Progress

JumpStart to Continuous Integration & Continuous Delivery (CI/CD)

© 2020 Progress. All Rights Reserved.

WHITEPAPER

Table of Contents

Overview 3

The Importance of DevOps 5

CI/CD Primer 7
- Continuous Integration
- Continuous Delivery

The Benefits of CI/CD 10

Teams & Tooling 11

The CI/CD Pipeline 13
- Phase 1
- Phase 2

Conclusion 17

Appendix A 18
- Tooling Descriptions

© 2020 Progress. All Rights Reserved.

Overview

In traditional software development, developers check out the code and work on it separately until they finish the task. Then all these changes get merged into base code at the choice of the lead developer.

The merge process tends to happen infrequently—sometimes weeks but sometimes months. The process of sorting merge conflicts can get very complicated since multiple people are working on same codebase and the number of changes can be significant.

Building toward a 'monolithic' merge and release is fraught with risk. There are too many assumptions, plenty of opportunity for miscommunication and added complexity if something needs to get "undone." With the tooling available today, build and merges should happen at least daily. In doing so, the amount of risk is reduced exponentially.

So, integrating the source code frequently is highly recommended and a best practice. Each time the code is integrated, it should be tested. In traditional development teams, this involves the following steps:

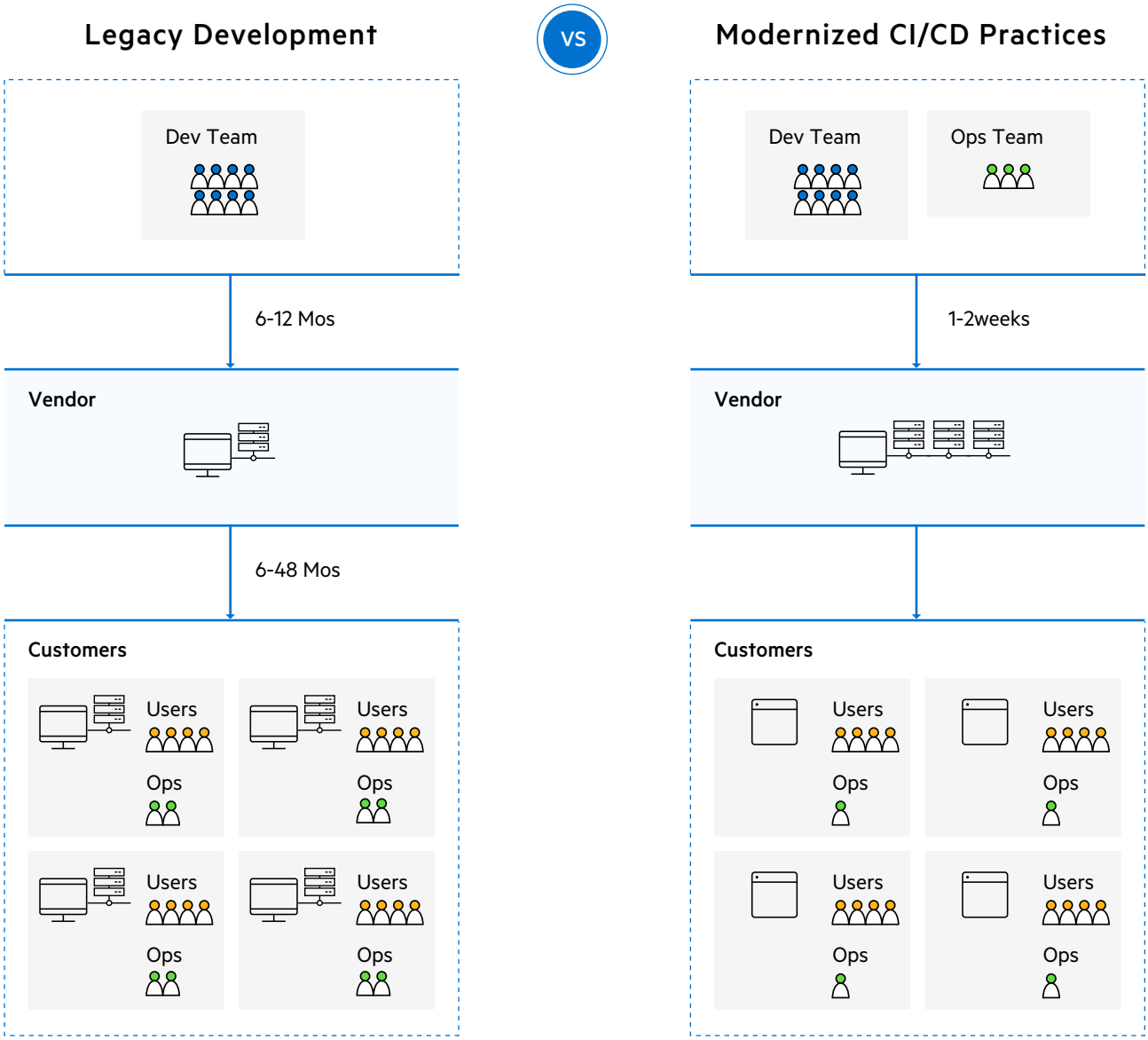
- Integrating the source code when a developer commits the changes to source code
- Build/release engineer builds the software after the integration
- QA tests the build
- Feedback is provided to the respective developer in case of defects
- The software is then marked as usable

The goal of CI/CD (continuous integration/continuous delivery) is for these tasks to be automated. Not only will this increase efficiency, it will provide for repeatable processes that guarantee quality and provide the auditing metrics necessarily to take corrective action if processes need to be reversed. This is achieved through continuous integration.

Continuous integration includes a series of tasks that facilitate the software integration process. The CI engine monitors and merges your source code each time a developer checks in code. The moment there is a check-in, the continuous integration process picks up all the latest changes, builds and tests the changes and sends out artifacts with the appropriate feedback. Continuous integration involves multiple components and different people. This typically includes people with different responsibilities and skillsets. More about this later.

© 2020 Progress. All Rights Reserved.

