

Integration with Automation for Jira

- [Overview](#)
- [Concepts](#)
- [Usage Examples](#)
 - [Jenkins](#)
 - [Trigger a Jenkins project build from an issue](#)
 - [Jenkins configuration](#)
 - [Automation for Jira configuration](#)
 - [Trigger a Jenkins project build from a Test Plan and report the results back to it](#)
 - [Jenkins configuration](#)
 - [Automation for Jira configuration](#)
 - [Azure DevOps](#)
 - [Trigger a Azure DevOps pipeline from a Test Plan and report the results back to it](#)
 - [Azure DevOps configuration](#)
 - [Automation for Jira configuration](#)
 - [Travis CI](#)
 - [Trigger a TravisCI project build from a Test Plan and report the results back to it](#)
 - [TravisCI configuration](#)
 - [Automation configuration](#)
 - [Generic automation of processes](#)
 - [Copy fields from requirement/Story to Test whenever creating a Test or linking it to a story](#)
 - [Automation configuration](#)
 - [Reopen/transition Tests linked to a requirement whenever the requirement is transitioned or changed](#)
 - [Automation configuration](#)
- [References](#)

Overview

The [Automation for Jira](#) app/feature enables users to easily extend and implement automation in Jira without having to code.

This way, users can implement rules that are triggered upon some event, executed if certain condition(s) are met and that perform certain action(s).

Rules can also be triggered manually or may be scheduled.



Please note

Automation for Jira has become part of Jira Datacenter offering, enabling automation of flows for all Jira users. Please check Atlassian's [Jira Automation documentation](#) for more info.

Since Xray uses issue types for most of its entities and since Xray provides many JQL functions that allow you to obtain testing-related information, Automation for Jira can be used with Xray in a very straightforward way.

Automation rules are available and, can be created, from the project settings, namely from the "Automation" tab.

Project settings

Summary
Details
Re-index project
Delete project

Issue types
Acceptance Criterion
Bug
Capability
Epic
Improvement
Initiative
New Feature
Pre-Condition
Requirement
Resultset
▼ 13 more issue types

Workflows
Screens
Fields
Versions
Components
Users and roles
Permissions
Issue Security
Notifications
HyChat integration
Project automation

Automation

Global administration Filter rules Create rule

Name	Owner	Project	Enabled
Trigger Jenkins job	Administrator	Calculator (CALC)	✓
Trigger Jenkins job and link to Test Plan	Administrator	Calculator (CALC)	✓

All rules
Project rules
Global rules
Add label



Please note

The following examples are provided as-is, no warranties attached; use them carefully.

Please feel free to adapt them to your needs.

Note: We don't provide support for Automaton for Jira; if you have doubts concerning its usage, please contact [Automation for Jira's support](#).

Concepts

Jira Automation allows project administrators to implement rules that can make certain processes automated, guaranteeing efficiency and consistency.

The main concepts of Jira Automation follow a very simple approach for defining an automation **rule**: if a certain "thing" happens (**trigger**) and certain **conditions** are met, then execute one or more **actions**.

- **Trigger**: Triggers start the execution of a rule. Triggers can listen for events or be scheduled to run.
 - manual
 - upon field or workflow status changes
 - upon releasing
 - periodic
 - ...
- **Condition**: Actions will only execute if all conditions preceding them pass.
 - "If" statement
 - Issue fields condition
 - ...
- **Action**: Actions perform changes to a system.
 - change fields on issues
 - transition issues
 - web request
 - log
 - ...

It's also possible to run actions on issues that are **related** to the issue that triggered the rule, using **"branches"**.



Please note

Automation rules run asynchronously. Some actions can run in parallel but usually they're sequential. There is no interaction with the user (except if the trigger was set off manually from Jira's UI).

Usage Examples

Jenkins

Trigger a Jenkins project build from an issue

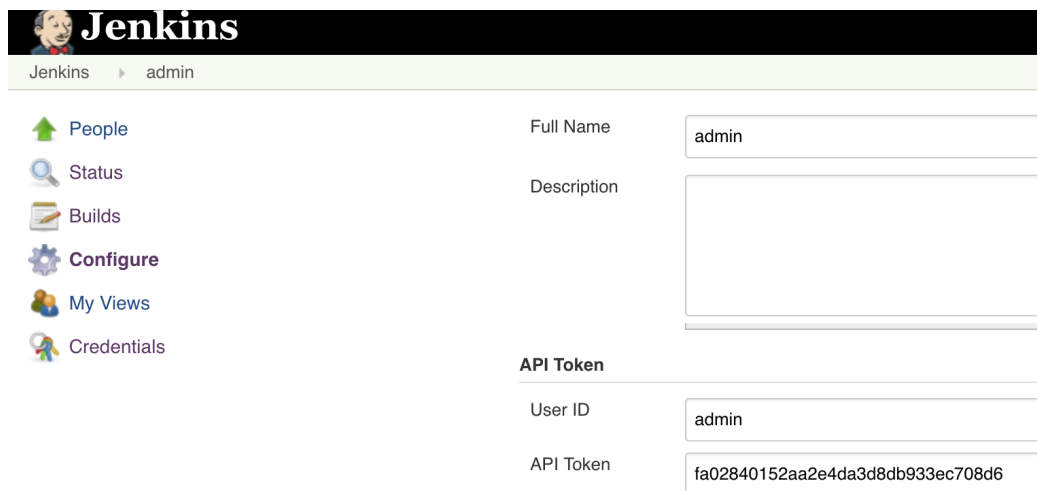
In this very simple scenario, we'll implement a rule, triggered manually, that will trigger a Jenkins project/job. The action will be available from within the "More" menu, in all issues of the selected project.

