Definitive Guide to

DEVOPS MODERNIZATION

Contents

The Evolving DevOps Landscape

DevOps is a set of practices that aims to automate and integrate the processes between software development and IT operations teams. The goal of DevOps is to improve the speed and quality of software delivery by fostering collaboration and communication between these two traditionally siloed departments. While DevOps has been widely adopted, the need to modernize DevOps practices has become increasingly apparent in recent years.

As applications become more complex and distributed, traditional DevOps practices struggle to keep up with the scale and intricacy of modern software architectures. Meanwhile, business demand for faster release cadences and security threats is growing. Teams are challenged to deliver more complicated systems faster and more safely. It’s hard to keep up.

The rise of AI coding assistants has helped developers generate more application code. However, legacy DevOps environments were already near their breaking points. They struggle to scan, test and release even more features. Something needs to change.

By modernizing DevOps, organizations can implement advanced automation, monitoring, and orchestration tools to meet this challenge. Modernizing DevOps allows organizations to leverage new technologies and tools to streamline their development and deployment processes, leading to faster time-to-market and improved competitiveness.

Developers play a crucial role in the success of DevOps practices, and were seen as half of the equation. Today, many of the tasks that used to be handled by “downstream” teams including QA, operations and security, have been added to the developers’ todo list as part of shift-left initiatives. While these approaches are necessary for accelerated delivery, they can also overwhelm the development teams leaving them with precious little time to write new, innovative code.

Challenges Modern DevOps Needs to Address

One of the key issues with DevOps is the lack of standardization and consistency across teams and organizations. Different teams may have varying interpretations of DevOps practices, leading to inconsistencies in processes, tools, and workflows. This lack of standardization can result in inefficiencies, miscommunication, and delays in software delivery. Inconsistent coding standards, ad-hoc deployment procedures, and fragmented toolchains can result in developers spending a significant amount of time resolving integration issues, debugging errors, and coordinating with team members to ensure alignment and compatibility across different environments. Improving standardization and establishing best practices within DevOps can help streamline processes, enhance collaboration, and ensure a more consistent and reliable software delivery pipeline.

Another issue with DevOps is the siloed nature of development and operations teams. Despite the goal of DevOps to break down silos and foster collaboration between these two departments, organizational barriers and cultural resistance can hinder effective communication and teamwork. Breaking down these silos and promoting a culture of shared responsibility and accountability is essential for success. By encouraging cross-functional collaboration and knowledge sharing, organizations can improve efficiency, reduce bottlenecks, and accelerate software delivery.

Security is another critical issue that needs to be addressed in DevOps. As cyber threats become more sophisticated and prevalent, organizations must prioritize security throughout the software development lifecycle. Integrating security practices into the DevOps pipeline, known as DevSecOps, is essential for identifying and mitigating security vulnerabilities early in the development process.

Incorporating security practices such as secure coding, vulnerability scanning, and penetration testing into the DevOps pipeline is essential for identifying and mitigating security risks early in the development process. Developers must stay informed about the latest security trends and best practices to ensure that their software is resilient against potential threats. By incorporating security measures seamlessly into DevOps practices, organizations can enhance the resilience and integrity of their software applications.

Governance is a related challenge. Many organizations find that giving developers permissions to author and update their pipelines means sacrificing certainty that appropriate approvals or checks have been executed. While locking developers out slows innovation and fosters dissatisfaction. A modernized DevOps model must wrestle with this challenge and enable significant developer control while ensuring that required governance processes are followed.

The sum of the challenges developers face is non-coding work referred to as “toil” – which impacts the efficiency and effectiveness of software development and delivery pipelines. These tasks can include manual testing, deployment processes, infrastructure provisioning, and troubleshooting issues in production environments. Developer toil not only consumes valuable time and resources but reduces job satisfaction among developers.

But the reality is, many organizations still use manual testing procedures, manual deployment scripts, and manual configuration management practices, which require developers to perform repetitive tasks that could be automated. Manual processes are error-prone, time-consuming, and inefficient, leading to delays in software delivery and increased risk of error. This toil can also overwhelm developers, leading to burnout, frustration, and decreased morale. Which means organizations should embrace automation to support dynamic and agile software development processes.

