Integration with GitLab

GitLab is a well-known CI/CD tool available on-premises and as SaaS.

Xray does not provide yet a plugin for GitLab. However, it is easy to setup GitLab in order to integrate it with Xray.

Since Xray provides a full REST API, you may interact with Xray, for submitting results for example.

JUnit example

In this scenario, we want to get visibility of the automated test results from some tests implemented in Java, using the JUnit framework.

This recipe could also be applied for other frameworks such as NUnit or Robot.

We need to setup a Git repository containing the code along with the configuration for GitLab build process.

The tests are implemented in a JUnit class as follows.

CalcTest.java

```
package com.xpand.java;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

public class CalcTest {

    @Before
    public void setUp() throws Exception {

    }

    @After
    public void tearDown() throws Exception {

    }

        @Test
    public void CanAddNumbers()
    {
        assertThat(Calculator.Add(1, 1), is(2));
        assertThat(Calculator.Add(-1, 1), is(0));
    }


    @Test
    public void CanSubtract()
    {
        assertThat(Calculator.Subtract(1, 1), is(0));
        assertThat(Calculator.Subtract(-1, -1), is(0));
        assertThat(Calculator.Subtract(100, 5), is(95));
    }


    @Test
    public void CanMultiply()
    {
        assertThat(Calculator.Multiply(1, 1), is(1));
        assertThat(Calculator.Multiply(-1, -1), is(1));
        assertThat(Calculator.Multiply(100, 5), is(500));
    }


    public void CanDivide()
    {
        assertThat(Calculator.Divide(1, 1), is(1));
        assertThat(Calculator.Divide(-1, -1), is(1));
        assertThat(Calculator.Divide(100, 5), is(20));
    }


    @Test
    public void CanDoStuff()
    {
        assertThat(true, is(true));
    }


}
```

The GitLab configuration file `.gitlab-ci.yml` contains the definition of the build steps, including running the automated tests and submitting the results.

