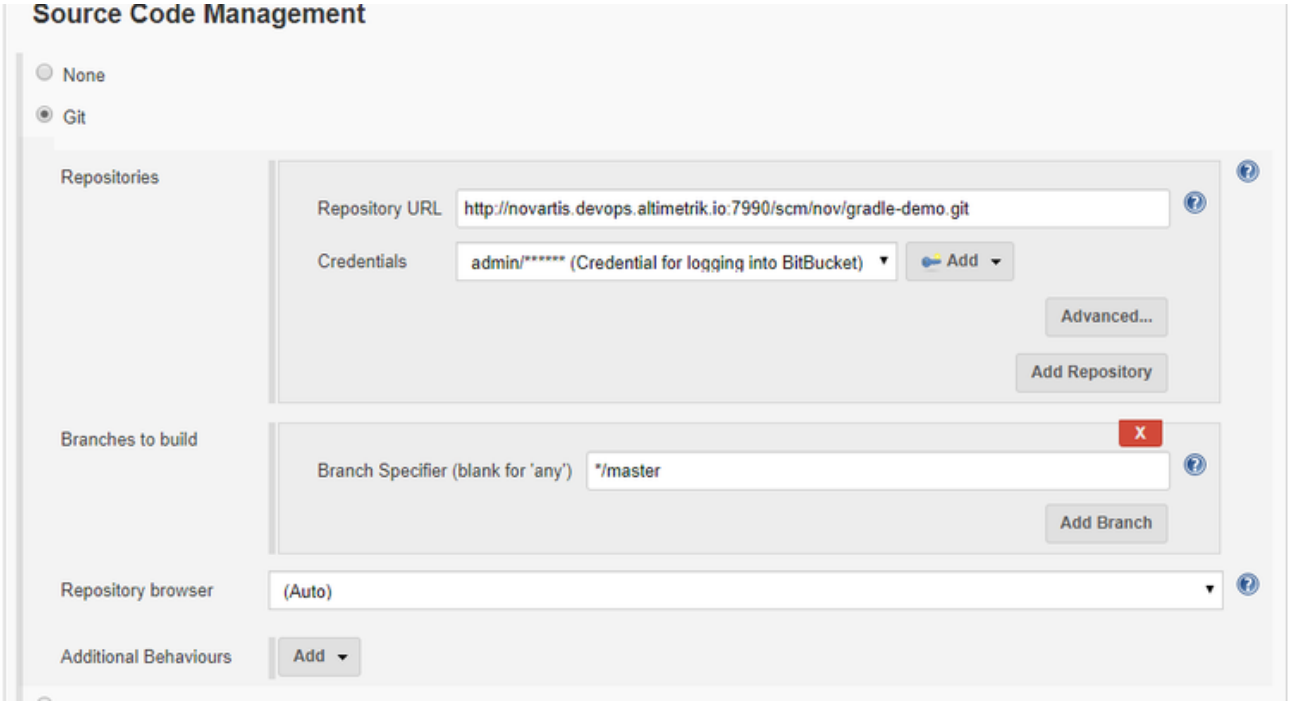
How to - End to End Jenkins integration with Gradle

1.1 Create a Jenkins job

1. Create new freestyle project
2. Under SourceCode select GIT



3. Create a "Build step" in the section "Build" by selecting "Invoke Gradle build script".

CMD : gradle build

Output: Build will successfully executed along with test cases

```
[test-gradle] $ /var/jenkins_home/tools/hudson.plugins.gradle.GradleInstallation/gradle6/bin/gradle build s
Starting a Gradle Daemon (subsequent builds will be faster)
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :jar UP-TO-DATE
> Task :startScripts UP-TO-DATE
> Task :distTar UP-TO-DATE
> Task :distZip UP-TO-DATE
> Task :assemble UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test UP-TO-DATE
> Task :check UP-TO-DATE
> Task :build UP-TO-DATE
```

1.2 Integrate sonar

Run Sonar with Build.gradle file

1. **Activate the SonarQube plugin in your build**

To get Sonar working in Gradle you need to apply the sonar plugin, like this:

```
########################
build.gradle

plugins {
  id "org.sonarqube" version  "2.7"
}


########################
```

