Integration with TeamCity



- Overview
- Release Notes
- Installation
  - Manual Installation
  - Configuration
    - Jira servers
    - Jira Cloud
- Build Steps
  - Xray: Cucumber Features Export Task
    - Configuration
  - Xray: Cucumber Features Import Task
  - Xray: Results Import Task
    - Configuration
    - Additional fields
- 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
- Troubleshooting
  - The build process is failing with status code 403
  - Invalid JIRA instance

Overview

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

> **Please note**
>
> Keep in mind that some endpoints may not be available for Xray Cloud, just for Xray for Jira server.

Release Notes

- Xray TeamCity Plugin 1.1.5 Release Notes
- Xray TeamCity Plugin 1.1.4 Release Notes
- Xray TeamCity Plugin 1.1.3 Release Notes
- Xray TeamCity Plugin 1.1.2 Release Notes

- Xray TeamCity Plugin 1.1.1 Release Notes
- Xray TeamCity Plugin 1.1.0 Release Notes
- Xray TeamCity 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.

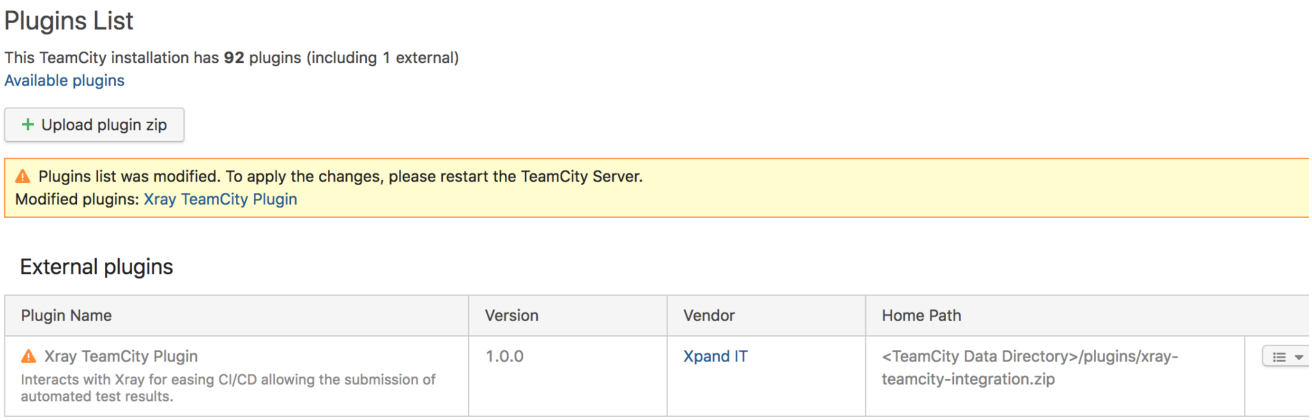
> **Requirements**
>
> The TeamCity baseline for this add-on is 2018.1 and it may not work properly with previous versions.

Manual Installation

1. Upload the plugin

> **Download the latest version of the TeamCity Plugin**
>
> You may download the latest version of the TeamCity plugin from the latest Release Notes.

2. Restart TeamCity (if needed as shown in the screenshot below)



Configuration

Xray for TeamCity is configured in the global settings configuration page **Administration** > **Integrations > Xray**.

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**: a friendly name for the configuration
2. **Server or Cloud:** Server
3. **Server Address**: The address of the Jira Server where Xray is running
4. Authentication:
    a. **User**: username
    b. **Password**.

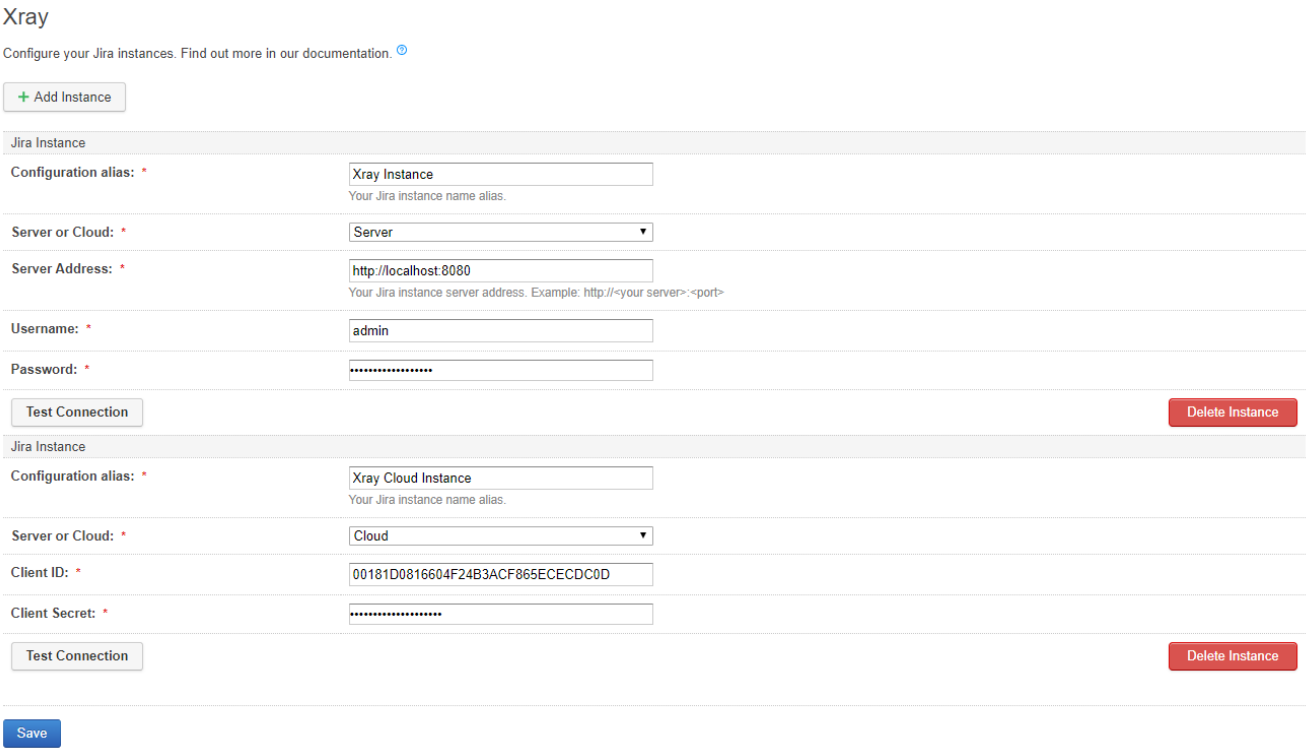
> **Please note**
>
> The user present in this configuration must exist in the JIRA instance and have permission to Create Test and Test Execution Issues

