Using batch and PowerShell scripts for Continuous Integration

In Windows scenarios, it is relatively easy to interact with Xray using its REST API.

Both batch or PowerShell scripts can be used in Continuous Integration, for example to submit the test automation results.

This may be quite useful whenever neither Jenkins nor Bamboo are available.

The following are just some examples of scripts that you can use, and customize, to easily submit test results.

For other formats, you should be able to adapt them accordingly.

Please note

Please use the latest version of PowerShell (i.e. >= 6.0) since previous versions had limitations concerning HTTP multipart support.

Examples

Submit results using Xray JSON format

In this example, we'll be making use of Xray's REST API for importing results using Xray's JSON format (more info here: Import Execution Results - REST). There's an initial request to perform the authentication and obtain a token.

It will create a Test Execution, with fixVersion "v1.0", Revision "123" for Test Environment "chrome", assigned to the Test Plan "XT-168".

submit_xray.ps1

```
try {
    $client_id = 'DA2258616A5944198E9BE44A9A000000'
    $client_secret = '5bae1aa5b49e5d263781da54ba55cc7deebd7840c68fe2fdfd2a077768000000'
    $version = 'v1.0'
    $revision = '123'
    $environment = 'chrome'
    $testplan = 'XT-168'


    $jira_base_url = 'https://xray.cloud.getxray.app/api/v2'
    $json = @"
    {
     "client_id": "$($client_id)", "client_secret": "$($client_secret)"
    }
"@

    $uri = "$($jira_base_url)/authenticate"
    $res = Invoke-WebRequest -Uri $uri -Body $json -Method POST -ContentType "application/json"
    $token = $res -replace '"',''
    # write-host $token


    $xrayjson = @"
    {
        "info" : {
            "summary" : "Execution of automated tests for release v3.0",
            "description" : "This execution is automatically created when importing execution results from an
external source",
            "version" : "$($version)",
            "revision" : "$($revision)",
            "testEnvironments": ["$($environment)"],
            "testPlanKey" : "$($testplan)"
        },
        "tests" : [
            {
                "testKey" : "XT-165",
                "start" : "2017-08-30T11:47:35+01:00",
                "finish" : "2017-08-30T11:50:56+01:00",
                "comment" : "Successful execution",
                "status" : "PASS"
            },
            {
                "testKey" : "XT-166",
                "start" : "2017-08-30T11:47:35+01:00",
                "finish" : "2017-08-30T11:50:56+01:00",
                "comment" : "Unsuccessful execution",
                "status" : "FAIL"
            }
         ]
    }
"@



    $uri = "$($jira_base_url)/import/execution"
    $res = Invoke-WebRequest -Uri $uri -UseBasicParsing -ContentType "application/json" -Body $xrayjson -Method
POST -Headers @{"Authorization" = "Bearer $token"}
    write-host $res
   }
   catch {
    write-host $_.Exception.Message
    write-host $_.Exception
   }
```

If you prefer to use a batch file along with some command-line utility (e.g. "curl"), you can also do so.

Sample batch (.bat) script to import results to Xray

```
@echo off
set client_id="DA2258616A5944198E9BE44A9A000000"
set client_secret="5bae1aa5b49e5d263781da54ba55cc7deebd7840c68fe2fdfd2a077768000000"
set report_file="xray.json"

set jira_base_url="https://xray.cloud.getxray.app/api/v2"


for /f %%i in ('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') do set token=%%i
rem echo %token%
curl -H "Content-Type: application/json" -X POST -H "Authorization: Bearer %token%"  --data @"%report_file%" "%
jira_base_url%/import/execution"
```

sample xray.json results file

```
    {
        "info" : {
            "summary" : "Execution of automated tests for release v3.0",
            "description" : "This execution is automatically created when importing execution results from an
external source",
            "version" : "v1.0",
            "revision" : "123",
            "testEnvironments": ["chrome"],
            "testPlanKey" : "XT-168"
        },
        "tests" : [
            {
                "testKey" : "XT-165",
                "start" : "2017-08-30T11:47:35+01:00",
                "finish" : "2017-08-30T11:50:56+01:00",
                "comment" : "Successful execution",
                "status" : "PASS"
            },
            {
                "testKey" : "XT-166",
                "start" : "2017-08-30T11:47:35+01:00",
                "finish" : "2017-08-30T11:50:56+01:00",
                "comment" : "Unsuccessful execution",
                "status" : "FAIL"
            }
         ]
    }
```

Submit JUnit results

In this example, we'll be making use of Xray's REST API for importing results for JUnit (more info here: Import Execution Results - REST). There's an initial request to perform the authentication and obtain a token.

