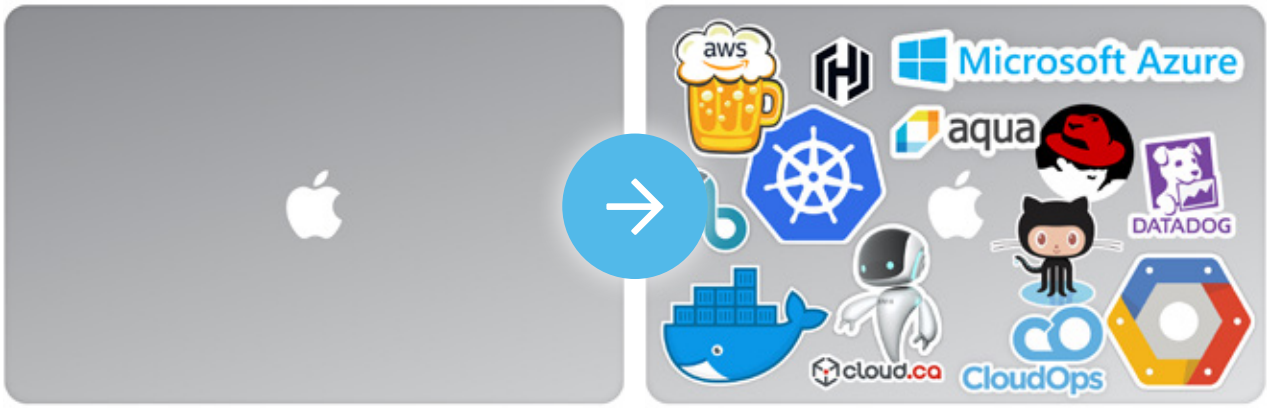
White Paper

How to Initiate DevOps Transformation

by Assessing Culture and Processes



CloudOps
Own your destiny in the cloud
Cloud and code agnostic, but opinionated

423 rue Saint-Nicolas, 2e étage, Montreal, QC H2Y 2P4
1 (888) 796-8364 | info@cloudops.com | @cloudops_

DevOps

DevOps is the combination of cultural philosophies, practices, and tools that increase an organization's ability to deliver applications and services at high velocity, evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. By removing traditional separations between development and operations, DevOps creates dynamic pipelines of continuous integration and delivery (CI/CD) that release software in faster cycles and smaller batches.

DevOps addresses the traditional conflict of interest between development and operations. While developers want to quickly release as many features as possible, operators desire stable and slow release cycles that minimize the risk of production environments breaking. Combined with a lack of automation for testing, building, and deploying releases, this often leads to wasted time, mismatched environments, and overlooked production defects. A lack of communication concerning needs and end-user problems diminishes trust between teams and leads to significant delays in value stream delivery. By enforcing collaboration throughout the software delivery cycle, DevOps methodologies downplay such risks and increase productivity and efficiency.

The adoption of DevOps tools and practices has consequently exploded in the past few years. The percentage of IT professionals working in DevOps teams increased from 16% in 2014 to 29% in 2018.[1] Those that successfully implemented its methodologies have demonstrated:

- 46 times more frequent code deployments;
- 440 times faster lead time from commit to deploy;
- 170 times faster mean time to recovery from downtime; and
- 5 times lower change failure rates.[2]

Teams that score highly in continuous delivery spend approximately 49% of their time on new work and 21% on unplanned work or rework, compared with teams who score poorly and spend 38% of their time on new work and 27% on unplanned work or rework.[3] DevOps accelerates release cycles and increases efficiency.

While there are obvious advantages to DevOps that many in the IT industry have benefitted from, roughly 31% are still not applying principles generally considered necessary for technology transformations.[4] The complexity of DevOps ecosystems makes it easy to underestimate the scope of DevOps transformation and overlook broken systems that need to be addressed. Executives are especially prone to making this mistake.[5] As information is typically filtered and sanitized on its way to the C-suite, executives often receive incomplete understandings of the tools and processes sustaining their software delivery pipelines and any gaps in their transformation strategies. This can lead to them overestimating the success of their DevOps initiatives and not allowing change where it's needed most. The planning and implementation of DevOps transformation initiatives require both executive buy-in and commitment to cultural change before a blueprint for innovating processes and tools sustaining software delivery pipelines can be defined. 43% of organizations are pushing out changes weekly, daily, or continuously. The increased speed of change introduces difficulty in understanding and reviewing the security consequences. There is not enough time to conduct exhaustive testing or reviews and the risk profile changes constantly.

[1] Puppet, Splunk. 2018 State of DevOps Report, p. 14.

[2] Nicole Forsgren, Jez Humble, Gene Kim, Accelerate: Building and Scaling High Performing Technology Organizations (Portland, 2018), p. 10.

[3] Accelerate, p. 52.

[4] Rob Stroud and Elinor Klavens with Eveline Oehrlich, Aaron Kinch, and Diane Lynch, DevOps Heat Map 2017, Forrester, (Cambridge, 2017)

[5] Rob Stroud and Elinor Klavens with Eveline Oehrlich, Aaron Kinch, and Diane Lynch, DevOps Heat Map 2017, Forrester, (Cambridge, 2017)

DevOps Culture

Culture is at the heart of DevOps transformation and should be addressed before all else. DevOps organizations are not monolithic or hierarchical in structure and are therefore not slow to pivot or deliver software releases. They are instead comprised of many small teams, each of which is multi-disciplinary and able to act quickly and autonomously. This structural change depends on cultural change to ensure teams remain aligned while independent.

The Westrum model can be used to describe culture with a scale that ranges from pathological to bureaucratic to generative. **Pathological** cultures are fear-based, power-oriented, and uncooperative. People often distort or withhold information for political reasons and shirk responsibilities in response to practices that punish novelty and failure. **Bureaucratic** cultures are rule-oriented. They involve more cooperation than pathological cultures but are still defined by narrow responsibilities. Novelty, failure, and inter-departmental communication are systematically discouraged. **Generative** cultures are performance-oriented and characterized by high levels of cooperation and openness. Risks are shared and failure leads to inquiry, creating a culture that trains innovation. Statistical analysis has shown that team culture correlates strongly with organizational performance and that high-trust, generative cultures are the foundation of high-performance.[6]

Generative cultures enable organizational structures of many small teams. A firm commitment to creating and maintaining a generative culture is a prerequisite for DevOps toolings and processes. Businesses that strive to integrate these nine principles into their team interactions and structures will help promote generative cultures.

