OPTIMIZING CI/CD IMPLEMENTATIONS

ActiveState

Executive Summary

High volume CI/CD is a high pressure/high stakes process that can lead to DevOps burnout. Three of the key issues that can throw a monkey wrench into any CI/CD pipeline are:

Reproducibility
Ensuring that all systems are consistent throughout the CI/CD chain, as well as development and production.

Transparency
Understanding the original source for all artifacts throughout the chain improves both security and compliance of production workloads.

Speed
"A hallmark of modern CI is spending 10 minutes to build a Docker image to run a process for 5s.[1]"

This paper proposes the ActiveState Platform as a solution to help DevOps:

Eliminate the "works on my machine" issue by providing a consistent, reproducible runtime environment that can be deployed to all systems with a single command.

Solve open source supply chain issues by delivering transparency for all language artifacts in production workloads.

Speed up CI/CD runs by using a pre-built runtime environment.

The result is a more secure and compliant CI/CD pipeline that eliminates the complexity and overhead imposed by many CI/CD systems when using custom runtimes. All of which means a reduction in pipeline maintenance time, which can enable more deliveries of code with greater confidence.

[1] https://twitter.com/indygreg/status/1231008674090344449

THE STATE OF CI/CD

Continuous Integration/Continuous Deployment (CI/CD) is a best practice that allows for better synchronization between development teams, who typically check in code frequently, and operations teams, who typically deploy code much less frequently in order to maximize application stability. CI/CD enables standard, consistent, automated testing and deployment so teams can deliver updates to production with confidence more frequently.

However, for most organizations, a fully automated CI/CD implementation remains aspirational. ActiveState recently undertook a survey involving organizations of every size that have deployed a CI/CD solution. Our findings show that less than 40% of the more than one thousand respondents surveyed have implemented a majority of CI/CD best practices to date.

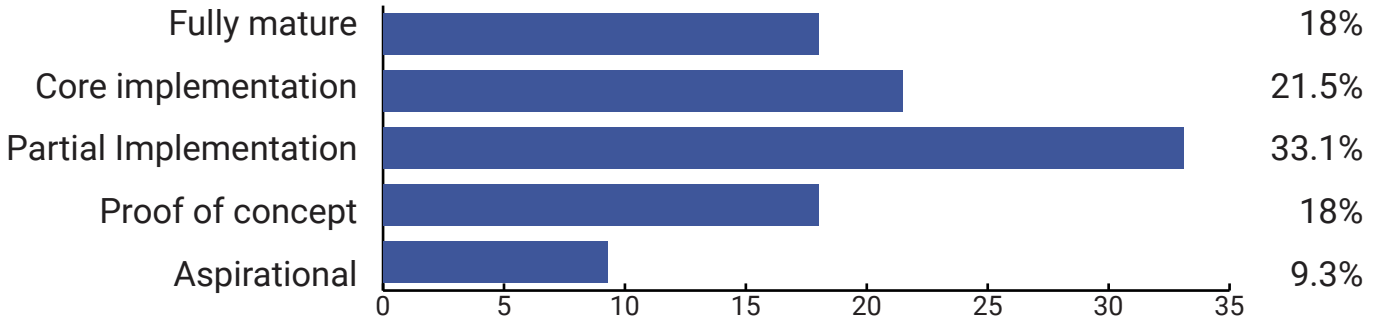


Figure 1: What best describes your team's CI/CD practice?

One of the key reasons holding organizations back from fully implementing CI/CD is the significant difference between continuous deployment and continuous delivery. While all organizations can benefit from continuous delivery to non-production systems, most organizations are reluctant to automatically deploy to production every code change that successfully passes the CI/CD pipeline. Frequent application updates can be disruptive to users, and put pressure on documentation, marketing and operations teams, in addition to DevOps. In fact, at least one survey has shown that modern teams who implement automated production deployment can be highly stressed:

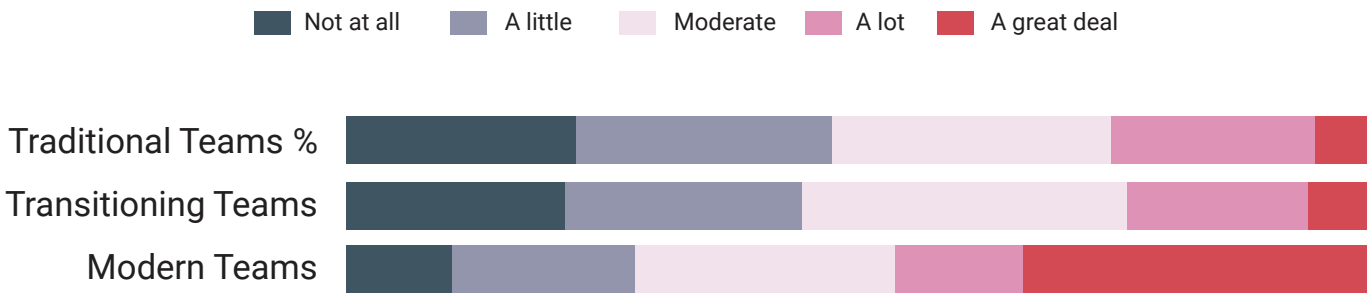


Figure 2: Team stress levels [2]

ActiveState's own survey also showed that only a minority of respondents (~10%) are extremely satisfied with their current CI/CD deployment. This level of dissatisfaction undoubtedly contributes to the stress teams are feeling.

[2] https://dzone.com/articles/devops-teams-are-stressed-and-dissatisfied-with-th

CI/CD CHALLENGES

The reason for this level of stress and dissatisfaction may be attributable to a number of common but critical challenges our respondents are experiencing with their language runtime environments, including:

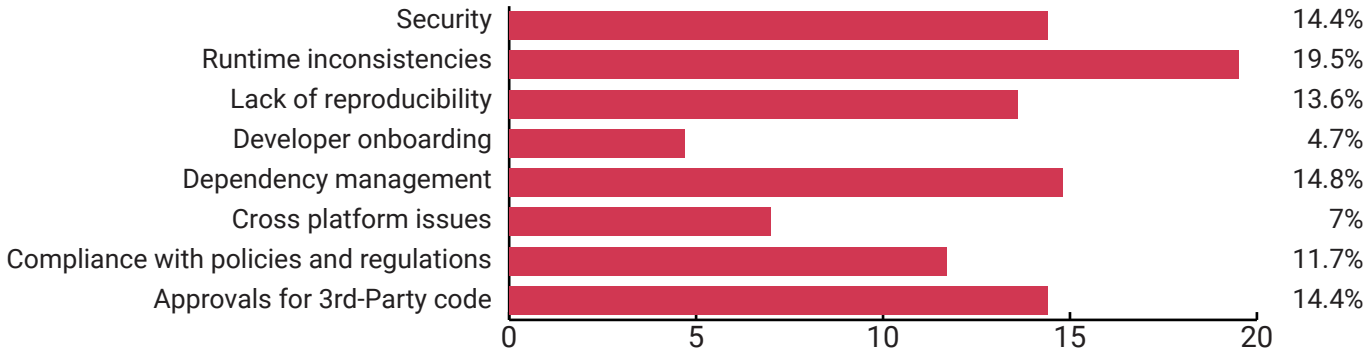


Figure 3: What are your top 3 challenges with managing language dependencies and runtimes?

Runtime Inconsistencies & Dependency Management

A runtime environment consists of a version of a language, as well as all the packages/dependencies required to run the application. Inconsistencies can occur when developers check in code that was developed on systems with differing development environments (the dreaded “works on my machine” error), as well as when the runtime environment of a CI/CD system differs from development or even production environments.

