

Jenkins pipeline integration



What you'll learn

Prerequisites

- [Jenkins Snippet Generator](#) How to Configure the Jenkins pipeline
- [Examples](#) Know and understand the different tasks available
 - [Validate that the test results are available in Jira](#)
 - [Create new Test Execution](#)
 - [Updating Test Execution](#)
 - [Pipeline step importing Junit multipart](#)
 - [Pipeline step importing Junit multipart \(inline JSON\)](#)
 - [Pipeline steps for Cucumber workflow \(Xray as master\)](#)
 - [Create new Test Execution](#)
 - [Updating Test Execution](#)
- [Tips](#)

Source code for this tutorial

- [code is available in GitHub](#)
- [Pipeline steps for Cucumber workflow using multipart endpoint \(Xray as master\)](#)

Overview

Jenkins is an orchestration tool that is mostly used in CI/CD scenarios. Jenkins allows you to configure it using the UI available or to use pipeline as code.

Xray has made available plugins and tasks that enable you to configure and use the tasks to import tests results to Xray or to import/export cucumber features directly from Jenkins.

Prerequisites

For this example we will use [Jenkins plugin](#), that will allow your Jenkins to ship test results directly to Xray.

What you need:

- A Jira instance with Xray installed and configured
- A Jenkins server with the Xray plugin installed

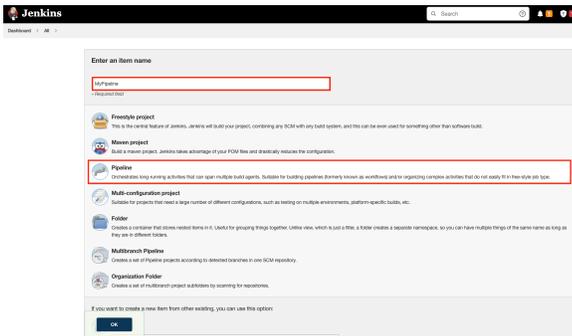
Jenkins Snippet Generator

Jenkins has a Snippet Generator that will assist in the configuration of the pipeline.

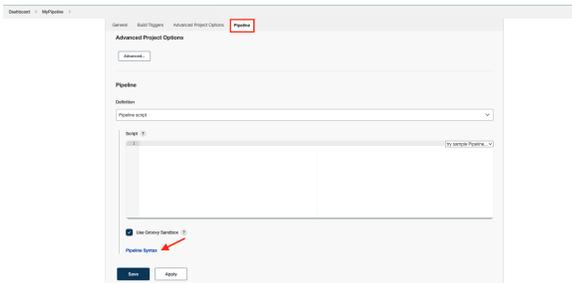
You have two ways of reaching the Snippet Generator: either when creating a new pipeline through the UI or editing a pipeline already created.

Creating a new pipeline

Using the UI you can choose to create a new pipeline and choose the Pipeline type, once you have provided a name for the pipeline and pressed the OK button you will be taken to the configuration panels.



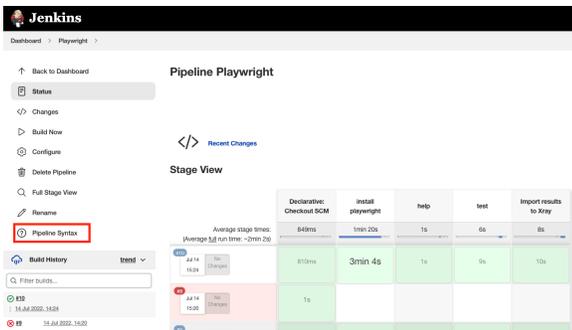
Whenever scrolling down or choosing the Pipeline tab, you will see a link called Pipeline Syntax that will take you to the Snippet Generator.



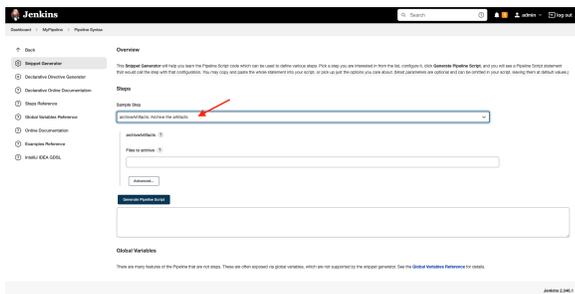
Once there, press the Sample Step dropdown to see all options you have available.

