Designing for DevOps:

Understand the Implications for Security, Code Visibility, Team Culture, and More

developerThings;

A Publication by Stackify ©2017

Table of Contents

At Stackify, we build with developers in mind. We experience the same frustrations, hurdles and successes as our customers because we live and breathe code. In our Retrospective series, we pool together all our knowledge and experience on the most relevant and integral topics in the community and deliver them to developers. We've done the research so you can do the coding.

In our Designing for DevOps guide, we share the lessons that we've learned, from giving our team ownership to removing barriers to create efficiency. We look into the growth of the DevOps movement, what it looks like when you lay out all the responsibilities across Operations and Development, and how your security can increase. We hope this guide provides you with practical and useful ways for you and your team to deploy with more confidence.

A Retrospective on the Rise of DevOps

Give Your Development Team Ownership

DevOps has been a hot topic for the last several years. Most argue it is all about team culture and collaboration. Vendors have done everything they can to latch their products onto the DevOps bandwagon. Here is Stackify's opinion about what DevOps is and what it should be.

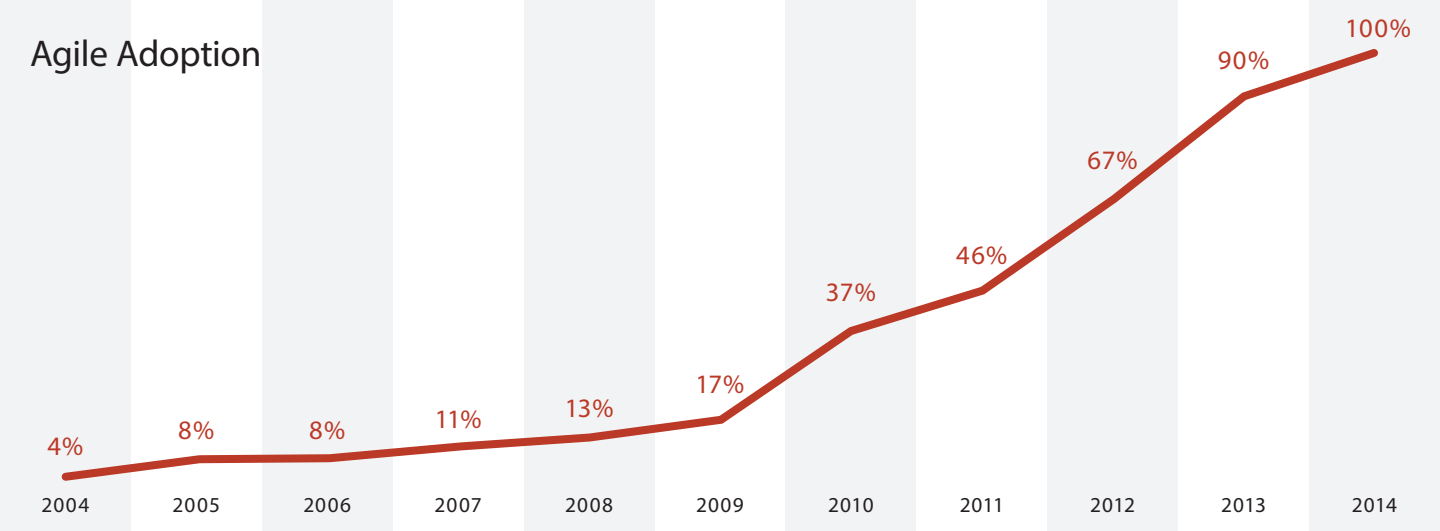
Why the DevOps Movement Was Started

DevOps first started as a movement around 2008. The goal of it was agile infrastructure. The movement has grown rapidly over the last several years. But how did we get here?

If you go back 10+ years, most enterprise applications were desktop applications. Shipping new software meant mailing CDs to your clients or downloading a new version. The transition from desktop to web (and mobile) applications in large enterprises have caused a massive growth of servers over the last 20 years. Now they have to provide the infrastructure to host these new web applications.

As part of this transition, businesses figured out that it was easy to ship new versions of code for web applications. This pressure created various agile development methodologies. It also created the necessity for tools like build servers and release automation. The entire way that most software is shipped has fundamentally changed in the last 20 years.

The rise of web applications and agile development are the real reason why DevOps became a movement. Let's discuss this graph from TechBeacon. This chart shows agile picking up speed around 2008-2009. It became the virtual universal standard in 2014.