We're assuming that:

- you just want to trigger a CI job, period; this job may be totally unrelated to the issue from where you triggered it
- what the CI job will do, including if it will report the results back to Xray or not, is not relevant

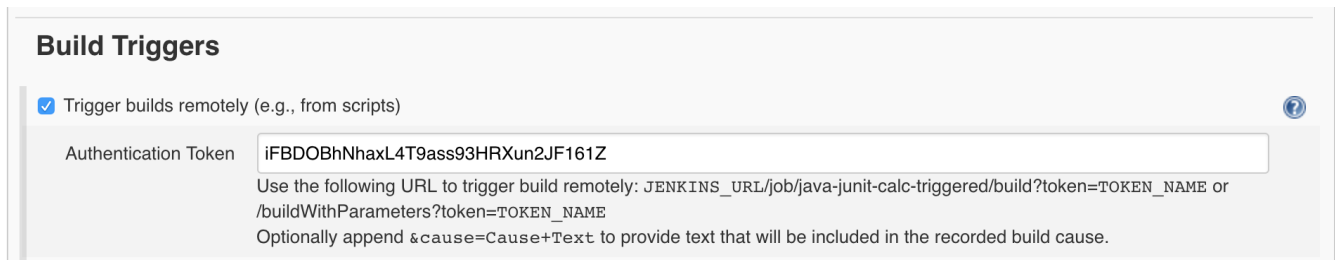
Jenkins configuration

In Jenkins, we need to generate an API token for some user, which can be done from the profile settings page.



The screenshot shows the Jenkins user profile settings for the 'admin' user. On the left is a sidebar with navigation links: People, Status, Builds, Configure, My Views, and Credentials. The main content area has a header 'Jenkins' with a user icon. Below it, the 'Full Name' is 'admin' and the 'Description' is empty. Under the 'API Token' section, the 'User ID' is 'admin' and the 'API Token' is 'fa02840152aa2e4da3d8db933ec708d6'.

At the project level, we need to enable remote build triggers, so we can obtain an "authentication token" to be used in the HTTP request afterwards.



The screenshot shows the 'Build Triggers' configuration page in Jenkins. The checkbox 'Trigger builds remotely (e.g., from scripts)' is checked. Below it, the 'Authentication Token' field contains the value 'iFBDOBhNhaxL4T9ass93HRXun2JF161Z'. A help icon is visible on the right. Below the token field, there is explanatory text: 'Use the following URL to trigger build remotely: JENKINS_URL/job/java-junit-calc-triggered/build?token=TOKEN_NAME or /buildWithParameters?token=TOKEN_NAME' and 'Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.'

The project itself is a normal one, without parameters.

Jenkins 3 Search

Jenkins » java-junit-calc-local-git »

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Description: creates a new Test Execution with the results from 4 junit tests. The revision field is populated with the build #

[Plain text] [Preview](#)

☒ Discard old builds

Strategy: Log Rotation

Days to keep builds:

If not empty, build records are only kept up to this number of days

Max # of builds to keep: 3

If not empty, only up to this number of build records are kept

[Advanced...](#)

☐ GitHub project

☐ This project is parameterized

☐ Throttle builds

☐ Disable this project

☐ Execute concurrent builds if necessary

[Advanced...](#)

Source Code Management

☐ None

☒ Git

Repositories

Repository URL: ssh://git@localhost/home/git/repos/automation-samples.git

Automation for Jira configuration

1. create a new rule and define the "When" (i.e. when it to should be triggered), to be "Manually triggered"

Automation DRAFT [Return to job](#)

Trigger Jenkins job

Rule details

Name: Trigger Jenkins job

Description: Trigger Jenkins job "java-junit-calc-local-git"

Projects: Calculator (CALC)

Projects can only be modified in the global administration.

Enabled: ☒

Allow rule trigger: ☒ Check to allow other rule actions to trigger this rule. Only enable this if you need this rule to execute in response to another rule.

Notify on error: Don't notify

Created: 3 hours ago

Owner: Administrator

The owner will receive emails when the rule fails.

Updated: 3 hours ago

Actor: Administrator

Actions defined in this rule will be performed by the user selected as the actor.

[Save](#) [Cancel](#)

When: Manually triggered

All logged in users can run rule.

Then: Send webhook

POST

http://192.168.56.102:8081/job/java-junit-calc-local-git/build?token=IFBDOBHnHaxL4T9ass93HRXun2JF161Z

[Add component](#)

2. define an action (i.e. the "Then") as "Send webhook" and configure it as follows

Automation ENABLED

Trigger Jenkins job

Rule details

Audit log

When: Manually triggered

All logged in users can run rule.

Then: Send webhook

POST

http://192.168.56.102:8081/job/java-junit-calc-local-git/build?token=IFBDOBHnHaxL4T9ass93HRXun2JF161Z

[Add component](#)

Send webhook

This action will send a HTTP POST to the url specified below:

Webhook URL:

Headers (optional)

Content-Type:

Authorization:

[Add](#)

HTTP method:

Webhook body:

[Save](#) [Cancel](#)

- the Webhook URL provided above follows this syntax:
 - <jenkins_base_url>/job/<name_of_jenkins_project_job>/build?token=<token>

- besides the "Content-Type" header that should be "application/json", define also an "Authorization" header having the value "Basic <auth>", where the base64 encoded <auth> can be [generated](#) using your Jenkins API credentials

After publishing the rule, you can go to the screen of an issue and trigger the Jenkins project/job.

The screenshot shows the JIRA interface for an issue titled "all my sum related tests for v3.0" (ID: CALC-3214). The issue is of type "Test Plan" with a "Major" priority. The "More" dropdown menu is open, displaying a list of actions. The "Trigger Jenkins job" option at the bottom of this menu is highlighted with a red rectangular box. Other visible options in the menu include "Trigger Bamboo Build w...", "Trigger Jenkins Build", "Trigger Jenkins build ...", "Synchronize Tests from...", "Assign", "Log work", "Agile Board", "Rank to Top", "Rank to Bottom", "Attach files", "Voters", "Stop watching", "Watchers", "Create sub-task", "Convert to sub-task", "Move", "Link", "Clone", "Labels", and "Delete". The "Overall Execution Status" is shown as "7 PASS" with a green progress bar.

Trigger a Jenkins project build from a Test Plan and report the results back to it


In this simple scenario, we'll implement a rule, triggered manually, that will trigger a Jenkins project/job. The action will be available from within the "More" menu, for all Test Plan issues of the selected project.

We're assuming that:


- you just want to trigger a CI job, period; this job may be totally unrelated to the issue from where you triggered it
- the results will be submitted back to Xray, if the project is configured to do so in Jenkins


Jenkins configuration


In Jenkins, we need to generate an API token for some user, which can be done from the profile settings page.


Jenkins


Jenkins > admin


People

Status

Builds

Configure

My Views

Credentials

Full Name

admin

Description

API Token

User ID

admin

API Token

fa02840152aa2e4da3d8db933ec708d6

At the project level, we need to enable remote build triggers, so we can obtain an "authentication token" to be used in the HTTP request afterwards.