Another driver of toil and cognitive load for developers is the proliferation of point DevOps tools. On a daily basis, there are a multitude of places to run things or check for updates.Point products, such as individual tools for version control, build automation, testing, deployment, and monitoring, have long been the norm in software development It's difficult to be successful with the tools while still finding focus time to drive actual innovation. When new projects need to be configured, it's worse. New services must be configured across all the tools, with appropriate integrations and hooks between them wired together.

It's important to note that developer toil not only impacts individuals but also has broader implications for organizational efficiency and competitiveness. The time and effort spent on manual processes detract from valuable development work, such as designing new features, optimizing performance, and enhancing user experience. By automating repetitive tasks, organizations can free up developers to focus on high-value activities that drive innovation, accelerate time-to-market, and deliver greater business value.

Shifting from Point Products to Platforms

To counteract the burden of a proliferation of point tools on developers, and to drive cost savings and simplification, the industry is moving towards a platform approach to DevOps toolchains. This shift reflects a growing recognition of the need for a more holistic and streamlined approach to software development and delivery.

Software delivery platforms offer a unified and integrated solution that combines multiple tools and capabilities into a single platform. These platforms provide end-to-end support for the entire software development lifecycle, from code creation and collaboration to testing, deployment, and monitoring. By consolidating tools and processes within a single platform, organizations can streamline their development workflows, reduce complexity, and improve visibility and control over the entire software delivery pipeline.

One of the key benefits of software delivery platforms is the seamless integration and interoperability of tools and processes. Developers can work within a unified environment that provides a consistent user experience, standardized workflows, and automated pipelines for building, testing, and deploying software. Instead of learning how to create pipelines in a CI tool, a CD tool, and an Infrastructure as Code tool, a good DevOps platform will provide a consistent experience. Developers also won't have to build integrations between CI and CD to hand off builds, or between infrastructure and CD to hand off environments. That should be part of an integrated platform. This integration eliminates the need for manual handoffs between different tools and teams, reducing errors, delays, and miscommunication in the development process.

Software delivery platforms also promote collaboration and communication among cross-functional teams by providing shared visibility into project status, progress, and dependencies. Developers, testers, operations teams, and other stakeholders can collaborate more effectively, share knowledge and insights, and coordinate efforts to deliver high-quality software efficiently. This collaborative approach fosters a culture of transparency, accountability, and continuous improvement within the organization.

The danger inherent in moving to a platform approach is that a particular portion of a platform may not meet your needs for some applications. Teams are advised to consider this possibility carefully evaluating platform depth of capability as well as its ability to cooperate with the occasional point tool. Regardless, the benefits of platform approaches are making them compelling as DevOps matures from a novel field to a well understood practice that is ready to scale.

Accelerating Development Pipelines

In today's fast-paced and competitive software development landscape, the need for faster developer pipelines has become increasingly critical. Developer pipelines, which encompass the processes involved in building, testing, and deploying software, play a crucial role in determining the speed, efficiency, and quality of software delivery. By optimizing and accelerating developer pipelines, organizations can improve their agility, responsiveness, and competitiveness in delivering value to customers and staying ahead of market trends.

One of the primary reasons behind the need for faster developer pipelines is the increasing demand for rapid innovation and time-to-market. In today's digital economy, organizations must respond quickly to changing customer needs, market dynamics, and competitive pressures. Slow and inefficient developer pipelines can delay the release of new features, bug fixes, and updates, hindering the organization's ability to stay relevant and meet customer expectations. By speeding up developer pipelines, organizations can shorten the feedback loop, iterate more quickly, and deliver value to customers faster.

Modernization will typically involve faster feedback from builds driven by optimized selection of tests to skip based on what code changed, smart caching and improved build infrastructure. Meanwhile, approaches like canary deployments and cohort by cohort feature releases using feature flags enable real-world validation of new capabilities in production with the safety of being able to stop an experiment at any time.

