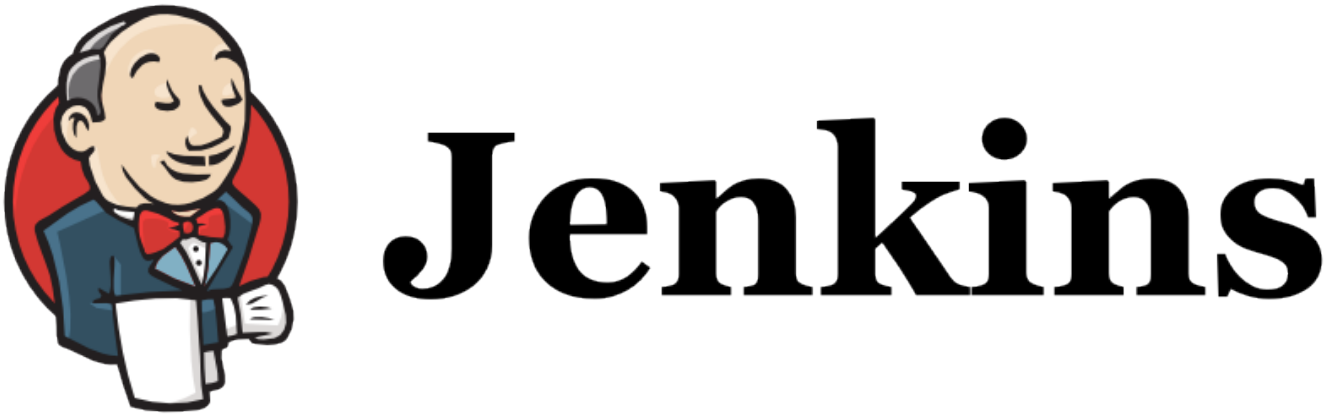
Integration with Jenkins



- Overview
- Release Notes
- Installation
  - Manual Installation
  - Jenkins Native Installation (via web UI)
- Configuration
  - Jira servers
- Creating a new Project
- Build Steps
  - Xray: Cucumber Features Export Task
    - Configuration
  - Xray: Cucumber Features Import Task
  - Xray: Results Import Task
    - Configuration
    - Additional fields
  - Xray: Build Enviroment Variables
- Examples
  - Cucumber
    - Exporting Cucumber features
    - Importing Cucumber features
    - Importing the execution results
    - Importing the execution results with user-defined field values
  - JUnit
    - Importing the execution results
- Pipeline projects support
  - Examples
    - JUnit
    - JUnit multipart
    - Cucumber ("standard" workflow)
    - Cucumber ("VCS/Git based" workflow)
    - Using parameters
  - Recommendations
- Troubleshooting
  - The build process is failing with status code 403

Overview

Xray enables easy integration with Jenkins through the "Xray for JIRA Jenkins Plugin", providing the means for successful Continuous Integration by allowing users to report automated testing results.

Release Notes

- Xray for JIRA Jenkins Plugin 2.2.0 Release Notes
- Xray for JIRA Jenkins Plugin 2.1.2 Release Notes
- Xray for JIRA Jenkins Plugin 2.1.1 Release Notes
- Xray for JIRA Jenkins Plugin 2.0.0 Release Notes
- Xray for JIRA Jenkins Plugin 1.3.0 Release Notes

- Xray for JIRA Jenkins Plugin 1.2.1 Release Notes
- Xray for JIRA Jenkins Plugin 1.2.0 Release Notes
- Xray for JIRA Jenkins Plugin 1.1.0 Release Notes
- Xray for JIRA Jenkins Plugin 1.0.0 Release Notes

Installation

The installation is made manually. For more information on how to install add-ons, please refer to how to install add-ons.

Requirements

The Jenkins baseline for this app is v2.60.3 and it may not work properly with previous versions.

Manual Installation

Download the latest version of the Jenkins Plugin

You may download the latest version of the Jenkins plugin from the latest Release Notes.

If you have the actual `xray-connector.hpi` file,

1. Go to the Update Center of Jenkins in Manage Jenkins > Manage Plugins.
2. Select the advanced tab
3. In the Upload Plugin section, click upload and select the file `xray-connector.hpi` file.

Jenkins Native Installation (via web UI)

Since version 2.1.0, you can install the plugin by using the Jenkins native Web UI. You can read more about how to to it here.

Configuration

Xray for Jenkins is configured in the global settings configuration page **Manage Jenkins** > **Configure System > Xray for Jira configuration**.

Jira servers

The Jira servers configuration defines connections with Jira instances.

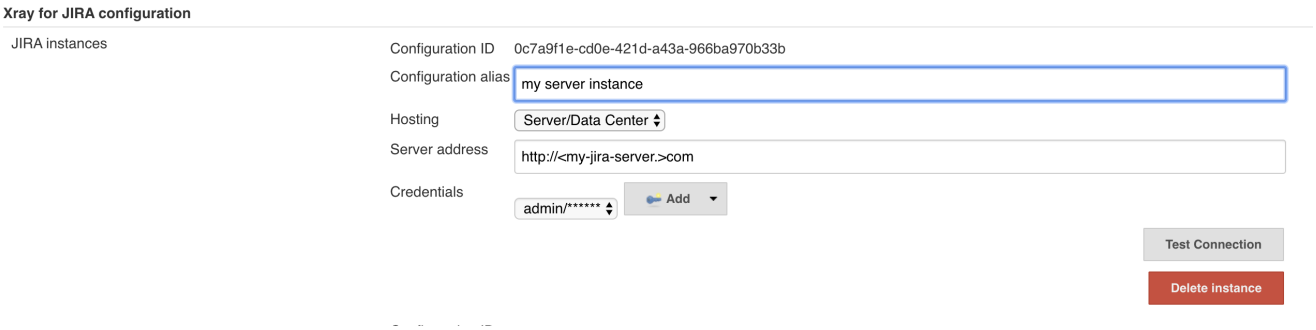
To add a new Jira instance connection, you need to specify some properties:

1. **Configuration alias**
2. **Hosting:** Hosting (instance type) in this case Server/Data Center.
3. **Server Address**: The address of the Jira Server where Xray is running
4. Credentials:
    a. Use the **Jenkins Credentials Plugin** to set the username/password (if you are using a Server/Data Center instance).
    b. Make sure that the user you are using have the following permissions in the projects where you want to import the results and import /export feature files: **View, Edit, Create.**

note: the Configuration ID is not editable. This value can be used in the pipelines scripts.