Removing Silos

DevOps is all about removing the silos that have traditionally separated development and operations. It's about making developers aware of operational constraints so that they can own the solution, while simultaneously making operators work with developers to enable visibility into infrastructure and application monitoring constraints. Tasks that usually took place later in the release pipeline are baked into the development phase, aligning everyone's goals and allowing code to be released in smaller batches, more quickly and frequently.

Command and Control

By removing traditional siloes, DevOps encourages alignment between teams. Instead of large teams taking ownership of specific tasks, DevOps has small teams taking ownership of individual projects. These teams are empowered to make their own choices and see projects to completion.

Trust

A culture filled with trust is at the heart of DevOps. Employees that are trusted by their managers are more likely to perform well and feel happy at work. When not micromanaged, they can take initiative for the organization. Teams that trust each other are more likely to work in alignment and collaborate. Trust is created by open communication and kept promises.

Managing Failure

DevOps philosophies view failure as something to be expected rather than avoided. When failure happens, it's acknowledged and managed. This gives employees the freedom to hold themselves accountable for their mistakes, learn from them, and improve both themselves and the organization.

No Fear

When failure is accepted and managed instead of reprimanded, teams lose their fear of failing. This prevents stagnation and encourages an honest flow of information between teams.

Experimentation

Employees that lose their fear of failure are more able to experiment and take risks. When management encourages experimentation, teams can find creative solutions to sustain innovation.

[6] Humble, Molesky, O'Reilly, Lean Enterprise: How High Performance Organizations Innovate at Scale

Knowledge Sharing

Teams are open to learning and share what they learn with other teams. Failure is seen as an opportunity for all teams to grow from, and teams know more about what has worked for the organization and what hasn't. Knowledge-sharing helps organizations take full advantage of experimentation.

Empowerment

DevOps empowers teams to own the full software release cycle, starting from development and ending with production. They are empowered to make their own choices.

Business value

Well executed, DevOps will center your development and operational teams on business imperatives. It will allow you to deliver features faster and with the ability to quickly verify their impacts on your product. The successful adoption of DevOps cultural practices will lead to high-velocity software releases that are focused on customer needs and that position you as a market leader and innovator.

DevOps ultimately isn't a fixed destination but a constant process of improvement. Once principles of trust and learning are instituted, teams become empowered to act quickly. They can be allowed to take initiative, experiment, and share knowledge. Teams become aligned yet autonomous and are therefore able to release software at a greatly accelerated pace, overcoming the inherent conflict of interest that traditionally created tension between development and operations. Changing culture first will empower teams to create processes and choose tools that most suit their use cases and create avenues for feature velocity to accelerate.

An organization that hires expertise for building and operating advanced CI/CD pipelines without changing culture will be disappointed by the result. When Google decided to investigate what made their teams effective in a two-year research project that accounted for over 180 active teams, they found that "who is on the team matters less than how the team members interact, structure their work, and view their contributions."[7] While DevOps tools may contain the horsepower to deliver software quickly, the processes that sustain continuous integration and delivery are enabled by cultural shifts.

Building and Executing a Strategy for DevOps Transformation

Once leaders have shifted the culture of an organization, they can focus on defining a blueprint for adopting tools and processes that will sustain accelerated software delivery. As DevOps is an organization-wide approach, it should be adopted incrementally. While it will ultimately mean three unique CI/CD pipelines in the application code, the application platform, and the infrastructure respectively, transformation initiatives should focus on one realm at a time. Application development is typically the first realm to integrate DevOps tools and practices as it offers the most immediate business value. The application platform and infrastructure tend to follow, and their transformation will each support feature releases in application development.

Differentiated Value
Application Development/Code

Undifferentiated Value
DevOps App Platform/Toolchain

Undifferentiated Value
Infrastructure

Each realm must implement complete DevOps pipelines. The perception that DevOps is a role that can be filled by an engineer or a team should be avoided in each realm. Not only is there a lack of qualified technical talent, but any ideas, products, and projects introduced by digital unicorns will only succeed if aligned with operational models, technologies, and processes across the organization.

[7] Google, "The Five Keys to a Successful Google Team," ReWork blog. November 17, 2015.

DevOps Processes

Value Stream Mapping

Team leaders can focus on providing visibility into their existing processes. Value stream mapping should be the first step to improving processes. It becomes a tool to further cultural change by providing data that you can measure your performance against. Value stream mapping is the exercise of realizing which products are attached to which departments and which processes are sustaining their delivery. It evaluates the success of each value stream and identifies roadblocks and areas where resources can be shared and processes automated. By mapping value streams before evolving processes, teams will have the means to put practice into action.

Value stream mapping allows teams to embrace lean principles, which always emphasize first delivering minimum viable products that fulfill immediate customer needs. By skimping on additional unnecessary features, organizations can pivot ruthlessly according to customer responses. This has been shown to shorten product life cycles and eliminate waste. Ultimately, lean principles allow teams to increase profitability by responding quickly to customer needs. Once processes have been simplified and understood through value stream mapping, they can be aligned with DevOps approaches and the SAFe DevOps Health Radar.

Within individual realms, technical managers should focus on transforming one team at a time. They should lead individual teams to independently work on innovative projects with new tools and processes and take advantage of opportunities to learn from failures. Successful DevOps transformations often begin as ripples in a pond that later expand to farther corners of the business.[8] The success seen by individual teams can spread to other teams, and then to other departments.

DevOps Pipelines

Pipelines should consist of four distinct phases: continuous exploration, continuous integration, continuous deployment, and release on demand. Technical leaders should understand their current pipeline before initiating action plans for change. The SAFe DevOps Health Radar from Scaled Agile can be used to assess the performance of teams in each phase, and provide a benchmark for measuring future improvements. Release cycles move clockwise around the radar, improving in performance as they get closer to the center. Advanced organizations can iterate through all steps and release new features in under twenty-four hours, flying across the radar. Those with legacy systems can take months as they crawl across or even sit in the Radar.

