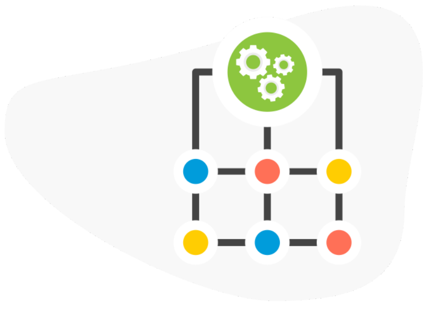
Effective DevOps:

Automation for Configuration, Collaboration & Precision

An eBook By RTS Labs



Contents

What is DevOps?

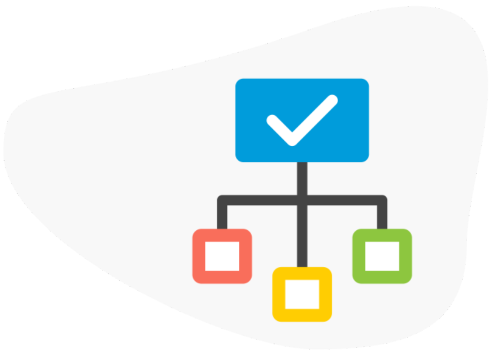
DevOps is the integration of company philosophies and tools which help organizations deliver software at much higher velocity than before while reducing complaints, errors and problems.

The Shortcomings of Traditional Software

In today's environment, traditional methods fall short.

What if the big change doesn't have any features that users want?

What if users don't have patience to wait until the big change and decide to move to another provider?



Today's software development requires constant deployment, multiple iterations and faces ever-changing expectations from customers.

Prior to the advent of DevOps, Development and Operations teams worked separately and in silos, where one wanted new features, and the other wanted stability. Now, their roles are merging and every single task has become everyone's shared responsibility.

The shared sense of goal-making inculcates team spirit, helping ship software before deadlines and making required changes at lightning fast pace.

This outlook has changed the delivery approach, too. Instead of delivering big changes after inordinate delays which could take as much as an entire year, smaller changes are implemented which take only a few weeks or days.

The heart of DevOps is an approach that incorporates feedback, adapts tiny changes and ships consistent deliveries.

Building Products with DevOps

Encourage Accountability

Code ownership and feature testing generally have been under the purview of either Development or Operations team.

Instead of separate teams handling coding and post production, a single team becomes responsible for the entirety of the processes involved in development and shipping of software.

If a team is working on a feature, then that team is responsible for everything surrounding that particular feature.

A uniform service like a messaging board keeps everyone on board with the status of the project and informs stakeholders about current issues in the system.

Continuous Delivery with DevOps (Everything-As-Code)

DevOps makes consistent delivery easy to do.

Considering the lines between Dev and Ops have been blurred to avoid physical resources altogether, traditional software tools can be applied to the Ops side as well, and that takes the form of Infrastructure-As-Code. Due to the advent of Public Cloud providers such as AWS, Azure, GCP, Digital Ocean, etc., traditional methods of purchasing, configuring and operating on physical resources is no longer the only option for a business to run its infrastructure. DevOps gains importance due to the strategic advantage of having the operational knowledge of infrastructure and being able to apply coding principles so that not only can an application be consistently delivered, but also the infrastructure in which it sits on. Now that hardware has been codified in any given cloud space, continuous delivery no longer suffers from resource bottlenecks that once hindered application delivery of the past.

Software pipeline

The software pipeline keeps track of all software processes, and allows for inputs and outputs to flow freely within the pipe to end up with an end-to-end means of building an application to its deployment into the desired environment.

A software pipeline tool should make it easy to test changes and ship them into production as soon as possible.

What are the Major Tenets of Software Delivery with DevOps?

Make it Work, Make it Right, Make it Fast

This is a common tenet found amongst the software development crowd, but it absolutely applies to the DevOps methodology. The tools that are used within the DevOps toolchain require code and configuration changes within themselves. But if DevOps is even being considered, new tools need to be reviewed and tested to solve for a given use-case, optimized to fit the the overall business need, and finally automated to be distributed throughout the business.

The DevOps Toolchain

Without the myriad combination of tools required for the entire software development lifecycle, DevOps is a pipe dream. The stages of the DevOps Toolchain are contested, but generally follow a pattern of: Planning, Creating, Verifying, Packaging, Releasing, Configuring, and Monitoring. There are many small tools for every single stage that allow for inputs and outputs to flow freely between them that results in true Continuous Delivery/Continuous Integration Architecture.

