SAND2020-6634C

Overcoming Productivity Plateaus: A Story of Automation Tools and Developer Productivity

A whitepaper for the 2020 Collegeville Workshop on Scientific Software, focusing on Developer Productivity.

Miranda R. Mundt, Aaron L. Levine, John Siirola, *Sandia National Laboratories*

We've all heard the stories about the great lifts teams make and struggles they endure in order to adopt and implement new processes, particularly those that involve setting up new automation. Far less often do we discuss those teams who have been effectively utilizing tools and processes who are no longer benefiting from the original promise of that automation. These teams frequently are mature and stable, have regular development cycles and processes, and have reached a limitation with their existing toolset. They tend to experience systemic issues with automation tools (such as slowdowns or bottlenecks) and are searching for alternative solutions. These teams have reached a productivity plateau.

Pyomo, one such team, is a Python-based open-source software package that supports a diverse set of optimization capabilities for formulating and analyzing optimization models. The Pyomo software has utilized a large array of Continuous Integration (CI)/Continuous Delivery (CD) (CI/CD) automation tools, including Jenkins, Travis, and Appveyor, to support their extensive software and development testing. Though their variety of CI/CD tools provided Pyomo developers with plenty of differing platforms, Python versions, and optimization solvers with which to test new development work, each of these CI/CD tools has limitations, particularly in max concurrent runs. These limitations were directly contributing to Pyomo's productivity plateau.

Pyomo regularly has 10+ active pull requests. With this level of traffic, the testing infrastructure was causing noticeable bottlenecks and slowdowns in developer productivity. In November 2019, GitHub released their own CI/CD tool: GitHub Actions. One month after its release, Pyomo conducted a development summit in which contributors focused on improving Pyomo's overall performance. One such effort was to explore GitHub Actions in hopes of gaining a development productivity boost.

This whitepaper will discuss the particular Pyomo productivity plateau and the efforts to achieve accelerated test results through the use of GitHub Actions.

Existing Tools

Pyomo has avidly adopted automation tools. As of November 2019, Pyomo utilized Travis, Appveyor, and Jenkins for automated testing. All three utilities would kick off a series of tests whenever a pull request was created on Pyomo's GitHub repository. Appveyor was used to test the Windows 10, Travis to test Ubuntu 16.04, and Jenkins to test Red Hat Enterprise Linux (RHEL). The specific stengths and weaknesses are in the table below.

Strengths	Weaknesses
Cost	Simultaneous Job Limit
Maturity	Development Bottlenecks
Reliable Images	Limited OS Support
Optimization Solvers	Technical Debt
Variable Python Versions (2.7, 3.4+, PyPy)	32-bit Limitations
Visual Results Review	

The biggest strength was the cost: free. Travis and Appveyor provide basic services at no cost. They additionally have paid-service options available, which provide a greater number of concurrent runs and features, depending on the tier.

All of the existing services had another strength on their side: maturity. They were well-established, well-maintained, and well-documented, which made implementation and maintenance much easier. Partially as a result of maturity of the services, the images provided were reliable. Travis tests were implemented on a custom Docker image that allowed a pre-built environment. Jenkins tests were implemented on a self-hosted machine in a tightly controlled and administered environment.

All services also allowed testing of a matrix of Python versions. Since Pyomo is released and supported for many versions and stresses the importance of backwards compatibility, it was esssential that each was regularly tested. On top of that, Pyomo has several optimization solver interfaces (such as GAMS and IPOPT). The automation tools were configured to include numerous optimization solvers to ensure regular testing of those interfaces. As a final note, all three automation tools provided a GUI interface on GitHub from which to conduct a visual review of the results.

The weaknesses, however, imposed some difficult limitations. First and foremost, Pyomo was constantly adversely affected by the simultaneous job limit imposed by the no-cost versions of their automation tools. As part of the free service, Appveyor offers one job to run at a time and Travis allows up to five concurrent jobs. Jenkins allows only one pull request at a time, that is, all tests within a certain pull request must finish before a new pull request test can begin. The trade-off for the minimal cost to implement the automation resulted in systemic delays in development productivity.

In addition, an entire operating system was untested. Travis and Jenkins both tested Linux distributions, and Appveyor covered Windows. There was a gap surrounding MacOS. To introduce this new OS with the existing tools would only exacerbate the already existing bottleneck. The currently covered operating systems had their flaws as well, particularly Windows. The version that Appveyor offered was a 32-bit version, which caused issues with newer versions of optimization solvers.

