

Integration with Azure DevOps

Azure DevOps (previously known as VSTS) is an ALM solution that provides a set of cloud based tools for helping building and deploy software to the cloud.

Some teams may be using subsets of the Azure DevOps tool portfolio.

In this example we'll explore the scenario where you aim to use Azure DevOps to build and run the automated tests and finally report the results back to Xray.



Please note

Currently, Xray does not provide any plugin for Azure DevOps in order to submit automation related results. However, that can be easily achieved using the [REST API](#), as shown in this tutorial.

- [Integration scenarios](#)
 - [Using Azure DevOps for CI](#)
 - [JUnit example](#)
 - [Using Azure DevOps to store code and Jenkins to manage the builds](#)
 - [JUnit example](#)
- [Triggering automation from Xray side](#)
- [References](#)

Integration scenarios

There are several integration possibilities with Azure DevOps; this documentation depicts some them.

As Azure DevOps is a "full" ALM solution, you can use some parts of it and use other tools to "replace" some built-in features.

Lets explore some integration possibilities.

Using Azure DevOps for CI

This scenario explores usage of the Pipelines facility from Azure DevOps to, similarly to what happens with other well-known CI tools, build the software and run the automated tests.

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.

Within the Repos section of Azure DevOps, we need to setup the code repository containing the code along with the configuration file for Azure Pipelines 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 Azure Pipelines configuration file `azure-pipelines.yml` contains the definition of the build steps, including running the automated tests and submitting the results.

azure-pipelines.yml

```
# Maven
# Build your Java project and run tests with Apache Maven.
# Add steps that analyze code, save build artifacts, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/java

trigger:
- master

pool:
  vmImage: 'Ubuntu-16.04'

steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'java-junit-calc/pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.11'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: false
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    goals: 'clean compile test'
- bash: |

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

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 pipeline's configuration file. Therefore, we'll use some environment variables defined in Pipeline "variables" 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`)
- **project_key**: project key



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.

Azure DevOps

sergiofreire / DEMO2 / Pipelines

Search

DEMO2

Overview Boards Repos Pipelines Builds Releases Library Task groups

YAML Variables Triggers History Save & queue Discard Summary Queue

Pipeline variables

Variable groups

Predefined variables

Name ↑	Value
jira_password	
jira_server_url	https://sandbox.xpand-it.com/
jira_user	
project_key	CALC
system.collectionId	14a7327e-955b-48b8-b789-e785547c47a4
system.definitionId	2
system.teamProject	DEMO2

In `azure-pipelines.yml` a "bash" based 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" "$(jira_server_url)/rest/raven/1.0/import/execution/junit?projectKey=$(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 Azure DevOps.

We may now run the Pipeline build...

Azure DevOps

sergiofreire / DEMO2 / Pipelines / Builds / DEMO2 / #20190418.9

Search

DEMO2

Overview Boards Repos Pipelines Builds Releases Library Task groups Deployment groups Test Plans Artifacts

#20190418.9: Update azure-pipelines.yml for Azure Pipelines

Manually run thu at 18:04 by Sérgio Freire DEMO2 master 22cdc72

Release All logs

Logs Summary Tests

Job Job

Pool: Hosted Ubuntu 1604 · Agent: Hosted Agent

Started: 18/04/2019, 18:07:09 2m 24s

Prepare job · succeeded	<1s
Initialize job · succeeded	1s
Checkout · succeeded	5s
Maven · succeeded	2m 14s
Bash · succeeded	2s
Post-job: Checkout · succeeded	<1s
Finalize Job · succeeded	<1s
Report build status · succeeded	<1s

After results are imported, you can check them in Xray.



Calculator / CALC-1967

Execution results - TEST-com.xpand.java.CalcTest.xml - [1555607373453]

Edit Comment Assign More Close Issue Reopen Issue Admin

Overall Execution Status

4 PASS

Total Tests: 4

FILTERS

Test Set	Assignee	Status	Component	Search
All	All			Contains text Clear

Dates

Created: Just now
Updated: Just now
Resolved: Just now
Begin Date: Just now
End Date: Just now

Agile

[View on Board](#)

Key	Summary	Test Type	#Req	#Def	Test Sets	Assignee	Status
1	CALC-1203 CanDoStuff	Generic	0	0		Xpand IT Admin	PASS
2	CALC-1204 CanMultiply	Generic	0	0		Xpand IT Admin	PASS
3	CALC-1205 CanSubtract	Generic	0	0		Xpand IT Admin	PASS
4	CALC-1202 CanAddNumbers	Generic	1	0		Xpand IT Admin	PASS

Using Azure DevOps to store code and Jenkins to manage the builds

This scenario explores usage of the Repos facility from Azure DevOps to store the code and uses Jenkins as the build tool.

JUnit example

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

The source code is stored and managed by Repos component of Azure DevOps.

Similarly to configuring any SCVS (e.g. Git, SVN, CVS), we need to configure Jenkins to checkout the code from Azure DevOps' Repos.

Jenkins » java-junit-calc_azure-devops »

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Source Code Management

☐ None
☒ Git

Repositories

Repository URL

Credentials Add

Advanced...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

Add Branch

Please check the URL of your repository in Azure DevOps; it should be something similar to:

`https://dev.azure.com/{orgName}/{projectName}/_git/{projectName}`

Next, you can configure the remaining build process, including the build steps, in Jenkins as usual; thus, you can see the results back in Jira and Xray 😊

Jenkins » java-junit-calc_azure-devops »

General Source Code Management Build Triggers Build Environment Build **Post-build Actions**

Xray: Results Import Task

JIRA Instance: xray-vm

Format: JUnit XML

Parameters

Import to Same Test Execution	☒	When this option is check, if you are importing multiple execution report files using a glob expression, the results will be imported to the same Test Execution
Execution Report File (file path with file name)	java-junit-calc/target/surefire-reports/*.xml	
Project Key	CALC	
Test Execution Key		
Test Plan Key		
Test Environments		
Revision		
Fix Version	v3.0	

[Click here for more details](#)

Triggering automation from Xray side

Please have a look at [Integration with Automation for Jira](#) to see some examples of how automation can be triggered from Xray side.

References

- <https://docs.microsoft.com/en-us/azure/devops/index?view=azure-devops&viewFallbackFrom=vsts>
- <https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=schema>
- <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/variables?view=azure-devops&tabs=yaml%2Cbatch>
- <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/test/publish-test-results?view=azure-devops&tabs=yaml>
- <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/integrate-jenkins-pipelines-cicd?view=azure-devops&tabs=yaml>
- <https://docs.microsoft.com/en-us/azure/devops/service-hooks/services/jenkins?view=azure-devops>
- <https://docs.microsoft.com/pt-pt/azure/devops/pipelines/languages/dotnet-core?view=azure-devops#run-your-tests>
- <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/test/vstest?view=azure-devops>