Integration with GitHub

blocked URLblocked URL

GitHub is a well-known platform hosting thousands of source-code repositories.

Besides, it also provides issue tracking and basic project management capabilities.

More recently, GitHub provided the ability to automate workflows using GitHub Actions.

With GitHub Actions, it's possible to implement CI/CD directly in GitHub and reuse already available actions from GitHub Marketplace to automate steps .

An introduction to GitHub actions can be seen here.

Main concepts

In a nutshell, **workflows** are automated processes described as YAML files, stored under *.github/workflows.* These are usually triggered by events (e.g. code-commit, pull-request) or can also be scheduled.

One or more workflows can be defined. Each **workflow** is in turn composed by one or more **jobs**, that can run sequentially or in parallel. A **job** performs a set of sequential **steps** to achieve a certain goal. A **step** is an individual automation task; it can be either an **action** or simply a shell command.

An **action** abstracts some automation task; it can be named and versioned. Actions can be implemented directly in Javascript or as Docker containers. GitHub also supports composite actions built of multiple inner steps.

Actions and workflows can be stored in the local repository; actions can also be published in the GitHub Marketplace.

Each time a workflow is triggered, a **workflow run** is created; it contains a specific context. Each job in the workflow uses a fresh virtual environment (e.g. ubuntu-latest) sharing the same virtual file system.

Accessing and sharing data

A job can generate output variables that can be used by another job that depends on it; this is the preferred way to share data between jobs.

Another way of sharing data, especially between jobs, would be to produce and store artifact(s) in a job and obtain them in another job.

Environment variables can also be used to access some data and share them, with care. Environment variables are available at workflow, job or step level. GitHub fills out some environment variables by default.

It's also possible to access secrets defined in GitHub project settings, as environment variables or as a step input.

Examples

Basic JUnit example

In this basic example showcasing a dummy calculator, we want to get visibility of the automated test results from some tests implemented in Java, using the JUnit framework.

Please note

The source code for this example is available in this GitHub repository.

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));
    }


}
```

To implement the continuous integration, we'll implement a specific *workflow* for it and store it in `.github/workflows/CI-jira-cloud-example-using-action.yaml.`

We’ll use the actions/checkout action to checkout the code from our repository to the virtual environment. This action is one of the "standard" actions provided by GitHub (check full list here).

To compile the code, we need to use a JDK; we can use the action actions/setup-java which allows us to choose the specific Java version.

We use Maven to build and run the tests.

In order to submit those results to Xray, we'll just need to invoke the REST API (as detailed in Import Execution Results - REST); we can do that using an utility (e.g. "curl") or we can use the open-source, community provided, GitHub Action "xray-action" which is easier to use.

The following example uses "xray-action" to submit test automation results to Xray.

.github/workflows/CI-jira-cloud-example-using-action.yaml

