WHITE PAPER

DevOps for Digital Enterprises

Abstract

DevOps is fast assuming greater importance in deciding the agility of an enterprise. A robust DevOps setup is crucial for successful agile delivery and minimal risks. It greatly optimizes release management costs and team productivity, resulting in reduced time to market. At the same time, DevOps enables organizations to make rapid product releases with increased quality and manage customers' expectations.

In this paper we explore various aspects of DevOps. We look at key success attributes, main processes, tools, and frameworks that play an elementary role in DevOps.

Infosys
be more

Brief Introduction to DevOps

Scope

DevOps is a practice of optimizing development and operations activities through structured processes, automation, and collaboration. It aims to synergize processes between development and operations teams to make them more efficient.

Key factors that drive efficiencies

The main factors that enable the development of effective DevOps are listed below:

- **Continuous and iterative delivery:** Processes should enable iterative delivery such that business capabilities are delivered in iterations and each iteration is thoroughly tested.
- **Strong collaboration:** All teams should successfuly collaborate during development, testing, and release managment activities.
- **Automation:** A majority of release managment activities such as development, static code analysis, testing (unit, functional, integration, load, performance), and deployment should be automated using tools and scripts. This would greatly enhance team productivity and improve the quality of the deliverable.
- **Early and iterative testing:** Each release should be tested iteratively in early stages. This would reduce the defect rate and risk involved in regression testing.
- **Continous integration (CI):** Carrying out frequent, integrated builds from a centralized source control system is key. Integrations with enterprise interfaces should be done iteratively to reduce integration risks.
- **Robust source control processes:** Processes related to source control management (such as check in, check out, locking, etc.) should enable geographically distributed teams to collaborate successfully.
- **Governance:** Standard set of well-defined processes should be established for release management and deployment.
- **Metrics:** Suitable metrics and KPIs to track the impact and success of DevOps processes should be defined. These metrics could include overall release time, time per release, etc.
- **Consistent and standard processes:** Common goals, SLAs, tools, uniform policies, and well-defined processes for DevOps activities should be established.
- **Agility:** All processes related to development, integration, testing, and release and deployment should be agile so that it is easy to absorb, test, and deploy changes.

Challenges for successful DevOps

Key challenges that come in the way of successful DevOps are listed below:

- **Inconsistent release management processes:** If various development teams have distinct release management processes, then we would not be able to fully realize the potential of DevOps. We need to establish a standard set of processes across the board
- **Lack of team collaboration:** Culture differences, lack of collaborative ecosystem would come in the way of a DevOps success. Each team having different metrics, policies would come in the way of successful DevOps
- **Technology ecosystem:** In some scenarios the tools, frameworks, technology, and infrastructure components used may not be mutually compatible. This poses challenges during integration and comes in the way of setting up a standard infrastructure.
- **Non-standard tracking metrics:** If various teams involved adopt different goals and SLAs, or if the processes are not agile in nature, it would not be possible to establish a standard DevOps governance.

Implementing a successful DevOps setup

Let us look at the various steps in implementing successful DevOps for an organization. In the process, we will also understand various phases in the DevOps transformation journey. The diagram below depicts the key phases of a DevOps implementation program.

	Assessment phase	Implementation phase	Monitoring and maintenance
Activities	• Review current processes • Gap analysis • Analysis of current challenges and pain points in release management, build, and deployment processes • Identification of opportunities for continuous improvement and agile delivery	• Standardize processes, metrics, and SLAs • Automate the build, release, and deployment processes using tools, frameworks, and scripts • Adopt agile delivery model • Adopt DevOps best practices such as continuous integration	• Continuously monitor and measure the metrics and SLAs • Implement continuous improvement framework
Key Outcomes	• Identification of automation opportunities and process improvement opportunities • Identification of tools and scripts for automation • Definition of common metrics, processes, goals, and SLAs • Defining continuous improvement plans • Defining DevOps roadmap	• Execution of PoCs and pilot setups to assess feasibility and improvements • Automated build, release, and deployment setup • Reports related to build, code quality, release, and deployment • Establishment of DevOps dashboard	• Metrics-based monitoring • Adoption of continuous improvement framework • Adoption of DevOps across enterprise-wide programs • Establishment of centralized knowledge repository

Essentially the transformation journey to a successful DevOps consists of three phases:

Assessment

During this phase we assess the current state of the organization, its capabilities, processes, and technologies with respect to development and operations. We focus on identifying existing challenges and gaps, and identifying opportunities for automation to create standardization. Therefore, we lay out the DevOps roadmap for future phases by identifying all the tools, technologies, and frameworks needed for automation and for implementing various processes.