Jira Cloud

The Jira Cloud configuration defines connections with Jira Cloud.

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

1. **Configuration alias**: a friendly name for the configuration
2. **Server or Cloud**: Cloud
3. Authentication:
    a. **Client ID**: obtained from Xray Cloud (more info here)
    b. **Client Secret**: obtained from Xray Cloud (more info here)



Build Steps

Build steps are the building blocks of the build process. These need to be defined in the build 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 build step which publishes the execution results back to Jira.

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: server or cloud).

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 documentation for the server or cloud) 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 is only available for the server and 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 for the server or cloud). Therefore, it mimics the endpoints input parameters.

It supports importing results in Xray's own JSON format, Cucumber, Behave, JUnit, and NUnit, among others. However, there are some endpoints in the server that may not yet be available for Xray Cloud.

These are the available endpoints.

	Server	Cloud
Xray JSON	✓	✓
Cucumber JSON	✓	✓
Cucumber JSON multipart	✓	✗
Behave JSON	✓	✗
Behave JSON multipart	✓	✗
JUnit XML	✓	✓
JUnit XML multipart	✓	✗
NUnit XML	✓	✓
NUnit XML multipart	✓	✗
Robot XML	✓	✗
Robot XML multipart	✓	✗
Compressed .zip file	✓	✗
TestNG XML	✓	✓
TestNG XML multipart	✓	✗

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 "Glob" expressions are supported for: • JUnit • TestNG • NUnit • Robot framework

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	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)

Examples

Cucumber

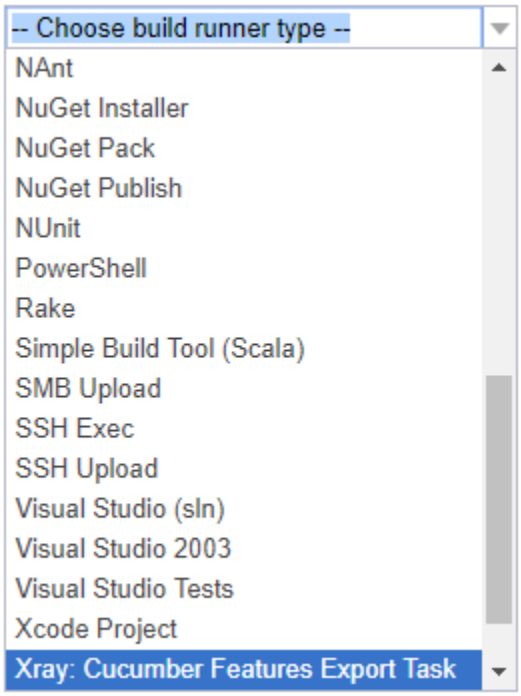
In a typical Cucumber Workflow, after having created a Cucumber project and the Cucumber tests specified in Jira, you may want to have a project that **exports** the features from Jira, executes the automated tests on a CI environment and then **imports** back its results.

