IT OPS & DEVOPS PRODUCTIVITY REPORT 2013

TOOLS, METHODOLOGIES AND PEOPLE

TABLE OF CONTENTS

INTRODUCTION
TO IT OPS & DEVOPS PRODUCTIVITY SURVEY

“There never was a grand plan about thinking about DevOps as a word, I was busy working on a project which had Agile development and I was so envying these people about their methodology and I like to do something similar in my world system administration, so I called it initially Agile System Administration, but it’s pretty long. I just picked the word DevOpsDays as Dev and Ops working together because Agile System Administration was also too narrow on focusing on sysadmins only.”

Patrick Debois,
Father of DevOps

What is DevOps?

The word "DevOps" has been thrown around quite a lot lately. Job boards are awash with requisitions for "DevOps Engineers" with varying descriptions. What is it really? You might think that it's simply Dev + Ops in a single person or team, i.e. the ideal candidate for any organization that will burn themselves out in 6 months trying to write, test, release and continuously update their own apps. But this isn't exactly right.

With enough people tinkering with the definition, Wikipedia defines DevOps as:

"DevOps (a portmanteau of development and operations) is a software development method that stresses communication, collaboration and integration between software developers and information technology (IT) professionals. DevOps is a response to the interdependence of software development and IT operations. It aims to help an organization rapidly produce software products and services."

Ok, but what gave rise to it, and how does it differ from traditional IT Ops?

In an interview with InfoQ, the "father of DevOps" Patrick Debois remarked that the concept had originally been coined "Agile System Administration", but the term was too clunky and narrowly focused, so "DevOps" was chosen instead. But generally, DevOps searches to solve issues related to the following conditions:

- **Traditionally siloed team structures don't scale**
- **Developers and Operations have opposing philosophies**
- **Technology continues to evolve and the same old processes cannot handle newer paradigms**

Put briefly: Traditional IT Ops, by comparison, often live inside of a walled garden, with little support for communication and collaboration between between Ops, QA and Devs in order to make releases.

TRADITIONALLY SILOED TEAM STRUCTURES DON'T SCALE

Today's business and casual users increasingly rely on software and expect it to meet their constantly evolving needs 24/7, whether they're at their desks or mobile. As a result, software teams need to respond to change and release updates quickly and efficiently without compromising on quality. Fail to do so, and they risk driving users to competitors or other alternatives.

However, releasing apps quickly comes with its own drawbacks. It strains functionally siloed teams--Dev, QA and Ops--often resulting in software defects, delays and stress. One of the reasons for this could be related to the different philosophies among Dev and Ops teams:

DEVS ♥ CHANGE	OPS ♥ STABILITY
Development pros are incentivized to improve the product and push for change in order to better meet user needs. They have more than a decade of practice with Agile, collaboration and continuous integration tools. All of these help devs produce new releases at a blazingly fast rate.	Operations pros don't favor change as readily. They care about making sure that everything stays up and running without any hiccups that could disrupt users. Stability and reliability is most important. Change can bring improvement, but it can also blow up production environments and cause service downtime.

The lack of regular communication between the two groups further exacerbates the issue and can lead to a snowball effect of finger-pointing and bad vibes.

TECHNOLOGY EVOLVES, AND SO MUST YOU

A couple decades ago, if you had 20 servers you had a lot! We've come a long way since then. The huge proliferation of web-based apps and the magic of the cloud, virtualization and hosted environments has spurred environments with hundreds, thousands and sometimes even more servers.

If you're setting up or troubleshooting environments that large, manual maintenance won't get you very far. You have to automate your infrastructure using code to spin up servers, configure and monitor them, and react based on events that can take your service down.

So you could be in for a big surprise if you think your existing resources--whether human, technological or financial--will automatically grow along with major sea changes in the technology landscape.

ENTER DEVOPS

Let's make one thing clear. DevOps is not NoOps. Nor is it akin to putting a Dev in Ops clothing. DevOps should be synergistic, rather than cannibalistic, so it's important that both Dev and Ops team bring a unique set of skills and experience to software development and delivery. DevOps is simply a culture that brings development and operations teams together so that through understanding each others' perspectives and concerns, they can build and deliver resilient software products that are production ready, in a timely manner.

Together, Dev and Ops can focus on what's most important - the end user experience. They produce high quality software with high availability and a full understanding of each other's limitations that can be addressed well ahead of time. As a result, much of what is considered "operations" is written and packaged as code both in and out of the app. This brings forth smarter apps and environments that can adapt to changes in user requirements faster.

So the next time you see a job description titled "DevOps Engineer", understand that the recruiter is looking for someone who has operations expertise, is proficient with automation, and is comfortable working closely with developers in a workplace that promotes a shared culture. Otherwise, they maybe just saw that job title on the interwebs and picked it because it sounds good.

DEVOPS IS HERE TO STAY

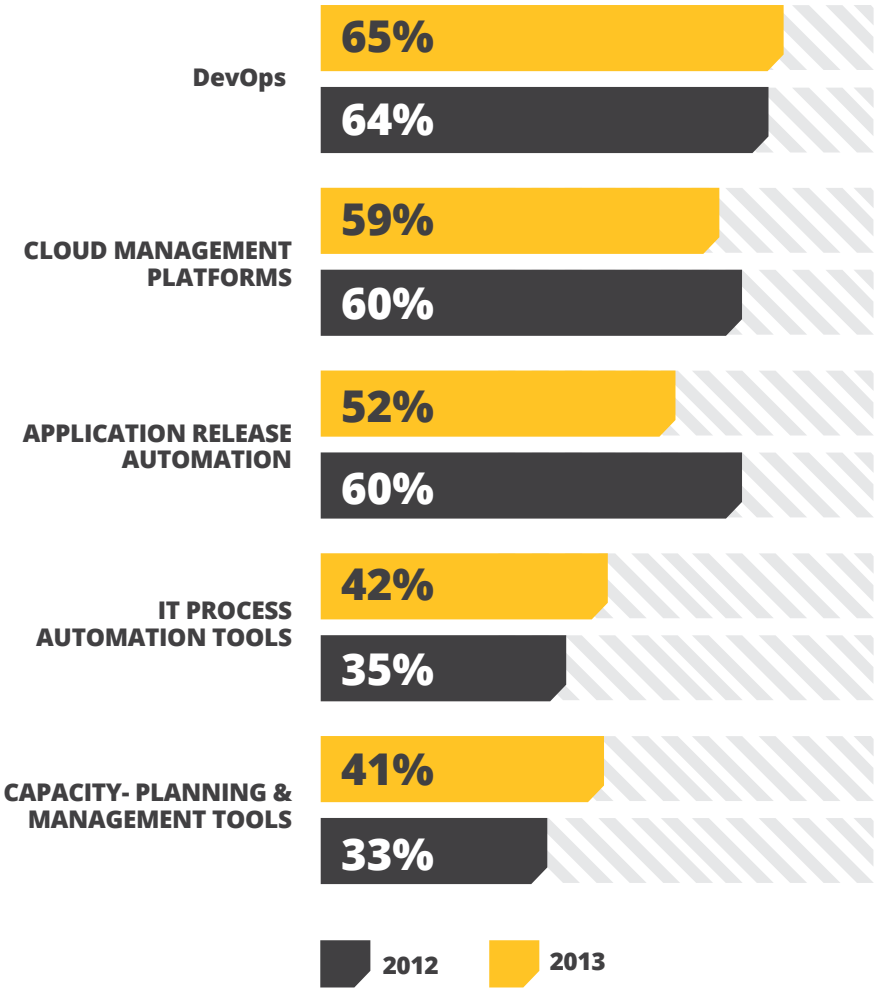
We asked those who took our survey to tell us what key initiatives their organizations had planned for in 2012 and 2013. Here's what the poll came to. Over the two years, "DevOps" related initiatives have come up on top. Surprised? Don't be. There is a strong need for agility to respond to ever changing and expanding market needs. Software teams are under pressure to help meet them.

A total of 64% of respondents said that DevOps was a key initiative within their organization in 2012. This figure stepped up to 65% in 2013.

The second and third places on the poll are held by "Application Release Automation" and "Cloud Management Platforms". Both help expedite application delivery and require some level of collaboration between development and operations teams to get right.

Stepping back a little, you see that the focus in 2012 and 2013 was on productivity through collaboration, system and process consolidation, and better management. Software teams are priming themselves to become lean, mean machines that develop awesome products to address market opportunities and placate users constantly hungry for the latest and greatest feature.

KEY INITIATIVES UNDER DISCUSSION



THE IT OPS & DEVOPS PRODUCTIVITY REPORT

We surveyed 620 engineers to discover what they do to keep everything running like clockwork – by examining their day-to-day activities, key processes, tools, and challenges they face. To give a little extra context, we then studied their responses side by side to understand what distinguishes DevOps and traditionally siloed approaches from each other.

Although the number of responses should be statistically significant for the conclusions we draw, there is always a possibility of bias, as the respondents were self-selected from the audience we could reach through various channels. In particular, there is a bias towards the Java platform, as a lot of our readers and contacts use it, in addition to the DevOps community over traditional Ops, as that community is more active and centralized. We note the results that can be affected by this bias and will interpret them accordingly.

We can further break down the structure of this report into 4 sections:

Part I - The Week-Long Grind: From DevOps and Traditional IT Ops Perspectives

How much time, on average, do our respondents spend doing key tasks related to operations each week? How does the traditionally siloed IT Operations and DevOps cultures differ? How do you adopt one over the other? Choose an approach that works for your organization.

Part II - App Failures and Stellar Recoveries

What causes application failures and how often do they occur? How are teams alerted and do issues get fixed quickly? How are production releases affected by human error or by apps not “built for production”?

Part III - Tools of Choice: Popular Productivity Boosters

Looking to adopt new tools? See which tools are most popular to help achieve daily tasks and responsibilities from an operations perspective.

Part IV - Releasing Software: Control, or be Controlled

Take a closer look at how the releases are carried out. How long do they take on average? When are they typically carried out? Where do you stand?

The findings in this report are only as good as the input we receive from all our survey takers. On that note, we’d like to thank them for their time and get into the good stuff!