Process Automation

No longer are the days and weeks of waiting for servers and resources to be configured in a server closet or data center farm when their cloud-equivalents can be scripted and deployed within minutes; this is just one example of process automation. Elimination of pain-points sits at the heart of DevOps, and the easiest way to do that is by automation through various tools and scripts.

Process Automation makes the underlying resources with which people work on traceable, repeatable, and predictable. The major boon to automation is the resulting elimination of human error given a process, as well as saving time that could be better spent learning and enhancing. Following this precedent, every stage of the software development lifecycle gradually becomes less visible simply because they are automated away from human responsibility to suit the business' needs.

Continuous Learning

DevOps is a widely-debated topic, but none will argue against the fact that one must continuously keep up and train on the newest technologies and race for participation in alpha and beta releases of tools. Nothing in DevOps stays consistent, as requirements and needs constantly shift both internally, and externally; this means that no one can fall in love with the tools that they use, but are still expected to be their subject matter experts. The "one size fits all" approach is being replaced by a rapid, disruptive technological ecosystem and tools have become "made to order." It is then the burden of DevOps to be able to quickly learn and execute, but also to know when to fail quickly and pursue another tool that possibly better fits their needs.

Fail Safe and Fail Fast

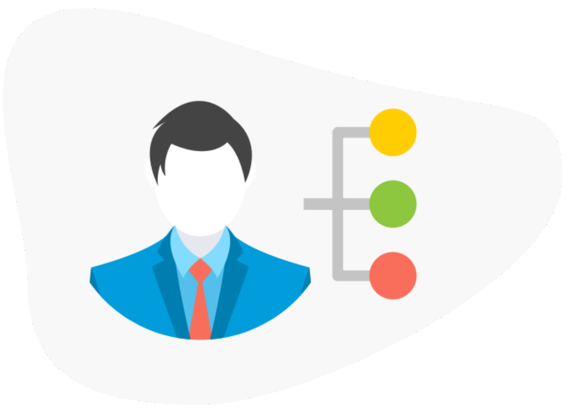
DevOps requires an attitude to consistently make breakthroughs in enhancements, but sometimes the tools that have been decided on won't be the solution to a given problem. While it might be easy to just jump ship from one tool and move on to the next, it's important to do so in a manner that won't cause things to break. By having entire structures templatized, scripted, and/or documented, DevOps encourages the idea that no one should be scared of failing since they can roll back to something that succeeds just as quickly as they failed. Canary releases and/or rolling updates are the bread and butter of DevOps-oriented software deployment cycles.

Quick Delivery

DevOps abandons the silo approach of development in favor of a cyclic approach where every team is consistent in their work and takes feedback from one another. Faster time-to-market goes hand-in-hand with an Agile methodology of making smaller enhancements or changes as opposed to giant releases. Developments teams can make these changes happen from their code, but the actual delivery of these changes gives DevOps importance. Making changes is one thing, but having them reflect in a production environment means having to change the approach to coding the applications in the first place. In today's development ecosystem, code ships are progressively getting smaller, and with the idea of making small changes, architectures are shifting evermore to portable microservices. Small changes are delivered quickly and keeps teams focused on that goal.

Be Proactive

After an incident occurs, its best to take into view whatever it is that caused it. Consider the following: *communicate early and consistently*. Resolving an incident is done easily if all communication channels are maintained and proper communication is carried through during an incident.



This plan has to be prepared prior to disaster striking. If not, you will be left scrambling until some form of order descends.

Ensure there's a communication plan, decide channels which you're going to use to communicate and keep a list of common issues and their solutions handy to tide through the problems.

It is extremely important to learn from past mistakes and be proactive in keeping extensive documentation of shortcomings that have been encountered, so that proper steps can be taken against the process to make sure the same mistakes are never made twice. A strong DevOps culture places a higher responsibility on enabling and equipping fellow team members across other specializations with the proper skills and tools needed to troubleshoot effectively.

Ryan Draga, the DevOps specialist at Scotia banks says that the biggest challenge with DevOps is complacency. Thinking that hackers are smarter than you and that's why anything that you do doesn't really count.

Audit Results Regularly

The remaining task is to run frequent audits to make sure the entire team is up to task and are good at resolving incidents. Documenting results and introducing better workflows for diagnosis and improvement is the key to betterment. The history of incidents can help zero down service level objectives, help monitor signals and visualize things that matter.