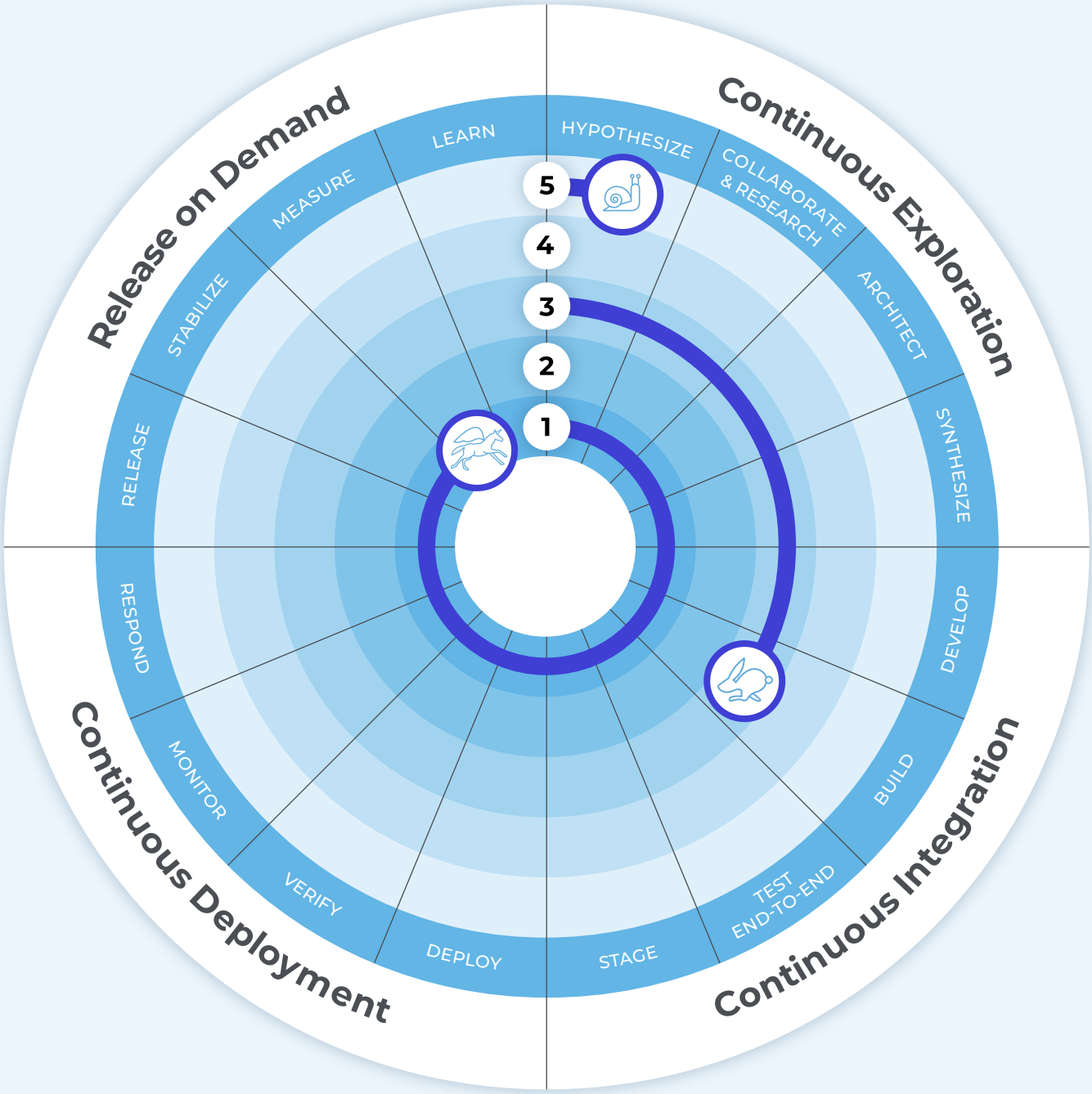
In assessing the performance of their pipelines, technical managers should give each team a score for all sixteen sub-dimensions. They should engage in value stream mapping, identifying the steps in each value chain, who is accountable in each step, and all lead and processing times.

The Health Radar will highlight which areas to focus on first. A delivery cycle can never go faster than its weakest link, which inevitably becomes a bottleneck for implementing change. Teams that are crawling across the radar should strive to walk in all sub-dimensions before running, instead of flying in some areas and crawling in others. It may take years of effort for organizations to synchronize flows closer to the center, but doing so will result in accelerated delivery cycles.

In assessing the current performance of their pipelines, technical managers should give each team a score for all sixteen sub-dimensions in the four phases. They should engage in value stream mapping to understand the overall flow. Identify the steps in each value chain, who is accountable in each step, and all lead and processing times. While team leaders may be surprised by how low their scores are in certain areas, they should use these scores as guidance on where to start.

[8] Puppet, Splunk. 2018 State of DevOps Report, p. 5.

The SAFe DevOps Health Radar[9]