This paper explains the features and benefits of using CI/CD, and the tooling used in your software development environment. Your tooling may vary but the key is to have an overall strategy that is automated, repeatable and efficient. Continuous integration and continuous delivery are best practices and a vital part of any DevOps initiative. The purpose of this document is to provide an overview of the CI/CD lifecycle for Progress® OpenEdge® applications.



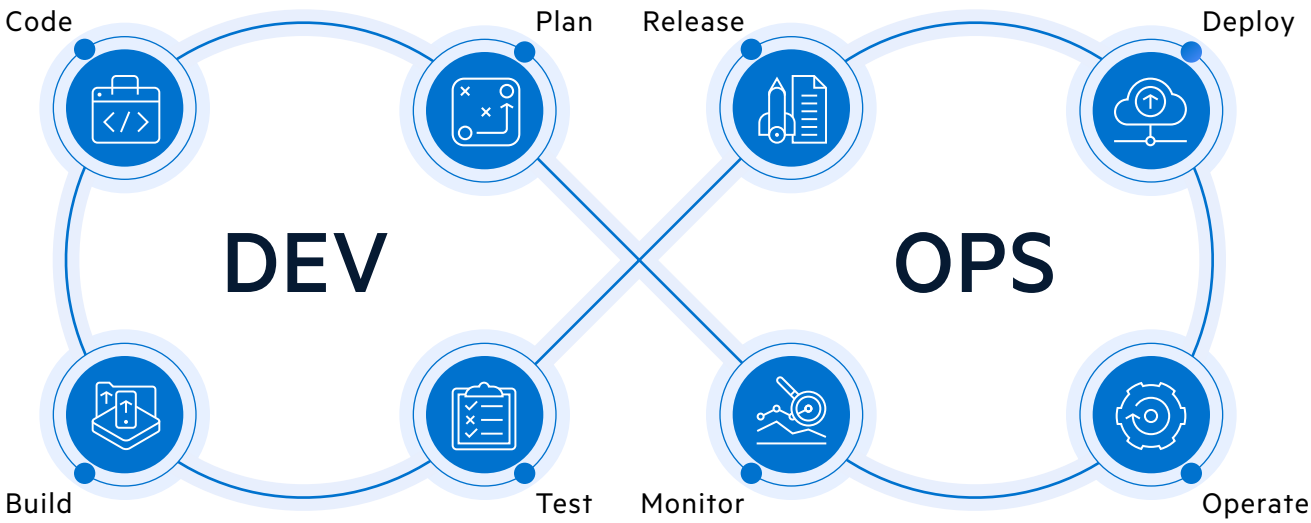
© 2020 Progress. All Rights Reserved.

The Importance of DevOps

Before jumping into CI/CD, let's take a step back and look at DevOps. DevOps is the combination of philosophies, practices, teams, and tools that improve an organization's ability to deliver applications and services at high velocity and scale. Using DevOps principles, organizations can evolve and improve products at a faster pace than organizations that use traditional software development and infrastructure management processes. This speed enables organizations to better serve their internal and external customers, based on constantly changing business drivers.

DevOps is collaboration between the development and operations teams. It's an integrated team committed to a unified process of building a working product utilizing continuous integration and continuous delivery, resulting in more value to the customer.

At its core, DevOps accelerates the process of delivering applications and software services at high velocity in a robust, measurable and repeatable manner. Also, it includes a cyclical feedback loop that minimizes risks by continuously delivering feedback to team members and stakeholders during the entire process. The timeliness of the feedback loop is extremely important because in traditional methodologies, feedback was delivered in less frequent intervals and at much later stage in the process, which provided ample opportunities for development and operations teams to lose touch with the delivery goals. This sometimes resulted in missed delivery targets as well as a fair amount of finger pointing.



© 2020 Progress. All Rights Reserved.

DevOps is the practice of operations and development engineers that work together in the entire project lifecycle, from design and development process to production releases and support. The team works together to achieve the common and desired goal. People having both development and operations skillsets working together, utilizing the same tools for CI/CD and monitoring.

This team approach minimizes working in "silos." With development and operations on the same team and with the same goals, there is better expectation setting and a more unified approach. With both groups being active participants in the process, feedback is given in frequent intervals, which enables developers and operations teams to solve problems and make changes far in advance of delivery deadlines.

In Summary, the benefits of DevOps can be defined by the following:

Collaboration between development and operations team

Break down the silos and create synergy. The development and operations teams work together toward a common goal through the entire software development cycle.

Velocity

Delivering quality releases quicker and faster.

Timely Delivery

Release software on a more frequent schedule or on demand in order to meet dynamic market conditions.

Improved Quality

By following best practices, automation and repeatable processes, defects are found and corrected far before delivery.

Security

Through automation, testing and validation, security requirements can be incorporated early in the process so that vulnerabilities are discovered and mitigated through entire development cycle.

Risk Mitigation

Because of continuous integration and feedback loops, early detection of issues and defects are uncovered and corrected as part of the normal collaboration process.

© 2020 Progress. All Rights Reserved.

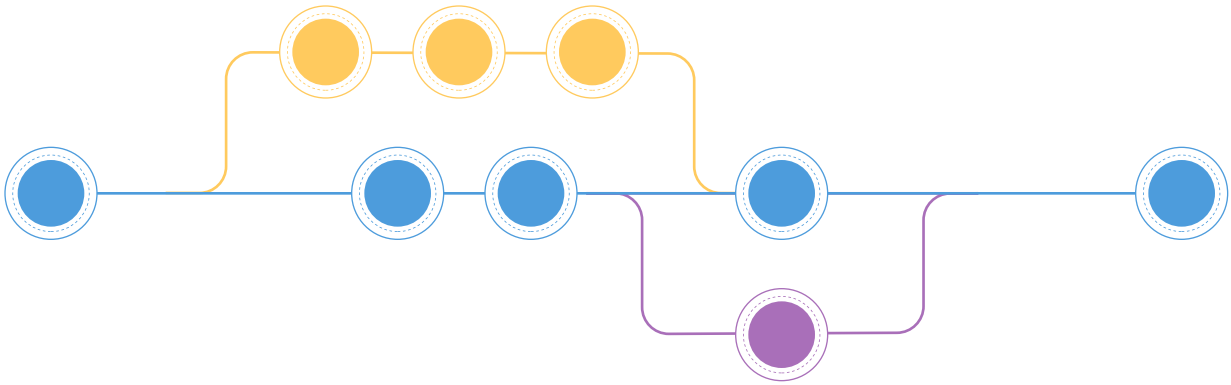
CI/CD Primer

Continuous Integration

One of the core tenets of continuous integration in a software development environment is that developers must regularly merge their code changes into a central repository. That enables automated builds and tests using the latest code. CI usually refers to the build or integration stage of the software release process and requires both automation (e.g., a CI or build service) and a best practice (e.g., the discipline to integrate frequently).

When done correctly, the CI process provides for the detection and resolution of defects more quickly. This results in improved software quality, and a reduction in the time it takes to validate and eventually release new software releases.

Because CI focuses on smaller commits and smaller code changes, developers commit code at regular intervals—sometimes multiple times per day. By pulling code from the centralized code repository frequently, the developer ensures the code on the local host is merged before pushing to the build server. This will reduce merge conflicts significantly. At that time, the build server runs the various tests and either validates or rejects the commit.



As explained above, in order to achieve consistency, efficiency and overall superior quality, more frequent commits to the common codebase are required. Because of the frequent commits, the opportunity to introduce a major defect is also reduced. Next, maintaining a single source code repository insures one version of the truth. Finally, automating builds and automating testing round out the key components of an effective CI process. Additional considerations may include testing in replica environments similar to production (e.g., staging and QA environment).

© 2020 Progress. All Rights Reserved.

Components involved in Continuous Integration

Primary components involved in Continuous Integration Environment are:

Development tool

The Developer Tooling will vary. Furthermore, different team members may utilize different tools depending on their expertise. Client-side developers (web developers and/or mobile developers) may utilize Eclipse or Visual Studio while ABL developers will utilize Progress Developer Studio (PDSOE), the Procedure editor or even VI to develop ABL classes and procedures. More about specific tools later.

Source Code Control

The Repository store, which contains committed source code. This also provides for version control.

Source Control Client

The Client that is integrated with the IDE, which enables developers to commit the source code to the Source Control Repository.

Build Tool

The tools that automate the compilation of source code to executable code.

Testing tool

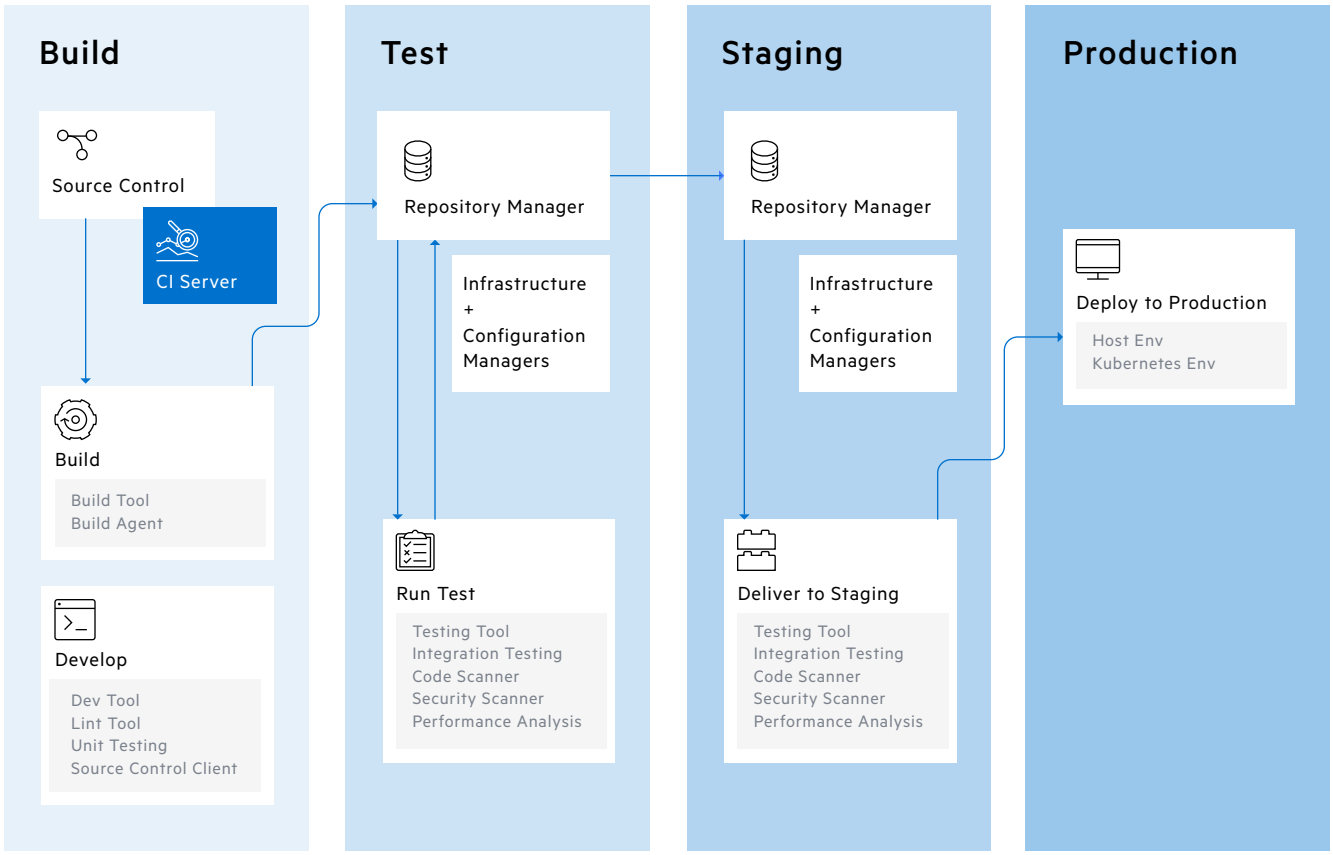
The tools that automate the tests scripts written for particular source code.

Continuous Integration Server

The tool that monitors the entire workflow. This is the workflow engine that builds the project every time code is committed to the repository. The CI server automatically checks out the code, does the build and verifies quality through automated tests. Finally, feedback is sent to the developer with generated build artifacts.

Continuous integration flow may include other source code analysis tools to analyze code and ensure standards are followed, identify possible technical debt or provide for performance testing capabilities.

© 2020 Progress. All Rights Reserved.



Continuous Delivery

Continuous delivery (CD) is facilitated through a successful continuous integration process. It is a software development practice where code changes are automatically built, tested and prepared for production release. It is an extension of continuous integration by enabling the deployment of all code changes to a testing environment, a production environment or both after the build stage has been successfully completed. Depending on your overall goals and the tools that are being implemented, CD may be fully automated with a workflow process or partially automated with manual steps at points along the pipeline. A key result of a properly implemented continuous delivery process is that developers have a build artifact that is deployable once it has passed a validated, quality assurance test process.

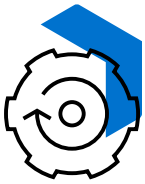
With workflow and effective CD processes, application revisions may be deployed to a production environment automatically, manually or on demand, making the entire software release process repeatable and highly configurable. You no longer need the infrastructure team to work on nights and/or weekends to deploy the latest release and possibly cause disruption or significant downtime to mission-critical applications in order to get the latest release. It should also be noted that one of the key CD advantages is that it allows for a timely and predictable customer feedback loop early and often within the product development lifecycle.

© 2020 Progress. All Rights Reserved.

Continuous delivery does NOT mean that the software is continuously deployed. This point should not be overlooked. The key is to understand that the CI/CD process is not to apply every single change to production but to insure that all changes are incorporated into the CI process so that when the decision to release is made, every code change leading up to that point in time would have been applied after being merged, tested and validated.

Having the right strategy makes the release process more of a business decision and not a technical one. Equally important, with a repeatable process that has been iterated and proven multiple times, the release becomes standard operating procedure. It is not disruptive and does not require a highly skilled technical team to push the release to production.

Benefits of CI/CD



Automation

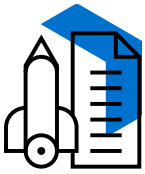
Build automation insures efficiency, faster build and less human intervention.

Code Stability

Automation enables quicker feedback loops in cases of code defects. Identification of defects and logging results in quicker resolution.

Metrics and Analytics

Quality dashboards (e.g. SonarQube) provide complete visibility and a complete view into build results. Software such as SonarQube can help with validating coding standards, best-practices, security vulnerabilities and performance pitfalls, using with automated static Code Analysis rules.



Faster Updates/Releases

Confidence in code quality and repeatable processes enables faster release cycles.

© 2020 Progress. All Rights Reserved.

Productivity

Developers will be able to focus on developing and spend far less energy on the CI process.

Quality

The combination of test automation and other CI benefits allows for more development focus, resulting in better quality.

Enables CD

Through a CI Server (e.g., Jenkins), resulting builds can be auto deployed to the target server as desired.

Teams & Tooling

Teams

Application Team

This is your traditional development team. They are the builders of the application. They write the code based on stories, and prepare unit tests. They develop features based on a documented specifications. The focus of the team should be application development and not infrastructure or operational concerns. The application team should be familiar with the platform and system configuration but only from the point of view of their interface and nothing more.

Infrastructure Team

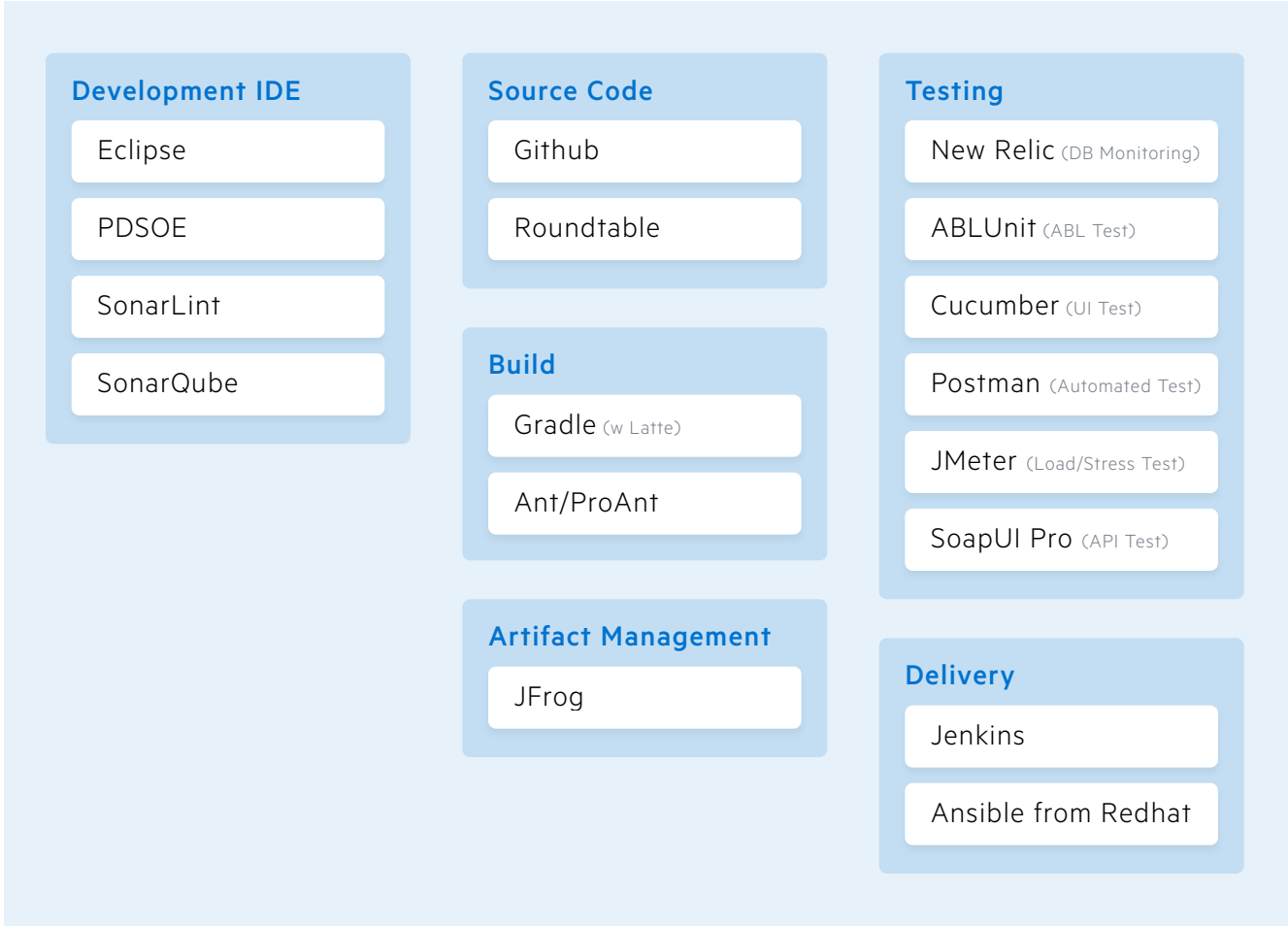
This team is focused on creating the code and processes required for the creation, configuration, management and ongoing maintenance of the infrastructure needed to run the CI/CD pipeline. This includes source control repositories, CI workflow engines, build environments, testing frameworks and artifact repositories. They are responsible for the specific tooling utilized within the pipeline. This team works closely with the application team to insure the CI/CD phases support the overall delivery required.

