Integration with Automation for Jira

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).

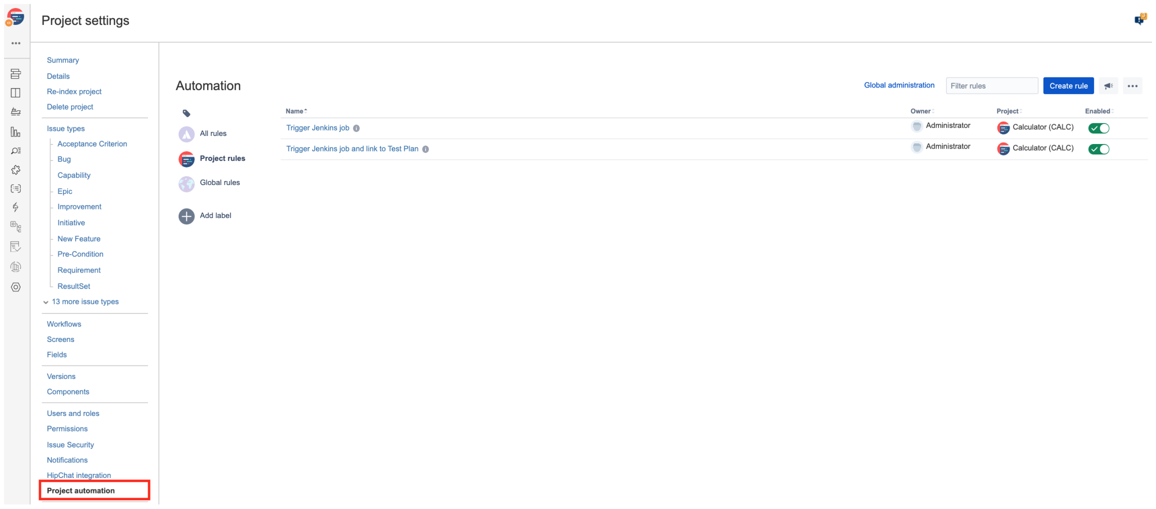
Rules can also be triggered manually or may be scheduled.

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.



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

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 **cond itions** 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

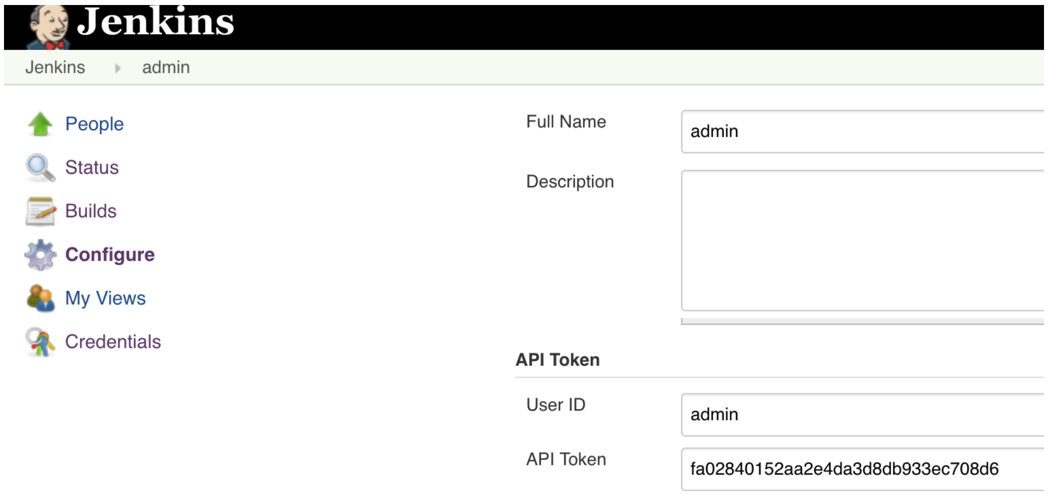
In this very simple scenario, we'll implement a rule, triggered manually, that will trigger a Jenkins project/job. The action will be available from within the "More" menu, in all issues of the selected project.

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.



Full Name: admin

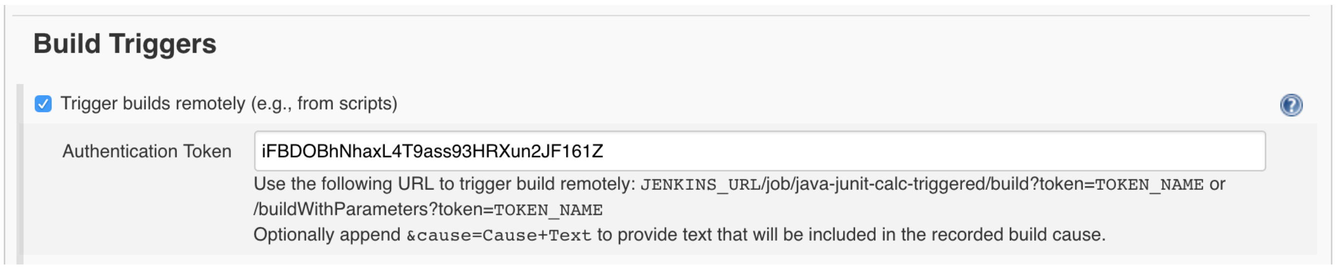
Description:

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.



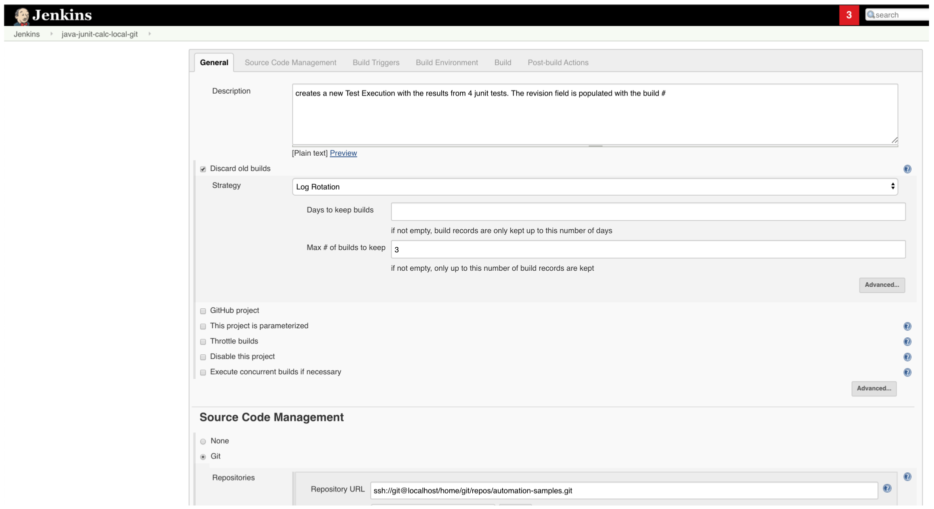
Build Triggers

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

Authentication Token: iFBDOBhNhaxL4T9ass93HRXun2JF161Z

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, without parameters.


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

☐ GitHub project
☐ This project is parameterized
☐ Throttle builds
☐ Disable this project
☐ Execute concurrent builds if necessary

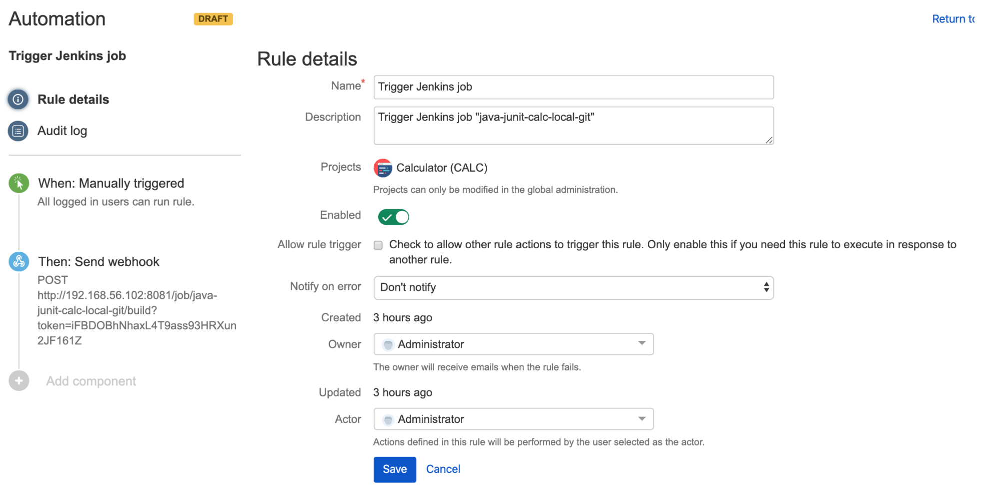
Source Code Management

○ 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

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

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

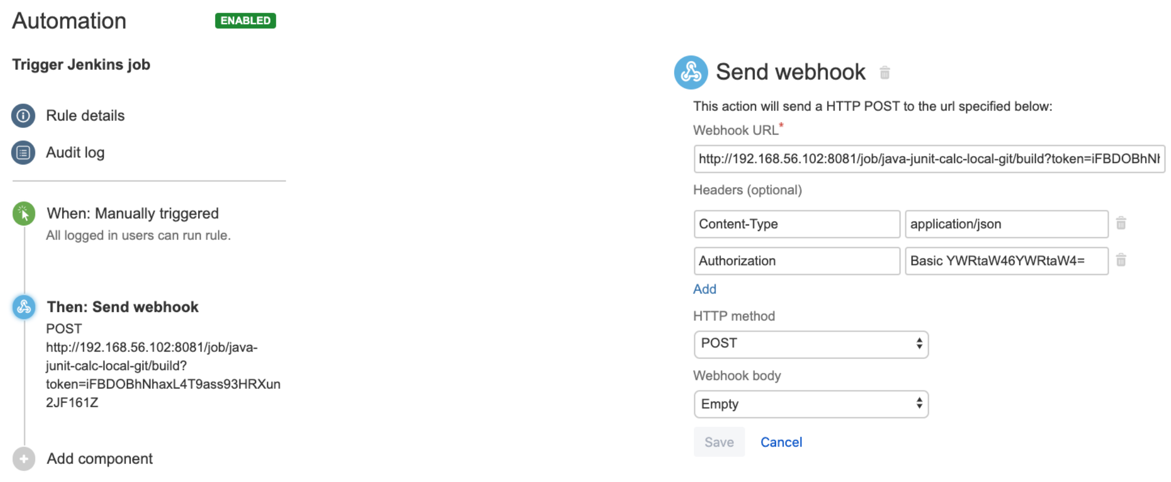
Owner: Administrator
The owner will receive emails when the rule fails.

Updated: 3 hours ago

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

Save | Cancel

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

Headers (optional)

Content-Type	application/json
Authorization	Basic YWRtaW46YWRtaW4=

Add

HTTP method: POST

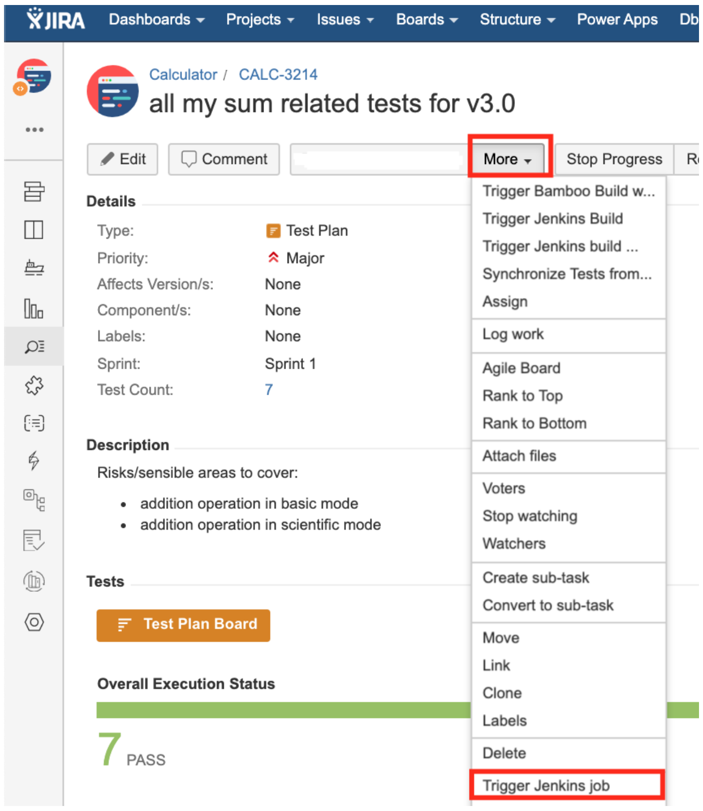
Webhook body: Empty

Save | Cancel

- the Webhook URL provided above follows this syntax:
  - <jenkins_base_url>/job/<name_of_jenkins_project_job>/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.



