Integrating Rule Apps into your CI/CD Pipeline

Introduction

One question that is regularly brought up during customer training sessions is how to integrate Rule Applications into a company's existing Continuous Integration and Continuous Delivery or Deployment (CI/CD) pipeline. CI/CD has become standard practice in many organizations because of the opportunities for automation and risk mitigation it provides, and it can be a great idea to include Rule Applications in that organizational standard.

- **Continuous Integration** involves automatically running an application through a series of tests following each code change to ensure that what was checked in did not break any existing functionality. This is a great way to mitigate the risks around applications that are frequently updated.
- **Continuous Delivery** (or Deployment) takes the updated code which was verified in the CI portion of the process and delivers it to the next stage (generally the next environment along the line to Production).

InRule offers a variety of ways to manually perform those tasks (irVerify's Test Scenarios and irCatalog's Rule Promotion), but many customers find that they want to automate portions of the process using functionalities exposed through InRule's irSDK.

This 2-post series will walk through the process of automating CI/CD of Rule Apps using Azure DevOps from a few architectural and functional standpoints, and the accompanying GitHub repo (https://github.com/InRule/DemoRuleCICDPipeline) contains samples that you can use as a reference while building out that automation for your own organization. While these samples use Azure's DevOps (https://dev.azure.com) pipeline, the same concepts will translate to just about any pipeline you may want to use. If you are using Azure DevOps, I found this (https://docs.microsoft.com/en-us/azure/devops/pipelines/process/templates?view=azure-devops) to be an excellent resource.

Part 1: GitHub Push triggered, File-sourced Rule App, irJS compiling pipeline

Scenario:

For the first sample, we're going to use two simple file-based rule applications ("rule apps") that target irJS, each checked into their own folders within a GitHub repository. Each rule app has test suites checked into the application folder, and the repository also contains several helper executable applications. Whenever a rule app is checked in, we want to run any tests in the folder against those rule apps to make sure everything is functioning as expected. If all's well, we will then want to submit the rule app to the irDistribution service to compile it into a JavaScript file, and include that file as a build artifact.

Let's walk through some key aspects of this pipeline:

Trigger from GitHub Check-in

We want our pipeline to run every time someone checks in a Rule App or test scenario, so we'll direct our pipeline to automatically execute whenever a file is changed in a location where those files will be stored.

```
trigger:
  branches:
    include:
    - master
  paths:
    include:
    - "fileSourced-IrJS-TestAndBuild RuleApps*"
```

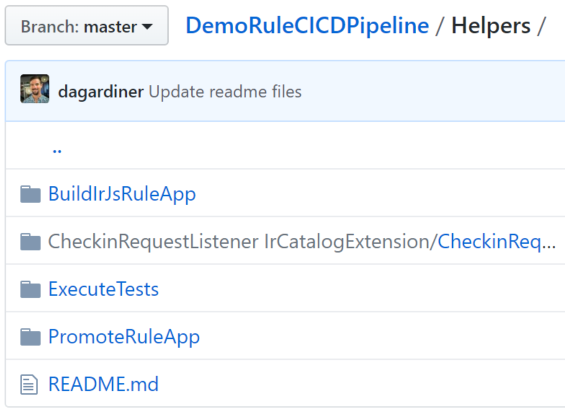
Pipeline Templates

While we could hard-code everything in the pipeline, it's always good practice to make components reusable. To accomplish this, the pipeline is built using templates, with a parameter passed in that allows it to execute on the desired rule app. This requires that a consistent naming structure be used throughout the rule apps in the repository, but it means that adding a new rule app to the pipeline is a trivial exercise. Note that this structure does mean that all jobs will be run when a trigger occurs.

```
jobs:
- template: "fileSourced-IrJS-TestAndBuild.yml"
  parameters:
    RuleAppName: MultiplicationApp
- template: "fileSourced-IrJS-TestAndBuild.yml"
  parameters:
    RuleAppName: AdditionApp
```

Helper Applications

Throughout these pipelines, we're using several different helper console applications that bootstrap irSDK functionality. To make those available to the pipeline, they are included in the GitHub repo in a "Helpers" directory. The three pre-built helpers are compiled from the Samples repository (https://github.com/InRule/Samples/tree/master/Developer%20Samples/CommandLineTools). We'll discuss the IrCatalogExtension in part 2.



