DevOps for Developers:

Building a Case for Your Organization

Capgemini | FLEXAGON

Table of Contents

Preface

Why did we write this guide?

As a developer, you may have found yourself drudging through slow, tedious manual tasks and getting called at all hours, disrupting your plans at short notice to help deploy code releases. Organizations demand more and more from their developers as you are the backbone of software changes, driving innovation from the most basic level through to end users.

More is required from you as companies face increasing competitive pressure. However, you're stuck in the trenches with these time-consuming tasks and unable to offer the true value of your software development skills. Budget constraints often shut down the possibility of investing in automation tools that could provide the help you need.

Luckily, there is a solution!

DevOps is the answer to these continuous struggles for many companies. In fact, adoption of DevOps has skyrocketed in recent years. 74% of organizations have adopted DevOps to some degree, according to Redhat's 2021 State of Database DevOps report.

In this eBook, we will explore how you can implement DevOps at your organization and eliminate the pesky and time-consuming manual tasks that overrun your day, so you can take back your time to focus on what you love: developing, improving the user experience, and providing value.

Leadership is often skeptical of DevOps because of its "hipster" hype, the initial cost, and the likelihood (or unlikelihood) of achieving the many benefits touted. We'll show you how to appeal to leadership with language they understand and with metrics that appeal to them.

How do we know how to help?

Capgemini is one of the largest global consultancies delivering services from ideation to design, build, deliver, maintain and operate. Sometimes our customers want us to work with specific processes and tools, but when the customer is looking for guidance we elect to adopt a DevOps model and a preferred set of go-to tools. Flexagon's FlexDeploy is one such tool. Capgemini has been working with Flexagon for several years now to enable the adoption of DevOps to deliver value to our clients.

Flexagon is a leading provider of continuous delivery and release automation software. Flexagon's DevOps platform, FlexDeploy, helps businesses develop, deliver, and manage software solutions faster and with improved quality, cost, and risk. Since the inception of Flexagon, the focus has been on the developer experience. Previously, commercial DevOps tools were too complex and supported only the largest enterprise needs. FlexDeploy revolutionized the commercial DevOps space and allowed organizations of any size to reap the benefits of build automation, deployment automation, and release orchestration. In 2020 Flexagon was recognized as a CDRA Leader by Forrester Research.

Through our successful partnership, Capgemini and Flexagon have come together to create this eBook and succinctly outline how to successfully adopt a DevOps practice to achieve the benefits promised by DevOps evangelists. The implementation process is not simply licensing a product but requires a number of techniques, key organizational changes, and buy-in from leadership.

Let's explore how you can ensure DevOps is successfully adopted at your organization.

Part 1

DevOps Basics

Impossible Expectations

Running a successful business that delivers products and services is impossible without technology. Most enterprises have a wide and varied technology landscape to deliver value and respond to customers' needs.

Managing these tools soon becomes a cumbersome task for you as a developer. Manual tasks and scripting are required for each update and change. Maintaining efficient systems becomes even more difficult when you have to coordinate with other developers and various teams in the IT department.

Despite your increasing list of to-do's, the organization demands greater innovation and productivity from the IT department. They want the software development teams to deliver high-quality outputs even more frequently so they can be competitive.

You may have proposed expanding the team or purchasing a tool to help with the workload, but there are strict budgets in place that shut down these ideas. Without additional personnel or tools to assist with development, the IT team is stretched thinner, more out-of-hours work is required, and heavy workloads and tight timelines result in mistakes, which causes outages and even more headaches.

Every organization is striving to develop and deliver software in a quick, high-quality, and cost-effective manner. The goal is to wow customers.

Is it possible to develop high-quality software faster and more repeatedly, with lower cost and less risk?

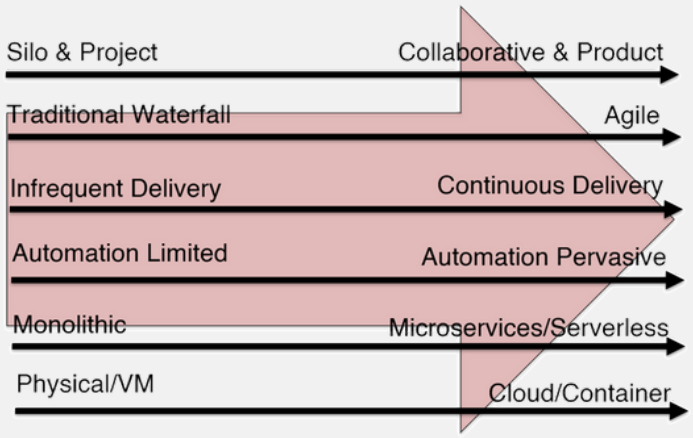
Meet DevOps

Companies turn to DevOps and automation solutions to solve these challenges. These solutions help them achieve speed at scale, whilst improving quality and more effectively managing the costs and risks associated with software delivery.

DevOps and automation help teams complete their extensive to-do lists while maintaining quality— all within business hours. Furthermore, with the tedious manual tasks out of the way, you have more time to focus on the work you enjoy: developing!

Each organization is somewhere along its Agile and automation journey, which has traditional waterfall methodologies and limited automation at one end of the spectrum and agile practices and extensive automation on the other. To be successful, organizations must continue to evolve their processes and enable the use of technology. The next step for you may be an evolution toward agile and automated processes. Maybe your next steps is to increase the rate at which new capabilities are delivered. Perhaps it is more general, like adopting new technologies, such as cloud, microservices, serverless, or containers.

> At the end of this book, there is a list of Terms and Definitions. Reading those short explanations before continuing may be helpful to you. Please reference it whenever you need a reminder or clarification.