Build Triggers

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

Authentication Token

iFBDOhNhaxL4T9ass93HRXun2JF161Z

Use the following URL to trigger build remotely: JENKINS_URL/job/java-junit-calc-triggered/build?token=TOKEN_NAME or /buildWithParameters?token=TOKEN_NAME
Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.

The project itself is a normal one; the only thing relevant to mention is that this project is a parameterized one, so it receives TESTPLAN, that in our case will be coming from Jira.

Jenkins

3

search

Jenkins > java-junit-calc-local-git-report-to-testplan >

GeneralSource Code ManagementBuild TriggersBuild EnvironmentBuild Post-build Actions

Description

creates a new Test Execution with the results from 4 junit tests. The revision field is populated with the build #

[Plain text] Preview

☒ Discard old builds

Strategy

Log Rotation

Days to keep builds

if not empty, build records are only kept up to this number of days

Max # of builds to keep

3

if not empty, only up to this number of build records are kept

Advanced...

☐ GitHub project

☒ This project is parameterized

String Parameter

Name

TESTPLAN

Default Value

Description

[Plain text] Preview

☒ Trim the string

Automation for Jira configuration

1. create a new rule and define the "When" (i.e. when it to should be triggered), to be "Manually triggered"

Automation

ENABLED

Trigger Jenkins job and link to Test Plan

Rule details

Audit log

When: Manually triggered

All logged in users can run rule.

If: Compare two values

Checks if:
{{issue.issue.type.name}} equals Test Plan

Then: Send webhook

POST
http://192.168.56.102:8081/job/java-junit-calc-local-git-report-to-testplan/buildWithParameters?token=iFBDOBhNhaxL4T9ass93HRXun2JF161Z&TESTPLAN={{issue.key}}

Add component

Rule details

Name* Trigger Jenkins job and link to Test Plan

Description Trigger Jenkins job "java-junit-calc"

Projects Calculator (CALC)

Projects can only be modified in the global administration.

Enabled ☒

Allow rule trigger ☐ Check to allow other rule actions to trigger this rule. Only enable this if you need this rule to execute in response to another rule.

Notify on error Don't notify

Created 3 hours ago

Owner Administrator

The owner will receive emails when the rule fails.

Updated 3 hours ago

Actor Administrator

Actions defined in this rule will be performed by the user selected as the actor.

Save Cancel

2. define the condition so that this rule can only be executed from Test Plan issue

Automation

ENABLED

Trigger Jenkins job and link to Test Plan

Rule details

Audit log

When: Manually triggered

All logged in users can run rule.

If: Compare two values

Checks if:
{{issue.issue.type.name}} equals Test Plan

Then: Send webhook

POST
http://192.168.56.102:8081/job/java-junit-calc-local-git-report-to-testplan/buildWithParameters?token=iFBDOBhNhaxL4T9ass93HRXun2JF161Z&TESTPLAN={{issue.key}}

Add component

Compare condition

Compares a value to another using value substitutions and regular expressions.

First value*

{{issue.issue.type.name}}

Condition

Equals

Second value

Test Plan

Save Cancel

What values can I compare?

3. define an action (i.e. the "Then") as "Send webhook" and configure it as follows

Automation

ENABLED

Trigger Jenkins job and link to Test Plan

① Rule details

② Audit log

When: Manually triggered

All logged in users can run rule.

If: Compare two values

Checks if:
{issue.issue.type.name} equals Test Plan

Then: Send webhook

POST
http://192.168.56.102:8081/job/java-junit-calc-local-git-report-to-testplan/buildWithParameters?
token=iFBDOBHnaxL4T9ass93HRXun2JF161Z&TESTPLAN={{issue.key}}

+ Add component

Send webhook

This action will send a HTTP POST to the url specified below:

Webhook URL*

s?token=iFBDOBHnaxL4T9ass93HRXun2JF161Z&TESTPLAN={{issue.key}}

Headers (optional)

Content-Type application/json

Authorization Basic YWRtaW46YWRtaW4=

Add

HTTP method

POST

Webhook body

Empty

Save Cancel

- the Webhook URL provided above follows this syntax:
 - <jenkins_base_url>/job/<name_of_jenkins_project_job>/buildWithParameters?token=<token>&TESTPLAN={{issue.key}}
- besides the "Content-Type" header that should be "application/json", define also an "Authorization" header having the value "Basic <auth>", where the base64 encoded <auth> can be [generated](#) using your Jenkins API credentials

After publishing the rule, you can go to the screen of an issue and trigger the Jenkins project/job.

Calculator / CALC-3214

all my sum related tests for v3.0

Edit

Comment

More

Stop Progress

Resolve Issue

Close Issue

Admin

Details

Type: Test Plan

Priority: Major

Affects Version/s: None

Component/s: None

Labels: None

Sprint: Sprint 1

Test Count: 7

Description

Risks/sensible areas to cover:

- addition operation in basic mode
- addition operation in scientific mode

Tests

Test Plan Board

Overall Execution Status

7 PASS

TOTAL TESTS: 7

Filter(s)

Trigger Bamboo Build w...

Trigger Jenkins Build

Trigger Jenkins build ...

Synchronize Tests from...

Assign

Log work

Agile Board

Rank to Top

Rank to Bottom

Attach files

Voters

Stop watching

Watchers

Create sub-task

Convert to sub-task

Move

Link

Clone

Labels

Delete

Trigger Jenkins job

Trigger Jenkins job an...