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**

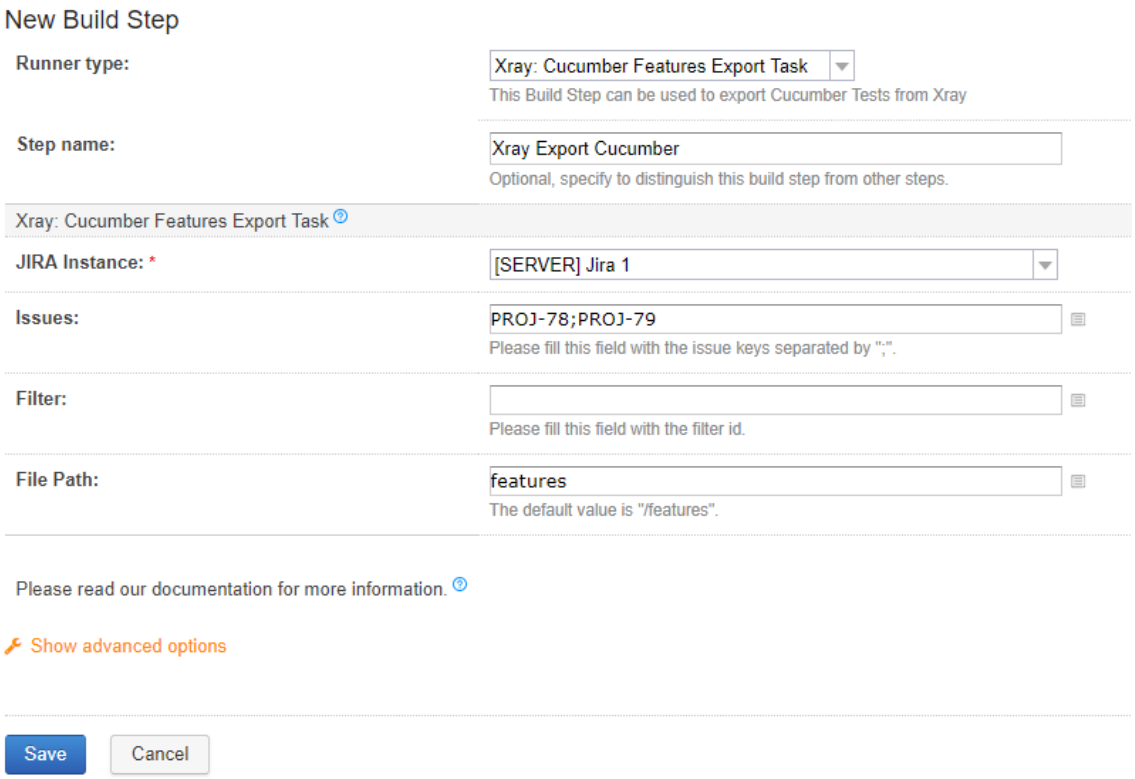
Exporting Cucumber features

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



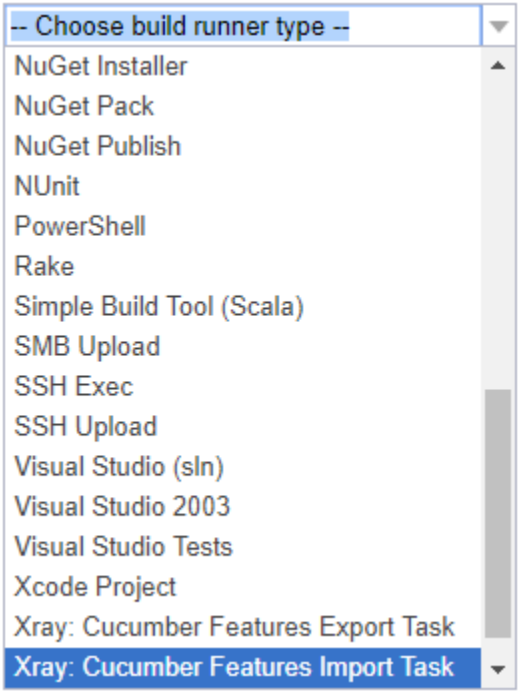
After that, configure it.

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.