Calculator / CALC-3214

all my sum related tests for v3.0

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

Overall Execution Status

7 PASS

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

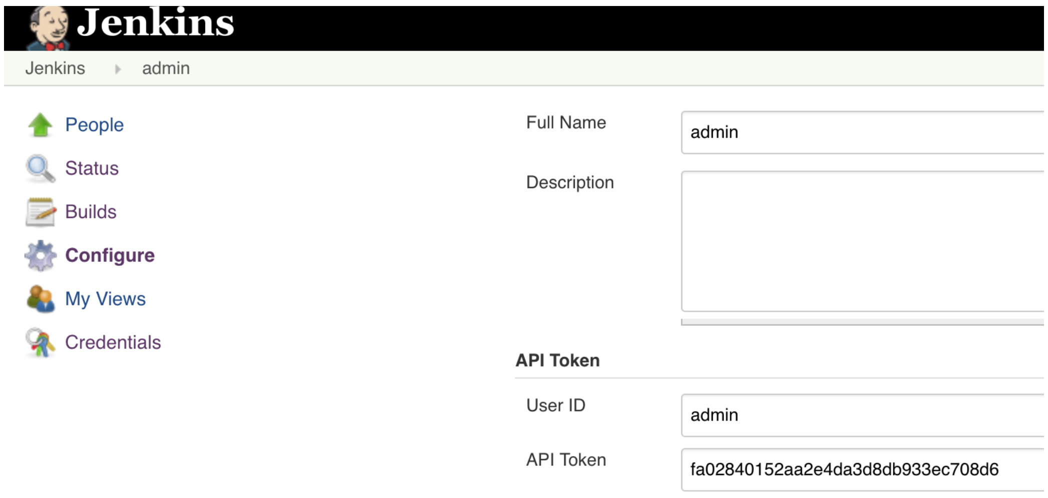
In this simple scenario, we'll implement a rule, triggered manually, that will trigger a Jenkins project/job. The action will be available from within the "More" menu, for all Test Plan issues of the selected project.

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.



Full Name: admin

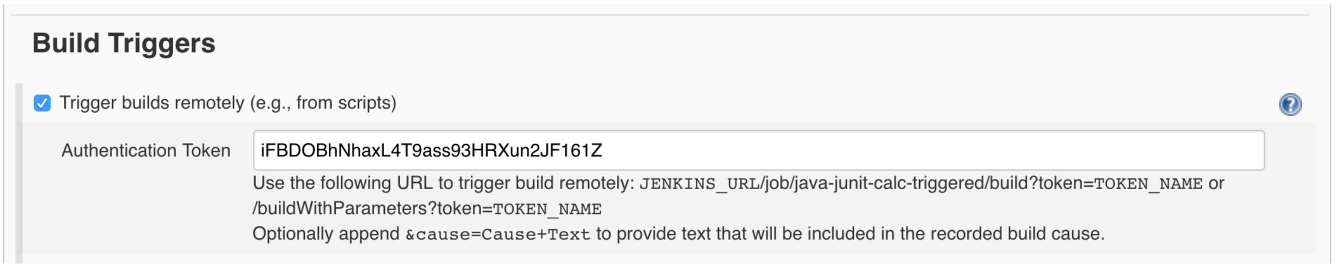
Description:

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.



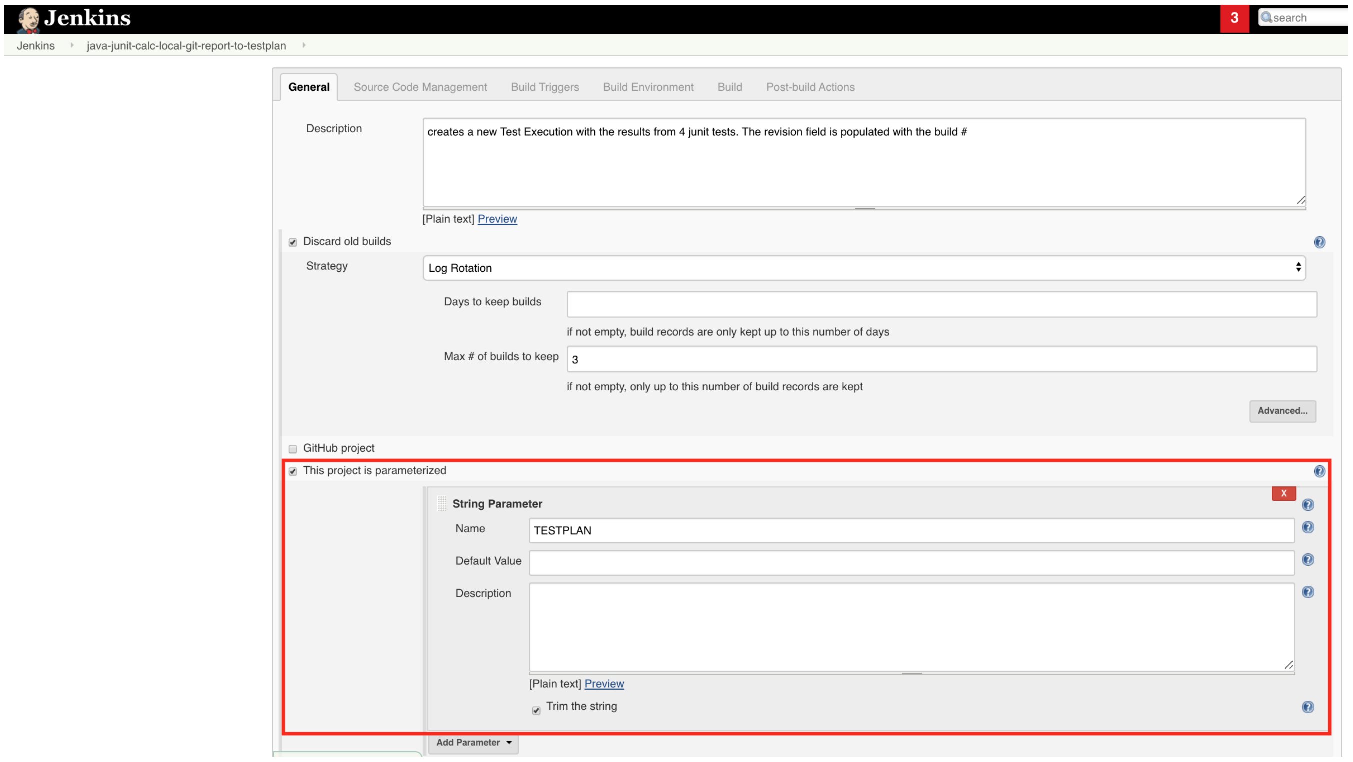
Build Triggers

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

Authentication Token: iFBDOBhNhaxL4T9ass93HRXun2JF161Z

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.



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

☐ GitHub project

✓ This project is parameterized

String Parameter

Name: TESTPLAN

Default Value:

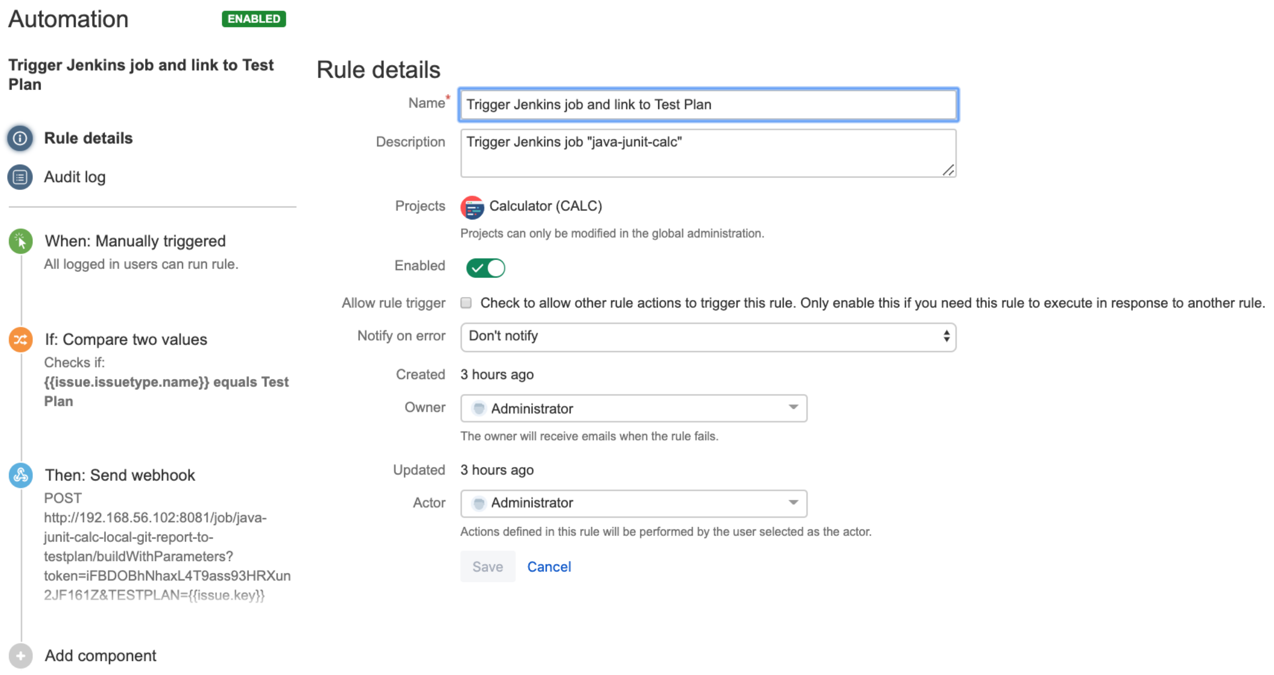
Description:

[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.issuetype.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

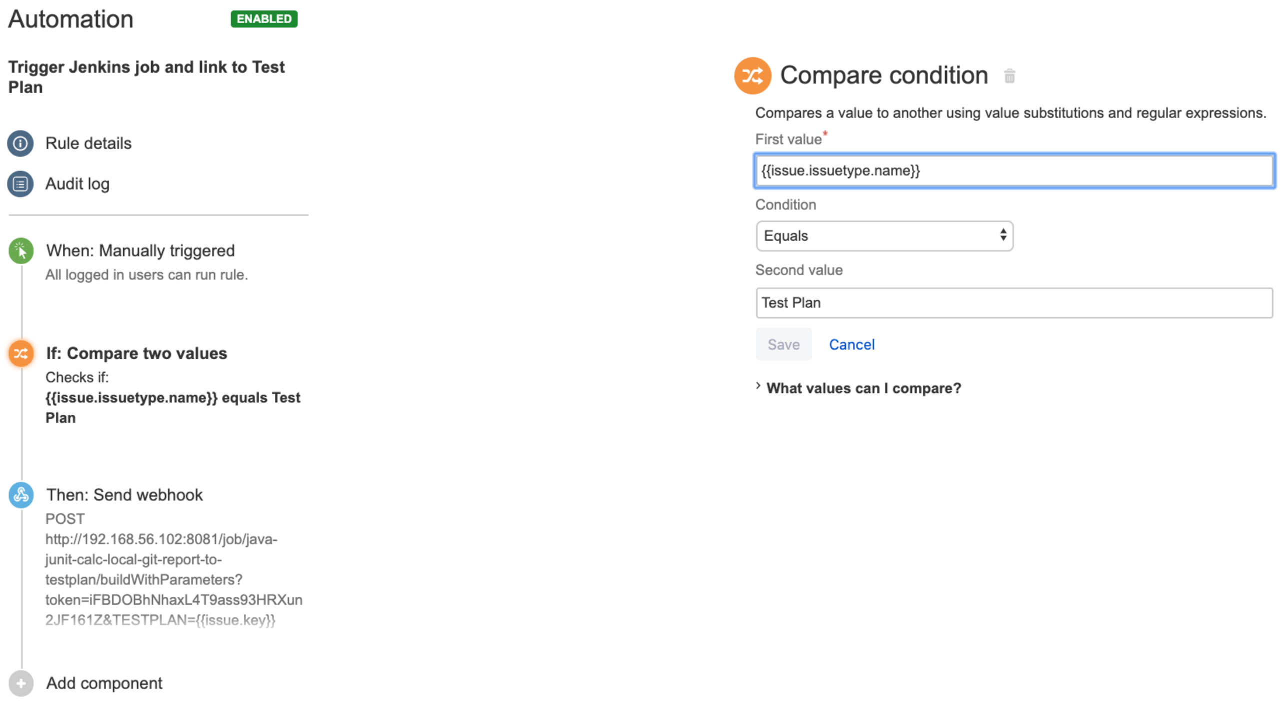
Owner Administrator
The owner will receive emails when the rule fails.