Edit an existent pipeline

If you already have a Jenkinsfile being imported from your source control or if you have already created the pipeline you can reach the Snippet Generator using the left menu entry called Pipeline Syntax in the configuration of your build.



This will take us to the same configuration panel of the Snippet Generator.



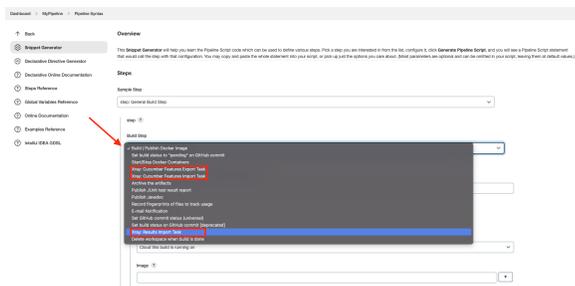
Choose Step: General Build Step



You can see that all the steps available will appear in the Build Step dropdown.

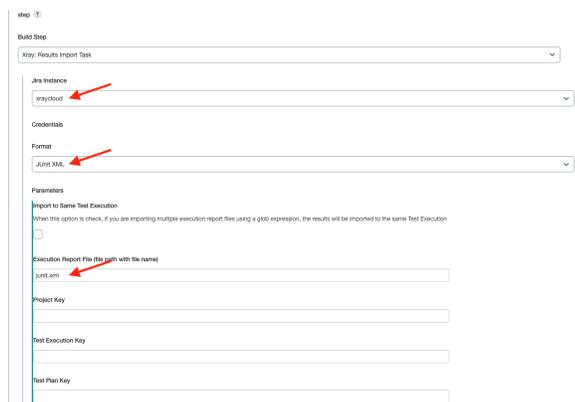
We can see the three steps available:

- **Xray: Cucumber Features Export Task** - Export feature files from Jira to your Jenkins job workspace
- **Xray: Cucumber Features Import Task** - Import feature files from Jenkins to Jira
- **Xray: Results Import Task** - Import test results (JUnit, NUnit, etc...) from your Jenkins job to Jira



You can find more information on the possible values for the parameters of each task [here](#).

In this example we want to import the test results back to Xray so we have chosen **Xray: Results Import Task**.



Test Environments

Revision

Fix Version

Import in parallel
 Import all results files in parallel, using all available CPU cores.

[Click here for more details](#)

Generate Pipeline Script

After filling all the default parameters we can generate the script as a test step ready to be included in your pipeline definition.

Generate Pipeline Script

```
step([$class: 'XrayImportBuilder', endpointName: '/junit', importFilePath: 'xray.xml', importToSameExecution: false, serverInstance: 'CLOUD-360(173-441-451-825-882M5000)'])
```

Examples

In each example we will have the test result file used, the test code (when necessary) and the Jenkinsfile that you can use as a template for your case.

We will showcase different ways to import test results using the Xray Jenkins task and also different workflows that can be used to handle Cucumber tests.

Pipeline step importing Junit

In this case we are importing Junit test results to Xray, this step should be inserted after the test execution step.

This is the step that will import the Junit test results to Xray using the task `XrayImportBuilder`.

Create new Test Execution

In this first example we are creating a new Test Execution in each import with the defined `fixVersion` and `revision`.

Jenkinsfile

```
stage('Import results to Xray') {
  steps {
    step([$class: 'XrayImportBuilder', endpointName: '/junit',
importFilePath: 'xray-report.xml', importToSameExecution: 'true',
projectKey: '<PROJECT_KEY>', fixVersion: '1.2', revision: 'commit
eee455', serverInstance: '<SERVER_INSTANCE>'])
  }
}
```

Updating Test Execution

You can also update a pre-existent Test Execution with the test results as you can see below.