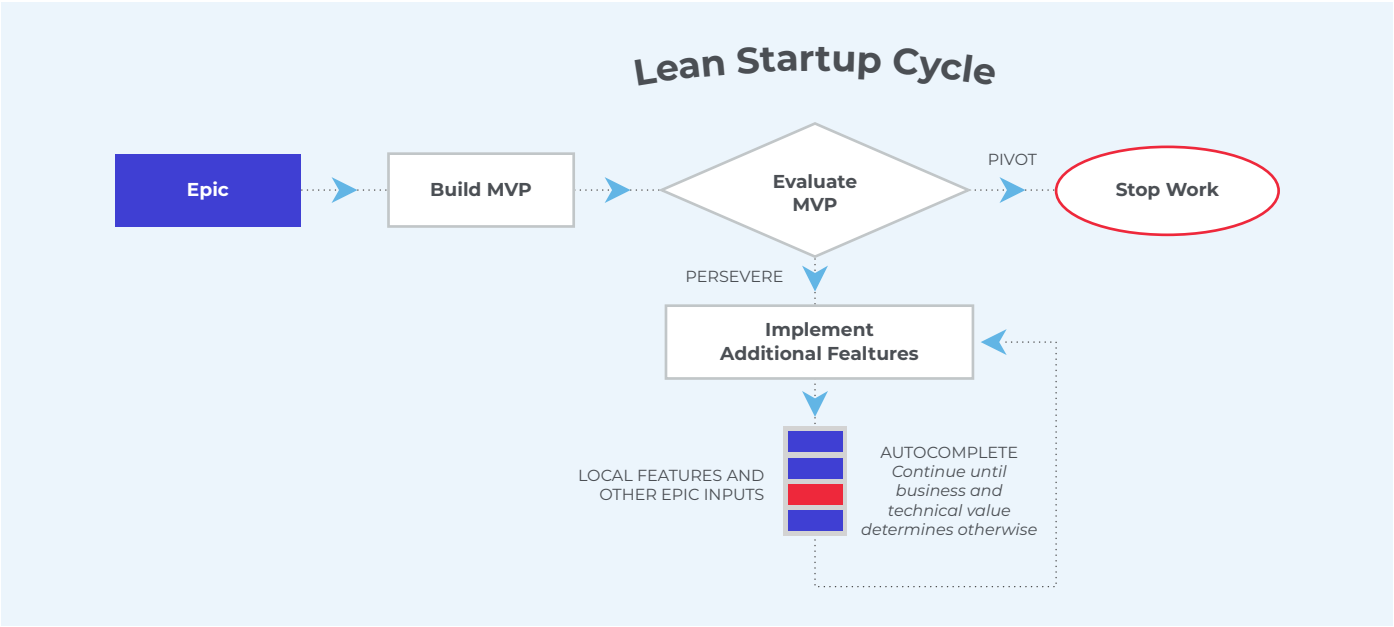


A delivery cycle can never go faster than its weakest link, which inevitably becomes a bottleneck for implementing change. Teams that are crawling across the radar should strive to walk across the radar before running, instead of flying in some sub-dimensions and crawling in others. It may take years of effort for organizations to synchronize flows closer to the center, but doing so will increase the velocity and efficiency of feature release cycles.

[9] Scaled Agile, DevOps SAFe Radar.

Continuous Exploration

Through engaging in practices of continuous exploration teams can provide each other with honest feedback and compare their performance to the industry standard. It is an extension of the lean startup methodology, which acknowledges that problems often come from persevering for too long, and encourages teams to retrospect, learn from their mistakes, and improve with each project. Business agility depends on decision-makers being informed enough to know when to persevere and when to pivot, and having the means to quickly change direction when required. Continuous exploration enables the former. Strong performance in this phase will allow teams to identify the viability of diverse ideas and become clear about their goals.



Hypothesize

All feature releases should begin with clearly defined hypotheses that can consistently be validated throughout the pipeline. Hypotheses should ideally be expressed as minimum viable products (MVPs) to be confirmed or removed and allow teams to persevere or pivot. Products and features can be difficult to measure with traditional accounting standards, so use associated accounting metrics that measure real engagement. Team skills in lean startup and innovation will help teams fly through this sub-dimension, always expressing hypotheses as MVPs with measurable outcomes. In contrast, teams sitting in this sub-dimension will have vague and poorly defined ideas that will take months to release.

Collaborate and Research

Collaborate with a product manager and research the needs both of your customers and multiple stakeholders. Your hypotheses will, through this process, become more closely aligned with market needs. Lean User Experience (Lean UX) processes can be used to implement functionalities in minimum viable increments (MVIs) and determine success by measuring results against benefit hypotheses. The validity of a benefit hypothesis can be tested by breaking features into minimum marketable features (MMFs). Teams sitting in this sub-dimension will have product management roles and responsibilities that are neither defined nor followed. They will fly through when product management collaborates with business, development, and operations.

Architect

Architect the solution for continuous delivery. Systems that cannot be readily tested cannot be readily changed, and systems designed for testability reduce the amount of time required for all jobs. Enable the ability to deploy and verify in both production and release on demand by separating deployments to production from releases and hiding new functionalities under toggles. Decouple release elements as different parts of a solution will require different release strategies.

Shifting left becomes important as you architect the solution. Infosec concerns, in particular, should start early. Identify potential security threats and automate security into your architecture, infrastructure, and application. Build logging and telemetry capabilities into applications and the solution as a whole to account for future operational needs. Build capabilities for fast recovery or fix-forward and to remove or downgrade services when there are incidents or high loads. Solutions should be architected to enable and shift various strategies over time, remaining aligned with value delivery.

Synthesize

Synthesize the hypothesis, research, collaboration, and architecture into a common vision, roadmap, and program backlog. Skills in feature writing, behavior-driven development, economic prioritization, and planning will be helpful in this sub-dimension.