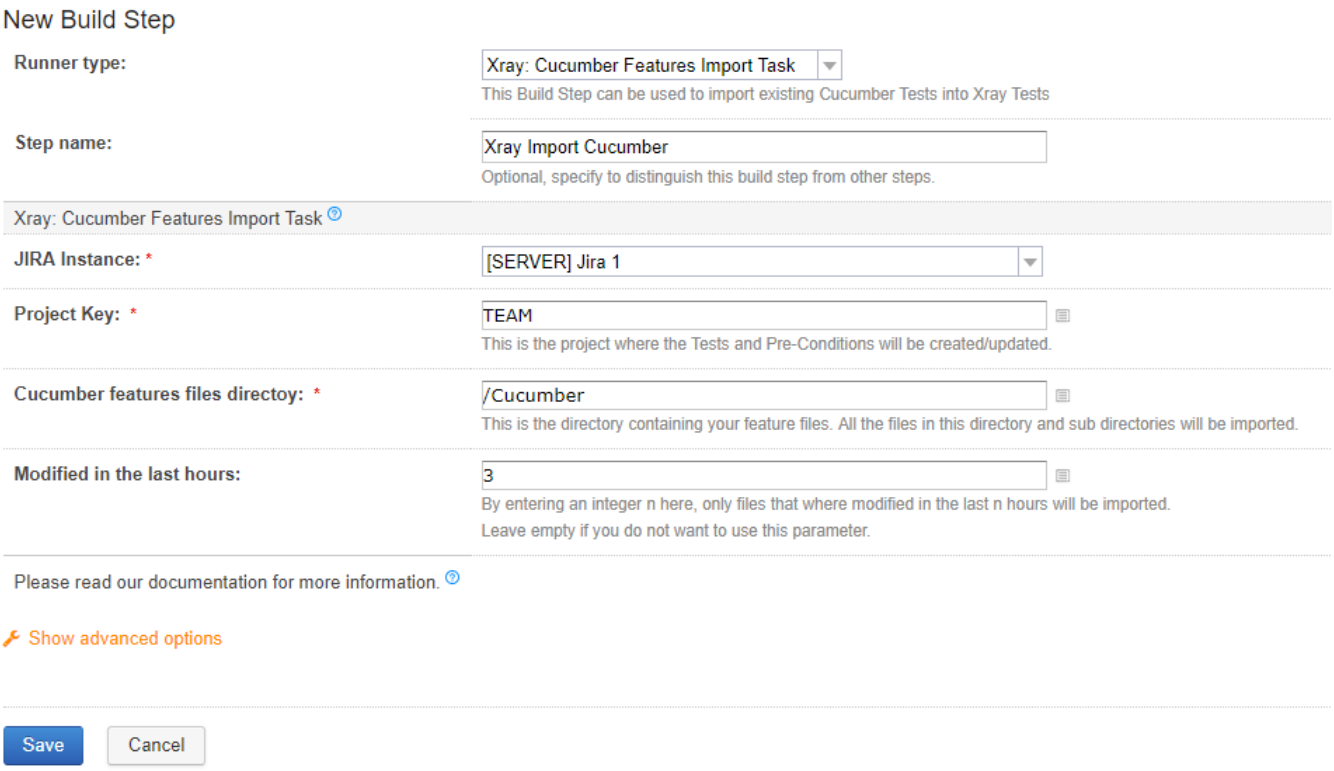
Importing Cucumber features

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



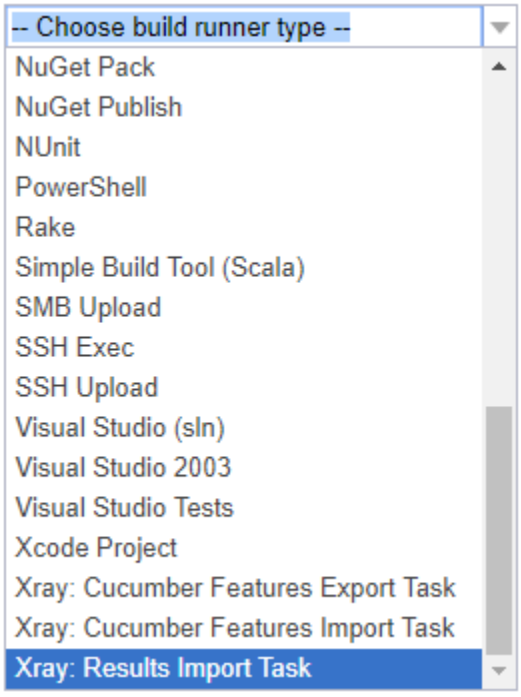
After that, configure it.

In this example, we configured the task to import to the Project TEAM 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.