Runtime inconsistencies was the #1 reported challenge by our survey respondents, despite the fact that the majority of them use Docker or a similar container solution in their processes. Containers are designed to be ephemeral, built for a specific purpose and discarded when that purpose has been served. But if the container is not built in a consistent manner, with the exact same manifest every time, discrepancies can arise. Common runtime environment inconsistencies can crop up both within and between development systems and CI/CD systems for a number of reasons, including:

- Some (but not all) developers including the latest version of a package to work around a bug or vulnerability
- Unpinned dependencies pulling in the latest version of a package, rather than the required one
- Differing OS-level dependencies
- Poorly named vendored dependencies (ie., dependencies you've modified or created) conflicting with similarly named public packages
- And so on

The ActiveState Survey also highlights a number of drawbacks with CI/CD systems, including:

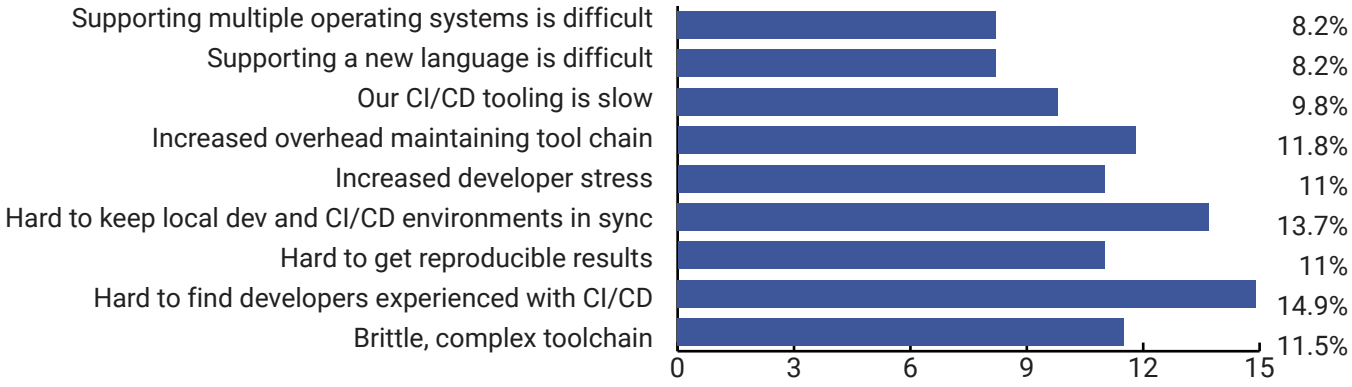


Figure 4: Which major drawbacks of CI/CD has your organization experienced?

Synchronizing Dev & CI/CD Systems

Similar to the issue of inconsistencies, the #2 drawback our survey respondents noted was the difficulty in keeping local development and CI/CD systems in sync. Over time, these systems tend to diverge, again, despite the widespread use of containers.

One of the key reasons for this is that developers rarely rebuild/reinstall their development environment from scratch if they need to update a single package or OS-level dependency. Over time, this can result in significant differences both within and between developer and CI/CD systems.

Unfortunately, developers rarely think about how easy it might be to reproduce an environment in order to test their code. As a result, when something fails during CI, developers may be forced to spend time reproducing the CI environment where the defect was detected. But building that environment can often take longer than fixing the defect.

Again, the ActiveState Platform can help. When a project's runtime environment is automatically updated and built on the ActiveState Platform, it can then be propagated to every system with a single command. In this way, developer and CI/CD systems can always be kept in sync, and developers can spend their time fixing defects rather than building environments.

Toolchain Maintenance & Multi-OS/Multi-Language Support

Finally, the #3 drawback with CI/CD cited by our survey respondents was increased overhead due to maintaining multiple toolchains, one for each OS and/or language.