Please note

The user present in this configuration must exist in the JIRA instance and have permission to Create Test and Test Execution Issues



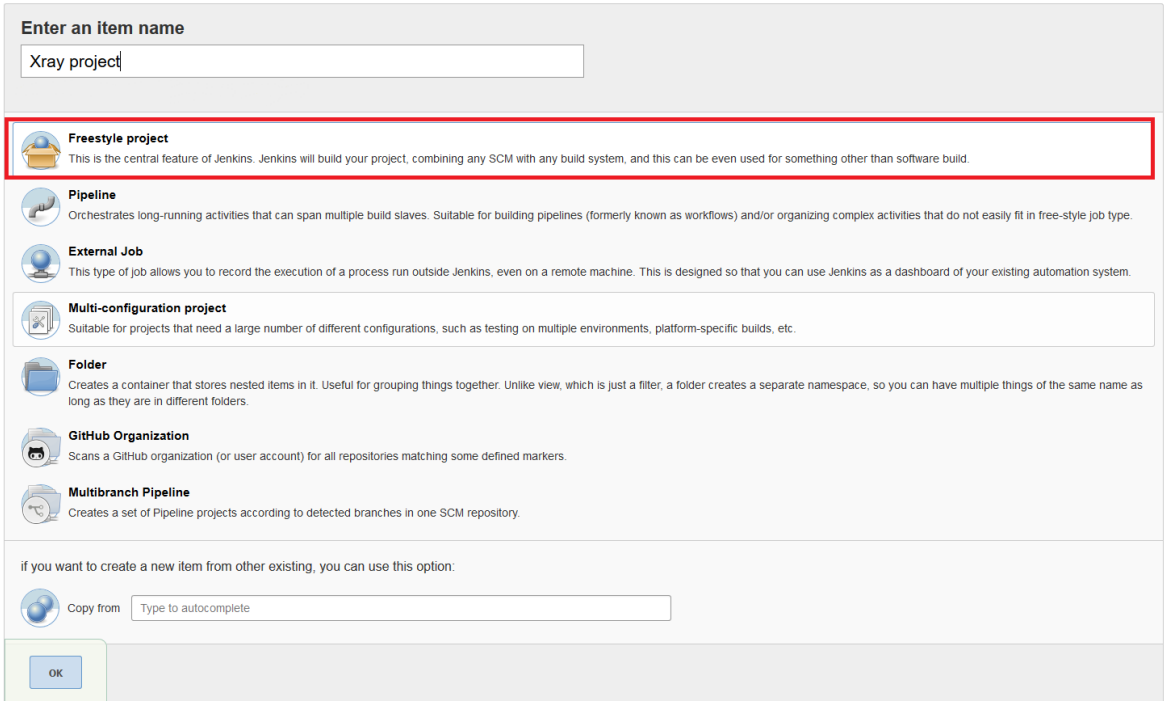
Creating a new Project

The project is where the work that should be performed by Jenkins is configured.

For this app, you can configure:

- Freestyle projects
- Maven Projects
- Multi-configuration Projects
- Pipeline Projects

In the home page, clicking for example New Item > Freestyle project, provide a name, and then click OK.



Build Steps

Build steps are the building blocks of the build process. These need to be defined in the project configuration.

The app provides

- one build step for exporting Cucumber Scenario/Scenario Outlines from Jira as .feature files
- one build step for importing Cucumber Tests from existing Cucumber features into Jira.
- one post-build action which publishes the execution results back to Jira, regardless of the build process status.

Please note

The fields of the tasks may take advantage of the Jenkins Environment variables, which can be used to populate fields such as the "Revision" for specifying the source code's revision. For more information, please see Jenkins set environment variables.

Xray: Cucumber Features Export Task

This build step will export the Cucumber Tests (i.e., Scenario/Scenario Outlines) in .feature or bundled in a .zip file. The rules for exporting are defined here.

It invokes Xray's Export Cucumber Tests REST API endpoint (see more information here).

Configuration

Some fields need to be configured in order to export the Cucumber Tests. As input, you can either specify issue keys (see the endpoint documention here) or the ID of the saved filter in Jira.

field	description
Jira instance	The Jira instance where Xray is running
Issue keys	Set of issue keys separated by ";"
Filter ID	A number that indicates the filter ID
File path	The relative path of the directory where the features should be exported to; normally, this corresponds to the "features" folder of the Cucumber project that has the implementation steps. Note: The directory will be created if it does not exist.

Xray: Cucumber Features Import Task

This build step will import existing cucumber Tests from existing Cucumber feature files into Xray issues. This Task will import from .feature files and also from .zip files.

It invokes Xray's Import Cumcumber Tests REST API endpoint (see more information here)

field	decription
JIRA instance	The Jira instance where Xray is running.
Project Key	This is the project where the Tests and Pre-Conditions will be created/updated.
Cucumber feature files directory	This is the directory containing your feature files. All the files in this directory and sub directories will be imported.
Modified in the last hours	By entering an integer *n* here, only files that where modified in the last *n* hours will be imported. Leave empty if you do not want to use this parameter.

Xray: Results Import Task

The app provides easy access to Xray's Import Execution Results REST API endpoints (see more information here). Therefore, it mimics the endpoints input parameters.

It supports importing results in Xray's own JSON format, Cucumber, Behave, JUnit, and NUnit, among others.

Using a glob expression, you can import multiple results files in the following formats:

- JUnit
- TestNG
- NUnit
- Robot framework

For those formats, the file path needs to be relative to the workspace.

Configuration

field	description
Jira instance	The Jira instance where Xray is running
Format	A list of test result formats and its specific endpoint
Execution Report File	The results relative file path Note: glob expressions are supported for • JUnit • JUnit Multipart • TestNG • TestNG Multipart • NUnit • NUnit Multipart • Robot framework • Robot framework Multipart

Additional fields

Depending on the chose test result format and endpoint, some additional fields may need to be configured.

Format and specific endpoint	Field	Description
Behave JSON multipart Cucumber JSON multipart NUnit XML multipart JUnit XML multipart Robot XML multipart TestNG XML multipart	Import to Same Test Execution	When this option is check, if you are importing multiple execution report files using a glob expression, the results will be imported to the same Test Execution
	Test execution fields	An object (JSON) specifying the fields for the issue. You may specify the object either directly in the field or in the file path. **Learn more** The custom field IDs can be obtained using the Jira REST API Browser tool included in Jira. Each ID is of the form "**customfield_ID**". Another option, which does not require Jira administration rights, is to invoke the "Get edit issue meta" in an existing issue (e.g., in a Test issue) as mentioned here. Example: **GET http://yourserver/rest/api/2/issue/CALC-1/editmeta**
NUnit XML JUnit XML Robot XML TestNG XML	Import to Same Test Execution	When this option is check, if you are importing multiple execution report files using a glob expression, the results will be imported to the same Test Execution
	Project key	Key of the project where the Test Execution (if the **Test Execution Key** field wasn't provided) and the Tests (if they aren't created yet) are going to be created
	Test execution key	Key of the Test Execution
	Test plan key	Key of the Test Plan
	Test environments	List of Test Environments separated by ";"
	Revision	Source code's revision being target by the Test Execution
	Fix version	The Fix Version associated with the test execution (it supports only one value)