Optimizing DevOps within the enterprise requires both the flexibility to leverage a broad range of tools across the toolchain while standardizing the end-to-end software delivery process. Many organizations want to consolidate tools to streamline processes and allow cross-team sharing. Orchestrating the tools in an efficient, repeatable, auditable way is a must-have for establishing enterprise scale DevOps and continuous delivery practices.

A mature DevOps practice has a plethora of benefits for developers, operations, release managers, QA, system administrators, product managers, test engineers, and every other role involved in software development and delivery. Even more importantly, DevOps improves both products/services and experiences for end users.

After adopting DevOps, companies enjoy:

- Automated, integrated, and repeatable delivery processes from provisioning and configuration management through the build/deploy/test/release process
- Efficiency driven by simplified processes
- Significant reductions in scripting and manual tasks
- More frequent, less costly deliveries with fewer errors and outages
- Visibility and continuous feedback across all aspects of solution development and delivery
- Easier and faster troubleshooting
- Compliance and audit effectiveness
- Simplicity of a single DevOps platform to manage the entire toolchain
- Reduction in costs to implement and support the overall software delivery lifecycle

These positive results explain why DevOps has gained prominence and why adoption rates are increasing.

Despite these benefits and the growing popularity of DevOps, your organization may not be adopting these practices. Why not? Sometimes the concept is still undiscovered. Perhaps the issue of budget is holding back your department. Maybe leadership is too stuck in their ways to implement these “cutting edge” or “hipster” ideas.

Getting leadership on board is necessary to implement and practice DevOps. The benefits of DevOps may be most immediately obvious to the developer, but the cultural change to capitalize on it depends on management supporting it with investment and leadership.

Before we dive into how to sell DevOps to leadership, it is important to first understand the various aspects of DevOps, including the key benefits, core principles, and critical changes required.

Getting leadership on board is necessary
to implement and practice DevOps.

The Benefits of DevOps

At its essence, the benefits of DevOps can be summarized as:

- Delivering more with less
- Delivering more frequently with improved quality and stability
- Meeting customer needs in a more timely manner with the right features

These points might seem vague, but the many benefits all fit into one or more of these few categories. So let's unpack these benefits to see how they are realized.

But let's be brutally honest, if you have an established history of working in other ways, you need to start the DevOps journey and potentially undergo some changes. During these changes it is likely you won't see these benefits. In fact, you may see things get worse for a brief period. This is no different to the concepts of team building as you pass through the stages of forming, storming, norming and finally performing, as Bruce Tuckman described in the 1960s. But just like the five stages of team development, moving through the process generates tremendous outcomes.

Delivering more with less

A significant aspect of the DevOps journey is the introduction of automation, from coding through to production deployment. This acts to make the spotting and preventing of bugs faster and easier. This saves time. Although it may not feel like that, if there is presently low code coverage or a backlog of unit and other automated testing is being addressed to drive up coverage. But this also increases confidence in software quality as it passes through processes going to production. Each time a release cycle is started the need to build that assurance up through manual or semi-manual testing contracts.

If the effort required to release software decreases, releases can take place more frequently. If you don't change the release tempo, then you've freed up people's time to do other things; To put it another way: you're doing more with less. Of course, if that spare capacity can be invested into increasing the automation, then you'll see a virtuous cycle developing, where more resources are available to drive more improvement.

> Both the Dev and Ops team should have an active stake in ensuring software is operationally easy to manage and is unlikely to need urgent intervention.

Both the Dev and Ops team should have an active stake in ensuring software is operationally easy to manage and is unlikely to need urgent intervention. Let's face it, no one likes being woken up at strange hours to sort out a work problem. When you have 'skin in the game' you are far more likely to incorporate preventative measures, or at least make it very easy for first line support to understand and run rectifying actions. Not to mention time spent on Ops tasks and fixing problems is time away from the more interesting activities of feature development.

Delivering more frequently with improved quality and stability

With more agile processes, batches of code are smaller and are released more frequently. Any errors or issues are easily identified and can be more quickly solved when releases are smaller, versus the larger releases common among organizations with traditional processes.

Any errors or issues are easily identified and can be more quickly solved when releases are smaller...

Meeting customer needs in a more timely manner

The best proof that software is working and doing what is actually needed is to expose it to the real world. That means getting users to try things out as early as possible, both for functional and nonfunctional assessment (particularly for user experience). Even without user involvement, such as report generators, AI and ML, having software in production helps us understand whether the solution is on target or course adjustments are needed.

The best proof that software is working and doing what is actually needed is to expose it to the real world.

The importance of customer engagement cannot be overstated. The ability to turn around changes for the users to see adjustments reflecting their feedback reinforces their engagement. This positive relationship will see you through any moments of difficulty.

To achieve all of this you must have the ability to turn around software quickly (continuously) and with high quality. This enables feedback with lower risk and ultimately ensures the right features are being provided.

The Core DevOps Principles

Before you propose DevOps to your organization, you must know what you are proposing. The following core principles are the pillars on which DevOps stands and succeeds. There are technical aspects to implement (many of which are defined in the Term and Definitions section located at the end of this book), but a new mindset must also be adopted. In fact, this aspect is so important, DevOps is more commonly defined as a mindset than as a set of practices.

Let's look at 8 Core DevOps principles that are essential for implementing and maintaining a successful practice.

Automation

Automation is the foundation on which everything else is built. Automating repetitive, manual tasks is the primary goal of DevOps. It closely ties into other goals as well. For example, more frequent deliveries are made possible when automation pushes small batches of changes through the pipeline.