Faster developer pipelines also help address the issues around the complexity and scale of modern software systems. As applications become more distributed, interconnected, and feature-rich, the number of code changes, dependencies, and integration points increases exponentially. Traditional manual processes and fragmented toolchains can struggle to keep pace with the complexity of modern software development, leading to bottlenecks, errors, and inefficiencies in the software delivery pipeline. By modernizing, each change is implemented more quickly, so that fewer changes are in flight at once. This brings the organization closer to the Lean ideal of “single piece flow.”

We also need to consider how the rise of DevOps and agile methodologies has underscored the importance of faster developer pipelines in enabling collaboration, transparency, and continuous improvement within development teams. DevOps emphasizes the integration of development and operations teams, automation of manual tasks, and alignment of goals and processes to accelerate software delivery and enhance quality. Agile methodologies promote iterative development, frequent releases, and customer feedback to drive innovation and adaptability. Faster developer pipelines are essential for supporting these agile and DevOps practices, enabling organizations to deliver high-quality software efficiently, respond to change rapidly, and foster a culture of continuous learning and improvement.

Delivering Progressively

Modern DevOps typically features progressive delivery - a cutting-edge approach that prioritizes controlled, gradual software releases, allowing teams to minimize risks, gather feedback early, and improve the quality of applications. This methodology is anchored in two main practices: intelligent deployment strategies and feature flags (or feature toggles). Each plays a crucial role in achieving a seamless, iterative release process that aligns with modern DevOps principles.

Progressive delivery seeks to minimize risk by deploying new versions of the software alongside existing ones, carefully controlling when and how the new version is exposed to users. The canary deployment strategy is a good example. In a defined progression, a small subset of users get the new version of the software. The performance of the new version, using both technical and business metrics, is evaluated against a baseline. If everything looks good, more and more users will be moved to it until all users are on the new version and the old is retired. If the new version is worse, it is rolled back and users are routed to the old.

This method acts as a real-world litmus test, providing invaluable insights into how the new version of the software performs under actual use conditions. It minimizes the impact of potential issues on the broader user base, ensuring that any problems are identified and rectified early in the release cycle. This approach not only enhances the reliability and stability of applications but also builds user trust by reducing the incidence of widespread issues.

Feature toggles, on the other hand, offer a dynamic way to manage features within an application. They allow developers to enable or disable features without deploying new code, thereby decoupling release from deployment. This flexibility is crucial for testing in production environments, facilitating A/B testing, and gradually introducing new functionalities. Feature toggles empower development teams to experiment and iterate rapidly, fostering a culture of innovation and continuous improvement. Features may be progressively rolled out to larger and larger cohorts like software versions undergoing canary testing, however with controls at the feature level, rather than the version, validation is more fine-grained.

The benefits of progressive delivery are clear. Progressive delivery:

- Significantly reduces the risk associated with deploying new features, as it offers a mechanism for real-time feedback and incremental adjustments. I
- Enhances operational efficiency by allowing for more predictable deployments and minimizing the downtime typically associated with traditional release processes.
- Fosters a closer alignment between development teams and end-users, as it encourages ongoing feedback and engagement, leading to more user-centric applications.

Embracing progressive delivery through canary deployments and feature toggles represents a strategic evolution in software development and deployment. This approach aligns with the agile, resilient nature of DevOps practices, offering a path towards more reliable, user-focused, and iterative software delivery.

Accelerating Quality and Security

Another challenge developers encounter with DevOps is the need to deliver with speed and quality. DevOps emphasizes rapid iteration and frequent releases to deliver value to users quickly. However, focusing only on speed can sometimes compromise the quality and stability of the software – most notably around secure code. A team shipping low-quality or insecure code will inevitably be told to slow down and do it right. Therefore, the only path to sustainable speed is high-velocity security and quality.

One key aspect of accelerating quality in the developer pipeline is the adoption of automated testing. Automated testing allows developers to quickly identify and fix bugs, ensuring that the code meets the required quality standards before release. Continuous integration and continuous deployment (CI/CD) pipelines play a crucial role in automating the testing and deployment processes, enabling faster and more frequent releases without compromising on quality.

