dynatrace | EVOLVING SOLUTIONS

DevOps for digital transformation

How DevOps can transform IT and the business

What's inside

CHAPTER 1

Introduction

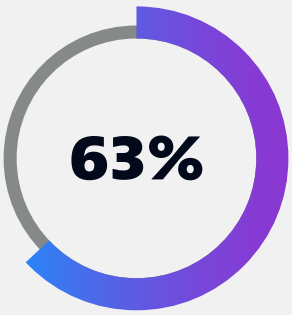
More and more organizations have adopted DevOps practices to streamline software development, increase developer productivity, and enhance continuous delivery workflows to deliver better software faster.

As DevOps pioneer Patrick Debois noted in 2009, tactics — not just technology solutions — define a successful approach to DevOps that can fundamentally transform IT. But while this tactical focus offers increased flexibility for teams, it can quickly lead to data and communication silos across the organization which can impair software quality and speed of delivery. Without help, it can be extremely difficult to gain strategic insight into how development teams perform their day-to-day tasks, how to automate DevOps pipelines, and how to architect software for reliability and resiliency in modern cloud-native environments.

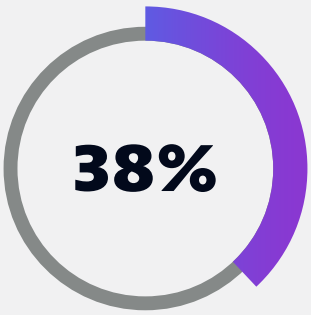
That help comes in the form of artificial intelligence (AI). AIOps, the discipline of applying AI and advanced analytics to IT operations, has transformed how organizations manage complex systems. Using these same principles, organizations can take a more intelligent approach to DevOps — an AIOps approach to DevOps that leverages AI throughout the software development life cycle (SDLC). DevOps, together with complementary technologies and tactics, such as site reliability engineering (SRE), has the potential to transform the business.

To better understand the transformative power of DevOps, we'll explore the basics of DevOps and the growing role of SRE; delve into key DevOps benefits and challenges; discuss DevOps best practices and key DevOps metrics; and examine how AI and automation at every stage of the DevOps lifecycle can transform the way organizations develop and deliver better software faster.

DevOps has potential power to reshape your business by streamlining your IT to deliver better value:[1]

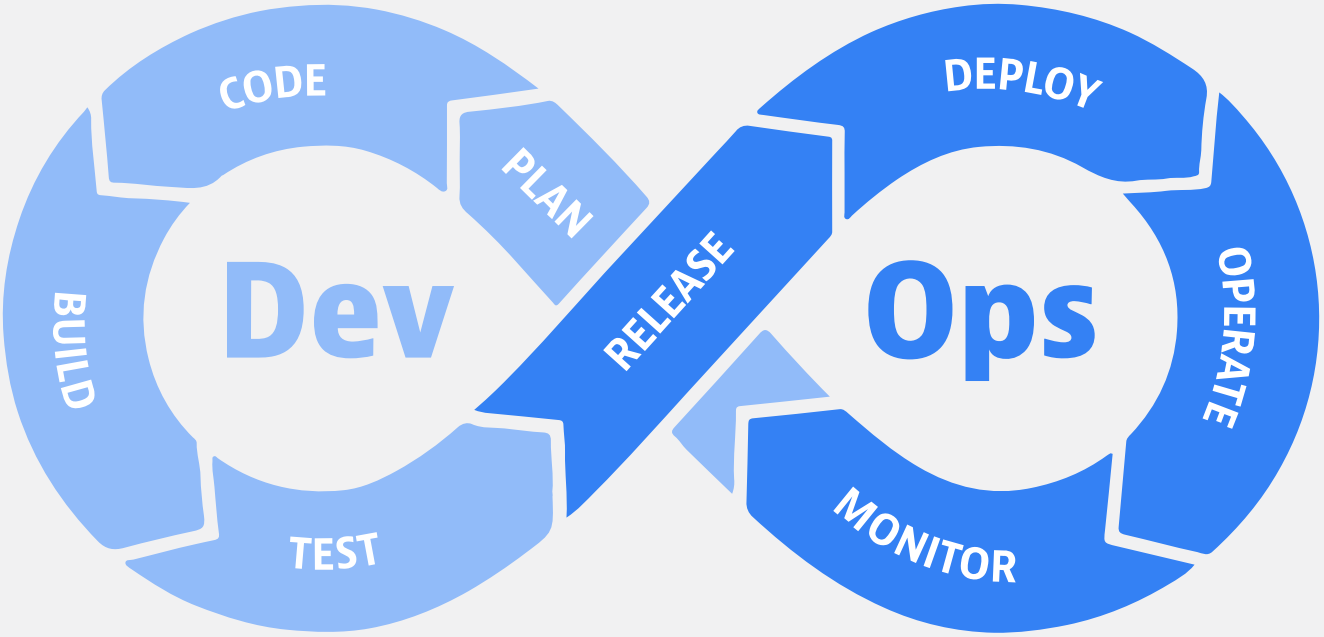


of organizations that use DevOps practices have **improved the quality of their deployments.**



of organizations report **improved code quality overall.**

[1] *DevOps Stats for Doubters.* UpGuard. https://www.upguard.com/blog/devops-success-stats#toc-1. August 2021.



CHAPTER 2

Getting to know DevOps and SRE

What is DevOps? Development meets IT Ops

DevOps is a flexible framework of software development practices organizations use to create and deliver software by aligning and coordinating software development efforts — "Dev" — with IT operations — "Ops."

The easiest way to conceptualize DevOps is as a continuous loop. Instead of discrete processes, development and operations become part of an ongoing cycle that includes planning, coding, building, testing, releasing, deploying, operating, and monitoring applications and services.