Implementation

We begin this phase with feasibility and effectiveness checks through proof-of-concepts (PoCs) and pilot runs to understand the impact of implementation. Once these checks give us the green signal, we roll out process improvements and automate processes that have been identified in the assessment phase. Both development and operations teams work in collaboration using newly defined processes and best practices such as agile delivery model (to deliver in iterations), continuous integration, and more. At the end of this phase, we set up a comprehensive DevOps dashboard to monitor various activities and overall condition of the project.

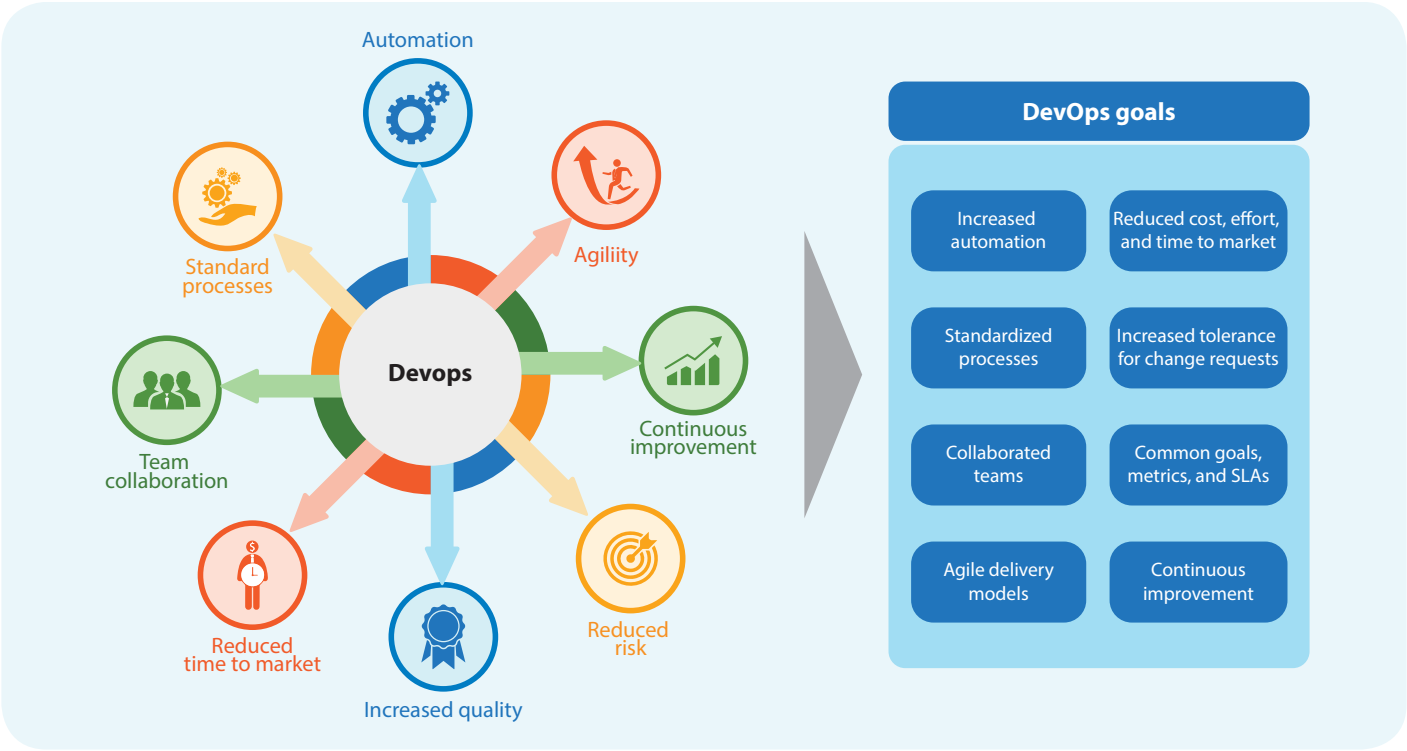
Monitoring and maintenance

This is an ongoing phase wherein we regularly monitor the defined SLAs and KPIs of DevOps processes using continuous improvement plans and governance models.