```
    name: CI (Jira cloud example with GH action)
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v1
    - name: Set up Java
      uses: actions/setup-java@v1
      with:
        java-version: '1.8'
    - name: Cache Maven packages
      uses: actions/cache@v2
      with:
        path: ~/.m2
        key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
        restore-keys: ${{ runner.os }}-m2
    - name: Build with Maven
      run: mvn clean compile test --file pom.xml
    - name: Submit results to Xray
      uses: mikepenz/xray-action@v2.4.5
      with:
        username: ${{ secrets.client_id }}
        password: ${{ secrets.client_secret }}
        testFormat: "junit"
        testPaths: "**/surefire-reports/*.xml"
        projectKey: "CALC"
```

As mentioned, you could also do it by yourself using "curl", for example, in case you want to.

.github/workflows/CI-jira-cloud-example.yaml

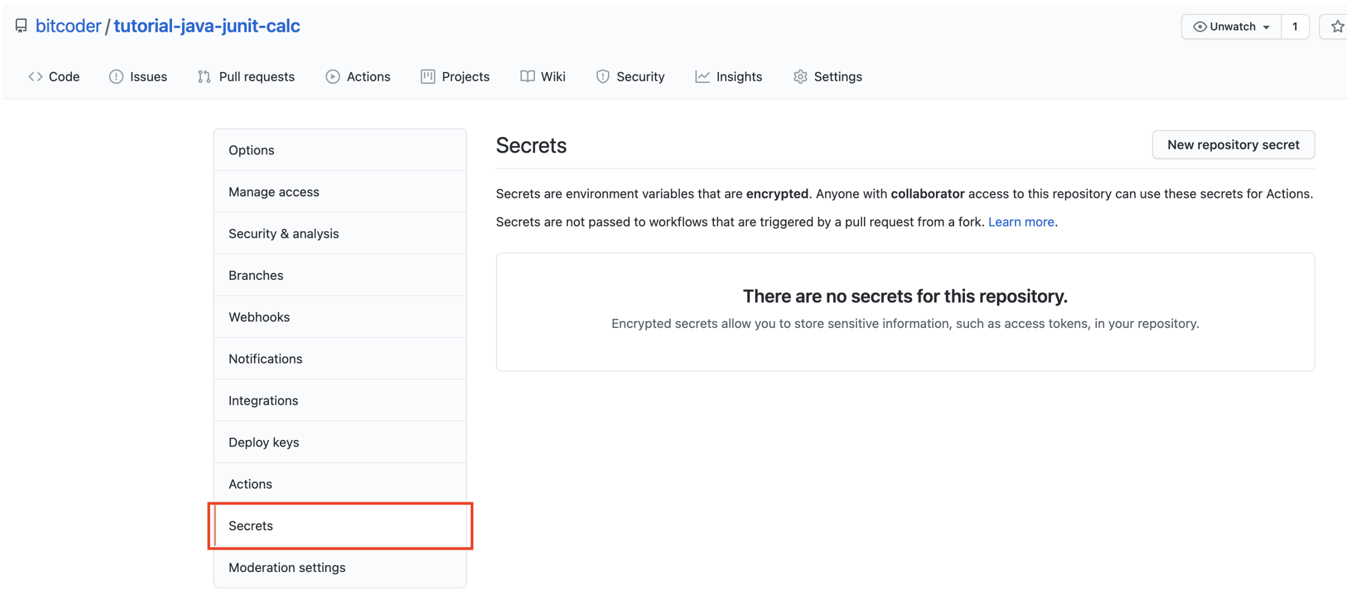
```
    name: CI (Jira cloud example)
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v1
    - name: Set up Java
      uses: actions/setup-java@v1
      with:
        java-version: '1.8'
    - name: Cache Maven packages
      uses: actions/cache@v2
      with:
        path: ~/.m2
        key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
        restore-keys: ${{ runner.os }}-m2
    - name: Build with Maven
      run: mvn clean compile test --file pom.xml
    - name: Get Xray Cloud API token
      env:
        CLIENT_ID: ${{ secrets.client_id }}
        CLIENT_SECRET: ${{ secrets.client_secret }}
      id: xray-token
      run: |
        echo ::set-output name=XRAY_TOKEN::$(curl -H "Content-Type: application/json" -X POST --data "{ \"
client_id\": \"$CLIENT_ID\",\"client_secret\": \"$CLIENT_SECRET\" }" https://xray.cloud.getxray.app/api/v1
/authenticate| tr -d '"')
    - name: Submit results to Xray
      run: 'curl -H "Content-Type: text/xml" -H "Authorization: Bearer ${{ steps.xray-token.outputs.XRAY_TOKEN
}}" --data @target/surefire-reports/TEST-com.xpand.java.CalcTest.xml  "https://xray.cloud.getxray.app/api/v2
/import/execution/junit?projectKey=CALC"'
```

Note that the Xray credentials are not hardcoded in the configuration file. We use some secret variables defined in GitHub project settings.

- **client_id**: the client_id associated with the API key created in the Xray cloud instance
- **client_secret**: the client_secret associated with the API key created in the Xray cloud instance

Please note

The user associated with Xray's API key must have permission to Create Test and Test Execution Issues.



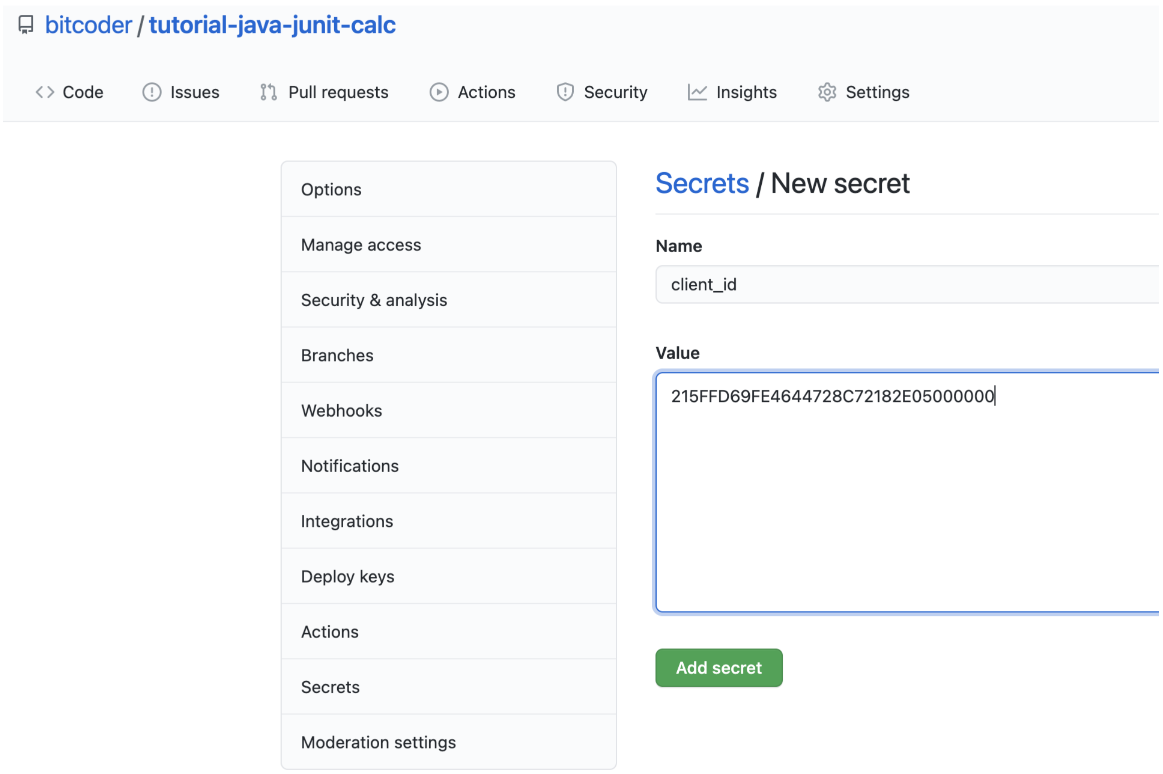
Secrets

Secrets are environment variables that are **encrypted**. Anyone with **collaborator** access to this repository can use these secrets for Actions.

Secrets are not passed to workflows that are triggered by a pull request from a fork. Learn more.

There are no secrets for this repository.

Encrypted secrets allow you to store sensitive information, such as access tokens, in your repository.



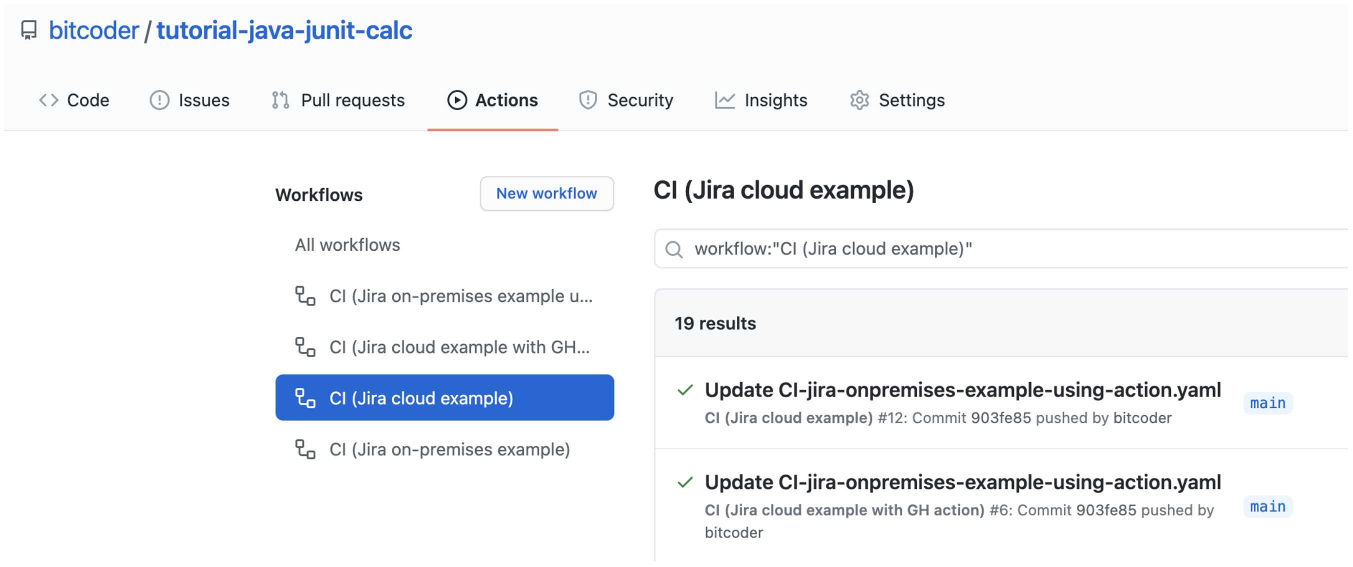
Secrets / New secret

Name: client_id

Value: 215FFD69FE4644728C72182E05000000

Some parameters may be hardcoded on the HTTP request used to submit the result; this is up to you to define what makes sense to be explicit on the request or what could be set, for example, using a secret variable in GitHub.