2.Adding code coverage to our build

If we want to add code coverage to such a project we need to add jacoco in the version corresponding to the jacoco-gradle-plugin to our libs in `build.gradle`

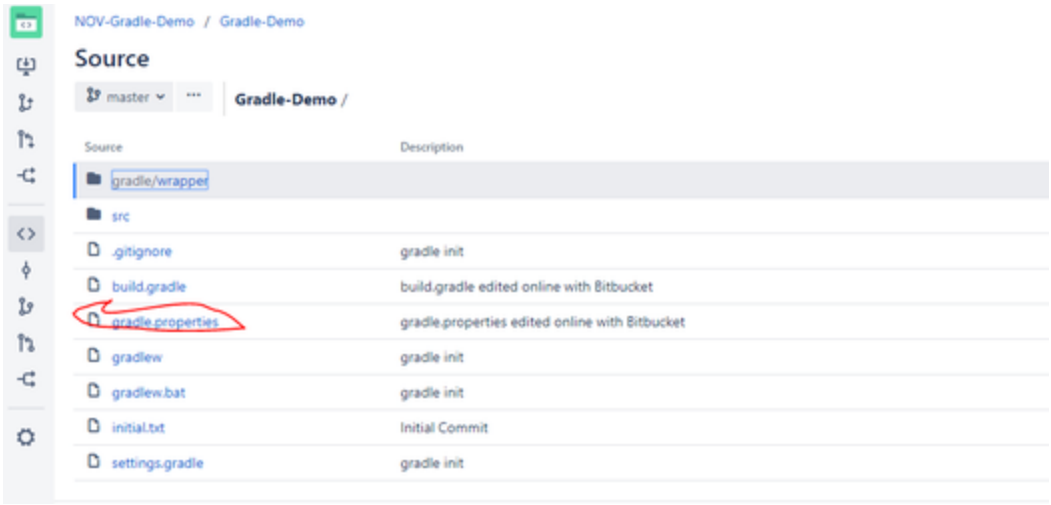
```
##############################

build.gradle

plugins {

id 'jacoco'

}

##############################
```

3.Configure the Scanner

Installation is automatic, but certain global properties should still be configured.Be aware that the scanner uses system properties so all properties should be prefixed by `systemProp`.

```
############################################

project root directory global.properties


systemProp.sonar.host.url=http://novartis.devops.altimetrik.io:9000/
systemProp.sonar.sourceEncoding=UTF-8
systemProp.sonar.forceAuthentication=true


#----- Token generated from an account with 'publish analysis'
permission
systemProp.sonar.login=<token>


############################################
```

under project root directory global.properties



4.Configure analysis properties

The SonarQube plugin leverages information contained in Gradle's object model to provide smart defaults for many of the standard SonarQube properties. The defaults are summarized in the tables below.

Gradle defaults for standard SonarQube properties

Property	Gradle default
`sonar.projectKey`	"$project.group:$project.name"
`sonar.projectName`	project.name
`sonar.projectDescription`	project.description
`sonar.projectVersion`	project.version
`sonar.projectBaseDir`	project.projectDir
`sonar.working.directory`	"$project.buildDir/sonar"

Additional defaults when `java-base` plugin is applied

Property	Gradle default
`sonar.java.source`	project.sourceCompatibility
`sonar.java.target`	project.targetCompatibility

Additional defaults when `java` plugin is applied

Property	Gradle default
`sonar.sources`	sourceSets.main.allSource.srcDirs (filtered to only include existing directories)

`sonar.tests`	sourceSets.test.allSource.srcDirs (filtered to only include existing directories)
`sonar.java.binaries/sonar.binaries`	sourceSets.main.runtimeClasspath (filtered to only include directories)
`sonar.java.libraries/sonar.libraries`	sourceSets.main.runtimeClasspath (filtering to only include files; `rt.jar` added if necessary)
`sonar.java.test.binaries`	sourceSets.test.runtimeClasspath (filtered to only include directories)
`sonar.java.test.libraries`	sourceSets.test.runtimeClasspath (filtering to only include files; `rt.jar` added if necessary)
`sonar.surefire.reportsPath`	test.testResultsDir (if the directory exists)
`sonar.junit.reportsPath`	test.testResultsDir (if the directory exists)