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.issuetype.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.issuetype.name}}

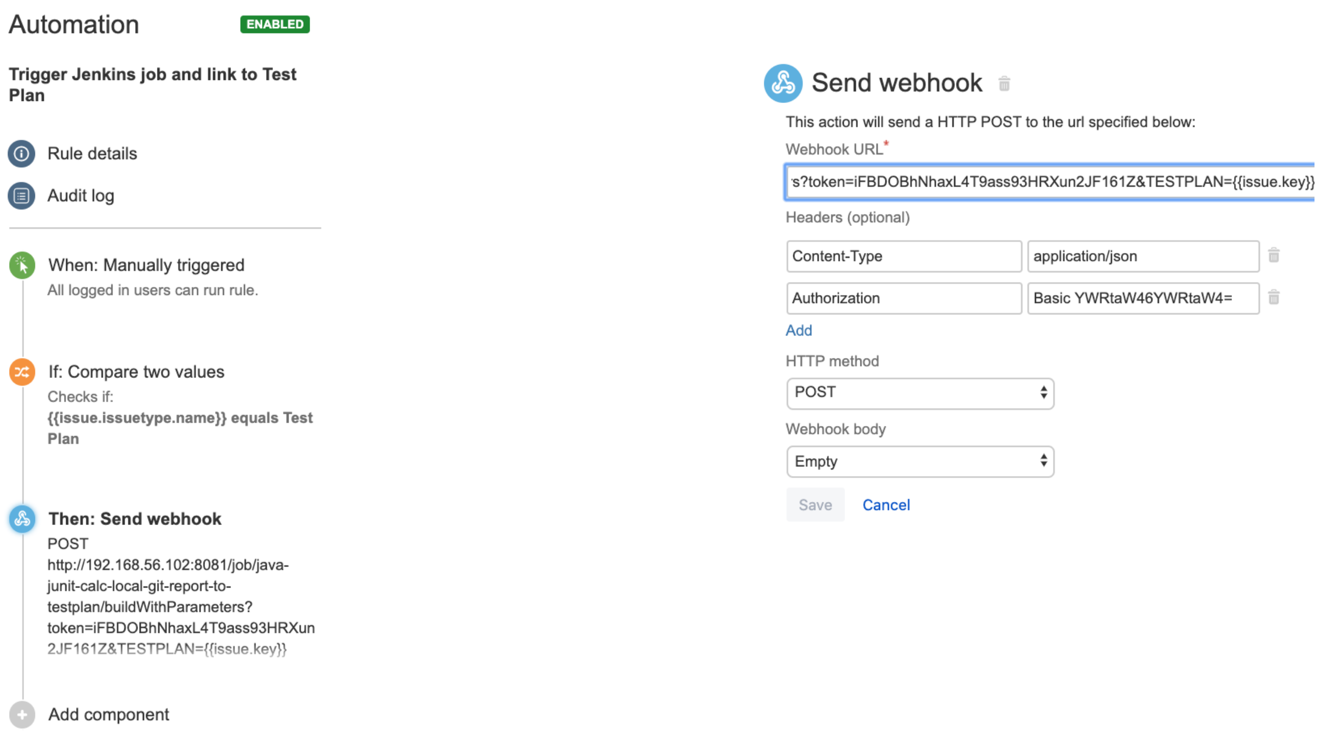
Condition Equals

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

Rule details

Audit log

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

If: Compare two values
Checks if:
{{issue.issuetype.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

Send webhook

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

Webhook URL*
s?token=iFBDOBhNhaxL4T9ass93HRXun2JF161Z&TESTPLAN={{issue.key}}

Headers (optional)

Content-Type	application/json
Authorization	Basic YWRtaW46YWRtaW4=

Add

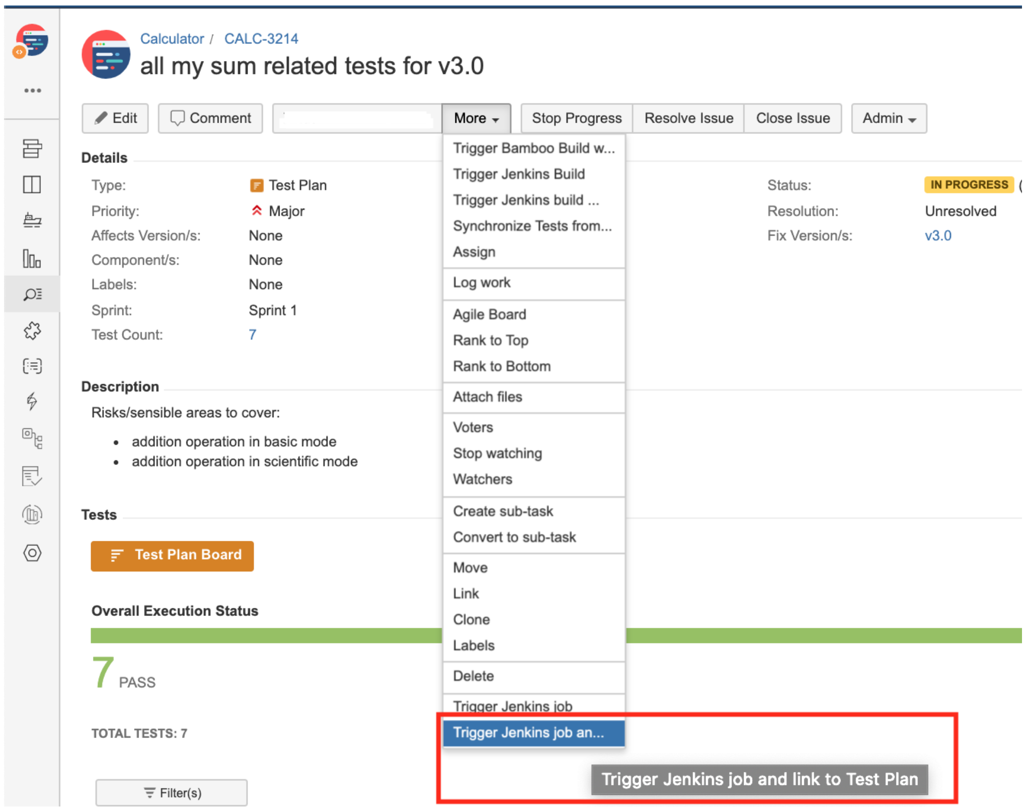
HTTP method
POST

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.

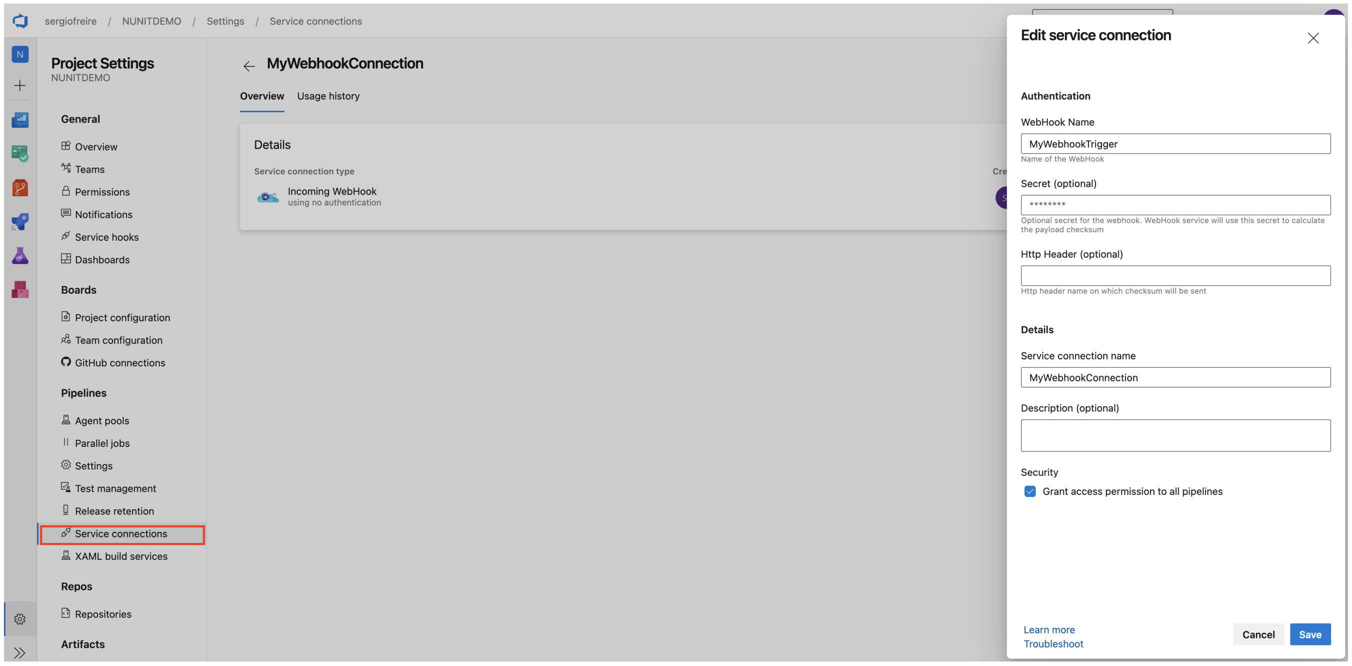


Azure DevOps

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.



