White Paper

Modern DevOps for Low-Code

Increasing productivity for the development team

By Derek Shue

Table of Contents

Low-Code and DevOps

A great developer will deliver their code quickly, to specification, and do so consistently. They will maintain their team's best practices, test often and sometimes offer a novel solution. Developers must also perform their duties while keeping in mind the full stack of dependent technologies. Front-end, back-end, and middle integration layers form a web of interconnections that become difficult to decipher as the complexity of the solution increases over time.

The answer to the increasing difficulty of this scenario is the low-code platform. According to Malcom Ross, the Vice President of Product at Appian, the low-code platform is the way to "turn your business's intentions into an application very quickly."[1] A low-code platform should inherently deliver security, scalability, and collaboration. Therefore, low-code represents the transition from the traditional paradigm that the developer is solely responsible for their solution to the notion that the platform must make the solution manageable so that the developer can maintain productivity.

Furthermore, low-code platforms should be compatible with development methodologies that are considerate of end-user expectations, such as Agile. The platform must be easily portable and maintainable to the development team in order to achieve its potential. A developer's code must be easy to build, test, and deploy, and it shouldn't be difficult for the developer to manage these operations. To fulfill the full potential of a development team, an appropriate Development and Operations (DevOps) stack must be used to support Continuous Integration (CI) and Continuous Deployment (CD).

The DevOps Stack

The DevOps stack is a set of technologies that form a pipeline for a development team. The general DevOps pipeline consists of several steps in a loop: plan, develop, integrate, deploy, and monitor. This pipeline exists beside the Agile framework to enforce the Agile principles of iteration and end-user feedback. Many different technologies fill the gaps in this pipeline to orchestrate the interconnected web of software development.

Plan: An example stack would be using JIRA to plan a set of users stories for the next project sprint.

Develop: Once the tickets are assigned, the designers would develop and test in their project code base technology.

Integrate: When a set of user stories is finished they are registered into a Git or SVN repository where the changes are integrated.

Deploy: At the end of a sprint, a set of automated tests will be applied against the stories. Any defects should be triaged appropriately and then the code is deployed up the stack to higher environments.

Monitor: System administrators are alerted when a production issue occurs. A system may require audits on user access or system performance. In the case of Appian, a combination of built-in system logs and health reports helps facilitate this phase of the pipeline.

Continuous Integration

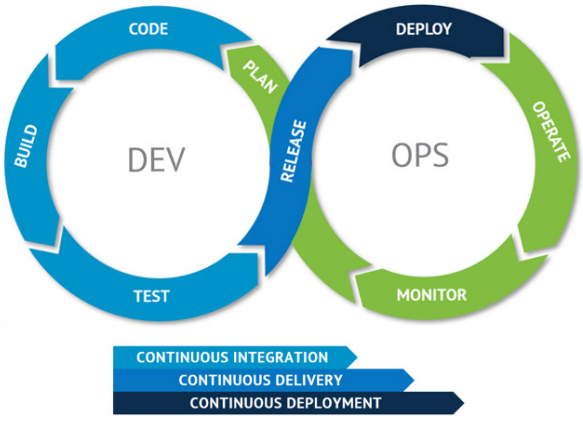
Continuous Integration (CI) is the foundation for securing one developer's changes against the many versioning and configuration conflicts that may occur between development and deployment.[2] It means that each change to code or configuration can be monitored, reviewed, and reverted, if necessary. In traditional programming this often means that each developer needs to be an expert in the repository of their choice to manage their code. Often a single mis-step from a developer can cause undesired merge conflicts and throw the code base into disarray. A high-maintenance of the code base simply does not work well with the concept of low-code and Agile.

Appian leads the vision of how continuous integration works with a low-code platform. The platform allows auditable versioning of each element that a developer works on, including business rules, user interfaces, APIs, and workflows. Furthermore, when concurrent development on shared resources is performed, Appian informs the developer of potential versioning issues and alerts them that another user is editing the resource. However, interconnections with repositories, software management, and automated build utilities are required to address concurrent builds.

