

CI/CD using Jenkins job

JUNE 20, 2018 ANUSHA SHARMA 5 COMMENTS

Jenkins

Jenkins is a self-contained (Java-based program), open source automation server. It is used by various organizations to perform DevOps practices, like Continuous Integration and Continuous delivery. With the help of its extendable architecture through plugins, it can easily be integrated with myriads of tools and can easily implement CI/CD using Jenkins job both manually as well as through script.

However, before configuring a Jenkins job to integrate above mentioned tools, make sure that the corresponding plugins and tools are installed and configured in Jenkins. Following are the links with the detail illustrations of the prerequisites :

Integrating Jenkins with –

1. SonarQube.
2. JFrog Artifactory.
3. Maven.

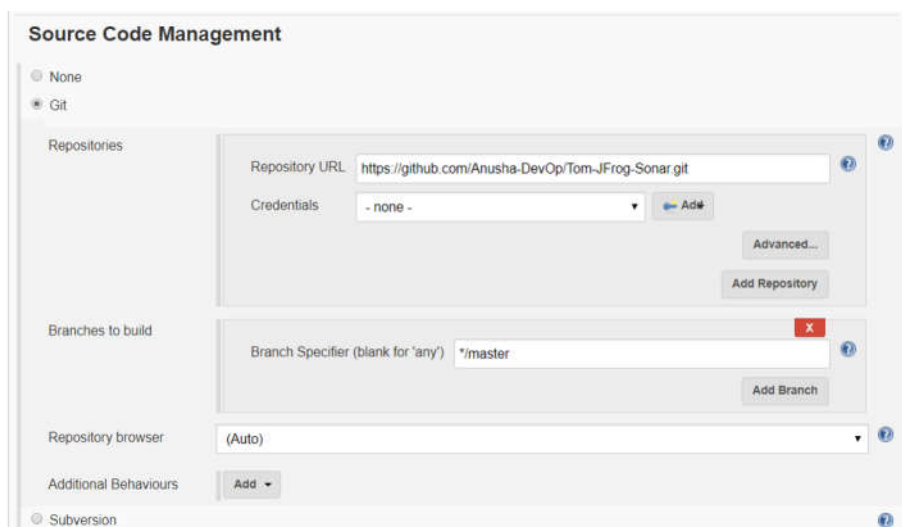
Integrating Jenkins WITH myriads of tools (Maven + Git + SonarQube + JFrog Artifactory + tomcat9x) manually

CI/CD using Jenkins job

Following are the quick steps to follow to integrate various tools and build a pipeline manually:

Step 1. Go to Jenkins Dashboard -> New Item -> Maven Project -> Configuration window of the project will appear.

Step 2. Go to SCM section of the job and provide the details (URL & credentials if any) of the source code repository.



The screenshot shows the 'Source Code Management' configuration page in Jenkins. The 'Git' option is selected under 'Source Code Management'. The 'Repositories' section contains a 'Repository URL' field with the value 'https://github.com/Anusha-DevOp/Tom-JFrog-Sonar.git' and a 'Credentials' dropdown menu set to '- none -'. There are 'Advanced...' and 'Add Repository' buttons. The 'Branches to build' section has a 'Branch Specifier (blank for \'any\')' field with the value '*/master' and an 'Add Branch' button. The 'Repository browser' dropdown is set to '(Auto)'. At the bottom, there is an 'Additional Behaviours' section with an 'Add' button and a 'Subversion' option.

Step 3. Go to the Build section and choose “resolve artifacts from artifactory” and after refresh repository, choose the dedicated repositories (either created one or pre-defined).

Build Environment

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s) ?
- ☐ Provide Configuration files ?
- ☐ Abort the build if it's stuck
- ☐ Add timestamps to the Console Output
- ☐ Enable Artifactory release management ?
- ☒ Resolve artifacts from Artifactory ?

Artifactory server

http://localhost:8081/artifactory ▼

Resolution releases repository

libs-release ▼

Different Value

Resolution snapshots repository

libs-snapshot ▼

Different Value

Refresh Repositories

☐ Override default credentials

Artifactory server specifications

Step 4. Go to pre-steps and configure the SonarQube scanner.

Pre Steps

Execute SonarQube Scanner

X

Task to run

scan

?

JDK

Java-1.8

?

JDK to be used for this SonarQube analysis

Path to project properties

?

Analysis properties

Required metadata
sonar.projectKey=Maven-Sonar-Jfrog-Tomcat-demo
sonar.projectName= Maven project analyzed with the SonarQub Runner
sonar.projectVersion=1.0
Comma-separated paths to directories with sources (required)
sonar.sources=src/main/webapp

#Language
sonar.language=java
Encoding of the source files
sonar.sourceEncoding=UTF-8

?

Additional arguments

▼

?

JVM Options

▼

?

Add pre-build step ▼

Sonar-project.properties

Following is the analysis property section for a webapp type Maven project.

```
# Required metadata
sonar.projectKey=Maven-Sonar-Jfrog-Tomcat-demo
sonar.projectName= Maven project analyzed with the SonarQub Runner
sonar.projectVersion=1.0

# Comma-separated paths to directories with sources (required)
sonar.sources=src/main/webapp

#Language
sonar.language=java

# Encoding of the source files
sonar.sourceEncoding=UTF-8
```

*Note: Although the project properties can be provided in the project configuration window (as described above), the best practice is to create a file “sonar-project.properties” in the workspace (in the root directory). Finally then provide the path in the “Path to project properties”.