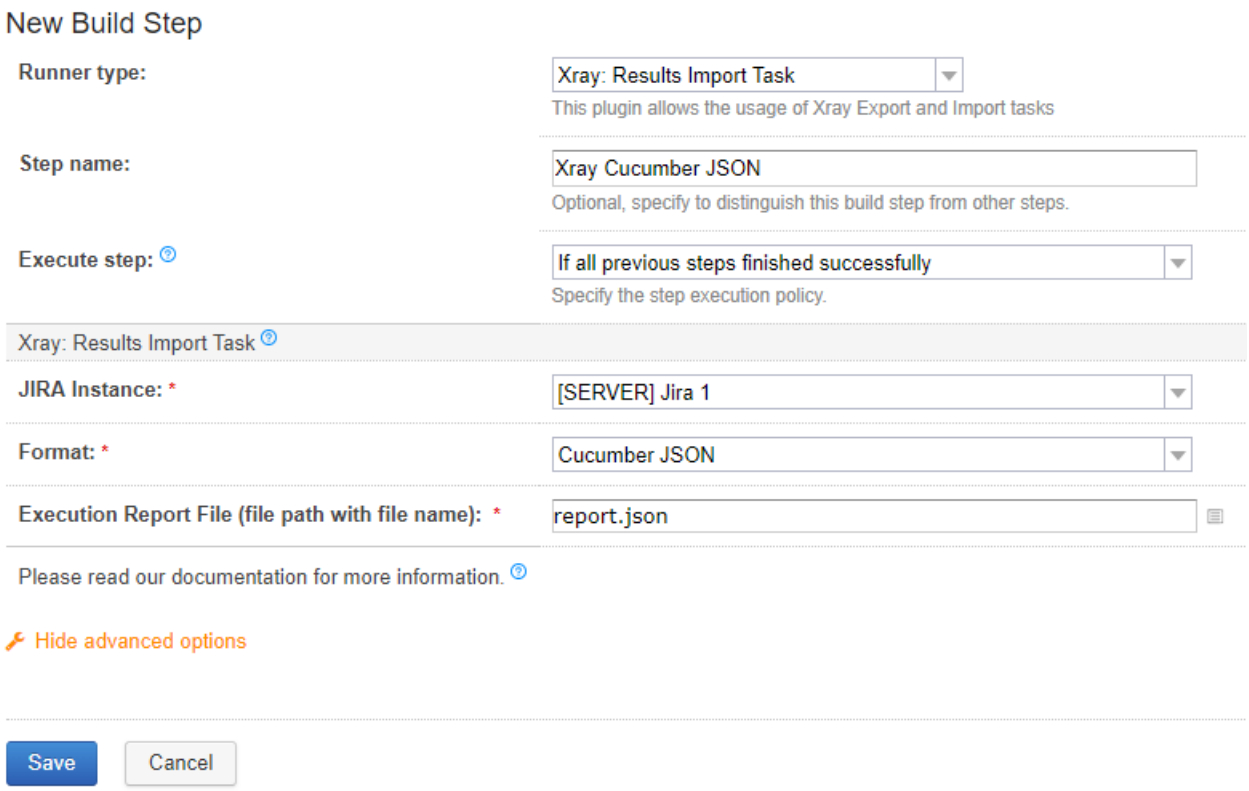
Importing the execution results

To start the configuration, add a build step *Xray: Results Import Task*.



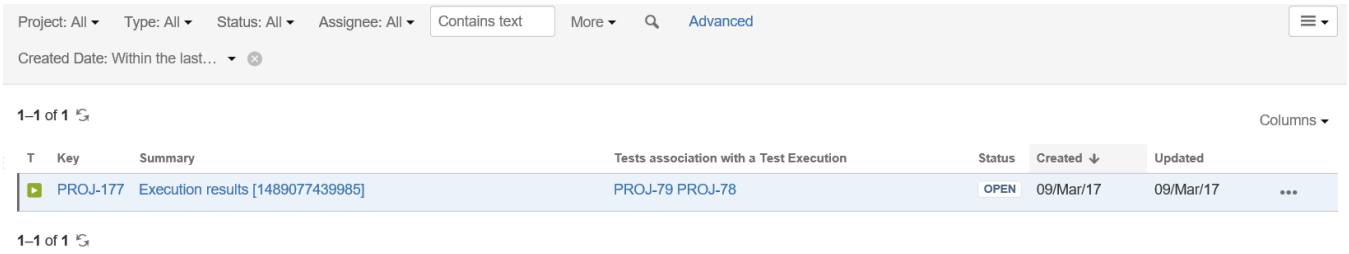
After that, you may configure it.

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.

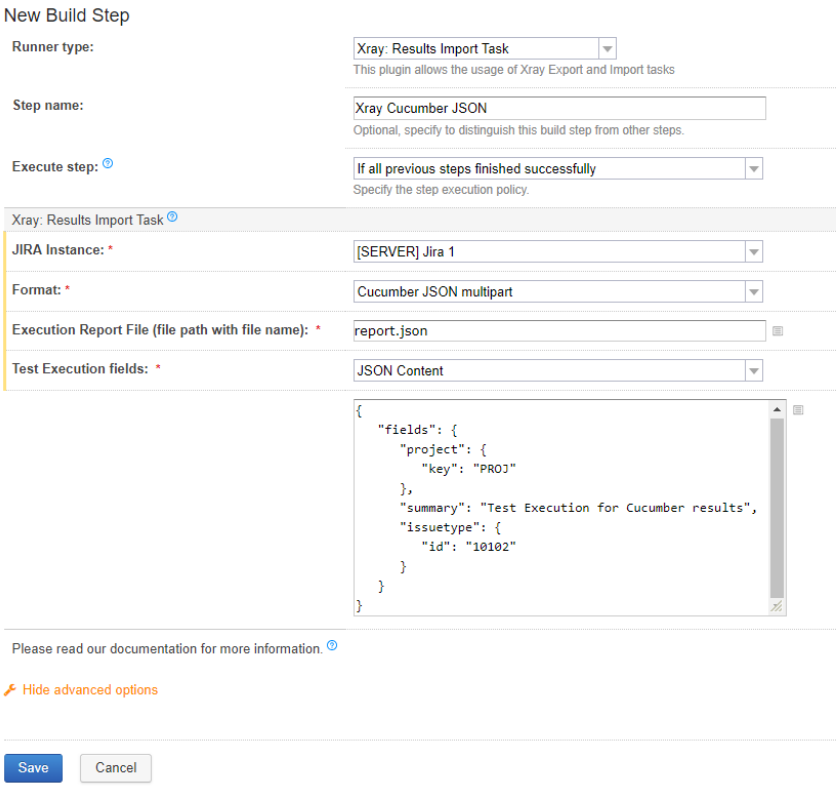
For this scenario and example, the import task needs to be configured with the **Cucumber JSON Multipart** format. When selecting this option, you can additionally configure the *Test Execution fields* in one of two ways:

- 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",
      "issuetype": {
         "id": "10102"
      }
   }
}
```

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



Once all configurations are done, click Save.

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 build step configuration.

JUnit

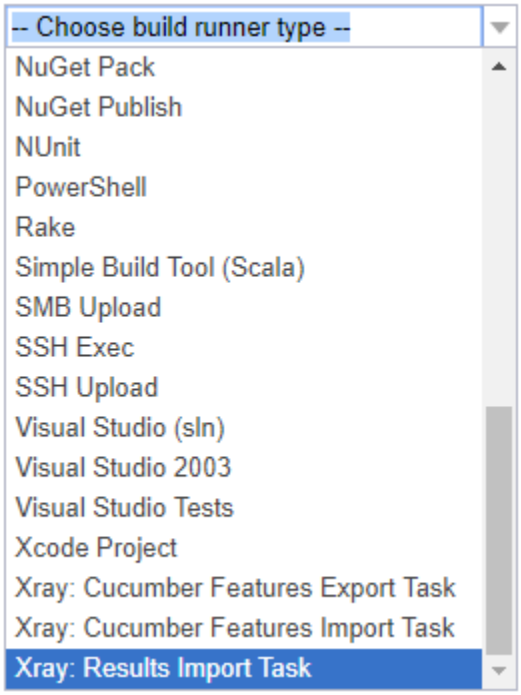
Apart from supporting Cucumber natively, Xray enables you to take advantage of many other testing frameworks like JUnit. In this sense, Xray for Jenkins lets you import results in other formats besides Cucumber JSON.

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 build step *Xray: Results Import Task*.

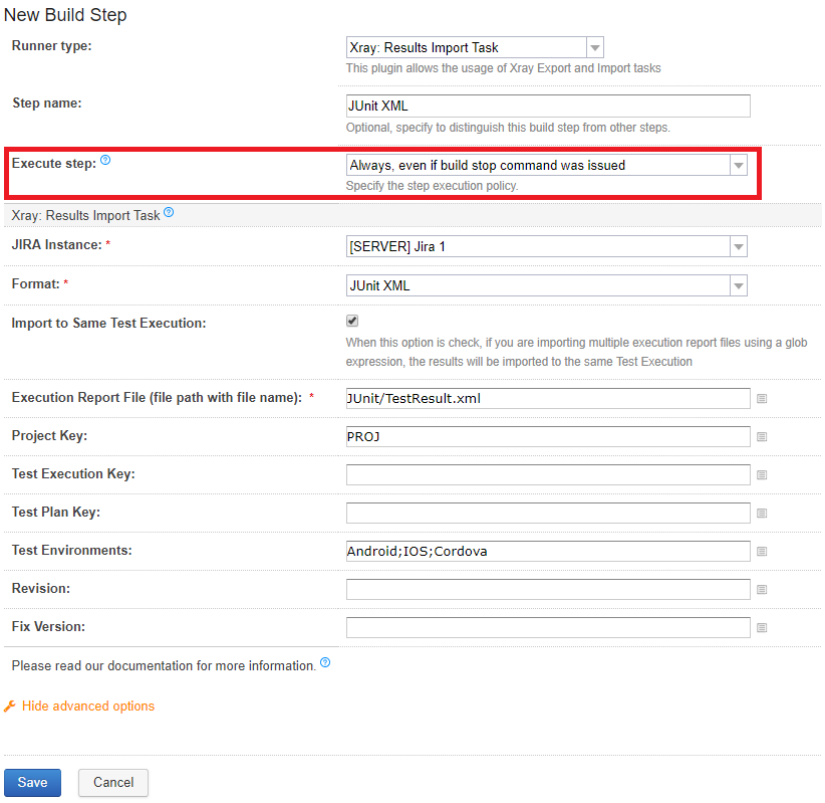


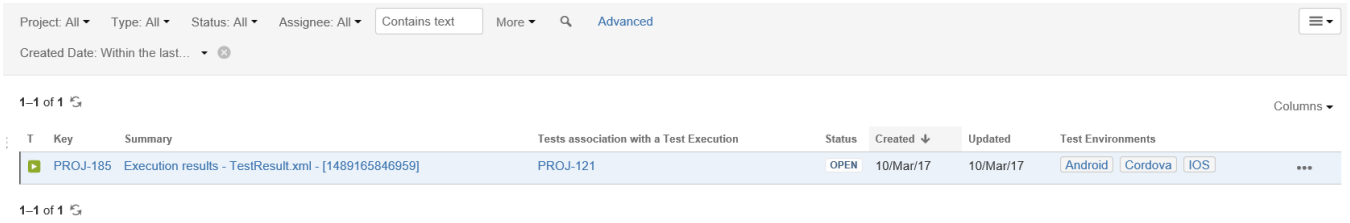
After that, configure it.