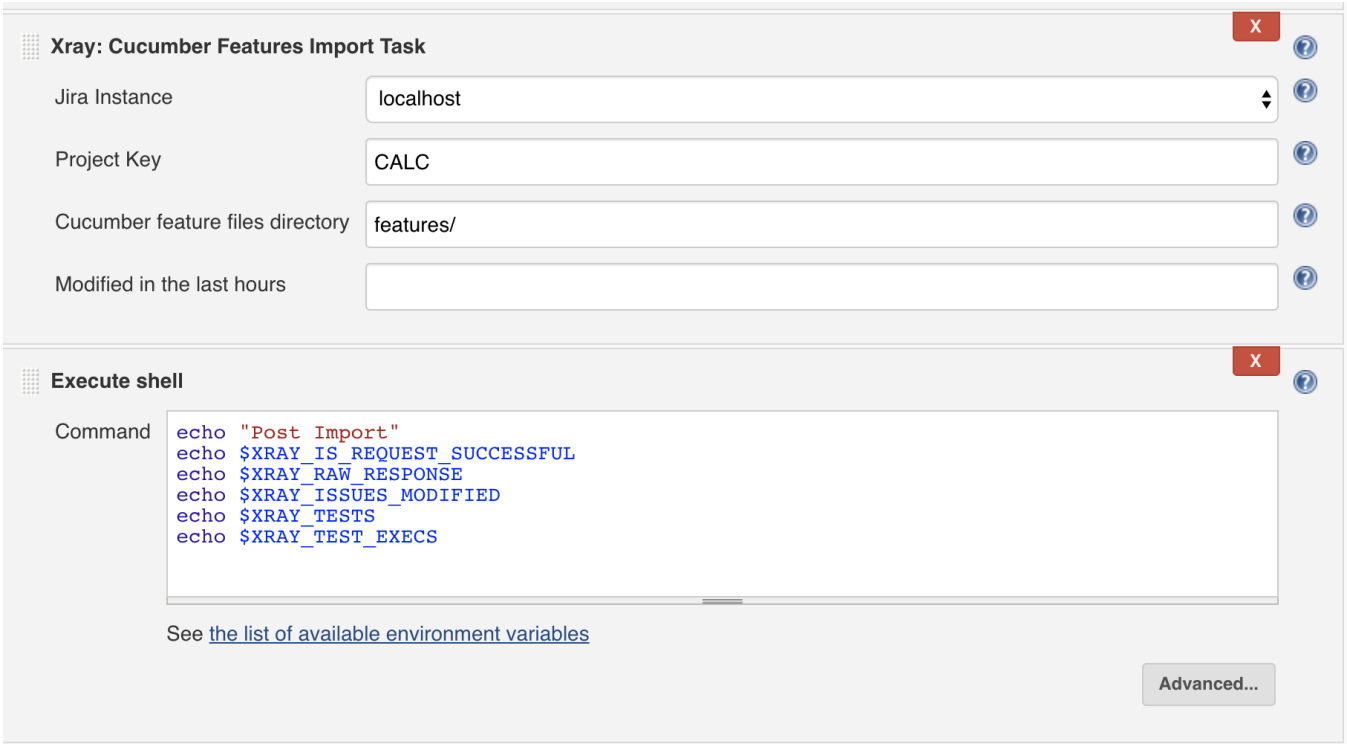
Xray: Build Enviroment Variables

Since version 2.2.0, the Xray plugin will now set some build enviroment variables according to the operation result after each of the Xray Steps mensioned above.

Build Enviroment Variable Name	Meaning and Value
XRAY_IS_REQUEST_SUCCESSFUL	Contains the string 'true' if all requests made by the step were sucesseful, or 'false' otherwise.
XRAY_ISSUES_MODIFIED	All Issue keys that were modified and/or created by the step, seperated by ';' with no duplicated entries (E.g. 'CALC-100;CALC-101;CALC-102').
XRAY_RAW_RESPONSE	The unprocessed JSON response of all requests made by the step, seperated by ';'.
XRAY_TEST_EXECS	All Test Execution Issue keys that were modified and/or created by the step, seperated by ';' with no duplicated entries (E.g. 'CALC-200;CALC-201;CALC-202'). Please note that in same cases, it will be not possible to determine the issue type of the Issue key returned in the request response and in that case, the key it will only be added to the *XRAY_ISSUES_MODIFIED* variable.
XRAY_TEST	All Test Issue keys that were modified and/or created by the step, seperated by ';' with no duplicated entries (E.g. 'CALC-300; CALC-301;CALC-302'). Please note that in same cases, it will be not possible to determine the issue type of the Issue key returned in the request response and in that case, the key it will only be added to the *XRAY_ISSUES_MODIFIED* variable.

Pipeline Project Limitations

Due to Jenkins limitations, these variables will not be set on Pipeline projects.



Examples

Cucumber

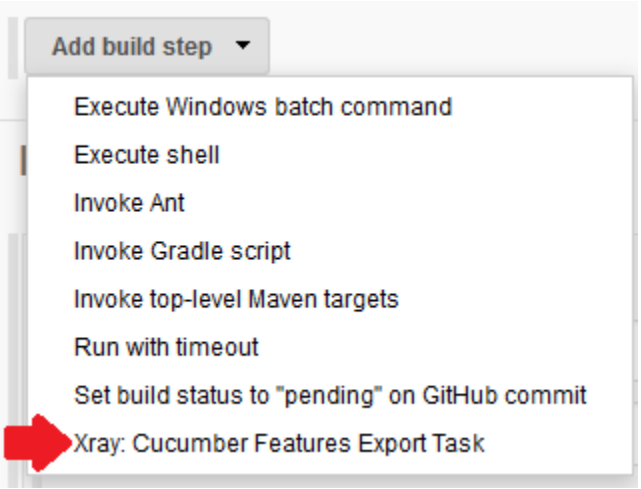
In a typical Cucumber Workflow, after having created a Cucumber project and the Cucumber tests specified in Jira, you may want to have a project that **exports** the features from Jira, executes the automated tests on a CI environment and then **imports** back its results.

For this scenario, the Jenkins project would be configured with a set of tasks responsible for:

1. Pulling the Cucumber project
2. **Exporting Cucumber features from Jira to your Cucumber project**
3. Executing the tests in the CI environment
4. **Importing the execution results back to Jira**

Exporting Cucumber features

To start the configuration, add the build step *Xray: Cucumber Features Export Task*.

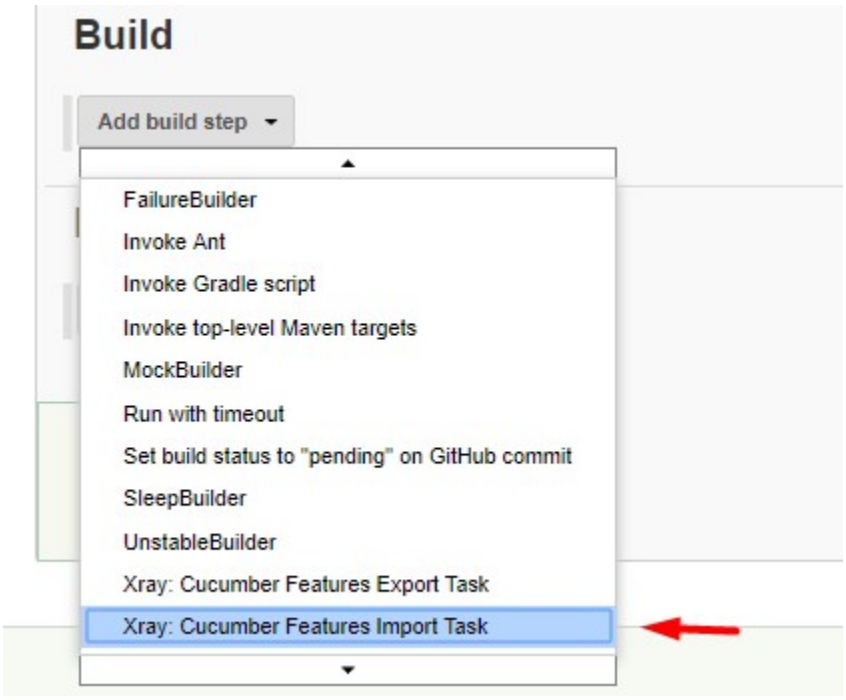


After that, configure it.

In this example, we configured the task to extract the *features* from a set of issues (PROJ-78 and PROJ-79) to the folder that holds the Cucumber project.

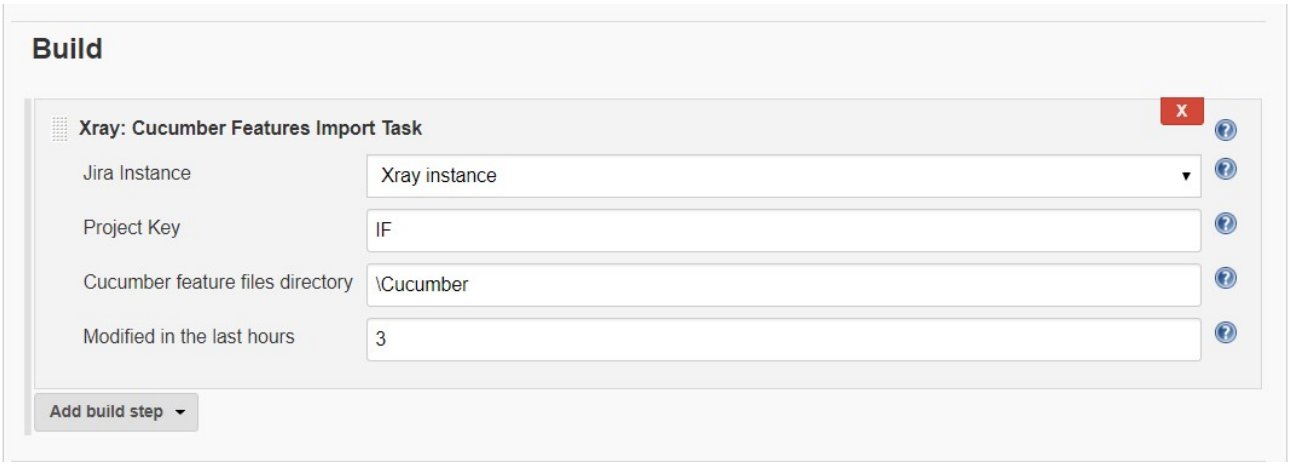
Importing Cucumber features

To start the configuration, add the build step *Xray: Cucumber Features Import Task.*



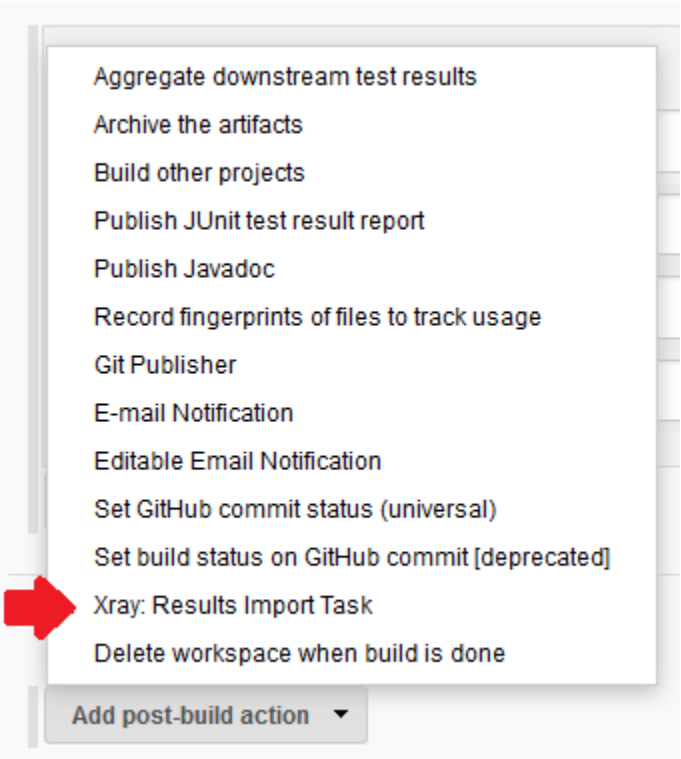
After that, configure it.

In this example, we configured the task to import to the Project IF of the Xray instance all the .features and .zip files that are contained in \Cucumber directory and sub directories, which were modified in the last 3 hours.



Importing the execution results

To start the configuration, add the post-build action *Xray: Results Import Task*.

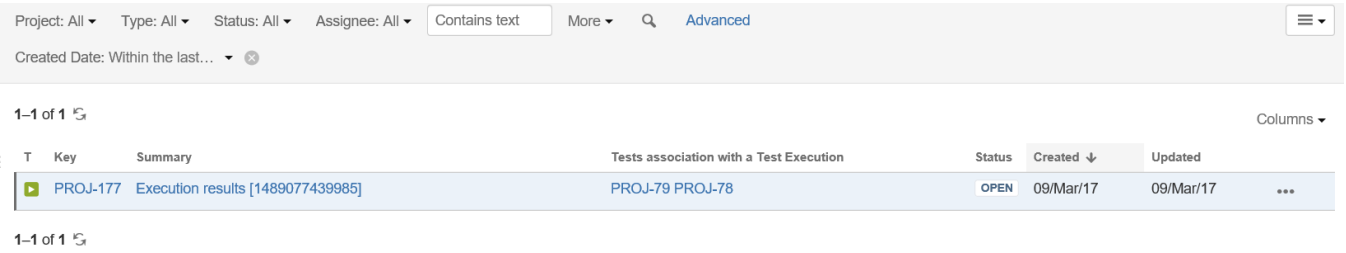


After that, configure it.

In this example, we configured the task to import the **Cucumber JSON** results back to Jira.

Once all configurations are done, click Save at the bottom of the page.

After running the job, the expected result is a new Test Execution issue created in the Jira instance.



Importing the execution results with user-defined field values

For Cucumber, Behave, JUnit, Nunit and Robot, Xray for Jenkins allows you to create new Test Executions and have control over newly-created Test Execution fields. You can send two files, the normal execution result file and a JSON file similar to the one Jira uses to create new issues. More details regarding how Jira creates new issues here.

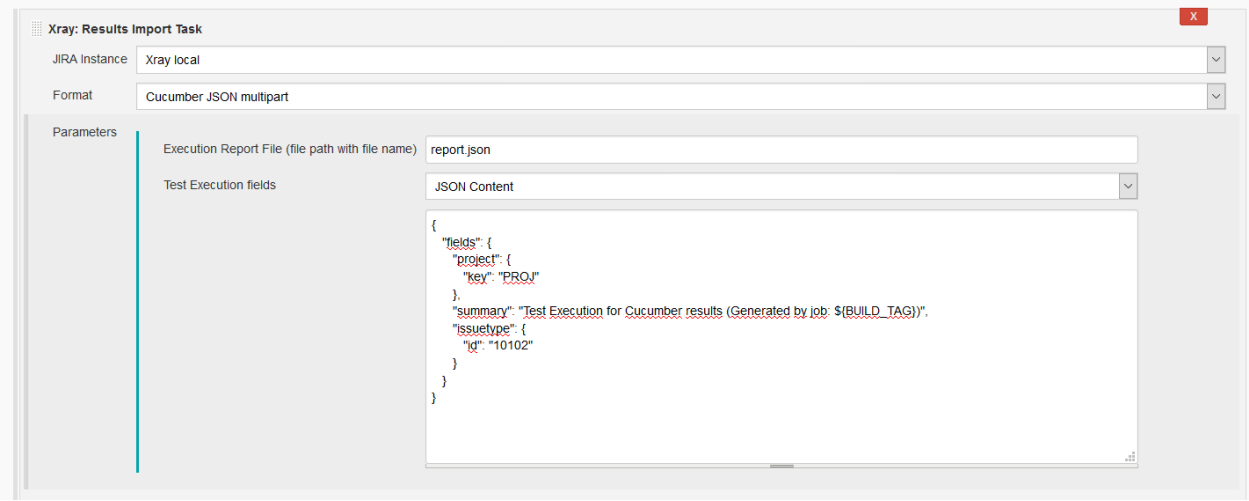
For this scenario and example, the import task needs to be configured with the **Cucumber JSON Multipart** format. When selecting this option, you can additionally configure the *Test Execution fields* in one of two ways:

- Insert the relative **path** to the JSON file containing the information, or
- Insert the **JSON content** directly in the field.

In this example, we configured the following object:

```
{
   "fields": {
      "project": {
         "key": "PROJ"
      },
      "summary": "Test Execution for Cucumber results (Generated by job: ${BUILD_TAG})",
      "issuetype": {
         "id": "10102"
      }
   }
}
```

And configured the task to import the **Cucumber JSON Multipart** results back to Jira.



Once all configurations are done, click Save at the bottom of the page.

After running the job, the expected result is a new Test Execution issue created in the Jira instance, with the Test Execution fields as specified in the Jenkins build step configuration.

JUnit

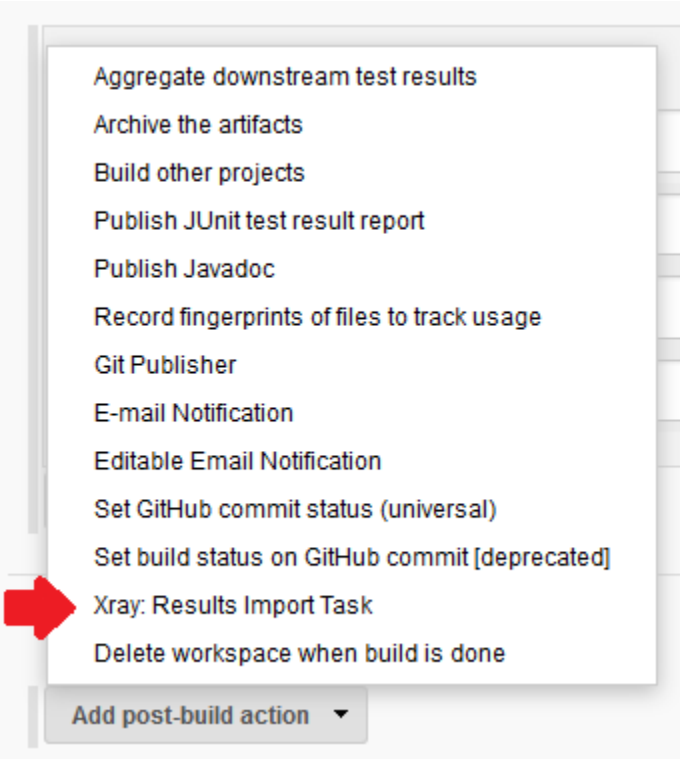
Apart from supporting Cucumber natively, Xray enables you to take advantage of many other testing frameworks like JUnit. In this sense, Xray for Jenkins lets you import results in other formats besides Cucumber JSON.

If you want to import **JUnit XML reports,** a typical Job outline would be:

1. Pulling the JUnit project
2. Executing the tests in the CI environment
3. **Importing the execution results, including Tests, to JIRA**

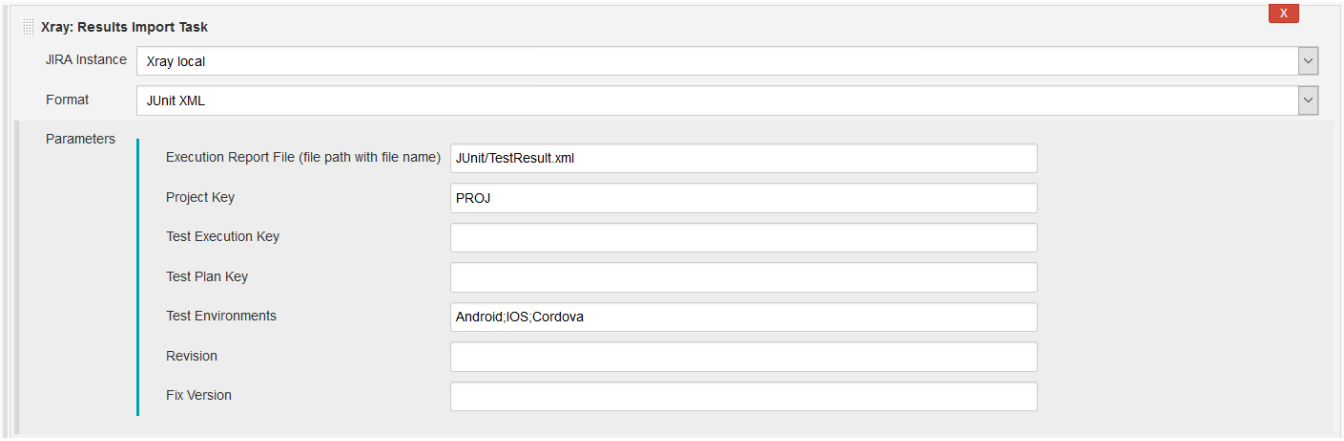
Importing the execution results

To start the configuration, add the post-build action *Xray: Results Import Task*.

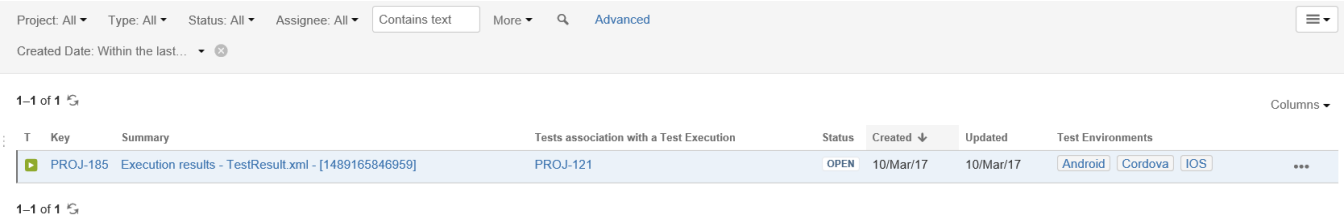


After that, configure it.