.gitlab-ci.yml

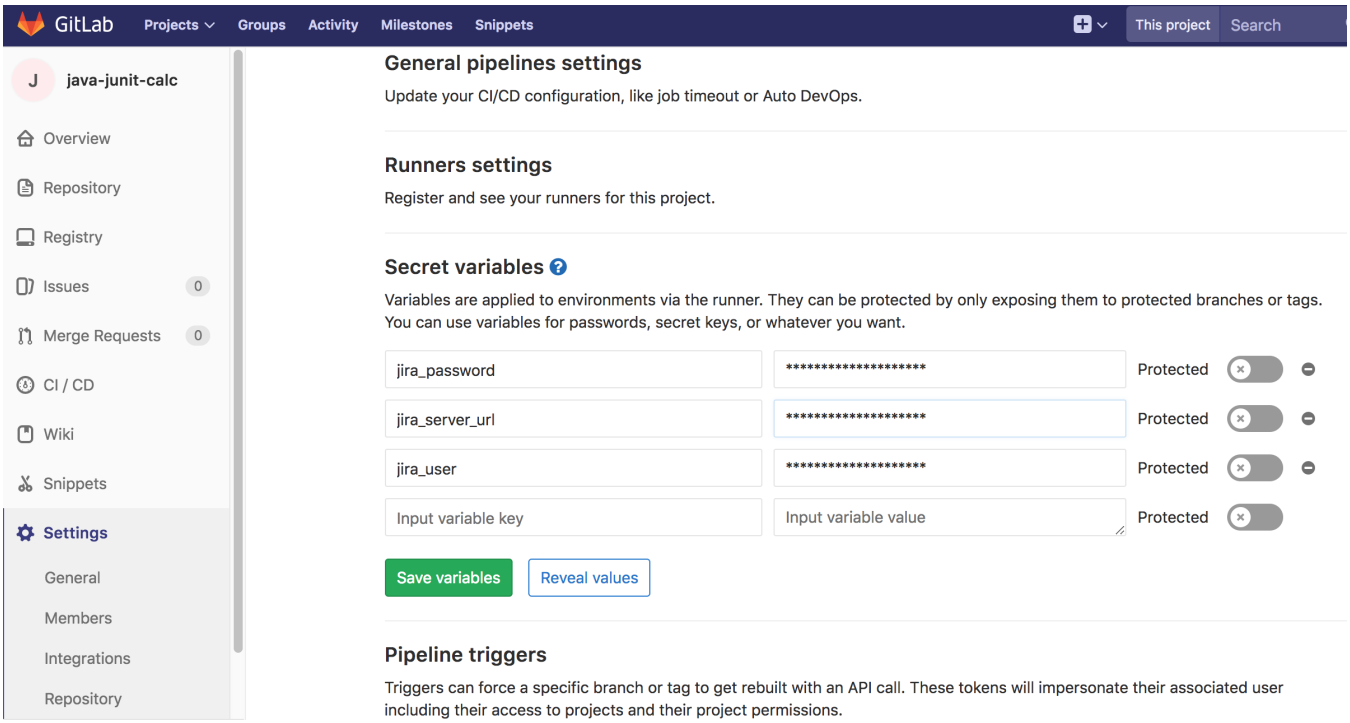
```
# Use Maven 3.5 and JDK8
image: maven:3.5-jdk-8

variables:
  # This will supress any download for dependencies and plugins or upload messages which would clutter the
console log.
  # `showDateTime` will show the passed time in milliseconds. You need to specify `--batch-mode` to make this
work.
  MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository -Dorg.slf4j.simpleLogger.log.org.apache.maven.cli.transfer.
Slf4jMavenTransferListener=WARN -Dorg.slf4j.simpleLogger.showDateTime=true -Djava.awt.headless=true"
  # As of Maven 3.3.0 instead of this you may define these options in `.mvn/maven.config` so the same config is
used
  # when running from the command line.
  # `installAtEnd` and `deployAtEnd`are only effective with recent version of the corresponding plugins.
  MAVEN_CLI_OPTS: "--batch-mode --errors --fail-at-end --show-version -DinstallAtEnd=true -DdeployAtEnd=true"


# Cache downloaded dependencies and plugins between builds.
# To keep cache across branches add 'key: "$CI_JOB_REF_NAME"'
cache:
  paths:
    - .m2/repository


maven_build:
  script:
    - echo "building my amazing repo..."
    - mvn test
    - 'curl -H "Content-Type: multipart/form-data" -u $jira_user:$jira_password -F "file=@target/surefire-
reports/TEST-com.xpand.java.CalcTest.xml" "$jira_server_url/rest/raven/1.0/import/execution/junit?
projectKey=CALC"'
    - echo "done"
```

In order to submit those results, we'll just need to invoke the REST API (as detailed in Import Execution Results - REST).

However, we do not want to have the JIRA credentials hardcoded in GitLab's configuration file. Therefore, we'll use some secret variables defined in GitLab project settings.

Please note

The user present in the configuration below must exist in the JIRA instance and have permission to Create Test and Test Execution Issues



In `.gitlab-ci.yml` a "step" must be included in the maven_build section, that will use "curl" in order to submit the results to the REST API.

```
curl -H "Content-Type: multipart/form-data" -u $jira_user:$jira_password -F "file=@target/surefire-reports/TEST-com.xpand.java.CalcTest.xml" "$jira_server_url/rest/raven/1.0/import/execution/junit?projectKey=CALC"
```

We're using "curl" utility that comes in Unix based OS'es but you can easily use another tool to make the HTTP request; however, "curl" is provided in the container used by GitLab.

Robot Framework example

In this scenario, we want to get visibility of the automated test results from some UI tests implemented in Robot Framework (Python) together with Selenium (using the "robotframework-seleniumlibrary"), and using Chrome for testing.

We need to set up a Git repository containing the code along with the configuration for GitLab build process.