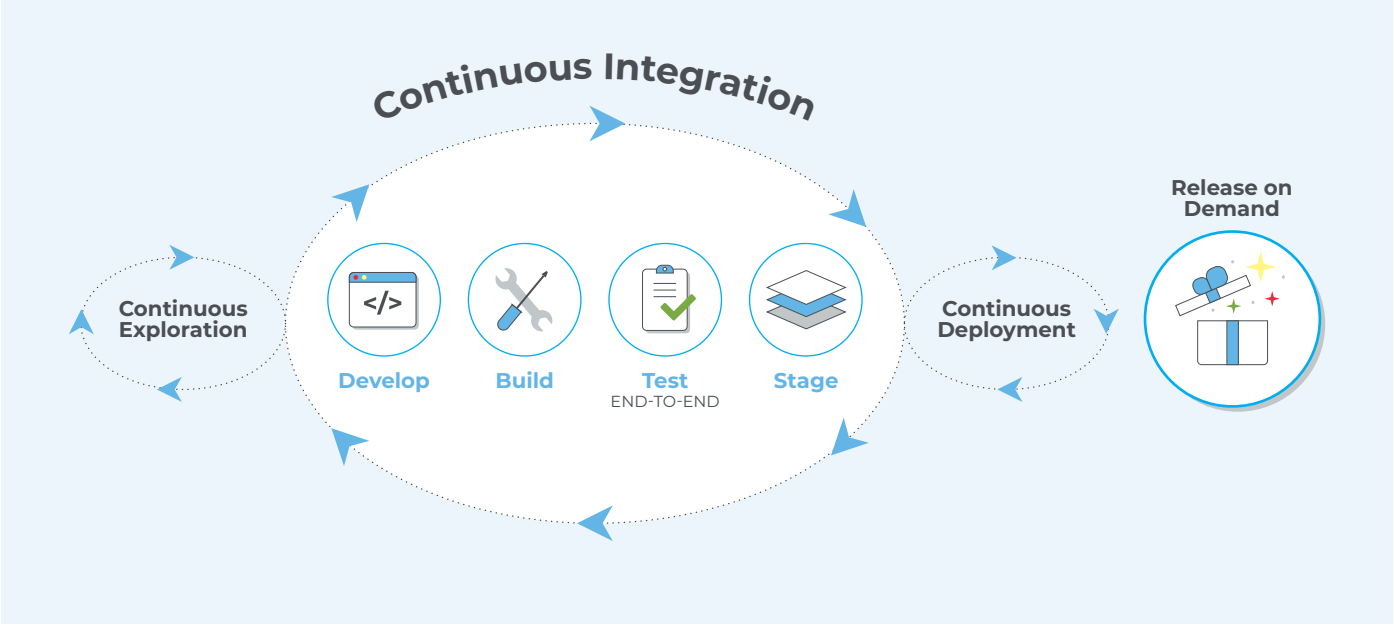
Marketing and product management teams should be familiar with feature writing, which describes justifying implementations and costs with benefit hypotheses to provide business perspectives for scope decisions. It refines and completes acceptance criteria during backlog grooming. Feature writing reflects both functional and non-functional requirements and tends to fit into single-program increments, which could span multiple sprints.

Cadence-based program increment (PI) planning is critical for Agile deployments. All team members should attend PI planning sessions, which should take place on a regular cadence that covers multiple sprints. Product management can own feature priorities, agile teams can own story planning and high-level estimates, and engineering and UX teams can become intermediaries for governance, interfaces, and dependencies. Program backlogs should become collections of minimum marketable features that are prioritized and matched to delivery capacity.

DevOps requires feedback from all teams, and teams should have defined processes for researching and improving ideas before they start building. Product owners should be empowered to critically define needs in collaboration with all other teams.

Continuous Integration

DevOps teams continuously integrate components, systems, and solutions using a prioritized backlog from continuous exploration. This is the build phase and it is owned by the developer. As much as possible should be automated, and the process of committing, containerizing, testing, and deploying code should ideally take no more than a few minutes. Continuous integration covers the development, building, testing, and staging of features.



Develop the Solution

Teams developing the solution will benefit from having skills in breaking features into stories and tasks, version control, engineering practices, pair work, application telemetry, and threat modeling.

Workflow steps, data variations, and planning poker are some examples of techniques for splitting features into stories within boundaries of PI and iteration. All assets, including requirements, code, configuration, and tests, should be maintained under version control, which improves traceability for automating compliance.

Pair work is essential for improving system quality, design decision-making, knowledge sharing, and team velocity. It is broader and less constraining than pair programming but is still a collaborative effort of any two team members. Between 20% and 80% of a team member's time should be spent pairing. Emphasize spontaneous pairing and purposeful rotation, remember to occasionally reshuffle pairs, be mindful of how pairs get along and prevent reviews from becoming ego matches.

Application telemetry is used for identifying problems within production incidents and should cover all levels of code - from methods to components to services for the entire application. Application design should account for the operational health of its telemetry systems. Features should include the ability to measure a benefit hypothesis against both leading and trailing indicators.

Program backlogs should exist and be used to manage daily work. Code should be checked multiple times a day, tests should be written into features early on in the build phase, and built-in quality principles, such as pair work, should be regular.

Build Continuously

Source files should be compiled into deployable binaries, code functions should be verified, and dev branches and trunks should be merged. Ensure that your team has skills in continuous code integration, build and test automation, trunk-based development, gated commits, and application security.

Building and testing should have automated components. Initiate builds often, preferably with every commit, and run unit tests and static code analysis as part of every build. Visualize and monitor processes, test APIs depended on by components, and report failures immediately. Broken builds should be the highest priority, so ensure gated commits are in place to prevent broken code from blocking other parts of the pipeline and only merge commits that have made it through building and testing into the trunk. Apply tools to automatically identify security vulnerabilities in the code during the build process. Assess open source libraries continuously for known vulnerabilities to identify risks during development or build processes.

DevOps requires circular patterns of building, testing, and deploying features. Flying through this sub-dimension will mean running builds on every commit, and including static code analysis and security testing in builds. It will mean using gated commits to prevent defects from entering the version control. Dev branches will be merged on every commit.

Test End-to-End

The aim of this sub-dimension is for change to be validated against acceptance criteria in an integrated production simulated environment. Skills in test and production environment congruity, test automation, test data management, service virtualization, and non-functional requirements (NFRs) are needed. Testing automation should be integrated as features are built. As manual testing can take a while, it should be reserved for strategic tasks.

Test environments should match production closely and configuration changes maintained under version control. Many team leaders only have 50% of their code in development and testing environments matched in production. Invest in more accurate testing. Functional, integration, regression, performance, security, exploratory, and penetration are all types of tests that should be run. Most but not all tests should be automated. Testing data should be stored in a repository for consistency. Emulate production data to ensure that tests reflect realistic situations. Service virtualization will provide the ability to spawn environments that match production with test and support different types of testing. Maintain environment data in source control. All features should have associated NFRs, system qualities that support end-user functionalities and system goals. Healthy performance in end-to-end testing will show build triggers that automatically deploy to production as well as tests that are automated, run in parallel, and have changes that are fully validated after every commit.

