Continuous Integration with Jenkins

Last updated: 2 November 2018

Contents

About this document

This document describes a full scripted Continuous Integration Multibranch Pipeline in Jenkins as Continuous Integration server. GitHub is used as source code repository, SonarQube for static code analysis and code coverage (using "JaCoCo"), and JFrog Artifactory as binary repository.

The whole Jenkins Multibranch Pipeline chain requires 3 servers.

1. A Jenkins server
2. A SonarQube server
3. A JFrog Artifactory server

At least the Jenkins server must be accessible from the public internet, if you want to use GitHub Webhooks. However, this document also describes a workaround to expose private servers to the public internet via "ngrok", if the Jenkins server is not publicly accessible.

Source code download

The source code for the example Jenkins Multibranch Pipeline is available from GitHub:
https://github.com/brunobosshard/ci-jenkins.git

Key terms

Static code analysis

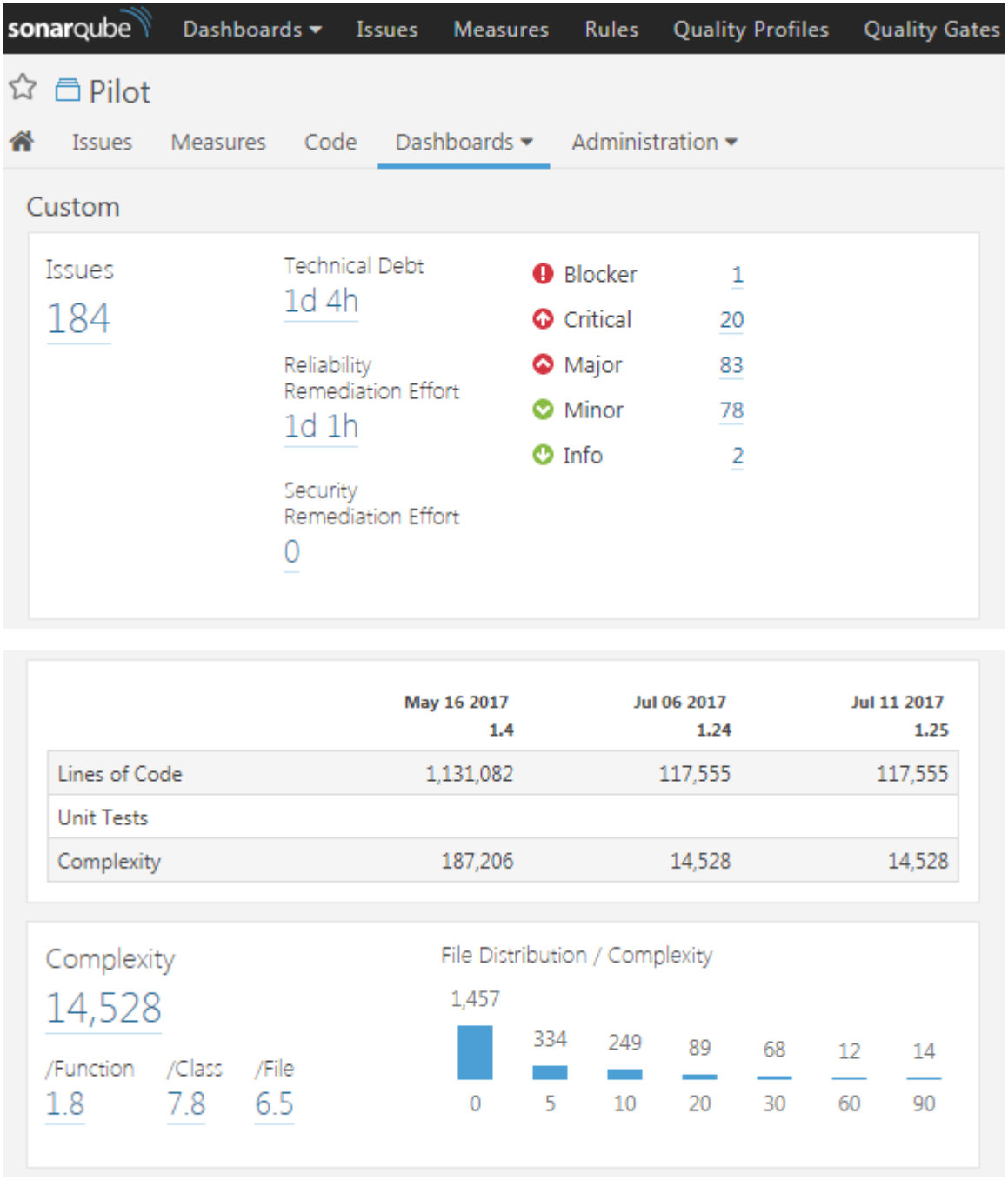
Static code analysis, also commonly called white-box testing, is a form of software testing that looks for the structural qualities of the program code. For example, it answers how robust or maintainable the code is. Static code analysis is performed without actually executing programs. It is different from functional testing, which looks into the functional aspects of software, and is dynamic.

Static code analysis is the evaluation of software's inner structures. For example, is there a piece of program code used repetitively? Does the code contain lots of commented lines? How complex is the code? Using the metrics defined, an analysis report is generated that shows the code quality regarding maintainability. It doesn't question the code's functionality.

Some of the static code analysis tools like SonarQube come with a dashboard, which shows various metrics and statistics of each run.

Usually, as part of Continuous Integration (CI), static code analysis is triggered every time a build runs. Static code analysis can also be included before a developer tries to check-in his code. This excludes code of low quality right at the initial stage.

The following screenshots illustrate a static code analysis report using SonarQube:



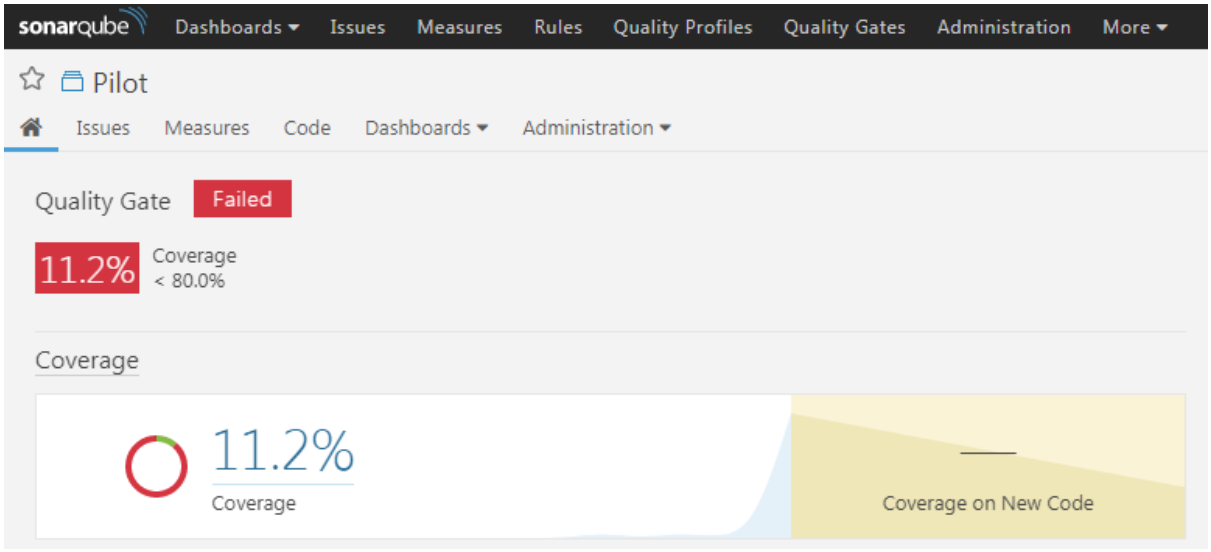
Code coverage

Code coverage is the amount of code (in percentage) that is covered by test cases. The metrics that you might see in your coverage reports could be:

Type of coverage	Description
Function	The number of functions called out of the total number of functions defined
Statement	The number of statements in the program that are truly called out of the total number
Branches	The number of branches of the control structures executed
Condition	The number of Boolean sub-expressions that are being tested for a true and a false value

Line	The number of lines of source code that are being tested out of the total number of lines present inside the code

This coverage percentage is calculated by dividing the number of items tested by the number of items found. The following screenshot illustrates a code coverage report from SonarQube:



This document describes and uses "JaCoCo" as code coverage tool.