PART I

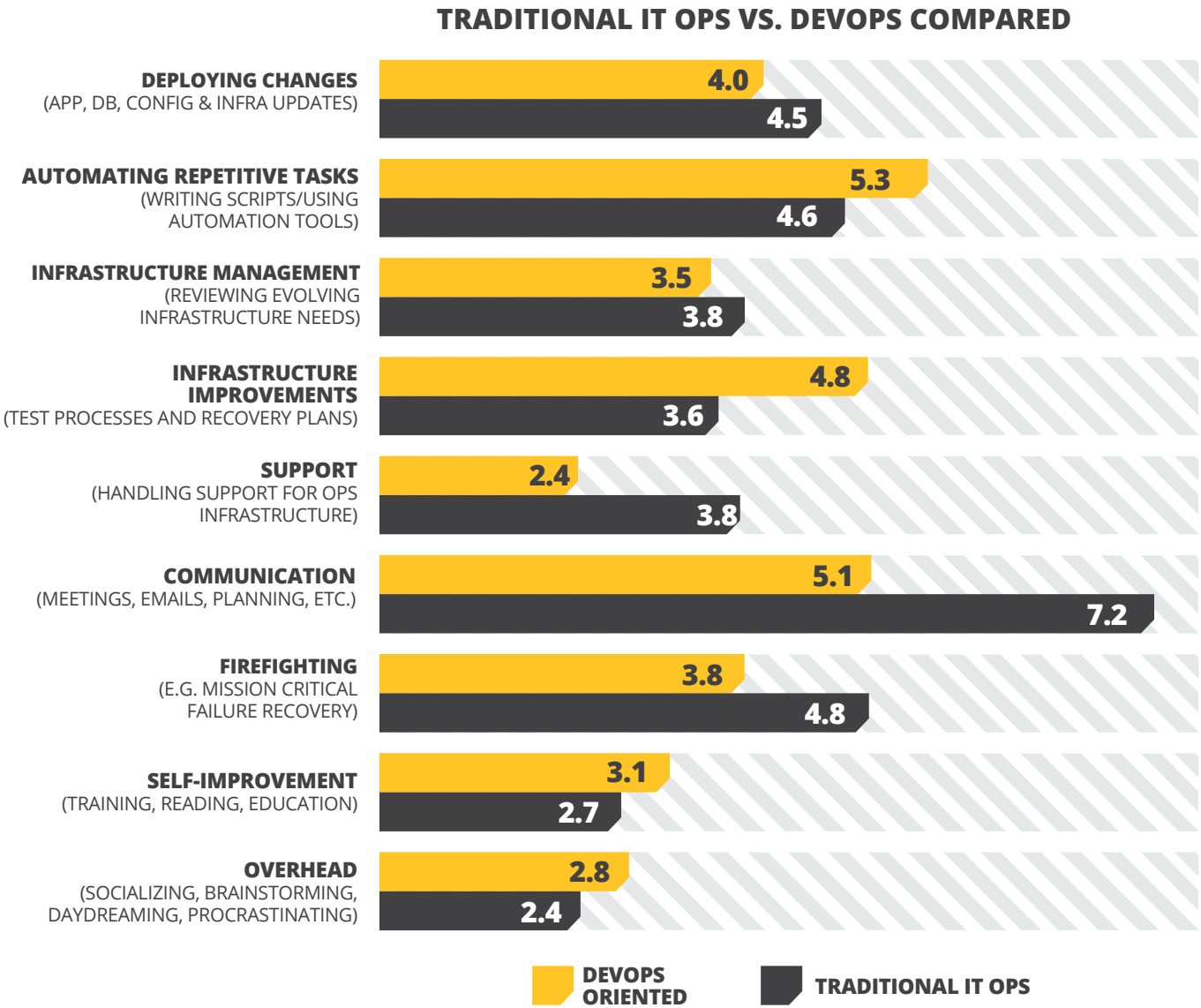
THE WEEK-LONG GRIND: FROM DEVOPS & TRADITIONAL IT OPS PERSPECTIVES

On a weekly basis, DevOps oriented teams spend more time doing positive or neutral tasks (i.e. automating repetitive tasks, infrastructure improvements & self education), while traditional IT Ops teams spend more time in negative or neutral tasks (i.e. firefighting, support, communication).

Hours Spent on Day-to-day Activities: Traditional IT Ops vs. DevOps

Lets take a close look into the daily lives of folks involved in operations. What keeps them busy during a typical week? How much time do they spend doing productive activities, like improving infrastructure and setting up automation for repetitive tasks? How much time to they end up spending fighting fires and communicating in the organization? What does it take to keep the lights on?

While we're asking about the day-to-day activities that teams releasing their apps typically complete, we also thought it would be cool to compare responses between traditional Ops and DevOps teams; i.e., those belonging to those from IT Operations belonging to traditionally more isolated teams vs. those belonging to DevOps-oriented teams. Let's see what's different between the two approaches.



Let's go over the categories of daytime tasks and activities we asked about in order to get a better understanding of their relevance.

DEPLOYING CHANGES

This includes updates to apps, databases, configuration and infrastructure. Of respondents from IT Operations belonging to traditionally siloed teams, 25% spent 8 hours or more a week deploying changes. This is 5% more than those from DevOps oriented teams.

AUTOMATING REPETITIVE TASKS

Tasks along the application release process can be repetitive and time consuming. Writing scripts or using tools to automate tasks increases efficiency and consistency, reduces human error and take software to users quickly. Around 32% of respondents from DevOps oriented teams spent 8 hours or more a week automating tasks, as opposed to 24% of those from traditionally siloed teams.

INFRASTRUCTURE MANAGEMENT

The last decade has seen a lot of advancement in how we acquire and manage infrastructure to increase productivity and manageability, and reduce cost and overhead. With virtual, cloud and elastic environments, we've abandoned server rooms for virtual ones where we never see the infrastructure we deploy on. In a DevOps oriented workplace, as apps evolve, infrastructure evolves with it to address new capabilities, fix bugs and make the apps more resilient.

INFRASTRUCTURE IMPROVEMENT

As apps get more complex, operations teams have to not only deploy apps, but also manage the environments they run on. This includes monitoring apps, servers, network infrastructure and any third party services, and continuing to innovate. They need to regularly test and set recovery plans in place in case of application failures. Around 28% of respondents from DevOps oriented teams reported spending 8 hours or more improving infrastructure compared to 19% of those from IT Operations belonging to traditionally siloed teams.

ADMINISTRATION SUPPORT

Where there is infrastructure, there is support associated with accounts, services and procurement. Only 11% of respondents who classified themselves as belonging to DevOps oriented workplaces spent 8 hours or more on support per week. Just 32% spent 30 min or less a week. On average, respondents from IT Operations belonging to traditionally siloed teams spent 1.4 hours more per week on support cases than their peers in DevOps oriented teams.

COMMUNICATION

This includes time spent in meetings, writing emails, planning and so forth. More respondents from IT Operations belonging to traditionally siloed teams reported spending 8 hours or more a week just communicating. 18% of them reported 16 hours or more doing so! Could this be because they need to spend more time do so in siloed workplaces? Does DevOps make communication more efficient by fostering an environment of collaboration?

FIREFIGHTING

When an app failure occurs in production, software teams must scramble to bring service back up. Downtime impacts revenue and brand image. When driven by a DevOps culture, software teams collaborate to build apps that are resilient and less likely to fail in production. On average, DevOps oriented team members spent an hour less a week firefighting. Only 19% of them fought fires 8 hours or more a week compared to 37% of those from IT Operations belonging to traditionally siloed teams.

SELF IMPROVEMENT

This refers to training, reading, and education through various media to acquire new knowledge or skills. DevOps oriented team members spent more time than their peers in traditionally siloed IT Operations teams on self improvement.

OVERHEAD

This category includes time spent on social networks, procrastinating, leisure and so on. Respondents from DevOps oriented teams gain time from efficiency and productivity, and spend more time indulging in these activities!

Conclusions we can draw from the results

DevOps adherents spend more time on task automation and infrastructure improvement
DevOps teams spend more time writing scripts, automating infrastructure and improving testing, logging and monitoring capabilities. As a result, teams spend fewer hours deploying apps (even if they do so frequently) and managing infrastructure.

DevOps requires less time for communication
Respondents belonging to a DevOps oriented team spend 2 fewer hours communicating each week. This could be because DevOps fosters better collaboration and keeps Dev and Ops teams in sync with each other, meaning less frequent meetings to "get everyone on board". Shared tooling, like group chats, task managers and social tools might also help bring everyone closer together.

DevOps don't need to fight fires as much
A key tenet of the DevOps methodology is embracing failure. This means understanding the fact that failure will happen in production and the important part is quick reaction and recovery. Using continuous testing and monitoring with both metric charts and alerts enables reacting as soon as the failure is happening. Programmable infrastructure and automated deployment provides a quick recovery from most scenarios with less user impact. Of course, a tighter relationship with the Development gets things fixed quickly on that side as well.

DevOps spend less time on administrative support
This could be a result of both better communication, higher level of automation and the availability of self-service tools/scripts for most of the support tasks. If provisioning and automation is high here, then there is no reason why admin support shouldn't dwindle down to a very small time drain.

DevOps leads to less claims on work hours each week
The data indicates that those that operate in DevOps environments allocate 4 hours less in a week than their counterparts. Interpreting this is left as an exercise to the reader, but we'd like to think it's related to better communication, automation and more "production-ready" deliverables. :)

DevOps leads to fewer days with work after-hours
We asked our survey takers to tell us how many times a week they work outside of normal business hours (note: this may mean 9AM to 5PM in some organizations, but respondents were asked how often they work outside of their regular work hours, which may be, for example, 11 AM - 7 PM). Here's what they told us:

DAYS WORKED AFTER HOURS	TRADITIONAL IT OPS	DEVOPS ORIENTED
Average	2.3	1.5
Mode	2	0
Standard Deviation	1.7	1.7

According to these results, members of teams that adopt a DevOps culture lead to a more balanced life, spend more time on automation and infrastructure improvement, spend less time fighting fires, and allegedly work less hours (especially outside of normal business hours).
DevOps oriented teams also seem to spend more time doing positive or neutral tasks (i.e. automating repetitive tasks, infrastructure improvements, self education) while traditional IT Ops teams hours are consumed more by negative or neutral tasks (i.e. firefighting, support, communication).

Take what you like from that, but it sounds like the job profile of a DevOps-minded engineer could easily be in high demand!