To see the runs for your workflows (i.e. workflow runs), you may access Actions tab in your repository browser.


bitcoder / tutorial-java-junit-calc
Code
Issues
Pull requests
Actions
Security
Insights
Settings
Workflows
New workflow
All workflows
CI (Jira on-premises example u...
CI (Jira cloud example with GH...
CI (Jira cloud example)
CI (Jira on-premises example)

CI (Jira cloud example)

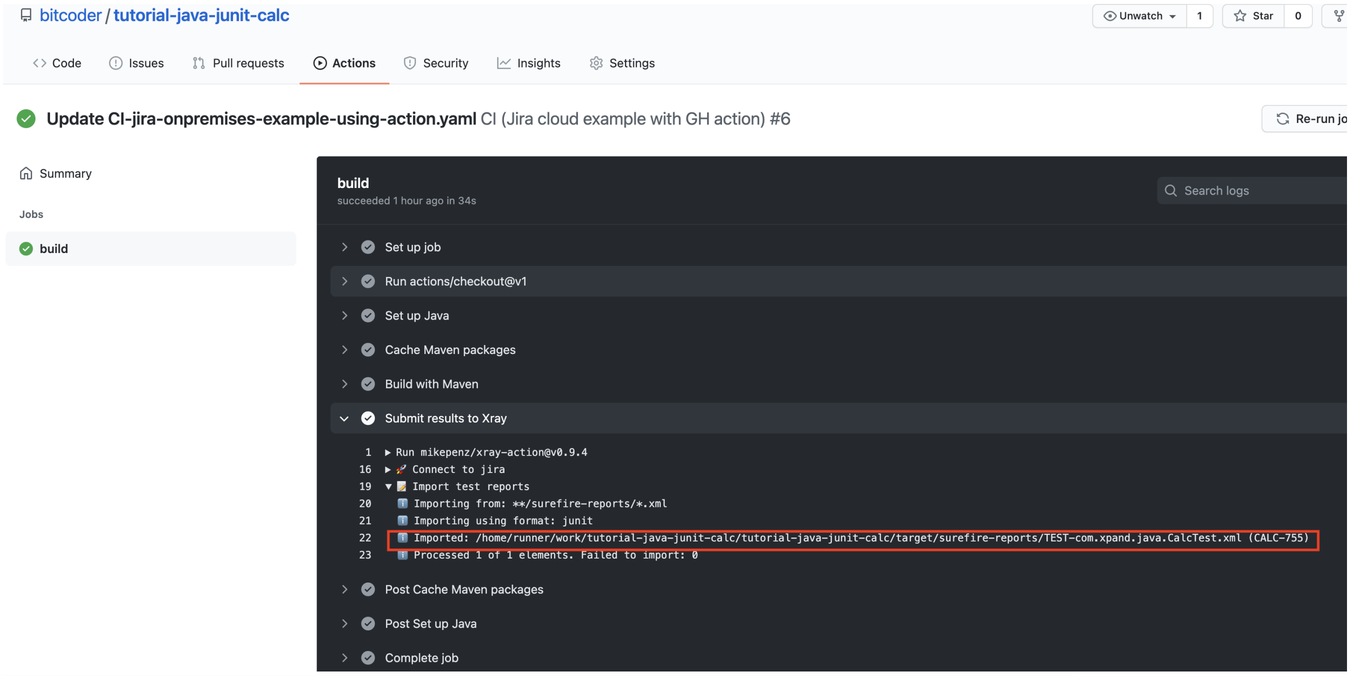
workflow:"CI (Jira cloud example)"

19 results

✓ Update CI-jira-onpremises-example-using-action.yaml
CI (Jira cloud example) #12: Commit 903fe85 pushed by bitcoder
main

✓ Update CI-jira-onpremises-example-using-action.yaml
CI (Jira cloud example with GH action) #6: Commit 903fe85 pushed by bitcoder
main

Clicking in the last event that triggered the workflow run will show the details (screenshots using "xray-action" and using "curl" utility).



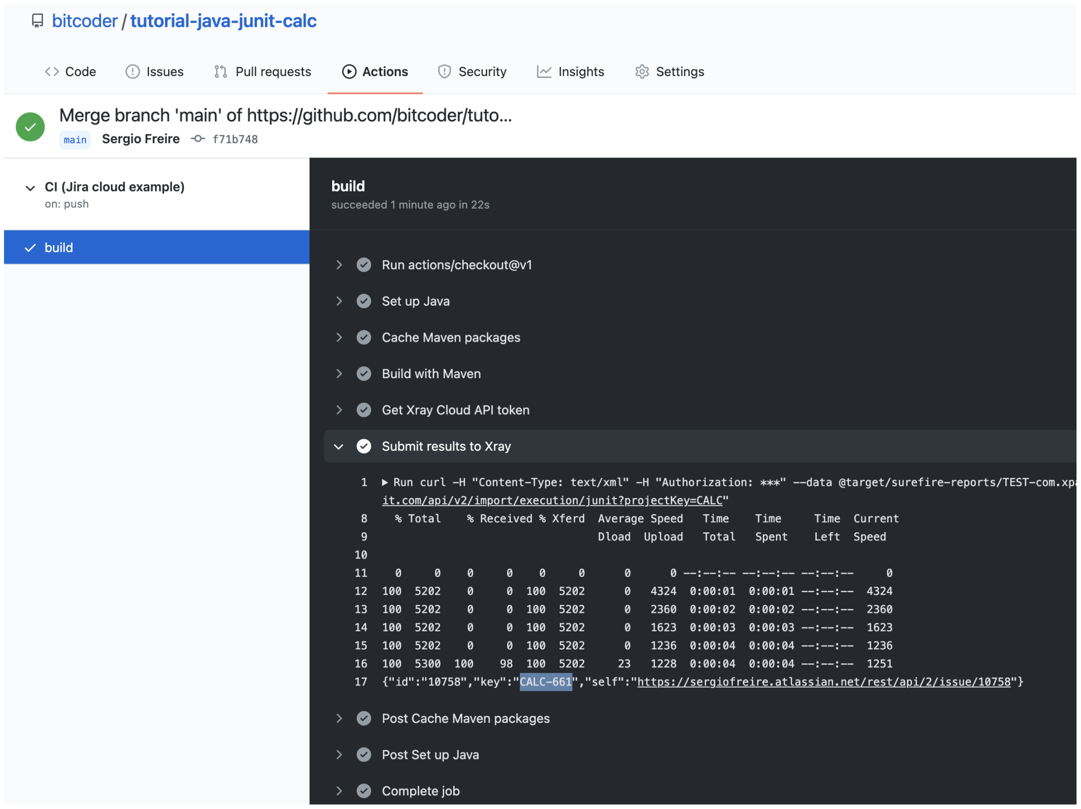
Update CI-jira-onpremises-example-using-action.yaml CI (Jira cloud example with GH action) #6

build
succeeded 1 hour ago in 34s

Set up job
Run actions/checkout@v1
Set up Java
Cache Maven packages
Build with Maven
Submit results to Xray