Continuous Integration in Action

Consider the scenario of two development teams working on the same code base for a financial firm. One team may be performing an enhancement stream while a completely separate team is developing bug fixes. In this scenario, the bug-fix stream may be considered a critical path. Since the business is a financial entity, it is of the utmost important to end-users that funds are protected from incorrect transactions. In this case, the bug fixes should be expedited, quickly tested, and deployed to all environments.

However, the enhancement development stream may be more thorough and methodical. The users may be waiting for the deployment of their new work-stream into the platform, but the business has aligned training such that the go-live is on an expected date a couple months out.



The code repository, whether the technology is Git, SVN, Azure, or another solution, is key to addressing these multiple streams. In the traditional model, each developer would take out their own branch from the repository before starting their work. Then, they would prepare their code changes and attempt to merge them into a common branch, before the code is pushed to another environment. Eventually this requires a pull request, to manually compare the differences. This may result in days of work to do proper merges and testing to produce a release candidate that considers the proper changes from both streams.

Solving Continuous Integration for Low-Code

The low-code DevOps stack seeks to optimize this process. Any release could introduce sweeping changes to interfaces and bug fixes that can be critical to keeping the data secure. The modern development team must rise above the difficulties of maintaining synchronous code bases and low-code platforms offer several features to ease this process.

As previously mentioned, the Appian platform maintains versioning of internal objects. Additionally, in the admin suite of the platform, the system administrators have a set of DevOps tools to ease the burden on developers. One of these tools allows each connected Appian environment in the stack, generally Dev, Test, UAT and Production, to make direct comparisons between application containers to identify the code discrepancies.

With a single button push the developer can view the line-by-line differences in each user interface configuration. This feature is inherent to the product and doesn't require an outside repository to function making it a great foundation for starting the DevOps process. Applied to our example above, the development team could utilize an additional Appian environment as a pre-production merging area for code. The team can quickly build and test a release candidate exactly as it will be released to production with the additional confidence that they have seen the user interfaces working already.

However, the built-in tools of a low-code platform may not consider all the interconnections and database configurations that exist outside of the domain of the platform. To further refine the DevOps stack it becomes necessary to integrate external repository systems such as Git or SVN to help integrate and build the project as a whole. It is recommended that the team use separate branches for Test, UAT, and Production builds of the code. By organizing the code base into separate branches for each environment we are able to construct a pipeline where each branch represents a more refined application version. Automated testing tools such as Selenium can provide quick verification on UAT and production branches to ensure that regression issues are not present in the release candidate.

Appian Versioning Tool

The Appian platform offers a free community tool called the Automated Deployment Plug-in to facilitate integrations with external repositories. The Appian Versioning Tool allows any developer to easily merge their code into a code repository. The team may use multiple branch management strategies such as story-specific branches or sprint-specific branches.

In the case of a story-specific branch, each team member will be responsible for checking out a new branch for each story they do in a sprint. Then at the end of the sprint, all of the story branches will be merged into the UAT branch. If the team opts to use a sprint branch model then each developer will deploy their changes to the sprint branch directly. In this case, the versioning tool will only need to run on the newest changes that have been applied to the environment. The start and end-hash parameters of the script allow the developers to define this window appropriately.

Automated Testing

A critical feature of the build phase is verifying the correctness of your application. Although automated tests should not be considered as a replacement for functional testing resources, each project should maintain a suite of automated tests that are managed by the development team. Selenium-based solutions such as Cucumber and FitNesse have helped ease the process of developing these scripts for developers by making easy-to-understand commands that pair well with low-code platforms.

Sometimes a project may be significantly progressed when the need for more mature DevOps practices becomes a priority. When building automated tests in a significantly mature project, a team may struggle to decide which features should be considered in their testing suite. In this case the team should focus on high-level features instead of edge cases. By focusing on high-level features, the team gets immediate value when the test can easily prove whether or not a build has critically failed. During future iterations, the team will be able to consider stories on a sprint-by-sprint basis and register them in their own repository.

The automated testing suite should act as a barrier to the UAT and Production branches of the repository. The release management software should automate the build by testing the release before it continues to the deployment phase. In the case of a failure, the team will realize that a regression is apparent in their build and they will need to fix the issue before any pull requests are merged from their sprint branch.