PART II
APP FAILURES AND STELLAR RECOVERIES

The average team experiences 2 app failures each month, and it takes at least half an hour to recover from a failure for over 50% of respondents. What does this cost you?

Nobody likes application failures in production. They disrupt business processes, impact revenues and frustrate customers. The web is full of instances where app failures have cost companies millions of dollars and impacted users. Just search for "software failures."

Even though production failures often fall into the category of Operations, other departments, like Project Management, Development, QA and other business stakeholders cannot help but be affected when this happens. In this section, we get a better understanding of:

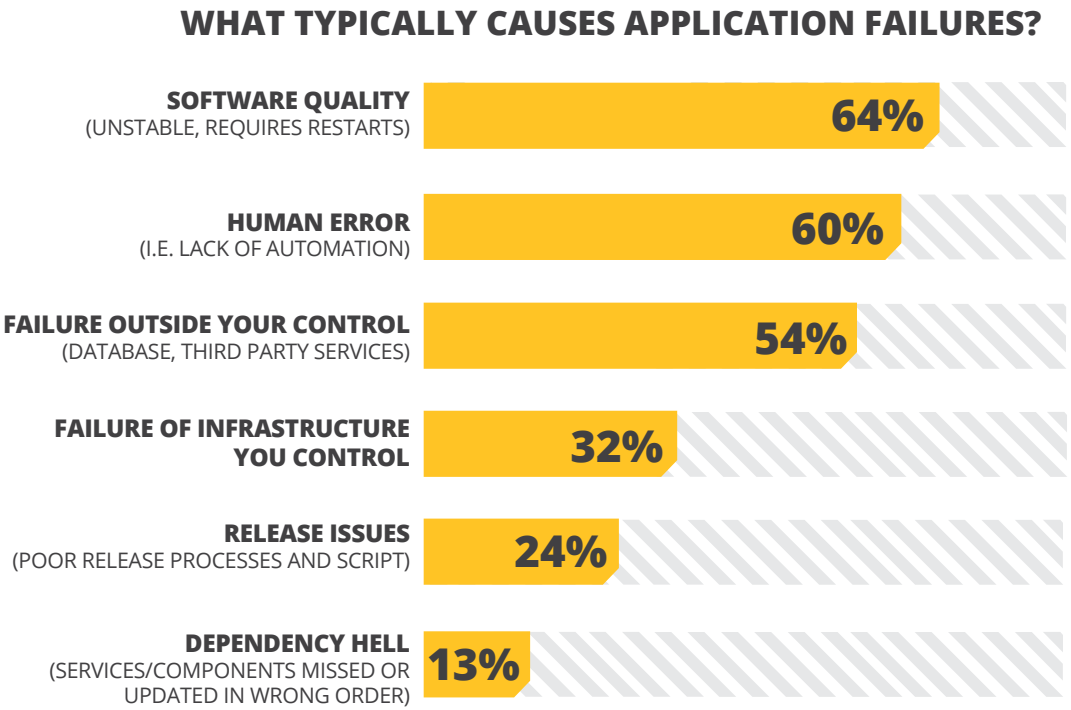
- What are the typical causes of app failures?
- How do teams get alerted to failures?
- How do teams respond to these failures?
- How often are recoveries from failures needed?
- How long do recoveries take?
- Are recovery processes typically tested?

Quick shameless plug 1 of 2: ZeroTurnaround's own LiveRebel automates whole app releases - code, database and conf - all in-sync, onto physical, virtual or cloud environments quickly, safely and consistently with no downtime or overhead. Releases with LiveRebel are versioned, automated, fully reversible and testable. It helps achieve the kind of automation DevOps calls for. Shameless plug plugged.

So let's get started with identifying the causes of application failures.

What typically causes application failures?

According to the results in the chart below, we can see that software quality and human error came up on top with 64% and 60% of the respondents stating that they caused application failures in production. This alone makes a strong case for DevOps adoption by helping development teams design more "production-ready" code and automating repetitive processes rather than doing them manually.



Without squeezing the grey matter too hard, you can derive a couple of conclusions based on these findings:

1. APPS NEED TO BE BUILT FOR PRODUCTION

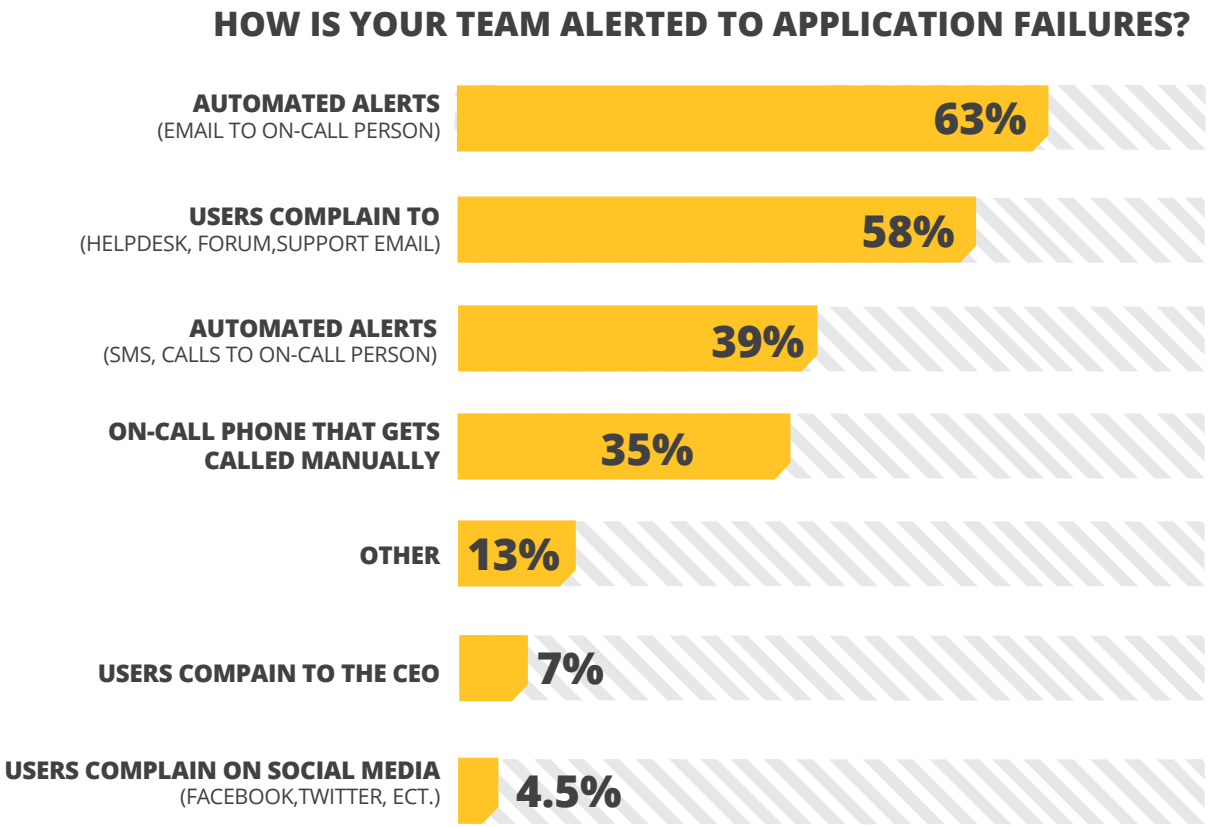
Traditionally the development team builds apps, passes it to the QA team for testing, which then hands them to the operations team for deployment. There is minimal dialogue between silos as a result of which, apps are not built for production environments. Consequently, bugs arise and they fail to perform as expected in production. DevOps fosters close collaboration between the dev and operations teams so that the sev team can build apps for production environments that are resilient and aware of their deployment environment.

2. YOU CAN ELIMINATE HUMAN ERROR WITH RELEASE AND PROVISIONING AUTOMATION

Humans are not known for their great ability to precisely repeat tasks day-after-day. Even in the the world of atoms, boring or menial tasks are getting increasingly automated as gains in quality, consistency and speed more than offset the increased cost. It is even more so in the world of bits and bytes--there is no sense in having a human do something that a computer can do better. It's pretty much that simple.

How are teams alerted to failures?

Software teams, particularly in operations, need to be alerted right after a failure occurs so that they can act quickly and restore service.



The good news is that the most popular alert method (63%) is automated messages to an on-call person who then rallies up the team to resolve the issue that caused the failure. The bad news is that the second most common notification is end user complaints! At this stage it is already too late. The app is losing revenue, disrupting current users and causing a painful ripple effect that can reach future users. Ideally, teams should be able to tackle issues proactively before users are impacted.

How often are recoveries from failure needed?

More often than we'd like! When an app fails in production, bringing the services back up is crucial. Approaches include rolling back to the previous stable version or finding some intermediate solution so that business can return to normal. Once done, the team can then go on to identify the root cause and devise a suitable fix.

