Integration with CircleCI

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

Xray does not provide yet a plugin for CircleCI. However, it is easy to setup CircleCI 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, TestNG or Robot.

We need to setup a project based on a Git repository containing the code along with the configuration for CircleCI 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 CircleCI configuration file `.circleci/config.yml` contains the definition of the build steps, including running the automated tests and submitting the results.

.circleci/config.yml

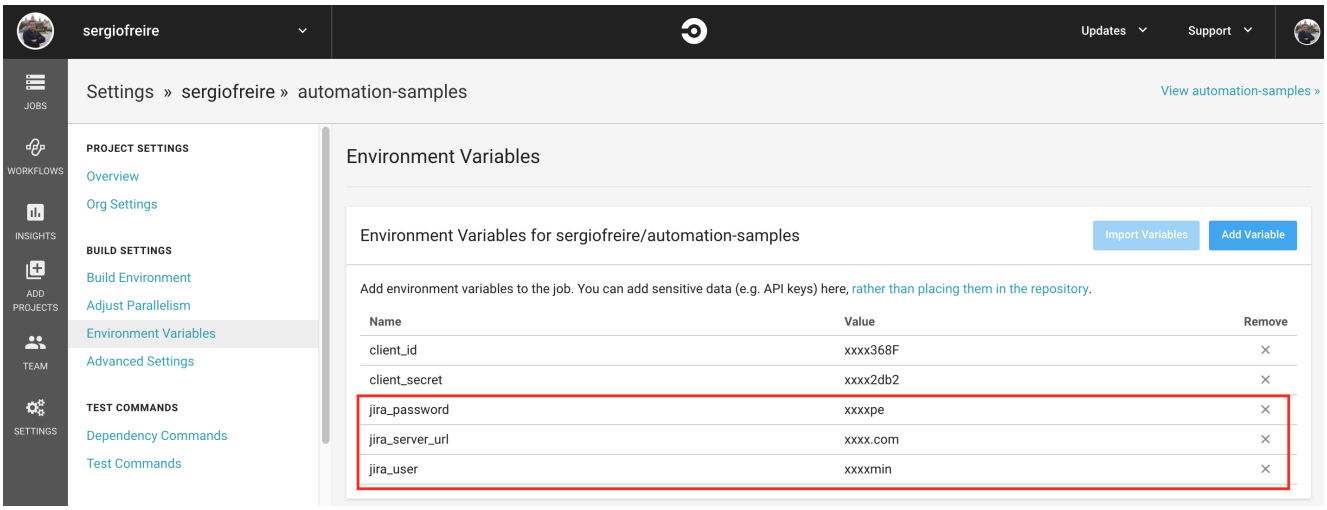
```
version: 2 # use CircleCI 2.0
jobs: # a collection of steps
  build: # runs not using Workflows must have a `build` job as entry point

    working_directory: ~/demo/java-junit-calc # directory where steps will run

    docker: # run the steps with Docker
      - image: circleci/openjdk:8-jdk-browsers # ...with this image as the primary container; this is where all
`steps` will run

    steps: # a collection of executable commands

      - checkout: # check out source code to working directory
          path: ~/demo

      - restore_cache: # restore the saved cache after the first run or if `pom.xml` has changed
          key: circleci-java-junit-calc-demo # circleci-java-junit-calc-demo-{{ checksum "pom.xml" }}

      - run: mvn dependency:go-offline # gets the project dependencies

      - run: mvn test # run the actual tests

      - save_cache: # saves the project dependencies
          paths:
            - ~/.m2
          key: circleci-java-junit-calc-demo # circleci-java-junit-calc-demo-{{ checksum "pom.xml" }}

      - store_test_results: # uploads the test metadata from the `target/surefire-reports` directory so that it
can show up in the CircleCI dashboard.
          path: target/surefire-reports

      - run: '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"'
```

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 credentials hardcoded in CircleCI's configuration file. Therefore, we'll use some environment variables defined in project settings, including:

- **jira_user**: for the Jira username
- **jira_password**: for the Jira user's password
- **jira_server_url**: for the Jira's base URL (e.g. http://yourjiraserver)

> **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 `.circleci/config.yml` a "step" must be included 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 CircleCI.

References

- https://circleci.com/docs/2.0/configuration-reference/