The table below lists typical transformations that occur post successful DevOps implementation:

Category	Before implementation	After implementation
Infrastructure	Nonstandard, disjointed, and fragmented	Organized and standardized technology stack
	Manual infrastructure setup and provisioning	Automated, on-demand, infrastructure provisioning with metrics-based monitoring tools
Teams	Development and operations teams with different goals and processes	Development and operations team working as a single global team with common set of goals, metrics, and processes.
	Each team spends considerable, manual effort to execute, develop, test, and release management activities.	Increased collaboration across all teams using automated processes and consistent goals. This leads to increased team productivity and lowered operations cost.
Delivery model	Big bang or waterfall model	Agile and iterative delivery model based on user stories leading to shorter time to market
	Longer release cycles	Incremental releases
Development and testing processes	Traditional	Iterative / agile development and testing with incremental releases
	Manual	Automated testing and continuous validation
	Costly and error prone	Reduced cost and risk due to continuous integration and testing
Integration model	In advanced project phases only	Continuous and frequent
Effectiveness of end user feedback and change requests	Inability to overcome challenges in handling change requests and enhancements leads to longer change implementation period	Higher effectiveness of change requests, feedback, and enhancements due to agile delivery

The key tenets and goals of successful DevOps are depicted in the following diagram:



Metrics and tools

Metrics

We could use the following metrics to monitor and track the effectiveness of DevOps processes:

- % reduction in overall release time
- % reduction in defects detected in UAT / preproduction testing
- % reduction in manual effort for overall release management
- % reduction in change / enhancement implementation time
- % increase in automation of testing, static code analysis, and deployment
- % increase in code coverage
- % increase in testing automation
- % increase in team productivity
- % increase in overall release quality

Tools

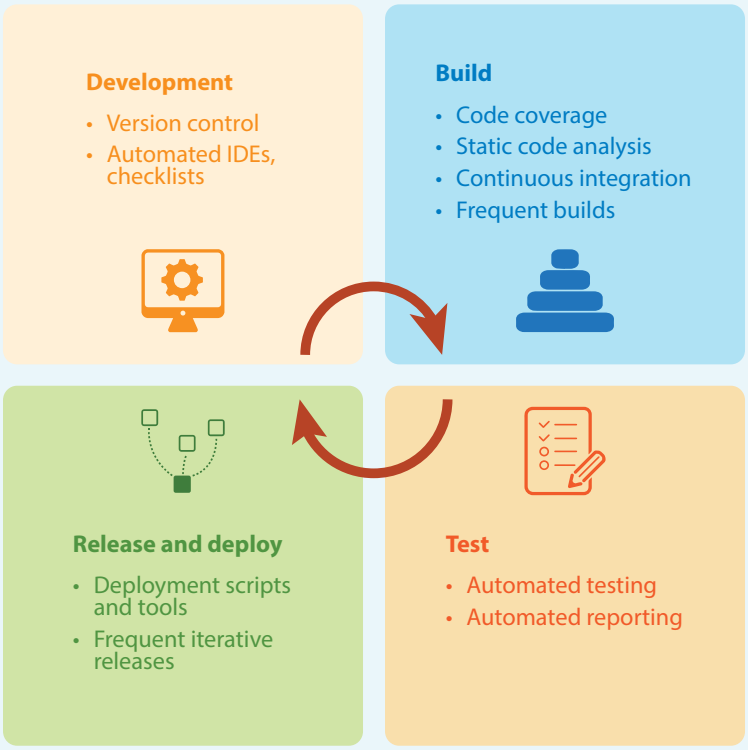
Listed below are various tools that are mainly used for automating processes at various phases of the software development life cycle (SDLC).

Project life cycle phase	Purpose	Tools that can be used
Development	Source control management	Git, SVN (Apache SubVersion), CVS (Concurrent Versions System)
	Automated static code analysis	Checkstyle, PMD, FindBugs, SonarQube
	Implementation of continuous integration	Jenkins, Anthill, Hudson, Cruise Control, Puppet

Testing	Unit, performance, web, and services testing	Junit, TestNG, JMeter, Selenium, Cucumber, HtmlUnit, SOAPUI
	Code coverage	Jacaco, Cobertura
Release	Build and release activities	ANT (Another Neat Tool), Maven, Gradle
Deployment	Automation of deployment activity	Custom deployment scripts, file copy scripts, deployment plugins for Jenkins / CI tools
Monitoring and maintenance	Continuously monitor the application and server environment post application deployment	Web analytics scripts, Gomez, application health-check monitoring tools, server monitoring tools, real-time user monitoring tools

Continuous Integration (CI) – A key process

CI is a software development practice which involves building, integrating, and testing software components continuously on an iterative basis. It detects defects early on, and also reduces the risk of low quality for the overall project. A sample of CI activities at various phases is shown below:

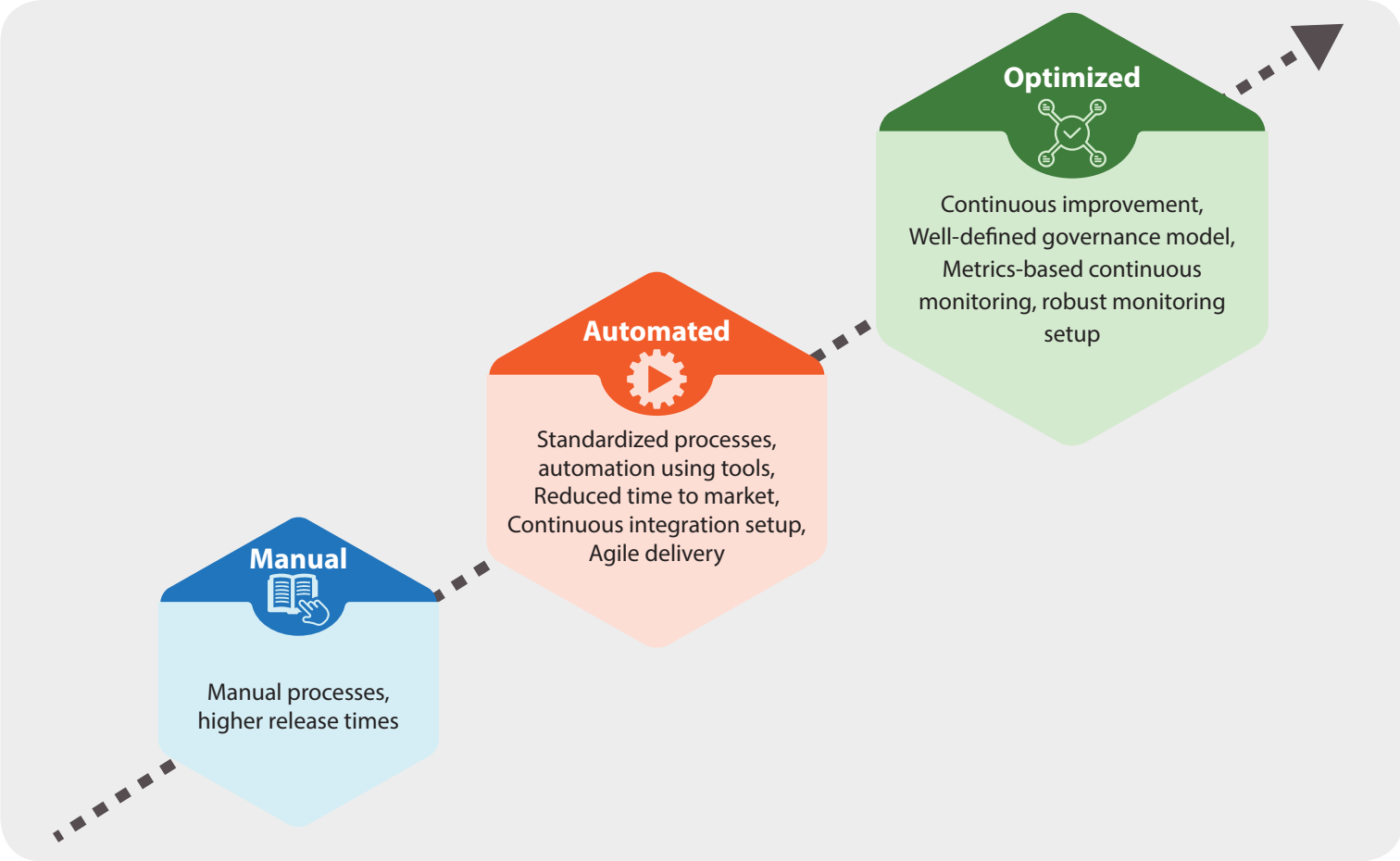


CI best practices

- Use a centralized source control system to maintain all code and project artefacts
- Build, test, and integrate early in the project life cycle
- Automate build, test, and deployment activities
- Adopt test-driven development (TDD) practices
- Adopt continous integration and deployment

DevOps roadmap

A sample DevOps roadmap is given below:



We initially start with an existing system which uses manual processes, traditional release, and deployment activities. Using the three-step DevOps implementation process, we would identify tasks and processes which can be automated. In the 'Automated' phase, we create a standard set of processes to manage development and release activities. In the 'Optimized' phase, we create a truly self-service environment with a continuous improvement model and unified processes.