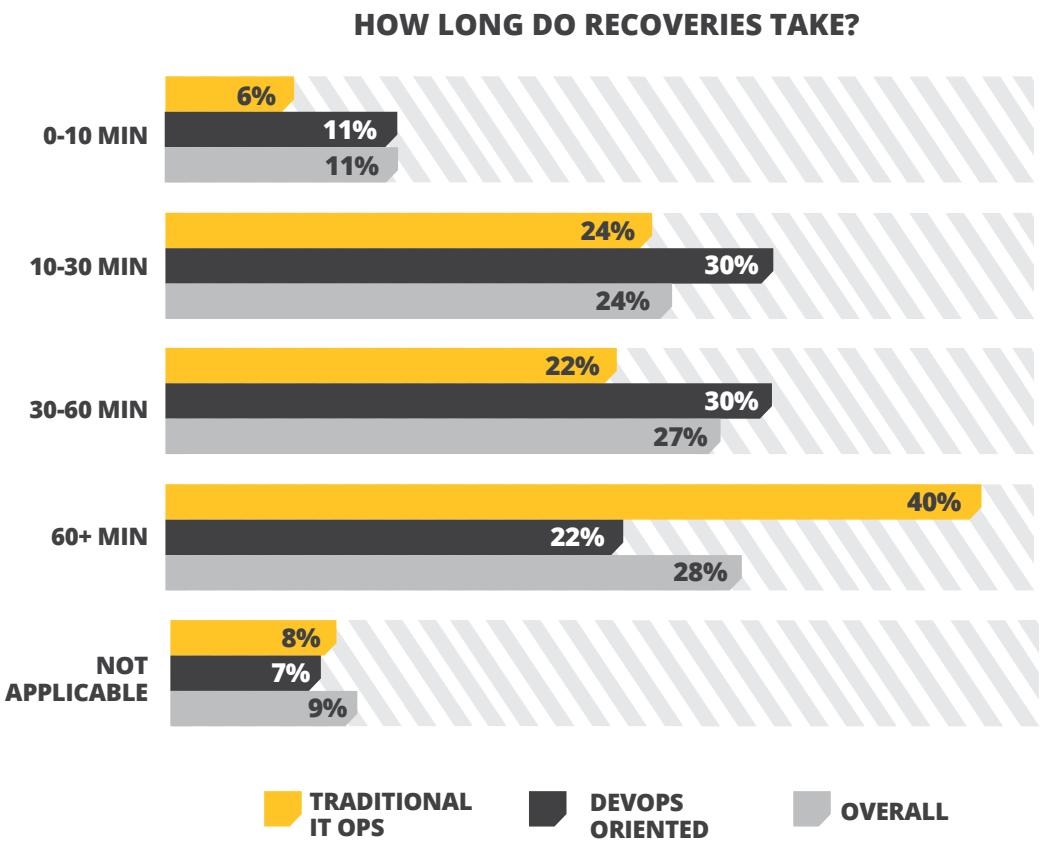
Here is how many times respondents to our survey recover production environments each month.

	NUMBER OF RECOVERIES A MONTH
Average	2.13
Median	1
Mode	1
Standard Deviation	3.7

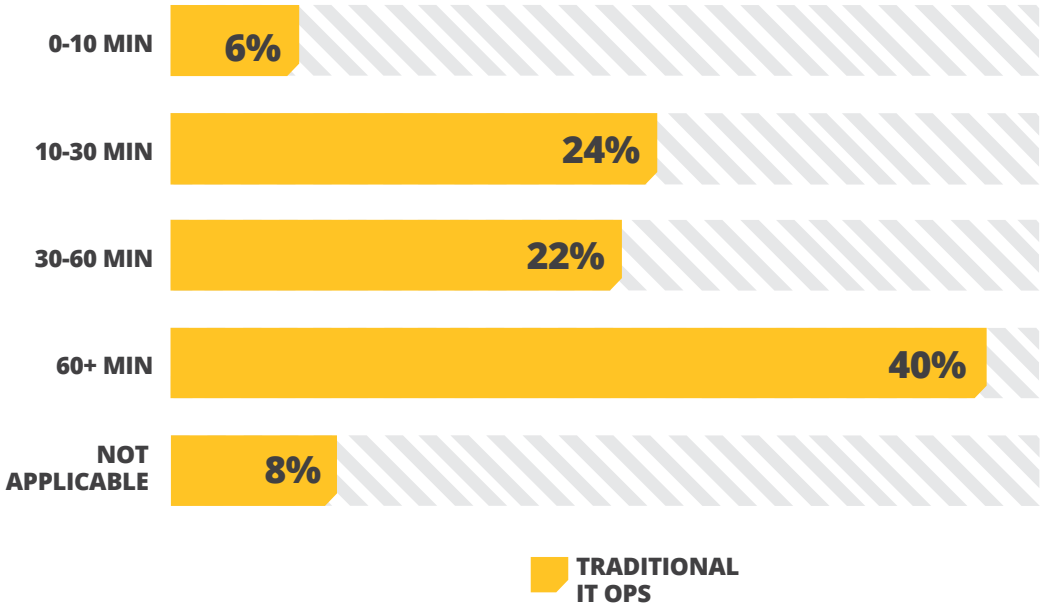
How long do recoveries take?

Recovering from a production failure is no fun. It typically involves getting key people from the development, operations and business teams out of bed and working after-hours in a war room with the sole focus to restore service. Stress is very high, and there is a tendency to blame everyone else in these situations.

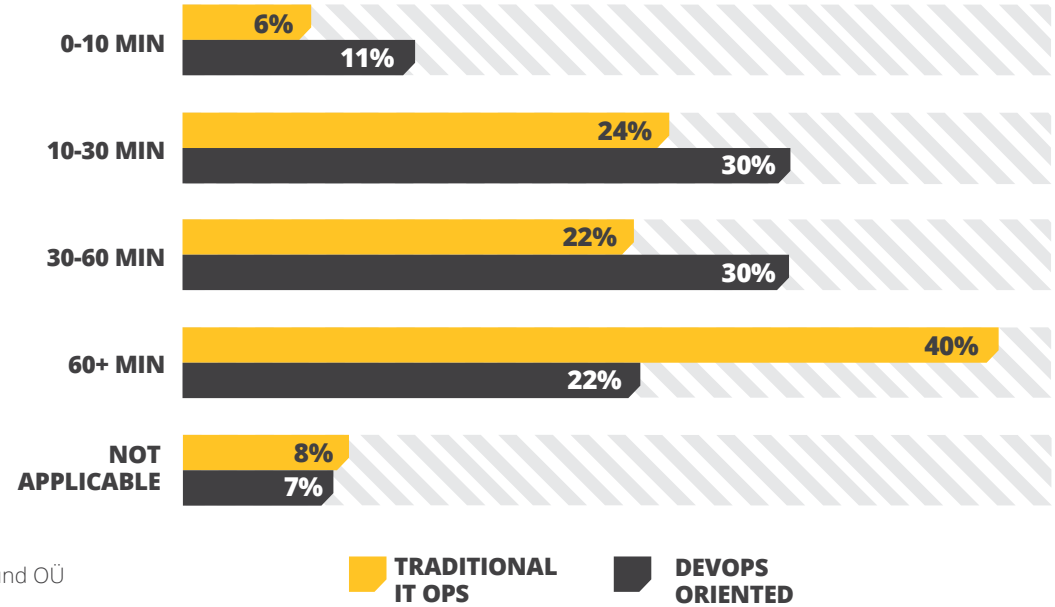
Recovery times are too long for both groups: based on the responses we received, 28% of respondents take more than an hour on average! This means that service is crippled in the meantime. Only 11% of our respondents, on average, restore service in under 10 minutes.



HOW LONG DO RECOVERIES TAKE?



HOW LONG DO RECOVERIES TAKE?

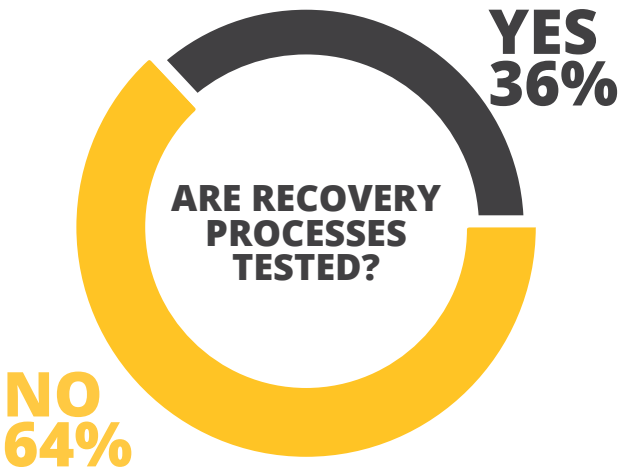


TRADITIONAL IT TEAMS	DEVOPS ORIENTED TEAMS
Close to 40% of respondents from this camp said that they take 60 minutes or longer to restore service, and a majority need a minimum of 30 minutes to restore service. This means long war rooms, lots of pizza and more than lots of unhappy users.	Close to 30% of the respondents said that they take 10 to 30 minutes to restore service. A little over 22% take 60 minutes or more. Interestingly, DevOps oriented teams are almost twice as likely to recover in 10 minutes or less than traditionally siloed teams.

So there you have it . If we split the teams between traditionally siloed Ops and DevOps, we find that 60+ minute recoveries are nearly twice as likely for traditionally siloed teams. Clearly, a DevOps oriented culture fosters an environment that helps recover service more quickly.

Are recovery processes tested?

By now, it should be evident how important recovery processes are in the event of application failure. However, only 35.8% of respondents said that their recovery processes are tested to ensure that they work when needed.



With suitable release automation tools, recoveries can be effortless and fast. Production systems should be able to automatically reset to a previous working state so that service is restored before users and revenue are impacted. This will then allow the software team to focus on identifying the root cause and fixing the problem without added stress.

PART III
TOOLS OF CHOICE: POPULAR PRODUCTIVITY BOOSTERS

Java applications, Puppet & Chef, Selenium, Nagios, Shell scripts and vi/vim dominate the tool sets, yet automated smoke testing and infrastructure as code remain little used.

Boosting productivity is a topic that's right in people's faces right now. For a while now in software development, any downtime due to bottlenecks in the process, like environment activation, compilation, deployment, and publishing times (hint: Exo IDE) or long app server restarts (hint: JRebel), are under fire.

Today's rapidly evolving economy means that expanding markets and new competition put pressure on businesses to perform and deliver exceptional results. Some of this pressure is directly transferred on to software teams that then look to improve productivity to deliver technology and innovation more rapidly to support business goals. This is achieved with tools, methods and concepts that increase productivity and reduce waste.

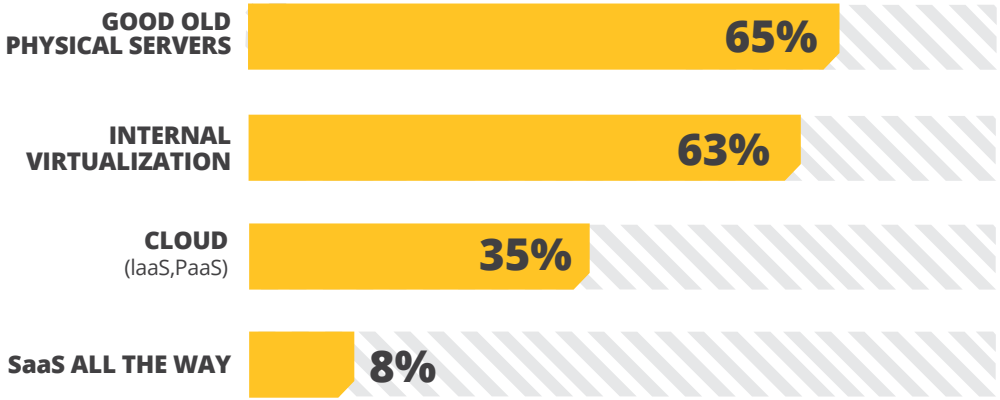
This trend has partly contributed to the DevOps movement and has led to software teams adopting new tools and processes more rapidly that before, to help them work smart while they work hard. Lets go over some of the popular choices based on our survey responses.

Note: Naturally, there are many tools available in the industry, and it's impossible to cover all of them. We selected tools that we have some experience using, or discussing, in addition to tools proffered by respondents. So in the interest of starting somewhere, we selected the tools you see below. Feel free to suggest any that we're missing!

INFRASTRUCTURE

This is the bedrock upon which we deploy our apps and services that we make available to the world. If the bedrock crumbles, so does the app. Infrastructure can typically be provisioned on-premises, in the public cloud provided by the likes of Amazon and Rackspace, or across both.

WHAT KINDS OF INFRASTRUCTURE DO YOU USE?

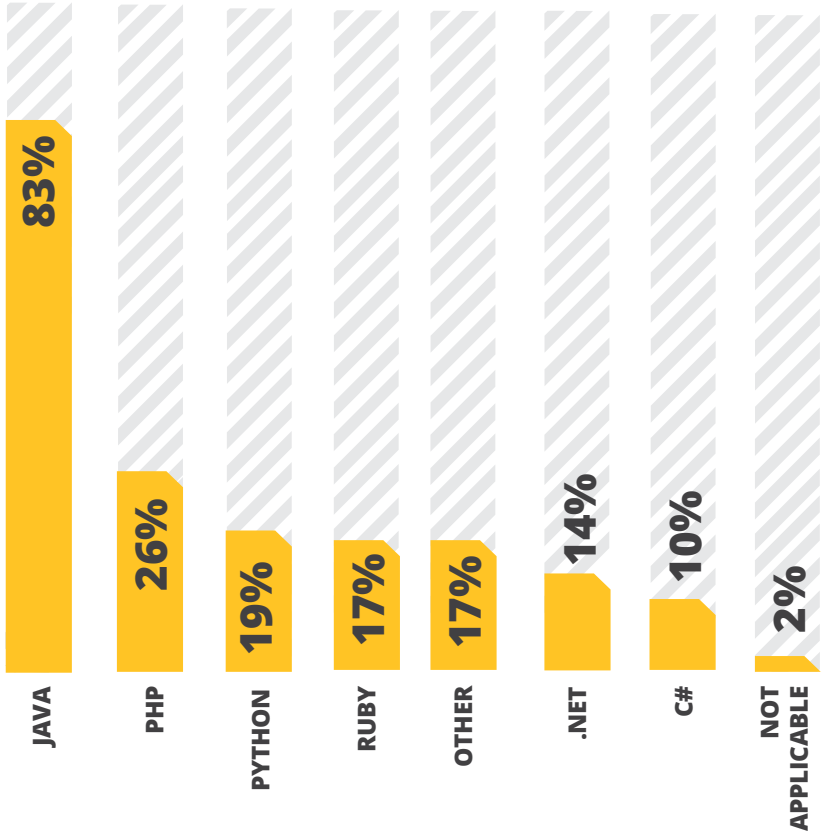


The last decade has seen a lot of innovation within the infrastructure space, taking it from physical servers to virtualization, cloud and hybrid environments that can intelligently scale up and down based on capacity.

TYPE OF APPS

We asked respondents to mark the platforms that they are managing. Java is ahead, but this is mostly a function of the audience we could reach, not the actual difference in popularity. We believe that PHP and Python are actually more popular than shown here.

WHAT KINDS OF APPLICATION DO YOU MANAGE?

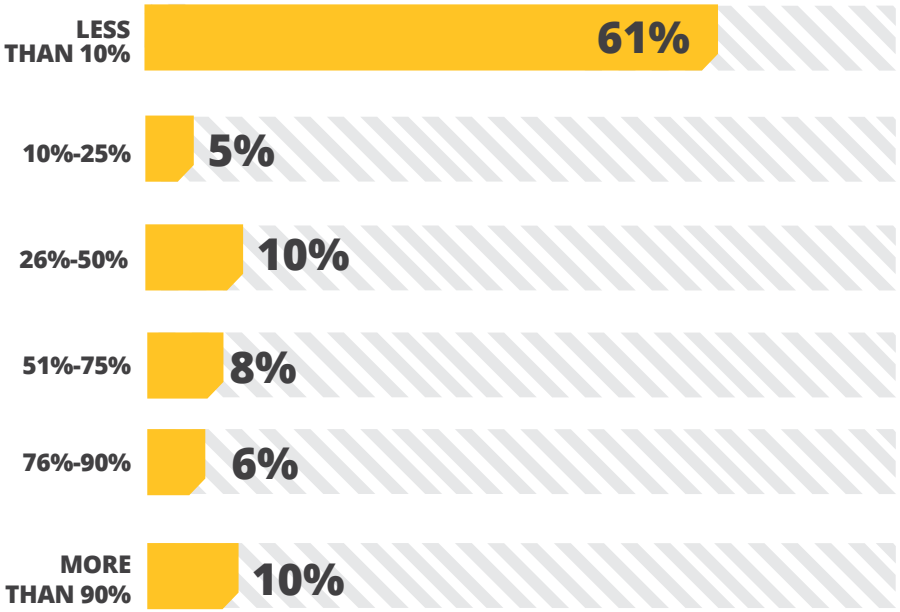


INFRASTRUCTURE AS CODE

Any organization that is serious about adopting DevOps as a culture looks to control infrastructure programmatically by treating it as code. There are clear benefits to this: your infrastructure is updated and checked in with each new release, and this frees up time from carrying out fairly repetitive and occasionally error-prone (=manual) configuration tasks. Plus, it increases the agility of your team as a whole through automation.

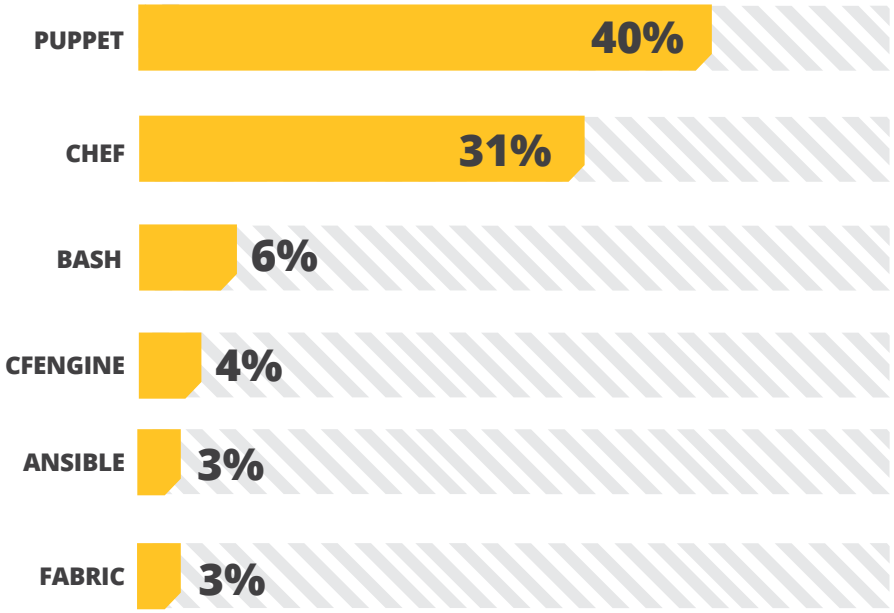
Based on our survey, 61% of respondents have less than 10% of their infrastructure as code. Only 10% of them have more than 90% of their infrastructure as code, and those are the early adopters that manage their whole infrastructure with provisioning tools. We did not offer a 0% option for this survey.

HOW MUCH OF YOUR INFRASTRUCTURE DO YOU CONFIGURE WITH CODE?



We also asked about infrastructure configuration tools, and ranked them in terms of popularity. Puppet and Chef are clearly the tools of choice when it comes to provisioning infrastructure as code, and there is a minority segment being battled over by Bash, CFEngine, Ansible and Fabric.

POPULAR INFRASTRUCTURE CONFIGURATION TOOLS

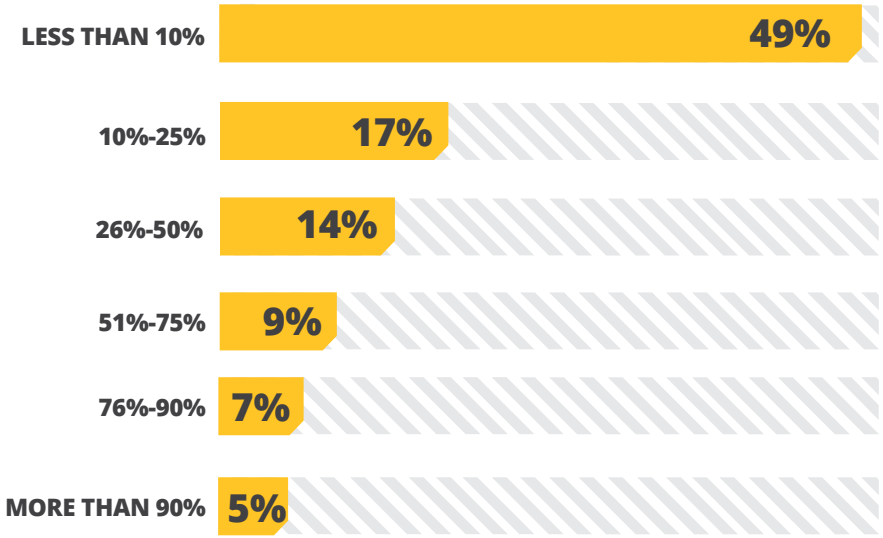


This isn't anything to rush in to, and managing all your infrastructure as code is not something that is achievable overnight. It is a gradual process where you work towards fully automating the process of configuring your infrastructure.

AUTOMATED SMOKE TESTS

When you set up automated smoke tests, the idea is that they will play back predefined actions and compare results with behavior that you would expect from the app. They can easily be repeated and extended as we add more features to the app. It makes sense to automate these tests so that you can eliminate manual, repetitive and time-consuming tasks.

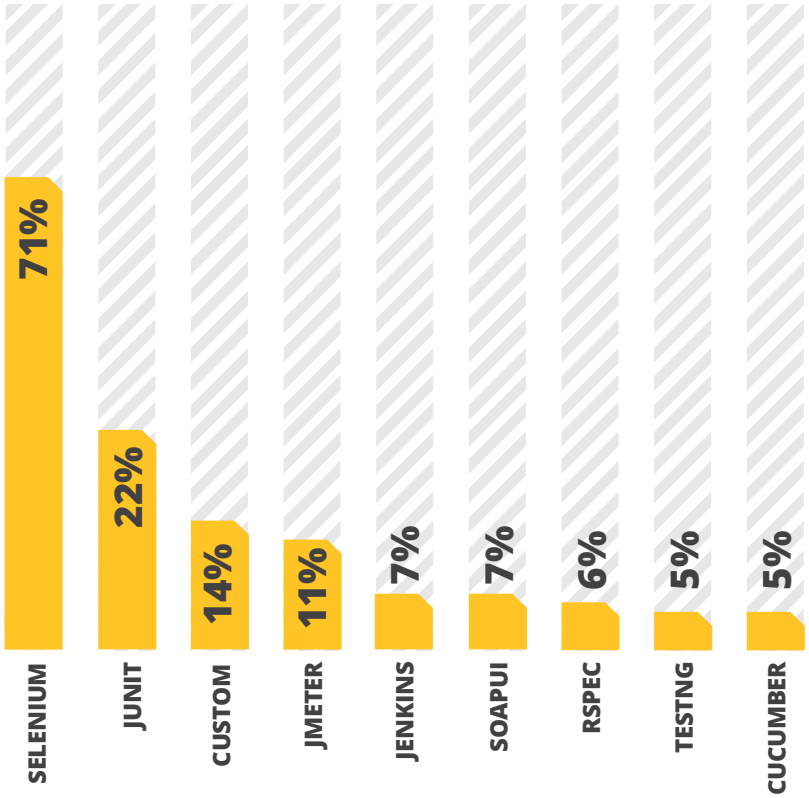
HOW MUCH OF YOUR SMOKE TESTING IS AUTOMATED?



Nearly 50% of respondents have less than 10% of their smoke tests automated, and only 5% have more than 90% of their tests automated.

When we asked about the most popular test automation tools amongst respondents, Selenium has the widest adoption followed by JUnit and custom test tools that are built inhouse. Jenkins, as a popular Continuous Integration server, and TestNG, which expands the realm of unit testing from JUnit's capabilities, are also in the list.

POPULAR TEST AUTOMATION TOOLS



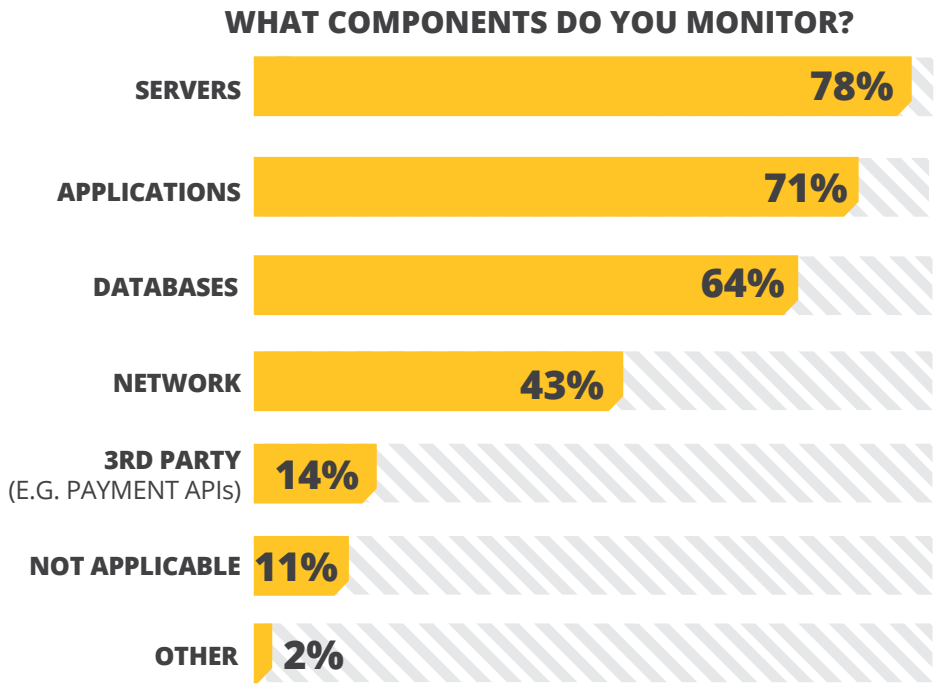
It's surprising that so little smoke testing is automated, considering that the tooling have been around for a long while. At ZeroTurnaround, we use Selenium to regularly and automatically test online registration, download and purchase processes.

PERFORMANCE MONITORING

Today's complex systems have grown exponentially, and it's almost impossible to monitor all components for slowdowns or errors, including databases, application or web servers, third party services and ISP's all the time.

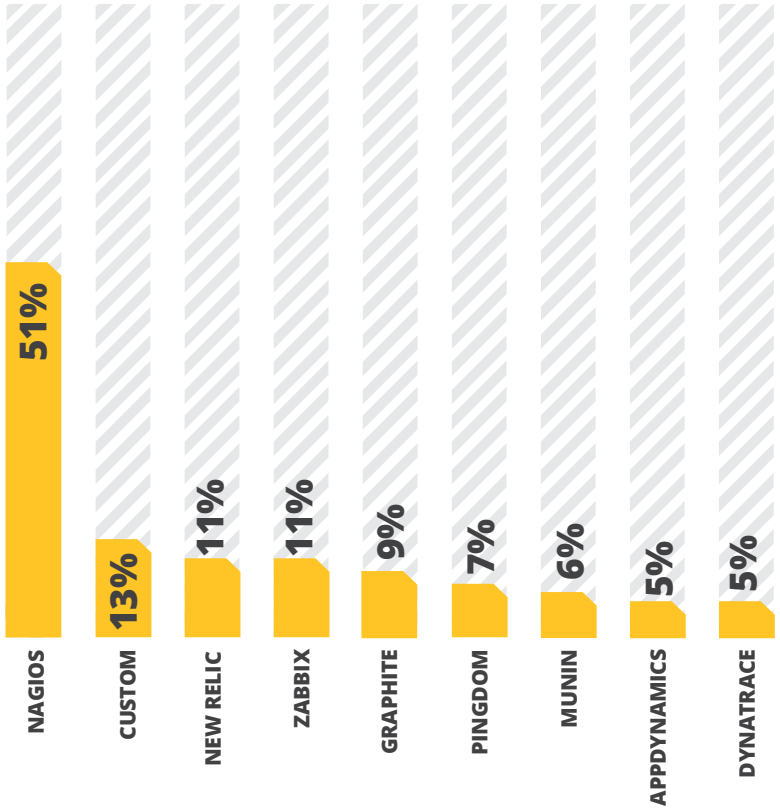
Performance monitoring is critical, mainly because you will be the one taking the blame if something were to slow down or fail, even if it is beyond your control. It's purpose is to identify issues before users are impacted to improve service, better understand needs and app performance, and lower costs.

Based on our survey responses, here are the components that are monitored the most. Server, application and database monitoring lead the pack.



We also asked our survey takers to tell us which monitoring tools they use. Here are the most popular ones. Quite often, DevOps oriented teams try to release software quickly and frequently, so they want to pay special attention to performance monitoring and ensure that changes do not slow down or disrupt service.

POPULAR MONITORING TOOLS



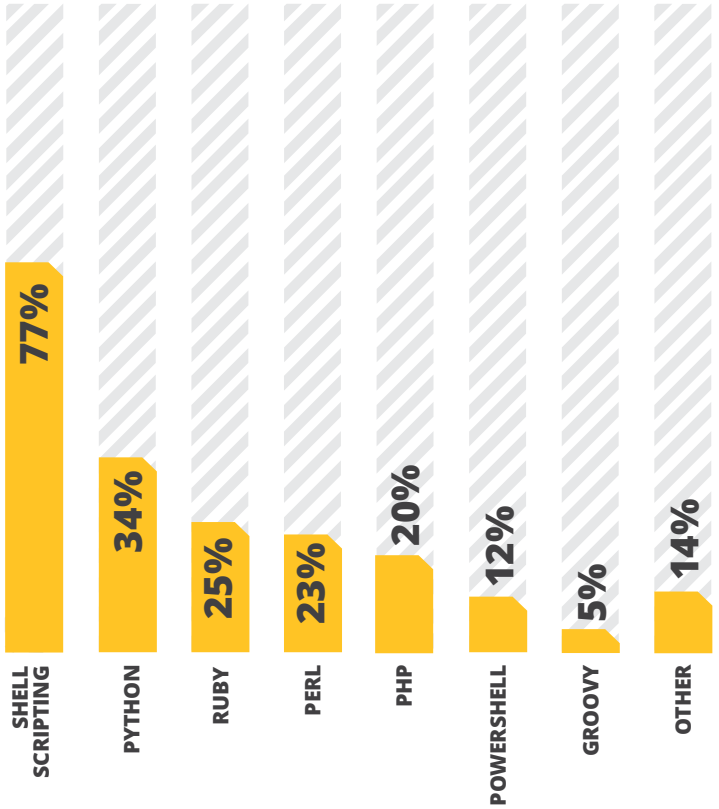
Nagios, which has an open source and a commercial version, is used by just over half the respondents. It is followed by custom monitoring tools built inhouse.

With nearly two-thirds of respondents using either Nagios or custom built tools, it's interesting to note the plethora of other tools in use, provided by well-known companies like AppDynamics, dynaTrace (acquired by Compuware in 2011) and New Relic.

