

Trigger a Jenkins Pipeline



What you'll learn

- [Send results back to Xray](#)
 - How to configure the remote jobs triggering feature
- [Steps](#)
 - How to trigger remote jobs from Test Plans
- [Tips](#)
 - How to configure and validate shipping the test results in Jira
- [References](#)

Source-code for this tutorial

- code is available in [GitHub](#)

Overview

In this example we are configuring a Remote Job Trigger for Jenkins that executes Playwright tests and sends the execution results back to Xray.

Prerequisites

For this example we will use [Jenkins](#) as the CI/CD tool that will execute [Playwright](#) tests.

What you need:

- Access to a Jenkins instance
- Xray Enterprise installed in your Jira instance
- Have a Jenkins job that you can adapt/use to invoke remotely
- Understand Jenkinsfile

Configure a new RJT for Jenkins in Xray

This example requires configuration in both sides (Xray and Jenkins) so that we can take advantage of the combination of both tools.

The jenkinsfile will configure a multi-step pipeline that extracts the Playwright test code, executes it and ships the execution results back to Xray.

Configure Jenkins using a jenkinsfile

We use a jenkinsfile to configure the pipeline in Jenkins.

jenkinsfile

```
pipeline {
  parameters {
    string(name: 'projectKey', defaultValue: '')
    string(name: 'testPlanKey', defaultValue: '')
  }
  agent {
    docker {
      image 'mcr.microsoft.com/playwright:v1.27.0-focal'
    }
  }
  stages {
    stage('install playwright') {
      steps {
        sh '''
          npm i -D @playwright/test
          npx playwright install
        '''
      }
    }
    stage('test') {
      steps {
        catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {
          sh '''
            PLAYWRIGHT_JUNIT_OUTPUT_NAME=xray-report.xml npx playwright
test
            '''
          }
        }
      }
    }
    stage('Import results to Xray') {
      steps {
        step([$class: 'XrayImportBuilder', endpointName: '/junit',
importFilePath: 'xray-report.xml', importToSameExecution: 'true',
projectKey: params.projectKey, testPlanKey: params.testPlanKey,
fixVersion: '1.2', revision: '131', serverInstance: '10be58cc-2776-49a7-be60-b615dc99f4c0'])
      }
    }
    stage('Extract Variable from log'){
      steps {
        script {
          def logContent = Jenkins.getInstance().getItemByFullName(env.
JOB_NAME).getBuildByNumber(Integer.parseInt(env.BUILD_NUMBER)).logFile.text
          env.testExecs = (logContent =~ /XRAY_TEST_EXECS:.*/).findAll().
first()
          echo testExecs
        }
      }
    }
  }
  post {
    {
      always {
        junit '*.xml'
      }
    }
  }
}
```

On the above jenkinsfile we are defining two parameters that will be passed when the build is invoked.

jenkinsfile

```
...
parameters {
    string(name: 'projectKey', defaultValue: '')
    string(name: 'testPlanKey', defaultValue: '')
}
...
```

The received parameters can be used in the remaining steps of the pipeline. In order to define what are the parameters we have added a parameters section with the name of the parameter and a possible default value.

Within the step to run the tests we have added a function to make sure that the next step is performed even if the actual step fails. This is important to assure that the results are sent to Xray.

jenkinsfile-failure

```
...
catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {
...
}
```

Using the `catchError` method in case of an error, we are forcing the step to continue the Pipeline execution from the statement following the `catchError` step.

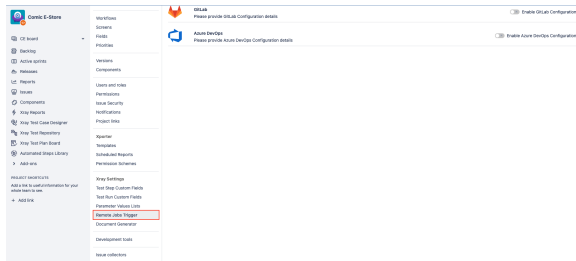
The behavior of the step when an exception is thrown can be configured to print a message, set a build result other than failure, change the stage result, or ignore certain kinds of exceptions that are used to interrupt the build.