Service connection type: Incoming WebHook using no authentication

Edit service connection

Authentication

WebHook Name: MyWebhookTrigger
Name of the WebHook

Secret (optional): ********
Optional secret for the webhook. WebHook service will use this secret to calculate the payload checksum

Http Header (optional):
Http header name on which checksum will be sent

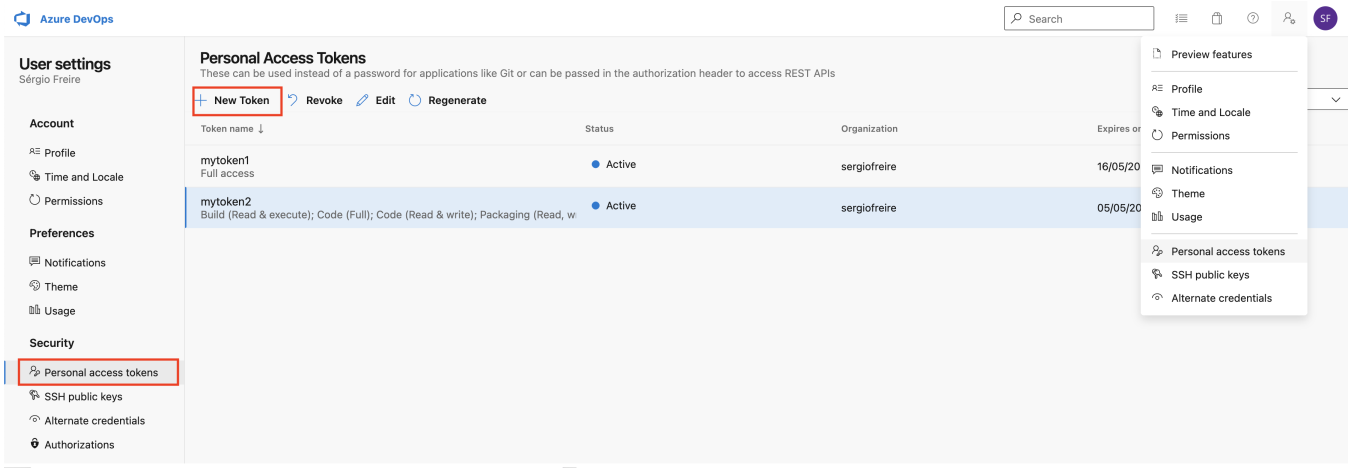
Details

Service connection name: MyWebhookConnection

Description (optional):

Security
[✓] Grant access permission to all pipelines

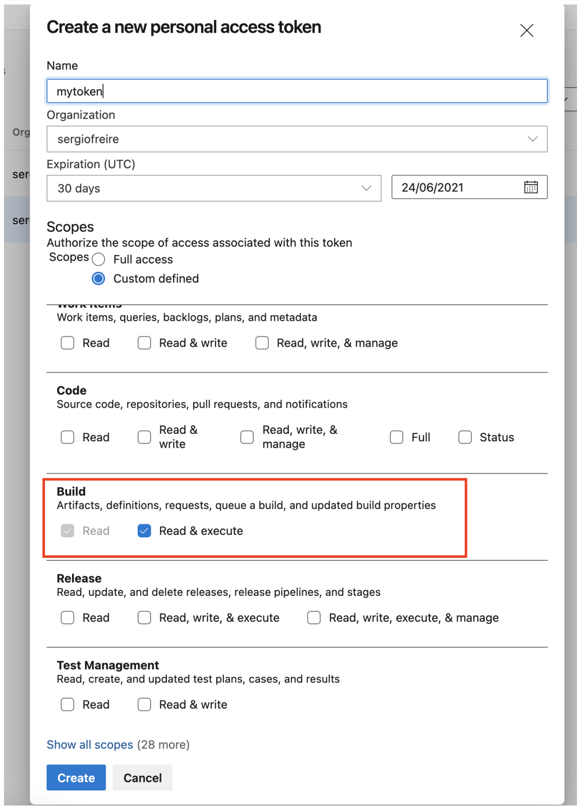
Create a Personal Access Token (PAT), so you can use it as the password in API requests, along with the "organization name" as username.



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

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



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
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

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.

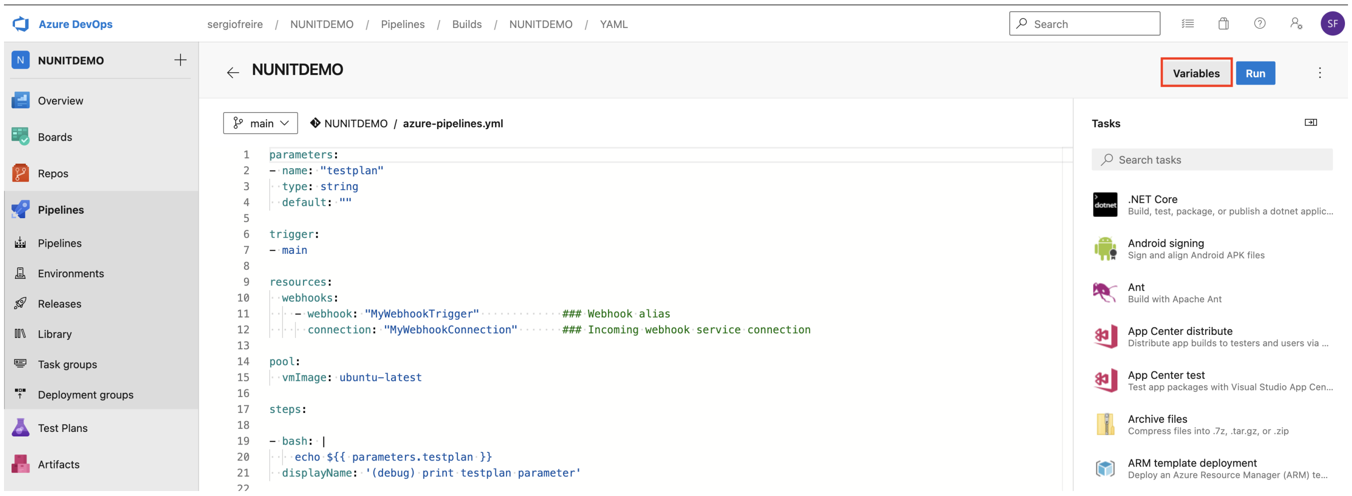
In the following example, the pipeline will receive the Test Plan issue key as an input parameter. It will then run the build, including the automated tests, and in the end it will report the results back to Xray using "curl" utility.

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

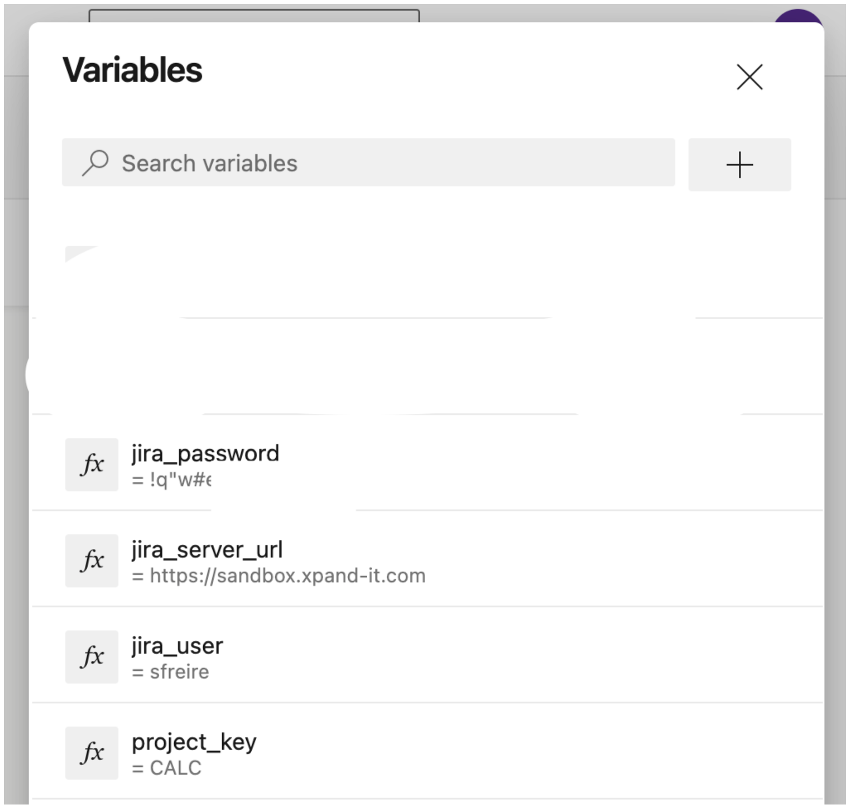
/azure-pipelines.yml

```yaml
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.


Azure DevOps
sergiofreire / NUNITDEMO / Pipelines / Builds / NUNITDEMO / YAML
Search
NUNITDEMO
Overview
Boards
Repos
Pipelines
Pipelines
Environments
Releases
Library
Task groups
Deployment groups
Test Plans
Artifacts
NUNITDEMO
Variables
Run
main
NUNITDEMO / azure-pipelines.yml
Tasks
Search tasks
.NET Core
Build, test, package, or publish a dotnet applic...
Android signing
Sign and align Android APK files
Ant
Build with Apache Ant
App Center distribute
Distribute app builds to testers and users via ...
App Center test
Test app packages with Visual Studio App Cen...
Archive files
Compress files into .7z, .tar.gz, or .zip
ARM template deployment
Deploy an Azure Resource Manager (ARM) te...