© 2020 Progress. All Rights Reserved.

The team should have skills in infrastructure provisioning methods using the tools that have been selected. The team must also include the ability to develop configuration automation skills. They also need to be monitoring the process, identifying any areas that need improvement.

It should be noted in large development environments, the infrastructure team may be broken into two teams: infrastructure and tools. This will allow for a higher degree of specialization, which may be required at scale.

Tooling

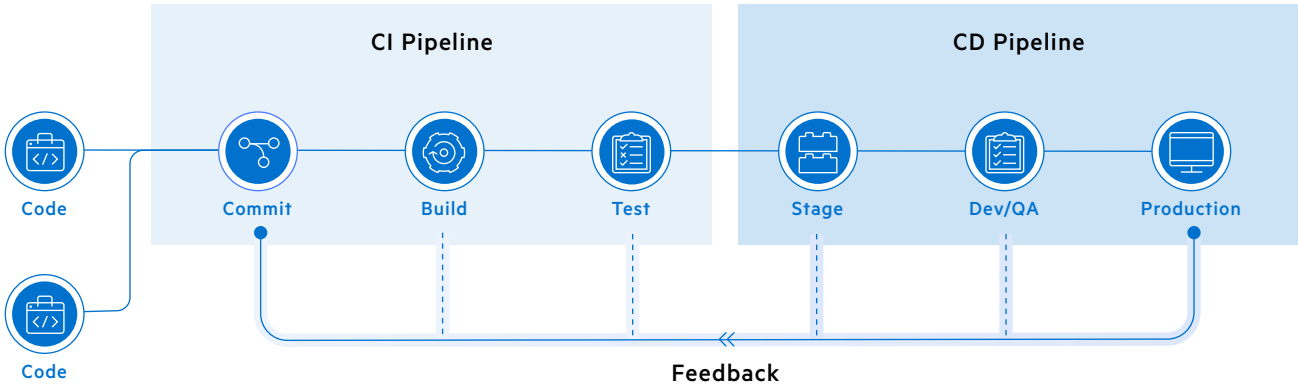


See Appendix A for additional information.

© 2020 Progress. All Rights Reserved.

THE CI/CD Pipeline

Now that CI/CD has been defined and benefits have been outlined, a logical next step would be to take a closer look how to implement CI/CD. CI/CD is typically visualized as a pipeline. An important aspect of the pipleine is the ability to iterate, test and validate, provide feedback at critical points and to fail back (undo tasks) if required.



Developing a mature CI/CD pipeline is a journey. It is best to start small, identifying key requirements and tools along the pipeline, iterate and adjust as needed. Based on the diagram above, it all starts with the source code being committed and merged into the code repository. Then, moving through the build, staging and eventually production environment when the code is released.

Each stage of the pipeline also serves as a gate that verifies certain aspects of the code before it is promoted to the next stage. Because of the tooling, auditing and validation of the code, it should be of a higher quality as it moves down the pipe. As mentioned earlier, a key function within each stage is to provide feedback. Once a defect is discovered, feedback is provided immediately, and the code will not be able to progress to the next stage. Test results are available immediately and feedback is sent to the team in order for remediation to occur.

It is highly recommended to build out your pipeline in phases. In other words, start small, iterate and learn as you go. There will be more opportunities as lessons are learned to build out the right number of environments, testing phases and various forms of automation.

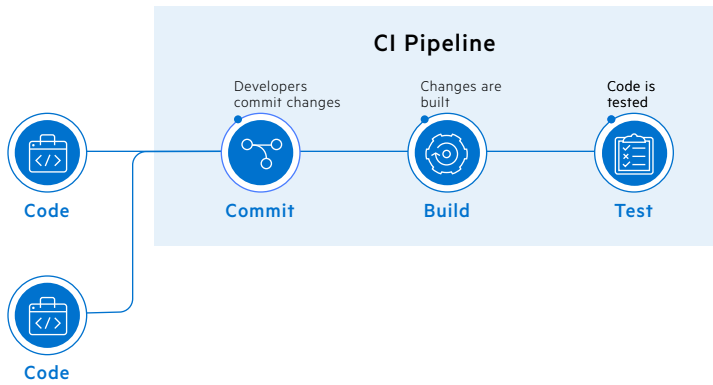
© 2020 Progress. All Rights Reserved.