Once the jenkinsfile is uploaded in Jenkins it is ready to start to perform the tasks defined in the stages and steps.

We are going to dive into the last part of the jenkinsfile, where we send the results back to Xray, in another section.

Configure a Remote Jobs Trigger in Xray for Jenkins

In order to use the configuration of a RJT in Xray you need to access the *Project Settings* area and click on the *Remote Jobs Trigger* option.



This will open the *Remote Jobs Trigger* configuration page where we can configure remote jobs for Jenkins, Bamboo, GitLab, GitHub and Azure DevOps. You can activate and deactivate the configurations by switching the toggles next to each.

Project settings

Summary
Details
AuthNrg
Remote project
Create project

Issue types
Bug
Task
Pre-Condition
Story
Sub-task
Sub-Test Execution
Task
Test
Test Execution
Test Plan
Test Set

Workflows
Scripts
Fields
Overview

Remote Jobs Trigger
This page contains current configurations for remote jobs triggers

Jenkins
Please provide Jenkins Configuration details

Jenkins Configurations

Configuration Name	Job Name	Last Update	Status	Action
Playwright Tests - TP	Playwright	08 April 2023	<input checked="" type="checkbox"/>	...
OWASP-Playwright Tests - TP	Playwright RUT	04 May 2023	<input checked="" type="checkbox"/>	...
Bamboo Please provide Bamboo Configuration details			<input type="checkbox"/>	Enable Bamboo Configuration
GitLab Please provide GitLab Configuration details			<input type="checkbox"/>	Enable GitLab Configuration
GitLab Please provide GitLab Configuration details			<input type="checkbox"/>	Enable GitLab Configuration
Azure DevOps Please provide Azure DevOps Configuration details			<input type="checkbox"/>	Enable Azure DevOps Configuration

New Jenkins Configuration

Only one configuration can be active at each time.

Once you have activated the configuration that you are interested, in our case Jenkins Configuration, click the *New Jenkins Configuration* to configure your new job. This opens a new form that must filled with proper information.

New Jenkins Configuration

Configuration Name*

Job Name*

API Token*

Username*

cristiano.cunha

API URL*

Password*

Parameters(optional)

Name (eg. content-type) Value (eg. application/ json) X

+ Add

Cancel Add Configuration

Fill the **Configuration Name** with meaningful information that allows you to know what is the purpose of the job just by reading the name.

The **Job Name** is the name of the project/job in Jenkins, make sure that they match.

Next we have the **API Token**, this can be defined in the project configuration in Jenkins. Access your project page in Jenkins and scroll down to the *Build Triggers* area.

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☐ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

☐ Quiet period ?

☒ Trigger builds remotely (e.g., from scripts) ?

Authentication Token

MyToken

Use the following URL to trigger build remotely: `curl -X POST https://jenkins.example.com/job/ProjectName/build?token=MyToken`

Optionally append <cause=Reason> to provide text that will be included in the recorded build cause.

The **Authentication Token** defined in Jenkins must be the same defined in Xray **API Token** of the *Remote Jobs Trigger* configuration, in our case 'MyToken'.

In the **Username** field define the name of the Jenkins user with proper permissions to invoke the Jenkins Job. Make sure it has the right permissions in Jenkins or the job will return an authentication error when used.

Fill the **API URL** with the base URL of your Jenkins instance and in the **Password** field put the password used by the user identified by the Username above.

Finally we have reached the last configuration fields available: Parameters (Optional), that is a list of key /value pairs that you have defined in you Jenkins pipeline and want to pass from Xray.

You can define static ones or you can use dynamic filled fields available in Xray (more info [here](#)). In our case we want to pass the Project key and the Test Plan key so that we can use them when shipping the execution results back to Xray.

For that we use the following options available from Xray:

- **\${PROJECT_KEY}** that will be filled with the key of the project from where the job is called.
- **\${ISSUE_KEY}** that will be filled with the issue key, in our case the Test Plan key, from where the job was started from.

Make sure that the names of the parameters match between what you have configured in the jenkinsfile and the Remote Jobs Triggers configuration.