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

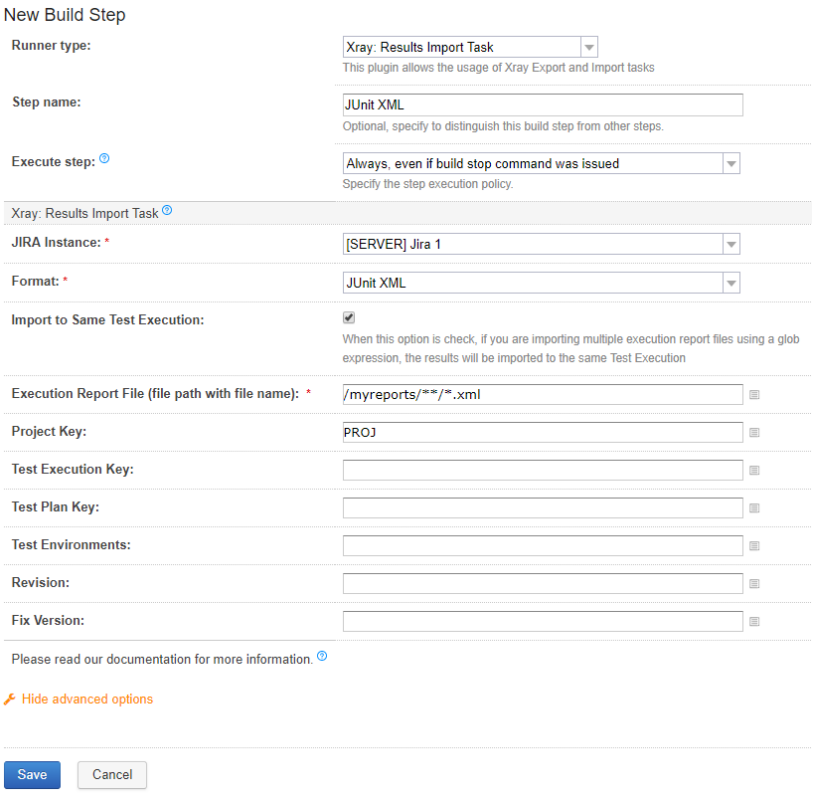
After running the plan, the expected result is a new Test Execution issue created in the JIRA instance.

Make sure to select "Always, even if build stop command was issued" so, even some previous step fail, the tests will still be imported.





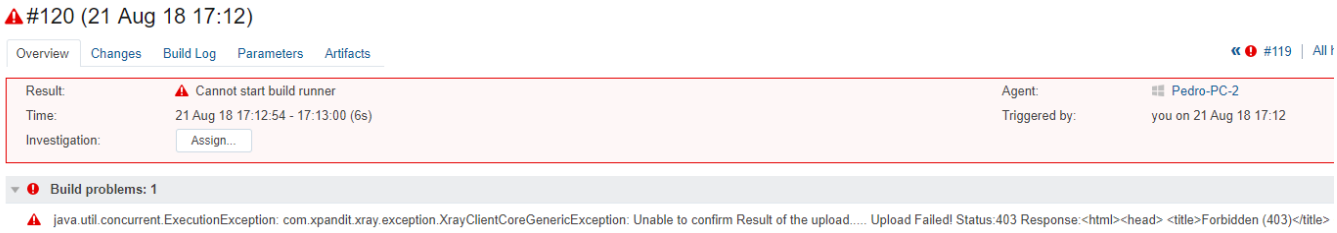
You can also import multiple results using a glob expression, like in the following example



Troubleshooting

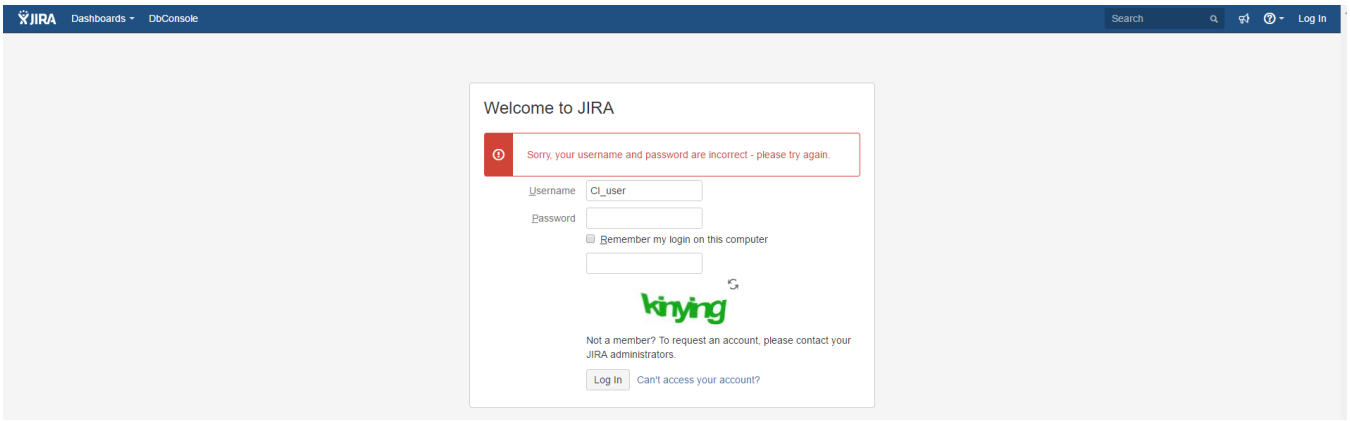
*The build process is failing with status code **403***

When you check the log, it has the following:

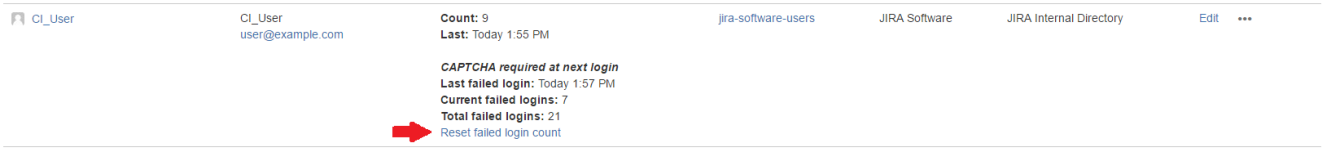