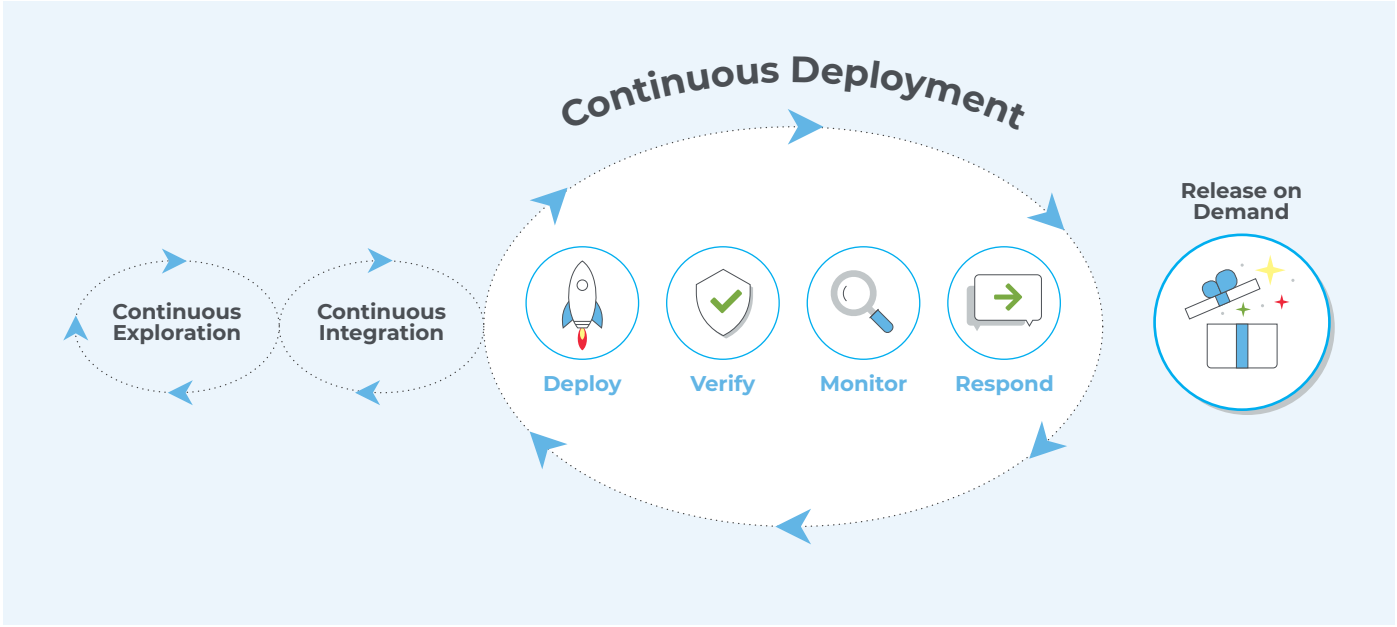
Validate on a Staging Environment

Fully validated systems are ideally hosted in production-grade environments before being deployed in production. Knowledge about maintaining staging environments, blue-green deployments, and system demos is useful in this sub-dimension. Maintain a staging environment that matches production. Maintain two environments, idle and live, for Blue/Green deployments, and allow switching environments to be done through a load balancer. System demos should involve new features that work together and with existing functionalities, staging environments that resemble production, and should receive feedback from program stakeholders. Teams that fly through this sub-dimension will auto-deploy stories, changes, and infrastructure to staging environments and have validated deployments.

In continuous integration, developers should automate as much as possible and shift operations left. This will help prevent deployments and releases from seeing operational problems.

Continuous Deployment

Improved performance in continuous deployment involves reviewing each sub-dimension and comparing the performance of all teams to the industry standard. Features are deployed to production in this phase, and teams aspire to consistently release features that can then be utilized by the business.



Deploy to Production

DevOps encourages the frequent deployment of low-risk changes into production. Skills in dark launches, feature toggles, deployment automation, selective deployments, self-service deployments, and version control are needed. Dark launches allow deployments to be separated from releases and enable the background and foreground processes of production environments to be tested before new functionalities are exposed to users. Feature toggles, which separate deployments from releases, should have hidden functionalities, be tested to ensure toggles are off and be treated with care to minimize the risk of toggle overload that can complicate code and testing.

Automate deployments to production, and eliminate manual steps from code commit to production deployment. Environment and package information should be stored in version control and the deployment process should be tested itself. Nonetheless, complete automation from code commit to production deployment is not always possible, in which case the deployment of the package to production should still be automated and only require simple controls. Selective deployments can be differentiated by data center, geography, or customer, and allow for a gradual deployment to production for incremental verification or release. Features should ultimately be deployed continuously throughout each iteration, with dev teams initiating deployments directly with continuous delivery pipeline tools. If releases are fully decoupled from deployments, then dark releases can be the norm.

Verify the Solution

No deployment solution should be released to end-users unless it has been shown to behave as expected in production. Production testing, build test automation, and end-to-end data management and non-functional requirements will be needed to ensure deployments are verified in production. Features should be tested in live environments, and this includes both functional and non-functional testing. Run synthetic transactions through services to verify its utility and warranty. Strive to run automated production tests and feed monitoring systems on an ongoing basis. Teams successful in this sub-dimension will be able to roll back failed deployments instantly or have them fix forwarded through the entire continuous delivery pipeline.

Monitor for Problems

Full-stack telemetry, visual displays, and federated monitoring will lead to quantitative measurements of systems and user behavior in real-time. Monitor both technical and business data. Applications should be architected to support full-stack telemetry for clearly logging and reporting meaningful events and activities. Visual displays can visualize telemetry for the entire organization, and visible information radiators should project the health of the application and its systems at all times. Information about key DevOps metrics, such as average lead time and time since the latest deployment, should be included as well as data that has been aggregated from various sources into a collection point. Flying through this sub-dimension will mean having a federated monitoring platform that provides one-stop access to full-stack insights, allowing data to be used to gauge system performance and business value.