This continuous workflow approach enables teams to immediately identify and address issues related to both form and function earlier in the process to avoid problems before software is released into production.

The recent development of cloud-native technologies, open-source solutions, and flexible APIs have further enhanced DevOps efficiency. With its roots in Agile development, DevOps is ideally suited to help teams keep pace with accelerating development and release models, such as continuous integration and continuous delivery (CI/CD).

What is SRE? Software resiliency built in

SRE is a software operations practice that manages the details and big-picture concerns of software resiliency to ensure software systems' availability, latency, performance, and capacity. Site reliability engineers understand the needs of software systems and set up processes and structures to meet those needs.

Google VP of Engineering, Ben Sloss, coined the term SRE in 2003 when he and his team began to apply software engineering principles to software operations to create more reliable and scalable software systems. Implementing SRE can help organizations reduce friction between development and operations components of DevOps teams — streamlining efficiency and reducing error rates.

SRE complements DevOps practices by offering increased automation to reduce reliance on manual tasks. These practices help Dev teams solve their problems and deliver reliability-by-design earlier in the development process.

Ultimately, SRE helps organizations achieve their operational goals, such as reduced downtime or faster resolutions, by defining and automating service level objectives (SLOs).

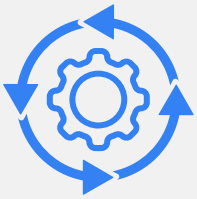
How do SRE and DevOps interact?

SRE and DevOps are essentially two sides of the same coin. While DevOps frameworks focus on whole-lifecycle collaboration and breaking down silos, robust SRE helps implement and automate DevOps practices using SLOs and ensures those systems—and the software they produce—are resilient.

According to Andi Grabner, DevOps Activist at Dynatrace, "DevOps and SRE are a balance between speed and safety." While DevOps helps organizations move from left to right along the development and operations lifecycle to boost overall speed, SRE moves right to left to help reduce failure rates earlier in the development cycle.

While it's possible to have DevOps without the SRE, these two processes work best as a pair, effectively creating a continuous cycle that delivers ongoing improvements across both directions of the CI/CD pipeline. As a combined practice, companies can seek to increase automation, speed up delivery, improve software quality and much more.

With this background information in hand, let's now dive into the fundamental components of the DevOps mindset to get a better idea of where your team will need to concentrate its efforts to get the most out of DevOps practices.



SREs came into practice to increase the resiliency of organizations through **automation of many manual tasks** early in development.

CHAPTER 3

Fundamentals of DevOps

DevOps is a cultural shift that requires vision, planning, executive buy-in, and tight collaboration to successfully establish a more integrated way of developing and delivering applications. By embracing a few fundamental practices, teams can improve their efficiency and develop a deeper understanding of their workflows, toolsets, and processes so they can release better software faster.

Because DevOps is a continuum, these practices should also be continuous and ongoing. This chapter covers the basic tenets or practices that form the fundamentals of adopting a DevOps approach.

Continuous integration

Continuous integration (CI) is a software development practice in which developers regularly commit their code to a shared repository. Because microservices architecture is distributed, CI allows developers to own discreet, manageable chunks of code, and individual features and work on them in parallel. The distributed nature of these applications allows for frequent updates — often multiple times daily. However, developers can't just push build updates haphazardly. CI is tightly controlled; new commits trigger the creation of fresh test builds via

the build management system. Redundant code is rejected, and breaking changes are minimized once master branches are altered. Incremental changes are encouraged. Additionally, reduced reconciliation prevents mandatory code freezes that commonly stem from conflicts. Overall, continuous integration enables teams to build and test software faster and more efficiently. By regularly merging code, teams also always have an up-to-date build that speeds up testing and bug fixing, boosts merge confidence, and helps to shorten the development pipeline.

Continuous delivery

While CI focuses on regular, independent code updates to a central repository, continuous delivery (CD) focuses on releasing completed code blocks to a repository at regular intervals. These blocks of code should always be in a deployable state for testing or release to production. CD is often confused with continuous deployment — the next process in line — which releases finalized code into production. Deployment is the act of making new and updated software available to end users. Accordingly, the CD primarily denotes "continuous delivery," or both "continuous delivery and deployment," but rarely just continuous deployment. CD takes code and adds it to a repository, such as GitHub or, in the case of a microservices-based environment, a container registry. The end goal is to increase release consistency by perpetually keeping code in a deployable state. Software development becomes more nimble and more predictable as a result.

Continuous testing and validation

Continuous testing in DevOps is important at every stage of the SDLC. It involves many stakeholders including the development team, quality assurance, and operational staff. The goal of continuous testing is to evaluate the quality of software as it progresses through each stage of the delivery lifecycle. This not only stops bad to code in its tracks but also provides fast and continuous feedback to the Development teams with the information they need to address any quality concerns.

Whilst continuous testing is important to scale, manual validation of test results derails the software delivery process. This is where continuous validation comes in — automating the evaluation process of test results against your pre-defined service level objectives. Continuous validation compliments the implementation of continuous testing by eliminating any manual analysis required whether it's comparing data on dashboards or checking off boxes on a spreadsheet. Instead, DevOps teams can set up techniques like quality gates that automatically enforce predefined quality criteria and prevent bad code from progressing to the next stage.

Continuous monitoring and observability

Though organizations strive for airtight CI/CD processes, there are often opportunities for improvement. Monitoring and observability are key to understanding viability of code as it progresses through the pipeline. While detecting issues and vulnerabilities is always important, the sheer amount of observability data associated with modern multicloud apps create means there's simply no way to manually track everything that's occurring across the software stack.