Jenkinsfile

```
stage('Import results to Xray') {
    steps {
        step([$class: 'XrayImportBuilder', endpointName: '/junit',
importFilePath: 'xray-report.xml', importToSameExecution: 'true',
projectKey: '<PROJECT_KEY>', testExecKey: '<TEST_EXECUTION_KEY>',
fixVersion: '1.2', revision: 'commit eee455', serverInstance:
'<SERVER_INSTANCE>'])
    }
}
```

Pipeline step importing Junit multipart

If you need to define more information to the Test Execution, for example, labels, summary, components, environments or associate it to a Test Plan you can by using the multipart request that receives the test result file and two extra files where we can define those details.



For this example we are using pre defined **components** and **environments** that **need to exist in order to assure the import will be successful**.

You can adapt to your reality and replace the **components** and **environments** by the ones that exist in your project or create them, for this case we have two components: `Pets` and `Models` and referring to one environment: `firefox`.

Jenkinsfile

```
stages {
    stage('Import results to Xray (multipart)') {
        steps {
            step([$class: 'XrayImportBuilder', endpointName: '/junit
/multipart', importFilePath: 'importJunitMultipart/*.xml',
importToSameExecution: 'true', projectKey: '<PROJECT_KEY>',
serverInstance: '<SERVER_INSTANCE>', importInParallel: 'true', importInfo:
'importJunitMultipart/my-test-exec-info.json', testImportInfo:
'importJunitMultipart/my-test-import-info.json', inputInfoSwitcher:
'filePath', inputTestInfoSwitcher: 'filePath' ])
        }
    }
}
```

Notice that we are including 4 new parameters:

- **importInfo** - Where you define extra labels and the project to associate this run with.
- **inputInfoSwitcher** - Defining the origin of the `importInfo` that can have two values: `filePath` when you are including the information in form of a file or `fileContent` if you are including the content inline.
- **testImportInfo** - Where you define extra parameters such as summary, associate with components or environments.
- **inputTestInfoSwitcher** - Defining the origin of the `testImportInfo` that can have two values: `filePath` when you are including the information in form of a file or `fileContent` if you are including the content inline (we have an example of this usage in the next section).

Resuming: in the `importInfo` field we are passing the below file defining the project we want to associate the execution and adding a label.

my-test-exec-info.json

```
{
  "fields": {
    "project": {
      "key": "<PROJECT_KEY>"
    },
    "labels": ["firefox"]
  }
}
```

In the `testImportInfo` we are defining a new summary, associating to two components and one Test Plan and defining the environment.

my-test-import-info.json

```
{
  "fields": {
    "project": {
      "key": "EWB"
    },
    "summary": "Login validation [Firefox]",
    "issuetype": {
      "name": "Test Execution"
    },
    "components": [
      {
        "name": "Pets"
      },
      {
        "name": "Modules"
      }
    ]
  }
}
```

Pipeline step importing Junit multipart (inline JSON)

The difference between this request to import results and the previous is that instead of passing the the content of the `ImportInfo` in a file we are defining it inline.

Requirements

For this example we are using **Components** that exist in the Project: `Pets`, `Modules`. If you do not have them defined please do in order to have this example working (or redefine the values in the file to Components that exist in your Project).

We are also using **environments** that must exist in the project before performing the upload, in our case: `firefox`.

Notice that `importInfo` is defined inline in the Jenkins stage, this will allow, for example, to use pre defined Jenkins variables such as `${BUILD_NUMBER}`.

Jenkinsfile

```
stages {
  stage('Import results to Xray (multipart)') {
    steps {
      step([$class: 'XrayImportBuilder', endpointName: '/junit
/multipart', importFilePath: 'importJunitMultipartInline/*.xml',
importToSameExecution: 'true', projectKey: '<PROJECT_KEY>',
serverInstance: '<SERVER_INSTANCE>', importInParallel: 'true',
testImportInfo: 'importJunitMultipartInline/my-test-import-info.json',
inputTestInfoSwitcher: 'filePath', inputInfoSwitcher: 'fileContent',
importInfo: ""{
  "fields": {
    "project": {
      "key": "<PROJECT_KEY>"
    },
    "summary": "Test Execution for java junit ${BUILD_NUMBER}",
    "issuetype": {
      "name": "Test Execution"
    },
    "components" : [
      {
        "name": "Pets"
      },
      {
        "name": "Modules"
      }
    ]
  }
}""])
    }
  }
}
```