The tests are implemented in Robot Framework .robot files as follows.

valid_login.robot

```
*** Settings ***
Documentation     A test suite with a single test for valid login.
...
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource          resource.robot

*** Test Cases ***
Valid Login
    [Tags]  UI
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

The GitLab configuration file `.gitlab-ci.yml` contains the definition of the build steps, including running the automated tests and submitting the results, as two different stages.

.gitlab-ci.yml

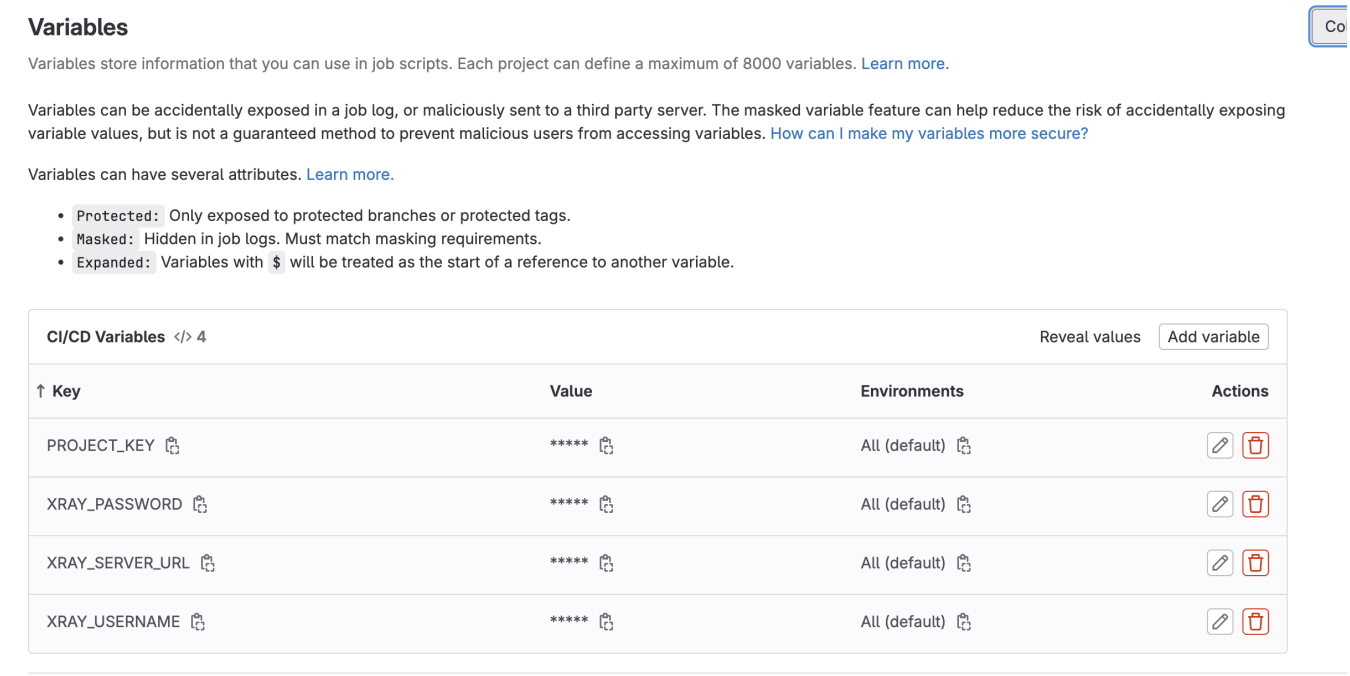
```
# Official language image. Look for the different tagged releases at:
# https://hub.docker.com/r/library/python/tags/
image: python:3.12.2

# Change pip's cache directory to be inside the project directory since we can
# only cache local items.
variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

# https://pip.pypa.io/en/stable/topics/caching/
cache:
  paths:
    - .cache/pip

stages:
  - execute_automated_tests
  - upload_test_results

before_script:
  - python --version ; pip --version  # For debugging
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate
  - pip install -r requirements.txt
  - apt-get update