Phase 1 - Continuous Integration (CI)

First, there must be maturity and discipline in continuous integration. This starts with all developers committing their code to a centralized repository on a regular basis, which enables all changes to be merged into a specific release branch. No code should be held in isolation. Code that is merged early and often will have less problems downstream. This will require discipline on the part of the entire development staff.

Another important best practice is to develop unit tests early in the process. The unit tests should be created and run against the code before it is submitted to the repository. Errors that are caught early in the development process are the easiest to fix.

Continuous Integration



After the code is submitted to the code repository, a CI workflow engine will invoke a build and run unit tests against that code for the specific branch that it was saved. Depending on the outcome, feedback will be provided immediately. There may also be other quality checks done at this stage for coding standards, unit test coverage, quality checks and possible more. Finally, the build artifact will be created. This may also result in other artifacts being generated, including images, documentation stylesheets.

In Phase 1, the levels of testing include:

- **Unit testing**

  Tests specifics to the code functionality. Is the code behaving as desired? Unit testing is done by the development team.

- **Static Code Analysis**

  This analysis will help uncover bug defects, security issues, coding standards, data flow and code coverage.

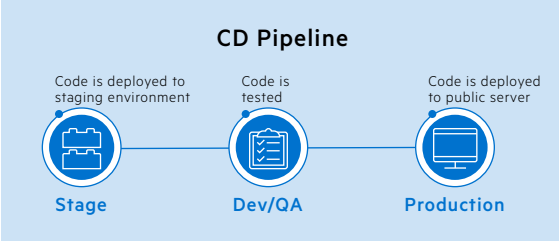
© 2020 Progress. All Rights Reserved.

Phase 2 - Continuous Delivery (CD)

The next phase is continuous delivery (CD). This involves deploying the application code into a staging environment, which should be an exact replica of the production environment. This is important because you want to be able to eliminate any potential points of failure. Having an exact replica will verify that a build that runs in one stack should run identically in its replica.

The staging environment may be a static platform that is already built for further testing. Alternatively, you could provision a dynamic staging environment as part of the automation within the CD process.

Continuous Delivery



In the staging environment, testing includes:

- **Integration Testing**

  Many applications must interface with internal and external systems. This testing ensures various components interface seamlessly.

- **Component Testing**

  This primarily involves message passaging between various components and validating the results, including abnormal or unexpected inputs, corner cases or large volumes of data. The desired outcome is consistent or idempotent results.

- **Performance Testing**

  Determines the performance and scalability of the application under varying workloads. Performance tests are used for benchmarking and include load or stress testing.

- **System Testing**

  This performs end-to-end testing of the system. Does the software answer the business requirement? This could include user interface, API contracts, business logic and data access.

© 2020 Progress. All Rights Reserved.

- **Compliance testing**

  Refers to the nonfunctional specifications. Is the application meeting defined standards? For example, are multi-lingual or accessibility standards being met?

- **User Acceptance Testing (UAT)**

  This validates workflow and business flow as well as look and feel. This is typically done by end users or power users.

Similarly, the production environment should be provisioned in the same way. Again, having a replica eliminates variables that can prove to be points of failure. Once this process is set up, it is repeatable and very efficient.

This results in the ability to deploy to production at the tail end of the pipeline. Optimally, this includes full automation of the software release process from start to finish, including a push to production. Furthermore, due to automation and repetition, this resulting build can be deployed with confidence in an extremely efficient way, in a much smaller timeframe resulting in minimal disruption to the production systems.

As stated earlier, CI/CD is a journey. The recommendation was to start small, iterate and build as your process matures.

Fine Tuning Your CI/CD

Areas where additional opportunities for fine tuning your CI/CD journey may include but are not limited to:

- Multiple staging environments for other specific use cases. Individual staging environments may be preferable for performance testing, security, user acceptance testing (UAT) and compliance.

- Integration with other systems or environments.

- Schema changes introduce another level of complexity. Specific testing and staging for schema changes may need to become part of your process.

© 2020 Progress. All Rights Reserved.

Deployment

There are many different strategies that can be implemented for deploying your application. Popular deployment methods include:

- In Place
- Rolling
- Blue/Green

The details concerning deployment strategies are not in scope for this document. Please see the Progress Deployment Strategy White Paper for this information.

Conclusion

In the agile and dynamic software development environments of today, having a mature CI/CD strategy is mandatory. By implementing simple, repeatable processes, your developers will be able to focus on application development. This will result in better team efficiencies and code quality.

The best outcome is to be able to deploy your code with a high level of confidence based on a mature CI/CD process. The deployment decision should be a business decision, not a technical one. Your entire development operation will benefit from having a non-disruptive, low-risk capability of delivering and deploying software updates automatically or on demand to meet today's demanding business challenges.

© 2020 Progress. All Rights Reserved.

Appendix A

Tooling Descriptions

The specific tools used within organizations that have implemented successful DevOps pipelines vary far and wide. Because having a comprehensive DevOps strategy is now front and center with modern software development organizations, new tools are being introduced weekly and existing tools are being upgraded with additional functionality or extended to meet the growing business drivers in the market today.

This overview is not meant to be a comprehensive list of available tools. Instead, we are focusing on the tooling we have used and have seen our customers utilize in the recent past. The analysis also looks internally at some of the tools used by the Engineering department at Progress.

GitHub

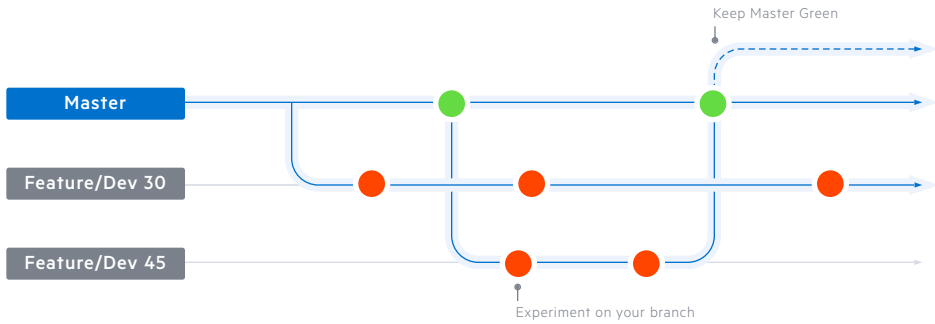
Like other version control systems, GitHub manages and stores revisions of projects. With Git, you copy the whole project repository to your own system and make your changes on your local copy. Then, when changes are "checked in" to the central server, changes are merged and are more granular since connecting to the server is not required after every change.