In addition to automated testing, integrating security into every stage of the development lifecycle is vital for enhancing the security posture of software applications. Security measures such as static code analysis, vulnerability scanning, and penetration testing should be incorporated into the CI/CD pipeline to detect and address security vulnerabilities early on. By making security a priority from the initial stages of development, organizations can proactively mitigate risks and protect their applications from potential threats.

Scalable configuration

The explosion of microservices has led to an ever growing number of pipelines required by an organization that has adopted DevOps. For many teams, this presents a double penalty. First, each new service needs a bespoke pipeline authored for it. Later, when new practices are adopted or a new test is supposed to be added, updating all the pipelines is a lengthier process.

Through experiencing these headaches, it’s become clear that a modern DevOps solution must wrestle with the challenges of making it easy to create and update hundreds or thousands of pipelines.

An initial step for many is to move to pipelines-as-code where the pipeline definitions can be represented as text documents and saved in version control. This helps with auditing and enables new pipeline creation by copying and pasting.

While representing pipelines as code is a good step, the copy/paste model only speeds up new pipeline generation, but its impact on upkeep is minimal. Duplication is the wrong form of reuse.

Instead, look for approaches that separate the common from the variable. This may mean using the same pipeline for test environments and production, overlaying environment-specific variables. More sophisticated approaches enable using pipeline templates for many similar applications, leveraging additional forms of overlays. Done well; instead of maintaining pipelines for each application, there are only a handful of pipeline templates to maintain.

A particularly pernicious element of configuration to scale is governance. That brings us to our next section, Governance that Supports Developers.

Governance that Supports Developers

Across a variety of industries and countries, governance of software delivery is an unavoidable requirement. A responsible organization will ensure that before code reaches production, appropriate checks, whether automated or manual, have taken place. However, many DevOps transformations have failed to achieve both agility and governance.

Modern approaches will reconcile these challenges using one or more of several approaches. One path drives standardization aggressively, providing easy access to “paved roads” consisting of standard pipelines for standard technology stacks and architectures. If the development team follows the norms, a pipeline crafted by a central team and made available through templates will “just work.” The central team can be concerned with maintaining a handful of pipelines and ensuring they meet the latest governance standards. Because the pipelines work, the application team doesn’t need significant edit access, they might not even need to know what tools are running their pipelines so long as logs are delivered to useful places.

Another option seeks to separate the pipeline definition from the enforcement of governance policies that apply to that pipeline. For example, if an organization had a policy that all pipelines releasing to production must have an approval process attached to them, traditionally that could only be assured throught trust in whomever had permissions to delete approval steps from pipelines. So, edit permissions to pipelines would be limited to only a few.

If policy is separated, and defined outside the pipeline, then pipelines could be automatically evaluated for compliance with the policy. Ideally, at the moment of saving a change to a pipeline or executing a release, the pipeline would be evaluated for compliance and actions on non-compliant pipelines would be blocked. A leading approach to achieving this model is Policy-as-Code, using technologies like the Open Policy Agent (OPA).

Where OPA policies are implemented as a governance layer on top of pipelines, organizations are free to empower individual application teams to control their pipelines. OPA policies defiinied centrally would ensure that any changes stay within the bounds of organizational controls.

As new controls are adopted, they can be implemented first as warnings and then as hard, enforced controls. New policies can therefore be implemented with fair notice and minimal impact across hundreds or thousands of pipelines. That's what modern DevOps governance looks like.

Tangible Benefits of Modernization

By modernizing your DevOps processes, you enable the adoption of new services and processes that can significantly enhance the efficiency and effectiveness of your software development lifecycle. DevOps, which combines development and operations practices to streamline the delivery of software, plays a crucial role in enabling organizations to innovate and respond to market demands quickly.

As we've mentioned several times thus far, the ability to automate repetitive tasks and workflows, which reduces manual errors and accelerates the pace of development, is the key benefit of modernizing DevOps processes. By implementing continuous integration and continuous deployment (CI/CD) pipelines, teams can automate the build, test, and deployment processes, enabling faster and more reliable delivery of software updates.