Emerging trends

In this section we examine some of the emerging trends in the DevOps space.

Docker

Docker provides lightweight, independent, and reusable containers which package development units and their dependencies to achieve self-sufficiency. Using container-based virtualization, it builds, distributes, and ports images to various environments. A great advantage of Docker is that its environment can be quickly set up and tested rapidly. At the same time, it eases development, testing, and various other release management activities because of its open source nature.

How does Docker help?

Let us understand the role Docker plays in addressing some of the DevOps challenges:

Traditional challenges	Docker-based DevOps
Development team and operations team need to ensure the availability of all required libraries, system privileges, and permissions on all environments (Dev, SIT, UAT, etc.) Besides this, other interfaces such as database, services, middleware, etc., should also be installed and configured. This often takes huge manual effort and is error prone. This approach also entails that all configuration changes are correctly propagated across environments	Docker containers ensure the unit can be seamlessly shifted from one environment to another as an image. They provide deployment flexibility with optimal deployment cost
Infrastructure team needs to ensure availability of proper elements in all environments	Infrastructure is managed by Docker containers
In many cases, the final build is installed in the target environment manually using release notes and deploy instructions. This manual effort increases the overall release time	Docker image can be used to easily port application build from one environment to another without any additional effort
Additional effort is needed to set up and configure disaster recovery (DR) environment	Docker image can be reused for DR setup as well

DevOps as an enabler for cloud adoption

DevOps enables enterprises to standardize development and operational processes, automate deployment activities, and ease the migration of applications to cloud. Therefore, businesses could use DevOps as a key component in the digital transformation journey.

DevOps as an essential for emerging technologies

Emerging technologies such as big data, unstructured data processing, and mobile-enabled application would require the standard set of DevOps processes. At the same time, micro-services-based applications, API-based integration, and agile delivery also rely on a robust DevOps setup. Enterprises can leverage DevOps tools and container virtualization features to successfully implement a platform based on emerging technologies.

Case study

Let us look at a case study on successfully implementing DevOps.

Background

This case study describes the improvement in release management and deployment activities for the IT department of a retail organization. The IT unit developed various applications, but due to the complex landscape of IT infrastructure, the release involved build and deployment of multiple applications and its dependencies.

Release management challenges:

- **Nonuniform processes:** Various development teams used disparate tools and modes for build and deployment. While some teams built project artefacts (such as .war file and .ear file) from IDEs, some other teams used ANT scripts and a few others used Maven scripts. Due to this variance the release and deployment process varied across projects
- **Manual activities:** Many of the activities such as file upload, release labeling, and code packaging were done manually. Even deployment was done manually using a release document. As a result, each release encountered regression issues during deployment
- **Absence of automated validation:** Most of the testing activities were done manually. This further impacted release timelines
- **Lengthy release times:** Due to the above factors, production releases, even for a small enhancement, would take, on an average, about six hours
- **Absence of health check reporting:** There was no system to automatically report build failures, build quality, code coverage, test case execution status, etc.

Implementation

In order to address the above challenges, DevOps processes were set up which fine-tuned processes as given below:

- Build and deployment processes across all projects were made consistent through Maven scripts. This also led to reuse of existing scripts
- Jenkins CI was used as the continuous integration tool to develop a robust deployment framework. The CI tool reduced many of the manual activities such as manual build, manual testing, manual packaging, etc.
- Build and testing was automated using Jenkins plugins. Junit and Selenium frameworks were used to automate unit and web testing
- Jenkins dashboard was used as a unified project dashboard to monitor project status, build failures, code coverage, etc.
- Notification plugin was used to alert the administrator in case of build failures
- Automation and continous integration reduced the average production release time by 30 minutes

About the Author

Shailesh Kumar Shivakumar

Senior Technology Architect, Digital Practice , Infosys

Shailesh Kumar Shivakumar has over 14 years of industry experience. His areas of expertise include enterprise Java, portal technologies, web technologies, and performance engineering. He has published two books related to enterprise web architecture, enterprise portals, and User Experience Platform. He also has four patent applications and has published several papers and presented talks in IEEE conferences in the areas of web technologies and performance engineering. He has successfully lead several large-scale enterprise engagements for Fortune 500 clients.

He can be reached at **shailesh_shivakumar@infosys.com**

For more information, contact askus@infosys.com

Infosys be more