test:
  stage: execute_automated_tests
  before_script: |
    set -e
    apt-get install -yqq unzip curl
    # Install Chrome & chromedriver
    curl -sS -o - https://dl.google.com/linux/linux_signing_key.pub | apt-key add -
    echo "deb https://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google.list
    apt update && apt install google-chrome-stable -y
    wget -O /tmp/chromedriver.zip https://storage.googleapis.com/chrome-for-testing-public/121.0.6167.85/linux64
/chromedriver-linux64.zip
    ls -la /tmp/chromedriver.zip
    unzip -j /tmp/chromedriver.zip chromedriver-linux64/chromedriver -d /usr/local/bin/
    nohup python demoapp/server.py &
  script: |
    chromedriver -v && \
    pip install -r requirements.txt && \
    robot -x junit.xml -o output.xml login_tests || true
  allow_failure: true
  artifacts:
    paths:
      - output.xml
    when: always


upload_results_to_xray:
  stage: upload_test_results
  script:
    - echo "uploading results to Xray..."
    - 'curl -H "Content-Type: multipart/form-data" -u $XRAY_USERNAME:$XRAY_PASSWORD -F "file=@output.xml"
"$XRAY_SERVER_URL/rest/raven/2.0/import/execution/robot?projectKey=$PROJECT_KEY"'
    - echo "done"
  dependencies:
  - test
```

In order to submit those results, we'll just need to invoke the REST API (as detailed in Import Execution Results - REST).

However, we do not want to have the Xray API credentials hardcoded in the GitLab's configuration file. Therefore, we'll use environment variables defined in the project settings, including:

- **XRAY_SERVER_URL**: Jira's base URL
- **XRAY_USERNAME**: the username used in the REST API
- **XRAY_PASSWORD**: the password used in the REST API
- **PROJECT_KEY**: Jira project

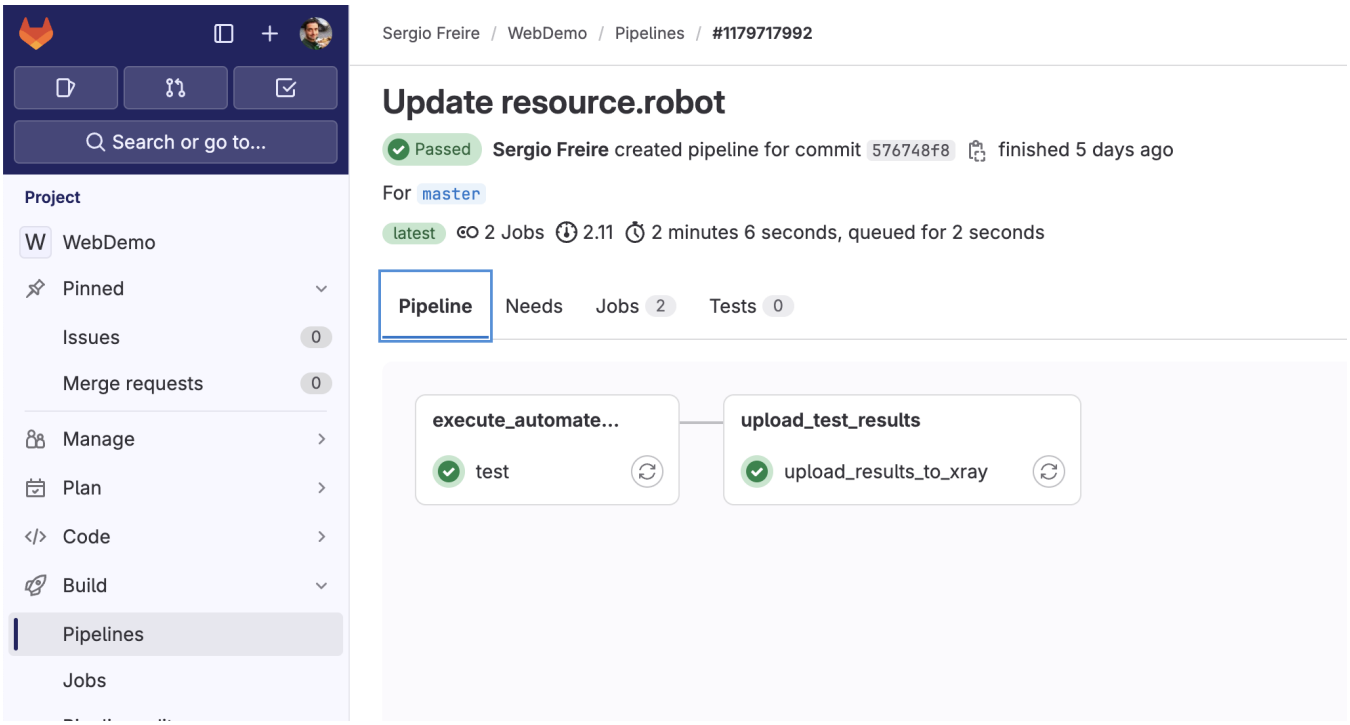
> **Please note**
>
> The user associated with the Xray's API key must have permissions to Create Test and Test Execution Issues.

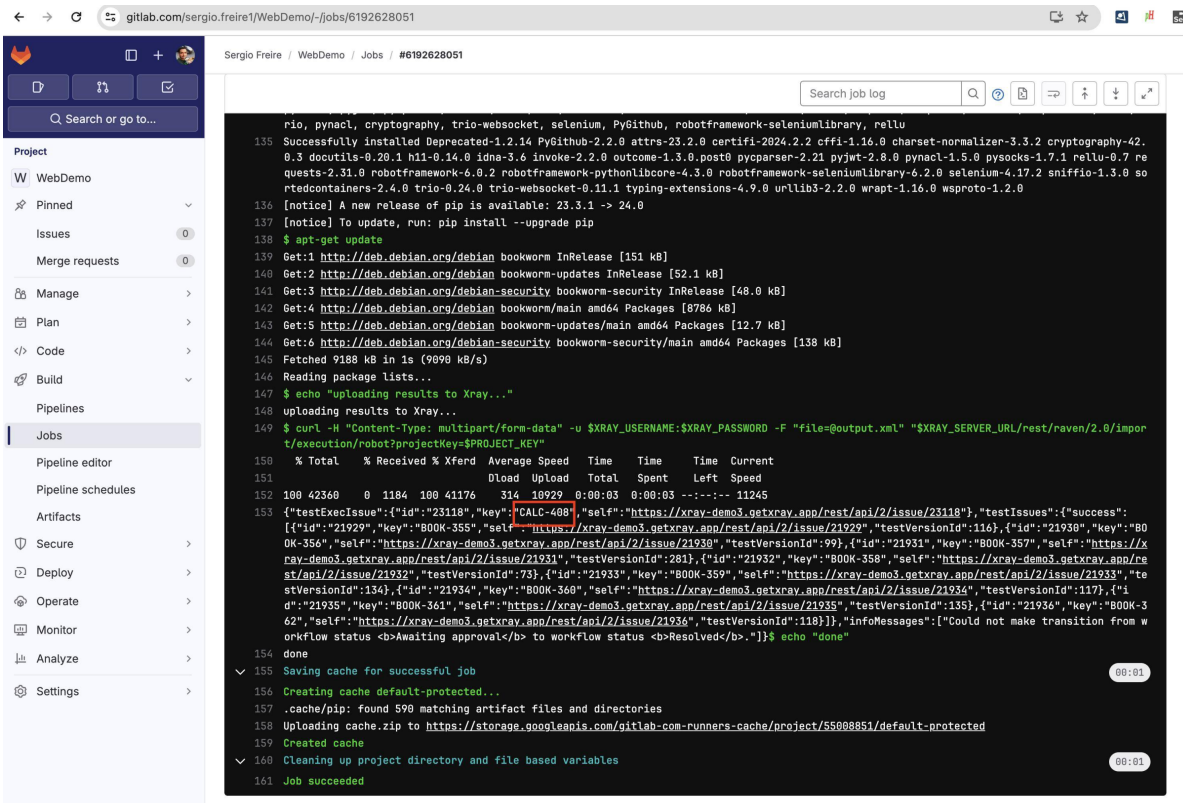


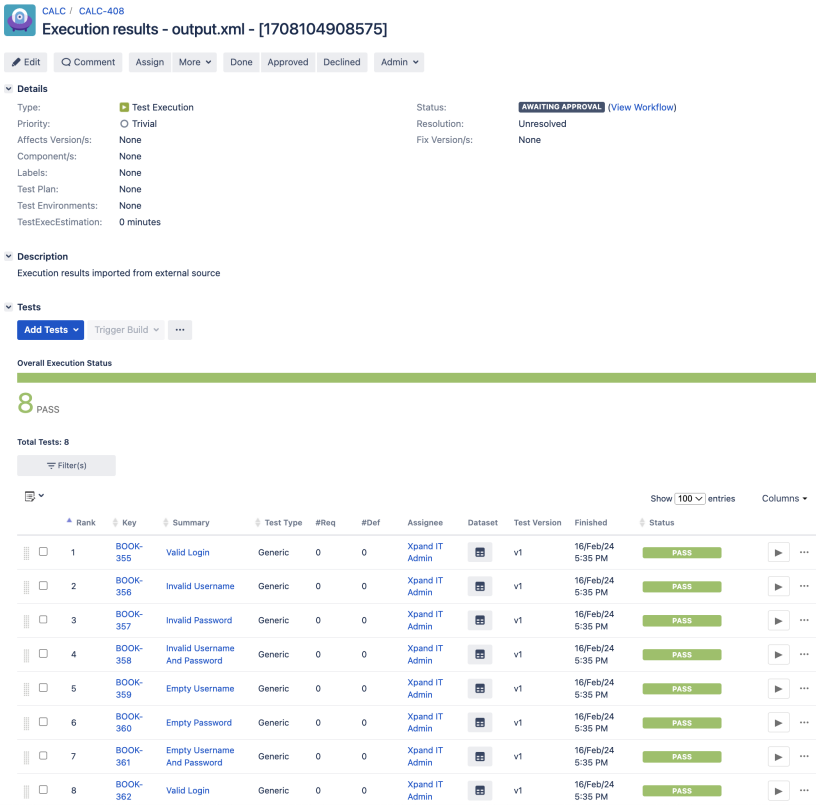
In `.gitlab-ci.yml` a "step" must be included that will use "curl" in order to submit the results to the REST API, using the Xray/Jira credentials.