Modernizing DevOps processes also allows for the integration of new tools and technologies that support collaboration, monitoring, and feedback loops throughout the development lifecycle. By leveraging tools such as containerization, orchestration platforms, and monitoring solutions, teams can improve visibility into their applications, identify issues proactively, and optimize performance and scalability.

Here are some of the other key competitive advantages of DevOps modernization:

- Faster Time-to-Market: DevOps modernization enables organizations to streamline their development pipelines, automate manual tasks, and implement continuous integration and deployment practices. This results in faster delivery of software updates, features, and enhancements to customers, allowing businesses to stay ahead of competitors and respond quickly to market demands.
- Improved Collaboration and Communication: Modernizing DevOps practices fosters a culture of collaboration and communication among development, operations, and other cross-functional teams. By breaking down silos and promoting transparency, organizations can enhance teamwork, share knowledge, and align goals to deliver high-quality products efficiently.

- Enhanced Quality and Reliability: DevOps modernization emphasizes automated testing, code reviews, and monitoring practices that help ensure the quality and reliability of software applications. By implementing chaos testing frameworks, performance monitoring tools, and feedback mechanisms, organizations can identify and address issues early in the development lifecycle, leading to more stable and resilient applications.

- Scalability and Resilience: Modernized DevOps practices enable organizations to scale their infrastructure, applications, and services effectively to meet growing demands and fluctuations in traffic. By leveraging cloud technologies, containerization, and orchestration platforms, businesses can achieve greater scalability, flexibility, and resilience in their operations.

- Cost Efficiency and Resource Optimization: DevOps modernization helps organizations optimize resource utilization, reduce waste, and minimize operational costs. By moving to a cohesive platform, integration costs are eliminated. Accelerated builds consume fewer build-minutes and ephemeral environments reduce infrastructure consumption. By automating repetitive tasks, optimizing workflows, and adopting efficient deployment strategies, businesses can achieve cost savings, improve efficiency, and maximize the value of their investments in technology and talent.

- Continuous Learning and Improvement: Modernizing DevOps practices encourages a culture of continuous learning, experimentation, and improvement within organizations. By embracing feedback loops, retrospectives, and data-driven decision-making, teams can iterate on their processes, learn from failures, and adapt to changing requirements, driving innovation and growth.

Strategies for DevOps modernization

It’s pretty safe to say that we’ve established that by adopting modern DevOps practices, companies can streamline their software development and deployment processes, improve collaboration between teams, and deliver high-quality products to market faster. Here are some key strategies for DevOps modernization:

- Embrace Automation: Automation is at the core of DevOps modernization. By automating repetitive tasks such as testing, deployment, and monitoring, teams can reduce manual errors, increase efficiency, and accelerate the delivery of software.
- Implement Continuous Integration/Continuous Deployment (CI/CD): CI/CD practices enable teams to continuously integrate code changes, test them automatically, and deploy them to production environments quickly and safely. This helps in reducing time-to-market and improving overall software quality.
- Foster Collaboration: DevOps modernization is not just about tools and processes; it’s also about fostering a culture of collaboration and communication between development, operations, and other stakeholders. Encouraging cross-functional teams and breaking down silos can lead to better outcomes and faster delivery.
- Focus on Security: Security should be integrated into every stage of the DevOps pipeline. Implementing security measures such as code scanning, vulnerability assessments, and access controls can help mitigate risks and protect sensitive data.
- Monitor and Measure Performance: Monitoring and measuring the performance of the DevOps pipeline is crucial for identifying bottlenecks, optimizing processes, and improving overall efficiency. Utilize metrics and analytics to track key performance indicators and make data-driven decisions.
- Embrace Cloud-Native Technologies: Leveraging cloud-native technologies such as containers, microservices, and serverless computing can enhance scalability, flexibility, and resilience in the DevOps environment. Adopting these technologies can enable faster innovation and deployment cycles.
- Invest in Training and Upskilling: DevOps modernization requires a skilled workforce that is proficient in modern tools and practices. Investing in training programs, workshops, and certifications can help teams stay up-to-date with the latest trends and technologies in the DevOps space.

harness

The Modern Software Delivery Platform™

Follow us on

𝕏 /harnessio

in /harnessinc

Contact us on

www.harness.io