SonarScanner para Gradle

Issue Tracker - Source
Versão atual: **SonarScanner para Gradle 3.0**

O SonarScanner for Gradle oferece uma maneira fácil de iniciar a análise SonarCloud de um projeto Gradle.

A capacidade de executar a análise do SonarCloud por meio de uma tarefa regular do Gradle a torna disponível em qualquer lugar que o Gradle esteja disponível (serviço de CI etc.), sem a necessidade de baixar, configurar e manter manualmente uma instalação do SonarScanner. A compilação do Gradle já contém muitas das informações necessárias para que o SonarCloud analise um projeto com sucesso. Ao configurar a análise com base nessas informações, a necessidade de configuração manual é reduzida significativamente.

Pré-requisitos

- Gradle versões 2.14+
- Java 8 ou superior. A versão 11 é recomendada.

Bytecode criado pela compilação javac é necessário para análise Java, incluindo projetos Android.

Configure o scanner

A instalação é automática, mas certas propriedades globais ainda devem ser configuradas. Um bom lugar para configurar propriedades globais é `~/.gradle/gradle.properties`. Esteja ciente de que o scanner usa propriedades do sistema, portanto, todas as propriedades devem ser prefixadas por `systemProp`.

```
systemProp.sonar.host.url=https://sonarcloud.io

# Token generated from an account with 'Execute analysis' permission. It can also
# be set with the environment variable SONAR_TOKEN.
systemProp.sonar.login=<token>
```

Analisando

Primeiro, ative o scanner em sua construção. Para Gradle 2.1+, em `build.gradle`:

```
plugins {
  id "org.sonarqube" version "2.7.1"
}
```

Mais detalhes em https://plugins.gradle.org/plugin/org.sonarqube

Execute `gradle build sonarqube` e aguarde até que a construção seja concluída e, em seguida, abra a página da web indicada na parte inferior da saída do console. Agora você deve conseguir navegar pelos resultados da análise.

Analisando compilações de vários projetos

Para analisar uma hierarquia de projeto, aplique o plugin SonarQube ao projeto raiz da hierarquia. Normalmente (mas não necessariamente), este será o projeto raiz da compilação do Gradle. As informações relativas à análise como um todo devem ser configuradas no bloco de sonarqube deste projeto. Todas as propriedades definidas na linha de comando também se aplicam a este projeto.

```
// build.gradle
sonarqube {
    properties {
        property "sonar.sourceEncoding", "UTF-8"
    }
}
```

A configuração compartilhada entre subprojetos pode ser configurada em um bloco de subprojetos.

```
// build.gradle
subprojects {
    sonarqube {
        properties {
            property "sonar.sources", "src"
        }
    }
}
```

As informações específicas do projeto são configuradas no `sonarqube` bloco do projeto correspondente.

```
// build.gradle
project(":project1") {
    sonarqube {
        properties {
            property "sonar.branch", "Foo"
        }
    }}
```

Para ignorar a análise de um subprojeto específico, defina `sonarqube.skipProject` como verdadeiro.

```
// build.gradle
project(":project2") {
    sonarqube {
        skipProject = true
    }
}
```

Dependências de tarefas

Todas as tarefas que produzem resultados que devem ser incluídos na análise precisam ser executadas antes da execução da `sonarqube` tarefa. Normalmente, essas são tarefas de compilação, tarefas de teste e tarefas de cobertura de código. Para atender a essas necessidades, o plugin adiciona uma dependência tarefa `sonarqube` em `test` que é aplicado o plugin Java. Outras dependências de tarefas podem ser adicionadas conforme necessário. Por exemplo:

```
// build.gradle
project.tasks["sonarqube"].dependsOn "anotherTask"
```

Projeto de amostra

Um exemplo simples de trabalho está disponível neste URL para que você possa verificar se tudo está configurado corretamente em seu ambiente: https://github.com/SonarSource/sonar-scanning-examples/tree/master/sonarqube-scanner-gradle

Padrões de propriedade de análise

O SonarScanner for Gradle usa informações contidas no modelo de objeto do Gradle para fornecer padrões inteligentes para a maioria dos parâmetros de análisepadrão , conforme listado abaixo.

Padrões do Gradle para propriedades SonarCloud padrão:

Propriedade	Gradle padrão
`sonar.projectKey`	`[${project.group}:]${project.name}` para o módulo raiz; `<root module key>:<module path>` para submódulos
`sonar.projectName`	`${project.name}`
`sonar.projectDescription`	`${project.description}`
`sonar.projectVersion`	`${project.version}`

Propriedade	Gradle padrão
`sonar.projectBaseDir`	`${project.projectDir}`
`sonar.working.directory`	`${project.buildDir}/sonar`

Observe que padrões adicionais são fornecidos para projetos que têm o plugin java-base ou java aplicado:

Propriedade	Gradle padrão
`sonar.sourceEncoding`	`${project.compileJava.options.encoding}`
`sonar.java.source`	`${project.sourceCompatibility}`
`sonar.java.target`	`${project.targetCompatibility}`
`sonar.sources`	`${sourceSets.main.allSource.srcDirs}` (filtrado para incluir apenas diretórios existentes)
`sonar.tests`	`${sourceSets.test.allSource.srcDirs}` (filtrado para incluir apenas diretórios existentes)
`sonar.java.binaries`	`${sourceSets.main.output.classesDir}`
`sonar.java.libraries`	`${sourceSets.main.compileClasspath}` (filtragem para incluir apenas arquivos; rt.jar e jfxrt.jar adicionados, se necessário)
`sonar.java.test.binaries`	`${sourceSets.test.output.classeDir}`
`sonar.java.test.libraries`	`${sourceSets.test.compileClasspath}` (filtragem para incluir apenas arquivos; rt.jar e jfxrt.jar adicionados, se necessário)
`sonar.junit.reportPaths`	`${test.testResultsDir}` (se o diretório existir)

Os projetos do Groovy obtêm todos os padrões Java, mais:

Propriedade	Gradle padrão
`sonar.groovy.binaries`	`${sourceSets.main.output.classesDir}`

Padrões adicionais quando o plugin JaCoCo é aplicado

Propriedade	Gradle padrão
`sonar.jacoco.reportPaths`	`${jacoco.destinationFile}`
`sonar.groovy.jacoco.reportPath`	`${jacoco.destinationFile}`

Padrões adicionais para projetos Android
(`com.android.application` , `com.android.library` ou `com.android.test`) Por padrão, a primeira variante do tipo "depuração" será usada para configurar a análise. Você pode substituir o nome da variante a ser usada usando o parâmetro 'androidVariant':

```
build.gradle
sonarqube {
    androidVariant 'fullDebug'
}
```

Propriedade	Gradle padrão
`sonar.sources` (para variantes sem teste)	`${variant.sourcesets.map}` (ManifestFile / CDirectories / AidlDirectories / AssetsDirectories / CppDirectories / JavaDirectories / RenderscriptDirectories / ResDirectories / ResourcesDirectories)
`sonar.tests` (para variantes de teste)	`${variant.sourcesets.map}` (ManifestFile / CDirectories / AidlDirectories / AssetsDirectories / CppDirectories / JavaDirectories / RenderscriptDirectories / ResDirectories / ResourcesDirectories)
`sonar.java[.test].binaries`	`${variant.destinationDir}`
`sonar.java[.test].libraries`	`${variant.javaCompile.classpath} + ${bootclasspath}`
`sonar.java.source`	`${variant.javaCompile.sourceCompatibility}`
`sonar.java.target`	`${variant.javaCompile.targetCompatibility}`

Transmitindo propriedades manuais / substituindo padrões

O SonarScanner for Gradle adiciona uma extensão SonarQubeExtension ao projeto e seus subprojetos, o que permite configurar / substituir as propriedades de análise.

```
// in build.gradle
sonarqube {
    properties {
        property "sonar.exclusions", "**/*Generated.java"
    }
}
```

As propriedades do SonarCloud também podem ser definidas na linha de comando ou definindo uma propriedade do sistema com o mesmo nome da propriedade SonarCloud em questão. Isso pode ser útil ao lidar com informações confidenciais (por exemplo, credenciais), informações de ambiente ou para configuração ad-hoc.

```
gradle sonarqube -Dsonar.host.url=http://sonar.mycompany.com -Dsonar.verbose=true
```

Embora certamente útil às vezes, recomendamos manter a maior parte da configuração em um script de construção (com versão), prontamente disponível para todos. Um valor de propriedade SonarCloud definido por meio de uma propriedade do sistema substitui qualquer valor definido em um script de construção (para a mesma propriedade). Ao analisar uma hierarquia de projeto, os valores definidos por meio das propriedades do sistema se aplicam ao projeto raiz da hierarquia analisada. Cada propriedade do sistema começando com `sonar.` será levada em consideração.

Analisando conjuntos de fontes personalizadas

Por padrão, o SonarScanner for Gradle passa o conjunto principal de fontes do projeto como fontes de produção e o conjunto de fontes de teste do projeto como fontes de teste. Isso funciona independentemente do layout do diretório de origem do projeto. Conjuntos de fonte adicionais podem ser adicionados conforme necessário.

```
// build.gradle
sonarqube {
    properties {
        properties["sonar.sources"] += sourceSets.custom.allSource.srcDirs
        properties["sonar.tests"] += sourceSets.integTest.allSource.srcDirs
    }
}
```

Tópicos avançados

Mais sobre configuração de propriedades

Vamos examinar o `sonarqube.properties {}` bloco mais de perto . Como já vimos nos exemplos, o `property()` método permite definir novas propriedades ou substituir as existentes. Além disso, todas as propriedades que foram configuradas até este ponto, incluindo todas as propriedades pré-configuradas pelo Gradle, estão disponíveis por meio do acessador de propriedades.

As entradas no mapa de propriedades podem ser lidas e gravadas com a sintaxe Groovy usual. Para facilitar sua manipulação, os valores ainda possuem seu tipo “idiomático” (Arquivo, Lista, etc.). Depois que o bloco sonarProperties foi avaliado, os valores são convertidos em Strings da seguinte maneira: Os valores da coleção são (recursivamente) convertidos em Strings separados por vírgula e todos os outros valores são convertidos chamando seus `toString()` métodos.

Como o `sonarProperties` bloco é avaliado lentamente, as propriedades do modelo de objeto do Gradle podem ser referenciadas com segurança de dentro do bloco, sem medo de que ainda não tenham sido definidas.