```
▶ Run mikepenz/xray-action@v0.9.4
▶ 🚀 Connect to jira
▼ 📝 Import test reports
  ℹ️ Importing from: **/surefire-reports/*.xml
  ℹ️ Importing using format: junit
  ℹ️ Imported: /home/runner/work/tutorial-java-junit-calc/tutorial-java-junit-calc/target/surefire-reports/TEST-com.xpand.java.CalcTest.xml (CALC-755)
  ℹ️ Processed 1 of 1 elements. Failed to import: 0
```

Post Cache Maven packages
Post Set up Java
Complete job



Merge branch 'main' of https://github.com/bitcoder/tuto...
main Sergio Freire f71b748

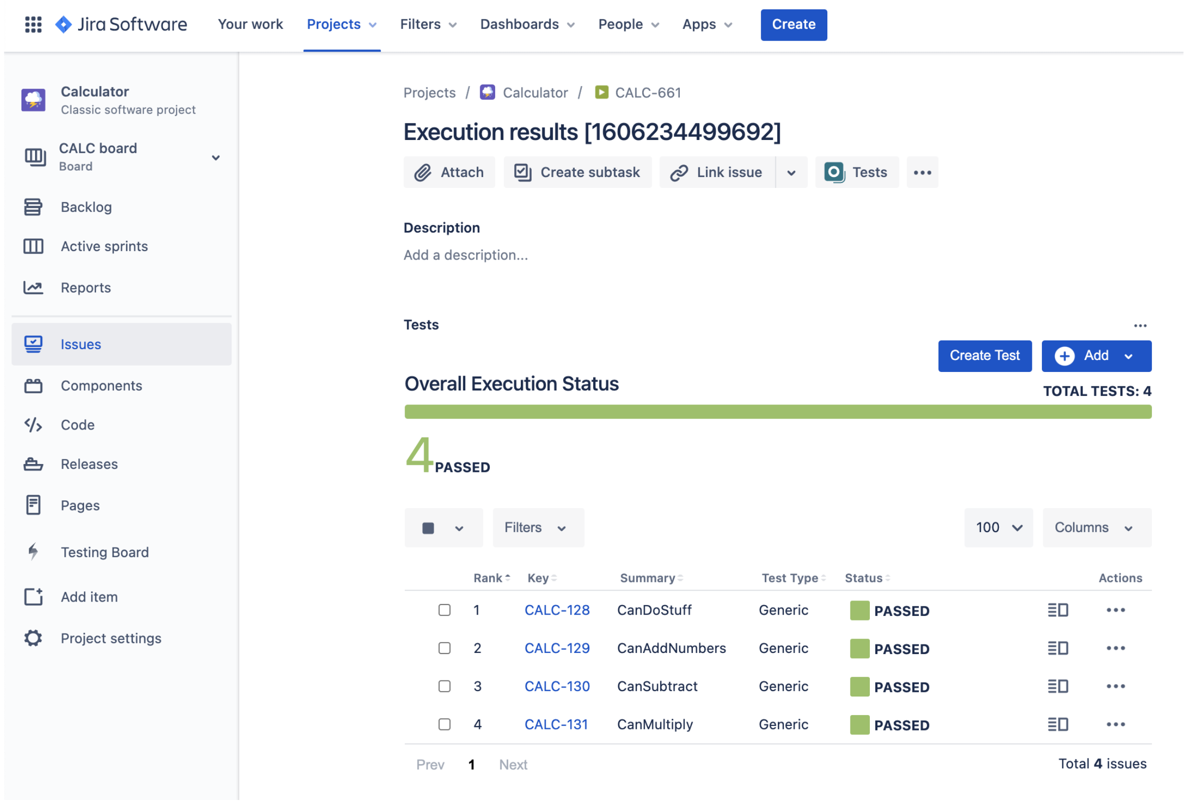
build
succeeded 1 minute ago in 22s

Run actions/checkout@v1
Set up Java
Cache Maven packages
Build with Maven
Get Xray Cloud API token
Submit results to Xray

```
▶ Run curl -H "Content-Type: text/xml" -H "Authorization: ***" --data @target/surefire-reports/TEST-com.xpa
it.com/api/v2/import/execution/junit?projectKey=CALC"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100  5202    0     0  100  5202      0   4324  0:00:01  0:00:01 --:--:--  4324
100  5202    0     0  100  5202      0   2360  0:00:02  0:00:02 --:--:--  2360
100  5202    0     0  100  5202      0   1623  0:00:03  0:00:03 --:--:--  1623
100  5202    0     0  100  5202      0   1236  0:00:04  0:00:04 --:--:--  1236
100  5300  100    98  100  5202     23   1228  0:00:04  0:00:04 --:--:--  1251
{"id":"10758","key":"CALC-661","self":"https://sergiofreire.atlassian.net/rest/api/2/issue/10758"}
```

Post Cache Maven packages
Post Set up Java
Complete job

In Jira, Xray now shows the results of the automated tests in a brand new Test Execution issue. Test issues corresponding to each test method will be auto-provisioned, if they don't exist yet; otherwise, results will be reported against existing Tests.



Projects / Calculator / CALC-661

Execution results [1606234499692]

Description

Add a description...

Tests

Overall Execution Status

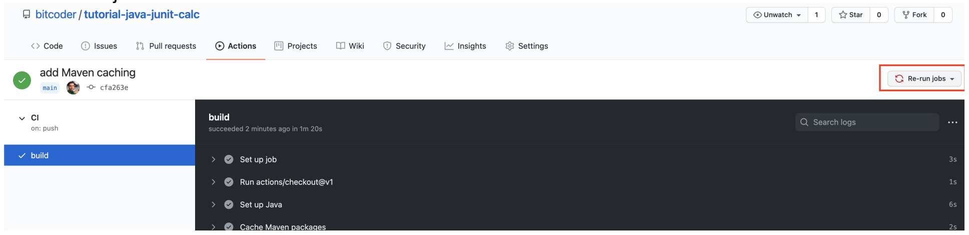
TOTAL TESTS: 4

4 PASSED

Rank	Key	Summary	Test Type	Status	Actions
1	CALC-128	CanDoStuff	Generic	PASSED	
2	CALC-129	CanAddNumbers	Generic	PASSED	
3	CALC-130	CanSubtract	Generic	PASSED	
4	CALC-131	CanMultiply	Generic	PASSED	

Total 4 issues

Tips

- for editing workflow YAML files, you can do it directly from GitHub UI as it provides syntax highlighting, auto-completion, and more
- in the workflow definition, configure it to cache Maven dependencies (more info here)
- it's possible to re-run jobs from GitHub UI
    - 

      add Maven caching
      main · cfa263e

      CI
      on: push
      build

      build
      succeeded 2 minutes ago in 1m 20s
      Set up job 3s
      Run actions/checkout@v1 1s
      Set up Java 6s
      Cache Maven packages 2s
- instead of using `curl` command to interact with Xray REST API, you can abstract it in a GitHub Action and use input parameters to be passed to the REST call

References

- Introduction to GitHub Actions
- Building and testing Java with Maven with GitHub Actions
- GitHub Action to submit results to Xray
- https://docs.github.com/en/free-pro-team@latest/actions/reference/workflow-commands-for-github-actions