It assumes the existence of a JUnit file named "junit.xml". Don't forget to update the credentials (i.e. client_id and client_secret) and the project key where you want the Test Execution to be created in.

submit_junit.ps1

```
try {
 $client_id = 'DA2258616A5944198E9BE44A9A000000'
 $client_secret = '5bae1aa5b49e5d263781da54ba55cc7deebd7840c68fe2fdfd2a077768000000'
 $project_key = 'XT'
 $report_file = 'junit.xml'

 $jira_base_url = 'https://xray.cloud.getxray.app/api/v2'
 $json = @"
 {
  "client_id": "$($client_id)", "client_secret": "$($client_secret)"
 }
"@

 $uri = "$($jira_base_url)/authenticate"
 $res = Invoke-WebRequest -Uri $uri -Body $json -Method POST -ContentType "application/json"
 $token = $res -replace '"',''
 #write-host $token


 $fileContent = Get-Content -Path $report_file -Raw
 $uri = "$($jira_base_url)/import/execution/junit?projectKey=$($project_key)"
 $res = Invoke-WebRequest -ContentType "text/xml" -Uri $uri -Body $fileContent -Method POST -Headers @
{"Authorization" = "Bearer $token"}
 write-host $res
}
catch {
 write-host $_.Exception.Message
}
```

If you prefer to use a batch file along with some command-line utility (e.g. "curl"), you can also do so.

Sample batch (.bat) script to import results to Xray

```
@echo off
set client_id="DA2258616A5944198E9BE44A9A000000"
set client_secret="5bae1aa5b49e5d263781da54ba55cc7deebd7840c68fe2fdfd2a077768000000"
set project_key="XT"
set report_file="junit.xml"

set jira_base_url="https://xray.cloud.getxray.app/api/v2"


for /f %%i in ('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') do set token=%%i
rem echo %token%
curl -H "Content-Type: text/xml" -X POST -H "Authorization: Bearer %token%"  --data @"%report_file%" "%
jira_base_url%/import/execution/junit?projectKey=%project_key%"
```

Submit JUnit results and customize Test Execution

In this example, we'll be making use of Xray's REST API for importing results for JUnit, namely the multipart endpoint (more info here: Import Execution Results - REST). There's an initial request to perform the authentication and obtain a token.

By using the multipart endpoint for JUnit, we're able to customize fields of the Test Execution that is going to be created in Jira. For that we need to specify a JSON content, either inline or in a file, and submit it in the same request. The format of this JSON object follows the syntax of Jira's own REST API for creating issues.

The example assumes the existence of a JUnit file named "junit.xml". Don't forget to update the credentials and the project key where you want the Test Execution to be created in.