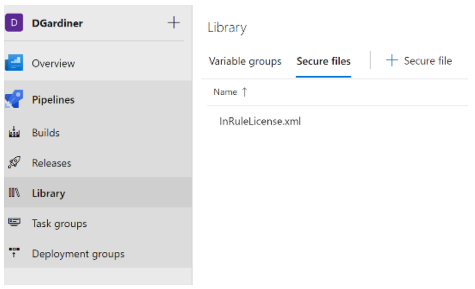
Branch: master / DemoRuleCICDPipeline / Helpers /

dagardiner Update readme files

..
BuildIrJsRuleApp
CheckinRequestListener IrCatalogExtension/CheckinReq...
ExecuteTests
PromoteRuleApp
README.md

Licensing irSDK Helper Applications

Those helper applications require an InRule license in order to execute, and since we are not able to run the InRule Activation Utility on the pipeline host, we'll need to include a license.xml file to allow the helper applications to execute successfully. Since we do not want to store our license in the repository itself, we can instead save it in our Pipeline's Library as a Secure File, copying that into our build Agent's TempDirectory as part of the pipeline. Then, we can copy the testing helper application into the same TempDirectory to be able to execute without issue.

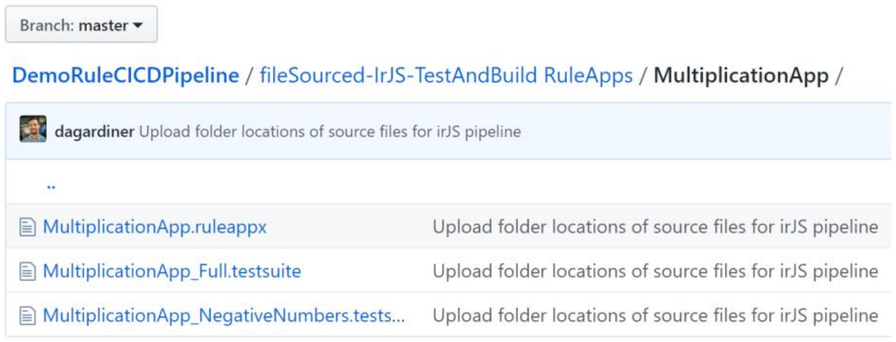


InRuleLicense.xml

```
- task: DownloadSecureFile@1
  inputs:
    secureFile: 'InRuleLicense.xml'
- task: CopyFiles@2
  inputs:
    SourceFolder: '$(System.DefaultWorkingDirectory)/Helpers/ExecuteTests'
    Contents: '**'
    TargetFolder: '$(Agent.TempDirectory)'
    OverWrite: true
```

Run All Test Suites

Once we have all those pieces in place, we're ready to run tests! Again, in keeping with the foundation of reusability, we're going to run a script that iterates through each test suite file in the directory. This script will automatically run all tests without any hard-coded names. As a side note on this, the Test Suite Manager in irSDK has some dependencies on full .NET Framework that prevent it from being compiled as a .NET Core application.



