Arcade Helix Job Sender Plan

The Current State of Things in Arcade

Currently, to send a job to Helix using Arcade, developers need to do the following: * Add the `Microsoft.DotNet.Helix.Sdk` package to their `global.json` * Send up `scriptrunner.py` as part of a correlation payload if they want to receive XUnit results * Create an MSBuild proj file which specifies source, type, build number, queue, and appropriate payloads (both correlation & work item). This is capable of handling single-directory wildcards (e.g. zip up all the directories in directory /xyz). * Doing some trickery to output the correlation ID from that project file to an environment variable that can be read by future tasks * Create a custom script file that monitors Helix for work item completion and reports whether tests succeeded or failed. * Add a variable group to their build which contains the appropriate Helix token secret

Then, during their build they must: * Do some work to prep all their test payloads into individual folders. * Call `dotnet msbuild` on their MSBuild proj file (regular MSBuild won't work) * If they want to send the same items to multiple queues, they must repeatedly call the same proj file while changing the `HelixTargetQueue` variable * Trigger their script to wait for Helix to finish

Where We Want to Go

While devs will still need to prepare their tests to send to Helix, we can improve this experience for them. To do this, we will be creating an MSBuild task included in Arcade's `eng/common` directory. This would all be wrapped in a YAML template for easy inclusion in CI builds.

The current thinking for the future process is as follows:

1. Devs take a dependency on Arcade and the `Microsoft.DotNet.Helix.Sdk`
2. Add a variable group to their build which contains the appropriate Helix token secret
3. Devs can choose to either use a YAML template or an MSBuild task (both located in Arcade's `eng/common` directory) to talk to Helix. The YAML template is simply a wrapper for the MSBuild task.
4. If they choose to use the YAML template, they provide the source, type, queues, and payloads as parameters to the template. One of the template parameters would allow for specifying pre- and post-task scripts to run, which would incorporate the functionality of `scriptrunner.py` today.
5. The MSBuild task would support multiqueueing and waiting on multiple jobs.
6. The task would then wait for the job to finish. Optional parameters would be provided for whether failed tests should fail the build or not and whether it should simply fire and forget.

The YAML template is not top priority as devs are familiar with MSBuild

tasks. However, having a template would be nice as it provides consistency with YAML.

Work to be Done

1. Add .zip file payload support to the SDK (assigned to jofortes; estimated 1 hour of work): PR here
2. Add multiqueueing support to the MSBuild task using MSBuild batching (assigned to jofortes; estimated 1 day work): PR here
3. Add multi-job waiting to the SDK and link it to MSBuild task (assigned to jofortes; estimated 2 days work)
4. Add pre- and post-task scripting functionality to the SDK and support XUnit result reporting (assigned to alperovi; estimated 2 days of work): PR here
5. Documentation for use of all this jazz (assigned to jofortes; estimated 1 day of work)
6. (Stretch goal) Add YAML template wrapper to MSBuild task (assigned to jofortes; estimated 1 day of work)

Completion Schedule:

- By end-of-day Fri Sep 14: Items 1 & 2 completed; item 3 in progress
- By end-of-day Wed Sep 19: Items 3 & 4 completed; item 5 in progress
- By end-of-day Fri Sep 21: Item 5 completed; Item 6 hopefully completed; in-progress if not

Was this helpful?