Automation is your biggest friend as it eliminates the tedious manual tasks from your workload.

Continuous Improvement

DevOps isn't a single switch you flip to take you from your before-DevOps phase to your after-DevOps phase. It's a journey. You start where you are today and take one step at a time. Prioritize your team's most pressing needs but never stop moving forward. This is continuous improvement, a VIP of the "Continuous Everything" club.

Shift Left

This term refers to the practice of focusing on quality in software development to prevent issues rather than detect them later in the process. If you picture the pipeline with Plan on the left and Production on the right, shifting left means running tests and quality checks as early in the process as possible. This prevents issues from popping up in Production.

Continuous testing and continuous deployment are key practices to successful shift left.

Configuration Management

Configuration management is the process by which an organization manages its various environments, configurations, and source code. The objective is to optimize the configuration of these environments so all the related resources - both hardware and software - are functioning properly and contributing to the success of your software development process.

Poor configuration management can result in some major issues including outages and data breaches.

Frequent Small Releases

Many of the horror stories told by companies of their pre-DevOps lives are the result of large, bulky releases. This process of deploying huge amounts of code which has been developed over a long period of time is often referred to as “merge hell.”

DevOps alleviates this issue through frequent small releases. Developers’ code is frequently updated to a shared source control tool through the process of continuous integration. These updates can be several times a day or any time a developer commits a change.

Frequent small releases allow development teams to commit and verify changes more often by automating builds and tests so any issues can be detected and resolved early. This increases the productivity of developers which is essential to delivering software faster and with higher quality.

Feedback Loops

Feedback enables informed decision making. When teams receive feedback from end users they can allocate their time and energy to the most important areas. This ensures that resources aren’t wasted on projects that won’t bring value to the organization’s stakeholders.

Remember: Focusing on the customer’s needs is critical to both the success of a DevOps practice and a business as a whole. Keep your customers in mind and allow this to guide your efforts.

Reporting and Measurements (KPIs)

Feedback comes from both end users and your systems. Visibility through the planning, release, and monitoring process and the ability to view both real-time and historic reports are integral to DevOps.

With this information you can improve decision making, streamline troubleshooting, adhere to audit compliance, and promote continuous improvement.

Collaboration and End-to-End Responsibility

Dev and Ops have a long running conflict rooted in their differing goals. Development wants to build software and integrate systems as quickly as possible. Operations teams want stability, security, and control, all adjectives not often associated with speed.

A successful DevOps practice fosters a collaborative culture at an organization, uniting Dev, Ops, and other teams in a way Romeo and Juliet never could with the Montague and Capulet families (and with notably fewer deaths). By focusing on common objectives, teams throughout the software development lifecycle become a larger team whose sum is greater than their parts.

As a result, both Dev and Ops are responsible for the project from beginning to end, or “cradle to grave.” There is no longer a handoff which causes inefficiencies and errors.

Collaboration and streamlined communication is arguably the most important aspect of DevOps, ensuring success both during the implementation and throughout maintenance. Let’s continue exploring the importance of cultural change management.

DevOps Culture

The adoption of DevOps principles, particularly the automation mindset, can be a huge challenge to an organization that has been working a particular way for a long time. The principles can be seen as a threat, and to some (such as manual test teams) a risk to jobs. To others it appears as a sudden expectation that they take on new and unfamiliar responsibilities, such as the blending of development and operations.

This can be further compounded for any organization that has strong compliance demands. They will likely find change even harder, as compliance and its auditability will be used as a case to resist change.

The heart of this isn't a technology issue, but that of organizational change. When we introduce a major new system such as an ERP to an organization, the need for training and transition for the business is understood as a distinct delivery stream.

The difficulty is that an IT change to an IT organization is not seen as the same level of change, yet for some organizations introducing DevOps is just as radical as a new ERP. In both cases, to get the impacted teams engaged with understanding the reason, you must show the potential benefits of the change for both the organization and individual in words they understand.

Commitment and Investment in Enabling DevOps

Over the years, we have seen organizations make decisions to align with Service Oriented Architecture (SOA), or "are going Agile." But within a year or so things are no better off, and - if you peel away the veneer - things haven't really changed.

The problem is that the adoption of these ideals require a level of faith, commitment, and investment. From an investment perspective it isn't simply money for everyone to have a nice training course. It is hiring seed resources who have battle scars to know where the unwitting traps may be and share that practical knowledge into a wider team. It is the willingness to address technical debt that can be a limitation. For example, automation of testing, building up the test coverage that will allow a realistic automated regression test for example. You need to ensure that in 6 months time you have a set of tools and know how and why it works.

The investment isn't a bottomless pit, but it needs to be there to get the process embedded and to smooth out the issues, whether that is a lack of automated test coverage or overhauling release processes. Eventually, the process will work like a gyroscope or a flywheel - become self-sustaining without external input.

Unlike the SOA and Agile cases, we should be able to more definitively link the investment to a dividend, the amount of debt cleared, the tooling investment and setup against code quality, and mean time to releases getting quicker.

Part 2

Implementing DevOps

Starting DevOps

Before you start a journey, you need to have a clear picture of what it is you're seeking to achieve. Trying to adopt DevOps because someone has heard it's a good idea can be problematic, if for no other reason, how can you show that you're making positive headway without knowing what success looks like? Is the goal about cost management, accelerating delivery cycles, and/or freeing up developers' time?

Groundwork

For any initiative to be truly successful laying the groundwork is essential. For DevOps - like most IT initiatives - the core technical processes and tooling need to be agreed upon and defined.