```
curl -H "Content-Type: multipart/form-data" -u $XRAY_USERNAME:$XRAY_PASSWORD -F "file=@output.xml"
"$XRAY_SERVER_URL/rest/raven/2.0/import/execution/robot?projectKey=$PROJECT_KEY"
```

We're using "curl" utility that comes in Unix based OS'es but you can easily use another tool to make the HTTP request; however, "curl" is provided in the container used by GitLab.


Search or go to...
Project
WebDemo
Pinned
Issues 0
Merge requests 0
Manage
Plan
Code
Build
Pipelines
Jobs
Sergio Freire / WebDemo / Pipelines / #1179717992
Update resource.robot
Passed Sergio Freire created pipeline for commit 576748f8 finished 5 days ago
For master
latest 2 Jobs 2.11 2 minutes 6 seconds, queued for 2 seconds
Pipeline Needs Jobs 2 Tests 0
execute_automate...
test
upload_test_results
upload_results_to_xray


gitlab.com/sergio.freire1/WebDemo/-/jobs/6192628051
Search or go to...
Project
WebDemo
Pinned
Issues 0
Merge requests 0
Manage
Plan
Code
Build
Pipelines
Jobs
Pipeline editor
Pipeline schedules
Artifacts
Secure
Deploy
Operate
Monitor
Analyze
Settings
Sergio Freire / WebDemo / Jobs / #6192628051
Search job log
rio, pynacl, cryptography, trio-websocket, selenium, PyGithub, robotframework-seleniumlibrary, rellu
Successfully installed Deprecated-1.2.14 PyGithub-2.2.0 attrs-23.2.0 certifi-2024.2.2 cffi-1.16.0 charset-normalizer-3.3.2 cryptography-42.0.3 docutils-0.20.1 h11-0.14.0 idna-3.6 invoke-2.2.0 outcome-1.3.0.post0 pycparser-2.21 pyjwt-2.8.0 pynacl-1.5.0 pysocks-1.7.1 rellu-0.7 requests-2.31.0 robotframework-6.0.2 robotframework-pythonlibcore-4.3.0 robotframework-seleniumlibrary-6.2.0 selenium-4.17.2 sniffio-1.3.0 sortedcontainers-2.4.0 trio-0.24.0 trio-websocket-0.11.1 typing-extensions-4.9.0 urllib3-2.2.0 wrapt-1.16.0 wsproto-1.2.0
[notice] A new release of pip is available: 23.3.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
$ apt-get update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [52.1 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8786 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [12.7 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [138 kB]
Fetched 9188 kB in 1s (9090 kB/s)
Reading package lists...
$ echo "uploading results to Xray..."
uploading results to Xray...
$ curl -H "Content-Type: multipart/form-data" -u $XRAY_USERNAME:$XRAY_PASSWORD -F "file=@output.xml" "$XRAY_SERVER_URL/rest/raven/2.0/import/execution/robot?projectKey=$PROJECT_KEY"
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 42360 0 1184 100 41176 314 10929 0:00:03 0:00:03 --:--:-- 11245
{"testExecIssue":{"id":"23118","key":"CALC-408","self":"https://xray-demo3.getxray.app/rest/api/2/issue/23118"},"testIssues":{"success":[{"id":"21929","key":"BOOK-355","self":"https://xray-demo3.getxray.app/rest/api/2/issue/21929","testVersionId":116},{"id":"21930","key":"BOOK-356","self":"https://xray-demo3.getxray.app/rest/api/2/issue/21930","testVersionId":99},{"id":"21931","key":"BOOK-357","self":"https://xray-demo3.getxray.app/rest/api/2/issue/21931","testVersionId":281},{"id":"21932","key":"BOOK-358","self":"https://xray-demo3.getxray.app/rest/api/2/issue/21932","testVersionId":73},{"id":"21933","key":"BOOK-359","self":"https://xray-demo3.getxray.app/rest/api/2/issue/21933","testVersionId":134},{"id":"21934","key":"BOOK-360","self":"https://xray-demo3.getxray.app/rest/api/2/issue/21934","testVersionId":117},{"id":"21935","key":"BOOK-361","self":"https://xray-demo3.getxray.app/rest/api/2/issue/21935","testVersionId":135},{"id":"21936","key":"BOOK-362","self":"https://xray-demo3.getxray.app/rest/api/2/issue/21936","testVersionId":118}]},"infoMessages":["Could not make transition from workflow status Awaiting approval to workflow status Resolved."]