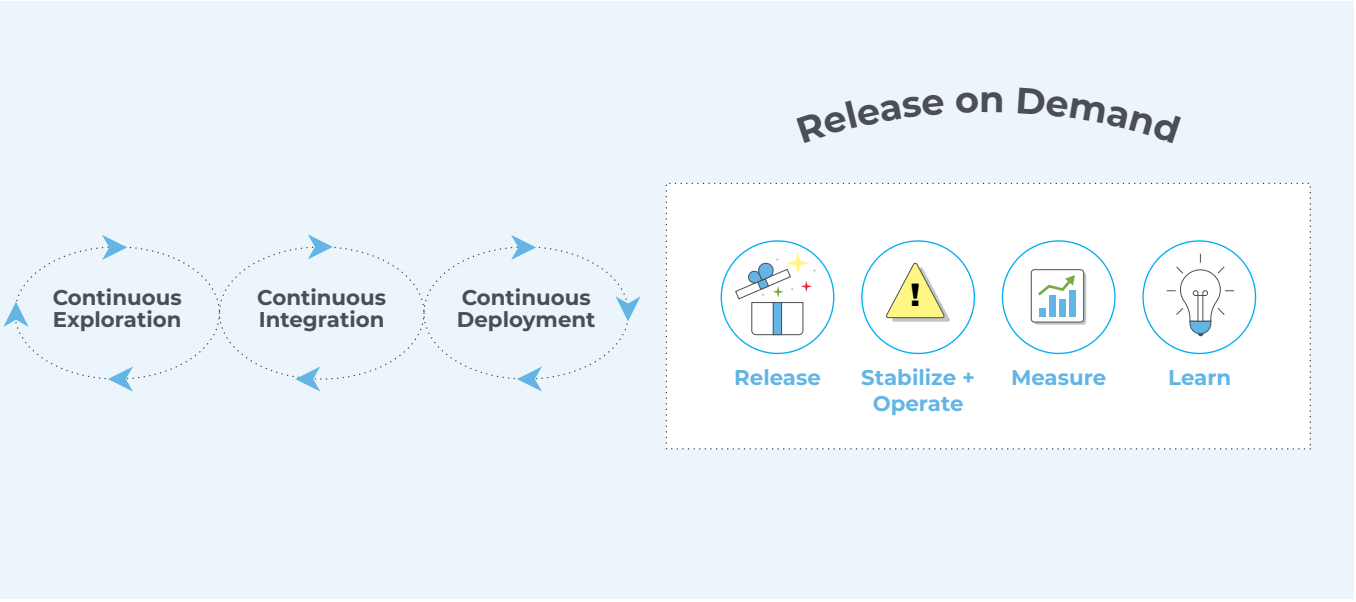
Respond and Recover

The purpose of this sub-dimension is to proactively detect and resolve production issues before they disrupt the business. Skills in proactive detection, cross-team collaboration, roll-back and fix-forward, immutable infrastructure, and version control are helpful. In proactive detection, deployments should be decoupled from releases to ensure problems are detected and not exposed to customers. Proactively look for problems and practice disaster recovery situations. Self-sabotage exercises, such as Netflix's *Chaos Monkey*, can help build resilience. Cross-team collaboration is critical for DevOps processes, and dealing with production issues must be everyone's responsibility. Teams from across the value stream should collaborate to solve production issues and identify root causes. The infrastructures of cloud native technology stacks are immutable, and this can be encouraged in this sub-dimension by deploying all changes through the continuous delivery pipeline. Avoid making changes directly in the production environment as this creates configuration drifts and has inherent risks. Teams that are successful here have monitoring systems that are based on carefully designed tolerance thresholds and alert them to dangerous conditions. Developers are held responsible for supporting their code and proactively fixing issues through the pipeline before users are impacted.

In continuous deployment, developers should automate as much as possible and shift operations left. This will help prevent deployments and releases from seeing operational problems.

Release On Demand

Through continuous deployments, businesses release features on demand, but not all features are necessarily or immediately visible to end-users. How features are ultimately consumed by end-users depends on decisions made in the release on demand phase of the DevOps Health Radar. While the frequency of feature releases is addressed in continuous integration and deployment, how, if, and when they are consumed by end-users depends on the business's requirements. In this phase, businesses ensure features provide optimal business value by releasing, stabilizing, measuring, and then learning from the complete cycle.



Release

Use skills in feature toggles, canary releases, release element decoupling, and dark launches to release features incrementally or all at once. Canary releases, the practice of releasing features only in certain regions or to certain users, can provide the ability to release value to a part of the user population and add or remove user segments based on business decisions. Combined with selective deployments, this can enable incremental deployments and rollouts and make releases more strategic for the business. Teams that fly through this sub-dimension can quickly and easily deploy features to individual segments of the user population and refactor feature toggles when no longer used.

Stabilize the Solution

Once a solution has been released, there must be high levels of business continuity, application service levels, and data protection. Teams will benefit from skills in disaster recovery, continuous security monitoring, architecting for operations, and non-functional requirements in this sub-dimension. As failures will occur, failover mechanisms will help services resume quickly without service interruption. Continuous service monitoring will detect intrusions and attacks and penetration testing will focus on preventing known vulnerabilities from getting to production. Healthy performance will maximize resiliency by rehearsing recovery procedures for faults that have been deliberately injected into the production environment.

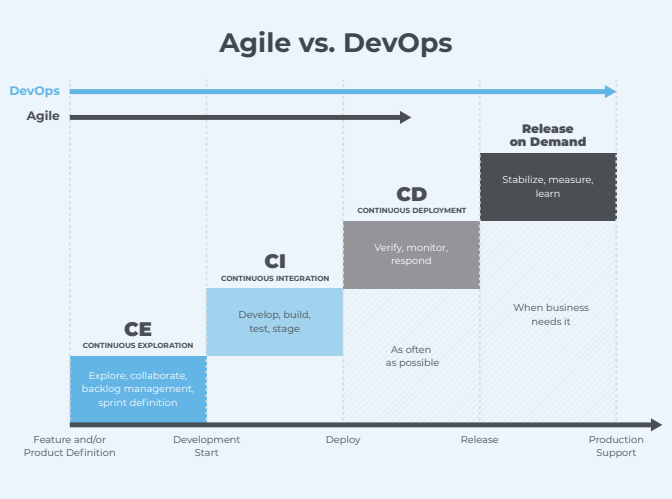
Measure Business Value

Aggregate qualitative and quantitative feedback with innovation accounting to objectively validate hypotheses and inform the decision to pivot or persevere. Application telemetry will help evaluate the business implications of hypotheses formulated in continuous exploration by measuring leading and lagging indicators. This should be done before operations take custody of a feature.

