Helix

Helix is a system/service that allows for very broad scaling of workloads and their results in a reliable and low cost fashion. The Azure back-end allows for cloud-level scalability without needing to maintain the underlying infrastructure.

The Helix C# Client and JobSender library can be used to programmatically interact with Helix.

Goals

- System can run any arbitrary work (not just .NET)
- Have a work execution system leveraging the cloud that enables us to deliver our products across all platforms without a dependency on internal Microsoft resources.
- Create a componentized system where each piece could be swapped out for something else
- Each component only requires HTTPS access
- Leverage, as much as is possible, released Microsoft technology

What Helix does for us

- Distributed work execution
  - Helix machines perform arbitrary work on custom-curated, sometimes-scalable machines. When users want to test on other OSes, or run large quantities of work across many machines even from the same OS, Helix enables distributed execution on dozens of different operating systems with work specified in the same, simple JSON-based format for all operating systems. Supported operating systems include macOS, various Windows configurations (Client and server SKUs, non-en-us languages, and variations), Linux systems, with the option to expand to anywhere Python 3 is supported.
- Telemetry
  - When an Engineering-Services-supported Azure DevOps pipeline kicks off a build or test run, telemetry is automated-ly gathered and sent through the Helix pipeline to be stored and reported.
  - This is accomplished by sending telemetry via the Helix API from Azure DevOps pipelines and the various machines performing work.
  - Telemetry is gathered and reported in near-real-time to provide minute-by-minute build and test updates, monitoring things like exit code, console and harness logs, output files, and even crash dumps.
- Helix API

- The Helix API allows users to submit work and telemetry, query previously run work items and jobs for logging and aggregation of results.
- Helix's Client libraries are used by most consumers whether it be for submitting work items, sending telemetry, or waiting on and determining the results of a run.
- Anonymous execution and machines are supported, as well as authenticated work using a GitHub authentication token. Sensitive workloads can be locked down to one or more GitHub user names, which must match to use these queues.
- Users are also able to directly interact with the Helix API via its swagger endpoints in both anonymous and authenticated fashion.

Additional Helix implementation details are available in the private .NET Engineering Services wiki: https://dev.azure.com/dnceng/internal/_wiki/wikis/DNCEng%20Services%20Wiki/606

Features of Helix

Job Distribution at Scale

With Helix we leverage the power of Azure to create and remove VMs as needed. This allows us to create the scale we want at whatever budget we can afford. We also have the ability to add physical machines for other scenarios such as executing work on macOS, ARM, ARM64, dedicated physical hardware for performance, and on virtually any device capable of running Python 3.5 or greater with internet connectivity.

When designing Helix our goal was to leverage the cloud in a way that gave us the reliability we needed, the performance we desired, and simplicity throughout. Azure gives us the ability to have reliable services that we can communicate to over HTTPS.

Cross Platform Support

A major requirement for this system was to be able to execute work across multiple platforms. A wide, constantly-growing variety of operating systems are available, generally removed within a month or two of end-of-life. To see this list live, visit https://helix.dot.net/.

Docker Image Support

For any Helix queue where Docker is installed (viewable via the Queue info API), work items may be executed in a docker image via the syntax `(Fake.Queue.Name)real.queue.name@docker.tag`. This allows running on OS combinations we either can't install on Azure (e.g. Fedora Linuxes) or which only support Docker (Windows Nano, etc.)

Was this helpful?