Additional defaults when `jacoco` plugin is applied

Property	Gradle default
`sonar.jacoco.reportPath`	jacoco.destinationFile

5.Run analysis

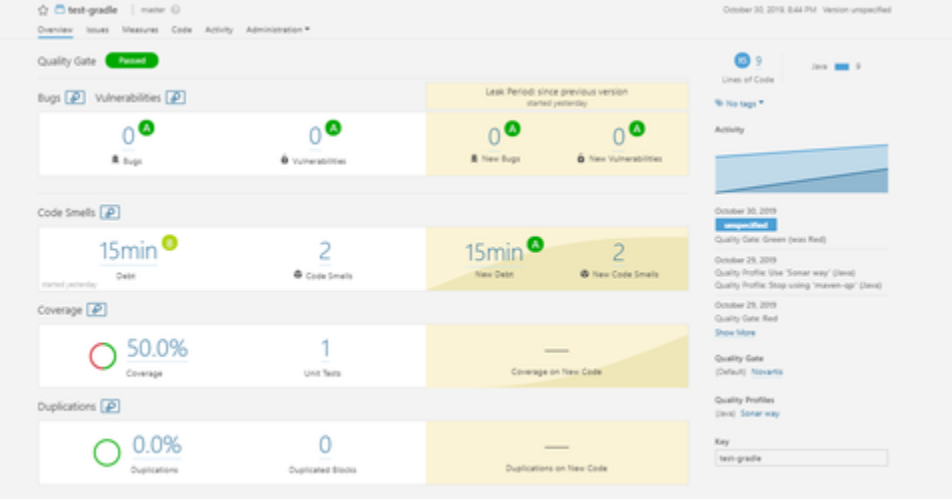
Execute `gradle sonarqube` and wait until the build has completed

```
##############
To run sonar

gradle sonarqube
###############
```

Output:

```
[test-gradle] $ /var/jenkins_home/tools/hudson.plugins.gradle.GradleInstallation/gradle6/bin/gradle build sonarqube
Starting a Gradle Daemon (subsequent builds will be faster)
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :jar UP-TO-DATE
> Task :startScripts UP-TO-DATE
> Task :distTar UP-TO-DATE
> Task :distZip UP-TO-DATE
> Task :assemble UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test UP-TO-DATE
> Task :check UP-TO-DATE
> Task :build UP-TO-DATE
> Task :sonarqube
```

Go to sonar dashboard



1.3 Integrate with nexus

Run Nexus with Build.gradle file

1. **Activate the nexus plugin in your build**

To get nexus working in Gradle you need to apply the maven plugin, like this:

```
##################################
To push artifacts in to nexus build.gradle

plugins {

id  'maven'

}



##################################
```

2. Configure properties in build file

build.gradle file that configures a build to publish artifacts to a novartis-maven repository

```
################################

build.gradle


repositories {
    maven {
          url
"http://novartis.devops.altimetrik.io:8082/repository/novartis-maven/"
    }
}


uploadArchives {
    repositories {
       mavenDeployer {
             repository(url:
"http://novartis.devops.altimetrik.io:8082/repository/novartis-maven/")
{
             authentication(userName: "admin", password: "password123")
}
             pom.version = "1.0-SNAPSHOT"
             pom.artifactId = "gradle-demo"
             pom.groupId = "com.demo"
       }
    }
}


################################
```

3. **Pushing jars into Nexus Repo**
To upload artifacts from this project, run "gradle upload"