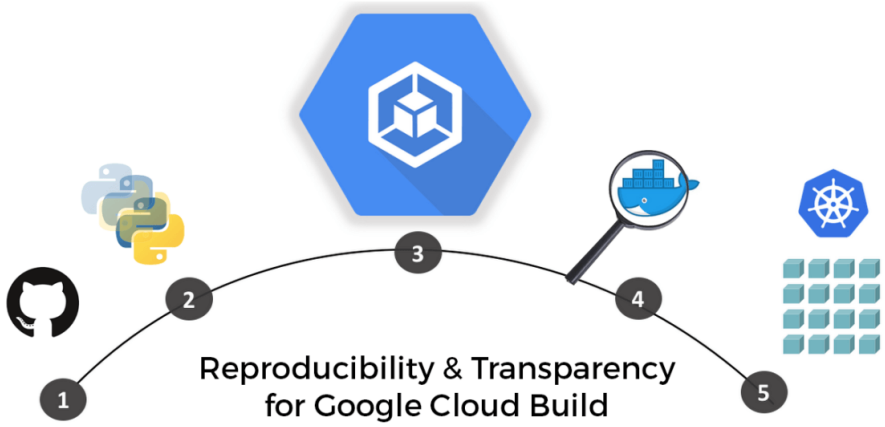
Most modern development teams work with their OS of choice, which means that organizations must support Linux, Windows and Mac systems. Further, while they may develop on the latest version of Ubuntu, they may deploy on a previous version of Redhat Enterprise Linux (RHEL). Such a scenario would entail creating and updating a build environment for four different target OSes. With four different toolchains for a single project, teams are spending more time on tool maintenance and less time coding. And if your tech stack includes more than one language, maintenance overhead increases dramatically.

The ActiveState Platform can help here, as well. It provides a single, cloud-based toolchain for all popular OSes (Linux, Windows and Mac), as well as support for both Python and Perl, and soon Ruby. As a result, you no longer need to maintain your own build environments, and can leverage a single, universal package management solution (ActiveState's State Tool) to empower all your teams.

HOW ACTIVESTATE CAN HELP OPTIMIZE CI/CD

There are a number of challenges with CI/CD systems that prevent organizations from fully adopting, implementing and deriving greater value from them. While the ActiveState Platform is not a silver bullet for solving all CI/CD challenges, it can help you optimize your existing implementation.

The ActiveState Platform can be integrated with all of the most popular CI/CD platforms quickly and easily. In this model, your preferred Version Control System (VCS) acts as the source of truth for your project's code, and the ActiveState Platform acts as the source of truth for the runtime environment, which includes a language and all the packages your project requires. Your CI/CD solution will grab the runtime from the ActiveState Platform and the source code from your VCS, and then build and run its tests. This process is valid for both on premise solutions, like Jenkins, as well as cloud-based solutions like GitHub Actions, Google Cloud Build, Azure Pipelines, etc.



Incorporating the ActiveState Platform into your CI/CD processes offers a number of advantages, including:

Reproducibility - eliminate “works on my machine” issues and simplify the troubleshooting of bugs. ActiveState’s single, central source of truth for your runtime ensures development environments and CI/CD environments are always in sync.

Speed - pre-built runtime environments decrease the time to build containers. Of course, caching can help cut down on runtime creation times for repeated runs, but not when you’re doing rapid development and changing your dependencies.

Simplicity - using non-standard runtimes with cloud-based CI/CD solutions can be complex. But by prebuilding your custom runtime environment on the ActiveState Platform, you can simply pull it into your cloud-based CI/CD chain.

And finally, because the ActiveState Platform builds all the components in a runtime environment from source code, you can improve both the security and compliance of your production workloads.

CONCLUSIONS

Just as a code repository can simplify the creation of complex software, a runtime management solution like the ActiveState Platform can simplify and streamline CI/CD by acting as the single source of truth for the pre-built runtime environment used by all the systems across development and the CI/CD chain.

The ActiveState Platform provides a single, cloud-based toolchain that developers can use on-demand to automatically build a custom Python, Perl or Tcl runtime environment from source for Windows, Mac or Linux systems. The ActiveState Platform catalog includes tens of thousands of Python, Perl and Tcl packages pulled from their respective open source repositories (as well as other definitive sources) on a regular basis to ensure you can build your runtimes with the latest versions of your dependencies.

By using the ActiveState Platform organizations can:

Build language runtimes from source quickly and easily, thereby deriving the benefits of open source provenance without having to maintain their own build infrastructure.

Reproduce development environments with a single command, thereby streamlining onboarding of new developers, simplifying maintenance of older projects, and eliminating “works on my machine” issues.

Simplify and speed up CI/CD runs by using a prebuilt runtime environment.

For more information, please visit

www.activestate.com/ci-cd-resources