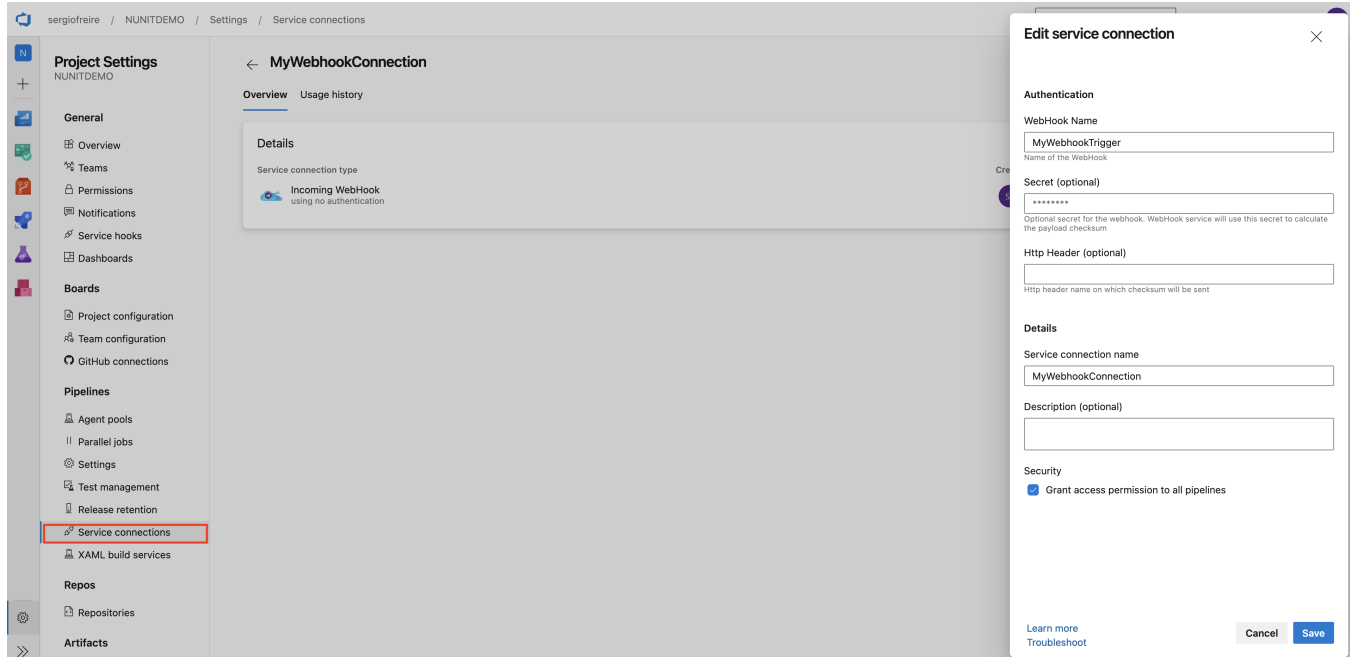
Trigger Jenkins job and link to Test Plan

Azure DevOps

Trigger a Azure DevOps pipeline from a Test Plan and report the results back to it

Azure DevOps configuration

We need to create a service connection, using the "incoming webhook" template, so that we can use Azure DevOps API later on.



Create a [Personal Access Token \(PAT\)](#), so you can use it as the password in API requests, along with the "organization name" as username.

Azure DevOps

Search

User settings
Sérgio Freire

Account

- Profile
- Time and Locale
- Permissions

Preferences

- Notifications
- Theme
- Usage

Security

- Personal access tokens**
- SSH public keys
- Alternate credentials
- Authorizations

Personal Access Tokens

These can be used instead of a password for applications like Git or can be passed in the authorization header to access REST APIs

+ New Token Revoke Edit Regenerate

Token name ↓	Status	Organization	Expires on
mytoken1 Full access	Active	sergiofreire	16/05/20
mytoken2 Build (Read & execute); Code (Full); Code (Read & write); Packaging (Read, w	Active	sergiofreire	05/05/20

Preview features

- Profile
- Time and Locale
- Permissions
- Notifications
- Theme
- Usage
- Personal access tokens
- SSH public keys
- Alternate credentials

Create a new personal access token

Name
mytoken

Organization
sergiofreire

Expiration (UTC)
30 days 24/06/2021

Scopes
Authorize the scope of access associated with this token
Scopes ☐ Full access ☒ Custom defined

Work items, queries, backlogs, plans, and metadata
☐ Read ☐ Read & write ☐ Read, write, & manage

Code
Source code, repositories, pull requests, and notifications
☐ Read ☐ Read & write ☐ Read, write, & manage ☐ Full ☐ Status

Build
Artifacts, definitions, requests, queue a build, and updated build properties
☐ Read ☒ Read & execute

Release
Read, update, and delete releases, release pipelines, and stages
☐ Read ☐ Read, write, & execute ☐ Read, write, execute, & manage

Test Management
Read, create, and updated test plans, cases, and results
☐ Read ☐ Read & write

Show all scopes (28 more)

Create Cancel

Then, in your Azure DevOps repository containing the project's code and tests, create a pipeline `/azure-pipelines.yml`; this pipeline will be triggered using Azure DevOps API.

In the following example, the pipeline will receive the Test Plan issue key as an input parameter. It will then run the build, including the automated tests, and in the end it will report the results back to Xray using "curl" utility.

We need to define a `resources` section, that contains a reference to the webhook configured earlier.

/azure-pipelines.yml

```
parameters:
- name: "testplan"
  type: string
  default: ""

trigger:
- main

resources:
  webhooks:
    - webhook: "MyWebhookTrigger"          ### Webhook alias
      connection: "MyWebhookConnection"    ### Incoming webhook service connection

pool:
  vmImage: ubuntu-latest

steps:
- bash: |
  echo ${ parameters.testplan }
  displayName: '(debug) print testplan parameter'

- script: dotnet restore
  displayName: 'install build dependencies'

- script: |
  dotnet test -s nunit.runsettings
  displayName: 'Run tests'
- bash: |
  set -x
  curl -o - -H "Content-Type: multipart/form-data" -u '${jira_user}:${jira_password}' -F "file=@./bin/Debug
/net5.0/TestResults/nunit_webdriver_tests.xml" "${jira_server_url}/rest/raven/2.0/import/execution/nunit?
projectKey=${project_key}&testPlanKey=${TESTPLAN}"
  displayName: 'Import results to Xray server'
```

Xray endpoint's base URL and the API key credentials (i.e. client id + client secret) are defined in Azure DevOps as variables. These may be marked as secret.

The screenshot displays the Azure DevOps web interface. On the left, a sidebar contains navigation links for Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The 'Pipelines' link is selected. The main area shows the configuration for a pipeline named 'NUNITDEMO'. At the top right of this area are 'Variables' and 'Run' buttons. Below the pipeline name, a dropdown menu shows 'main' and a link to 'NUNITDEMO / azure-pipelines.yml'. The pipeline definition is shown in a code editor with line numbers 1 through 22. The configuration matches the YAML code provided in the first block. On the right side of the interface, there is a 'Tasks' panel with a search bar and a list of tasks including .NET Core, Android signing, Ant, App Center distribute, App Center test, Archive files, and ARM template deployment.

= CALC

2. define the condition so that this rule can only be executed from Test Plan issue

Automation

ENABLED

trigger Azure DevOps pipeline for this Test Plan

Rule details

Audit log

When: Manually triggered
All logged in users can run rule.

If: Issue Type equals
Test Plan

Issue fields condition

Check whether an issue's field meets a certain criteria

Field *

Issue Type

Condition *

equals

Value Field

Test Plan

3. define an action (i.e. the "Then") as "Send web request" and configure it as follows

Automation

ENABLED

trigger Azure DevOps pipeline for this Test Plan

Rule details

Audit log

When: Manually triggered
All logged in users can run rule.

If: Issue Type equals
Test Plan

Then: Send web request
POST
https://dev.azure.com/sergiofreire/NUNIT
DEMO/_apis/build/builds?
ignoreWarnings=true&api-version=6.0

Add component

Send web request

This action will send a HTTP request to the url specified below:

Webhook URL *

https://dev.azure.com/sergiofreire/NUNITDEMO/_apis/build/builds?ignoreWarni

Request parameters must be url encoded, smart values should use: {{value.urlEncode}}.

Headers (optional)

Content-Type

application/json

Authorization

Basic c2VyZ2lvLmZyZWlyZUB4c

Add

HTTP method

POST

Webhook body

Custom data

Wait for response

☐ Delay execution of subsequent rule actions until we've received a response for this webhook

Custom data *

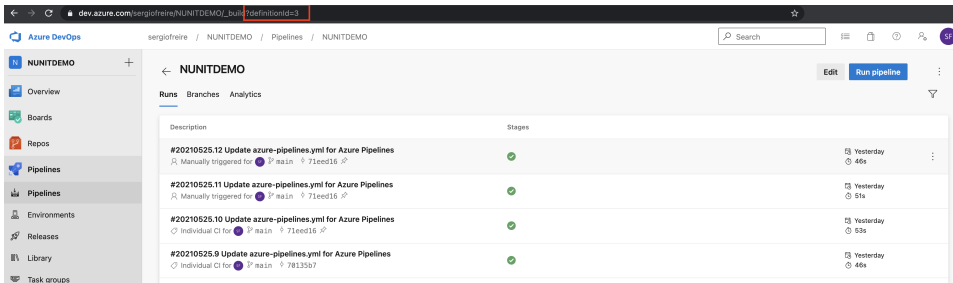
```
{
  "parameters": "{ \"testplan\": \"{{issue.key}}\" }",
  "definition": {
    "id": 3
  }
}
```

- the web request URL provided above is from [Azure DevOps API](#), for [queueing builds](#), and follows this syntax:
 - https://dev.azure.com/<organization_name>/<project>/_apis/build/builds?ignoreWarnings=true&api-version=6.0
- authentication is done using the organization name plus the personal access token, created earlier in Azure DevOps, as the login:password pair used to calculate the Base64 content of the Authorization header
- the "Content-Type" header should be "application/json"
- the HTTP POST body content, defined in the "Custom data" field, will be used to identify the [build definition](#) and also the original Test Plan issue key;

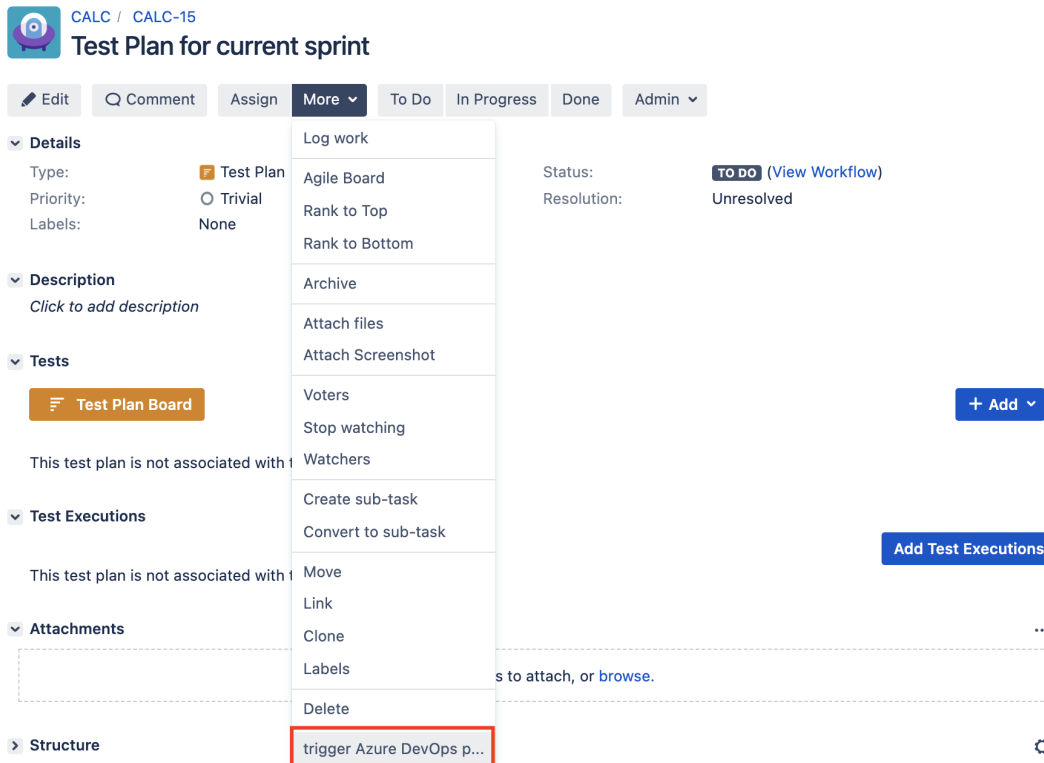
custom data (i.e. HTTP body content)

```
{
  "parameters": "{ \"testplan\": \"{{issue.key}}\" }",
  "definition": {
    "id": 3
  }
}
```

Note: to find the *definition id*, you can click on the pipeline in Azure DevOps and its id is shown as part of the URL



After publishing the rule, you can go to the screen of an issue and trigger a pipeline run in Azure DevOps.