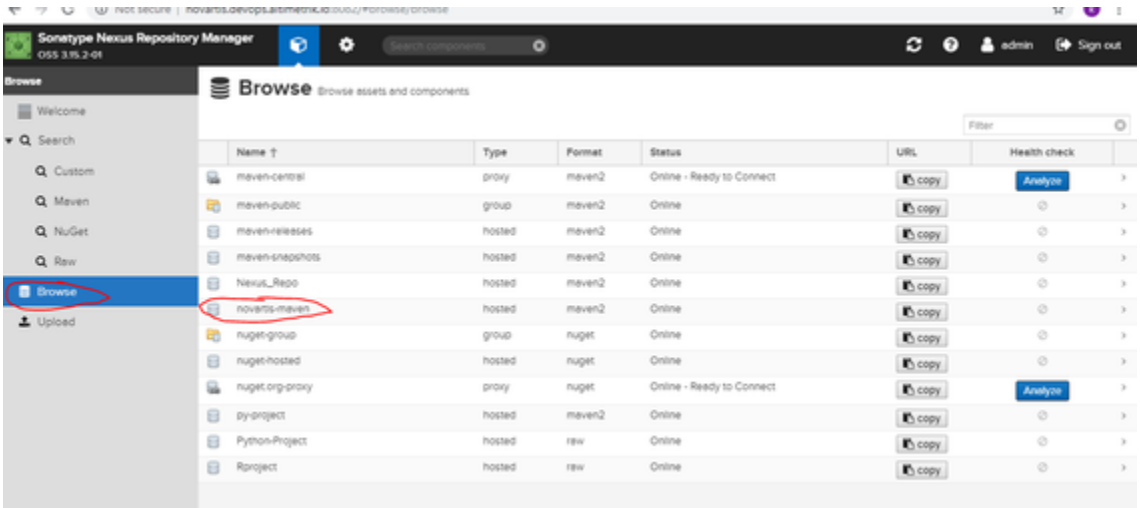
```
######################
To push artifacts

[gradle upload]
######################
```

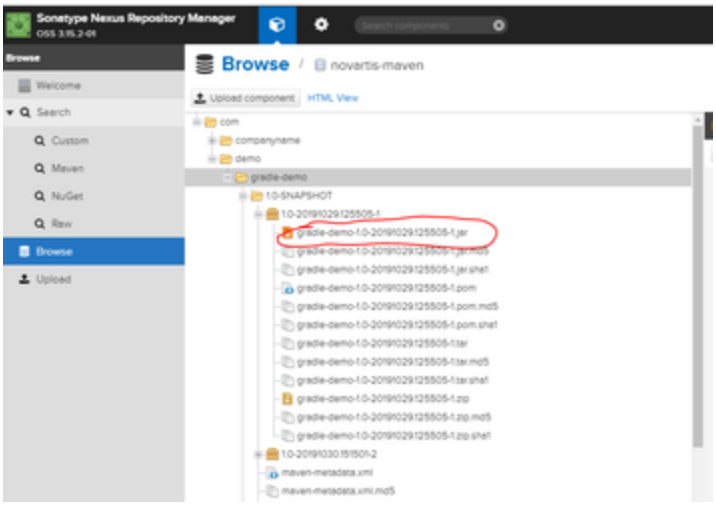
output:

```
[test-gradle] $ /var/jenkins_home/tools/hudson.plugins.gradle.GradleInstallation/gradle6/bin/gradle build sonarqube upload
Starting a Gradle Daemon (subsequent builds will be faster)
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :jar UP-TO-DATE
> Task :startScripts UP-TO-DATE
> Task :distTar UP-TO-DATE
> Task :distZip UP-TO-DATE
> Task :assemble UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test UP-TO-DATE
> Task :check UP-TO-DATE
> Task :build UP-TO-DATE
> Task :sonarqube
> Task :uploadArchives
```

Go to nexus repo

http://novartis.devops.altimetrik.io:8082/





1.4 Pipeline As a code

Pipeline as Code describes a set of features that allow Jenkins users to define pipelined job processes with code, stored and versioned in a source repository. These features allow Jenkins to discover, manage, and run jobs for multiple source repositories and branches.

To use *Pipeline as Code*, projects must contain a file named `Jenkinsfile` in the repository root, which contains a "Pipeline script."