}$ echo "done"
done
Saving cache for successful job
00:01
Creating cache default-protected...
.cache/pip: found 590 matching artifact files and directories
Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/55008851/default-protected
Created cache
Cleaning up project directory and file based variables
00:01
Job succeeded


CALC / CALC-408
Execution results - output.xml - [1708104908575]
Edit
Comment
Assign
More
Done
Approved
Declined
Admin
Details
Type: Test Execution
Priority: Trivial
Affects Version/s: None
Component/s: None
Labels: None
Test Plan: None
Test Environments: None
TestExecEstimation: 0 minutes
Status: AWAITING APPROVAL (View Workflow)
Resolution: Unresolved
Fix Version/s: None
Description
Execution results imported from external source
Tests
Add Tests
Trigger Build
Overall Execution Status
8 PASS
Total Tests: 8
Filter(s)
Show 100 entries
Columns
Rank Key Summary Test Type #Req #Def Assignee Dataset Test Version Finished Status
1 BOOK-355 Valid Login Generic 0 0 Xpand IT Admin v1 16/Feb/24 5:35 PM PASS
2 BOOK-356 Invalid Username Generic 0 0 Xpand IT Admin v1 16/Feb/24 5:35 PM PASS
3 BOOK-357 Invalid Password Generic 0 0 Xpand IT Admin v1 16/Feb/24 5:35 PM PASS
4 BOOK-358 Invalid Username And Password Generic 0 0 Xpand IT Admin v1 16/Feb/24 5:35 PM PASS
5 BOOK-359 Empty Username Generic 0 0 Xpand IT Admin v1 16/Feb/24 5:35 PM PASS
6 BOOK-360 Empty Password Generic 0 0 Xpand IT Admin v1 16/Feb/24 5:35 PM PASS
7 BOOK-361 Empty Username And Password Generic 0 0 Xpand IT Admin v1 16/Feb/24 5:35 PM PASS
8 BOOK-362 Valid Login Generic 0 0 Xpand IT Admin v1 16/Feb/24 5:35 PM PASS

Triggering automation from Xray

If you aim to trigger automation from the Xray/Jira side, please have a look at Taking advantage of Jira Cloud built-in automation capabilities page where you can see an example of triggering a GitLab pipeline from a Test Plan and reporting results back to it.

Cucumber example

Standard workflow (Xray as master)

In this scenario, we are managing the specification of Cucumber Scenarios/Scenario Outline(s) based tests in Jira, using Xray, as detailed in the "standard workflow" mentioned in Testing in BDD with Gherkin based frameworks (e.g. Cucumber)

Then we need to extract this specification from Jira (i.e. generate related Cucumber .feature files), and run it in GitLab against the code that actually implements each step that are part of those scenarios.

Finally, we can then submit the results back to JIRA and they'll be reflected on the related entities.

The GitLab configuration file `.gitlab-ci.yml` contains the definition of the build steps, including extracting the cucumber specification from Xray, running the automated tests and submitting back the results.

.gitlab-ci.yml