```yaml
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'
```



Variables

Search variables

[redacted]

jira_password
= !q"w#ɛ

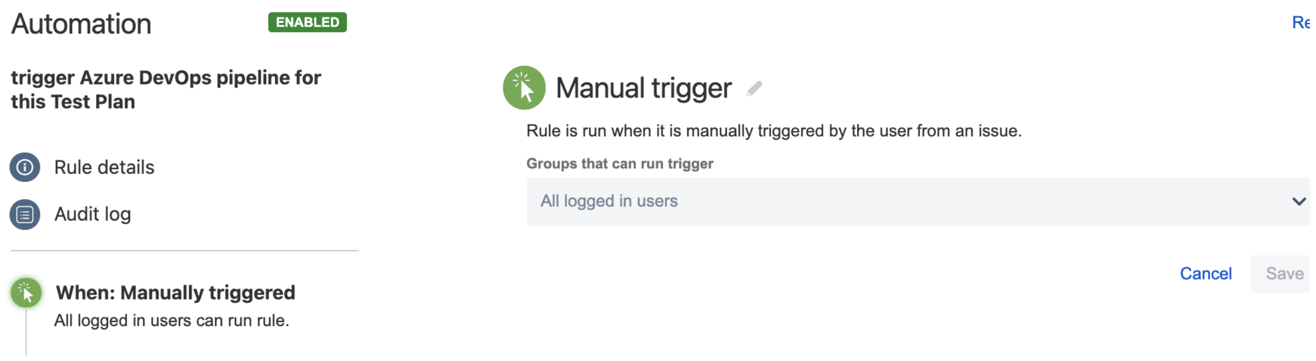
jira_server_url
= https://sandbox.xpand-it.com

jira_user
= sfreire

project_key
= CALC

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 Azure DevOps pipeline for this Test Plan

Rule details

Audit log

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

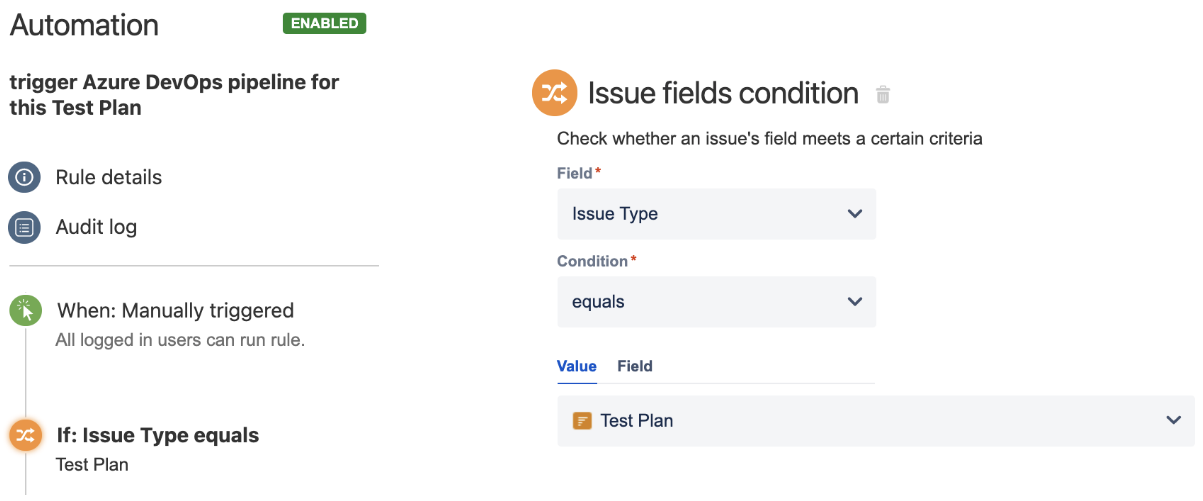
Manual trigger

Rule is run when it is manually triggered by the user from an issue.

Groups that can run trigger: All logged in users

Cancel Save

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



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

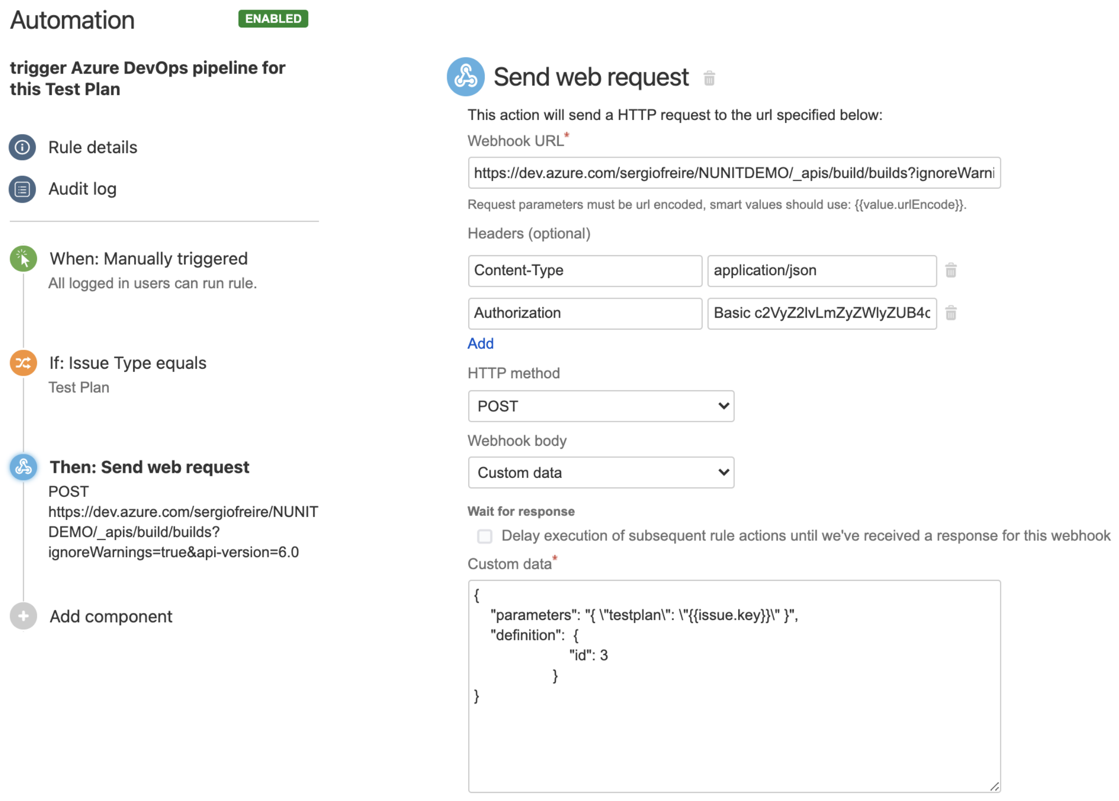
Field*: Issue Type

Condition*: equals

Value | Field

Test Plan

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



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/NUNITDEMO/_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*

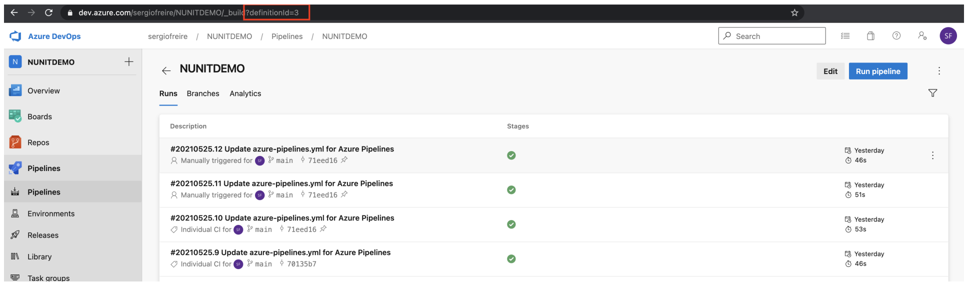
```
{
   "parameters": "{ \"testplan\": \"{{issue.key}}\" }",
   "definition":  {
                  "id": 3
            }
}
```

- 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)

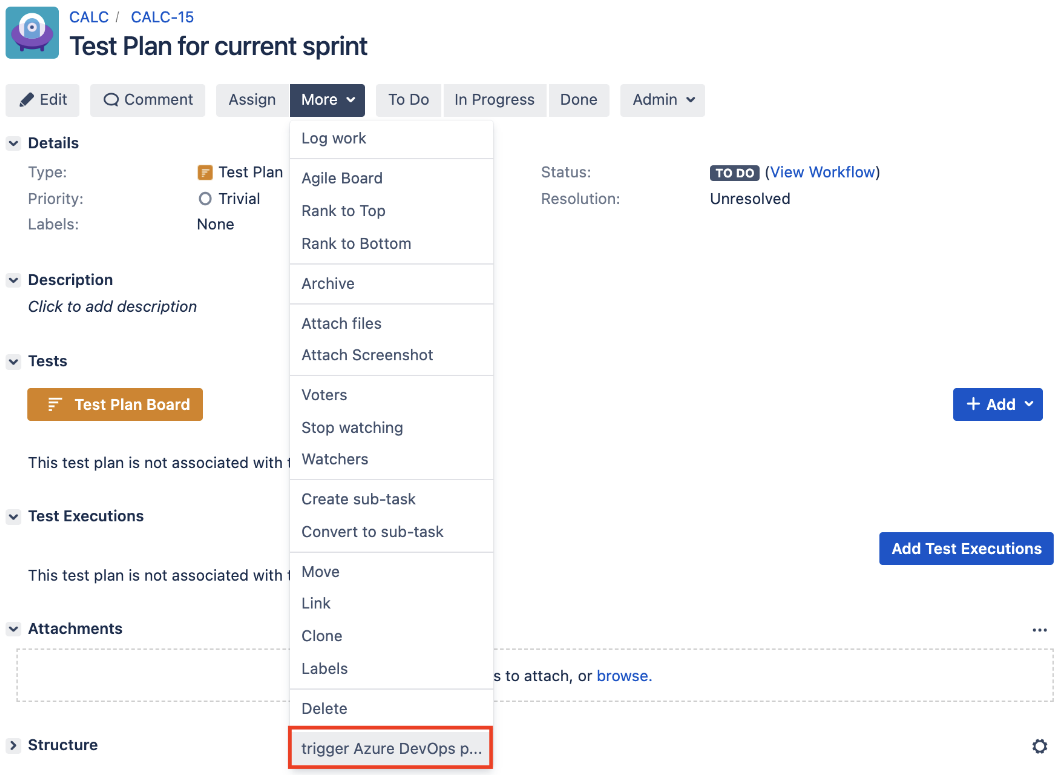
```
{
    "parameters": "{ \"testplan\": \"{{issue.key}}\" }",
    "definition":  {
                       "id": 3
                   }
}
```

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



Description	Stages	
#20210525.12 Update azure-pipelines.yml for Azure Pipelines Manually triggered for SF main 71eed16	✓	Yesterday 46s
#20210525.11 Update azure-pipelines.yml for Azure Pipelines Manually triggered for SF main 71eed16	✓	Yesterday 51s
#20210525.10 Update azure-pipelines.yml for Azure Pipelines Individual CI for SF main 71eed16	✓	Yesterday 53s
#20210525.9 Update azure-pipelines.yml for Azure Pipelines Individual CI for SF main 70135b7	✓	Yesterday 46s

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



Test Plan for current sprint

Details

Type: Test Plan
Priority: Trivial
Labels: None
Status: TO DO (View Workflow)
Resolution: Unresolved

Description

Click to add description

Tests

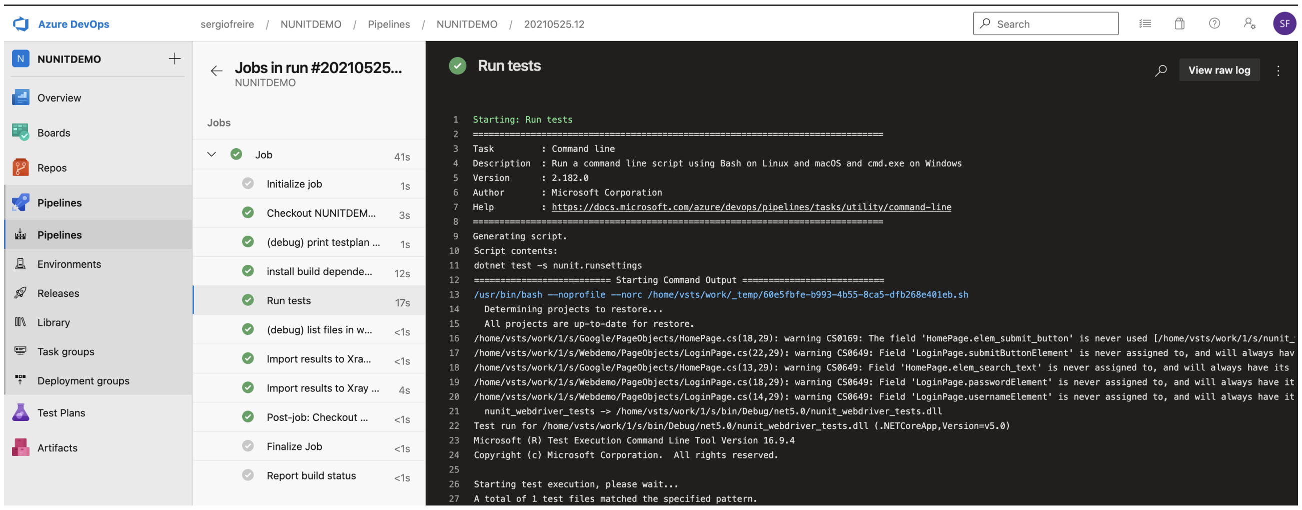
This test plan is not associated with t

Test Executions

This test plan is not associated with t

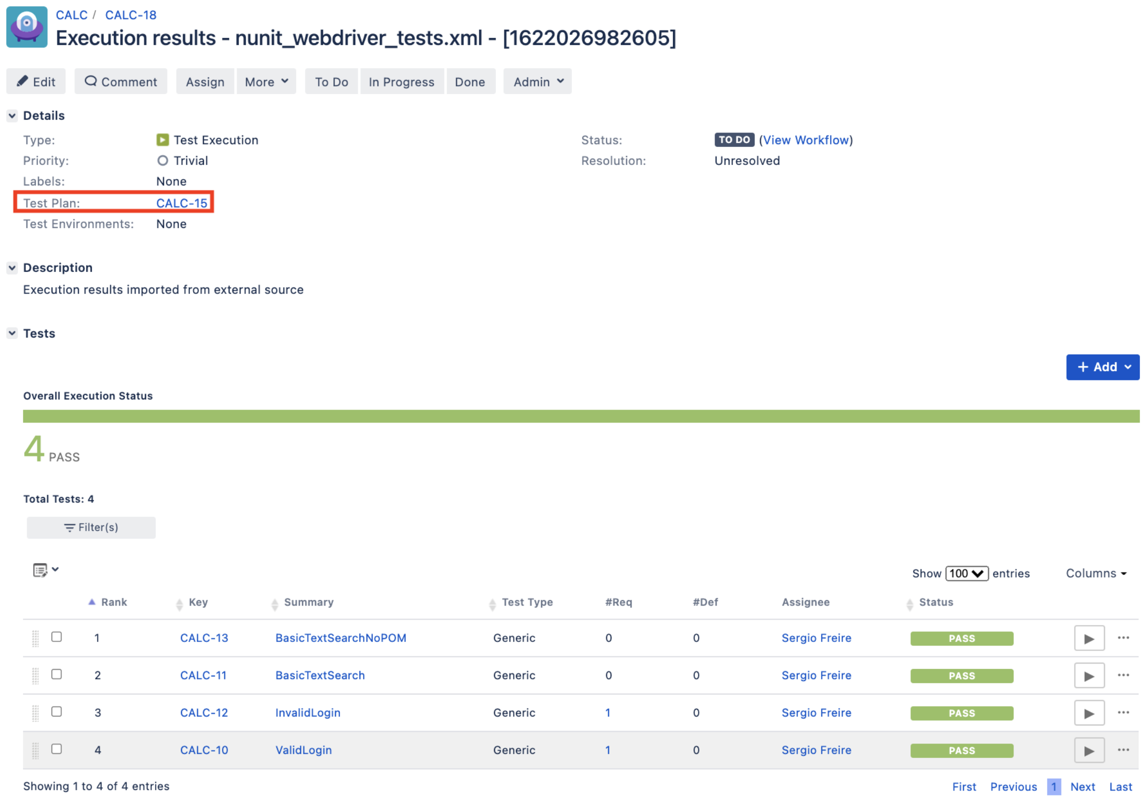
Attachments

s to attach, or browse.



```
Starting: Run tests
==============================================================================
Task         : Command line
Description  : Run a command line script using Bash on Linux and macOS and cmd.exe on Windows
Version      : 2.182.0
Author       : Microsoft Corporation
Help         : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/command-line
==============================================================================
Generating script.
Script contents:
dotnet test -s nunit.runsettings
========================== Starting Command Output ===========================
/usr/bin/bash --noprofile --norc /home/vsts/work/_temp/60e5fbfe-b993-4b55-8ca5-dfb268e401eb.sh
  Determining projects to restore...
  All projects are up-to-date for restore.
/home/vsts/work/1/s/Google/PageObjects/HomePage.cs(18,29): warning CS0169: The field 'HomePage.elem_submit_button' is never used [/home/vsts/work/1/s/nunit_
/home/vsts/work/1/s/Webdemo/PageObjects/LoginPage.cs(22,29): warning CS0649: Field 'LoginPage.submitButtonElement' is never assigned to, and will always hav
/home/vsts/work/1/s/Google/PageObjects/HomePage.cs(13,29): warning CS0649: Field 'HomePage.elem_search_text' is never assigned to, and will always have its
/home/vsts/work/1/s/Webdemo/PageObjects/LoginPage.cs(18,29): warning CS0649: Field 'LoginPage.passwordElement' is never assigned to, and will always have it
/home/vsts/work/1/s/Webdemo/PageObjects/LoginPage.cs(14,29): warning CS0649: Field 'LoginPage.usernameElement' is never assigned to, and will always have it
  nunit_webdriver_tests -> /home/vsts/work/1/s/bin/Debug/net5.0/nunit_webdriver_tests.dll
Test run for /home/vsts/work/1/s/bin/Debug/net5.0/nunit_webdriver_tests.dll (.NETCoreApp,Version=v5.0)
Microsoft (R) Test Execution Command Line Tool Version 16.9.4
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
```

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.



Execution results - nunit_webdriver_tests.xml - [1622026982605]

Details

Type:	Test Execution	Status:	TO DO (View Workflow)
Priority:	Trivial	Resolution:	Unresolved
Labels:	None		
Test Plan:	CALC-15		
Test Environments:	None		

Description

Execution results imported from external source

Tests

Overall Execution Status

4 PASS

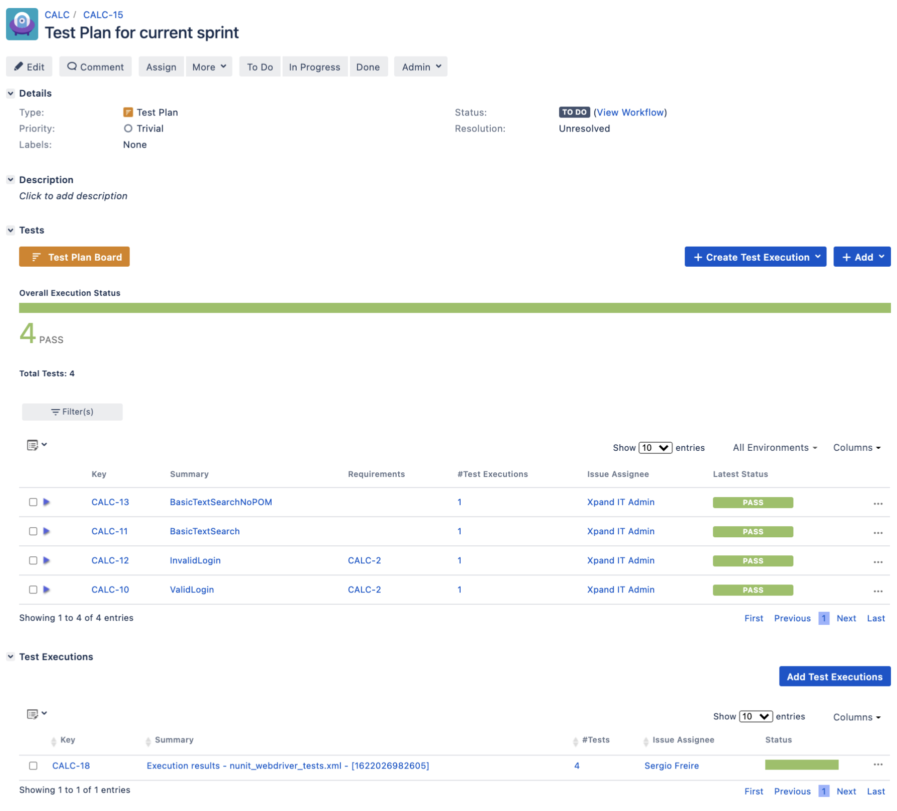
Total Tests: 4

Show 100 entries

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

Details

Type:	Test Plan	Status:	TO DO (View Workflow)
Priority:	Trivial	Resolution:	Unresolved
Labels:	None		

Description

Click to add description

Tests

Overall Execution Status

4 PASS

Total Tests: 4

Key	Summary	Requirements	#Test Executions	Issue Assignee	Latest Status
CALC-13	BasicTextSearchNoPOM		1	Xpand IT Admin	PASS
CALC-11	BasicTextSearch		1	Xpand IT Admin	PASS
CALC-12	InvalidLogin	CALC-2	1	Xpand IT Admin	PASS
CALC-10	ValidLogin	CALC-2	1	Xpand IT Admin	PASS

Showing 1 to 4 of 4 entries

Test Executions

Key	Summary	#Tests	Issue Assignee	Status
CALC-18	Execution results - nunit_webdriver_tests.xml - [1622026982605]	4	Sergio Freire	🟩

Showing 1 to 1 of 1 entries

Travis CI

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

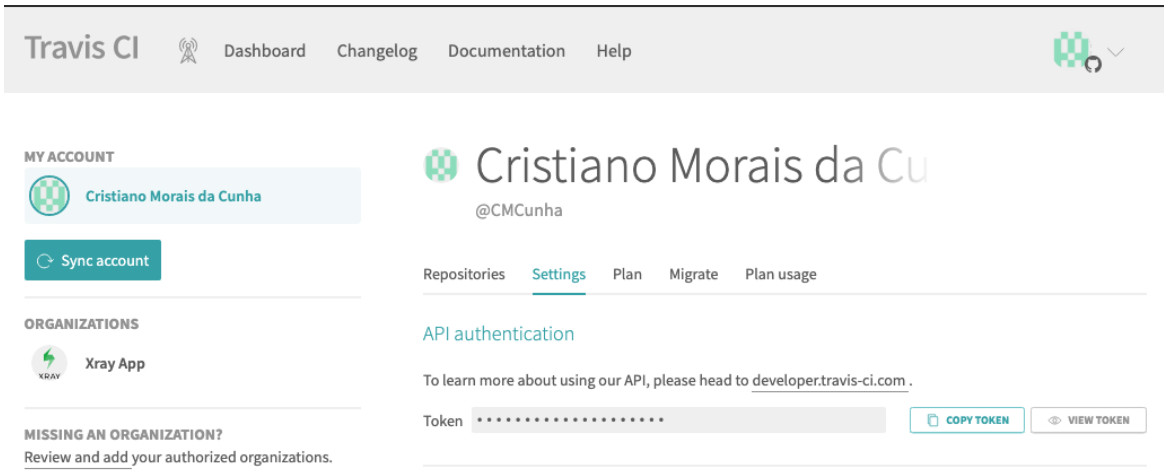
In this simple scenario, we'll implement a rule, triggered manually, that will trigger a TravisCI project/job. The action will be available from the "Automation" panel, for all Test Plan issues of the selected project.

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.



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.

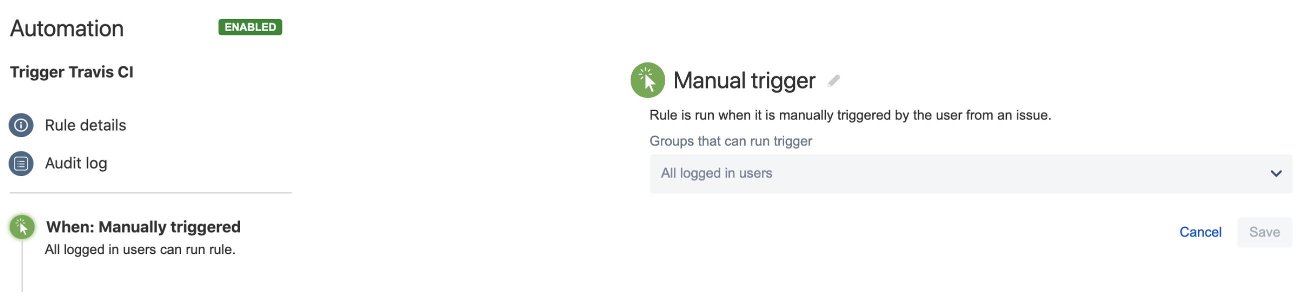
- *PROJECTKEY* - The key that identifies the project on the Jira side.
- *TESTPLAN* - The Test Plan key used to identify the Test Plan to associate the execution with.

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"



Automation ENABLED

Trigger Travis CI

Rule details

Audit log

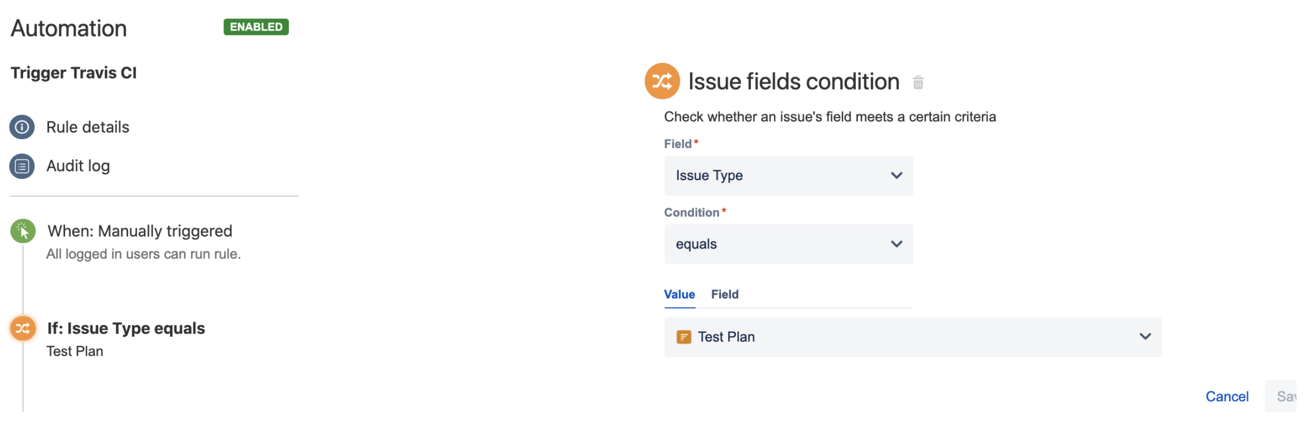
When: Manually triggered
All logged in users can run rule.

Manual trigger

Rule is run when it is manually triggered by the user from an issue.

Groups that can run trigger: All logged in users

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:



Automation ENABLED

Trigger Travis CI

Rule details

Audit log

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

If: Issue Type equals
Test Plan

Issue fields condition

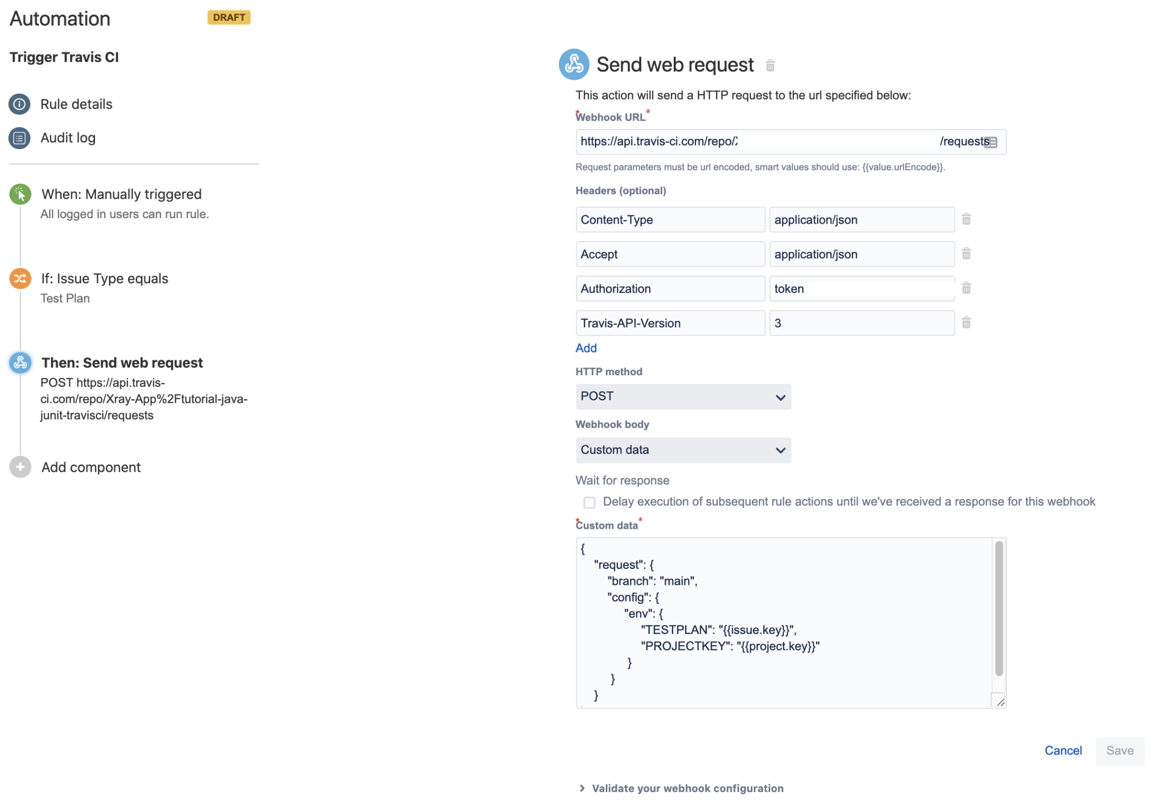
Check whether an issue's field meets a certain criteria

Field*: Issue Type

Condition*: equals

Value: Test Plan

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*

https://api.travis-ci.com/repo/ /requests

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

Headers (optional)

Content-Type	application/json
Accept	application/json
Authorization	token
Travis-API-Version	3

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*

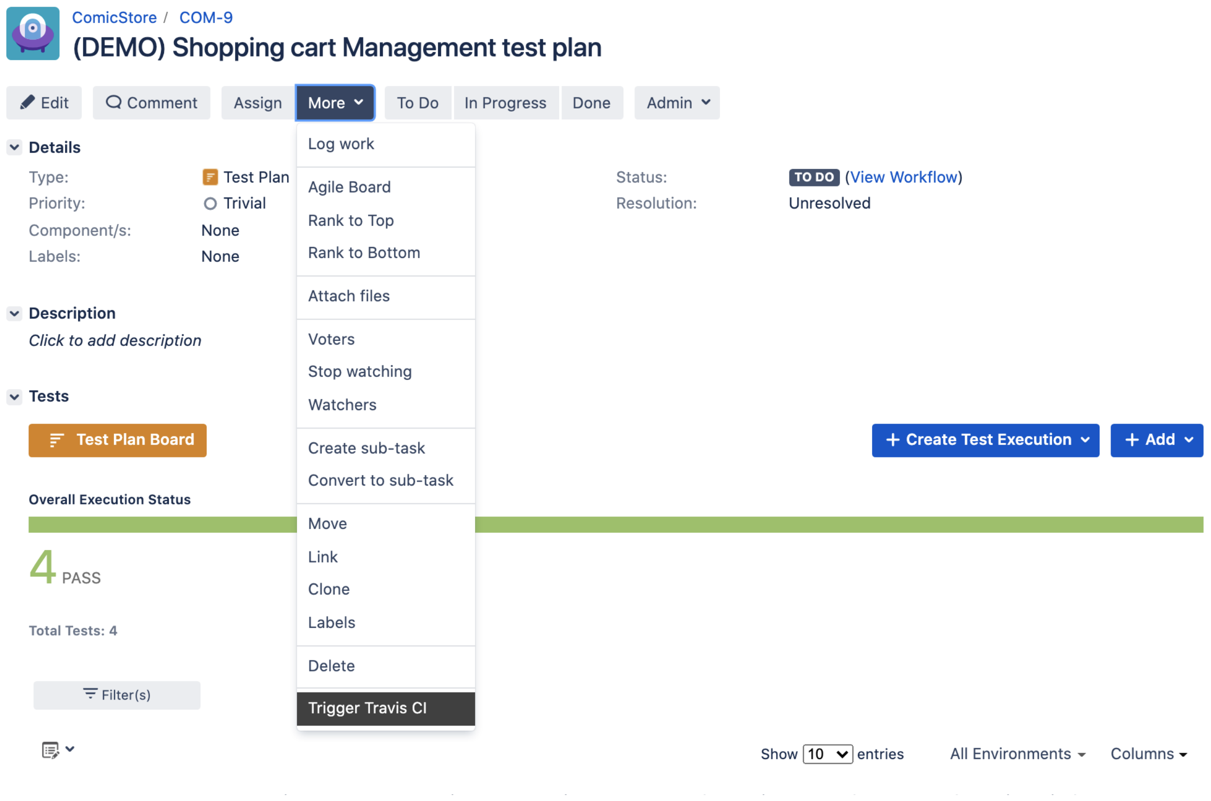
```
{
   "request": {
      "branch": "main",
      "config": {
          "env": {
              "TESTPLAN": "{{issue.key}}",
              "PROJECTKEY": "{{project.key}}"
          }
      }
   }
```

Cancel Save

Validate your webhook configuration

- the Webhook URL provided above follows this syntax:
  - *<TravisCI_API_URL>/repo/{slug|id}/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

Details

Type:	Test Plan	Status:	TO DO (View Workflow)
Priority:	Trivial	Resolution:	Unresolved
Component/s:	None		
Labels:	None		

Description

Click to add description

Tests

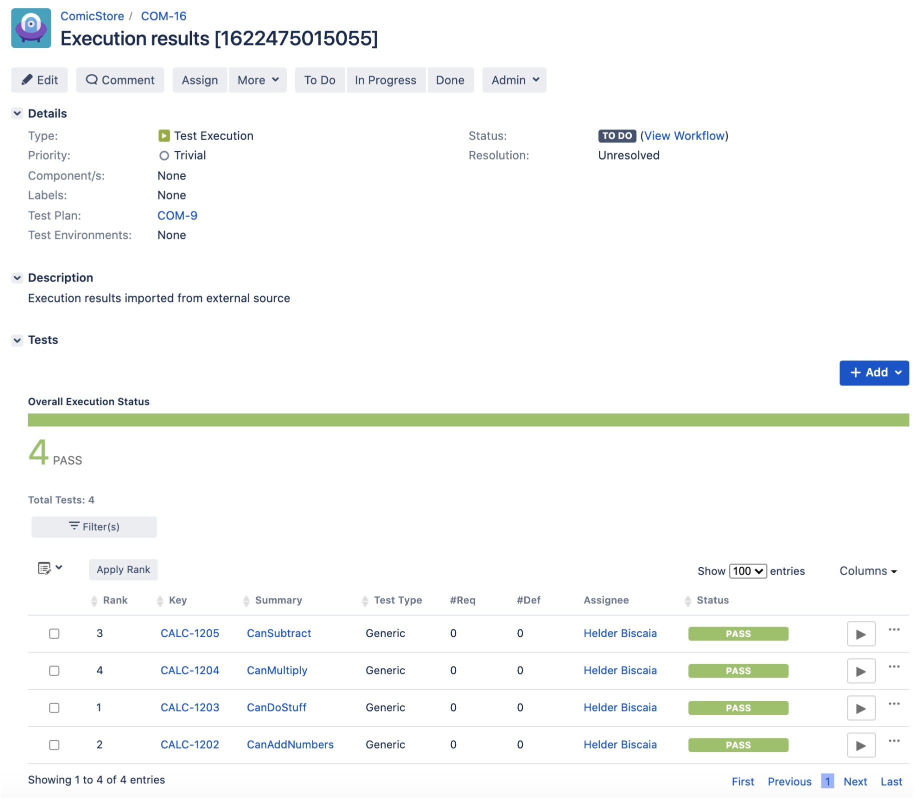
Overall Execution Status

4 PASS

Total Tests: 4

Show 10 entries

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]

Details

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

Description