Branch: master

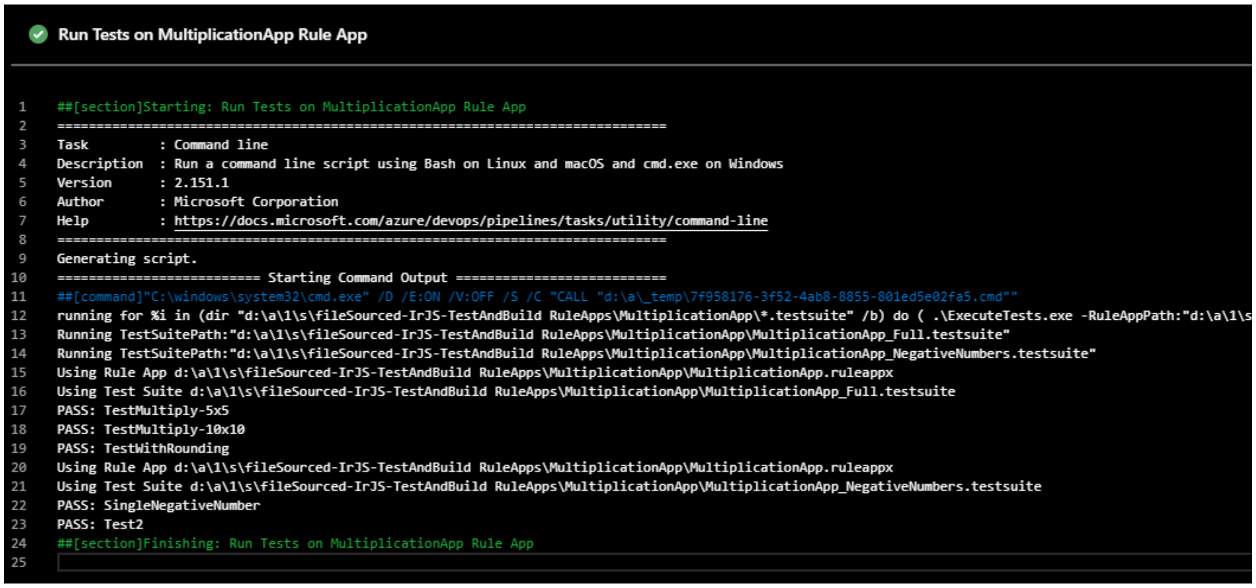
DemoRuleCICDPipeline / fileSourced-IrJS-TestAndBuild RuleApps / MultiplicationApp /

dagardiner Upload folder locations of source files for irJS pipeline

..

MultiplicationApp.ruleappx	Upload folder locations of source files for irJS pipeline
MultiplicationApp_Full.testsuite	Upload folder locations of source files for irJS pipeline
MultiplicationApp_NegativeNumbers.tests...	Upload folder locations of source files for irJS pipeline

```
- script: |
    echo running for %%i in (dir "$(RuleFolder)\*.testsuite" /b) do ( .\ExecuteTests.exe -RuleAppPath:"$(RuleAppLocation)" -TestSuitePath:"%%i" )
    for %%i in ("$(RuleFolder)\*.testsuite") do ( echo Running TestSuitePath:"%%i" )
    for %%i in ("$(RuleFolder)\*.testsuite") do ( .\ExecuteTests.exe -RuleAppPath:"$(RuleAppLocation)" -TestSuitePath:"%%i" )
  displayName: 'Run Tests on $(RuleAppName) Rule App'
  workingDirectory: $(Agent.TempDirectory)
```



Run Tests on MultiplicationApp Rule App

```
##[section]Starting: Run Tests on MultiplicationApp Rule App
==============================================================================
Task         : Command line
Description  : Run a command line script using Bash on Linux and macOS and cmd.exe on Windows
Version      : 2.151.1
Author       : Microsoft Corporation
Help         : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/command-line
==============================================================================
Generating script.
========================== Starting Command Output ===========================
##[command]"C:\windows\system32\cmd.exe" /D /E:ON /V:OFF /S /C "CALL "d:\a\_temp\7f958176-3f52-4ab8-8855-801ed5e02fa5.cmd""
running for %i in (dir "d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\*.testsuite" /b) do ( .\ExecuteTests.exe -RuleAppPath:"d:\a\1\s
Running TestSuitePath:"d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\MultiplicationApp_Full.testsuite"
Running TestSuitePath:"d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\MultiplicationApp_NegativeNumbers.testsuite"
Using Rule App d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\MultiplicationApp.ruleappx
Using Test Suite d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\MultiplicationApp_Full.testsuite
PASS: TestMultiply-5x5
PASS: TestMultiply-10x10
PASS: TestWithRounding
Using Rule App d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\MultiplicationApp.ruleappx
Using Test Suite d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\MultiplicationApp_NegativeNumbers.testsuite
PASS: SingleNegativeNumber
PASS: Test2
##[section]Finishing: Run Tests on MultiplicationApp Rule App
```