Git is a command line tool. However, it also supports a web-based graphical interface. It provides access control and several collaboration features, such as wikis and basic task management tools.

The key functionality of GitHub is "branching"— the ability to copy a repository from one user's account to another. This enables developers to make changes to projects that they may not have write access to and modify it under their own account. Later, a notification called a "pull request" can be sent to the original owner if changes were made that you would like to be integrated into the original repository. With the click of a button, the changes will be merged within the original repository.

These are the three key features of GitHub: fork, pull request and merge.

Basic Workflow



© 2020 Progress. All Rights Reserved.

See Appendix A for links that provide details concerning Progress Develop Studio for OpenEdge (PDSOE) and the GIT plugin. You can also read this knowledge base article.

Roundtable

Roundtable Team is another popular solution within the Progress community. This solution supports team collaboration and parallel development tools, providing a complete system for tracking and managing ongoing changes to development projects, application source code and related files.

Roundtable provides seamless integration to OpenEdge and Eclipse which is the foundation of the PDSOE. Built around the Eclipse team API, Roundtable Team is easily installed and tightly integrated into your PDSOE environment, providing a familiar version control interface for all project types.

Additional details concerning PDSOE and the Roundtable plug-in can be found on the Roundtable website.

Progress Developer Studio for OpenEdge (PDSOE)

PDSOE is an Eclipse-based OpenEdge development environment that you use to create, test, debug and deploy OpenEdge ABL applications.

Code insight, which is the ability for the program to interpret what is typed out, is one of the most helpful tools that an IDE can provide. The program can change text color to represent different classes, functions and variables. PDSOE offers something called IntelliSense that can also predict what you are typing out and finish your words for you.

Another important IDE feature is the ability to debug your code. This is arguably one of the most important tools to deploying a successful program. Testing is crucial to avoiding incidents where the code does not handle a specific error and crash the program, tarnishing the user experience. Debugging provides you with the ability to run through the program, stopping the code at specified points to check values of variables or other interests as needed, to verify that the code and functions are running as intended.

© 2020 Progress. All Rights Reserved.

SonarLint

SonarLint Is a Code Analyzer/Lint tool. PDSOE has a plugin for SonarLint and also integrates well with SonarQube. SonarLint helps detect and fix quality issues as the code is written. This is far more efficient than finding issues after the code is committed. The OpenEdge plugin for SonarQube (using SonarLint) enables analysis of OpenEdge procedural and object-oriented code, and reports on the following:

- Metrics (lines of code, comment density, complexity, shared objects, ...)
- Issues found in the source code and DF files, such as performance killers, usage of deprecated keywords, etc.
- Code coverage from unit tests
- Code duplication

To learn more, please see this GitHub repository.

SonarQube

SonarQube is an automatic code review tool to detect bugs, vulnerabilities and code smells. It can integrate with your existing workflow to enable continuous code inspection across your project branches and pull requests. SonarQube is an open platform to manage code quality, as such, it covers the seven axis of code quality, specifically:

1. Architecture and design
2. Code duplication
3. Unit tests
4. Code complexity
5. Potential bugs
6. Coding rules
7. Comments

Jenkins

Jenkins offers a simple way to set up a continuous integration and continuous delivery environment for almost any combination of languages and source code repositories. Jenkins does not eliminate the need to create scripts for individual steps, however, it provides an efficient and scalable way to integrate your entire CI/CD pipeline, including build, test and push-button deployment, which frees up developer time and reduces human error and downtime.

© 2020 Progress. All Rights Reserved.

Ansible from Redhat

Ansible is a comprehensive IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration and many other IT needs. Designed for multi-tier deployments, Ansible models your IT infrastructure by integrating all of your inter-related systems in a seamless way.

Gradle

Gradle is a build automation tool often used for a variety of languages. Gradle tasks can be configured to perform specific jobs such as compile jars, run tests, create documentation to name a few. Another feature of Gradle worth mentioning is that you have the ability to check-in its bootstrap to your source code, and it works anywhere without the need to install it.

Latte

Latte is a plugin for Gradle providing language support for OpenEdge ABL. It provides Gradle tasks to compile ABL code and run unit tests using ABLUnit. All the hard work is done by the Progress Compile Tool (PCT), thanks to Gradle's integration with Ant.

For more information, see the Gradle GitHub page.

Ant/ProAnt

Ant/ProAnt is a software tool for automating software build processes. It has built-in tasks that enable you to compile, assemble, test and run applications.

PDSOE can be configured with the Ant Editor and Builder and can execute Ant jobs. ProAnt is Progress-provided shell that sets the environment for a Progress application. ProAnt/Ant use PCT (Progress Compilation Tools) for compiling Progress artifacts. PCT is designed and written by Gilles Querret of Riverside Software. PCT is included with OpenEdge install.

OpenEdge release 11.7 (and greater) ships with a copy of the Ant build tool. A component called 'proant' configures and runs the script is found in the $DLC/bin directory.

For more information, read this knowledge base article.

New Relic

The New Relic plugin for the OpenEdge database is a great choice for displaying, monitoring and sharing critical information about database performance with the organization.

With the New Relic plugin for the OpenEdge database, you can capture, view and monitor critical database performance metrics. As an administrator, you can configure the performance metrics that are essential to your business.

© 2020 Progress. All Rights Reserved.

The New Relic plugin for the OpenEdge database can be installed on both Windows and OpenEdge-supported UNIX platforms. You can edit the metrics.regex property in the oedbnr.properties file, and specify the metrics that are important for your business.

The New Relic plugin for the OpenEdge database captures the following metrics:

- Activity Summary
- AI Activity
- BI Activity
- Buffer Activity
- Record Activity

JFrog

JFrog Artifactory is touted as a universal repository. It acts as the mission-critical single source of truth for all packages, container images and other artifacts that describe deployment services across the entire DevOps pipeline.