Binary repository

A binary repository tool such as JFrog Artifactory is a Version Control System for binary files. This should not be confused with a Version Control System for source code, such as Git.

Git is responsible for versioning the source code, while Artifactory is for other (binary) files, such as .rar, .war, .exe, .msi, and so on. Along with managing built artifacts, a binary repository tool can also manage 3rd-party binaries that are required for a build. For example, the Apache Maven plugin always downloads the plugins required to build the program code. Rather than downloading the plugins again and again from the public internet, they can also be managed using a private binary repository tool.

Difference between Unit Tests and Integration Tests

Unit Tests are usually written by developers to test their program code. Ideally, all program code should be covered by Unit Tests. This will automatically be the case, if developers use a "test first" approach like the Test-Driven Development (TDD) methodology. Code coverage analysis tools like "JaCoCo" can be used to report on the completeness of code coverage.

Integration Tests are often written by testers or "Software Development Engineers in Test" (SDET's). Integration Tests usually test the integration between different parts of the application or system, or test whole business processes (end-to-end scenarios) from an end-user perspective. They often use tools like Selenium for web automation, or Appium for mobile apps.

Maven Surefire is used for Unit Tests, while Maven Failsafe is used for Integration Tests. The usual naming convention for Unit Test classes is "name of class + Test" while the naming convention for Integration Test classes is "name of class + IT".

By default, the Maven Surefire Plugin will automatically include all test classes with the following wildcard patterns:

- "`**/Test*.java`" - includes all of its subdirectories and all Java filenames that start with "`Test`".
- "`**/*Test.java`" - includes all of its subdirectories and all Java filenames that end with "`Test`".
- "`**/*Tests.java`" - includes all of its subdirectories and all Java filenames that end with "`Tests`".
- "`**/*TestCase.java`" - includes all of its subdirectories and all Java filenames that end with "`TestCase`".

By default, the Maven Failsafe Plugin will automatically include all test classes with the following wildcard patterns:

- "`**/IT*.java`" - includes all of its subdirectories and all Java filenames that start with "`IT`".
- "`**/*IT.java`" - includes all of its subdirectories and all Java filenames that end with "`IT`".
- "`**/*ITCase.java`" - includes all of its subdirectories and all Java filenames that end with "`ITCase`".

Jenkins standalone server installation in Linux

Install Oracle Java JDK 1.8 (Standard Edition)

1. **sudo add-apt-repository ppa:webupd8team/java**
2. **sudo apt update**
3. **sudo apt install oracle-java8-installer**
4. Set Java environment variables:
   sudo apt install oracle-java8-set-default
5. Check the installed Java version:
   java -version

Install Apache Maven 3

1. **sudo apt-get install maven**
2. Check the installed Maven version:
   mvn -version

Install Git

1. **sudo apt-get install git-core**
2. Confirm the installation:
   git --version
3. Configure the Git user name:
   git config --global user.name "*testuser*"
4. Configure the Git user email address:
   git config --global user.email "*testuser*@*example.com*"
5. Verify the configuration changes:
   git config --list

Install Jenkins standalone on Ubuntu Linux

The following commands will install the latest stable version of Jenkins on Ubuntu Linux flavours (including Mate, Lubuntu etc.):

1. Add the repository key to the system (this should result in "OK" as output):
   wget --no-check-certificate -q -O - https://pkg.jenkins.io/debian-stable/jenkins-ci.org.key | sudo apt-key add -
2. Append the package repository address:
   echo deb http://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list
3. **sudo apt-get update**
4. **sudo apt-get install jenkins**

Jenkins is now ready for use. By default, the Jenkins service runs on port 8080. To access Jenkins, type "http://localhost:8080/" or "http://<Jenkins server IP address>:8080/" in a browser .

Optional: Install Docker on the Jenkins server

1. If you are using a Debian based Linux, follow the instructions in this document to install Docker Community Edition on Ubuntu Linux: https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-docker-ce
2. Linux requires adding the current user to the "`docker`" group.
    a. The "`docker`" group can be created with this command. If the "`docker`" group already exists, then a message will say so, but it will do no harm:
    `sudo groupadd docker`
    b. The following command will add the user to the "`docker`" group. After this command, it is recommended to completely log out of the account and log back in (if in doubt, reboot!):
    `sudo usermod -a -G docker $USER`
3. The correct installation of Docker can be verified by running this command. On the first execution of this command, the image "`hello-world`" will automatically be pulled:
`docker run hello-world`

Starting, stopping, and restarting Jenkins on Ubuntu Linux

Jenkins starts automatically by default. Here are the commands to start, stop, restart, and check the status of the Jenkins service:

Starting Jenkins

`sudo systemctl start Jenkins`

Stopping Jenkins

`sudo systemctl stop jenkins`

Restarting Jenkins

`sudo systemctl restart jenkins`

Check Jenkins status

`sudo systemctl status jenkins`

The Jenkins Getting Started wizard

When you access Jenkins for the first time, you are presented with the Getting Started wizard.

You are asked to unlock Jenkins using a secret initial admin password. This password is stored inside the file "`initialAdminPassword`", which is located inside the "jenkins_home" directory: ("`/var/lib/jenkins/secrets/`").

The wizard asks for the plugins to be installed. If unsure, just accept "Install suggested plugins".

The wizard will then ask for a "First Admin User" (in addition to the built-in Jenkins admin account). An often used account name is "`jenkins_admin`". Choose a suitable password.

Clicking on the "Save and Continue" will show the Jenkins URL.

Clicking on the "Save and Finish" button will finish the wizard and allow you to click on a button to start Jenkins.

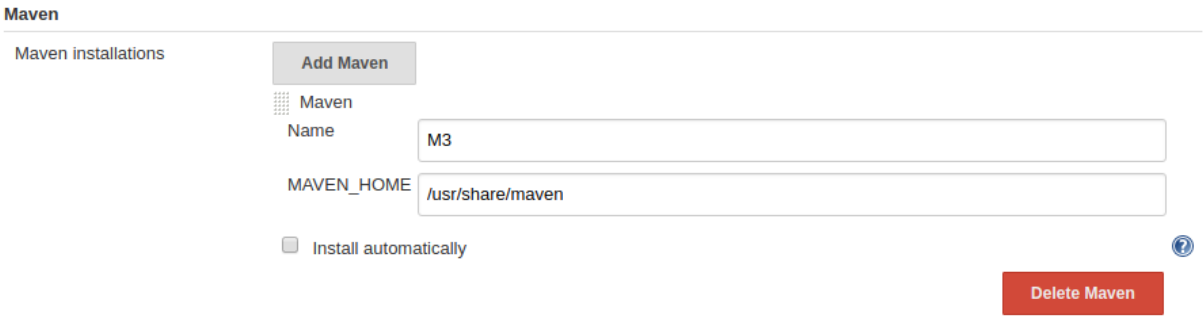
Note: If you just get a blank page after login, then it might be required to restart the Jenkins service (or alternatively to reboot the machine).

Global Tool Configuration

This is the place where you configure tools that you think will be used globally across projects in Jenkins, for example Java, Maven, Git, and so on.

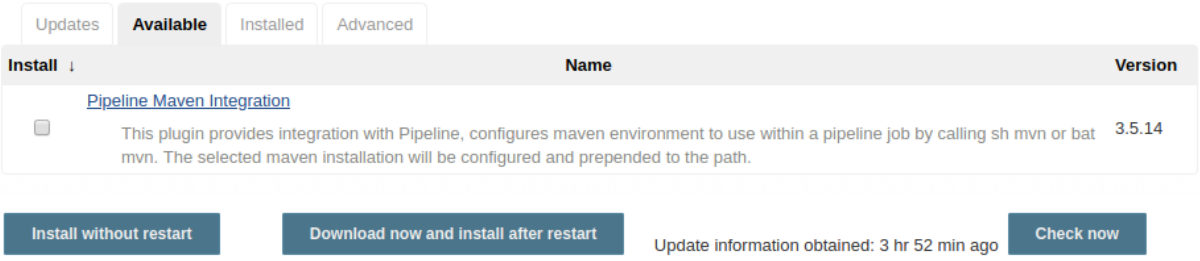
On the Jenkins home page, click on "Manage Jenkins", and then click on "Global Tool Configuration".