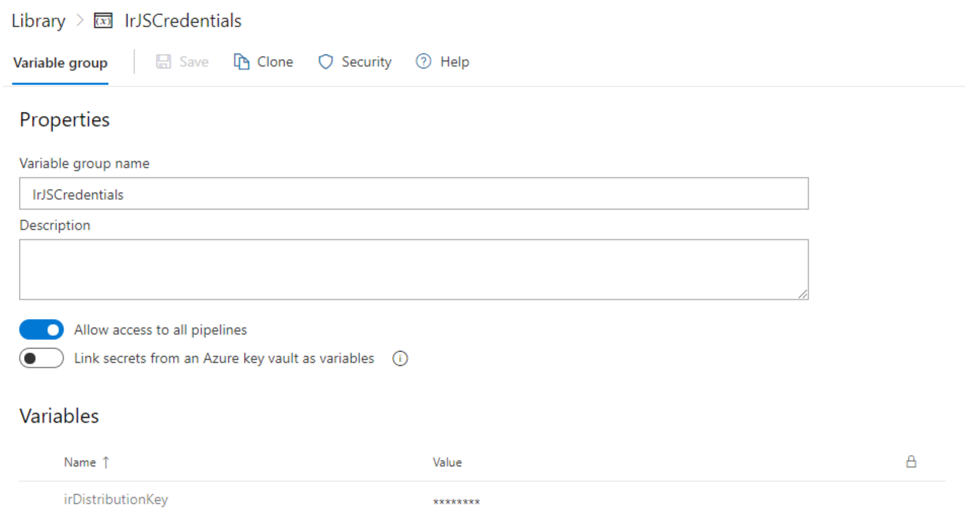
Conditional Next Step

The next task we want to do is compile the rule app using the irDistribution Service, but ONLY if all the tests executed successfully. To do that, we can make the Packaging job dependent on the successful execution of the Testing job.

```
- job: Package_${{ parameters.RuleAppName }}
  dependsOn: Test_${{ parameters.RuleAppName }}
```

Using Secure Credentials in Execution

Calling the irDistribution Service requires an authentication token to be passed in, which (like the license file) should not be stored in the GitHub repository. Because this is a simple text string, this token can be stored in the Library as a Secret variable in a Variable Group and imported into the job.



Properties

Variable group name: IrJSCredentials

Description:

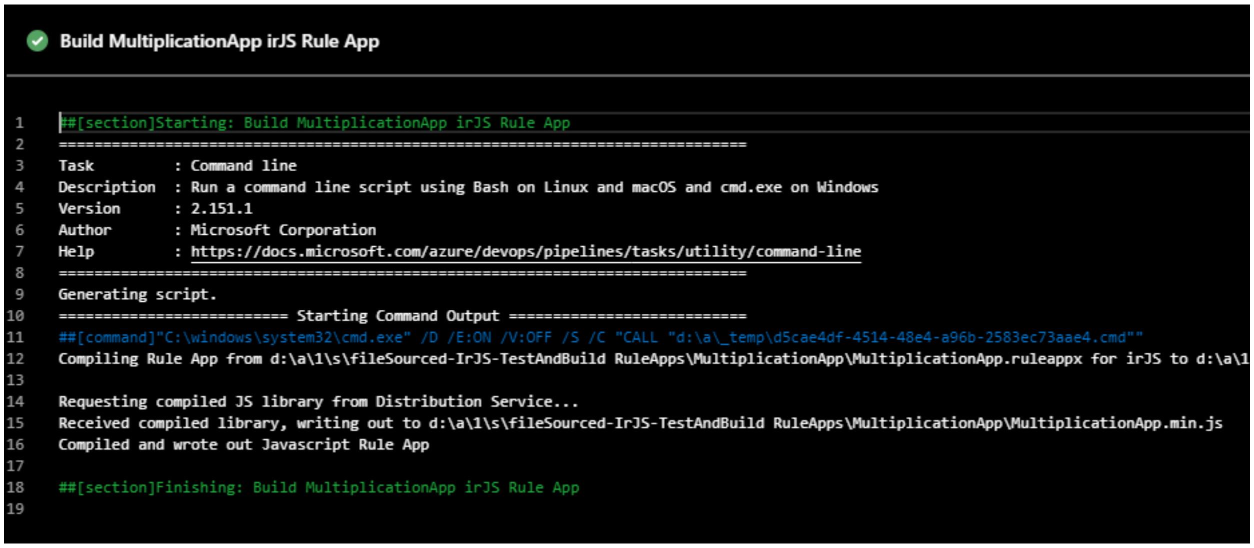
Allow access to all pipelines

Link secrets from an Azure key vault as variables

Variables

Name ↑	Value
irDistributionKey	********

```
variables:
- group: IrJSCredentials
- name: RuleAppName
  value: ${{ parameters.RuleAppName }}
- name: irJsOutputLocation
  value: $(System.DefaultWorkingDirectory)\fileSourced-IrJS-TestAndBuild RuleApps\$(RuleAppName)\$(RuleAppName).min.js
- name: RuleAppLocation
  value: $(System.DefaultWorkingDirectory)\fileSourced-IrJS-TestAndBuild RuleApps\$(RuleAppName)\$(RuleAppName).ruleappx
```



Build MultiplicationApp irJS Rule App

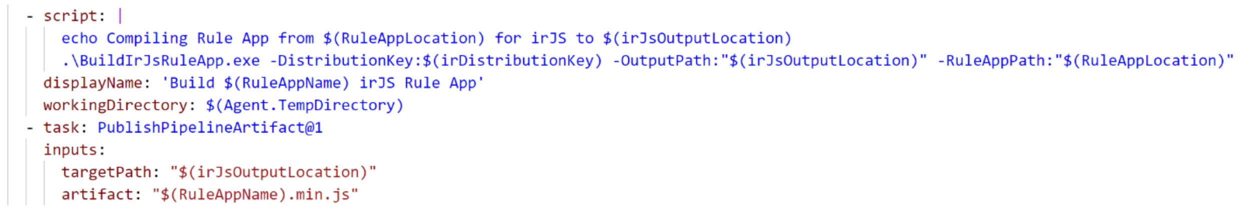
```
##[section]Starting: Build MultiplicationApp irJS Rule App
==============================================================================
Task         : Command line
Description  : Run a command line script using Bash on Linux and macOS and cmd.exe on Windows
Version      : 2.151.1
Author       : Microsoft Corporation
Help         : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/command-line
==============================================================================
Generating script.
========================== Starting Command Output ===========================
##[command]"C:\windows\system32\cmd.exe" /D /E:ON /V:OFF /S /C "CALL "d:\a\_temp\d5cae4df-4514-48e4-a96b-2583ec73aae4.cmd""
Compiling Rule App from d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\MultiplicationApp.ruleappx for irJS to d:\a\1

Requesting compiled JS library from Distribution Service...
Received compiled library, writing out to d:\a\1\s\fileSourced-IrJS-TestAndBuild RuleApps\MultiplicationApp\MultiplicationApp.min.js
Compiled and wrote out Javascript Rule App

##[section]Finishing: Build MultiplicationApp irJS Rule App
```

Creating the Artifact

Once we've copied over the Build IrJS Helper application to the same directory we previously placed the license file, we're then able to execute the wrapper for the irDistribution Service and save the compiled irJS file. That can then be published as a Pipeline Artifact to be used down the line as needed.



```
- script: |
    echo Compiling Rule App from $(RuleAppLocation) for irJS to $(irJsOutputLocation)
    .\BuildIrJsRuleApp.exe -DistributionKey:$(irDistributionKey) -OutputPath:"$(irJsOutputLocation)" -RuleAppPath:"$(RuleAppLocation)"
  displayName: 'Build $(RuleAppName) irJS Rule App'
  workingDirectory: $(Agent.TempDirectory)
- task: PublishPipelineArtifact@1
  inputs:
    targetPath: "$(irJsOutputLocation)"
    artifact: "$(RuleAppName).min.js"
```

We Did It!

The full pipeline and GitHub structure of the fileSourced-IrJS-TestAndBuild-pipeline is available at https://github.com/InRule/DemoRuleCICDPipeline. This is just one example of how these pieces can be put together, but you can certainly adapt the elements to whatever your needs are and build out a CI/CD pipeline that give you exactly the functionality you need in your organization.

Stay tuned for part 2, where we'll cover a pipeline with integrations into irCatalog!