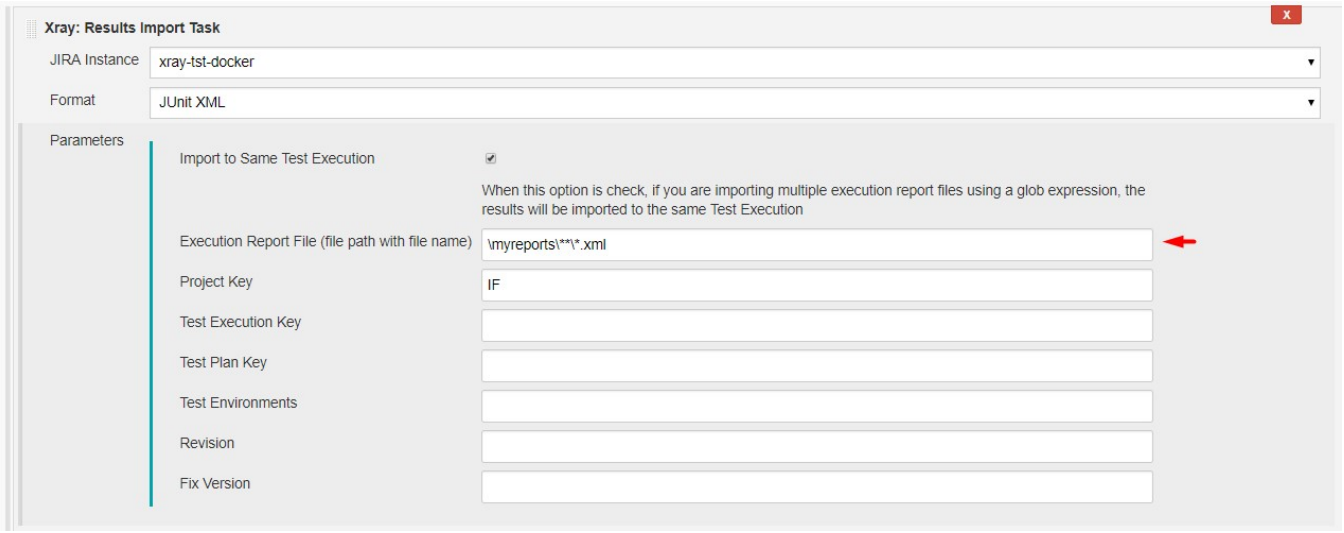
In this example, we have a configuration where the **JUnit XML** format is chosen.



After running the plan, the expected result is a new Test Execution issue created in the JIRA instance.

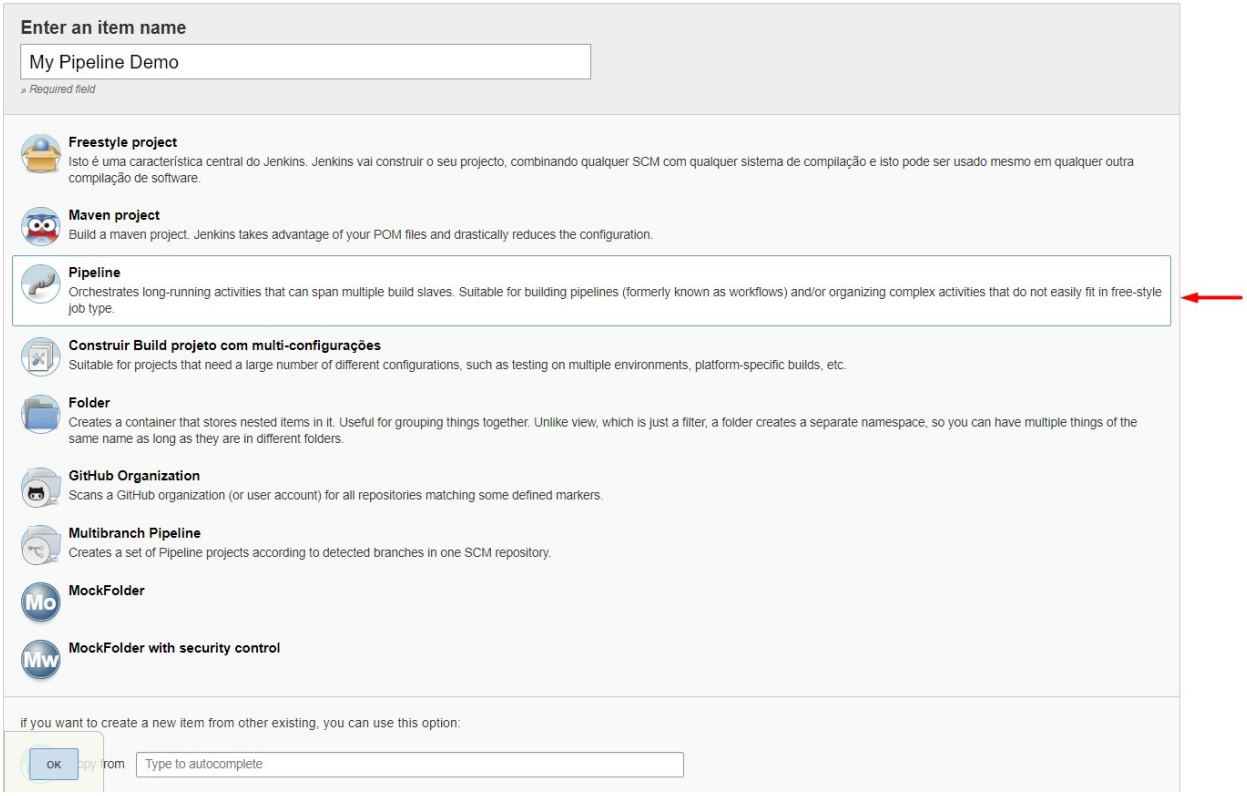


You can also import multiple results using a glob expression, like in the following example



Pipeline projects support

Xray for Jenkins provides support for pipelines projects, allowing you to use Xray specific tasks.



Here is a simple example of a pipeline script using the Xray: Cucumber Features Export Task

Jenkinsfile example (declarative)

```
pipeline {
    agent any
    stages {
        stage('Export Cucumber') {
            steps {
               step([$class: 'XrayExportBuilder', filePath: '\\features', issues: 'IF-1', serverInstance:
'2ffc3a3e-9e2f-4279-abcd-e9301fe47bed'])
            }
        }
    }
}
```

Learn more

For Pipeline specific documentation, you may want to give a look at:

- https://jenkins.io/doc/book/pipeline/
- https://jenkins.io/doc/book/pipeline/syntax/#declarative-pipeline
- https://github.com/jenkinsci/pipeline-plugin/blob/master/TUTORIAL.md

Examples

JUnit

This is a declarative example, for JUnit based tests.

Jenkinsfile example (declarative)

```
pipeline {
    agent any
    stages {
        stage('Compile'){
            steps {
                checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations:
false, extensions: [[$class: 'SparseCheckoutPaths', sparseCheckoutPaths: [[path: 'java-junit-calc/']]]],
submoduleCfg: [], userRemoteConfigs: [[credentialsId: 'a3285253-a867-4ea7-a843-da349fd36490', url:
'ssh://git@localhost/home/git/repos/automation-samples.git']]])
                sh "mvn clean compile -f java-junit-calc/pom.xml"
            }
        }

        stage('Test'){
            steps{
                sh "mvn test -f java-junit-calc/pom.xml"
            }
        }

        stage('Import results to Xray') {
            steps {
                step([$class: 'XrayImportBuilder', endpointName: '/junit', fixVersion: 'v3.0', importFilePath:
'java-junit-calc/target/surefire-reports/*.xml', importToSameExecution: 'true', projectKey: 'CALC',
serverInstance: '552d0cb6-6f8d-48ba-bbad-50e94f39b722'])
            }
        }
    }
}
```

JUnit multipart

This is a declarative example, for JUnit based tests using the multipart variant/endpoint which allows customization over the Test Execution issue fields.

By changing the value of the *endpointName* variable, you can easily adapt it for other automation frameworks (e.g. Robot framework, TestNG, NUnit).