The content of the second file remains identical.

my-test-exec-info.json

```
{
  "fields": {
    "project": {
      "key": "<PROJECT_KEY>"
    },
    "summary": "Login validation [Firefox]",
    "issuetype": {
      "name": "Test Execution"
    },
    "components" : [
      {
        "name": "Pets"
      },
      {
        "name": "Modules"
      }
    ]
  }
}
```

Pipeline steps for Cucumber workflow (Xray as master)

When using Cucumber tests you need to define the flow you want to use, this is primarily decided by the place you are editing your cucumber scenarios. You can have two options, one is to define and edit scenarios in Xray (in your Jira instance), if you do so this means that all changes in scenarios are done in Xray and then exported to your IDE or CI/CD tool. The second option is to define the scenarios in your code, in this case each time you create or edit scenarios in your code you need to keep Xray synchronized so you need to import the changes into Xray.

For the example we are considering Xray as the source of truth, so all changes of scenarios are done in Xray and exported to Jenkins to be executed.



Notice that for this example to work, you will need to have a cucumber scenario in Xray that you can then export.

Create new Test Execution

In this flow centred in Xray we need an extra step to extract the scenario from Xray to Jenkins (where we have previously extracted the code also), then we will run the tests and import the test execution results back to Xray into a new Test Execution.

The first stage is to export the feature file from Xray to Jenkins, for that we define the server instance that we want to extract those features from and the issue (or issues list separated by ;), this can be also done using a Jira filter (for that you need to replace `issues` with `filter` and pass the filter key).

More information about the export stage [here](#).

Jenkinsfile

```
stages {
  stage('Export feature files') {
    steps {
      step([$class: 'XrayExportBuilder', issues: '<CUCUMBER_ISSUE_KEY>',
serverInstance: '<SERVER_INSTANCE>'])
    }
  }
  stage('Run tests') {
    steps {
      echo 'Testing..'
    }
  }
  stage('Import results to Xray') {
    steps {
      step([$class: 'XrayImportBuilder', endpointName: '/cucumber',
importFilePath: 'CucumberFlowXrayMasterNewTestExec/login-feature.json',
importToSameExecution: 'true', projectKey: '<PROJECT_KEY>', fixVersion:
'1.2', revision: 'commit eee455', serverInstance: '<SERVER_INSTANCE>'])
    }
  }
}
```

The last stage is a normal import of the test results to Xray, this time using cucumber end point as we are importing cucumber json results.

Updating Test Execution

If you want to update a Test Execution instead of creating a new one you must include the parameter: `testExecKey`, indicating what is the Test Execution issue that will updated with the results uploaded.

Like the previous example we are extracting the scenarios defined in Xray into Jenkins (where the code is also present), executing the tests and importing the test results into a pre-existent Test Execution.

Jenkinsfile

```
stages {
  stage('Export feature files') {
    steps{
      step([$class: 'XrayExportBuilder', issues:
'<CUCUMBER_ISSUE_KEY>', serverInstance: '<SERVER_INSTANCE>'])
    }
  }
  stage('Run tests') {
    steps {
      echo 'Testing..'
    }
  }
  stage('Import results to Xray') {
    steps {
      step([$class: 'XrayImportBuilder', endpointName: '/cucumber',
importFilePath: 'CucumberFlowXrayMasterUpdateTestExec/login-feature.json',
importToSameExecution: 'true', projectKey: '<PROJECT_KEY>', testExecKey:
'<TEST_EXECUTION_KEY>', fixVersion: '1.2', revision: 'commit eee455',
serverInstance: '<SERVER_INSTANCE>'])
    }
  }
}
```

Pipeline steps for Cucumber workflow using multipart endpoint (Xray as master)

When using Cucumber tests you need to define the flow you want to use, this is primarily decided by the place you are editing your cucumber scenarios. You can have two options, one is to define and edit scenarios in Xray (in your Jira instance), if you do so this means that all changes in scenarios are done in Xray and then exported to your IDE or CI/CD tool. The second option is to define the scenarios in your code, in this case each time you create or edit scenarios in your code you need to keep Xray synchronized so you need to import the changes into Xray.