Traditionally, DevOps monitoring was closely associated with "ops" teams but has since evolved across the full software development lifecycle (SDLC) as key stakeholders are increasingly requiring answers to solve their challenges faster. These answers are possible when you have a system that is continually monitoring and analyzing observability data. Continual data capture can be empowering when leveraged intelligently, and this is where the introduction of observability is critical to DevOps. Observability is more than just collecting metrics and arranging them in dashboards. Having an AI engine that's working 24/7, 365 days of the year to analyze data and provide answers to anomalies and problems helps teams remediate issues faster and make better release decisions. This drives better code quality, better application performance which translates to better end user experiences. As software complexity increases, it is becoming harder for DevOps teams to deliver new features and releases faster without sacrificing quality. Therefore, empowering your teams with continuous observability and an AI engine to analyze all the data and provide answers is critical for success.

Continuous security

Another fundamental DevOps practice is continuous security based on testing, monitoring, authorization, and inventory tracking. This is the evolution towards DevSecOps. Simply put, continuous security is the process of making security part of the CI/CD process, covering the full SDLC, by adding an extra layer over the DevOps process and pipelines to ensure your infrastructure and applications don't have vulnerabilities and risks associated with them. Today, as we see environments become increasingly more complex, a "bolt-on" approach to security is not scalable or sustainable, and therefore baking security into your automated processes to enable the continual testing across your development lifecycle is imperative.

Similar to the shift-left mentality with regards to quality in testing, security measures should be baked into planning and creation from day one, and they should occur constantly throughout the development life cycle including when software is running in production. Further, any continuous security measures implemented should be automated as appropriate, as not to hamper efficiency. In terms

DevOps is a cultural shift

This new culture requires the adoption of best practices such as:

Continuous delivery

Continuous integration

Continuous testing

Continuous monitoring/ observability

Cross-team collaboration

of culture, security personnel must be regarded as full partners in the DevOps process, on par with developers and operations specialists — hence the shift towards DevSecOps.

Cross-team collaboration

Breaking down silos is paramount to ensuring good communication and unification across the DevOps pipeline. Effective DevOps execution means establishing a single source of truth — aggregating data from many sources into one collective location. Testers, engineers, QA, and even non-technical stakeholders can gain valuable insights from these bits of information. Under this paradigm, each will contribute in their own way to the creation of software that drives high-level business outcomes.

Seamless cooperation between developers and ops teams is especially critical. DevOps processes coexist in a continuous cycle known as a feedback loop. Different portions of a project are completed and reviewed by stakeholders, and feedback is returned from those steps. Code must be written, tested, validated, delivered, built, and ultimately deployed for end users.

Cross-team collaboration benefits from accelerating this cycle. Accordingly, automation has become a key ingredient in shortening the cycle from end to end — chiefly by reducing the friction caused by multiple parties working at the same time.

Taking DevOps to the next level

While we have established the fundamentals, organizations don't want to settle for the basics, they want to take their DevOps to the next level. To keep pace with innovation, and the need to deliver services to market faster, organizations need to take their DevOps practices to the next level. Evolving these fundamental practices into elite performing DevOps requires some best practices, which we'll cover in the next chapter.

CHAPTER 4

DevOps best practices

Many organizations claim to have a fully functioning DevOps process. But DevOps is more than just a workflow and a few tools your organization can implement and move on. It's helpful to think of it as a philosophy — a culture and mindset — that takes continuous optimization, creativity, and flexibility to maintain. Maybe an organization has implemented organizational changes and tools that lay the foundation for a good DevOps process, but they may be missing some of the benefits DevOps can offer.

With this journey of improvement in mind, let's explore some DevOps best practices that can take your investment in the fundamentals to the next level and ensure you're making the most of your DevOps strategy.

Automation

Automation is a cornerstone of every company's DevOps strategy. In short, automation reduces toil, helps you accelerate your delivery pipelines across the full SDLC, and enables you to scale your DevOps practice.

Traditionally, processes such as testing, monitoring, error discovery and remediation comprised a little automation and a lot of manual intervention. This worked when small teams worked on monolithic applications. But with modern microservices-based

applications and with digital transformation putting even more pressure on IT, automation is crucial to increase velocity and quality by driving consistent processes across every stage of the DevOps lifecycle. As a result, you can push code to production more frequently and produce consistent, reliable, and secure software whilst saving your DevOps team valuable time they can spend innovating.

Monitoring and observability

Monitoring and observability are essential to incorporate across every stage of the software development lifecycle, from pre-production to production. Whilst automating as many processes as possible increases the efficiency of your DevOps workflows, monitoring and observability provide your teams with visibility into those automated processes to detect and pinpoint the root causes of any problems or bottlenecks.

Many tools provide data and dashboards to track the health of individual systems. But to develop an effective observability strategy that yields actionable answers about systems throughout the DevOps toolchain, you need more than just data on dashboards – you need an intelligent approach.

1. **Make your systems observable —** adopt a standard, such as OpenTelemetry. Leverage an AI-based observability platform that can automatically instrument and detect anomalies, so you don't have to do it manually.
2. **Establish end-to-end observability** from pre-production to production for every application or environment.
3. Ensure you can understand the **business impact** of an event or transaction by analyzing it in context of the processes upstream and downstream from it.
4. **Leverage AI** to automatically detect issues and provide analysis to immediately pinpoint root causes and trigger auto-remediation.

This is an important practice to implement so your team can identify failures or performance problems before any impact is felt by your customers.

AIOps

Data can be an IT team's best friend, especially when it comes to testing and delivering code and monitoring services more efficiently. However, processing the massive amount of data created by today's applications is beyond the ability of humans alone. This paves way for an AI engine that can constantly analyze all the observable data down to the code-level detail, and that gives the development team the power to identify issues, get answers, and quickly remediate problems when they happen.

Harnessing AI as part of your DevOps processes enables you to enhance functionality and automation in development, testing, security, delivery, and release cycles as well as constantly monitoring the performance of deployed software far more efficiently than using manual efforts.

Shift-left quality

SREs live and breathe service level objectives (SLOs). Ensuring production service levels are on track requires continuous evaluation of service level indicators (SLIs) against SLOs. But that begs the question: why shouldn't developers ensure the code they build meets the same production SLOs? This concept of shifting left improves software quality, helps detect issues much earlier in the lifecycle, and

prevents code that doesn't meet production SLOs from progressing to the next stage. The results are fewer SLO violations in production, time and money saved due to fewer or no war rooms, but more importantly, ensuring 100% of business service level agreements (SLAs) are met.

One way to automate this shift-left process is through quality gates, which allow you to automatically compare SLIs from any pipeline tool (such as monitoring and testing) against pre-defined SLOs. If code does not pass the SLO-based quality gate, it cannot progress to the next stage, and the system automatically notifies the development team to remediate the problem.

Shift-right reliability

Progressive delivery (also referred to as shift-right) focuses on expanding overall CI/CD practices to help deliver applications and services with more control. It allows organizations to precisely manage how and when new features, updates, and fixes are delivered to minimize the potential negative impact to the user base. Some common practices include blue-green deployments, A/B testing, canary deployments, and feature flags.

- **Blue-green deployments**

  This application release model gradually transitions users from a current version of an application or service (the "blue" version) to a new release (the "green" version) while both blue and green are running in production. This change should feel seamless to the user, and blue can stand by in case an unforeseen problem with green requires rollback to the earlier, more stable version.

- **A/B testing**

  Also known as split testing, A/B testing refers to randomized experimentation processes where two or more versions of some variable — for example, a service, webpage, or page element— are shown to different end-users. From there, you can monitor app performance as well as user behavior and satisfaction to determine which option is best for business goals.

- **Feature flags**

  Also known as toggles, feature flags are a development practice that allows software and development teams to enable and disable parts of a codebase with a simple switch (or flag). Feature flags help organizations decouple code deployments from feature releases, allowing them to make code changes in production that remain hidden from the users until they are activated. This results in increased deployment speeds, improved system stability, and better cross-team collaboration.

- **Canary deployments**

  All deployments in production carry risk even with comprehensive monitoring and testing. One method for developers to mitigate serious disruption is through canary deployment. The term originates from when canaries were used to detect toxic gases within coal mines. If the canary died, miners would know to get out before the gas reached them. A canary deployment is a release of software that's deployed to a small percentage — referred to as the canary — of the whole userbase. If things run well in your canary, you can then deploy the release to the rest of the userbase. If things don't run well, at least the impact is much smaller, less disruptive, and you can roll back the software. Canary deployments provide the ability to test actual users, who can provide real feedback, while reducing risk by mitigating impact if problems lead to a better-quality product.

Shift-left and shift-right security

The concepts of shift-left and shift-right also apply to security. First, let's talk about shift-left.

Ever since DevOps teams started using containers as a way to package applications and started releasing software at a faster cadence, there has been a desire to automate application security tests and provide test results earlier in the software development lifecycle. By providing test results earlier, software developers can fix security flaws faster and easier. They don't have to remember a change they made weeks ago that accidentally introduced a security vulnerability, and unravel everything that has been done since then.

In addition to information being provided earlier, automated release decisions can be done earlier, based on the security test results. This has been the holy grail for DevSecOps — providing more automation and less manual work. The result is better, higher-performing, and more secure software — with less work needed by human beings.

How about shift-right security? That's important too. After several years of "shifting left", enterprises are realizing they also need to maintain visibility in the production environment. We've seen many successful attacks against Kubernetes environments — from the malicious images that were inserted into Docker hub in 2020, to the attacks against Azure and Tesla by "cryptojackers". This is why 44% of enterprises say they are planning to adopt new runtime security controls (shift-right) over the next 12–24 months.

In a nutshell, here are the reasons why automated security (DevSecOps) can and should shift right into production environments:

- The production environment is connected to the Internet, which is where most attacks happen.
- Scanning source code in the development environment can't give you the same rich insights you can get by observing an application when it is running in production. For example, static source code scans can't show you what libraries are actually loaded, how they are used, whether a process is exposed to the Internet, or whether a process interacts with sensitive corporate data.
- Some applications running in production, such as those you purchase from third parties, may not have run through your dev environment, so they never had a chance to be scanned by security tools in development.
- New zero-day vulnerabilities are often discovered after an app has been deployed into production. By implementing continuous application security monitoring in production, you can be aware of these risks.

Building resiliency with chaos engineering

Chaos engineering is a development discipline that subjects software to failures in a simulated production environment as a way to build resilience into distributed production software systems. This practice builds confidence in software's ability to withstand unexpected or unlikely circumstances, such as outages, slowdowns, excessive loads, and so on.

Testing the performance of your application under random and extreme circumstances is a helpful exercise to ensure your team delivers durable, reliable, and highly available systems in any given situation. The only way of doing this is in production environments with real users and actual load levels.

Adopting a platform approach for your DevOps value stream

There is no shortage of DevOps tools IT teams can use today to execute different parts of the DevOps lifecycle. But as your DevOps approach matures and you look to scale DevOps across multiple applications, toolchain sprawl becomes a reality. What once worked well becomes manual, cumbersome, costly, and reverts to a siloed approach. Imagine having multiple teams trying to use the same tools each for their own applications.

Standardizing on a platform approach that provides automation, intelligence, and observability on top of the regular DevOps processes helps reduce overhead, reduce toil, and improve efficiency. An all-in-one platform approach creates a single source of truth that tears down silos, integrates toolchains, and enables self-service models. This approach helps automate the entire development pipeline and gives developers and operations teams the right tools and data for every stage of the DevOps cycle — from coding to delivery and back again.

Driving futuristic development today

The goal of any business and technology leader is to make the development of apps and services easier. These emerging best practices include efficient ways to develop critical applications, code, and services. Further, by leveraging these emerging tools, you can deliver a more proactive and prescriptive development architecture capable of meeting today's digital demands.

The fundamentals of DevOps are a set of processes and technologies that **empower automation, monitoring and observability, and AIOps.** These processes lead to a shift-left and shift-right mindset that result in **continuous delivery of quality and resilient releases.**

CHAPTER 5

Benefits of DevOps

As teams embrace shifts in both culture and processes, DevOps' holistic approach to software and infrastructure creation can pay dividends, even at the organizational level. Once teams have established some best practices and key metrics to monitor and manage, teams can expect to see some core benefits.

Increased speed of delivery

As companies digitally transform, the pressure is firmly on development teams to build and deliver software faster and more often without sacrificing quality. When surveyed, 63% of DevOps practitioners report that DevOps enables them to release software more frequently. In contrast, development takes 41% more time in organizations that don't use DevOps practices. That time could be better spent creating new features or implementing new processes. The difference? DevOps encourages teams to develop code in smaller chunks and democratize access to code, which means developers are working on smaller, more frequent releases and with tighter feedback loops to quickly iterate and release software faster.

DevOps processes break down silos and foster better collaboration and feedback loops between teams. This cross-functional connection and coordination reduces delivery lead times by enabling teams to automate processes such as monitoring, test evaluation, and remediation that used to be done

The benefits of DevOps

DevOps practices are an investment whose dividends increase with time and experience. Some of these benefits include:

Increased speed of delivery from improved processes

Higher quality software releases from better testing

Improved productivity and collaboration from empowered developers

Better business outcomes from happier customers

manually. Through this automation, teams can develop self-service models to accelerate delivery pipelines and scale DevOps processes across the organization.

Increased frequency of releases

A central philosophy of DevOps is to focus on smaller coding changes — opting for agility and quick feature pushes, as opposed to maintaining huge code bases and making infrequent monolithic releases. These smaller changes are easier to commit to code repositories such as GitHub or BitBucket, and easier to test.

More frequent releases mean users can access new features and functions more quickly. They also mean developers get real-world feedback more quickly, which means they can respond to issues and make optimizations more rapidly.

Frequent releases also promote continual improvements to DevOps processes and workflows. Some companies, such as Google, rank their teams based on their release performance, which fosters a never-settle attitude and emphasizes speedy response times.

Reduced risk and increased release confidence via higher quality software

Another central tenet of DevOps is frequent, automated testing at every stage of the development lifecycle, which reveals issues well before it hits production. Through practices like shift left, where code is evaluated against production SLOs, development teams find that bad code automatically gets stopped from progressing to the next stage thereby helping improve the overall quality of software and reducing failure and defect rates.

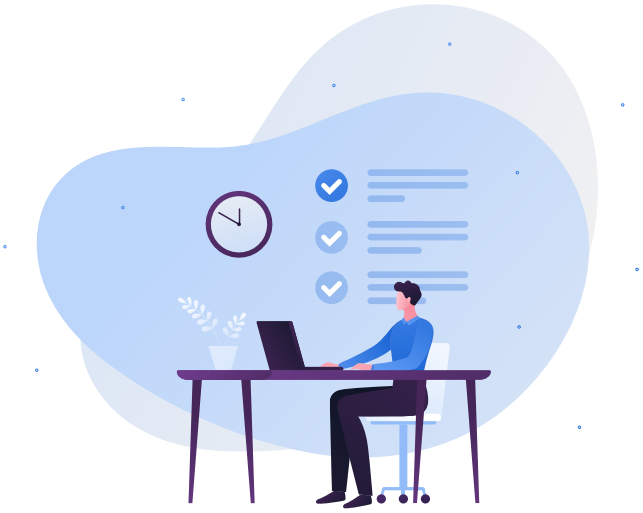
The results of continual testing are impressive: 63% of organizations that use DevOps practices have improved the quality of their deployments, while 38% report improved code quality overall. Teams also spend 21% less time extinguishing fires, which leaves more time to innovate and improve processes.

Reduced risk translates to increased release confidence. When more tests and processes, including release decisions, are reliably automated using SLO-driven quality gates, teams can release software knowing it is well tested and meets users' needs.

Developer empowerment

The planning, automation, and frequent testing that accompany DevOps practices provide developers with a rapid feedback loop that can increase their confidence and empower them to work independently.

By automating tedious tasks and routine approvals wherever possible, teams can maximize their efficiency and work confidently knowing they are following pre-approved workflows. These processes enable a culture of group ownership and accountability to develop within teams. When issues do arise, the DevOps culture refrains from assigning blame for mistakes. Instead, the focus is on process improvement, lessons learned, and growth. This culture can boost morale and contribute to better software development.



Better collaboration and developer productivity

The DevOps culture also encourages strong teamwork, moving away from silos in favor of unified workflows. As teams are dealing with smaller code fragments and rapid feature development, teams can engage in meaningful collaboration to ensure the puzzle pieces fit together.

DevOps, and by extension DevSecOps, strengthens ties between developers, IT, operations, and security professionals across an organization, as employees from each discipline have more awareness of each contribution. With a strong DevOps initiative, transparency, and visibility are the natural state of a project. Metrics gathering and an all-hands-on-deck approach keep everyone in the know. Centralized DevOps tools and orchestration platforms empower teams with shared information and resources.

Increased security

Since its introduction, DevOps has extended beyond Dev and Ops teams to now include Security teams — referred to as DevSecOps. This methodology seamlessly integrates of security testing and protection throughout the software development and deployment lifecycle. Much like DevOps, DevSecOps is about taking a collaborative approach, and it's important for organizations to adopt as the speed of DevOps can often lead to applications in the development cycle left vulnerable to security attacks.

Adopting DevSecOps enables your organization to maintain a collaborative approach through development whilst still ensuring security is not compromised. Security assessments cannot wait until after the development cycle, instead they must happen concurrently. In doing so, DevSecOps teams can detect and respond to software flaws in production quicker and more efficiently. This results in faster software development, innovation, and delivery.

Reduced mean time to resolution (MTTR)

Complex, multi-faceted systems inevitably experience failures, and teams must prepare accordingly. Reduced MTTR is a measurement of how long it takes to fix these failures (typically in hours or days). You might also envision this as average downtime across a crop of failures. Naturally, teams highly invested

in DevOps principles have lower MTTR. This is an accurate measure of both adherence to best practices and the overall collective skills.

In many cases, reducing MTTR is simply a function of how quickly teams can pinpoint and identify issues. To quickly identify issues with precision, teams must have full observability over their infrastructure in a DevOps environment. Reducing MTTR also relies on monitoring, precision analysis, and formulating remediation plans. Teams will also spend less time on support calls — not only because code is higher quality, but because teams can identify root causes more quickly.

Increased software reliability and resiliency

The DevOps practices of managing smaller code blocks, testing continuously, and automating processes also encourage teams to build reliability and resiliency into software from the beginning. When software is designed with reliability and resiliency in mind, it's much easier to roll code back as needed — or even remove live code from production when issues arise without breaking other features.

Since applications are collections of individual core functions (especially in the age of cloud platforms and microservices), it's easier to disable or remove faulty features without impacting the entire package. DevOps teams can then resolve issues and re-release services when ready, maintaining system reliability. Several tools make this possible in one or two clicks. Furthermore, the data captured during software operation allows teams to design better tests and prevent future problems.

Happier customers

DevOps encourages and enables teams to be more responsive to feedback. Processes such as rollbacks aren't relegated to just bug fixes — they're also useful for removing features that are ill-received and improving upon them by creating another agile method to meet user demands. End-users are also more prone to view software as "innovative" or "capable" when feature delivery is continuous and seamless.

Better business outcomes

We often talk about the benefits DevOps brings to the developer and operations teams directly, but the benefits of DevOps also extend to the scope of the entire business right up to the C-level. And Patrick Debois, known as the creator of the DevOps movement, notes the greatest advantage of DevOps is the understanding it gives.

DevOps allows a business to be more versatile and information-driven to meet the customer and business needs. The benefits adopting a DevOps approach yield, in turn lead to increased efficiency, work ethic, and ultimately higher profit which can be put back into the business for future growth.

Proving the progress

The dividends of DevOps practices are a benefit that increase with time and experience. Establishing a DevOps practice is an investment that also comes with its challenges.

CHAPTER 6

Challenges of DevOps

While the benefits of DevOps are clear, establishing the tools, processes, and culture changes required to achieve a successful DevOps practice can be challenging. By working together, development, IT, and operations teams can eliminate roadblocks and focus on improving how they create, deploy, and continuously monitor software.

Although working together is a vital component, it is not the only ingredient for success. Making the transition to DevOps requires planning and preparation. Here are some pitfalls to avoid as organizations plan and implement DevOps practices.

No buy-in from the top

For a modern organization to realize its business goals, leadership needs to trust its technical staff and develop an understanding of IT's goals and pain points.

Because DevOps integrates disciplines across the development and operations life cycle, from product design to customer support calls, strong DevOps can't happen unless organizations get buy-in from the decision-makers who head up the organization or the departments involved. The very leanest organizations may even need buy-in from all the way up the chain — the CEO. Others may only need a champion in the upper management tier. Either way, the senior

The challenges of DevOps

DevOps faces many challenges without proper strategy and discipline within your organization. These may include:

No leadership buy-in

Poor observability strategy

Manual processes

Not identifying/ obtaining the right metrics

Security vulnerabilities

management that calls the shots on purchasing and product strategy needs to understand what it takes for IT and DevOps teams to work together to improve how the organization delivers its goods and services to customers and promote the cultural changes required to make cross-organizational DevOps successful.

Not having an observability strategy

Observability is not the same as monitoring. In a monitoring scenario, preconfigured dashboards are meant to alert teams to the performance issues they expect to see later on. However, these dashboards rely on a key assumption: that teams can predict what kinds of problems they will encounter before they occur.

Observability is based on the outputs of a system and enables teams to understand exactly what is slow or broken. With adequate observability into cloud-native apps and platforms, development teams can leverage telemetry data to get more insights into apps and systems, automate more processes, and release higher quality code faster. Gaining end-to-end observability into a software environment requires a combination of careful consideration and powerful technology and is a critical part of ensuring DevOps scalability and success.

Not automating ALL manual processes

A central goal of DevOps is to automate as many processes and decision points as possible to improve throughput and software quality. Here, teams should automate testing, but also workflows, such as advancing software from test to release or committing code to a repository.

Teams may also rely too heavily on tribal knowledge that lives in the heads of a few, requiring manual approvals that create bottlenecks to automation.

As the number and type of technologies DevOps environments encompass constantly grow and fluctuate, auto-scaling based on demand becomes imperative. Automating certain types of responses, such as alerting or auto-correcting performance issues, is another key capability.

Not bringing AIOps to DevOps

AI can be thought of as a three-legged stool, along with observability and automation. Intelligent decision making with causation-based AI helps development teams understand the root cause and pinpoint precisely where errors are occurring and what caused an application failure.

But AI isn't just for reactive measures, AI helps predict potential SLO violations or application failures before

they even hit production, allowing teams to quickly remediate and address any issues before it impacts users. Leveraging AI across the pipeline helps IT scale their DevOps to include thousands of apps and microservices to analyze millions and billions of dependences.

Not adopting a self-service approach

A major part of IT transformation is the ability to give technologists the tools they require to be successful on a daily basis. Within application and services development, it's important to look at self-service as a way to reduce wait times, deploy new features, shorten feedback loops between different teams, align tooling, and improve CI/CD pipelines. Self-service is too often ignored until the end of a transformation effort or left off the table entirely. To unify DevOps teams, organizations should build a plan for self-service into the strategy. Establishing automation and observability in your DevOps strategy makes creating self-service models and scaling DevOps simple.

Not thinking about security when designing processes

Many organizations treat security as a separate expertise that's applied after code is developed. But any DevOps initiative should have a plan for integrating security as tightly as possible. Ideally, the security team should be a full partner in the software development life cycle on equal footing with development and operations. This is the meaning of DevSecOps. By shifting security left and baking it into the product at every stage of the development and delivery process, teams make apps and services more resilient against a greater number of threats now and in the future. DevSecOps grants visibility into code vulnerability, dependency mapping, secure SDLC reviews, a deep understanding of how a target tolerates a real attack, and just how far an attacker can go. Failing to include security in DevOps — or at least to create a roadmap to its inclusion in the future — is a critical misstep that will sacrifice countless important insights throughout the life of the product.

Not measuring the right metrics

If teams are scouring through error logs or reactively trying to piece together an issue, they may not be looking at the right metrics. Identifying what organizations need to measure is key for achieving valuable insights from the data systems produce.

Although there are certain benchmarks all organizations should watch, such as throughput and latency, average response times, queue time, errors, and impacts on memory, each organization will have metrics for systems and processes that are unique. Start with a set of key metrics that evaluate code quality and testing effectiveness, as well as workflow efficiency and incident response times. To ensure management buy-in, teams should understand key pain points that are of concern to the wider organization. Because DevOps tools and processes touch such a wide digital footprint, they afford organizations an opportunity to discover, measure, and improve key trends across the organization.

A team effort

Aligning teams from different disciplines is never an easy task. However, leaders in the technology space see the line blurring between IT, operations, and development as teams depend on each other to ensure business success. In a healthy ecosystem where IT and development operations are working together to achieve a common goal, organizations can drive true IT transformation efforts with full support for DevOps staff and initiatives.

CHAPTER 7

DevOps Metrics

At the heart of all successful DevOps and SRE practices are metrics. Telemetry from every stage of the DevOps workflow — from development and testing to deployment and operations — provides critical clues about how your software is performing and how efficient and effective your DevOps processes are.

Reliable, measurable data is required to automate testing, commits, and releases. To establish best practices, organizations can start with the project Google's DevOps Research and Assessment (DORA) team established, known as "The Four Keys", which defines the four basic metrics that indicate the performance of a DevOps team. So what are DORA's four keys, and what other key metrics can organizations track to improve their DevOps and SRE practices?

Deployment frequency

Deployment frequency measures how often an organization successfully releases to production.

DevOps and continuous integration/continuous delivery (CI/CD) go hand in hand. Teams now work with smaller code blocks, and continuous testing and validation allows for more rapid commits. This development pace means teams can release more frequently, often multiple releases per day.

Four key metrics

DevOps is data-driven and its success depends on reliable metrics. According to DORA these include (but are not limited to):

Deployment frequency

Lead time for changes

Change failure rate

Mean time to resolution (MTTR)

A high deployment frequency is critical for answering customer demand. The faster organizations can deliver bug fixes, improvements, and new features, the faster developers can receive valuable real-world feedback and users can realize value that bolsters your brand.

Deployment frequency is both a long-term and short-term metric. For example, you could measure how many code commits you're pushing daily or weekly in response to process changes — perhaps as a gauge of efficiency. Over longer periods, teams can track whether their deployment numbers are increasing over time. Slow-release schedules may indicate bottlenecks or service delays that need attention.

Lead time for changes

Lead time measures the amount of time it takes for committed code to get into production.

Lead time comes into play when responding to specific application-related issues and indicates how quickly your team can remediate a bug or a tooling glitch. Like development frequency, lead time for changes helps teams understand how effective their processes are.

Lead time is easy to average and quantify, making it a metric accessible to all application stakeholders.

However, lead time isn't 100% black and white. While longer lead times could indicate issues, they may also be the result of a team's focus on complex projects. These efforts will naturally take more time. It's important to investigate the context behind lead-time numbers and evaluate accordingly. While average organizations might have lead times ranging from one week to a month, some DevOps squads can push out production changes in under 24 hours.

Two important ways to improve lead time for changes is to implement quality assurance testing throughout multiple development environments and to automate testing and DevOps processes.

Change failure rate

Change failure rate measures the percentage of deployments that result in a failure in production that requires a bug fix or roll-back.

Teams can commit and deploy changes with high deployment frequency and lead-time-for-changes rates, but those efforts are diminished if problems sneak into production. A high change failure rate (above 40%) can indicate poor testing procedures, and requires teams to make frequent small changes, which erodes efficiency.

The goal behind measuring change failure rate is to evolve into a fully automated DevOps process. By automating testing and processes, released software is more consistent and reliable, and more likely to be successful in production.

Calculating change failure rate requires the ability to count total deployments and link them to notable incidents. An automated AIOps solution can find these incidents in GitHub or other code repository reports, and monitoring system alerts and user tickets. Since all organization's processes and systems are unique, exactly how to measure change failure rate can vary widely.

Mean time to resolution (MTTR)

Mean time to resolution measures how long it takes an organization to recover from a failure in production.

Users depend on feature availability, and 99.99+% uptime is the coveted goal. Time to resolution is essential for making sure teams recover from unplanned outages or service impairments immediately and as efficiently as possible — lest organizations trigger user frustration and lost revenue.

To determine MTTR, you can assess the time between when an incident occurred versus the time when it was resolved. What deployment resolved this incident? Observability into deployment data and user experience data is key to knowing whether service has been restored effectively. Extended restoration times can point to poor alerting or poor monitoring and can result in a larger number of affected systems.

A best practice to achieve quick MTTR is to deploy software in small increments to reduce risk and deploy automated monitoring solutions to preempt failure. MTTR is another metric that varies widely between systems.

Beyond DORA: Additional important metrics to track

While DORA's four keys are an essential foundation, there are a myriad metrics you can use to track the effectiveness of your DevOps and SRE processes. Here are six more metrics you can track to holistically assess your pipeline's effectiveness.

1. Defect escape rate

Also known as defect escape rate velocity, this metric measures the rate of issues that "escape" detection during development and are discovered in production. You can then calculate the defect rate per period of time, per release, or per deployment. Higher escape rates can point to testing issues and associated automation shortcomings. It's fully possible that tools are faulty when defects propagate.

2. Mean time to detection (MTTD)

MTTD measures how quickly teams discover issues on average. Broad system failures, vulnerabilities, and outages can wreak havoc on applications the longer they persist. Identifying and resolving these issues is paramount to reducing the overall impact of a problem across your applications, infrastructure, and users.

Effective monitoring is a central tenet of DevOps. Achieving a low MTTD requires teams to implement effective monitoring, alerting, and end-to-end observability to immediately detect an anomaly or service degradation. The source of an issue may be obvious, such as a central service outage, but it can also be more difficult to detect, such as a back-end failure, a problem with an open-source tool, or a faulty code snippet. Detecting issues like these requires full-stack monitoring with code-level visibility.

3. Percentage of code covered by automated testing

Another central goal of DevOps is to automate wherever possible. Accordingly, automated testing is integral to catching more errors with syntax, security, and compatibility within builds.

To accomplish this, organizations can implement testing environments that automatically simulate how code behaves under various circumstances. Boosting the overall percentage of code that goes through automated testing and validation makes testing quicker and easier, which speeds up the DevOps pipeline, and shortens the feedback loop from issue detection to resolution.

4. Application availability

Application availability is a measure used to evaluate the extent to which an application is fully functioning and available to meet the business and end user's requirements. A highly available system is designed to meet the gold standard KPI of five 9s (99.999%). Ensuring greater application availability keeps customers engaged and connected to your services.

While downtime isn't always expected, it's often planned as a result of maintenance. Communication between DevOps and SRE team members is crucial to resolving unforeseen failures and ensuring both the frontend and backend operate seamlessly.

5. Application usage and traffic

Application usage and traffic monitors the number of users accessing your system and informs many other metrics, including system uptime. Usage statistics are useful for teams, as it's not uncommon for application updates to impact user activity — good or bad. This can occur when issues arise, or when long-awaited features finally drop, causing traffic to spike. Having these metrics enables DevOps team members to react and manage these spikes effectively.

On the other end of the spectrum, when user activity slows to a crawl, this could suggest a service is interrupted in some form. Although sudden changes are much more telling for teams, it's also important to assess long-term trends for problems that develop over time.

DevOps is data-driven

Wherever you are on your journey to DevOps maturity, the ability to measure and make sense of the data coming from every stage of your workflow will help you perfect your strategies. In every area, from application performance, reliability, and stability to DevOps effectiveness and efficiency, metrics are the lens into your DevOps practices to help you drive continual improvement. Leveraging an observability solution that delivers high-fidelity data and analytics you can rely on to fuel these metrics enables teams to increase responsiveness, fine-tune processes, and deliver better software faster.

CHAPTER 8

DevOps and intelligent observability enable digital transformation

Digital transformation is now critical for enterprises to achieve business goals. By creating a continuously reinforcing feedback loop, DevOps offers a way for organizations to evolve corporate culture and empower the next generation of software creation, management, and security.

DevOps combines development and operations into a unified framework that breaks down silos and fosters whole-lifecycle collaboration. In this environment, SREs can implement operations that ensure software systems' availability, latency, performance, and resiliency. Likewise, CI/CD practices can provide well-aligned and automated development, testing, delivery, and deployment.

Efficient and effective DevOps practices also depend on monitoring key metrics, such as the DORA Four Keys that measure deployment frequency, lead time for changes, change failure rate, application availability, and mean time to restore service (MTTR), among others. With DevOps groundwork laid, organizations can develop best practices, such as automation, monitoring and observability, and AIOps, that empower continuous software delivery at scale.

DevOps doesn't just improve workflows — it also delivers measurable, end-to-end benefits, including increased speed of delivery, increased release frequency, reduced risk, and reduced MTTR.

To facilitate smooth DevOps, SRE, and CI/CD practices, Dynatrace's AI-powered Software Intelligence Platform seamlessly integrates with an organization's DevOps toolchain and automates tasks throughout the DevOps lifecycle. With continuous automation and precise root-cause determination, Dynatrace makes it possible for organizations to fulfill the potential of DevOps and simplify cloud complexity. By allowing faster innovation, more efficient collaboration, and the capability to integrate application security as part of emerging DevSecOps solutions, Dynatrace delivers precise answers for every phase of the software development and delivery lifecycle.

Ready to deliver on the transformative potential of digital delivery? Discover how Dynatrace can help.

Learn more

08.08.22 BAE1021_WP_USlet_te