Learn and React

Self-reflect to know when to pivot or persevere. Skills in lean startup, relentless improvement and value stream mapping will once again come in handy. Relentlessly improve by focusing on the root causes of bottlenecks and research incidents to identify where to improve. Continuously applying current and future state value stream mapping will reduce time-to-market and clarify objectives. This will help principles of continuous learning and experimentation, hallmarks of DevOps philosophies, become ingrained in the DNA of an organization.

Release on demand is an extension of traditional Agile methodologies, which end after continuous deployment and don't utilize DevOps tools and practices to decide how features will deliver business value.



This is largely because the wider DevOps tooling and application platform ecosystems can be overwhelming to navigate. The number of **cloud native tools** alone has proliferated. It can take enterprise architects months to become familiar with the quickly evolving, diverse options. The expertise required to design, build, and operate modern DevOps platforms should take years to develop. Likewise, the wider planning and implementation of DevOps transformation strategies typically takes years and requires both knowledge and care.

Conclusion

When embarking on a DevOps transformation, it is important to define a blueprint that systematically targets the weakest points in a pipeline and uniformly increases the efficiency of processes. Culture must be treated first and throughout the entire organization, as DevOps tools and processes are enabled by principles of trust, learning, and experimentation amongst others. Pipelines must then be carefully assessed and strategically planned. By implementing pipelines of continuous exploration, continuous integration, continuous deployment, and release on demand, organizations can ultimately accelerate the velocity of software releases and follow business goals.

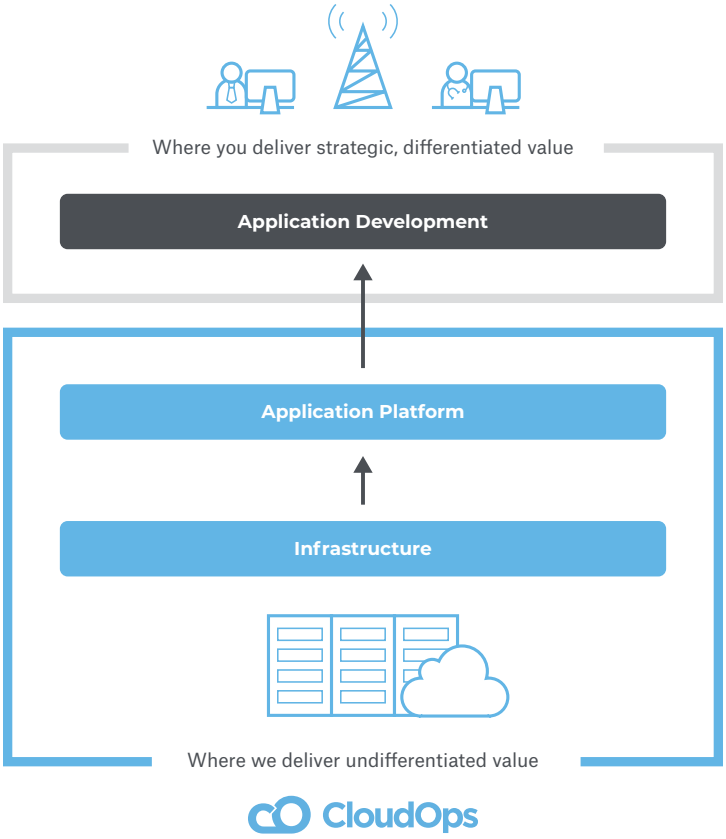
For that reason, around 47% of CEOs in 2017 faced pressure from their boards to digitally transform.[10] Once they have done so, their organizations:

- Become twice as likely to meet commercial goals concerning profitability, market share, and productivity;
- Become twice as likely to exceed noncommercial performance goals concerning the quantity and quality of products and services, operational efficiency, customer satisfaction, and overall mission goals; and
- See a 50% increase in market capitalization growth over three years.[11]

[10] Kasey Panetta, "Gartner CEO Survey," Gartner.com, April 27, 2017

[11] Nicole Forsgren, Jez Humble, Gene Kim, Accelerate: Building and Scaling High Performing Technology Organizations (Portland, 2018), p. 10.

DevOps enables digital transformation as it accelerates software delivery and the ability to pivot when needed, comply with regulations, and respond to customer feedback. It's been estimated that CIOs who have not evolved their team's capabilities to keep up with this demand will be displaced from their organization's digital leadership teams by 2020.[12] DevOps is becoming a requirement in today's cloudy world, but the complexity of its landscape often makes transformation an ambiguous journey.



DevOps Transformation

With almost fifteen years of experience navigating the DevOps and application platform landscapes, CloudOps' teams of experts have a wealth of collective knowledge that let us complete projects more quickly and efficiently than those adopting DevOps for the first time.

Our services often begin with a **DevOps Platform and Practices Assessment**, which will provide you with a blueprint for aligning your business goals with your technical output. It will identify the key gaps in your infrastructure and application platform preventing business stakeholders from seeing optimum results.

We can build, operate, and transfer value in your application platform. **Contact us** to learn more about how we can help you focus on what truly matters for your business - your application development.

[12] Gartner. Gartner Predicts. 2016.

CloudOps is a cloud consulting and services company focused on helping customers own their destiny on the cloud. CloudOps uses open source cloud platforms and networking and offers multi-cloud solutions for software companies, businesses, and telecommunications providers.

CloudOps

Own your destiny in the cloud

Cloud and code agnostic, but opinionated

423 rue Saint-Nicolas, 2e étage, Montreal, QC H2Y 2P4
1 (888) 796-8364 | info@cloudops.com | @cloudops_