Finally, there was a significant amount of technical debt associated with maintaining three different automation tools. Every tool has its own vernacular, interface, and capabilities, and as such, each one required its own expert. Since these experts doubled as regular Pyomo developers, this took time away from their ability to develop productively.

Design Considerations

The initial design objective was simple: overcome the productivity plateau. With an average turnaround time of four to six hours for a single pull request, any faster solution would improve the productivity of the developers. That being said, we had to carefully evaluate the solution space to truly meet the needs of the developers.

The first requirement was a given: improve productivity. The feedback on active development needed to come more quickly, thus enabling the developers to continue developing rather than waiting for tests to complete. It was not good enough for the tests to just be faster; however, they needed to be consistently faster, as in, multiple pull requests would not cause the bottlenecks that were currently being experienced.

Additionally, speed could not come at the cost of coverage. Testing coverage overall could not be lower than the 70% range provided by the average between the three existing tools. The tests also needed to function in the same process as the existing tools - that is, they needed to initiate after every submittal of or change to a pull request.

Implementation Considerations and Limitations

With the design consideration in mind, we now needed to address implementation considerations. GitHub Actions for public repositories is a free service (there is a tier-graded payment plan for private repositories) and can run 20 concurrent jobs with 3 different GitHub-hosted operating systems available (Windows, Ubuntu, MacOS). With more possible jobs and operating systems available at no cost, this would result in a definite improvement upon previous testing tools and would reduce bottlenecks, as well as address the limited OS support.

Next we needed to be wary of our limitations. A looming risk was the ongoing evolution of the technology. GitHub Actions is constantly evolving, meaning frequent changes that might cause spurious failures in test suites. In addition to this, we had very little control over the changes to GitHub Actions runners, making them less reliable than our previous images.

The most significant limitation of GitHub Actions at this time is the inability to rerun a single job. Travis and Appveyor both allow for reruns of a single job, rather than a whole suite, in the event of a failure. GitHub Actions, however, does not provide this capability. An entire suite must be rerun, which could possibly introduce the exact same bottleneck we were desperate to fix.

Implementation

With the requirements, considerations, and limitations in mind, we moved onto the implementation. For our initial testing purposes, we decided to leverage the existing workflows for Travis and Appveyor. Since the Travis jobs were built on an Ubuntu operating system, we created a basic Ubuntu workflow with four Python versions, mimicked after the Travis workflow. As it turns out, the logic between tools was almost seamless, requiring only a few syntax changes. This effort resulted in four concurrently running Ubuntu jobs, all of which finished after 7 minutes and followed Travis logic.

Simultaneously, we decided to do the same thing from Appveyor to GitHub Actions. The logic again was very similar between the two tools, though GitHub Actions had far more idiosyncracies for Windows. Ultimately our result was four concurrently running Windows jobs, all of which finished after 13 minutes and followed Appveyor logic.

Next we expanded our efforts to cover MacOS, which had previously never been tested. This was a larger challenge since there was no existing workflow. Since MacOS and Linux distributions generally use similar logic and commands, we started with the Ubuntu GitHub Actions workflow and built from there. Consequently, we found a corner-case of tests that were guaranteed to fail on MacOS without core code changes to file structure assumptions. As an end result, the four concurrently running tests all finished after 11 minutes, while leveraging the same workflow logic as Ubuntu.

By the end of our implementation phase, we had three workflow files: one for Ubuntu 20.04, Windows 10, and MacOS 10.15, which, if triggered at the same time, collectively finished in 13 minutes. Because they were built upon work from existing tools, rather than having to implement brand new logic and steps, the transition time took only a few weeks, and as developer Carl Laird noted, “I largely haven’t had to notice anything about [the transition] whatsoever. It just worked. It was seamless.”

Implementation Expansion

Though the original implementation provided results well higher than our initial expectations, it was inelegant and did not match all of the capabilities of the other automation tools. Additionally, we wanted to utilize GitHub Actions to automate other processes and provide branch testing in addition to pull request testing.

Merge All Workflows into a Single Driver Script: Maintaining three separate workflow files (i.e., driver scripts) was a technical debt we wished to reduce. GitHub Actions supports `bash` and `powershell` scripting on all three platforms. This capability made it possible for all three driver scripts to be merged into a single, universal master script. This reduces the maintenance cost and provides a consistency in appearance across all three operating systems.