SCRIPTING LANGUAGES

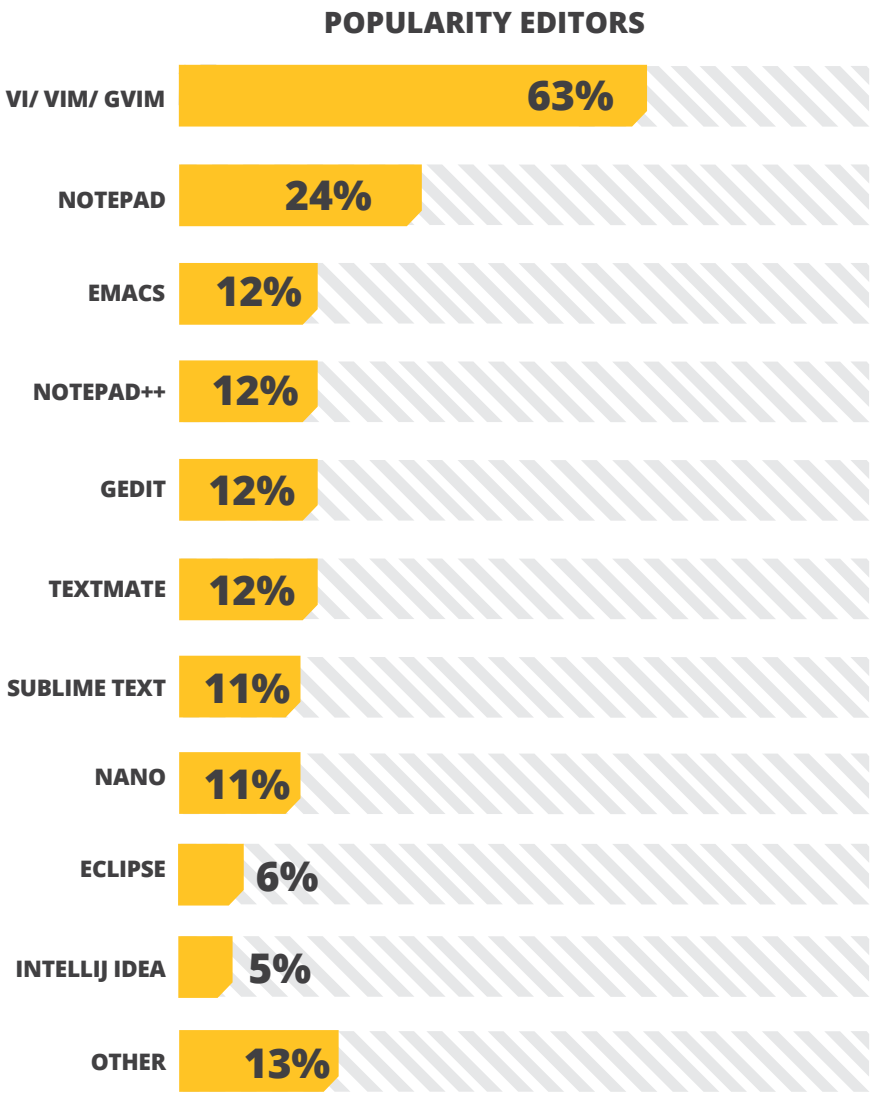
Script languages are the Swiss Army knife of any crafty engineer. Ops use them extensively. Here are the most popular scripting languages used by our survey takers. In case you were wondering, Shell scripting wins :)

POPULAR SCRIPTING LANGUAGES



EDITORS

This needs no introduction! Here are the most popular ones based on the responses we received.



Originally created as a text editor for Unix in the late 1970's, vi (and variants vim/gvim), is the most popular. Although there are many more intuitive, easy-to-use editors, vi still leads by far owing to its loyal user base and powerful capabilities.

sponsored by:

LiveRebel

ONE-CLICK APP RELEASES

FAILSAFE, ONE-CLICK RELEASES:
Versioned, automated, fully reversible and testable.

MANAGEMENT CONSOLE:
Manage and monitor apps and servers in real-time.

PLATFORM SUPPORT:
Java, PHP, Python, Ruby or Perl bundled with DB & config changes.

MAXIMUM FLEXIBILITY:
Deploy apps from the web UI, command line or CI server plugins.

TRY LIVEREBEL FOR FREE
and continue being awesome!

FOR MORE INFORMATION VISIT: WWW.LIVEREBEL.COM, EMAIL: LIVEREBEL@ZEROTURNAROUND.COM OR CALL: 1 (857) 277-1199

PART IV
RELEASING SOFTWARE: CONTROL, OR BE CONTROLLED

DevOps oriented teams can release applications in half as much time as traditional IT teams, averaging 36 minutes per app release, compared to 85 minutes for the latter.

The central theme of DevOps is to bridge the cultural silos between development and operations so that software teams can rapidly and efficiently respond to user needs without compromising on quality. This means they eventually become good at developing and delivering new releases as frequently as required. As they increase speed at which they release software, they begin to release software continuously in the form of minor updates, in effect, they are practicing Continuous Delivery.

Now let's take a look at some metrics on how effective release processes are, based on our survey responses.

How long does releasing apps take?

Deploying releases takes time away from doing what's important--improving apps and getting the next release under way. Hence it is important that release processes are quick and efficient without compromising on safety or quality.

Lets look at how long releasing apps take our survey takers from IT Operations belonging to traditional teams, and from DevOps oriented teams.

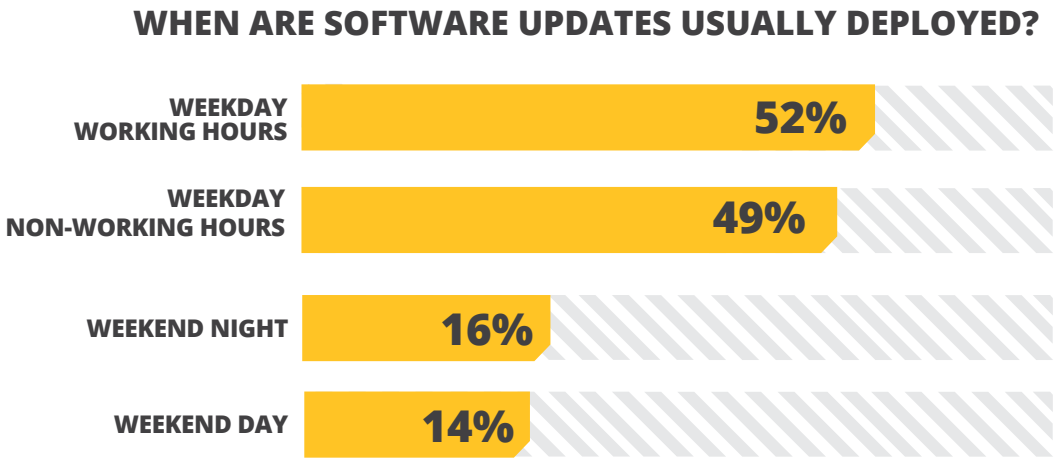
TIME IN MINUTES	TRADITIONAL IT OPS	DEVOPS ORIENTED
Average	85.1	36.3
Median	30	15
Standard Deviation	113.6	48.5

On average, DevOps oriented teams take less than half as long to release apps when compared to traditional siloed teams. Automation ftw!

When are releases deployed?

Software teams manage and plan for app deployments carefully to the impact on end users. Due to this, depending on the kind of release, deployments are sometimes carried out during off peak hours or weekends. Minor releases are often pushed out during business hours.

Let's take a look at when our survey takers conduct their release deployment processes.

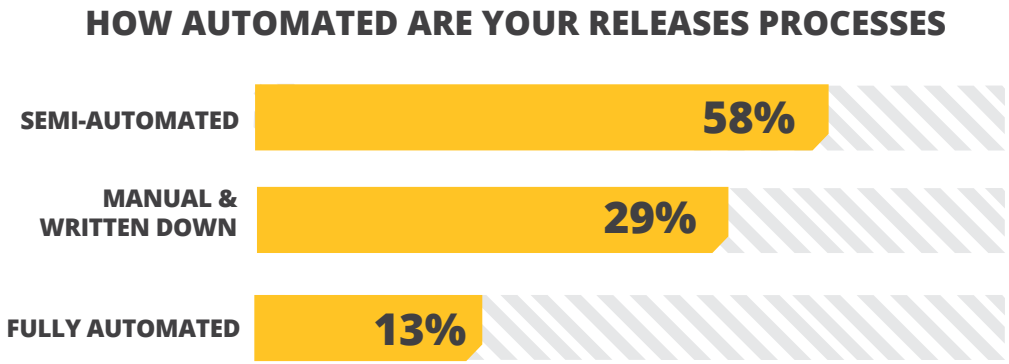


There is an even split between weekday working and nonworking hours. However, our respondents also deploy during weekends when required. This is necessary at times when a release can potentially impact users or business processes.

Ideally, if the team can conduct fast, well-managed deployments with no downtime, they can avoid having to work midnight shifts or weekends. Teams with DevOps practices will be hit hard if they don't find an automated and interruption-free method, especially if they release multiple times a week.

How automated are releases?

Automation saves time, effort and keeps processes consistent. These benefits apply to automating release processes as well. Software teams use a combination of tools and scripts in order to accomplish levels of automation they are comfortable with. Here is how our survey takers fare, when it comes to release automation (spoiler: Only 1 in 8 respondents reported a fully-automated release pipeline, which leaves 7 of you rushing to catch up!)



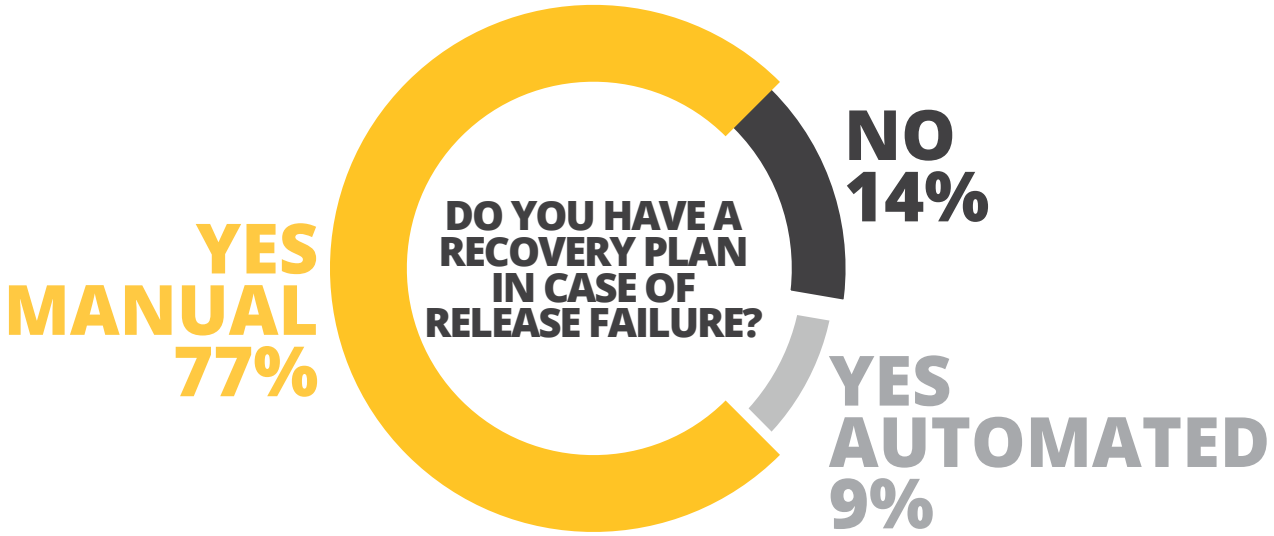
The good news is that around 60% of our respondents have at least some level of release automation, but a good 30% still release manually. With the increase in DevOps adoption, these folks will need to pay attention to some level of release automation in order to scale. This will come with tearing down the silos that Dev and Ops live in, as they learn more about each others processes, best practices and challenges.

How prepared are we for release failures?

Being able to quickly recover from a release deployment failure is perhaps more important than being able to release updates quickly. However, based on the responses to our survey, it appears as though most of them are fairly unprepared in the eventuality of a release failure.

Around 75% of our respondents rely on manual processes to recover from a release failure, which means that your mission critical fixes that must be completed with extreme urgency are also prone to human error. This could mean restoring code, rolling back databases and undo configuration changes manually. This get even harder if the deployment environment is fairly complex.

To release confidently and frequently while making sure that end users are always unaffected, robust and automated recovery processes should be set in place. They should rollback all changes quickly if a release failure is detected, before end users are impacted.



Quick shameless plug 2 of 2: Recovery automation and recovery testing are today quite a pain in the bottom area. That's why LiveRebel treats recovery as part of the release process. It rolls back deployments automatically if a release were to fail, and also allows users to initiate rollbacks when they choose. This is by far the hardest part of the Ops equation that we aim to solve, and we have a lot of fun while we're at it :) Shameless plug plugged.

TOO LONG, DIDN'T READ (TL;DR)

SUMMARY, CONCLUSION, FAREWELL AND A COMIC

“In devops we are happy even if is shit bad because we are know how much of worse is can able be.”

- @DEVOPS_BORAT

Summary (aka TL;DR)

Perhaps @DEVOPS_BORAT said it best: anyone in DevOps knows that things can be a lot worse. This is probably the most logical reason for embracing DevOps methodologies--if done right, the worst thing that can happen is that simply nothing improves. Although even this is difficult to imagine, since just discussing and reading about the culture of DevOps is a big step in the right direction.

In this report, we covered four major parts, and got some key findings from each:

PART I - THE WEEK-LONG GRIND: FROM DEVOPS AND TRADITIONAL IT OPS PERSPECTIVES

- Top 3 time-consuming tasks for Traditional IT Ops on a weekly basis:
  1. Communication (i.e. meetings, writing emails, planning) - 7.2 hours
  2. Firefighting - 4.8 hours
  3. Automating repetitive tasks - 4.6 hours

- Top 3 time consuming tasks for DevOps oriented teams on a weekly basis:
  1. Automating repetitive tasks - 5.3 hours
  2. Communication (i.e. meetings, writing emails, planning) - 5.1 hours
  3. Infrastructure improvements - 4.8 hours

- Traditional IT Ops teams require 41% more time for communication and 26% more time for fire-fighting than DevOps oriented teams, and spend less time on task automation and infrastructure improvements.

- DevOps oriented teams have 4 hours per week more to use for non-mandatory activities, spend 33% more time on infrastructure improvements and 15% more time on self-improvement and education.

PART II - APP FAILURES AND STELLAR RECOVERIES

- The top 2 causes of production failures are software quality (64%) and human error (60%)

- Although the most frequent alert method to production failures is an automated message to an on-call person (63%), the second most common is end user complaint to helpdesk (55%)

- On average, app failure recoveries happen over 2x per month, but the median occurrence is 1x monthly.

- A large minority (40%) of Traditional IT Ops teams require more than one hour to fully recover from a failure, while only 22% of DevOps oriented teams require that much time.

- DevOps oriented teams are nearly 2x more likely to recover from a failure in less than 10 minutes

- Nearly 2/3 of all organizations do not test their recovery processes in advance to ensure that they work when needed

PART III -
TOOLS OF CHOICE:
POPULAR PRODUCTIVITY BOOSTERS

- Puppet (40%) and Chef (31%) take the lead for popular infrastructure configuration tools
- Only half of respondents automate smoke testing at all. Only 1 in 20 automates all smoke testing.
- Selenium (71%) is by far the most popular test automation tool, \ followed by a distant JUnit (22%).
- Nagios (51%) takes the monster's share of the monitoring tools space, with custom tools in second place and New Relic & Zabbix used by 11% of respondents.
- In scripting languages, shell scripts are used by the majority of respondents, followed distantly by Python, Ruby and Perl.
- At 63%, vi/vim/gvim is the most popular editor, followed by good old Notepad (24%).

PART IV -
RELEASING SOFTWARE:
CONTROL, OR BE CONTROLLED

- DevOps oriented teams release apps in half as much time as Traditional IT Ops teams (average time per release: 36 min vs. 85 min, median: 15 min vs. 30 min).
- Apps deployed during weekday working hours 51% of the time, with all staff on hand in case there is a fire to fight. In 16% of cases, deployments happen during weekend nights. :-(
- When it comes to release automation, only 1 in 8 respondents have fully-automated processes. Nearly one-third do it manually.
- Around 75% of respondents rely on manual processes to recover from a release failure, which adds an increased risk of human error to an urgent recovery process.

Conclusion - DevOps just makes sense

Simply by discussing the DevOps culture, you are already "practicing" DevOps to some extent. Greater social interaction and communication between Dev, QA and IT Ops is going to make all sorts of things better, but in this report you can discern truly tangible benefits from embracing DevOps -- app delivery times are cut in half, and DevOps oriented teams get more time to spend on improving infrastructure & themselves, and so on.

Our conclusion: DevOps as a cultural and technological methodology should be a pure win for most organizations--although how painful the path will be getting there depends on different factors.

So we recommend you start by writing the next musical hit for your organization (you could call it "Call Me DevOps Shake"), because sitting on the sidelines and waiting for DevOps to come to you will blunt your strategic edge and lead you nowhere. Boom!

DEVOPS IS A JOURNEY

It is not something that you embark on and chalk off as success at the end. Adopting DevOps takes discipline and initiative to bring development and operations teams together. Read up on how other organizations approach adopting DevOps as a culture and learn from their successes and failures. Put to practice what makes sense within your group. Develop a maturity model that can guide you through your journey!

The goal is to make sure that dev and ops are on the same page, working together on everything, towards a common goal: continuous delivery of working software without handoffs, hand-washing, or finger-pointing.

SUPPORT THE COMMUNITY AND THE CAUSE

Dev and Ops need to look introspectively to understand their strengths and challenges, and look for ways to contribute towards breaking down silos. Together, they should seek to educate each other, culturally evolve roles, relationships, incentives, processes and put end user experience first.

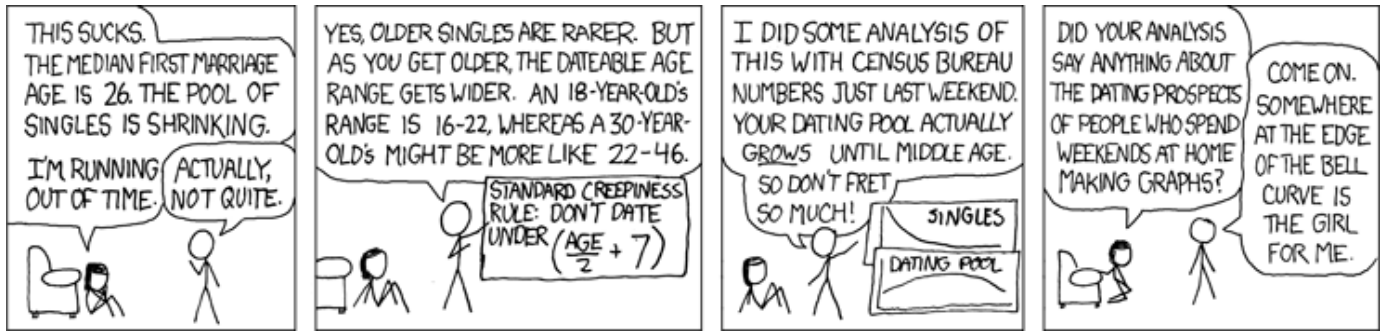
The DevOps community is small but burgeoning, and it's easy to find ways to get involved, like with the community-driven explosion of DevOpsDays conferences that occur around the world.

SET SMALL GOALS TO BE AWESOME!

Teams should collaborate to set achievable goals and milestones that can get them on the path to embracing a DevOps culture. Celebrate small successes and focus on continuous improvement. Before you know it, you will surely but gradually reap the benefits of bringing in a DevOps approach to application development and delivery.

Goodbye comic

We're big fans of XKCD, and since you've read a bunch of our awesome statistics, here is a comic about stats that _might_ be not so awesome for some ;-)



http://xkcd.com/314/

RebelLabs is the research & content division of ZeroTurnaround

Contact Us

Twitter: @RebelLabs
Web: http://zeroturnaround.com/rebellabs
Email: labs@zeroturnaround.com

Estonia
Ülikooli 2, 5th floor
Tartu, Estonia, 51003
Phone: +372 740 4533

USA
545 Boylston St., 4th flr.
Boston, MA, USA, 02116
Phone: 1(857)277-1199

Czech Republic
Osadní 35 - Building B
Prague, Czech Republic 170 00
Phone: +372 740 4533

This report is brought to you by:
Krish Badrinarayanan, Jevgeni Kabanov,
Ryan St. James & Oliver White