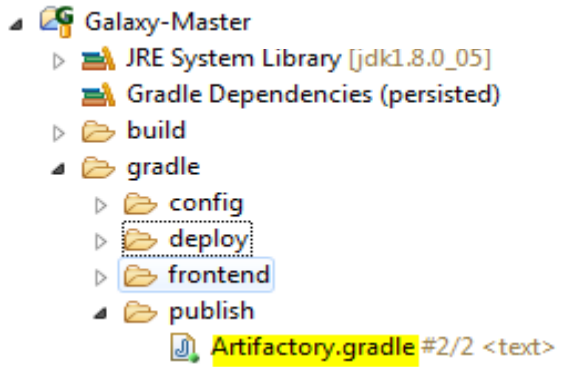
SONAR Dashboard



CI – Publish

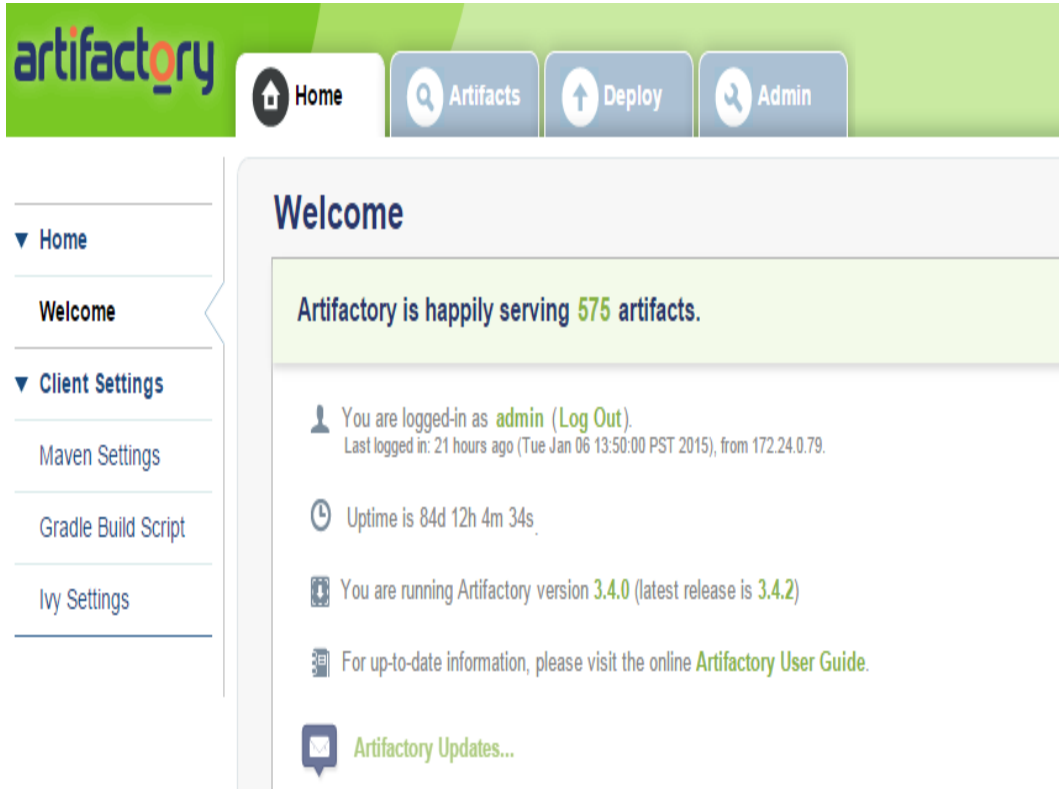


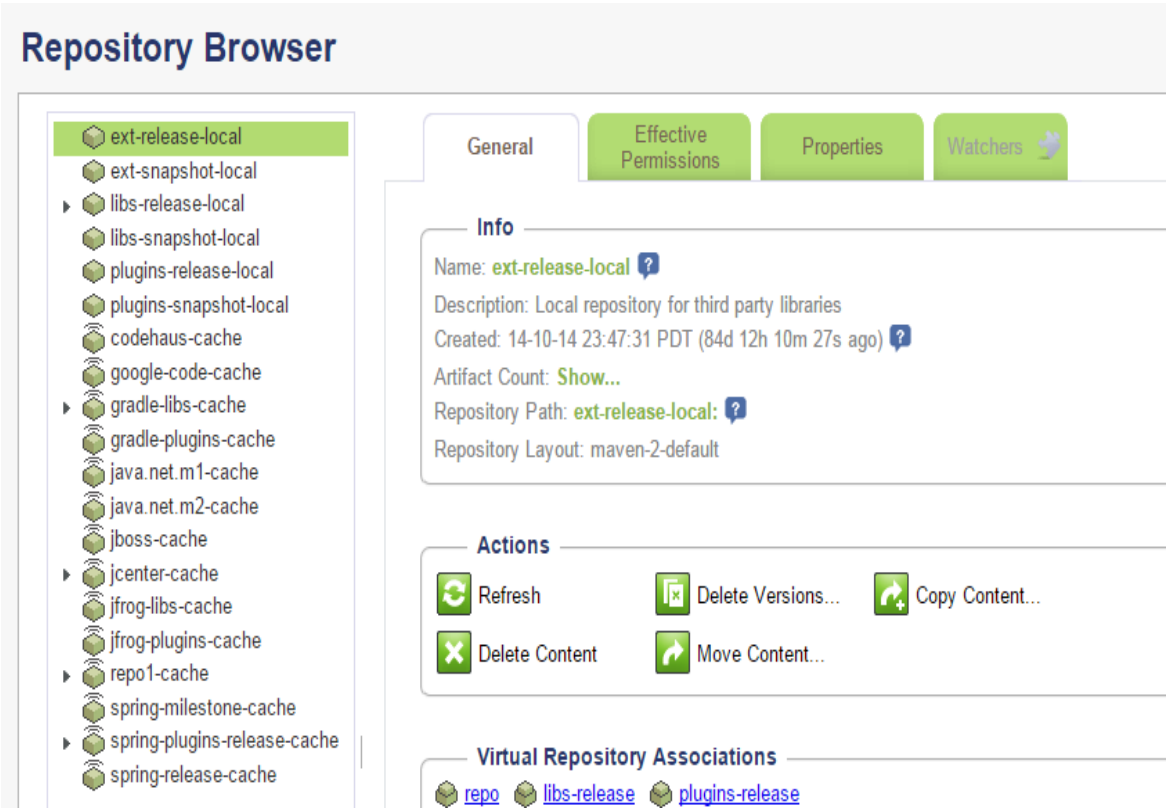
Artifactory Integration



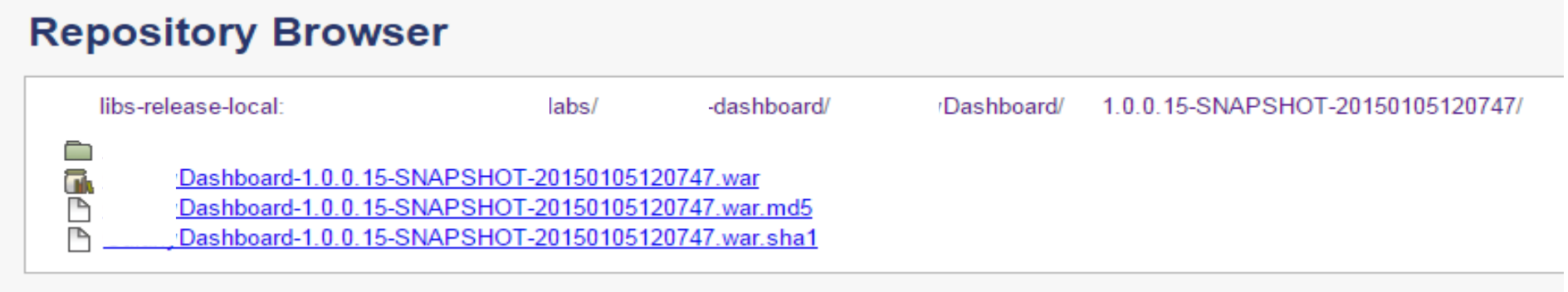

```
apply plugin: 'distribution'

apply plugin: "com.jfrog.artifactory"
apply plugin: "com.jfrog.artifactory-upload" //

artifactory {
    contextUrl = "${artifactory_contextUrl}"   //The base Artifactory URL if not overridden by the publisher/resolver
    publish {
        repository {
            repoKey = 'libs-release-local'
            username = "${artifactory_user}"
            password = "${artifactory_password}"
            maven = true
        }

        defaults {
          publications ('ivyJava')
          properties = ['build.status': "$it.project.status".toString()]
          publishPom = false //Publish generated POM files to Artifactory (true by default)
          publishIvy = true //Publish generated Ivy descriptor files to Artifactory (true by default)
        }

    }
    resolve {
        repository {
            repoKey = 'libs-release'
            username = "${artifactory_user}"
            password = "${artifactory_password}"
            maven = true
        }
    }
}
```

Artifactory Dashboard

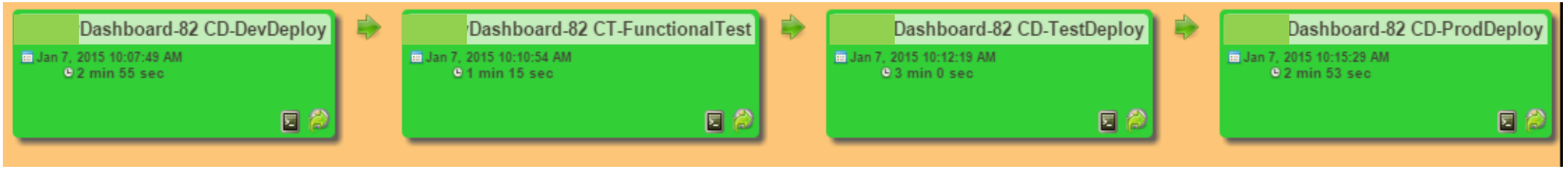




Deploy Artifacts



CI – Deploy


Dashboard-82 CD-DevDeploy
Jan 7, 2015 10:07:49 AM
2 min 55 sec
Dashboard-82 CT-FunctionalTest
Jan 7, 2015 10:10:54 AM
1 min 15 sec
Dashboard-82 CD-TestDeploy
Jan 7, 2015 10:12:19 AM
3 min 0 sec
Dashboard-82 CD-ProdDeploy
Jan 7, 2015 10:15:29 AM
2 min 53 sec

CI – Deploy

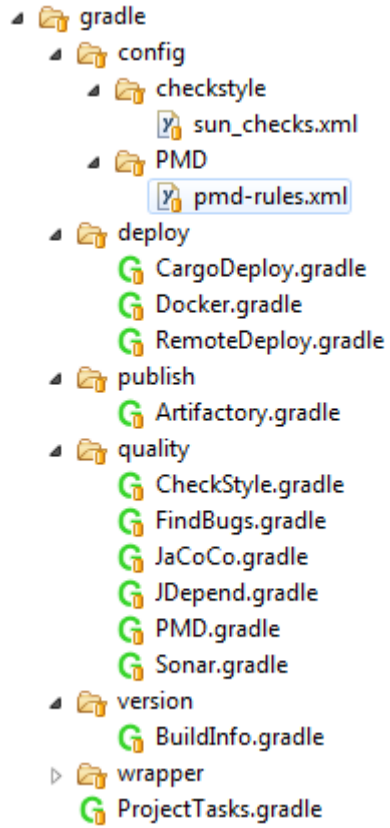
- ✓ Can Use scripts or Plugins which can directly deploy to Tomcat or any other application Servers.

Common Questions

Great! what all plugins you guys have used for Jenkins?

- ✓ Git Plugin to checkout source code from repo.
- ✓ Gitlab plugin for polling and start builds on each checkin.
- ✓ Gradle plugin to install gradle on CI and execute Gradle commands.
- ✓ Credentials Plugin to store the credentials securely.
- ✓ Build Pipeline plugin.
- ✓ Used Clone Workspace Plugin.
- ✓ Used Set Build Name Plugin.
- ✓ Parametrized Build Plugin.
- ✓ GitLab plugin for integration with Gitlab.

Great! what all plugins you guys have used for Gradle?



Do we need to copy these gradle plugins to every project? Code duplication?

- ✓ Keep it DRY (Don’t Repeat yourself)
- ✓ You can move all gradle plugins to your company wide gradle wrapper
- ✓ Individual projects can point to your company wide gradle wrapper to get new updates.

Do you do build on each check in?

- ✓ Set up Git Jenkins SSH integration so that we can invoke build on each check in. Works great for smaller teams.
- ✓ For larger teams, CI server polls the SCM and build every 10 minutes!

How to manage versioning for project?

- ✓ SNAPSHOT-BuildNumber for internal builds.
- ✓ RELEASE-x.0 for builds to be released.
- ✓ Versioning can be managed via gradle properties file where you can tag version type as SNAPSHOT or RELEASE
- ✓ OR
- ✓ Make changes in your CI server to have jobs which can tag builds based on your defined criteria.
- ✓ You can also have jobs which can promote builds from SNAPSHOT to RELEASE.

How do you do rollback?

- ✓ Rollback should be one click similar to deploy.
- ✓ Make your CI server Jobs intelligent to achieve this.

Thank you so much!

Twitter: @sunieldalal
Blog: http://sunieldalal.github.io/
Github: https://github.com/sunieldalal
LinkedIn: https://www.linkedin.com/in/sunieldalal