Azure DevOps interface showing a pipeline run for 'NUNITDEMO'. The left sidebar lists various components like Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The main area displays 'Jobs in run #20210525...' with a list of jobs including 'Initialize job', 'Checkout NUNITDEMO...', '(debug) print testplan ...', 'install build depende...', 'Run tests', '(debug) list files in w...', 'Import results to Xra...', 'Import results to Xray ...', 'Post-job: Checkout ...', 'Finalize Job', and 'Report build status'. The 'Run tests' job is expanded, showing a detailed log of the test execution process, including script generation, project restoration, and test results.

In this case, since the pipeline was configured to report results back to Xray, a new Test Execution would be created and linked back to the source Test Plan where the automation was triggered from.

Calc / CALC-18

Execution results - nunit_webdriver_tests.xml - [1622026982605]

Edit Comment Assign More To Do In Progress Done Admin

Details

Type: Test Execution
Priority: Trivial
Labels: None
Test Plan: **CALC-15**
Test Environments: None

Status: **TO DO** (View Workflow)
Resolution: Unresolved

Description

Execution results imported from external source

Tests

+ Add

Overall Execution Status

4 PASS

Total Tests: 4

Filter(s)

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
1	CALC-13	BasicTextSearchNoPOM	Generic	0	0	Sergio Freire	PASS
2	CALC-11	BasicTextSearch	Generic	0	0	Sergio Freire	PASS
3	CALC-12	InvalidLogin	Generic	1	0	Sergio Freire	PASS
4	CALC-10	ValidLogin	Generic	1	0	Sergio Freire	PASS

Showing 1 to 4 of 4 entries

First Previous 1 Next Last

CALC / CALC-15

Test Plan for current sprint

Edit

Comment

Assign

More

To Do

In Progress

Done

Admin

Details

Type: Test Plan

Priority: Trivial

Labels: None

Status: **TO DO** (View Workflow)

Resolution: Unresolved

Description

Click to add description

Tests

Test Plan Board

Create Test Execution

Add

Overall Execution Status

4 PASS

Total Tests: 4

Filter(s)

Show 10 entries

All Environments

Columns

Key	Summary	Requirements	#Test Executions	Issue Assignee	Latest Status	
<input type="checkbox"/> CALC-13	BasicTextSearchNoPOM		1	Xpand IT Admin	PASS	...
<input type="checkbox"/> CALC-11	BasicTextSearch		1	Xpand IT Admin	PASS	...
<input type="checkbox"/> CALC-12	InvalidLogin	CALC-2	1	Xpand IT Admin	PASS	...
<input type="checkbox"/> CALC-10	ValidLogin	CALC-2	1	Xpand IT Admin	PASS	...

Showing 1 to 4 of 4 entries

First Previous 1 Next Last

Test Executions

Add Test Executions

Show 10 entries

Columns

Key	Summary	#Tests	Issue Assignee	Status	
<input type="checkbox"/> CALC-18	Execution results - junit_webdriver_tests.xml - [1622026982605]	4	Sergio Freire		...

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Travis CI

Trigger a TravisCI project build from a Test Plan and report the results back to it

In this simple scenario, we'll implement a rule, triggered manually, that will trigger a TravisCI project/job. The action will be available from the "Automation" panel, for all Test Plan issues of the selected project.

We're assuming that:

- you just want to trigger a CI job, period; this job may be totally unrelated to the issue from where you triggered it
- the results will be submitted back to Xray, if the project is configured to do so in TravisCI

TravisCI configuration

In TravisCI, we need to generate an API authentication token for some user, which can be done from the My Account settings page.

MY ACCOUNT



Cristiano Morais da Cunha

Sync account

ORGANIZATIONS



Xray App

MISSING AN ORGANIZATION?

Review and add your authorized organizations.



Cristiano Morais da Cunha

@CMCunha

Repositories
Settings
Plan
Migrate
Plan usage

API authentication

To learn more about using our API, please head to developer.travis-ci.com.

Token

COPY TOKEN

VIEW TOKEN

Once we have the authentication token we followed the [TravisCI API documentation](#) to configure the following steps on the Jira side.

For the Travis CI the important change we must do is in the YAML file that will configure Travis CI pipeline, we use the following configuration to achieve that:

.travis.yml

```

sudo: false
language: java
jdk:
  - openjdk8
cache:
  directories:
    - "$HOME/.cache"

jobs:
  include:
    - stage: test and report to Xray
      script:
        - |
          echo "building repo..."
          mvn clean compile test --file pom.xml
          curl -H "Content-Type: multipart/form-data" -X POST -u $USERNAME:$PASSWORD -F "file=@target/surefire-reports/TEST-com.xpand.java.CalcTest.xml" "https://$JIRASERVER/rest/raven/2.0/import/execution/junit?projectKey=$PROJECTKEY&testPlanKey=$TESTPLAN"
          echo "done"

```

For more details about this configuration please check the [TravisCI tutorial documentation](#).

As you can see we are pushing results back to Xray with the last curl command:

curl command

```

curl -H "Content-Type: multipart/form-data" -X POST -u $USERNAME:$PASSWORD -F "file=@target/surefire-reports/TEST-com.xpand.java.CalcTest.xml" "https://$JIRASERVER/rest/raven/2.0/import/execution/junit?projectKey=$PROJECTKEY&testPlanKey=$TESTPLAN"

```

On this command we are passing the project key in order to report back to a specific Project on the Xray side. Further ahead we will show how it is populated.

- **PROJECTKEY** - The key that identifies the project on the Jira side.
- **TESTPLAN** - The Test Plan key used to identify the Test Plan to associate the execution with.

Once we have the authentication token, we follow the [TravisCI API documentation](#) to configure the following steps on the Jira side.

Automation configuration

On the Jira side we will use the Automation capabilities that it provides out of the box, so within the administration area go to the automation entry in the system settings and:

1. create a new rule and define the "When" (i.e. when it to should be triggered), to be "Manually triggered"

The screenshot shows the Jira Automation configuration interface. On the left, under 'Automation', there is a list of steps: 'Rule details', 'Audit log', and 'When: Manually triggered'. The 'When: Manually triggered' step is selected and expanded. The description for this step is 'All logged in users can run rule.' On the right, the configuration for the 'Manual trigger' is shown. It includes a description: 'Rule is run when it is manually triggered by the user from an issue.' and a dropdown menu for 'Groups that can run trigger' which is currently set to 'All logged in users'. There are 'Cancel' and 'Save' buttons at the bottom right.

2. Define a condition, in our case we will define that only Test Plan issue types will be allowed to trigger this pipeline, this is achieved with the following condition:

The screenshot shows the Jira Automation configuration interface. On the left, under 'Automation', there is a list of steps: 'Rule details', 'Audit log', 'When: Manually triggered', and 'If: Issue Type equals'. The 'If: Issue Type equals' step is selected and expanded. The description for this step is 'Test Plan'. On the right, the configuration for the 'Issue fields condition' is shown. It includes a description: 'Check whether an issue's field meets a certain criteria'. There are two dropdown menus: 'Field' which is set to 'Issue Type', and 'Condition' which is set to 'equals'. Below these, there is a 'Value' field which is set to 'Test Plan'. There are 'Cancel' and 'Save' buttons at the bottom right.

3. define an action (i.e. the "Then") as "Send webhook" and configure it as follows

Automation

DRAFT

Trigger Travis CI

Rule details

Audit log

When: Manually triggered
All logged in users can run rule.

If: Issue Type equals
Test Plan

Then: Send web request
POST https://api.travis-ci.com/repo/Xray-App%2Ftutorial-java-junit-travisci/requests

+ Add component

Send web request

This action will send a HTTP request to the url specified below:

Webhook URL

Request parameters must be url encoded, smart values should use: {{value.urlEncode}}

Headers (optional)

Content-Type	<input type="text" value="application/json"/>	
Accept	<input type="text" value="application/json"/>	
Authorization	<input type="text" value="token"/>	
Travis-API-Version	<input type="text" value="3"/>	

Add

HTTP method

Webhook body

Wait for response

☐ Delay execution of subsequent rule actions until we've received a response for this webhook

Custom data

```
{
  "request": {
    "branch": "main",
    "config": {
      "env": {
        "TESTPLAN": "{{issue.key}}",
        "PROJECTKEY": "{{project.key}}"
      }
    }
  }
}
```

[Cancel](#) [Save](#)

> Validate your webhook configuration

- the Webhook URL provided above follows this syntax:
 - <TravisCI_API_URL>/repo/{slugId}/requests (The %2F in the request URL is required so that the owner and repository name in the repository slug are interpreted as a single URL segment.)
- besides the "Content-Type" header that should be "application/json", define also an "Authorization" header having the value "token <token>", where you will place the authentication token obtained previously in the TravisCI page and the "Travis-API-Version" header is also mandatory and it will contain the version used.
- Custom data
 - We included the simplest possible just to trigger the pipeline from the master branch.
 - Added environment configuration variables to be used later in the TravisCI pipeline
 - TESTPLAN - that will be automatically filled with the test plan key from where the pipeline is triggered.
 - PROJECTKEY - that will be automatically filled in with the project key.

After publishing the rule, you can go to the screen of an issue and trigger the TravisCI project/job.



ComicStore / COM-9

(DEMO) Shopping cart Management test plan

Edit

Comment

Assign

More

To Do

In Progress

Done

Admin

Details

Type: **Test Plan**

Priority: **Trivial**

Component/s: **None**

Labels: **None**

Description

Click to add description

Tests

Test Plan Board

Overall Execution Status

4 PASS

Total Tests: 4

Filter(s)

Log work

Agile Board

Rank to Top

Rank to Bottom

Attach files

Voters

Stop watching

Watchers

Create sub-task

Convert to sub-task

Move

Link

Clone

Labels

Delete

Trigger Travis CI

Status: **TO DO** (View Workflow)

Resolution: **Unresolved**

+ Create Test Execution

+ Add

Show 10 entries

All Environments

Columns

In this case, since TravicCI was configured to report results back to Xray, a new Test Execution would be created in Jira/Xray.



ComicStore / COM-16

Execution results [1622475015055]

Edit

Comment

Assign

More

To Do

In Progress

Done

Admin

Details

Type: **Test Execution**

Priority: **Trivial**

Component/s: **None**

Labels: **None**

Test Plan: **COM-9**

Test Environments: **None**

Description

Execution results imported from external source

Tests

+ Add

Overall Execution Status

4 PASS

Total Tests: 4

Filter(s)

Apply Rank

Show 100 entries

Columns

	Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status		
<input type="checkbox"/>	3	CALC-1205	CanSubtract	Generic	0	0	Helder Biscaia	PASS		...
<input type="checkbox"/>	4	CALC-1204	CanMultiply	Generic	0	0	Helder Biscaia	PASS		...
<input type="checkbox"/>	1	CALC-1203	CanDoStuff	Generic	0	0	Helder Biscaia	PASS		...
<input type="checkbox"/>	2	CALC-1202	CanAddNumbers	Generic	0	0	Helder Biscaia	PASS		...

Showing 1 to 4 of 4 entries

First

Previous

1

Next

Last

Associated with the Test Plan that we have passed along:

ComicStore / COM-16
Execution results [1622475015055]

Edit Comment Assign More To Do In Progress Done Admin

Details
Type: Test Execution
Priority: Trivial
Component/s: None
Labels: None
Test Plan: COM-9
Test Environments: None
Status: TO DO (View Workflow)
Resolution: Unresolved

Description
Execution results imported from external source

Tests + Add

Overall Execution Status

4 PASS

Total Tests: 4
Filter(s)

Apply Rank Show 100 entries Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
3	CALC-1205	CanSubtract	Generic	0	0	Helder Biscaia	PASS
4	CALC-1204	CanMultiply	Generic	0	0	Helder Biscaia	PASS
1	CALC-1203	CanDoStuff	Generic	0	0	Helder Biscaia	PASS
2	CALC-1202	CanAddNumbers	Generic	0	0	Helder Biscaia	PASS

Showing 1 to 4 of 4 entries First Previous 1 Next Last

Generic automation of processes

Copy fields from requirement/Story to Test whenever creating a Test or linking it to a story

Sometimes it may be useful to copy some fields from the requirement/Story to the Tests that cover it.

Automation configuration

On the Jira side we will use the Automation capabilities that it provides out of the box, so within the administration area go to the automation entry in the system settings and:

1. create a new rule and define the "When" (i.e. when it should be triggered) to be "Issue linked". Since Xray, by default, uses the issue link type "Tests" to establish the coverage relation between a Test and the requirement, we can take advantage of that to trigger the rule whenever such issue link is created.