Monitoring Versus Alerts

Monitoring might involve keeping track of all running processes--and doesn't necessarily mandate sending an alert about each and everything that takes place. Monitoring ensures alerts aren't triggered for minor problems and attention is not diverted to mundane routine tasks. Careful monitoring and a resourceful alerting system keeps most incidents from happening. This provides oversight into system health and help keep troubles in check before they take an ugly turn.

Maintaining a backlog of project work and status updates keeps teams informed and reduces incidents before they happen. Review usage data to see which version of a feature is performing best against the goals defined during planning—and keep iterating and testing until the best version of that feature is determined.

Four signals that are most commonly taken are:

- Latency
- Traffic
- Errors
- Saturation

These essential signals should be monitored as a team to provide a great User Experience. Based on history of incidents, the company can understand service level objectives. Finally, the right signals are monitored to help visualize core things for all involved teams.

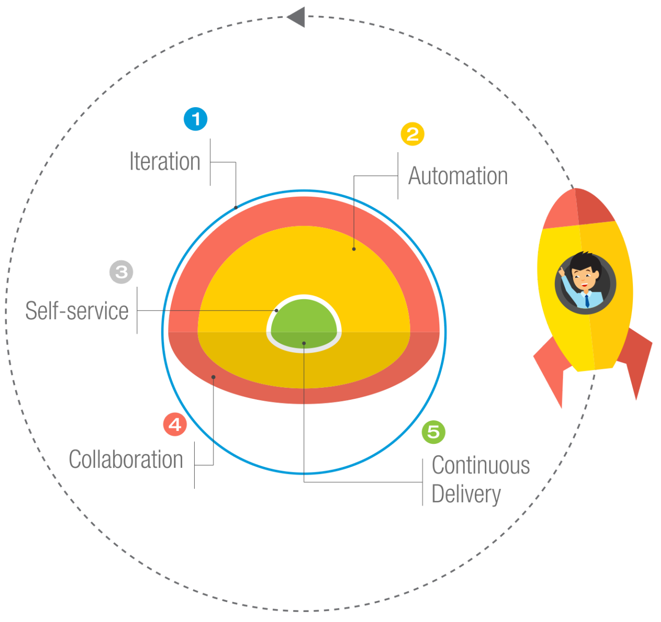
Automated Testing

Software testing is a time-intensive process that takes many man-hours. Instead of delegating the tasks to employees, automate software testing as much as possible to save man-hours and transfer them into more productive tasks.

Automation additionally reduces needless hires and keeps teams compact. This also reduces unnecessary communication between numerous members and reduces the task of keeping everyone updated.

Automate Yourself Out of a Job

In a world where constant change is the only constant, you can't cling to any one tool or methodology that you love; this idea extends to your job, and it's a common misconception that the more we are needed, the more secure our job is. To automate and free up more time for yourself for more important tasks means to not allow yourself to be the bottleneck of a given process, and more importantly, grant more bandwidth to take on more roles and greater responsibilities.



To automate yourself out of a job means to reduce the amount of time spent on any given task (or set of tasks) to allow energy and time to be invested in more important things. Automate yourself out of job and into the you want.

In every business there are parts that can be fully automated and parts that cannot be automated at all. The important thing is to properly identify what parts cannot be automated currently, and to be on the lookout for any new tools that potentially can.   Either minimize those non-automated parts to zero, r create processes that can efficiently handle those things with the least investment in manual work.

Major efforts should be made to the automation of all tasks, no matter how trivial. As day-to-day operations become more complicated with more (complicated) work, whatever that remains constant should be automated off of your plate to allow for more time to do the more important things.

Integrating agile approach into team management and teamwork is key here.

Spending time on processes to find out ones that sponge away most time and determining what could be done to reduce that lag should be the next thing on the block.

Figuring out pain points through monitoring and goal tracking helps with reducing time on minor tasks that doesn't necessarily require human hours to be completed.

Is Your Company Ready for DevOps?

Finally, the golden questions.

How do you determine if you're ready for DevOps?

What kind of processes need to be automated before one's ready for DevOps?

What kind of team framework should the team be in? Is the mindset right?

We need to answer all these questions before you decide if you're ready for DevOps or not.

As a team, you could follow certain rules and procedures.

For instance you could plan a two week sprint where teams are merged and different members can learn from each other.

The best thing to do to avoid problems is being proactive. Being proactive about the difficulties ahead and preparing for them in advance Is preferred to running to the hills when trouble strikes.

Based on past problems and a level of experience you can design and implement a solid strategy.

Every incident that occurs gives you the ability to learn and never repeat it.

We hope this guide provides a running start towards a new way of thinking about DevOps with a focus on software design, development, constant improvement and quality implementation.