You can use Artifactory as a registry to manage application packages, an operating system's component dependencies, open-source libraries, Docker containers and Helm charts, which are used to deploy packages within Kubernetes with full visibility of all dependencies. Artifactory integration capabilities helps overcome the complexity of working with different software package management systems and provide consistency to your CI/CD pipeline.

ABLUnit

ABLUnit is a unit-testing framework for the Advanced Business Language (ABL) program. It is similar to JUnit-based and XUnit-based unit-testing frameworks. ABLUnit for Progress Developer Studio for OpenEdge helps you to write and run repeatable unit test cases or test suites. The automated testing of your programs checks if the code is working as intended and is valid.
It helps you to:

- Identify bugs in your code
- Write modular and loosely-coupled code
- Test the code in parts

The ABLUnit testing framework supports two types of test cases:

- Test Class and Test Suite Class (.cls file)
- Test Procedure and Test Suite Procedure (.p file)

© 2020 Progress. All Rights Reserved.

ABLUnit testing framework includes the following components:

- Annotations for marking the test cases
- Assertions for testing expected results
- Test suites for easily organizing and running test cases
- Test runners for running a test case or test suite from the command line or launch configuration
- Results view for reviewing and analyzing the test results

For more information, please read this knowledge base article.

Cucumber

Cucumber is an open-source tool that supports behavior-driven development.

Because it utilizes plain English text, Cucumber can be defined as a simple and easy-to-use testing framework. It is a development aid that provides functionality for documentation and automated testing.

With the ability to create test cases in plain English text, Cucumber bridges the gap between business and technical language. In addition, test scripts can be written without knowledge of any code, which extends the functionality to non-programmers. It also supports end-to-end testing and is extensible supporting test code reusability.

Postman

Postman is a collaboration platform for API development. Postman's features simplify each step of building and testing an API and streamline collaboration so you can create and test APIs faster.

Request types

Postman supports many types of requests. You can create and save custom methods and send requests with the following body types:

- URL-encoded - The default content type for sending simple text data
- Multipart/form-data - For sending large quantities of binary data or text containing non-ASCII characters

© 2020 Progress. All Rights Reserved.

- Raw body editing - For sending data without any encoding
- Binary data - For sending image, audio, video, or text files

Rapid development

Instead of manually creating calls to send over the command line, all you need is a Postman Collection. Import a collection directly or generate one with one click from:

- An API schema in the RAML, WADL, OpenAPI, or GraphQL format
- A data file containing the cURL commands

Feedback

Status code, response time and response size are all viewable within the tool. Inspection of the response body is simplified by Postman's automatic language detection, link and syntax highlighting, search and text formatting.

Apache JMeter™

Apache JMeter is an open-source software application. It is designed to load test functional behavior and measure performance. Apache JMeter will provide testing capabilities on both static and dynamic resources as well as web dynamic applications.

JMeter can be used to simulate a heavy load on a server, group of servers, network or object. It can be used to test its strength or to analyze overall performance under different load types.

SoapUI Pro

SoapUI Pro is the fully featured professional version of SoapUI Open Source for teams and enterprises. SoapUI supports API testing, helping Agile and DevOps teams focused on scalable and high-quality APIs. It enables agile teams to create, manage and execute tests for REST and SOAP APIs quickly and easily.

© 2020 Progress. All Rights Reserved.

Meet The Authors

ALEX OLIVERI
SENIOR DIRECTOR

Alex Oliveri, as Senior Director of Professional Services, is passionate about current application development trends and believes in delivering solutions that solve our customers' business problems utilizing innovative technology, consulting and training. A re-emphasis in implementing and fine-tuning CI/CD processes within the community is driving many current services engagements. With experience as a direct end user, partner and employee of Progress, Alex brings a thoughtful and well-rounded awareness of technical solutions to each customer opportunity.

Jagdish Pai is a Principal Consultant who has been building character/GUI/web/mobile solutions using Progress OpenEdge since 1997. Jagdish has worked with a large number of clients across multiple verticals in a variety of roles such as developer, architect, mentor, etc. Over the years, Jagdish has developed many products/toolkits/methodologies that have helped clients modernize their applications. All of these projects have implemented some form of the CI/CD practices that are included in this whitepaper.

JAGDISH PAI
PRINCIPAL CONSULTANT

STEVE SOUTHWELL
PRINCIPAL CONSULTANT

Steve Southwell is a Principal Consultant who has been building web solutions using Progress OpenEdge since 1996. Steve has worked with clients in many industries, primarily focusing on security and performance. More recently, Steve has implemented several CI/CD solutions and is a huge advocate of the discipline. Working from the Progress office in Lewisville, Texas, he now assists clients with designing and developing innovative web applications utilizing the Progress Application Server for OpenEdge (PASOE).

Rob Marshall has spent over 35 years working in the Progress community, assisting everyone from end-users to application partners as part of the Professional Services Team at Progress. Rob holds numerous certifications in Project Management and Security, as a result, he is quite aware of the benefits of CI/CD workflows. Rob is currently completing his Ph.D. in Information Assurance and Cybersecurity.

ROB MARSHALL
SENOIR PRINCIPAL CONSULTANT

© 2020 Progress. All Rights Reserved.

Contact Us Today!

About Progress

Progress (NASDAQ: PRGS) offers the leading platform for developing and deploying strategic business applications. We enable customers and partners to deliver modern, high-impact digital experiences with a fraction of the effort, time and cost. Progress offers powerful tools for easily building adaptive user experiences across any type of device or touchpoint, the flexibility of a cloud-native app dev platform to deliver modern apps, leading data connectivity technology, web content management, business rules, secure file transfer, network monitoring, plus award-winning machine learning that enables cognitive capabilities to be a part of any application. Over 1,700 independent software vendors, 100,000 enterprise customers, and two million developers rely on Progress to power their applications. Learn about Progress at www.progress.com or +1-800-477-6473.

© 2020 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Progress®

© 2020 Progress. All Rights Reserved.