submit_junit_multipart.ps1

```
try {
 $client_id = 'DA2258616A5944198E9BE44A9A000000'
 $client_secret = '5bae1aa5b49e5d263781da54ba55cc7deebd7840c68fe2fdfd2a077768000000'
 $project_key = 'XT'
 $report_file = 'junit.xml'
 $infoFile = "issueFields.json"

 $jira_base_url = 'https://xray.cloud.getxray.app/api/v2'
 $json = @"
 {
  "client_id": "$($client_id)", "client_secret": "$($client_secret)"
 }
"@

 $uri = "$($jira_base_url)/authenticate"
 $res = Invoke-WebRequest -Uri $uri -Body $json -Method POST -ContentType "application/json"
 $token = $res -replace '"',''
 # write-host $token



 $multipartContent = New-Object System.Net.Http.MultipartFormDataContent
 $FsMode = [System.IO.FileMode]::Open
 $FsAccess = [System.IO.FileAccess]::Read
 $FsSharing = [System.IO.FileShare]::Read
 $FileStream = New-Object System.IO.FileStream($report_file, $FsMode, $FsAccess, $FsSharing)

 # additional information for the Test Execution issue; it follows the syntax of Jira REST API for updating
fields
$info = @"
{
    "fields": {
        "summary": "Test Execution for junit Execution",
        "project": {
            "key": "$($project_key)"
        },
        "issuetype": {
            "name": "Test Execution"
        },
        "components" : [
            {
            "name":"ui"
            },
            {
                "name":"core"
            }
        ]
    }
}
"@


$fileHeader = New-Object System.Net.Http.Headers.ContentDispositionHeaderValue("form-data")
$fileHeader.Name = '"info"'
$fileHeader.FileName = $infoFile
$stream = [IO.MemoryStream]::new([Text.Encoding]::UTF8.GetBytes($info))
$infoContent = New-Object System.Net.Http.StreamContent($stream)
$infoContent.Headers.ContentDisposition = $fileHeader
$infoContent.Headers.ContentType = [System.Net.Http.Headers.MediaTypeHeaderValue]::Parse("application/octet-
stream")
$multipartContent.Add($infoContent)


<#
 # in case you want to read  the  Json metainformation from a file instead
 $FsMode = [System.IO.FileMode]::Open
 $FsAccess = [System.IO.FileAccess]::Read
 $FsSharing = [System.IO.FileShare]::Read
```

```
 $FileStream = New-Object System.IO.FileStream($infoFile, $FsMode, $FsAccess, $FsSharing)
 $fileHeader = New-Object System.Net.Http.Headers.ContentDispositionHeaderValue("form-data")
 $fileHeader.Name = "info"
 $fileHeader.FileName = $infoFile
 $fileContent = New-Object System.Net.Http.StreamContent($FileStream)
 $fileContent.Headers.ContentDisposition = $fileHeader
 $fileContent.Headers.ContentType = [System.Net.Http.Headers.MediaTypeHeaderValue]::Parse("application/json")
 $multipartContent.Add($fileContent)
#>


$fileHeader = New-Object System.Net.Http.Headers.ContentDispositionHeaderValue("form-data")
 $fileHeader.Name = '"results"'
 $fileHeader.FileName = $report_file
 $fileContent = New-Object System.Net.Http.StreamContent($FileStream)
 $fileContent.Headers.ContentDisposition = $fileHeader
 $fileContent.Headers.ContentType = [System.Net.Http.Headers.MediaTypeHeaderValue]::Parse("application/xml")
 $multipartContent.Add($fileContent)



 $uri = "$($jira_base_url)/import/execution/junit/multipart"
 $res = Invoke-WebRequest -Uri $uri -UseBasicParsing -Body $multipartContent -Method POST -Headers @
{"Authorization" = "Bearer $token"}
 write-host $res
}
catch {
 write-host $_.Exception.Message
 write-host $_.Exception
}
```

If you prefer to use a batch file along with some command-line utility (e.g. "curl"), you can also do so.

Sample batch (.bat) script to import results to Xray

```
@echo off
set client_id="DA2258616A5944198E9BE44A9A000000"
set client_secret="5bae1aa5b49e5d263781da54ba55cc7deebd7840c68fe2fdfd2a077768000000"
set report_file="junit.xml"
set project_key="XT"
set info_file="issueFields.json"
set jira_base_url="https://xray.cloud.getxray.app/api/v2"


for /f %%i in ('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') do set token=%%i
rem echo %token%
curl -X POST -H "Content-Type: multipart/form-data" -H "Authorization: Bearer %token%" -F info=@"%info_file%" -
F results=@"%report_file%" "%jira_base_url%/import/execution/junit/multipart"
```

sample issueFields.json for "info" object

```
{
    "fields": {
        "summary": "Test Execution for junit Execution",
        "project": {
            "key": "XT"
        },
        "issuetype": {
            "name": "Test Execution"
        },
        "components" : [
            {
            "name":"ui"
            },
            {
                "name":"core"
            }
        ]
    }
}
```

References

- PowerShell
- PowerShell documentation on Invoke-WebRequest
- curl for Windows (there may be another ports)