This should cover at least:

- Ensuring configuration management and the branching and releasing is agreed including how assets are marked with versions
- Sufficient tooling to manage:
  - Source, binary and deployable artifacts
  - Automation and orchestration of build and deployment processes
  - Tooling for automated testing and any quality metrics
- Agreed measurements to demonstrate progress and improvement

We would recommend establishing just enough, but setup so you can keep building on what has gone before, keeping an eye on the likely future increments so each step doesn't have any unnecessary work.

Aim for 80/20

As we've mentioned DevOps should be about incremental steps and continuous improvement. If you're looking to introduce DevOps with a greenfield development program then choose part of the program that is pretty typical of the development needed. While this won't provide any comparative anecdotal evidence to the benefits, there is the opportunity to show how doing things early on (i.e., shifting left) contributes to the overall benefit.

In a brownfield development it's also worth choosing something that is typical of a lot of the overall solution development. Ideally this will also be something that has proven to be a source of bugs or regular change. Doing this will help at the very least provide anecdotal evidence of making things faster and easier. If there are pre-DevOps measures on delivery then even better(!), as a level of comparative value can be achieved.

The next step is to grow the adoption in two directions: breadth and depth. Extend the breadth of coverage with your initial steps, particularly if there is organizational resistance to change. Start by implementing practices with the development teams so they can experience the benefits of DevOps. These teams benefiting from the new approach are the best advocates into the rest of the organization.

Depth is also essential. You may have started with automated build and deployment along with successful unit testing. So look to create and automate larger tests, but with the spirit of shift left. Tooling on code quality will return benefits - flagging the likely source of bugs (complexity checks) and test coverage.

There is one very important consideration here. It is very tempting to use these tools and tell development teams to achieve near perfect code with 100% test coverage from now on. This can really backfire in several ways…

- Suddenly delivery slows down, despite faster delivery being touted as a DevOps benefit
- Development bad practices take place, such as stubbing unit tests that return successful results regardless of the reality
- Quality rules are modified in configuration to show positive outcomes, but the measures are so diluted they become meaningless

The begs the question: How do I improve the state of play?

The simple answer is pragmatically and progressively. The 80/20 principle tells us that the last 20% of work will cost us 80% of the time. So where does the cost/benefit start to breakdown (assuming no one subverts the processes to get the scores right)? For aircraft software - that cost benefit probably stops in the last 0.001%. But for a simple document editor, that threshold will be far lower. Bugs are annoying but how much will our users tolerate?

Firstly, in a brownfield you can't fail your existing code. It’s out there being used. So the question is: What’s wrong with it if it works?

The way forward is to start by setting the criteria so warnings are generated. Then incentivize the teams to shrink the warnings. After a ‘grace’ period move the warnings to build fails, by the time you’re doing this, the checks set as warnings are the exception. But time must be given for the improvements to be made.

Selling to Leadership

The drive for DevOps adoption can form within the development team as you can see and feel the benefits of such adoption, such as automation eliminating hand cranking tasks like build, deploy, and testing. You’re in the trenches everyday and know what processes would make your work more efficient and beneficial to the organization.

You focus on technology, while the leadership focuses on team productivity, outputs, quality, and expenses.

The challenge is convincing leadership to invest in the tooling and address technical debt, which can be perceived as not an issue as it is against functionality that is already in production. Adopt the right language and appeal to leadership with metrics and benefits that resonate with them.

Here are some benefits leadership will appreciate:

- With full automation, the development team is more productive, operating faster and more effectively to solve business problems.
- We can achieve speed at scale WITH quality all while managing cost and risk.
- By forming a cross-functional team we can achieve far more than we ever could as individual team members.
- We can deliver changes frequently and quickly respond to customer needs, which means we please more end users.
- With the time we save, we can focus on innovation and become increasingly competitive. We can do more with the same size team.

The best approach is to collect metrics for performance prior to adopting the DevOps model. These metrics are best when you can see details that easily align to organizational goals. For example, speed to deliver features or the ability to identify and react to software supply chain issues reducing the chances of becoming the next organization under scrutiny for a 3rd party library security exposure.

With these metrics, a pilot adoption with a basic setup and the same metrics can be run. The outcome should provide statistics showing the gains. This can be extrapolated to show wider adoption benefits. These benefits, when closely aligned to financial benefits, need to be tempered with clarity on the need to address existing debt.

Being clear about the indirect benefits can also be beneficial, particularly if they can be linked to undesirable issues or events that have occurred. This can range from customer complaints regarding bugs, to missed delivery commitments, to avoiding software supply chain issues due to security breaches.

Other arguments for DevOps worth considering:

- Dashboards and reports provide visibility and transparency that increase confidence in project status and risks. Automated processes and an integrated toolchain capture the necessary data and metrics.
- Retention of staff by modernizing skills.
- Better reliability in delivery timelines and easier to deliver smaller increments or urgent bug fixes to customers. Generally, it provides the ability to be more reactive to customers.

Organizational Change

Pitching the organizational aspect of DevOps can be tricky as there can be issues around authority, ownership, and influence involved. When these are challenges, it is best to increase collaboration between Development and Operations so both sides gain influence. The goal is to get operational insight and practices supported earlier.

Ultimately Dev and Ops should become a unified team. This may not be practical. If that is the case, try to get Ops staff embedded into development cycles and Dev teams experiencing Ops issues (rotations in the Ops team, or becoming second line support including call outs).

The gains achieved through these overlaps that will get attention are around how the team dynamics can change:

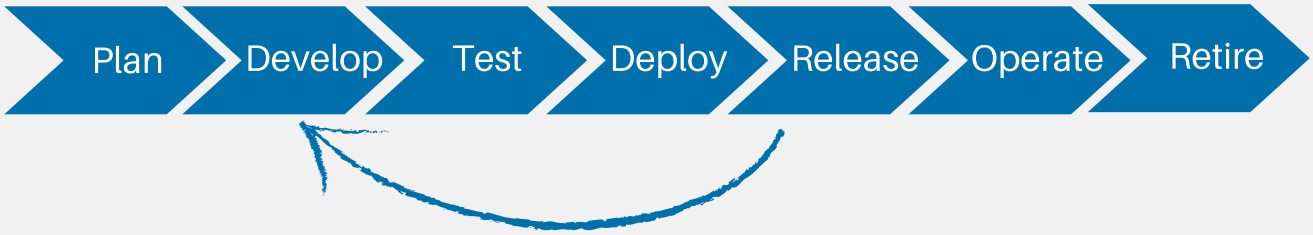
- More effective knowledge transfer between Dev and Ops. Let's be honest, development teams aren't fans of documentation, so active collaboration will increase the sharing of what is not written.
- The 'throw it over the fence' behaviors that occur between teams under pressure and ultimately are more costly in time loss overall are less likely as the Dev team are likely to experience pain from such behavior.
- In addition to not throwing things over the fence, the likelihood of blame culture decreases and the team becomes more collective.
- Appreciation of operational needs will lead to improved operations and likely reduce support effort and more critically shrink time from fault to resolution.

The Phases of a DevOps Pipeline

DevOps pipelines vary from one organization to another. Although all pipelines begin with a plan and end with the deployment and monitoring of changes, the number and names of phases are defined by teams.

Although pipelines look different, they have the same goal: segment the software development and delivery process to simplify planning and streamline execution.

Let's review seven phases often used:



Plan

The Plan phase is driven by a Product/Project Manager and supported by the team. They create a roadmap to guide the development of the product/project. The requirements for the content is based both on internal feedback and customer requests.

The Product/Project Manager writes this roadmap including identifying steps, including Epics, Features, and User Stories. This planning is typically documented in an agile project/ticket management system which includes timelines, a process tracker, and milestone markers. The plan is iterative and broken in smaller deliverables that are developed, tested, and delivered into the hands of the end users.

Develop

This step is sometimes referred to as the Code phase or the Build phase. During this time, developers begin developing and committing their code to a shared repository.

Teams use a variety of tools and plugins to assist with this process ensuring code quality and collaboration. Some common tools include source control management, sandboxes, and frameworks.

Organizations practicing DevOps typically use continuous integration, which means code is frequently updated to the shared source control tool. These frequent updates - occurring via pull requests and code reviews - and automated unit tests minimizes integration issues. Any errors that do occur can be quickly identified and resolved because they are part of a small batch of code.

Test

The Test phase is iterative and is where the code is deployed to the staging environments for further testing. Automation begins to shine in this phase as the code is automatically moved to the staging environments where both automated and manual tests take place. All tests are determined and set up by each organization to fit their unique needs.

There are multiple types of tests that you can include in this phase. Some tests are part of a more traditional software development life cycle (SDLC), such as unit testing, integration testing, system testing, and acceptance testing. You should also consider other types of tests such as static code analysis, software composition analysis, web vulnerability scanning, and container security scanning.

Deploy

Deployments occur across the test stages iteratively, including at the point the various build artifacts are deployed into Production. Thanks to the last two phases, teams are confident that the release will work seamlessly in Prod. This means the deployment process can be automated.

This is the phase where deployment strategies are implemented as best practice.

Release

At the Release phase, teams decide whether the changes are ready for deployment into the Production environment or need further refinement. At this point, the code has passed the tests in the previous phase and the operations team can rule any breakages in Prod unlikely.

The majority of organizations opt for a regular release cadence and choose to manually approve releases. Additionally, there may be a dedicated Release Manager overseeing this phase.

Organizations with the most mature DevOps practices automate deployments via continuous deployment. This maturity results in multiple deployments of products every day, which is often considered DevOps nirvana.

Operate (and Monitor)

The changes are now live and the team enters the final phase. This phase is called Operate by some organizations, others call it Monitor, and some break it down into two or more concurrent phases phases.

End users are using and interacting with changes. The Operations team is checking that the new changes are running smoothly. Teams are collecting end user feedback and other generated data. This information must be communicated back to the product/project manager and the Development team so they can incorporate it into future changes.

Customers are the best source of feedback and will assuredly find any errors or weak areas. Incorporating feedback into releases ensures happy and heard customers and an optimized product.

From a tooling perspective, this is where Application Performance Monitoring (APM) comes in. Tools like Splunk, New Relic, and Datadog are used to monitor what is going on in a production environment and to facilitate the feedback lifecycle. Some of this is information relates to the user experience (UX) and some is more operational in nature. For example, teams use APM to proactively identify performance issues and address these prior to an end user impact.

Retire (optional)

Sometimes a team will want to remove a release from production. This could be because the system release is redundant, needs to be replaced, or has become obsolete. Endless growth of code drives up costs, so IT departments can decrease expenses by stopping builds of components no longer needed and removing code that would otherwise need to be tested.

The process of purging data no longer needed (or undeploying) is completed in the Retirement phase.

There are several tools on the market that can be used to undeploy and to freeze a release for a period of time before undeployment.

Applying 'Continuous' to Your DevOps Pipeline

Becoming agile and maturing your SDLC means involving more and more ‘continuous’ into your processes. This allows you to be responsive, quickly receiving feedback and making changes.

Let’s look at several aspects of ‘continuous’ that you should incorporate into your DevOps pipeline.

Continuous Planning

Continuous planning eliminates the predefined monthly or annually planning for releases. Instead, planning development and releases is part of everything you do. You no longer sit down to brainstorm and plan, it is now closely intertwined with everyday activities. This means plans are up-to-date and accurate.

Continuous Development

Continuous development takes normal development and transforms it into an iterating cycle. Releases provide new functionality, improvements, patches for bugs, updates to address environmental changes, and so on.

As a result, the development process needs to accommodate multiple development paths through code and configuration branching allowing bug fixes to be implemented without disrupting feature releases. This may include retrofitting changes to earlier versions.

As modern development builds on many 3rd party dependencies, it may be necessary to incorporate into the process tooling to verify dependencies don’t present problems. For example, verifying that they don’t carry critical vulnerabilities that could be detrimental to the software being developed.

Continuous Integration

As the development processes are scaled up, the process of drawing through many developer changes every minute and hour needs to be automated. This builds upon the continuous development, but when changes occur in a specific branch you must test it. This process may include merging changes from different sources together and driving any necessary activities to help deconflict changes.

Continuous Testing

This moves the tests to a condition by which the different testing phases and outcomes are controlled by the choreography process. This can extend to management and resetting of external stubs, simulators or even real solutions.

The most valuable aspect of continuous testing is the ability to provide regression, so any new change does not break what has already gone before it. Through automation we can drive these activities every time changes are applied to the code base

Continuous Monitoring

For many organizations, monitoring means Ops personnel actively check the health of the system. Continuous monitoring transforms it from a reactive state to notification system - drawing Ops attention only when manual intervention is needed. This can extend to taking identified issues and generating issues using products such as ServiceNow and Jira.

Collecting performance data and feeding back to the development team allows analysis of volume use. Teams look for any problem patterns that can occur in distributed deployments.

Continuous monitoring may extend into areas such as capturing analytics of a production solution where A/B testing is used to determine if change is providing more value.

Continuous Deployment

In the most mature DevOps organizations the decision for promoting code changes into production doesn't involve any human intervention. Instead, software is controlled into production through the deployment orchestration processes. These changes are typically small incremental changes onto a known and controlled baseline, so the risk of a fault is minimized.

Continuous Operations

Continuous operations reflects the automation of processes, particularly tasks such as housekeeping (backup, outage management, etc). It involves scaling the environment up and down to meet the demand being identified by continuous monitoring.

This may include feeding information into audit trails and enterprise level configuration management tools.

Continuous Feedback

Continuous feedback isn't so much a stage of the cycle, but a way of life. At every point in the lifecycle we should be seeking feedback. Just like control systems are needed to determine what happens, test feedback determines if we can go to production and so on.

Feedback shouldn't be limited in the stages we have direct responsibility for, instead consider it for the Operate phase. This way we can better understand how software is used or where it performs below our desires. Additionally, the feedback helps the development team better understand what parts of the software are used most.

DevOps Best Practices

DevOps is not a set-it-and-forget-it process. There are various best practices you should adhere to in order to ensure a successful DevOps practice.

Implement Automation Wherever Possible

Many tasks are better when they are automated. Automation removes the tasks from someone's to-do list and eliminates the risk of human error, resulting in faster and more consistent development and delivery while freeing up time to focus on more important activities.

Take advantage of continuous integration, continuous delivery, and even continuous deployment. Although a streamlined and automated traditional software development and delivery process provide benefits, it further helps to break down the work into smaller and more manageable pieces and deliver the results to the end users faster. In addition to gathering the feedback from the end users more rapidly, the smaller and typically less complex changes are easier to develop and manage.

There are some tasks you may wish to complete manually such as edge conditions for testing to ensure quality or gathering insights to the end user's experience. However, automate what you can to remove waste and time from the process while improving the quality of deliverables.

Continue to shift left from problem detection to prevention. In addition to automating functional tests and moving them earlier in the process, also focus on areas such as security and performance. Identifying issues or potential issues earlier will reduce the risk and cost to get the deliverables into the end users hands at the quality levels you desire.

Take Advantage of Deployment Strategies

Deployment strategies are used to ensure deployments are high-quality and risks are managed. This is done through both processes and enabling technologies which ultimately get the capabilities into the end users' hands faster and enables feedback and validation in a more controlled fashion.

There are many deployment strategies such as A/B, Blue/Green, and Canary. Each recognizes that there are many types of changes and not all are the same standard. Applying such strategies means testing changes to verify the results in a live environment and executing them with confidence so you can adjust as needed. This process can be repeated via continuous experimentation - yet another 'continuous'.

Applying deployment strategies may not be the first tasks you undertake as you implement DevOps, but don't miss out on the benefits of applying more advanced forms of deployment as your DevOps implementation matures.

Incorporate Agile Project Management

Agile project management methodologies are commonplace today, including the supporting tools that help teams to plan, develop and focus on clear schedules, and follow through with software changes. Remember that DevOps and Agile require a cultural change which includes roles, processes, and tools that facilitate collaboration and promote an agile mindset.

Although Agile and DevOps are not one and the same, applying both helps optimize the outcomes. Plan projects with a simple set of steps, which can be as easy as: To Do, In Progress, In Review, and Complete. Break down large projects into small, manageable steps. This ensures that everything is covered without intimidating the team with one large looming task. It helps to tie your agile project management tickets to source control changes and ultimately to the release of changes to the end users. This traceability improves the insight which can be gathered down the road to assess the value of changes and also improve the ability to execute audits more effectively.

Agile methods aren't just for application development, and should also be considered for delivering and managing infrastructure resources on premises and in the cloud.