The following will add a Maven installation called "M3". This name "M3" can then be used in pipeline scripts to refer to this particular installation.



Recommended Plugins

The example Jenkins pipeline requires the "Pipeline Maven Integration" plugin to be installed.



Make the Jenkins URL is accessible from the internet

GitHub Webhooks require the Jenkins server to be accessible from the internet. If the Jenkins server is not on a public IP address or publicly available DNS entry, then exposing the localhost server to the internet in required.

An application / service named "ngrok" can be used to achieve this feat. The free (unregistered) service will keep the session open for 8 hours, but there are also paid plans available, which will work for unlimited time and offer more flexibility with features like reserved names. Please check https://ngrok.com for details.

In Linux, perform the following steps to make your Jenkins server accessible over the internet using "ngrok":

1. Log in to the Jenkins server machine.

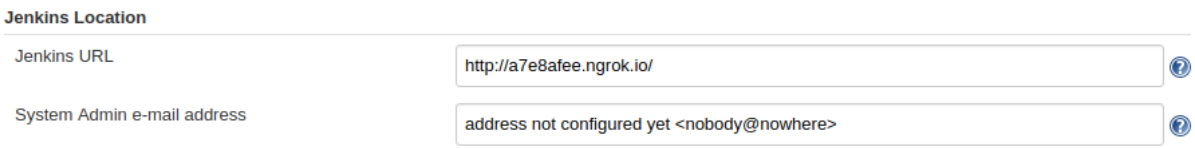
2. Download the appropriate "Linux" version of the "ngrok" application from https://ngrok.com/download.
3. Extract it using the unzip command:
   `unzip /path/to/ngrok.zip`
4. Make "ngrok" executable:
   `chmod +x /path/to/ngrok`
5. To run "ngrok" on Linux, execute the following command:
   `./ngrok http 8080`
   Alternatively, run the following command:
   `nohup ./ngrok http 8080 &`
6. You should see a similar output, as shown as follows. The highlighted text is the public URL of localhost:8080. "ngrok" automatically provides both http and https addresses:

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Session Expires               7 hours, 59 minutes
Version                       2.2.8
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://a7e8afee.ngrok.io -> localhost:8080
Forwarding                    https://a7e8afee.ngrok.io -> localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

7. Copy the public (http) URL (in the example "**`http://a7e8afee.ngrok.io`**").
8. Log in to your Jenkins server. From the Jenkins dashboard, navigate to "Manage Jenkins" and then to "Configure System".
9. On the Jenkins configuration page, scroll all the way down to the "Jenkins Location" section and add the public URL generated using "ngrok" inside the "Jenkins URL" field.
10. Click on the "Save" button to save the settings.
11. You will now be able to access your Jenkins server using the public URL over the internet.
12. While creating Webhooks on GitHub, use the public (http) URL generated using "ngrok".

Set the Jenkins URL

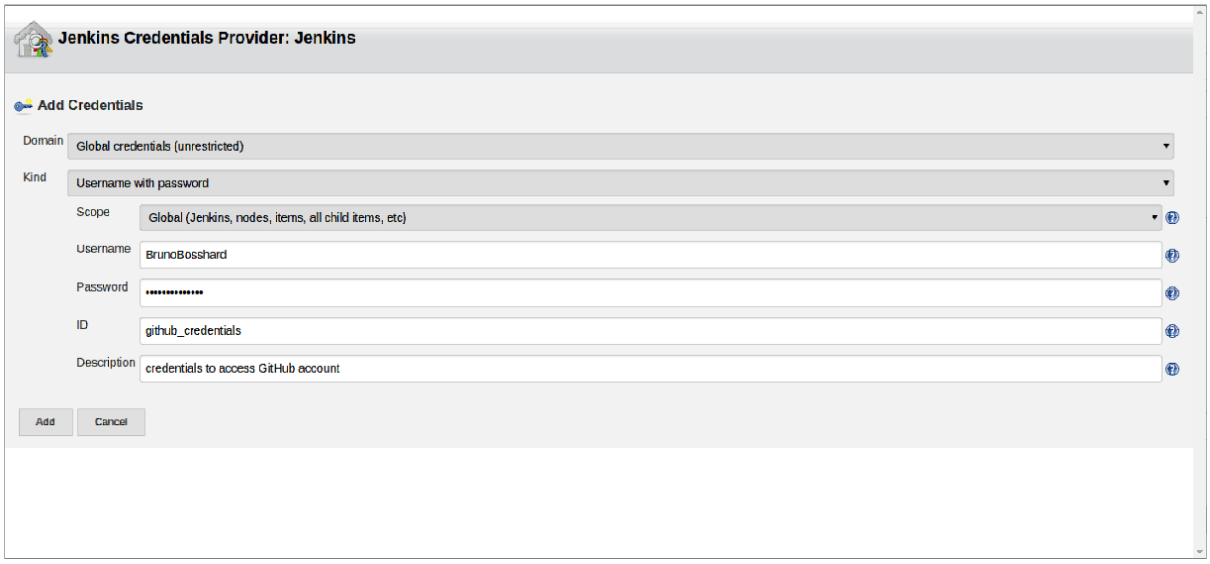
The public URL must be set in Jenkins under "Manage Jenkins", followed by "Configure System". If you are using "ngrok", then you will have just done this in the previous paragraph. If you use your own publicly available IP number or DNS name, then enter your IP number or DNS name instead of the "…ngrok.io" URL of the following screenshot:



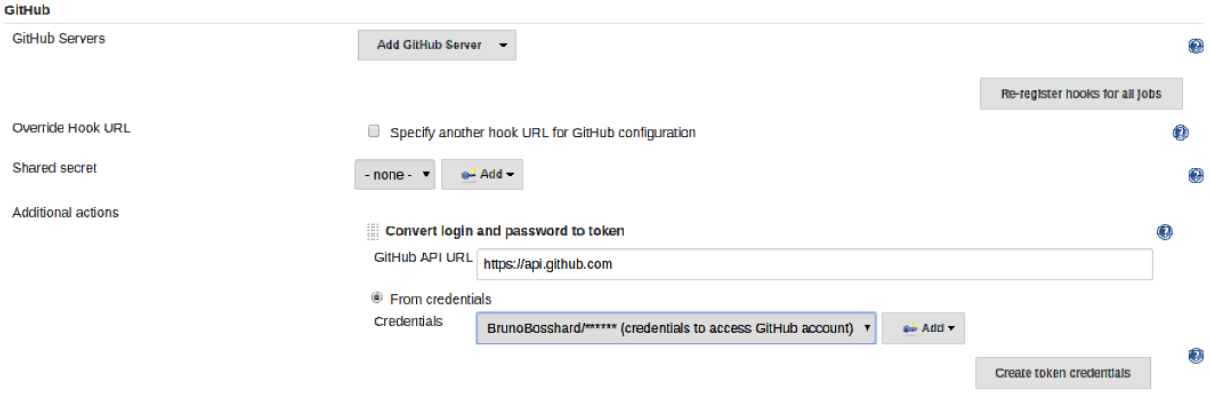
Connect Jenkins to GitHub

1. From the Jenkins home page, click on "Manage Jenkins", then on "Configure System".
2. In the GitHub section, click on the "Advanced..." button at the bottom of the "GitHub" section.