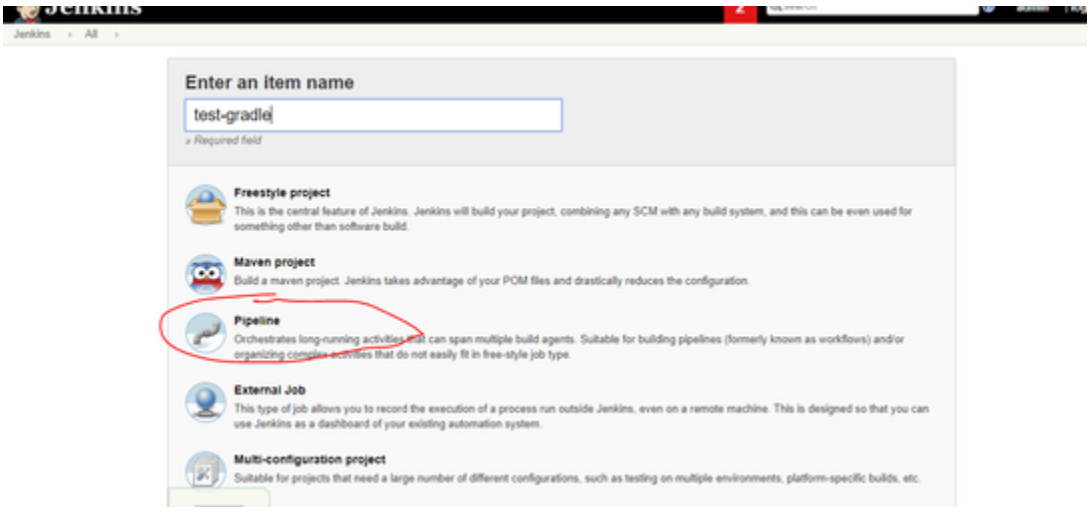
Jenkinsfile

The `Jenkinsfile` should contain a Pipeline script, specifying the steps to execute the job.

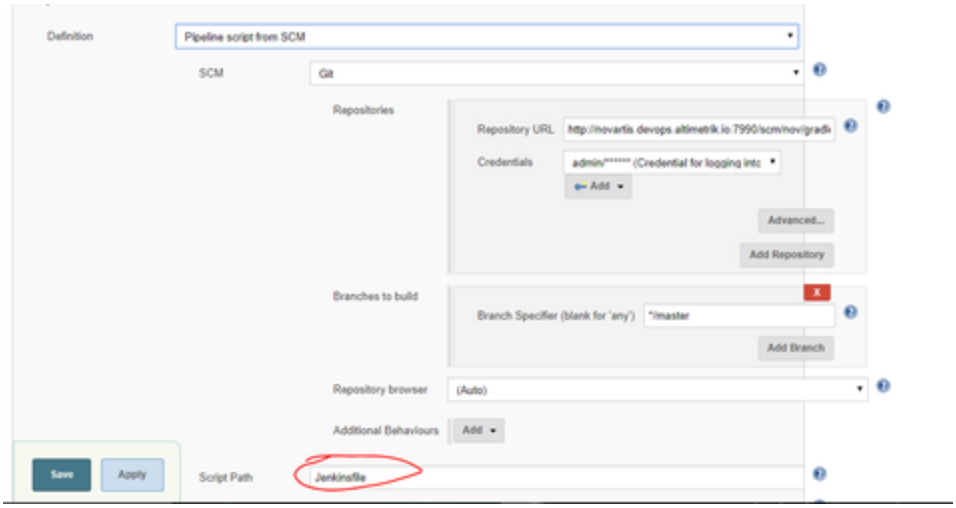
```
#########################################################
Jenkinsfile

node  {
stage("Poll SCM ") {
   git credentialsId: 'BitCred', url:
'http://novartis.devops.altimetrik.io:7990/scm/nov/gradle-demo.git'
  }
  stage("gradle build stage ") {
            sh "${tool name: 'gradle6', type: 'gradle'}/bin/gradle
build"

  }

  stage("gradle sonar ") {
            sh "${tool name: 'gradle6', type: 'gradle'}/bin/gradle
sonarqube"

  }
  stage("publish artifact") {
  sh "${tool name: 'gradle6', type: 'gradle'}/bin/gradle upload"
 }
}
#########################################################
```

1.5 Create pipeline job:

Goto Jenkins dashboard > Newitem> pipeline



Select pipeline as a script with scm



Run build pipeline