Never Stop Gathering Feedback

Feedback can come from internal and external users and from data. Establish processes for continuing to collect this feedback as it can guide your decision making by revealing top priorities, existing or looming issues and inefficiencies, UX optimizations, and more.

Comments and critiques from end users should be highly valued, as it is the core of customer-centricity. This feedback helps you prioritize what projects to undertake and what new features should be added to your products/services.

Apply AI/ML to support advanced feedback gathering and insights. Applying these approaches in concert with continuous delivery compounds the impact of fast feedback cycles.

Track Logs and Metrics

Access to and insights from data is essential to successful and efficient processes. This data can come in the form of logs, traces, and metrics captured throughout the development, delivery, and ongoing operations. Immediate access to technical information such as logs and traces can shorten the troubleshooting process and ability to restore service when a problem occurs. Capturing metrics for the frequency and quality of deployments ensures continuous learning from insight from this data throughout the value stream.

Automated alerts and notifications help provide immediate awareness of failures or potential issues which can be quickly identified and resolved. In some cases human intervention is required and in others the systems can be self-corrected to minimize or eliminate the impact.

Implement source control tools to document software versions and allow you to rollback to previous versions if necessary.

Ensure Security

Security is critical and teams must take the proper steps to protect against data and system breaches and vulnerabilities. This is where DevSecOps comes in, ensuring security is integrated throughout your software planning, development, delivery, and operations lifecycle.

There are many aspects of security which should be considered as part of your DevOps practices including role based segregation of duties, SSO, MFA, static code analysis, web vulnerability scanning, container scanning, secrets management, and many more.

DevOps KPIs

How do you know if DevOps is working for you? How do you track your maturity over time? What are the metrics you should be measuring?

DevOps KPIs shine a spotlight on the most important metrics for teams to evaluate when they are analyzing their DevOps practice. There are many KPIs, but here are 4 recognized by the DevOps Research and Assessment (DORA) team:

1. Deployment frequency
2. Lead time for changes
3. Change failure rate
4. Time to restore service

Deployment frequency indicates how often releases are successfully pushed to production. This can be weekly, daily, or any time a change is made. It can also either be on a stable schedule or increase as a DevOps practice matures.

Lead time for changes is the amount of time it takes for a change to move through the pipeline into the Production environment. In general this indicates the efficiency of the development process.

Change failure rate is the percentage of releases that result in outages or failures in Production.

Time to restore service (i.e., mean time to recovery or MTTR) shows how quickly (or slowly) an organization responds to and recovers from outages and failures in Production. This is one of the best known and common KPIs.

According to DORA, “At a high level, Deployment Frequency and Lead Time for Changes measure velocity, while Change Failure Rate and Time to Restore Service measure stability.”

Continue calculating these values and making improvements to streamline processes for the development and delivery teams and to achieve greater results for the organization as a whole.

Final Words

Implementing DevOps at your organization can be a daunting task with so many changes both to technology, processes, and people. Most often knowing how to build a case for leadership support is the biggest hurdle. However, it is possible!

Keep in mind that DevOps is a journey. Remember that this change will benefit you, the various IT teams, leadership, and end users. It is a worthwhile pursuit with great benefits along the journey. You won't receive every benefit right away and transformation doesn't happen overnight. Don't expect it and don't promise it.

Take one step at a time. Mature your processes and adopt more automation and tooling to facilitate the necessary changes.

When communicating to leadership, share the benefits that are most appealing to them, such as increased productivity, greater delivery frequency, enhanced agility to respond to customer needs and competitive shifts, and quick ROI.

Continue learning about DevOps and its benefits so you can formulate a strong case and get this transformation underway at your organization!

Interested in adopting DevOps at your organization?

Schedule time to talk to an expert. We can help you identify your DevOps needs and find a solution.

Talk to an Expert

Appendix

Resources and Definitions

Additional Resources

The best way to present the case for DevOps at your organization is to be educated on the topic. Our team of developers, architects, and leadership have compiled some of our favorite resources that we found informative and helped us throughout our careers.

Video: What is DevOps? - In Simple English

"Easy to understand and uses simple terms and does not require prior knowledge of DevOps terms."

~ Keith, Software Developer

Websites: DevOps.com, DZone, and the DevOps Institute

These websites have a wealth of resources, from articles to webinars. They are especially helpful if you are looking to learn more about a specific topic within DevOps.

Video: How Netflix Thinks of DevOps

"This one could be considered blasphemy in the tech world but it really goes into detail about how having the right DevOps practice in combination with fewer rules and regulations for developers enables more productivity and creativity."

~ Joel, Senior Software Engineer

Newsletter: DevOps'ish

This DevOps newsletter links to some great resources in various DevOps sectors, generally provided by experts in the field. It will keep you up-to-date on trends and other news in DevOps.

Paper: 6 Benefits of a DevOps Platform and Calculating ROI

This paper outlines 6 benefits of DevOps, including shortening the delivery lifecycle, reduce manual processes, and achieving ROI quickly.

Webinar: An Introduction to DevOps

This webinar explains the important aspects of DevOps. The goal of the webinar is how you can achieve the key benefits of DevOps, including those we outlined in this eBook.

eBook: Securing DevOps by Julien Vehent

"Provides both sound ideas and real-world examples. A must-read."

~ Adrien Saladin, PeopleDoc

Blog: Adventures in DevOps

This blog is written and run by Pushpa, a DevOps Engineer. She has a variety of book and articles available on her blog, covering a wide variety of technical topics.

Article: The Current State of DevSecOps Metrics

This Carnegie Mellon University article was written by Bill Nichols, who has a doctorate in physical and serves as a senior member of the technical staff at the Software Engineering Institute. This article dives into DevSecOps metrics that software development stakeholders should know about their projects and how you can improve your results.

Video: Design Microservice Architectures the Right Way

"DevOps is only a subsection of this video but the video goes into detail about how modern projects should be architected, everything from code generation, microservices to continuous integration and delivery."

~ Joel, Senior Software Engineer

Guide: Atlassian Git Tutorials

For any organization newly adopting Git, there isn't a more easily understandable and thorough set of tutorials than what Atlassian offers.

Article: A Successful Git Branching Model

"A good blog about branching strategy with helpful examples and clear illustrations."

~ Muthu, Senior Architect

Website: Baeldung

"A very good website with good examples for all java and web technologies. Explains the basic concept with example and also share source code for execution. Very useful"

~ Muthu, Senior Architect

Report: Annual DevOps Report from Puppet

"Puppet publishes a DevOps report every year which provides a broad perspective on the subject."

~ Dan Goerdt, President of Flexagon

Webinar: Companies Share Their Success with DevOps

Want to hear from companies about their journeys to DevOps? This webinar features three Flexagon customers who tell their stories and explain the results they now enjoy.

Important Terms and Definitions

DevOps

DevOps is the collaboration of Development, Operations, and QA to create a culture of communication, improve processes, and advance in technology and tools. This process includes three very important steps: Build with Continuous Integration, Deploy with Continuous Delivery, and Release with Release Orchestration.

It is important to note that there are many definitions for DevOps.

Agile

Agile is a methodology based on the practices first outlined in the book Agile Manifesto and its 12 principles. It is a mindset, rather than a set of strict protocol. Agile focuses on people, with fundamental values including teamwork, self-organization, and accountability.

Agile is frequently contrasted with the older waterfall methodology. It was developed to better deliver software, focusing on collaboration, communication, and constant change.

Additionally, an Agile mindset stresses testing, gathering feedback, and adjusting as a primary method to developing and delivering the best software. This means small batches of code are released frequently so feedback can be collected, and any corrections can be quickly made.

Continuous Integration

Continuous integration (CI) is the frequent updating of code to a shared source control tool (such as Git, SVN, TFS) and automated build and testing in a lower environment. This allows many different individuals or teams to work on a project simultaneously, ultimately speeding up processes and timelines. As long as the code is updated and centralized frequently, disparate teams have access to the most updated version of code, helping to eliminate conflicts and duplication of efforts.

Continuous Delivery

Continuous Delivery (CD) is the process of ensuring code changes can be safely deployed at any time. The goal is for developers to generate and move code through the lifecycle quickly and efficiently, allowing for rapid and regular delivery to customers. This means eliminating manual interventions and automating the process of building, testing, and pushing code changes to the next stage or environment.

Continuous delivery incorporates people (both in Development and Operations), process, and tooling to transform the way software is delivered, establishing repeatable, high quality, and cost-effective software solutions.

Adopting continuous delivery increases the velocity of the delivery and feedback lifecycle, while helping to manage the cost and risk of delivering and maintaining software-based solutions. Continuous delivery enables teams to cut release time and deliver features to market more frequently.

CI/CD Pipeline

A CI/CD pipeline is a series of stages that software must move through before being delivered to end users. Stages typically include Plan, Build, Test, Release, Deploy, and Monitor. However, these steps can vary from one organization to another.

CI/CD pipelines are fundamental components of DevOps and pervasive automation.

Continuous Deployment

Continuous deployment (also CD) is the process that moves releases that pass the tests in the Test stage of the pipeline through to the Production environment.

With continuous deployment, code changes can pass through the entire pipeline without human intervention. For this reason, this ‘continuous’ is typically reserved for organizations with mature DevOps practices that have strong quality checks in place.

Build Automation

Build automation is the process of creating and building software without direct human intervention. With build automation, tasks that were traditionally the responsibility of a developer are standardized, to become scripted, repeatable, automated steps to moving a new software forward to its final form.

Build automation is critical to DevOps, as it must be established before an organization can implement other DevOps processes, including continuous integration, continuous delivery, and continuous deployment.

Deployment Automation

Deployment automation is the act of deploying software to a test, or production environment, without the need for additional manual intervention or configuration. It is a critical step in mature DevOps – the automation helps to further streamline processes and improve the software delivery cycle, while acting as a critical support measure for continuous delivery.

Release Automation

Application Release Automation (ARA) is the process of packaging and deploying new and updated applications across environments in an automated fashion, without dealing with manual and scripted processes. It gives IT teams a consistent process that saves them time and headaches.

Release Orchestration

Release orchestration is the process of coordinating the tasks related to deploying software updates across environments and into the hands of end users – improving planning, tracking, and communication throughout the software release cycle across different teams, departments, and technologies. Release orchestration tools are critical to accelerate the benefits of DevOps.

With release orchestration, a company can achieve visibility into the entire release pipeline, to improve planning and scheduling of software updates. This visibility allows for process streamlining, operational efficiency maximization, and early identification of errors or problems before a software application goes live. Release orchestration also provides the framework for process control, helping a company to improve reliability and prepare for compliance and audits.

Test Automation

Test automation is the process of assuring quality in the Test phase of the pipeline via triggering and executing tests automatically and managing test data to make decisions. The test results should be actionable, meaning there is visibility and insight that developers, testers, release managers, and other roles can use to better the end-to-end delivery processes.

Test automation is integral to continuous delivery and continuous testing.

DevOps for Developers

Building a Case for Your Organization

Any questions or comments?

info@flexagon.com