CALC Summary Details Audit log Re-index project Delete project Issue types Bug Epic Improvement New Feature Pre-Condition Story Sub-task Sub-Test Execution Task 4 more issue types Workflows Screens Fields Priorities Versions Components Users and roles Permissions Issue Security Notifications Project links Project automation

Automation
Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)

Copy fields from requirement to Tests ENABLED

Rule details Audit log

When: Issue linked
Types: Tests

If: Issue Type equals Test

Then: Edit Issue fields
Urgency, Priority

Add component

Issue linked
Rule executes when an issue is linked to another issue. (IssueA) will always refer to the source issue, so if ISSUE-A is blocked by ISSUE-B, this rule will execute on ISSUE-B. To access ISSUE-A, use {{destinationIssue}}, and to access the link type, use {{linkType}} (e.g. {{linkType}}="Duplicate").

Link types
Tests X

More options

Cancel Save

a.

2. create a condition to ensure that the rule only runs for Test issues

Automation

Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)

Copy fields from requirement to Tests

ENABLED

Rule details

Audit log

When: Issue linked
Types: Tests

If: Issue Type equals
Test

Issue fields condition

Check whether an issue's field meets a certain criteria

Field *

Issue Type

Condition *

equals

Value

Field

Test

a.

3. use an "Edit issue" action to set the fields on the Test based on the fields of the linked requirement/Story (i.e., the *destination* issue of the linking event). In this example, we'll copy the values of Urgency and Probability custom fields.

Automation

Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)

Copy fields from requirement to Tests

ENABLED

Rule details

Audit log

When: Issue linked
Types: Tests

If: Issue Type equals
Test

Then: Edit issue fields
Urgency, Probability

a.

Edit issue

Set values for fields on the issue. Simply add the fields you want to edit.

Choose fields to set...

Urgency



Copy Urgency from Destination issue



Probability



Copy Probability from Destination issue



More options

Cancel

Save

This rule will run:

- whenever a Test is created from the requirement/Story issue screen
- whenever a Test has been initially created and later on linked to the requirement

Reopen/transition Tests linked to a requirement whenever the requirement is transitioned or changed

Whenever you change the specification of a requirement/story, you most probably will need to review the Tests that you have already specified.

The following rule tries to perform a transition of **all** Tests linked to a requirement.

Automation configuration

On the Jira side we will use the Automation capabilities that it provides out of the box, so within the administration area go to the automation entry in the system settings and:

1. create a new rule and define the "When" (i.e. when it should be triggered) to be "Field value changed"

Project settings

Summary
Details
Audit log
Re-index project
Delete project

Issue types
Bug
Epic
Improvement
New Feature
Pre-Condition
Story
Sub-task
Sub-Test Execution
Task
Test
4 more issue types

Workflows
Screens
Fields
Priorities

Versions
Components

Users and roles
Permissions
Issue Security
Notifications
Project links

Project automation

Automation
Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)

reopen Tests if requirement changes **ENABLED**

Rule details
Audit log

When: Value changes for Description

If: Issue Type is one of
Story, Epic

For JQL
Issue in requirementTests(((issue.key)))

Then: Transition the issue to
TO DO

Add component

Field value changed

This rule will trigger when the value of the fields selected below changes:
Fields to monitor for changes*

Description (Field)

You can also match field names using regular expressions.

For
All issue operations

Optionally select which operations this trigger will execute for. Leave blank for all operations.

More options

How do I access the changed value in my rule?

Cancel Save

- a.
- b. Note: we could also define the trigger to be based on the transition of the requirement issue to a certain workflow status; in that case we would define it, for example, as shown below.

Automation
Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)

reopen Tests if requirement changes **DRAFT**

Rule details
Audit log

When: Issue transitioned

TO
To Do

Issue transitioned

Rule is run when an issue is transitioned through its workflow.

From status
Leave blank to match all statuses...

To status
To Do

More options

- i.
2. create a condition to ensure that the rule only runs for Story and Epic issues; adjust these to include all the "requirement" issue types (i.e., the ones that you can cover with Tests)

Automation
Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)

reopen Tests if requirement changes **DRAFT**

Rule details
Audit log

When: Issue transitioned

TO
To Do

If: Issue Type is one of
Story, Epic

Issue fields condition

Check whether an issue's field meets a certain criteria

Field*
Issue Type

Condition*
is one of

Value Field

Story x Epic x

Cancel

- a.
3. create a "branch rule / related issues" to obtain related Tests using JQL and the `requirementTests()` JQL function and run one, or more, action(s) on them

Automation

Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)

reopen Tests if requirement changes ENABLED

Rule details
Audit log

When: Value changes for Description

If: Issue Type is one of Story, Epic

For JQL
issue in requirementTests({{issue.key}})

Then: Transition the issue to **TO DO**

Branch rule / related issues

Please select which related issue you would like to perform actions against.

Type of related issues*
JQL

JQL*
issue in requirementTests({{issue.key}}) Validate query

☐ Only include issues that have changed since the last time this rule executed

May cause performance issues

Rule restricted to projects
CALC (CALC)

Only issues from the above projects will be considered. You can change this restriction in the 'Rule details' section.

[More options](#)

[Cancel](#) [Save](#)

[What issue data can I access in conditions and actions for related issues?](#)

a.

- under the "For JQL" block, create a action "Transition the issue to" in order to reopen the related Test issues

Automation

Automation rules allow you to automate repetitive tasks based on criteria that you set. Here you can manage existing rules and create new ones. [Learn more about automation](#)

reopen Tests if requirement changes ENABLED

Rule details
Audit log

When: Value changes for Description

If: Issue Type is one of Story, Epic

For JQL
issue in requirementTests({{issue.key}})

Then: Transition the issue to
TO DO

Transition issue

Transition the issue by:
☒ Selecting the destination status
☐ Selecting a specific transition

Destination status*
To Do

Ensure a transition from the issue's source status to your selected destination status exists; [more info](#).

[+ add regex to distinguish between multiple transitions to the same status](#)

Choose fields to set...

[More options](#)

[Cancel](#)

a.

References

- [Automation for Jira in the Atlassian Marketplace](#)
- [Jira Automation in Jira DC](#)
 - [Jira smart values - issues](#)
 - [Jira smart values - lists](#)
 - [Jira smart values - text fields](#)
 - [Jira smart values - users](#)
 - [Jira smart values - conditional logic](#)
 - [Jira smart values - JSON functions](#)
 - [Branch automation rules to perform actions on related issues](#)