Jenkinsfile example (declarative)

```
pipeline {
    agent any
    stages {
        stage('Compile'){
            steps {
                checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations:
false, extensions: [[$class: 'SparseCheckoutPaths', sparseCheckoutPaths: [[path: 'java-junit-calc/']]]],
submoduleCfg: [], userRemoteConfigs: [[credentialsId: 'a3285253-a867-4ea7-a843-da349fd36490', url:
'ssh://git@localhost/home/git/repos/automation-samples.git']]])
                sh "mvn clean compile -f java-junit-calc/pom.xml"
            }
        }

        stage('Test'){
            steps{
                sh "mvn test -f java-junit-calc/pom.xml"
            }
        }

        stage('Import results to Xray') {
             steps {
                step([$class: 'XrayImportBuilder', endpointName: '/junit/multipart', importFilePath: 'java-
junit-calc/target/surefire-reports/TEST-com.xpand.java.CalcTest.xml', importInfo: '''{
           "fields": {
              "project": {
                 "key": "CALC"
              },
              "summary": "Test Execution for java junit ${BUILD_NUMBER}",
              "issuetype": {
                 "id": "9"
              },
              "customfield_11807": [
                 "CALC-1200"
              ]
           }
        }''', inputInfoSwitcher: 'fileContent', serverInstance: '552d0cb6-6f8d-48ba-bbad-50e94f39b722'])
             }
        }

    }
}
```

Cucumber ("standard" workflow)

This is a declarative example, for Cucumber tests using the "standard" workflow (see Testing in BDD with Gherkin based frameworks (e.g. Cucumber)).

Jenkinsfile example (declarative)

```
pipeline {
    agent any
    stages {
        stage('Export features from Xray'){
            steps {
                checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations:
false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: 'a3285253-a867-4ea7-a843-
da349fd36490', url: 'ssh://git@localhost/home/git/repos/automation-samples.git']]])
                step([$class: 'XrayExportBuilder', filePath: 'cucumber_xray_tests/features', filter: '11400',
serverInstance: '552d0cb6-6f8d-48ba-bbad-50e94f39b722'])
            }
        }

        stage('Test'){
            steps{
                sh "cd cucumber_xray_tests && cucumber -x -f json -o data.json"
            }
        }

        stage('Import results to Xray') {
            steps {
                step([$class: 'XrayImportBuilder', endpointName: '/cucumber', importFilePath:
'cucumber_xray_tests/data.json', serverInstance: '552d0cb6-6f8d-48ba-bbad-50e94f39b722'])
            }
        }
    }
}
```

Cucumber ("VCS/Git based" workflow)

This is a declarative example, for Cucumber tests using the "VCS/Git based" workflow (see Testing in BDD with Gherkin based frameworks (e.g. Cucumber)).

Jenkinsfile example (declarative)

```
pipeline {
    agent any
    stages {
         stage('Synch (update) recent tests to Xray'){
            steps {
                checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations:
false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: 'a3285253-a867-4ea7-a843-
da349fd36490', url: 'ssh://git@localhost/home/git/repos/automation-samples.git']]])
                step([$class: 'XrayImportFeatureBuilder', folderPath: 'cucumber_xray_tests/features',
lastModified: '10', projectKey: 'CALC', serverInstance: '552d0cb6-6f8d-48ba-bbad-50e94f39b722'])
            }
        }

        stage('Export features from Xray'){
            steps {
                checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations:
false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: 'a3285253-a867-4ea7-a843-
da349fd36490', url: 'ssh://git@localhost/home/git/repos/automation-samples.git']]])
                sh "rm -rf cucumber_xray_tests/features"
                step([$class: 'XrayExportBuilder', filePath: 'cucumber_xray_tests/features', filter: '11400',
serverInstance: '552d0cb6-6f8d-48ba-bbad-50e94f39b722'])
            }
        }

        stage('Test'){
            steps{
                sh "cd cucumber_xray_tests && cucumber -x -f json -o data.json"
            }
        }

        stage('Import results to Xray') {
            steps {
                step([$class: 'XrayImportBuilder', endpointName: '/cucumber', importFilePath:
'cucumber_xray_tests/data.json', serverInstance: '552d0cb6-6f8d-48ba-bbad-50e94f39b722'])
            }
        }
    }
}
```