Execution results imported from external source

Tests

Overall Execution Status

4 PASS

Total Tests: 4

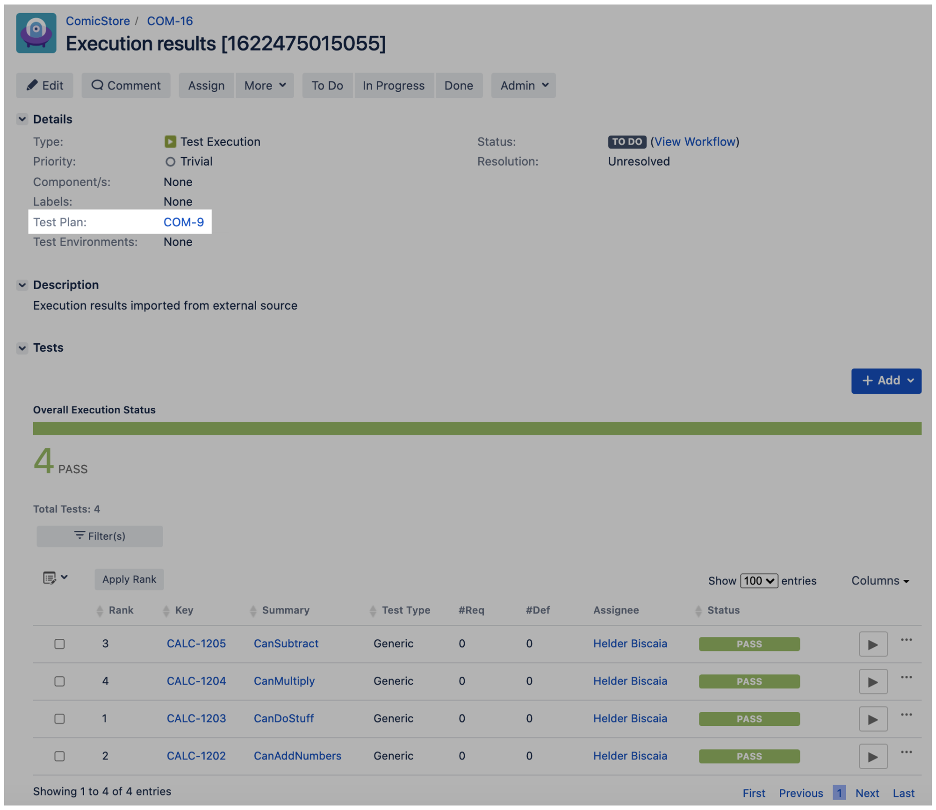
Show 100 entries

	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

Associated with the Test Plan that we have passed along:



Execution results [1622475015055]

Details

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

Description

Execution results imported from external source

Tests

Overall Execution Status

4 PASS

Total Tests: 4

Show 100 entries

	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

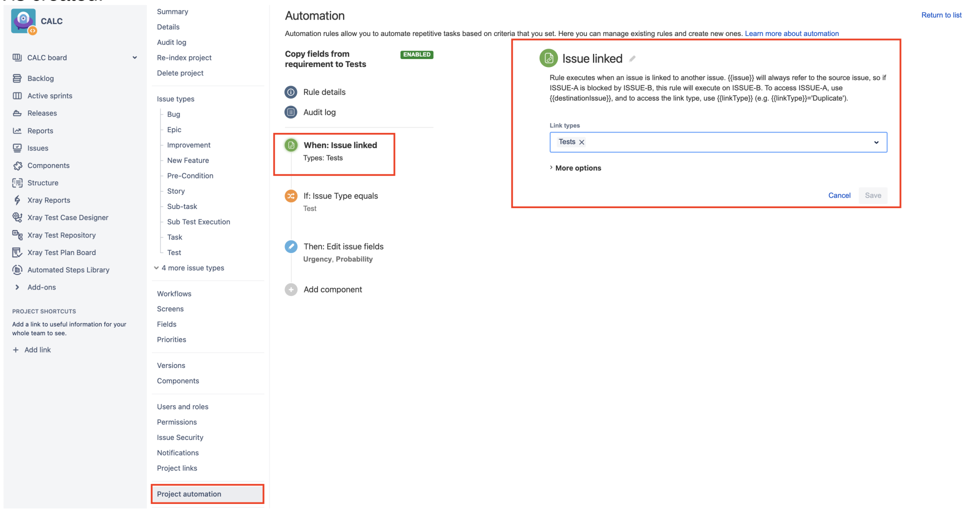
Generic automation of processes

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.
    a. 

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

When: Issue linked
Types: Tests

If: Issue Type equals
Test

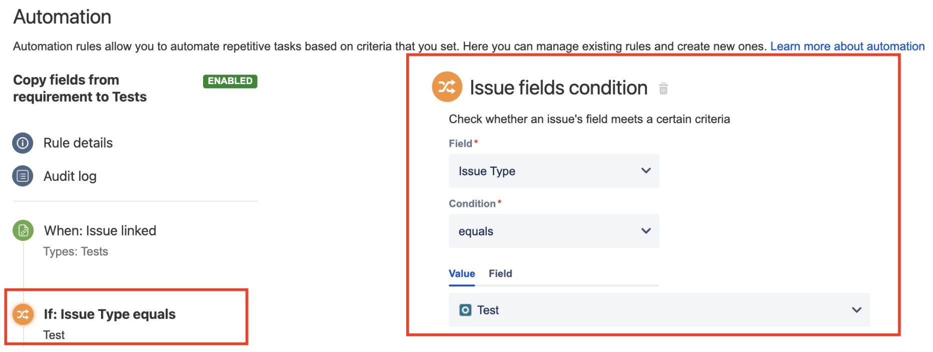
Then: Edit issue fields
Urgency, Probability

Issue linked

Rule executes when an issue is linked to another issue. {{issue}} 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

More options
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

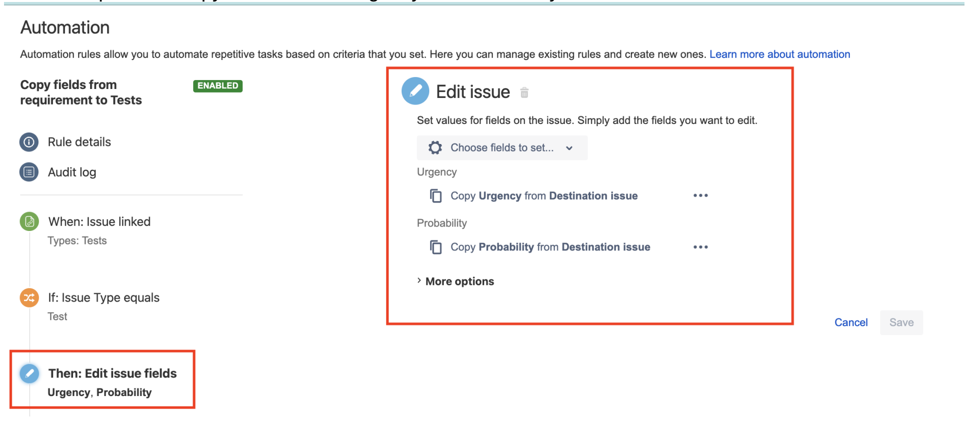
Condition*: equals

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**

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

a.

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/transitionTests linked to a requirement whenever the requirement is transitioned or changed

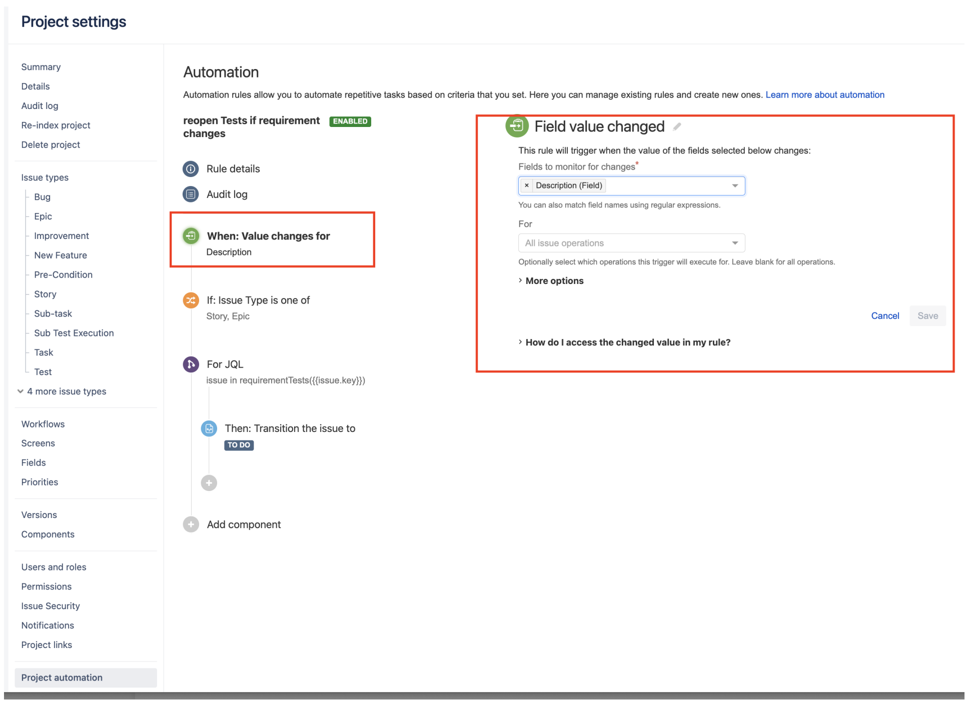
Whenever you change the specification of a requirement/story, you most probably will need to review the Tests that you have already specified.

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"

a. 
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

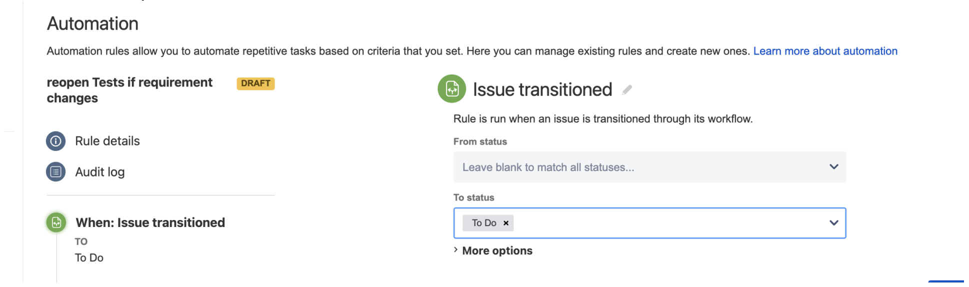
Field value changed

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**

Cancel Save

› **How do I access the changed value in my rule?**

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.

i. 
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

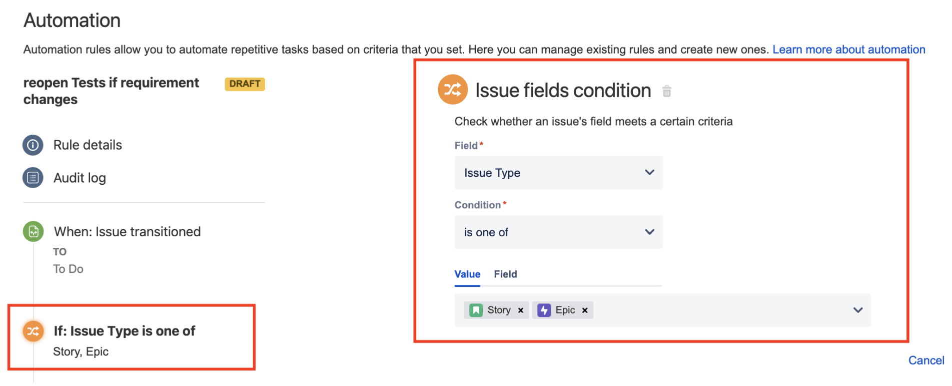
Rule details
Audit log

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**

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)

a. 
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

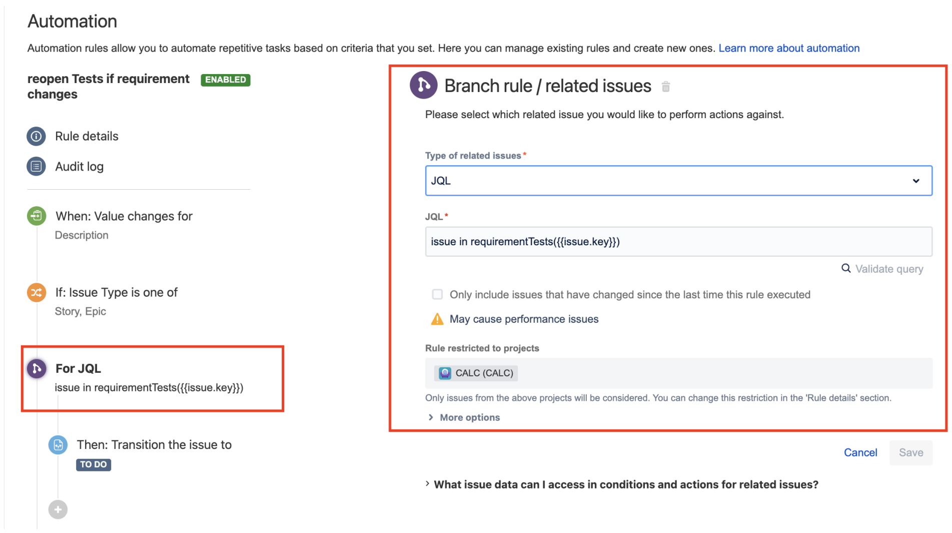
If: Issue Type is one of
Story, Epic

Issue fields condition

Check whether an issue's field meets a certain criteria
Field*
Issue Type
Condition*
is one of
Value Field
Story × Epic ×

Cancel

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.

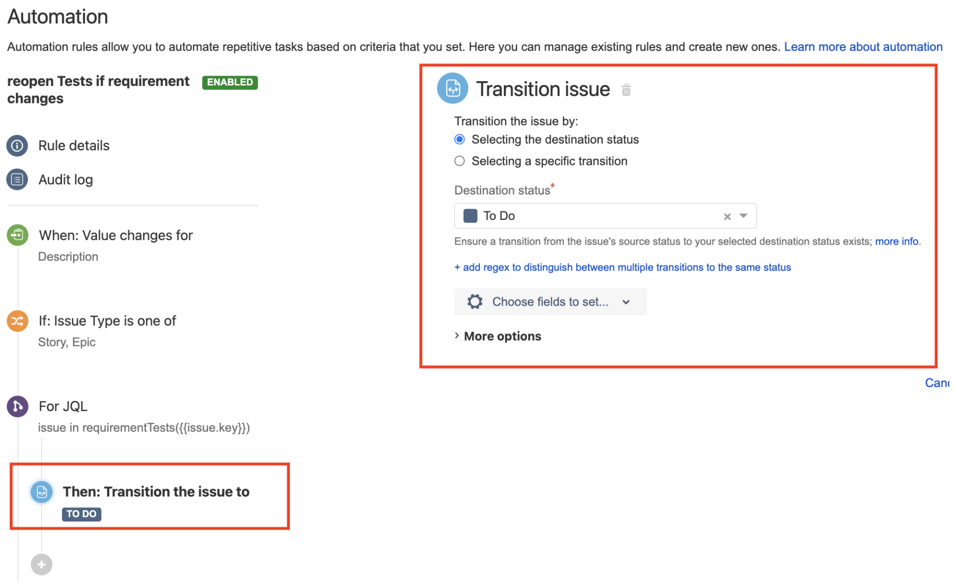
More options

Cancel Save

What issue data can I access in conditions and actions for related issues?

a.

4. 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

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