For the example we are considering the first option described, Xray as master, so all changes of scenarios are done in Xray and exported to Jenkins to be executed.

Requirements

Notice that for this example to work you will need to have a cucumber scenario in Xray available to export.

For this example we are using **Components** that exist in the Project: `Pets`, `Modules`. If you do not have them defined please do in order to have this example working (or redefine the values in the file to Components that exist in your Project).

We are also using **environments** that must exist in the project before performing the upload, in our case: `firefox`.

What differs from the previous example is that in this one we want to define some extra information to be added to Xray, such as adding labels, defining a summary or associate to pre-existent components. This is only achievable using the multipart endpoint where we can define two extra files with that extra information.

As previously mentioned, we will have a step to extract the cucumber scenarios from Xray into your Jenkins, one step to execute the tests, and one final step to import the test results back into Xray. In this last importation of results we are using the multipart endpoint that allows adding two files with extra information.

Jenkinsfile

```
stages {
  stage('Export feature files') {
    steps{
      step([$class: 'XrayExportBuilder', issues:
'<CUCUMBER_ISSUE_KEY>', serverInstance: '<SERVER_INSTANCE>'])
    }
  }
  stage('Run tests') {
    steps {
      echo 'Testing..'
    }
  }
  stage('Import results to Xray') {
    steps {
      step([$class: 'XrayImportBuilder', endpointName: '/cucumber
/multipart', importFilePath: 'CucumberFlowXrayMasterMultipart/login-
feature.json', importToSameExecution: 'true', projectKey: '<PROJECT_KEY>',
fixVersion: '1.2', revision: 'commit eee455', importInfo: 'my-test-exec-
info.json', testImportInfo: 'my-test-import-info.json',
inputTestInfoSwitcher: 'filePath', inputInfoSwitcher: 'filePath',
serverInstance: '<SERVER_INSTANCE>'])
    }
  }
}
```

One of the extra files contains the project that we want to use in Jira and a labels list that we want to associate with the execution.

my-test-exec-info.json

```
{
  "fields": {
    "project": {
      "key": "<PROJECT_KEY>"
    },
    "labels" : ["firefox"]
  }
}
```

The other file allows to define a summary, associate to one or a list of pre-existent components. We have special field called `xrayFields` where we define the Test Plan we want to associate to this test run and define an environment (or list of environments) to be set in the execution.

my-test-import-info.json

```
{
  "fields": {
    "project": {
      "key": "<PROJECT_KEY>"
    },
    "summary": "Login validation [Firefox]",
    "issuetype": {
      "name": "Test Execution"
    },
    "components" : [
      {
        "name": "Pets"
      },
      {
        "name": "Modules"
      }
    ]
  }
}
```

Tips

- after results are imported in Jira, Tests can be linked to existing requirements/user stories, so you can track the impact of their coverage.
- results from multiple builds can be linked to an existing Test Plan in order to facilitate the analysis of test result trends across builds.
- results can be associated with a Test Environment, in case you want to analyze coverage and test results by that environment later on. A Test Environment can be a testing stage (e.g. dev, staging, preprod, prod) or an identifier of the device/application used to interact with the system (e.g. browser, mobile OS).
- [Overview](#)
- [Prerequisites](#)
- [Jenkins Snippet Generator](#)
- [Examples](#)
 - [Pipeline step importing Junit](#)
 - [Create new Test Execution](#)
 - [Updating Test Execution](#)
 - [Pipeline step importing Junit multipart](#)
 - [Pipeline step importing Junit multipart \(inline JSON\)](#)
 - [Pipeline steps for Cucumber workflow \(Xray as master\)](#)
 - [Create new Test Execution](#)
 - [Updating Test Execution](#)
 - [Pipeline steps for Cucumber workflow using multipart endpoint \(Xray as master\)](#)
- [Tips](#)