The configuration with all the fields filled will look like this:

Edit Jenkins Configuration

Configuration Name*

DEMO-Playwright Tests - TP

Job Name*

Playwright-RJT

API Token*

MyToken

Username*

admin

API URL*

7a85-2-82-76-116.ngrok.io

Password*

admin

Parameters(optional)

projectKey

\$(PROJECT_KEY)

X

testPlanKey

\$(ISSUE_KEY)

X

Cancel

Save

Once done click the **Save** button and activate the configuration by switching the **Status** toggle to on:

Remote Jobs Trigger

This page contains current configurations for remote jobs triggers

Jenkins

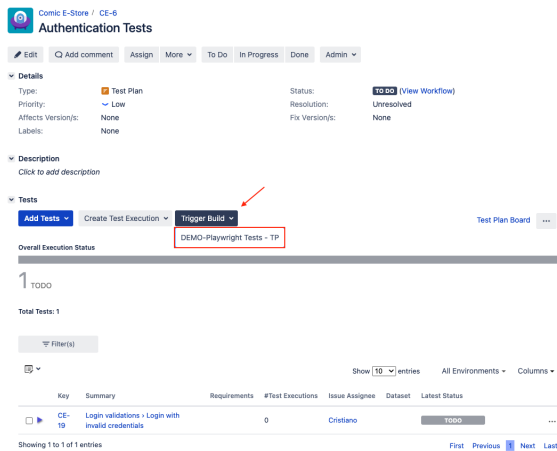
Please provide Jenkins Configuration details

Disable Jenkins Configuration

Jenkins Configurations				
Configuration Name	Job Name	Last Update	Status	Actions
Playwright Tests - TP	Playwright	06 April 2023	Off	...
DEMO-Playwright Tests - TP	Playwright-RJT	04 May 2023	On	...

Trigger

Once the configuration is done and active you can navigate to, e. g. the Test Plan, from where you want to trigger the job, there you will see that a new button is available to trigger the job you just have configured.



When you choose to call the **Remote Jobs Trigger** job a new confirmation window will appear with the status of the invocation, either success or failure.

Remote job configuration trigger

The remote job configuration was triggered with **success**.

Ok



Notice that the status is of the invocation of the job only (not the status of the execution of the job). If you get a failure status please review the configuration in the previous sections.

Send results back to Xray

In the above example we have defined a job in another tool and invoked it from Xray. This job will execute the Playwright tests and generate a Junit report.

Now we need to have those results back into Xray for full visibility. The following section will show you two ways you can use to have those results shipped back into Xray.

Steps

The Xray plugin for Jenkins provides steps that enable you to import the results to Xray.

More information on the Jenkins integration available [here](#).

Let us look into more detail over the step *XrayImportBuilder* available in Jenkins and that can be used in your pipeline definition.

Remember that in our jenkinsfile we have defined parameters in the beginning of the file and one step to import the results into Xray.

jenkinsfile

```
pipeline {
  parameters {
    string(name: 'projectKey', defaultValue: '')
    string(name: 'testPlanKey', defaultValue: '')
  }
  ...
  stage('Import results to Xray') {
    steps {
      step([$class: 'XrayImportBuilder', endpointName: '/junit',
importFilePath: 'xray-report.xml', importToSameExecution: 'true',
projectKey: params.projectKey, testPlanKey: params.testPlanKey,
fixVersion: '1.2', revision: '131', serverInstance: '10be58cc-2776-49a7-
be60-b615dc99f4c0'])
    }
  }
  ...
}
```

Make sure that the parameters names match the ones configured in the *Xray Remote Jobs Trigger* configuration for Jenkins.

With these options we are saying to the Pipeline that we are going to receive two parameters:

- *projectKey*
- *testPlanKey*

On the import step we are using those parameters to define what is the Jira Project that we are import into and what is the Test Plan we will associate the result to.

Let's look into the step in more detail, explaining each option:

- **\$class**: the class used in this step (*XrayImportBuilder*).
- **endpointName**: Xray supports multiple [formats](#) but for this case we are using */junit*
- **importFilePath**: has the file name with the Junit test results that we want to import (*xray-report.xml*).
- **importToSameExecution**:
- **projectKey**: The key from the project that we want to import into (in our case passed from the parameter of the Xray Remote Jobs Triggering job).
- **testPlanKey**: The key from the Test Plan that we want to import into (in our case passed from the parameter of the Xray Remote Jobs Triggering job).
- **fixVersion**: fixVersion to be associated to the results imported.
- **revision**: Revision that we want to associate to the execution results imported.
- **serverInstance**: The server instance id define in Jenkins when we have configured the Jira instance available.