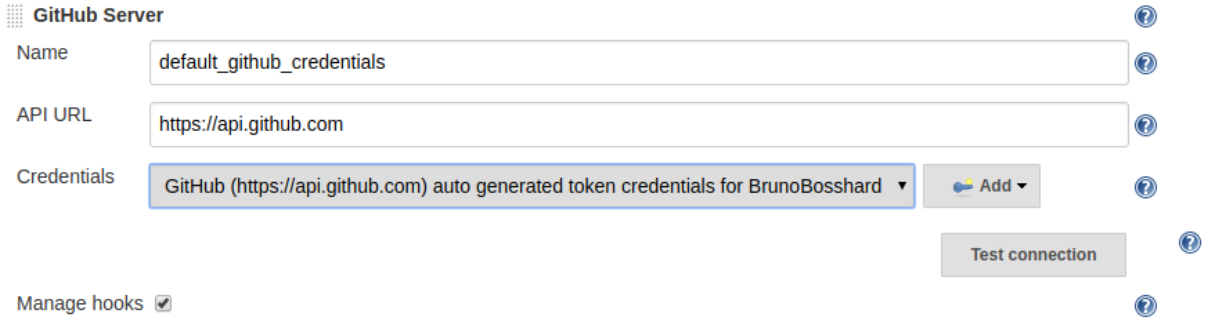
3. Click on the "Manage additional GitHub actions" button (with the "Convert login and password to token" sub-selection). You will see the "API URL" automatically prefilled to "`https://api.github.com`", which is correct.
4. Click on the "Add" button (in the "From credentials" selection).
5. Fill in the information with your GitHub username, your GitHub password, an ID (name) of your choice and a description of your choice:



6. Click on the "Add" button to close the dialog.
7. Select the credential that you have just created in the "Credentials" drop-down and click on the "Create token credentials" button:



8. You should see a message that your credentials have been created.
9. Click on the "Add GitHub Server" button (with the "GitHub Server" sub-selection) at the start of the GitHub section.
10. Fill in the information required. Give the GitHub Server connection a "Name" and select your credentials from the "Credentials" drop down. It should look like this:



11. Click on the "Test connection" button. You should get a message that your credentials have been verified.
12. Click on the "Save" button at the very bottom of the page to save your settings.

SonarQube standalone server installation in Linux

Install Oracle Java JDK 1.8 (Standard Edition)

1. **`sudo add-apt-repository ppa:webupd8team/java`**
2. **`sudo apt update`**
3. **`sudo apt install oracle-java8-installer`**
4. Set Java environment variables:
   `sudo apt install oracle-java8-set-default`
5. Check the installed Java version:
   `java -version`

Installing SonarQube

1. Download the latest Long Term Support (LTS) Community Edition version of the SonarQube from https://www.sonarqube.org/downloads/ .
2. Unzip the SonarQube ZIP package that you have just downloaded.
3. Rename the extracted folder to just "`sonarqube`" (without version number).
4. Move the "`sonarqube`" folder to the "`/opt`" folder:
   `sudo mv sonarqube /opt`
5. Give your user account permissions to make changes in this directory:
   `sudo chown -R username:username /opt/sonarqube`

Note: SonarQube sometimes runs into permission issues. It might be (temporarily) required to suspend the security of the "`sonarqube`" folder for debugging purposes, for example with "**`sudo chmod -R 777 /opt/sonarqube`**".

Start and stop SonarQube

1. Move to the correct SonarQube bin folder:
   `cd /opt/sonarqube/bin/linux-x86-64/`
2. Run the "`sonar.sh`" shell script to start SonarQube. This should result in a response of "`Starting SonarQube...`" and "`Started SonarQube.`":
   `./sonar.sh start`

The default URL for SonarQube is https://localhost:9000

The command to stop SonarQube is:
`./sonar.sh stop`

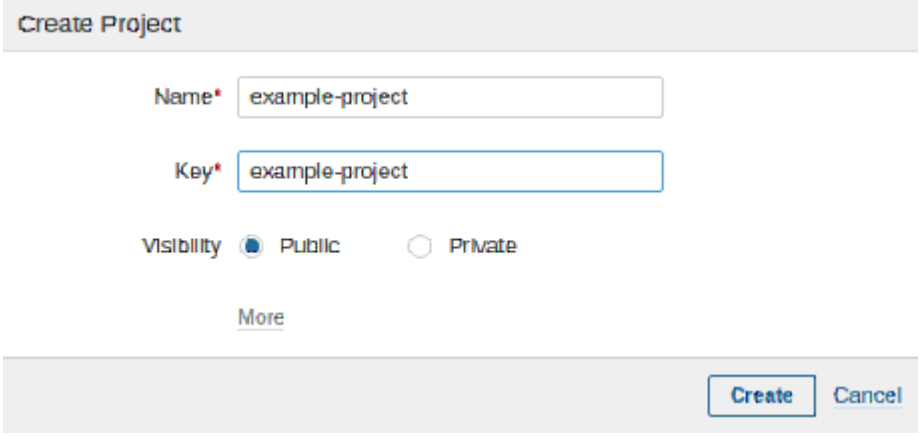
Resetting the default credentials and generating a token

1. Open SonarQube in a browser and log in as the default admin user (account: "admin", password: "admin").
2. Skip the tutorial, if it is displayed.
3. Click on "Administration", then "Security" with the sub-selection "Users".
4. Click on "Change Password" (the lock icon) and change the password of the admin account to something more secure.
5. Click on "Update tokens" (the list icon) and generate a new token for a meaningful name (for example "jenkins-sonarqube". Save the generated token for later use.

Creating a project inside SonarQube

This project is used to display the example static code analysis:

1. From the SonarQube dashboard, click on "Administration", then "Projects", then "Management".
2. Click on the "Create Project" button.
3. Create a project with the Name "example-project" and the Key "example-project":

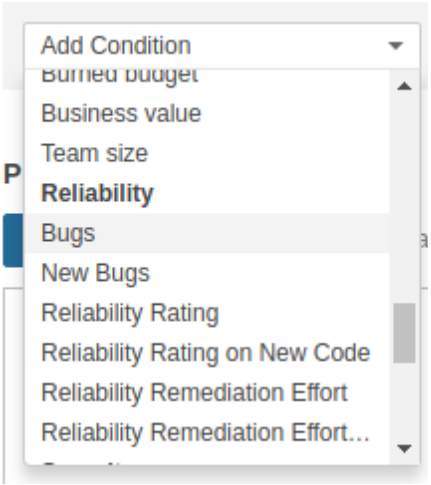


Creating quality gates

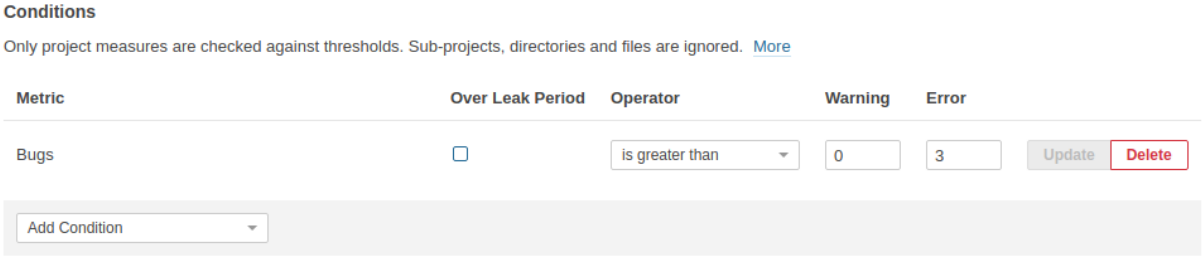
When the example Jenkins pipeline runs, it will execute the quality profiles and the quality gate. If the quality gate check passes successfully, then the Jenkins pipeline continues, but if it fails then the Jenkins pipeline is aborted. Nevertheless, the analysis still happens.

The following is just an example quality gate. In a production project, you would probably not tolerate any known bugs.