Step 4. In the Build section, specify the maven goals (e.g. clean package)

Build

Root POM	<input type="text" value="pom.xml"/>	?
Goals and options	<input type="text" value="clean package"/>	?
MAVEN_OPTS	<input type="text"/>	▼ ?

☐ Incremental build - only build changed modules ?
☐ Disable automatic artifact archiving ?
☐ Disable automatic site documentation artifact archiving ?
☐ Disable automatic fingerprinting of consumed and produced artifacts ?
☒ Enable triggering of downstream projects ?

☒ Block downstream trigger when building ?

☐ Build modules in parallel ?
☐ Use private Maven repository ?
☒ Resolve Dependencies during Pom parsing ?
☐ Run Headless ?
☐ Process Plugins during Pom parsing ?
☒ Use custom workspace ?

Directory	<input type="text" value="D:\DevOps\maven\mv-Tom-JF-Sonar-Jen\mv-tom-jfrog-demo"/>
Maven Validation Level	<input type="text" value="DEFAULT"/>

Maven build configuration

Step 5. Go to the Post-Build section -> Choose “Deploy artifacts to artifactory”

Post-build Actions

Deploy artifacts to Artifactory

Artifactory server

Target releases repository

Different Value

Target snapshot repository

Different Value

Custom staging configuration

Refresh

☐ Override default credentials

☐ Deploy even if the build is unstable ?
☐ Override build name ?
☒ Deploy maven artifacts ?

Include Patterns

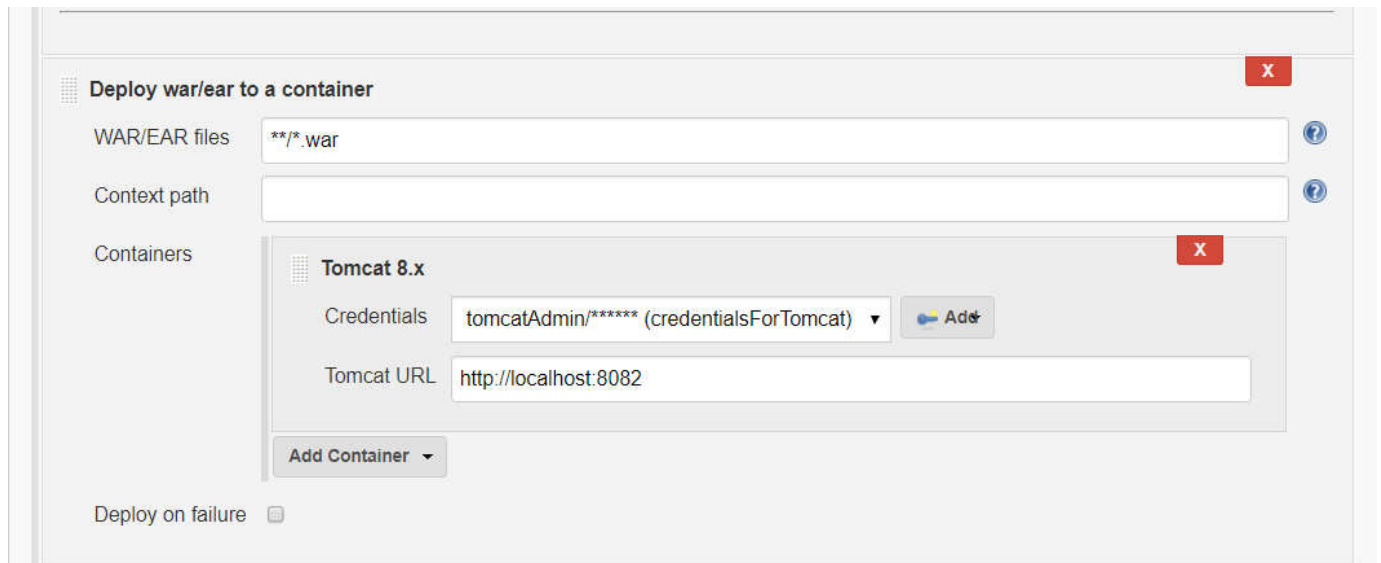
?

Exclude Patterns

?

Specifying target repositories in Artifactory

Step 6. Go to the Post-Build section -> Choose Deploy war/ear to a container & specify tomcat credentials specified in the tomcat-users.xml



The screenshot shows the Jenkins configuration page for 'Deploy war/ear to a container'. The page has a title bar with a red 'X' button. Below the title, there are three main sections: 'WAR/EAR files', 'Context path', and 'Containers'. The 'WAR/EAR files' section has a text input field containing '**/*.war'. The 'Context path' section has an empty text input field. The 'Containers' section is expanded, showing a list of containers. The first container is 'Tomcat 8.x', which has a red 'X' button in its top right corner. Below the container name, there are two fields: 'Credentials' with a dropdown menu showing 'tomcatAdmin/***** (credentialsForTomcat)' and an 'Add' button, and 'Tomcat URL' with a text input field containing 'http://localhost:8082'. At the bottom of the 'Containers' section, there is an 'Add Container' button. Below the 'Containers' section, there is a 'Deploy on failure' checkbox which is currently unchecked.

Deployment container specifications

Step 7. Go to Project/job window -> Click on Build Now.

Finally Congratulations !! You have successfully build your Jenkins job manually.

Furthermore, click [here](#) to do the same through pipeline script.