Local Branch Testing: Because GitHub Actions workflow files are not restricted to being triggered on pull requests, we were able to introduce branch and fork testing. This testing is triggered whenever a commit is pushed to any branch besides master on that main Pyomo repository or any branch on a developer's personal fork. For developers this means that they can take advantage of all the benefits of the pull request testing without having to open a pull request ans save computational cycles on the main Pyomo repository for completed development.

Release Process Wheel Creation Automation: GitHub Actions workflow files can also be triggered for other events, such as new releases. Pyomo has been manually creating wheels for distribution to PyPi and the Anaconda Cloud. As of version 5.7, we implemented an automated wheel creation workflow, which triggers as soon as a new release tag is pushed to the Pyomo repository. Because of this automation, Pyomo is able to create and release cythonized distributions of Pyomo, which provide a performance boost for users, with no extra effort from the developers.

Statement of Impact

This effort utilized existing work and technologies to deliver a new, productivity-enhancing solution to Pyomo's productivity plateau. Rather than stagnating on development work while waiting for pending results, developers can devote more time and effort to actively developing. In further detail:

Reduction in testing time: After the expansions, the Ubuntu/MacOS jobs take about 15 minutes, and the Windows jobs take about 25 minutes. That's a 3.33x speed-up for Ubuntu (previously 50 minute average) and a 3.6x speed-up for Windows (previously 90 minute average). In a testimonial from developer John Siirola, "One of the biggest thing is that the jobs run in about the same time as the old Travis jobs, but because we get 20 jobs instead of 5, GitHub Actions keeps up with the pace of development. When we had fairly active development in the old system, there was a good chance that you had to wait hours to get a set of test results back. Now you rarely have to wait more than 20 minutes." In a followup comment from developer Emma Johnson, "I push a change, go get a cup of tea, and see results when I get back. I don't have to switch gears and work on another project and then have to spin myself back up to figure out what I was doing."

Increased operating system coverage: MacOS was not regularly tested before GitHub Actions was implemented. Because of the addition of this operating system, Pyomo was able to resolve an entire corner-case of tests that failed due to misassumptions in file structure. Now with regular testing, this extension of coverage allows developers to find potential flaws on MacOS before releasing, thus preemptively saving bug-fix time on the backend. In a testimonial from developer Bethany Nicholson, "It's given us the ability to explicitly test on Mac, which we didn't have previously. We just assumed that if it worked on Linux, it would work on Mac, but having those [tests] allowed us to catch issues that we previously hadn't discovered."

Local testing: The ability to locally test against the same testing suite frees up compute cycles on the main Pyomo repository, meaning that unfinished work is not creating a bottleneck for work that's ready for review. In a testimonial from contributor David Bernal, "In my own fork of Pyomo, I have all of the GitHub Actions testing suite available. . . . With GitHub Actions, anytime I make a single change, I make sure that all the tests pass before I make a big PR to Pyomo itself. Decentralizing testing has immensely helped."

Single driver script: Having a single driver script that controls the three GitHub Actions workflows reduces maintenance. In a testimonial from John Siirola, "Having a single driver has been wonderful. The install process is not exactly the same - the Windows side uses Anaconda and the Unix side uses Pip, so we're catching oddities in our package distribution channels, which is a good thing. . . . Having one script to rule them all, and different jobs pick different paths, is a lifesaver in terms of maintenance."

Retirement of underperforming automation tools: As a result of GitHub Actions' performance, Pyomo was able to completely retire the use of Appveyor for its automated testing. This results in an equal amount of maintenance (as there are still three unique automation tools utilized). In a testimonial from developer Carl Laird, "I was able to delete my Appveyor account!"

Conclusion

Even mature projects that currently utilize advanced tools and automation techniques can reach a productivity plateau. With new tools, features, and capabilities continuously being released, it is important and significant to revisit and reevaluate decisions and workflows to see if there is room for improvement. This allows both the project and team to mature in terms of productivity.

Pyomo was a team that benefited from exactly this effort. Rather than allow current automation tools to impede developer productivity, we explored the new tool (GitHub Actions) and implemented it into our current workflow. We saved 70% extra time in pull request testing by using this new tool, freed up computational resources for the main Pyomo branch by implementing local branch testing, retired an existing tool that had been usurped by GitHub Actions' superior performance, and for the first time, added regular MacOS testing.

In the future, we are interested in exploring the capabilities of GitHub Actions further with the hope of fully automating the Pyomo release process. We will avoid the productivity plateau by regularly revisiting our workflows and tools usage to make sure they continue to fit our needs and to find new opportunities for improvement.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.