1. Click on the "Quality Gates" link from the menu bar.
2. Click on the "Create" button.
3. Give your Quality Gate a name (for example "example-quality-gate") and click the "Create" button.
4. From the "Add Condition" dropdown, select a condition, for example "Bugs":



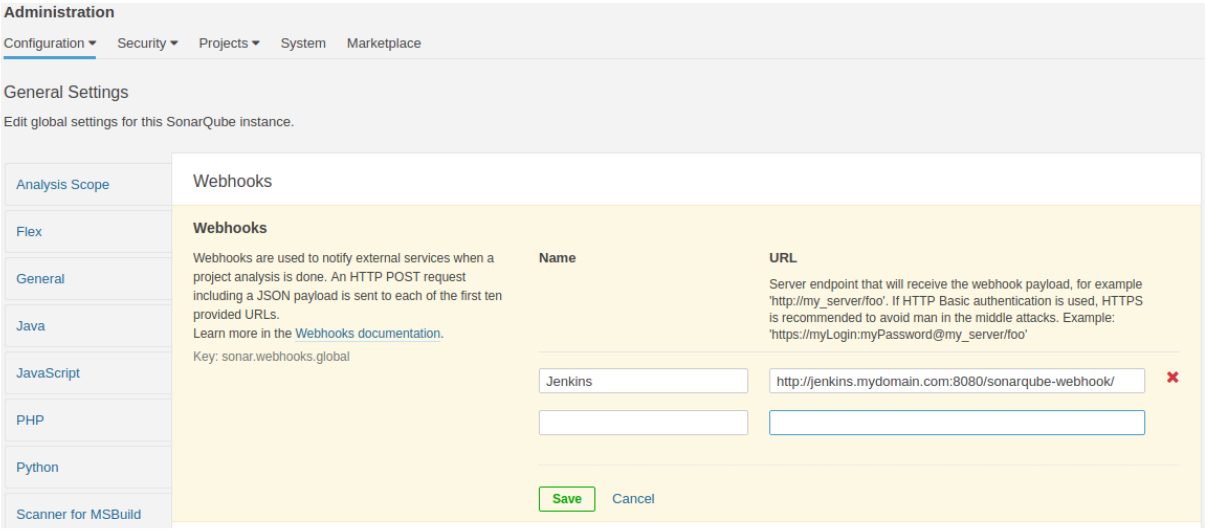
5. The following screenshot shows an example. If it's not more than 3 bugs it's a WARNING, and if it's greater than 3 bugs, it's an ERROR (fail):



6. Next, let's make sure that the example project that we created earlier in SonarQube uses our newly created quality gate. To do so, from the SonarQube dashboard, click on "Administration", then "Projects", then "Management".
7. Click on the sample project that you created earlier.
8. Click on "Administration", then "Quality Gate".
9. In the dropdown selection, choose the Quality Gate that you have just created.

Create a SonarQube Webhook entry for Jenkins

1. From the SonarQube dashboard, click on "Administration".
2. Under the tab "Configuration" (General Settings), click on the link "Webhooks".
3. Create a Webhook entry (for example with the name "Jenkins") in the format "`http://`<your Jenkins instance>`/sonarqube-webhook/`". Note that the final forward slash character ("`/`") is required:



4. Click on the "Save" button.

Optional: Importing JaCoCo code coverage reports into SonarQube

The "JaCoCo" plugin imports "JaCoCo" code coverage XML reports. The settings for the reports are defined in Maven and the report generation in triggered through the Jenkinsfile (the Jenkinsfile is explained later in this document).

1. From the SonarQube dashboard, click on "Administration".
2. Click on "Marketplace".
3. Scroll down the list of available plugin until you find "JaCoCo".

4. Click the "Install" button to install the "JaCoCo" plugin, if it hasn't already been installed.
5. Restart SonarQube to activate the "JaCoCo" plugin.

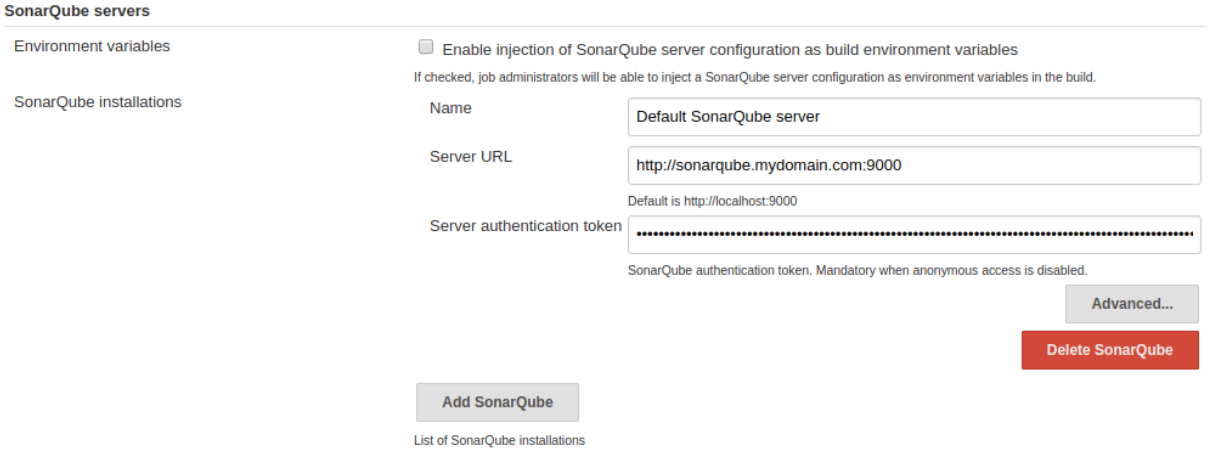
Installing the SonarQube plugin in Jenkins

Follow these steps to install the SonarQube plugin in Jenkins:

1. From the Jenkins dashboard, click on "Manage Jenkins", then on "Manage Plugins", then on the "Available" tab.
2. Type "`SonarQube`" in the Filter field.
3. From the available SonarQube plugins, select (tick) "SonarQube Scanner for Jenkins" from the list and then click on the "Install without restart" button.
4. Restart Jenkins if needed.

Configure the SonarQube Scanner plugin in Jenkins

1. From the Jenkins dashboard, click "Manage Jenkins", then "Configure System".
2. ON the "Configure System" page, scroll down all the way to the "SonarQube servers" section.
3. Under the "SonarQube servers" section, click on the "Add SonarQube" button.
4. Give your SonarQube server a name in the "Name" field.
5. Enter the SonarQube server URL in the "Server URL" field.
6. Add the previously created and saved SonarQube token in the "Server authentication token" field. Your settings should look similar to this screenshot:



7. Click on the "Save" button at the end of page.

JFrog Artifactory standalone server installation in Linux

Install Oracle Java JDK 1.8 (Standard Edition)

1. **`sudo add-apt-repository ppa:webupd8team/java`**
2. **`sudo apt update`**
3. **`sudo apt install oracle-java8-installer`**
4. Set Java environment variables:
   `sudo apt install oracle-java8-set-default`
5. Check the installed Java version:
   `java -version`

Installing JFrog Artifactory Open Source version

1. If you are working with a Debian-based Linux (such as Ubuntu), download the Debian package of the Open Source version of Artifactory from https://jfrog.com/open-source/. If not, then download the ZIP version and install it in the same way as you installed SonarQube.
2. On a Debian-based Linux (such as Ubuntu), use your Linux package installer to install the downloaded package (jfrog-artifactory-oss).

Only execute the following 3 steps if you are not using the Debian installer:

1. Give your user account permissions to make changes in the "artifactory" directory, for example:
   `sudo chown -R username:username /opt/artifactory`
2. Move to the correct "artifactory" bin folder, for example:
   `cd /opt/artifactory/bin`
3. Run the "`installService.sh`" shell script to install Artifactory:
   `sudo ./installService.sh`

Note: Artifactory sometimes runs into permission issues. It might be (temporarily) required to suspend the security of the "artifactory" folder for debugging purposes, for example with "`sudo chmod -R 777 /opt/artifactory`" (or "`sudo chmod -R 777 /opt/jfrog`").

Reboot the machine. Artifactory might establish and start a service automatically on reboot, so try if Artifactory is already running after a reboot (in this case, you can skip the following section about starting and stopping Artifactory). The default URL for Artifactory is https://localhost:8081

Start and stop JFrog Artifactory

You can start Artifactory by using any one of the following 3 commands:

```
sudo service artifactory start
sudo /etc/init.d/artifactory start
sudo systemctl start artifactory
```

You can check the Artifactory installation by executing any one of the following 3 commands:

```
service artifactory check
```

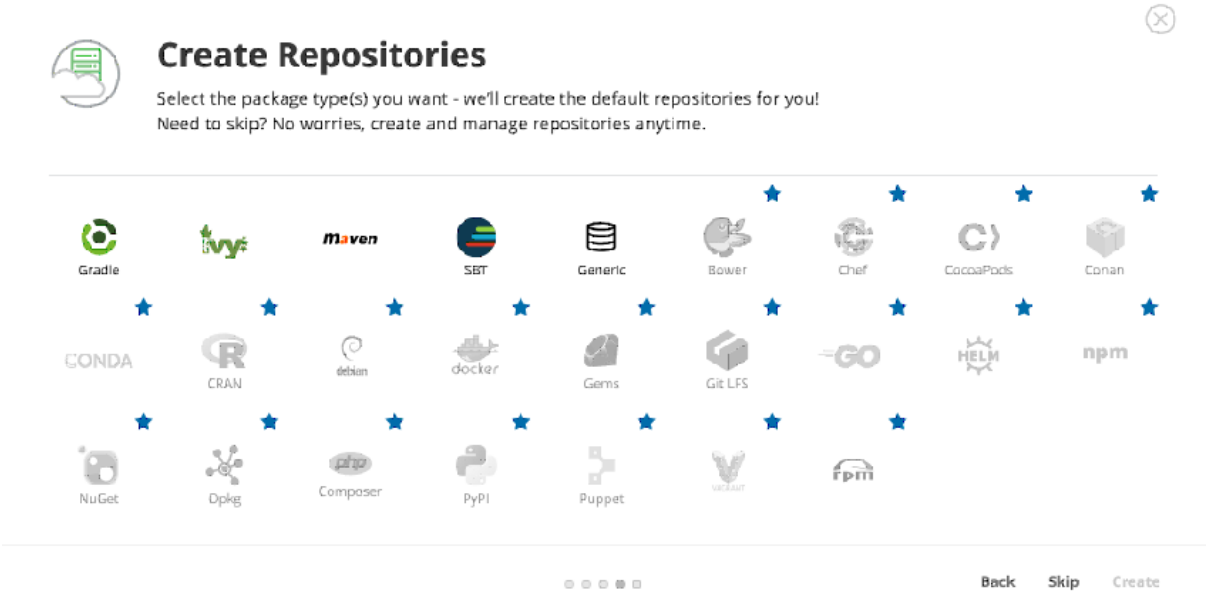
```
/etc/init.d/artifactory check
sudo ./artifactoryctl check
```

You can stop Artifactory by using any one of the following 3 commands:

```
sudo service artifactory stop
sudo /etc/init.d/artifactory stop
sudo systemctl stop artifactory
```

JFrog Artifactory setup wizard

On the first start of Artifactory, a wizard asks you to change the password for the admin account. It also allows you to set a proxy server (you can skip this step, if you don't use a proxy server), and it allows you to choose the type of repositories that you require (skip this step if unsure, you can always change it later):



Generate a JFrog Artifactory API key

1. Log in as admin user.
2. From the Artifactory dashboard, click on "Welcome, admin" and select "Edit Profile".
3. Enter your current password in the "Current Password" field and press the "Unlock" button.
4. Under "Email address", enter your email address.
5. Under the "Authentication Settings" section, click on the "Generate" key (gear logo) to generate a new API key.

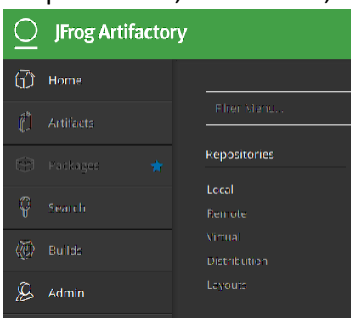
6. Copy the generated API key by clicking on the copy icon:

   Save this API key for later use.
7. Once done, click on the "Save" button.

Creating a repository in JFrog Artifactory

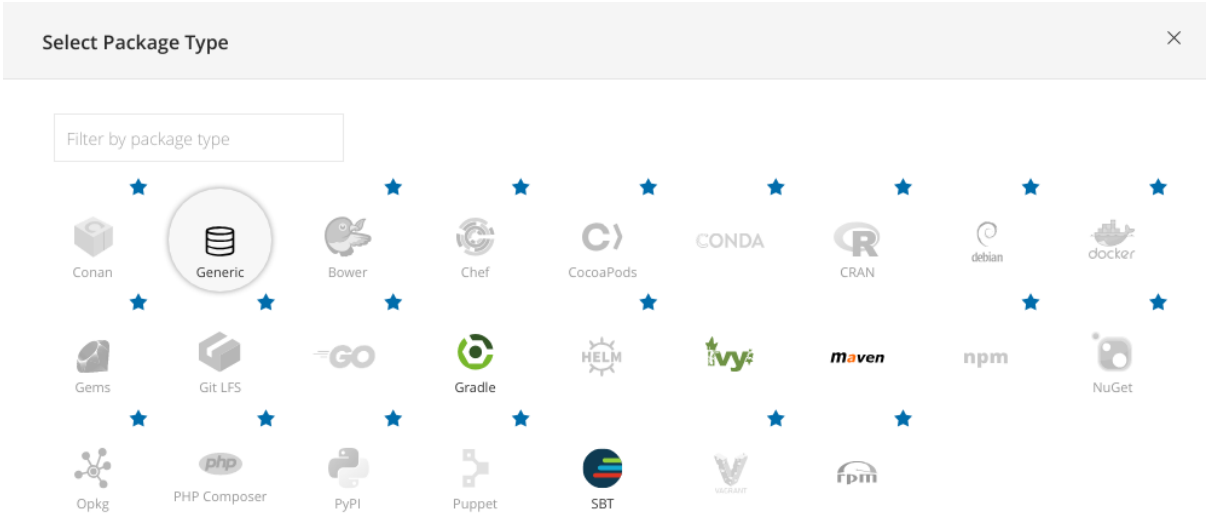
This will create a genetic repository inside Artifactory, which will be used to store build artifacts.

1. From the Artifactory dashboard, on the left-hand side menu, click on "Admin", "Repositories", and "Local", as shown in the following screenshot:

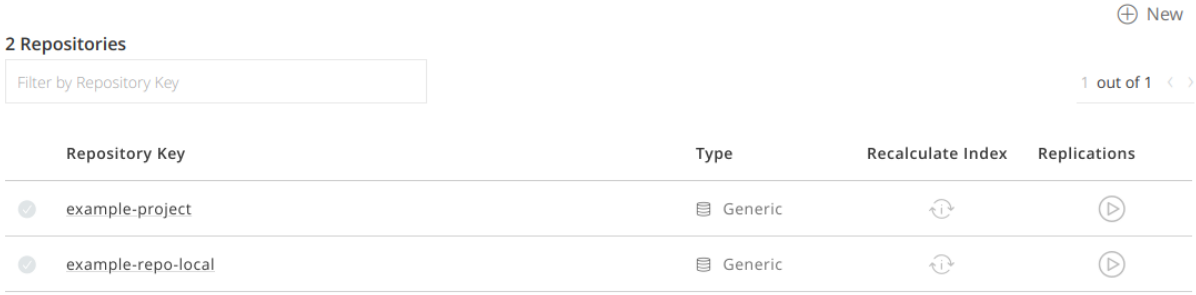


   The following page will show you all the Local Repositories currently available.
2. Click on the "New" link at the top-right corner to create a new local repository:

3. You will get a pop-up window with a list of various types of repositories to choose from. Choose the "Generic" type, as shown on this screenshot:



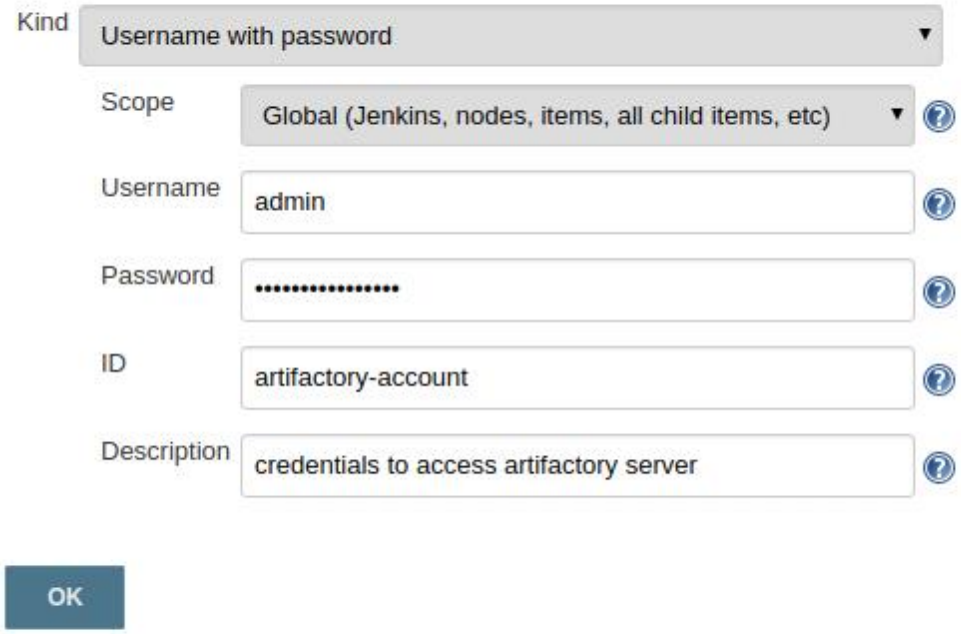
4. Give your repository a name (for example "example-project") by entering a value under the "Repository Key" field.
5. Click the "Save & Finish" button.
6. Your newly created repository should now be in the list of available repositories:



Create JFrog Artifactory credentials inside Jenkins

1. From the Jenkins dashboard, click on "Credentials", then "System", then "Global credentials (unrestricted)".
2. Click on the "Add Credentials" link on the left-hand side menu to create a new credential.
3. Choose "Kind" as "Username with Password".
4. Leave the "Scope" field on its default value of "Global".
5. Add the Artifactory username ("admin") in the "Username" field.
6. In the "Password" field, add the password for the account.
7. Add an ID (name) in the "ID" field (for example "artifactory-account")
8. Add a description in the "Description" field (for example "credentials to access artifactory server").

9. The dialog should now look similar to this screenshot:



10. Click on the "OK" button save your settings.

Installing the JFrog Artifactory plugin in Jenkins

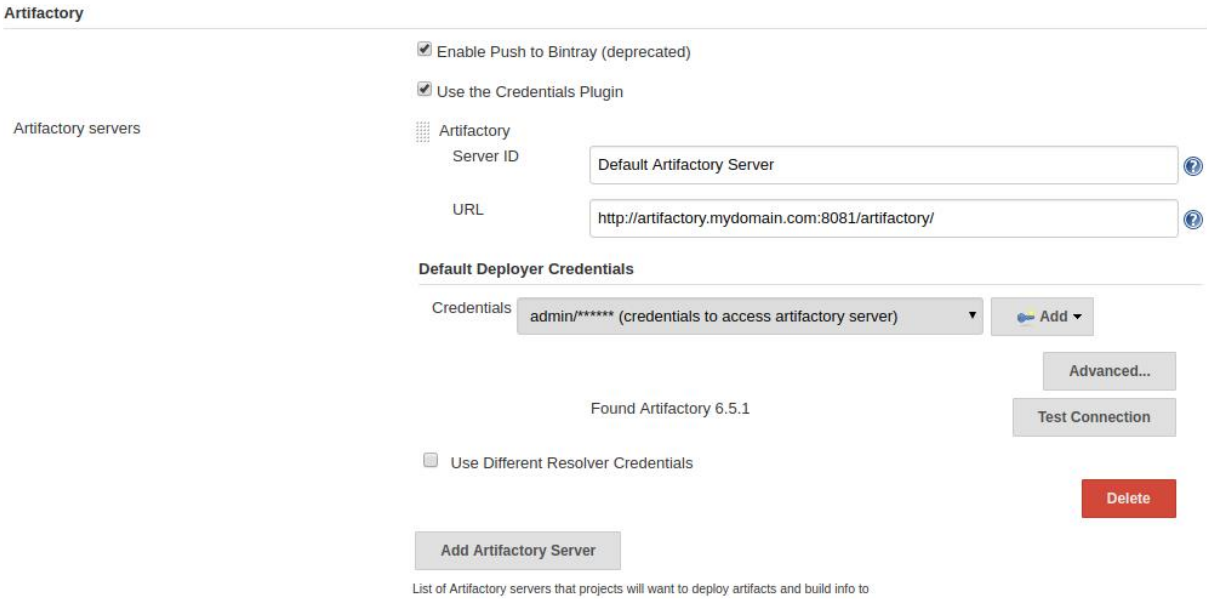
Follow these steps to install the Artifactory plugin in Jenkins:

1. From the Jenkins dashboard, click on "Manage Jenkins", then on "Manage Plugins", and then on the "Available" tab.
2. Type "`Artifactory`" in the Filter field.
3. From the available SonarQube plugins, select (tick) "SonarQube Scanner for Jenkins" from the list and click on the "Install without restart" button.
4. Restart Jenkins if needed.

Configure the JFrog Artifactory plugin in Jenkins

1. From the Jenkins dashboard, click "Manage Jenkins", then "Configure System".
2. Once on the "Configure System" page, scroll down all the way to the "Artifactory" section.
3. Tick the "Use the Credentials Plugin" check box, if you want to use the previously created credentials.
4. Under the "Artifactory servers" section, click on the "Add Artifactory Server" button.
5. Give your Artifactory server a name using the "Server ID" field.
6. Enter the Artifactory server URL in the "URL" field. **Please note that you need to add "`/artifactory/`" to the domain name (see following screenshot), even if your Artifactory is a standalone server!**
7. Select your Artifactory credentials in the "Credentials" drop-down.

8. Click on the "Test Connection" button to test your connection. Your screen should look similar to this:



9. Click on the "Save" button at the end of page.

Continuous Integration Example

Create a new repository on GitHub

Let's create a new repository on GitHub. Make sure you have Git installed on the machine that you use to perform the steps mentioned in the following section:

1. Log in to your GitHub account on https://github.com/ .
2. We use the source code from https://github.com/brunobosshard/ci-jenkins.git as an example.
3. Fork the repository mentioned in the previous link. To do so, just access the repository from your internet browser and click on the Fork button.
4. Once done, a replica of the repository will be visible under your GitHub account.

Using the SonarQube scanner for Maven

To use the SonarQube scanner plugin for Maven, you need to add the following code to the "`POM.xml`" file in your project:

```
<properties>
	<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	<sonar.language>java</sonar.language>
</properties>
```

Note: You don't need to do this if you have forked the example repository, because it is already included in it.

Optional: Generating JaCoCo code coverage reports for SonarQube from Maven

To generate "JaCoCo" code coverage reports from Maven, you need to add the following code to the "`POM.xml`" file in your project. These reports will then be imported in SonarQube.

Part 1 – Add the following SonarQube properties in the section that you have just created in the previous paragraph "Using the SonarQube scanner for Maven":

```
<!-- Sonar-JaCoCo properties -->
<sonar.java.coveragePlugin>jacoco</sonar.java.coveragePlugin>
<sonar.dynamicAnalysis>reuseReports</sonar.dynamicAnalysis>
<sonar.jacoco.reportPath>${project.basedir}/../target/jacoco.exec</sonar.jacoco.reportPath>
```

Part 2 – Add the following section inside "`<build><plugins>`":

```
<!-- Sonar-JaCoCo integration plugin -->
<plugin>
	<groupId>org.jacoco</groupId>
	<artifactId>jacoco-maven-plugin</artifactId>
	<version>0.8.2</version>
	<configuration>
		<destFile>${sonar.jacoco.reportPath}</destFile>
		<append>true</append>
	</configuration>
	<executions>
		<execution>
			<id>agent</id>
			<goals>
				<goal>prepare-agent</goal>
			</goals>
		</execution>
```

```
        </executions>
</plugin>
```

Note: You don't need to do this if you have forked the example repository, because it is already included in it.

Using a Jenkinsfile

A Jenkinsfile is a text file called "`Jenkinsfile`" that contains the definition of a Jenkins Multibranch Pipeline and is checked into source control.

A Jenkinsfile is written using the Groovy Domain-Specific Language (DSL). It can be created through a text/groovy editor, or through the configuration page on the Jenkins instance.

A Jenkinsfile is only valid for the version control branch that it is part of. If you put a Jenkinsfile in the "master" branch, then it will run the "master" branch only. If you have other branches (for example a testing branch), then you need to place a Jenkinsfile in each branch. This allows for great flexibility, because you can use a different Jenkinsfile (workflow) for each branch.

The following Jenkinsfile is used in this example. It executes on the Jenkins master node and runs through 6 stages:

1. Gets the source code from the Source Control Management (SCM) system.
2. Build and Unit Test.
3. SonarQube Scan to examine and report on the code quality. This stage also triggers a "JaCoCo" code coverage report for SonarQube.
4. SonarQube Quality Gate to check, if the code meets the quality criteria. If not, then the pipeline will terminate at this step.
5. Integration Test.
6. Save resulting items in the binary repository server, Artifactory. This example will just promote the "*.war" file to Artifactory in a folder with the Jenkins build number.

```
node('master') {
        stage('Poll') {
                checkout scm
        }
        stage('Build and Unit Test'){
                sh 'mvn clean verify -DskipITs=true';
                junit '**/target/surefire-reports/TEST-*.xml'
                archive 'target/*.jar'
        }
        stage('SonarQube Scan') {
                // Create JaCoCo code coverage report
                sh 'mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent install -
Dmaven.test.failure.ignore=false'
                withSonarQubeEnv('Default SonarQube server') {
                        sh 'mvn clean verify sonar:sonar -Dsonar.projectName=example-project -
Dsonar.projectKey=example-project -Dsonar.projectVersion=$BUILD_NUMBER';
                }
        }
        stage('SonarQube Quality Gate') {
                timeout(time: 1, unit: 'HOURS') { // Just in case something goes wrong,
pipeline will be killed after a timeout
                        def qg = waitForQualityGate() // Reuse taskId previously collected by
withSonarQubeEnv
                        if (qg.status != 'OK') {
                                error "Pipeline aborted due to quality gate failure:
${qg.status}"
                        }
                }
```

```
        }
        stage ('Integration Test'){
                sh 'mvn clean verify -Dsurefire.skip=true';
                junit '**/target/failsafe-reports/TEST-*.xml'
                archive 'target/*.jar'
        }
        stage ('Publish'){
                def server = Artifactory.server 'Default Artifactory Server'
                def uploadSpec = """{
                        "files": [
                                {
                                        "pattern": "target/*.war",
                                        "target": "example-project/${BUILD_NUMBER}/",
                                        "props": "Integration-Tested=Yes;Performance-Tested=No"
                                }
                        ]
                }"""
                server.upload(uploadSpec)
        }
}
```

Creating a Multibranch Pipeline in Jenkins

Follow these steps to create a new Jenkins Multibranch Pipeline job:

1. From the Jenkins dashboard, click on the "New Item" link.
2. Choose "Multibranch Pipeline", and give a name to your Multibranch Pipeline using the "Enter an item name" field (for example "ci-jenkins").
3. Click on the "OK" button at the bottom of the page.
4. Scroll to the "Branch Sources" section, click on the "Add Source" button, and choose "GitHub".
5. Select your GitHub credentials from the "Credentials" dropdown.
6. In the "Owner" field, type the name of your GitHub user account. This will then populate the dropdown selection under "Repository" with available GitHub repositories.
7. Choose "ci-jenkins" from the dropdown selection in the "Repository" field.
8. Leave the rest of the "Branch Sources" section options to their default values.
9. Scroll all the way down to the "Build Configuration" section. Make sure the "Mode" field is set to "by Jenkinsfile" and the Script Path field is set to "Jenkinsfile".
10. Scroll all the way down to the end of the page and click on the "Save" button.

(Re-)register the GitHub Webhooks

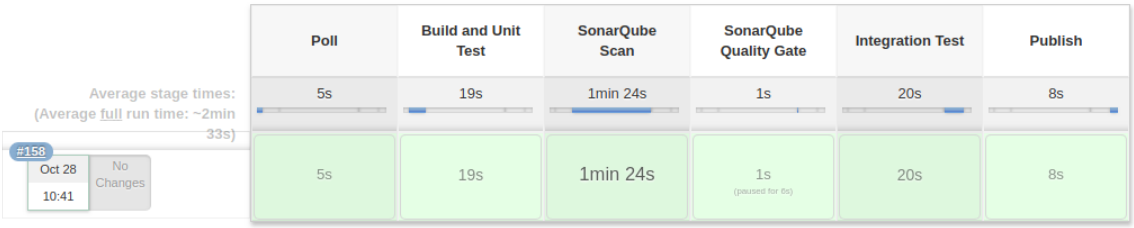
To (re-)register the GitHub Webhooks for all Jenkins pipelines:

1. On the Jenkins dashboard, click on "Manage Jenkins".
2. Click on "Configure System".
3. On the Jenkins configuration page, scroll to the "GitHub" section.
4. Under the "GitHub" section, click on the last (second) "Advanced…" button (the one with the notepad icon next to it).
5. This will display a few more fields and options. Click on the "Reregister hooks for all jobs" button. It you like to verify the success of this action, then you can do that on the GitHub web interface.
6. Click the "Save" button at the end of the page.

Results screenshots of the sample project

Jenkins

Stage View



SonarQube Quality Gate

example-project OK

server-side processing: Success

Test Result : (root)

0 failures

3 tests

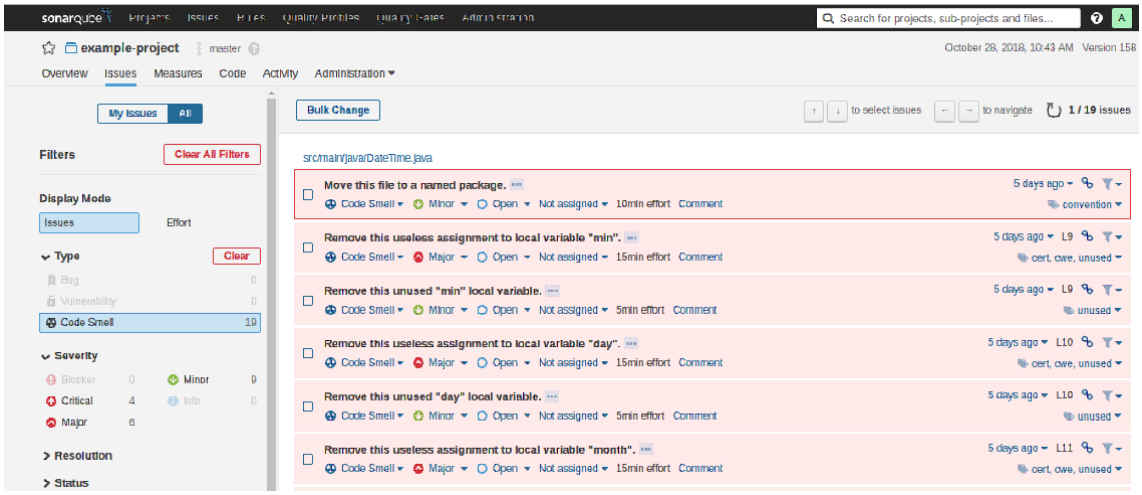
Took 53 ms.

add description

All Tests

Class	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
DateTimeTest	17 ms	0		0		1	+1	1	+1
GreetingMessageIT	0.44 sec	0		0		1	+1	1	+1
MessageTest	0.35 sec	0		0		1	+1	1	+1

SonarQube



JFrog Artifactory