Inflectra: Software Built For You

Our one goal is to help you succeed. We care deeply about giving you the best quality service and support you've ever had.

As many users, projects, tests, releases, items, API calls as you want. All pricing is based on concurrent users.

Flexible options to make your life easier. Use on desktop or mobile; your servers or our cloud, sensible add-ons, fairly priced.

DevOps in Seven Easy Steps

DevOps With Inflectra

- **Plan**: SpiraPlan® offers a turnkey solution for managing your projects and supports agile, as well as waterfall and hybrid
- **Create**: SpiraPlan® includes add-ons for the most popular IDEs. TaraVault® from Inflectra provides an enterprise-grade Source Code Management solution for both Git and Subversion
- **Verify**: SpiraPlan® offers world-class test case-based manual testing, as well as freeform exploratory testing. It also provides plugins for the most popular automated testing frameworks to tie unit tests directly to requirements. Rapise® is a powerful automation tool for both UI and API testing
- **Package**: SpiraPlan® helps you auto-generate release documentation, as well as the final code and installation files
- **Release**: SpiraPlan® supports change and release management with simple workflows that can be customized to include robust approval routing, signoffs, and electronic signatures
- **Monitor**: KronoDesk® offers streamlined customer support and helps you collect and action user feedback.

7 Elements of Effective DevOps

The marriage of agile methodologies, virtualized and cloud computing, and on-demand infrastructure has revolutionized the creation and management of software and IT resources. All organizations can benefit from these changes using DevOps - how much and in what ways depends on specific technological and regulatory constraints.

1. **Plan**: "define" the requirements and business value, then "plan" activities and milestones
2. **Create**: write and debug code, manage and version control the source code, then build using continuous integration
3. **Verify**: continuously test using a range of approaches to improve quality and provide feedback on business risks
4. **Package**: reliably create a package of the application, using artifact repositories and tailored deployment strategies
5. **Release**: manage each release, its artifacts and configuration, automate its deployment, then track and manage change
6. **Configure**: automate and configure infrastructure provisioning
7. **Monitor**: collect and act on monitoring of performance, security, business metrics, and end–user experience and feedback.

Visit us at www.inflectra.com for a free trial
Or get in touch with us: sales@inflectra.com, 1-866-572-5878 or +1 202-558-6885 (international)

inflectra®

1. Plan

Typically, in DevOps, planning is composed of two things: "**define**" and "**plan**". "Define" a common understanding of the requirements and business value of the product being developed. "Plan" out the requirements into a set of activities, milestones, and required roles. The nature of the plan will depend on the project management methodology being used (agile projects will use high-level and fluid user stories whereas waterfall projects will use more stable and prescribed requirements and deadlines).

2. Create

Writing & Debugging Code: The choice of tools to design, write, compile, and debug code is usually based on the technologies chosen to create the product. Choose tools that make documenting, refactoring, testing, and understanding the code easier.

Managing & Versioning Code: A robust set of tools and practices to store and manage source code is essential. The choice between Subversion and Git (the two-leading platforms) will depend on your needs. You may also need code analysis, code review, code documenting tools, or code security and performance analysis.

Building & Integrating Code: The final phase in this step is to integrate all code into a single build. Choose a Continuous Integration (CI) tool that works well with the rest of your DevOps toolchain. Use extensions that support your compilers, IDEs, SCM tools, databases, and deployment processes. Ideally, your CI tool should report directly into your ALM suite. Many CI tools can also execute jobs to orchestrate other parts of the DevOps toolchain, such as automated testing, staging, deployment, and notifications.

3. Verify

Testing and verification mitigates technical risks, helps management understand the quality of the software, and provide metrics to determine if the build is ready for deployment and production. Where possible, use continuous testing tools and processes that provide real-time feedback on the risks in the system.

Unit & Integration Testing: Most organizations using agile methodologies will practice Test Driven Development (TDD) to ensure there is good test coverage of the code. Ideally, you should also use the same unit testing frameworks for automated integration tests that test multiple modules of code at once.

User Interface Testing: UI is often the most frequently changing part of the application and the hardest to test. Make sure you have the appropriate tools for testing the type of UI you are using (e.g. web, mobile, desktop, console) and that you understand which parts of the UI testing should be automated.

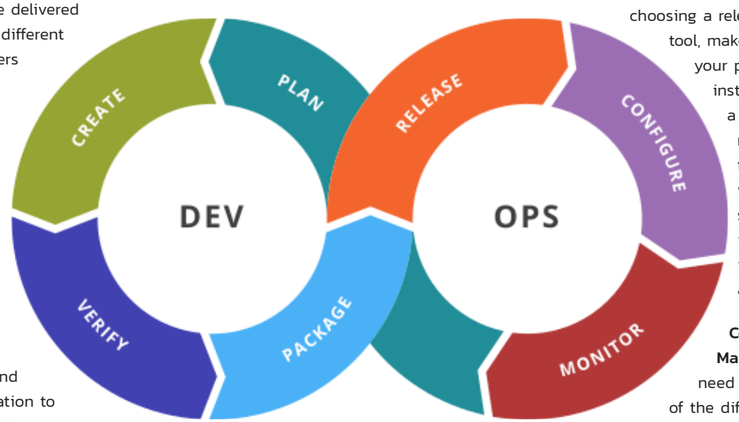
API Testing is important if the product is to be part of an ecosystem. The days of monolithic applications are limited: having well-maintained, versioned, and tested APIs can be a big differentiator for your products. External developers prefer not having to rewrite their code every time you ship a new version of your product.

Exploratory & Manual Testing: Automated tests will not catch all issues. You need skilled human testers looking at the application during and after development. Traditional, scripted manual testing or User Acceptance Testing (UAT) is often used in industries where there are regulatory and/or legal requirements to have users test every path. In other cases more freeform, unscripted, exploratory testing is helpful. Here, testers follow their own path and intuition to seek out issues that may have been overlooked.

Performance & Security Testing: Beyond product functionality, test performance (how well it works under different loads) and security (how vulnerable it is to security penetration). Make sure these tools interoperate with your ALM and DevOps toolchain.

4. Package

This vital aspect of DevOps is often overlooked. You need to have a repeatable, reliable process to package all the code, documentation, data, and other artifacts created in the CI build pipeline for deployment and release. The process depends on how the software is to be delivered and used: it will be very different for software that customers deploy themselves compared to Software as a Service (SaaS).



Artifact Repository: You need tools and processes that automate the identification and packaging of changed items in the software. You can use this repository to auto-generate release notes, and system or API documentation to reduce omissions or errors.

Cloud Deployment Strategies: For cloud-based deployment, package your software and associated artifacts to: facilitate easy distribution to your cloud platform; ensure it can be rolled back quickly if needed; reduce the burden on the release process; and align with your cloud tenancy model. For single-tenant systems the process may be similar to releasing a download product. For multi-tenant systems, consider tools such as "feature flags" so you can release the same deployment to all customers.

Images and Containers: For cloud applications, you can package a copy of the application, entire OS, and pre-requisite services (e.g. web and database servers). This is a "machine image" or virtual machine (VM) image. This is quick to deploy and restore but creates significant duplication of the stack and incurs additional license and maintenance costs. Alternatively, containerize your application with a minimal set of infrastructure services into a single unit (a container). These can be deployed multiple times in the same OS machine image, bringing deployment and scalability advantages, with reduced infrastructure overhead.

On-Premise Strategies: Determine how to combine all build artifacts into a single package that can easily be delivered to the customer. This package should ideally be self-installing. IT staff should be provided documentation for deployment and maintenance, and be able to configure the installation for their environment, understand the prerequisites. Alternatively, you can provide a Virtual Machine (VM) that can be deployed as a single image onto on-premise infrastructure.

5. Release

Release Management is how to describe, document, and manage the different versions of a product, and is the foundation of this DevOps step. The approach to release management depends on your software lifecycle and release tempo. Whatever the approach, document what releases are planned, what functionality each release contains, and what packages are to be deployed. This process should, ideally, be automated: if a feature is removed the system "recalculates" what assets (e.g. code, tests, documentation) to remove from the release. When choosing a release management tool, make sure you understand your process and needs. For instance, do you need a tool that supports release baselining, or formal release approval workflow, with signatures, signoff, and auditing? Or is creating a release and immediately deploying against it sufficient?

Configuration Management: You often need to maintain a history of the different configurations of a product. This may primarily be the source code, documentation, packages, and artifacts stored in the SCM system. More regulated environments may need a full inventory of items related to each release. Depending on the type of information, you may be able to leverage your existing SCM or ALM tool to manage any additional items. Make sure to also maintain a branching and merging strategy.

Change Management: A product is not static once it is released. There will be bugs to track, enhancements to record, and change requests to approve. Make sure there is a well-documented change management process, agreed by management, that aligns with any quality or security standards your organization follows (e.g. ISO 9001, PCI-DSS, SSAE 16). Some change management strategies are very lightweight, essentially an issue triaging workflow. Other strategies are more complex, with multiple levels of approval, external audit steps, and regulatory oversight. Ensure your change management tools include the necessary functionality to handle your process. Finally, a clear release and communications strategy for the production environment will help your users know what to expect.

Release Deployment Automation: For cloud solutions deployment may mean pushing the package to a production environment, applying the changes, initiating alerts and notifications to users, and changing feature flags to enable functionality. For on-premise products, it could involve uploading an installation package to a customer portal and emailing it to customers, or having an automated update process on a customer's environment pick up the package from a central service. Whatever the process, make it as automated as possible, and integrated into other parts of the release process to reduce the likelihood of errors creeping in.

6. Configure

This step in the DevOps toolchain refers to the trend away from large, fixed infrastructure resources toward more flexible, on-demand provisioning. First was the move from physical servers running a single OS to virtualized environments. Next, was the transition to Infrastructure as a Service (IaaS) platforms like Amazon Web Services (AWS) that turn the process of provisioning a new server from a month-long purchasing and installation project, to a five-minute task. Then came Infrastructure as Code (IaC) that allows you to design, implement, and deploy an application's infrastructure entirely through code. The most recent phase is Continuous Configuration Automation (CCA) that lets you change, configure, and automate infrastructure provisioning in the same way that CI tools let you automate the building of software packages.

7. Monitor

One of the most important parts of DevOps is having a robust set of monitoring, tracing and event reporting tools to ensure that unanticipated changes are detected and remediated before they impact users. Aspects of monitoring to consider include:

System Monitoring: Real-time, automated monitoring of your systems and infrastructure. This should cover: Functional Monitoring – are all systems working correctly, are the APIs available, are there any outages?; Performance Monitoring – is the system responding normally to the current user load?; and Security Monitoring – is the system secure with no exploitable vulnerabilities or active cyberattacks?

Business Monitoring: Tie business metrics to your DevOps monitoring platform. There are free tools (such as Google Analytics), plus several commercial applications that can monitor and alert based on business transactions and other metrics.

User Monitoring: Users are also an important part of the monitoring process. Ensure the tools used to support users are fully integrated into your monitoring environment. For example, your help desk system can be used to generate metrics around issues raised with each release. Beyond providing insight into the stability and usability of your system, feedback and ideas from your users are a critical resource that should be fed back into the start of the DevOps toolchain.