Using parameters

You can ask for human input in your pipeline builds by passing parameters

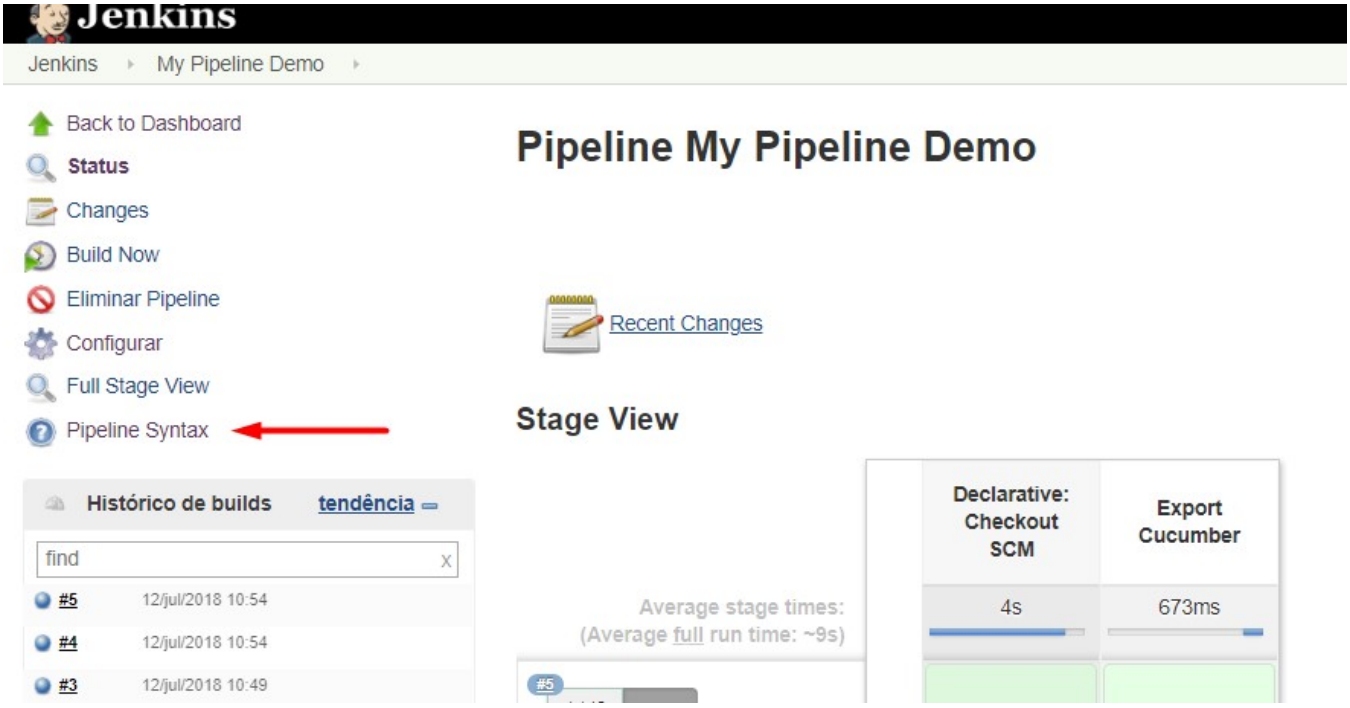
Parameters usage

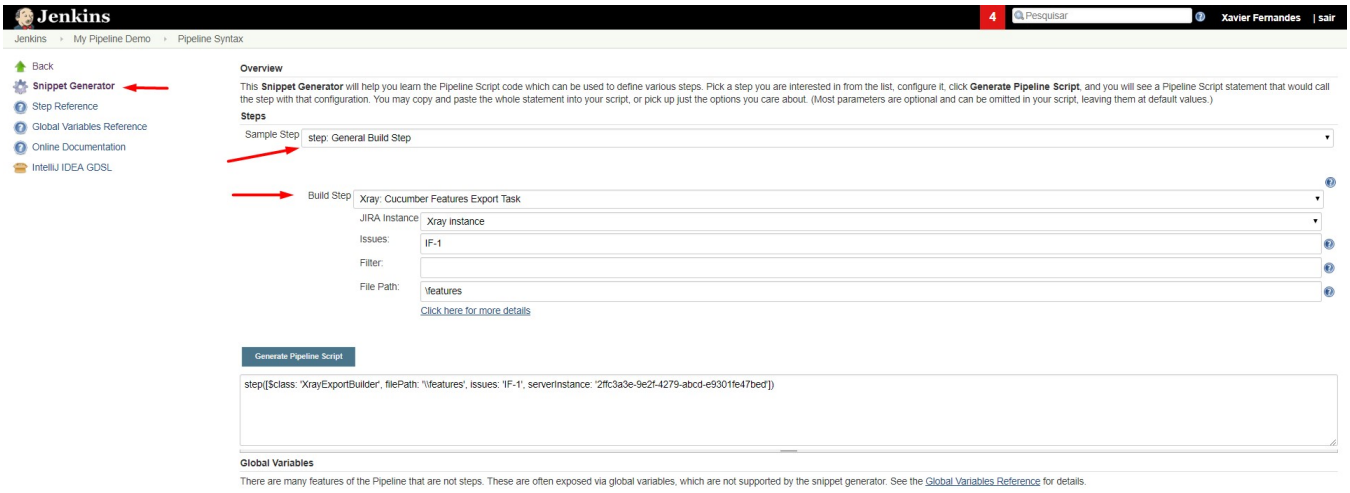
```
pipeline{
    agent any
    parameters {
        string(defaultValue: "NTP", description: '', name: 'projectKey')
        string(defaultValue: "Android", description: '', name: 'env')
    }
    stages {
        stage ('Import Results') {
            steps {
                step([$class: 'XrayImportBuilder',
                endpointName: '/junit',
                importFilePath: 'java-junit-calc/target/surefire-reports/*.xml',
                importToSameExecution: 'true',
                projectKey: params.projectKey,
                revision: params.projectKey + env.BUILD_NUMBER,
                serverInstance: '552d0cb6-6f8d-48ba-bbad-50e94f39b722',
                testEnvironments: params.env])
            }

        }
    }
}
```

Recommendations

You can automatically generate your step scripts using the Jenkins Snippet Generator.



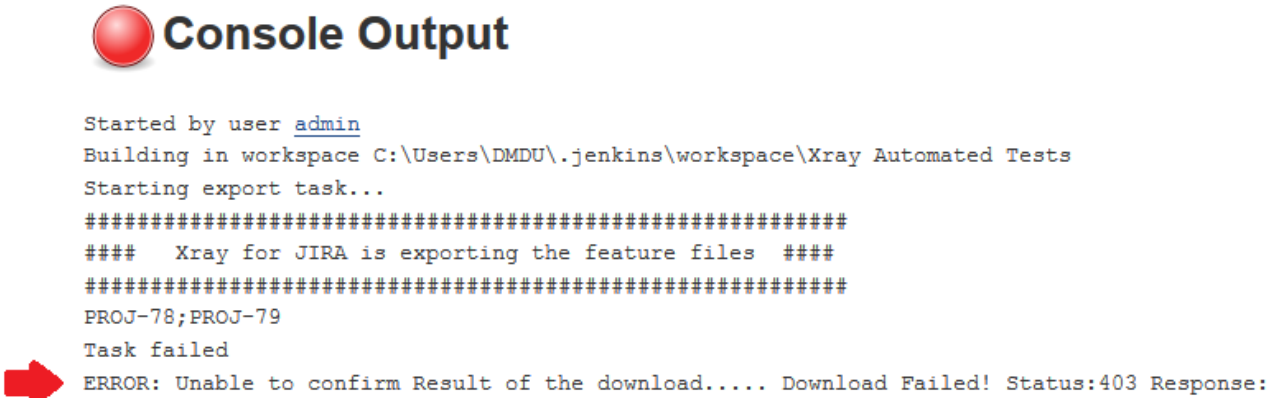


This is the simplest way to generate your step script, and we strongly recommend the use of this snippet due to the complexity of some task related parameters.

Troubleshooting

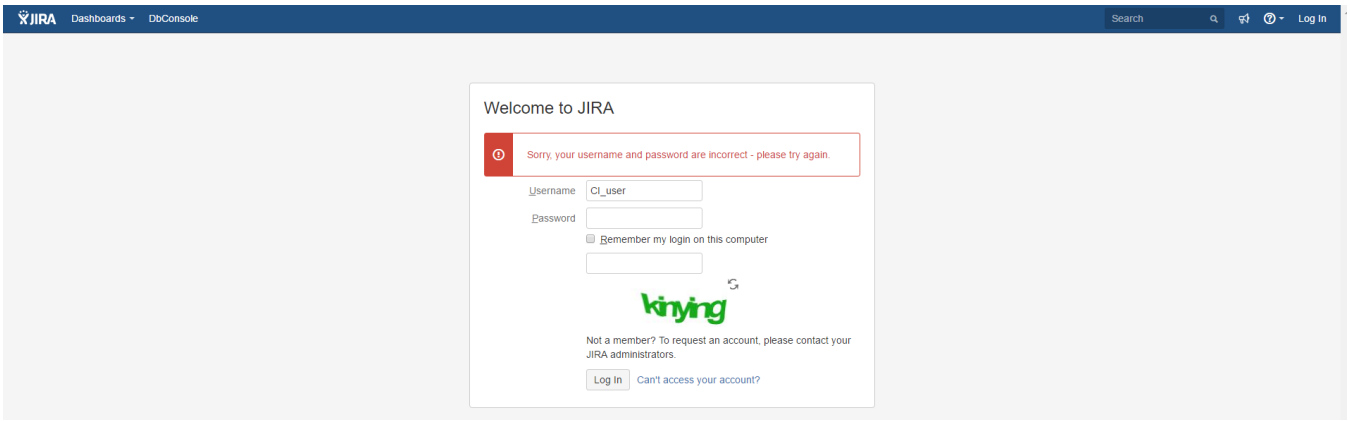
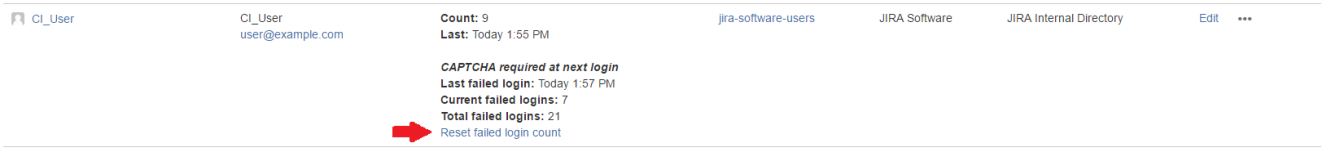
*The build process is failing with status code **403***

When you check the log, it has the following:



By default, when you successively try to log into Jira with the wrong credentials, the Jira instance will prompt you to provide a CAPTCHA the next time you try to log in. It is not possible to provide this information via the build process, so it will fail with status code **403 Forbidden**.

You will need to log into Jira via the browser and provide the CAPTCHA.



If you are a Jira administrator, you can go to Jira administration > User Management and reset the failed login.