Continuous Delivery / Development

Continuous Delivery/Deployment (CD) represents the deploy step in the DevOps stack, after the application has been built. Frequent deliveries must be made to end-users so that the value of the changes can be fully realized.

[2] The main difference between continuous delivery and deployment is that continuous delivery features a manual deployment of the code whereas continuous deployment is fully automated. It is recommended that a team beginning their DevOps pipeline should begin with a continuous delivery model first to enhance confidence and refine procedures before they are ready to automate the entire process. Jenkins and Bamboo are examples of release management platforms that ensure software builds are deployed on a frequent schedule. Generally, the build phase will provide a set of artifacts that can be used by the release management suite such as application objects, test results and environment configurations. This software can be optimized to perform automated deployments as soon as a package passes the automated test, or on a timed schedule so that maintenance periods can be communicated ahead of time to end-users.

Appian Automated Deployments

Appian provides another resource that is packaged alongside the Versioning Tool, the Appian Deployment Manager. This tool is composed of two elements: the deployment script and the automated deployment servlet. The script is responsible for taking the build artifact from the integration phase and deploying it to the servlet. The team should be careful to protect their credentials since the script will need to deploy as an administrator account. It is recommended that the deployment mechanism use encrypted credentials, such as, what is available in the Jenkins platform, to manage their accessibility.

Monitoring

Monitoring represents the final step in one iteration of the DevOps stack. Now that the changes have been integrated, built and deployed to production, the team must have a way to measure the impact of their changes. This impact may be on the general user-experience, or it may also affect overall system performance and health. Appian provides out-of-the-box options for monitoring applications such as built-in alert notifications and the Appian Health Check.

Service Management

Users may experience critical errors or impacts to their daily routine based on the changes in a release. Appian utilizes an alert system that can automatically direct emails to system administrators that include details about an error. Workflows can be viewed on a step-by-step basis to identify exactly why an issue occurred. Additionally log files are provided to system administrators so that they can properly audit user authorization and logins to the environment.

It is important to realize these issues and communicate them back to the development team for prioritization. Ticket management software such as ServiceNow or JIRA can be updated with information about issues in the system. These issues should be presented and triaged in the next planning phase of the DevOps cycle.

Appian Health Check

The Appian Health Check is a free utility available in the Appian Admin console that provides push-button access to reports on system performance and utilization. The tool makes it easy to recognize technical debt for the development team by automatically highlighting areas that contradict best practices and informing the administrators about performance concerns. The Health Check is configurable to run on a scheduled basis and can also be set to automatically send the results to system administrator email addresses. It is recommended that technical leads review the Health Check from the production application on a weekly basis. Any "High" or "Medium" priority issues should be either resolved and tracked as part of technical debt or documented if there are special conditions around them.

Conclusion

The DevOps pipeline has revolutionized how low-code platforms can automate otherwise rigorous tasks and allow for streamlined, consistent delivery. Continuous integration provides the means to integrate multiple change sources together quickly, and thoroughly produce a high-confidence release candidate. Continuous deployment ensures that this product is delivered to the end-users quickly to improve their productivity. Monitoring tools allow the developers insight into the effects of their changes and allow for highly impactful planning sessions. An effective DevOps pipeline increases the productivity of the modern development team and enables quick, consistent delivery that allows developers to focus first and foremost on quality.

About the Author

Derek Shue is an Appian Level-3 Certified, Appian Enterprise Architect at Macedon Technologies. He has delivered significant modernization solutions for healthcare, bio-pharmaceutical and financial organizations. Derek was one of the first Level-3 certified developers in the Appian A-Score program and has focused on enhancing and standardizing DevOps best practices for Macedon Technologies.

[1] Ross, Malcom. "Webinar: Low-Code and DevOps." Appian, 2019
http://www.appian.com/resources/webinar-low-code-devops/

[2] Pittet, Sten. "Continuous Integration vs. Continuous Delivery vs. Continuous Deployment." Atlassian
http://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment