Integration with Travis CI

- Overview
- Prerequisites
- Implementing tests
- Integrating with Travis CI
  - Authentication
  - Submit results
- Triggering automation from Xray side
- References

What you'll learn

- How to integrate with Travis CI
- Run the Travis CI pipeline push the test report to Xray
- Validate that the test results are available in Jira

Source-code for this tutorial

- code is available in GitHub

Overview

Travis CI is a continuous integration service used to build and test projects hosted at GitHub, Bitbucket, GitLab and Assembla.

It will automatically detects when a commit has been made and pushed to a repository that is configured with Travis CI, and each time this happens, it will try to execute what is defined in the Travis CI yaml file (build, pack, tests, etc).

Prerequisites

For this example we will use a simple calculator application that is part of the code available and tests defined in JUnit. We will use that repository to start a pipeline in Travis CI, where we will execute the compilation and tests, and report back to Xray.

What you'll need:

- Java and Maven installed
- Travis CI account linked to GitHub (in this case, you can also use Assembla, Bitbucket or GitLab)

Implementing tests

In the repository you will find a calculator application and also tests that are validating those abilities.

The test consists of validating the supported operations (addition, subtraction, division and multiplication) of calculator application.

./test/java/com/xpand/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));
    }


}
```

Once the code is implemented, we can compile and execute the tests with the following command:

```
mvn clean compile test --file pom.xml
```

The results are immediately available in the terminal



In this example, all test have succeed, the output generated to the terminal is the above one and the corresponding JUnit report is below:

Junit Report

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
schemaLocation="https://maven.apache.org/surefire/maven-surefire-plugin/xsd
/surefire-test-report.xsd" name="com.xpand.java.CalcTest" time="0.047"
tests="4" errors="0" skipped="0" failures="0">
  <properties>
    <property name="java.runtime.name" value="Java(TM) SE Runtime
Environment"/>
    <property name="java.vm.version" value="16.0.1+9-24"/>
    <property name="sun.boot.library.path" value="/Library/Java
/JavaVirtualMachines/jdk-16.0.1.jdk/Contents/Home/lib"/>
    <property name="maven.multiModuleProjectDirectory" value="/Users
/cristianocunha/Documents/Projects/tutorials/tutorial-java-junit-Travis CI"
/>
    <property name="java.vm.vendor" value="Oracle Corporation"/>
    <property name="java.vendor.url" value="https://java.oracle.com/"/>
    <property name="guice.disable.misplaced.annotation.check" value="true"
/>
    <property name="path.separator" value=":"/>
    <property name="java.vm.name" value="Java HotSpot(TM) 64-Bit Server VM"
/>
    <property name="user.country" value="PT"/>
    <property name="sun.java.launcher" value="SUN_STANDARD"/>
    <property name="java.vm.specification.name" value="Java Virtual
Machine Specification"/>
    <property name="user.dir" value="/Users/cristianocunha/Documents
/Projects/tutorials/tutorial-java-junit-Travis CI"/>
    <property name="java.vm.compressedOopsMode" value="Zero based"/>
    <property name="java.runtime.version" value="16.0.1+9-24"/>
    <property name="os.arch" value="x86_64"/>
    <property name="java.io.tmpdir" value="/var/folders/s1
/z2z7bgns1_qd21w_t5j33w200000gn/T/"/>
    <property name="line.separator" value="
"/>
    <property name="java.vm.specification.vendor" value="Oracle
Corporation"/>
    <property name="os.name" value="Mac OS X"/>
    <property name="classworlds.conf" value="/opt/apache-maven-3.8.1/bin
/m2.conf"/>
    <property name="sun.jnu.encoding" value="UTF-8"/>
    <property name="java.library.path" value="/Users/cristianocunha/Library
/Java/Extensions:/Library/Java/Extensions:/Network/Library/Java/Extensions:
/System/Library/Java/Extensions:/usr/lib/java:."/>
    <property name="maven.conf" value="/opt/apache-maven-3.8.1/conf"/>
    <property name="jdk.debug" value="release"/>
```

```
    <property name="java.class.version" value="60.0"/>
    <property name="java.specification.name" value="Java Platform API
Specification"/>
    <property name="sun.management.compiler" value="HotSpot 64-Bit Tiered
Compilers"/>
    <property name="os.version" value="10.15.7"/>
    <property name="library.jansi.path" value="/opt/apache-maven-3.8.1/lib
/jansi-native"/>
    <property name="http.nonProxyHosts" value="local|*.local|169.254/16|*.
169.254/16"/>
    <property name="user.home" value="/Users/cristianocunha"/>
    <property name="user.timezone" value="Europe/Lisbon"/>
    <property name="file.encoding" value="UTF-8"/>
    <property name="java.specification.version" value="16"/>
    <property name="user.name" value="cristianocunha"/>
    <property name="java.class.path" value="/opt/apache-maven-3.8.1/boot
/plexus-classworlds-2.6.0.jar"/>
    <property name="java.vm.specification.version" value="16"/>
    <property name="sun.arch.data.model" value="64"/>
    <property name="sun.java.command" value="org.codehaus.plexus.
classworlds.launcher.Launcher clean compile test --file pom.xml"/>
    <property name="java.home" value="/Library/Java/JavaVirtualMachines
/jdk-16.0.1.jdk/Contents/Home"/>
    <property name="user.language" value="en"/>
    <property name="java.specification.vendor" value="Oracle Corporation"/>
    <property name="java.vm.info" value="mixed mode, sharing"/>
    <property name="java.version" value="16.0.1"/>
    <property name="java.vendor" value="Oracle Corporation"/>
    <property name="maven.home" value="/opt/apache-maven-3.8.1"/>
    <property name="file.separator" value="/"/>
    <property name="java.version.date" value="2021-04-20"/>
    <property name="java.vendor.url.bug" value="https://bugreport.java.com
/bugreport/"/>
    <property name="sun.io.unicode.encoding" value="UnicodeBig"/>
    <property name="sun.cpu.endian" value="little"/>
    <property name="socksNonProxyHosts" value="local|*.local|169.254/16|*.
169.254/16"/>
    <property name="ftp.nonProxyHosts" value="local|*.local|169.254/16|*.
169.254/16"/>
  </properties>
  <testcase name="CanDoStuff" classname="com.xpand.java.CalcTest" time="
0.003"/>
  <testcase name="CanAddNumbers" classname="com.xpand.java.CalcTest" time="
0"/>
  <testcase name="CanSubtract" classname="com.xpand.java.CalcTest" time="0"
/>
  <testcase name="CanMultiply" classname="com.xpand.java.CalcTest" time="0"
/>
</testsuite>
```

Notes:

- For more information regarding the execution and integration of JUnit tests with Xray please check this article.

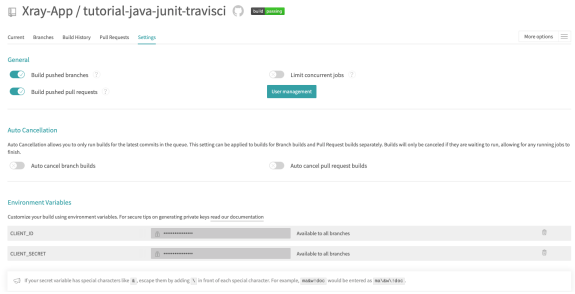
Integrating with Travis CI

As we saw in the previous example , where we are producing a JUnit report locally with the test results; it is now a matter of executing those tests in Travis CI and importing their results to your Jira instance. You can do this by simply submitting automation results to Xray through the REST API or using the Jira interface to do so. In this case we will show how to integrate with Travis CI using API calls.

Please follow the Travis CI documentation to configure your GitHub repo.

In order to integrate Travis CI with Xray, we define the *CLIENT_ID* and *CLIENT_SECRET* as environment variables so they can be later used in the Xray API calls mentioned in the *.travis.yml* file.

For that please go to the configuration of your repo in Travis CI and proceed as follows.



We have also added the necessary *.travis.yml* file to our project in GitHub in order to configure our pipeline.

.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
            export token=$(curl -H "Content-Type: application/json" -X
POST --data "{ \"client_id\": \"$CLIENT_ID\",\"client_secret\":
\"$CLIENT_SECRET\" }" https://xray.cloud.getxray.app/api/v2/authenticate|
tr -d '"')
            echo $token
            curl -H "Content-Type: text/xml" -H "Authorization: Bearer
$token" --data @target/surefire-reports/TEST-com.xpand.java.CalcTest.xml
"https://xray.cloud.getxray.app/api/v2/import/execution/junit?
projectKey=CAL&testPlanKey=CAL-18"
            echo "done"
```

On the file we can see one job definition that is compiling the code (like we did locally):

```
mvn clean compile test --file pom.xml
```

Besides, we can also see two API calls: one for authentication with Xray (with escaping characters) and another to submit the results.

Authentication

```
export token=$(curl -H "Content-Type: application/json" -X POST --data "{
\"client_id\": \"$CLIENT_ID\",\"client_secret\": \"$CLIENT_SECRET\" }"
https://Xray.cloud.getxray.app/api/v2/authenticate| tr -d '"')
```

Here is where we are using the *CLIENT_ID* and *CLIENT_SECRET* (based on an API key pair from your Xray instance) to authenticate with Xray and obtain the token used in the following requests (for more details about this process please check this documentation).

The response of this request will return the token to be used in the subsequent requests for authentication purposes.

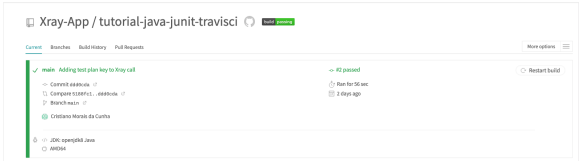
Submit results

```
curl -H "Content-Type: text/xml" -H "Authorization: Bearer $token" --data
@target/surefire-reports/TEST-com.xpand.java.CalcTest.xml  "https://Xray.
cloud.getxray.app/api/v2/import/execution/junit?
projectKey=CAL&testPlanKey=CAL-18"
```

With this command, you will create a new Test Execution for our tests, linked to the referred Test Plan "*CAL-18*". Tests will be autoprovisioned at the moment of the first import of results and will be reused afterward.

Once everything is configured you can start the pipeline by committing to your code repository (in our case pushing code in GitHub).

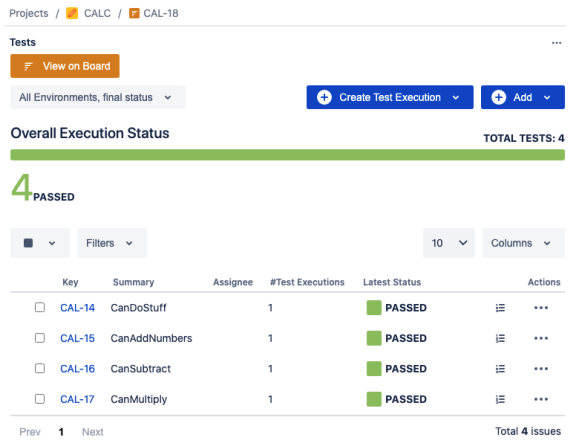
On Travis CI side it will look like this:



The log that will appear in the bottom of the page in Travis CI will output all steps we are taking:

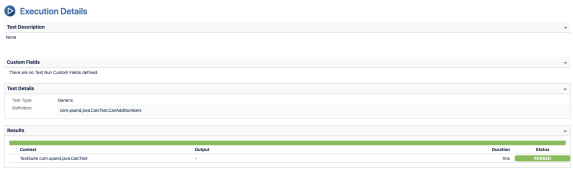


Let's take a look on what it looks like on the Xray side once the test report is ingested back in, namely on the TestPlan "*CAL-18*":



As we can see the test plan have 4 Tests, all currently marked as being passed.

If we drill down to the latest Test Run, we can see the details of that specific execution:



Triggering automation from Xray side

Please have a look at Taking advantage of Jira Cloud built-in automation capabilities to see some examples of how automation can be triggered from Xray side.

References

- https://github.com/Xray-App/tutorial-java-junit-travisci
- https://docs.travis-ci.com/user/tutorial/
- Taking advantage of Jira Cloud built-in automation capabilities

- Overview
- Prerequisites
- Implementing tests
- Integrating with Travis CI
  - Authentication
  - Submit results
- Triggering automation from Xray side
- References