Once the pipeline ends with success you can check details in the console output of the build in Jenkins to make sure all went as expected.



We can see that it has imported the execution results with success and that it has created a new Test Execution: *CE-26* (as well as other relevant information).

When accessing the Test Plan we can see that the results were ingested and the overall status is visible from the detail view of the issue.

Authentication Tests

[Edit](#)
[Add comment](#)
[Assign](#)
[More ...](#)
[To Do](#)
[In Progress](#)
[Done](#)
[Admin ...](#)

Details

Type:	Test Plan	Status:	PASS (Show Workflow)
Priority:	Low	Resolution:	Unassigned
Affects Version(s):	None	Fix Version(s):	None
Labels:	None		

Description

Click to add description

Tests

[Add tests ...](#)
[Create Test Execution](#)
[Trigger Build ...](#)
Test Plan Board ...

Overall Execution Status

2 PASS

Total Tests: 2

[Filter\(s\)](#)

Key	Summary	Requirements	#Test Executions	Issue Assignee	Dataset	Latest Status
CE-7	Unsuccessful login.	CE-1	4	Unassigned		PASS
CE-4	Successful login.	CE-1	2	Unassigned		PASS

Showing 1 to 2 of 2 entries

[First](#) [Previous](#) [Next](#) [Last](#)

Test Executions

[Add Test Executions](#)

Key	Summary	#Plans	Issue Assignee	Status
CE-36	Execution results - http-report.xml / 7683567411932	2	Cristiano	PASS
CE-35	Execution results - http-report.xml / 7683567405047	2	Cristiano	PASS
CE-34	Execution results - http-report.xml / 7683213285096	2	Cristiano	PASS
CE-33	Execution results - http-report.xml / 7683723049738	2	Cristiano	PASS

Shows 1 to 4 of 4 entries

[First](#) [Previous](#) [Next](#) [Last](#)

The screenshot displays the Jira interface for viewing test execution results. The top navigation bar includes 'Jira', 'Add comment', 'Assign', 'Mark as', 'To do', 'In progress', 'Done', and 'Assign to'. The main header shows 'Execution results - xray-report.xml' with a unique ID '[16835516132]'. The left sidebar contains navigation links for 'Details', 'Description', 'History', 'Test Results', 'Test Plan', 'Test Suite', 'Test Case', 'Test Run', 'Test Environment', and 'Test Runners'. The 'Details' section is active, showing a table of test results. The table has columns for 'Name', 'Status', 'Assignee', 'Created', 'Updated', 'Test Case', 'Test Run', 'Test Suite', 'Test Plan', 'Test Environment', and 'Test Runners'. The 'Test Case 1' row is highlighted in green, indicating a 'Pass' status. The 'Test Case 2' row is highlighted in red, indicating a 'Fail' status. The 'Fail' status is further detailed in a dropdown menu showing 'Execution results' and 'Error message with supplementary log'.

The screenshot shows the AWS IAM console interface. At the top, there's a navigation bar with 'IAM' and 'Users' tabs. The main content area is titled 'testuser' and shows the user's details. The 'Groups' section lists the 'AWS_IAM' group. The 'Permissions' section shows the 'AWS_IAM' policy. The 'Access Key' section shows a single access key. The console is in a light blue theme.

Tips

- Confirm that the user that you are using have the correct permissions in both tools.

References

- [Triggering Remote Jobs](#)
- [Jenkins pipeline integration](#)
- [Integration with Jenkins](#)
- [Playwright](#)
- [Overview](#)
- [Prerequisites](#)
- [Configure a new RJT for Jenkins in Xray](#)
 - [Configure Jenkins using a jenkinsfile](#)
 - [Configure a Remote Jobs Trigger in Xray for Jenkins](#)
- [Trigger](#)
- [Send results back to Xray](#)
 - [Steps](#)
- [Tips](#)
- [References](#)