By default, when you successively try to log into Jira with the wrong credentials, the Jira instance will prompt you to provide a CAPTCHA the next time you try to log in. It is not possible to provide this information via the build process, so it will fail with status code **403 Forbidden**.

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

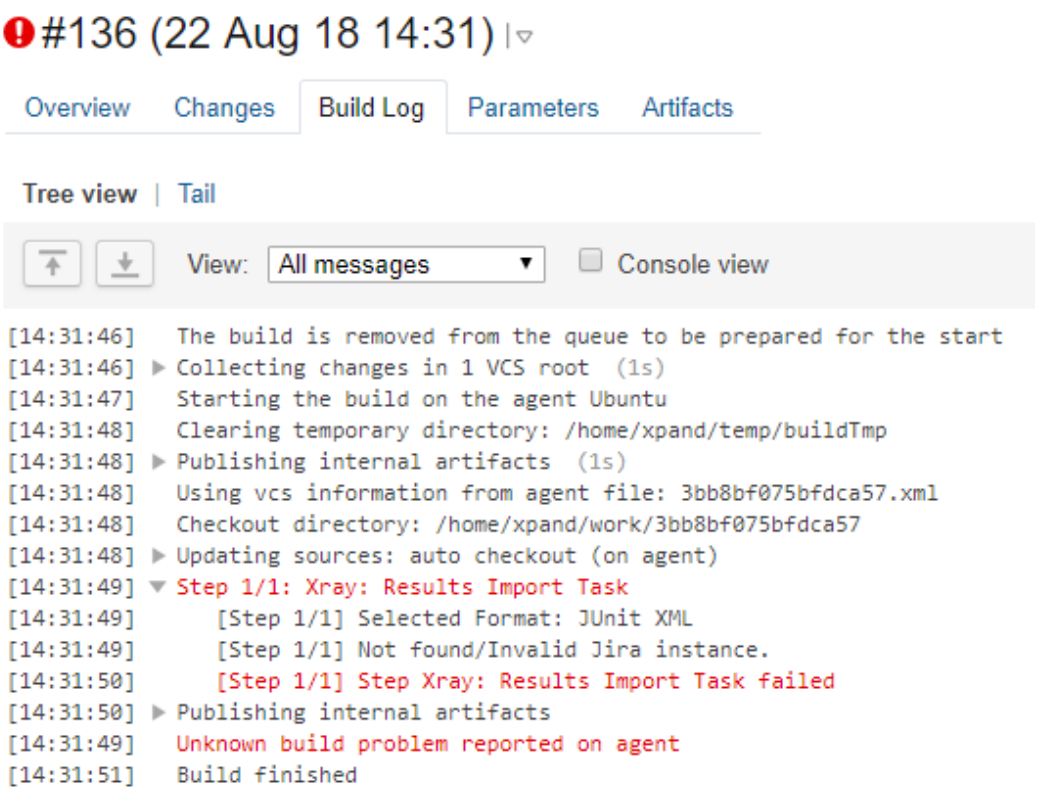


If you are a Jira administrator, you can go to Jira administration > User Management and reset the failed login.



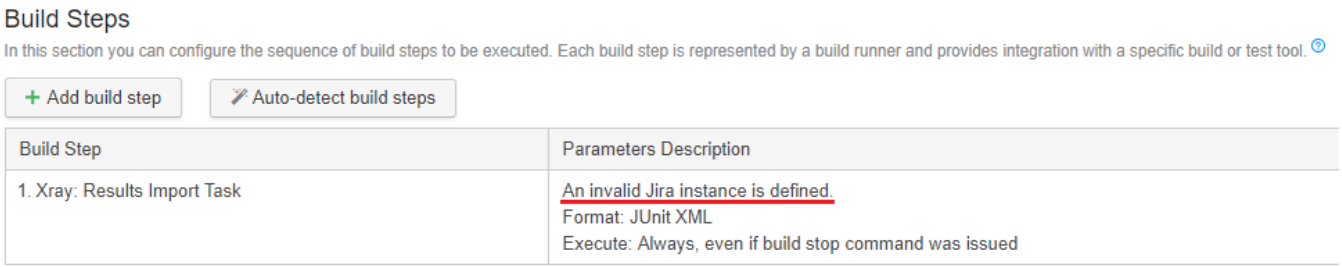
Invalid JIRA instance

When you check the log, it has the following:



This occurs when a JIRA instance is selected in the build step configuration and that instance is later deleted in the Xray global settings.

This error is shown in the build step.



To solve this, edit the build step, select a valid JIRA Instance and save.