```
image: "ruby:2.6"

test:
  script:
  - apt-get update -qq
  - apt-get install unzip
  - gem install cucumber
  - gem install rspec-expectations
  - 'curl -u $jira_user:$jira_password "$jira_server_url/rest/raven/1.0/export/test?keys=$cucumber_keys" -o
features/features.zip'
  - mkdir -p features
  - 'rm -f features/*.feature'
  - unzip -o features/features.zip -d features/
  - cucumber -x -f json -o data.json
  - 'curl -H "Content-Type: application/json" -u $jira_user:$jira_password --data @data.json "$jira_server_url
/rest/raven/1.0/import/execution/cucumber"'
  - echo "done"
```

In this example, we're using a variable **cucumber_keys** defined in the CI/CD project level settings in GitLab. This variable contains one or more keys of the issues that will be used as source data for generating the Cucumber .feature files; it can be the key(s) of Test Plan(s), Test Execution(s), Test(s), requirement(s). For more info, please see: Exporting Cucumber Tests - REST.

VCS workflow (Git as master)

In this scenario, we are managing (i.e. editing) the specification of Cucumber Scenarios/Scenario Outline(s) based tests outside Jira, as detailed in the "VCS workflow" mentioned in Testing in BDD with Gherkin based frameworks (e.g. Cucumber).

The GitLab configuration file `.gitlab-ci.yml` contains the definition of the build steps, including synchronizing the Scenarios/Backgrounds to Xray, extracting the cucumber specification from Xray, running the automated tests and submitting back the results.
.gitlab-ci.yml

```
image: "ruby:2.6"

test:
  script:
  - apt-get update -qq
  - apt-get -y install unzip zip
  - gem install cucumber
  - gem install rspec-expectations
  - 'cd features; zip -R features.zip "*.feature"; cd ..; curl -H "Content-Type: multipart/form-data" -u
$jira_user:$jira_password -F "file=@features/features.zip" "$jira_server_url/rest/raven/1.0/import/feature?
projectKey=CALC" '



  - mkdir -p features

  - 'rm -f features/*.feature'

  - 'curl -u $jira_user:$jira_password "$jira_server_url/rest/raven/1.0/export/test?filter=$filter_id" -o
features/features.zip'
  - unzip -o features/features.zip -d features/
  - cucumber -x -f json -o data.json || true
  - 'curl -H "Content-Type: application/json" -u $jira_user:$jira_password --data @data.json "$jira_server_url
/rest/raven/1.0/import/execution/cucumber"'
  - echo "done"
```

In this example, we're using a variable **filter_id** defined in the CI/CD project level settings in GitLab. This variable contains the *id* of the Jira issues based filter that will be used as source data for generating the Cucumber .feature files; it can be the key(s) of Test Plan(s), Test Execution(s), Test(s), requirement(s). For more info, please see: Exporting Cucumber Tests - REST.