The massive growth of servers and the pressure of rapid change led to gridlock at many large IT enterprises. They simply could not keep up with the demand of the business and development teams. DevOps was an outcry that there had to be a better way.

IT simply could not do anything fast enough. Operations had to become more agile and enable development teams to ship code faster. Server environments had to be deployed faster. These were massive challenges in large organizations.

Smaller companies can also benefit from streamlining how they do software deployments, provision servers, etc. It helps everyone, not just large companies. For example, if you worked for a medium-sized company during this change and found yourself using VMWare for the first time, imagine how surprising and helpful it would be to discover how quickly you could create a server. Abilities like that were proving immensely helpful.

Today, small companies and even teams within large companies are able to solve many of these problems by leveraging cloud computing. They can ship code whenever they want, spin up new servers on demand, and control their own destiny. They don't have IT Operations in their way. To us, that is what DevOps is really all about: developers doing operations tasks for their own apps.

What Developers Really Need

As a developer, nothing is more frustrating than working your tail off to then have to fight with IT operations to get a new server or deploy an application.

What every development team really wants is complete self-service. The goal is to remove every barrier possible to your development team. Give them the ability to ship code as fast as possible, while also giving them some guard rails to ensure they don't do anything wrong.

Every development team needs 3 key abilities:

1. The ability to quickly create new servers—be that virtual, cloud, whatever.
2. The ability to deploy software changes.
3. The ability to troubleshoot production application problems.

The goal is really simple: self-service.

You want your team to:

- Be able to do everything they need to do to write, ship and support their code.
- Be able to deploy a build whenever they are confident that it is ready to ship.
- Monitor production to ensure everything goes smoothly.
- Be on the hook to find and fix production problems if there are any.
- Get the 3 am text messages and phone calls when something goes wrong.
- Write code and ship it quickly.

You want your team to take ownership. You want everyone else to get the hell out of the way.

Give Your Development Team Ownership

If your company has a development team that creates software, we would argue that you should empower the team to take ownership. Give them all the access and tools that they need to have complete ownership of their success and failure of delivering a high-quality product. IT operations should help them achieve that goal.

Eliminating IT operations is not the goal. Every organization is different. DevOps is evolving into letting the operations team focus on the infrastructure and IT policies while empowering the developers to exercise tremendous ownership from the OS level and up.

We will always have IT operations. What we want is developers doing the operations tasks related to their own applications. We want them to take ownership. We want to remove any barriers that cause delays or excuses. DevOps is about improving velocity and quality.

We're not advocating that your production data center should be the wild wild west. IT operations can still help enforce smart security policies. Tools like Retrace can help give developers the insights they need to troubleshoot application problems, without giving them administrator level access.

Remove Barriers to Create Efficiency

Software development is complex. Every organization has unique challenges that slow down the development process. The goal of DevOps is to improve that process and remove barriers. At your company, culture or IT processes could be the biggest barrier. If you don't let your developers deploy code to production and IT operations is a nightmare to deal with, then yes, you have a culture and collaboration problem.

In other organizations, the barrier could be how long it takes to get new servers provisioned. The development team has decided that Redis can help improve their application. They got it all coded up and ready to go. If they can't get the new servers for it deployed, that becomes a big barrier and holds up the release.

However, there are lots of potential barriers that can exist. We would argue that anything that prevents developers from writing, shipping and supporting their code is a barrier. You need to remove these barriers if you want your software development process to be more efficient.

We have talked to companies that don't even allow their developers to access QA servers to look at log files or troubleshoot problems. This is clearly a barrier that slows down development.

The barriers are different in every company. It is not a universal problem. With Visual Studio, you can right-click and deploy an app to Azure with a couple clicks. You can log in to the Azure portal and your application monitoring tool to see how your app is performing. You have zero barriers preventing you from shipping and supporting your application. A junior developer can do this with no sysadmin knowledge. Cloud computing has eliminated a lot of barriers.

The development challenges that companies face are different. Facebook has tens of thousands of servers. The vast majority of business applications will never run on more than a few servers. They simply don't have the scale of traffic. You need to figure out what barriers are slowing you down; they differ from Facebook's.

Another good example could be QA. Perhaps your QA process slows down how fast you can do releases. That isn't a typical DevOps sort of problem you hear about. No matter what is slowing you down, you need work on it.

What Is DevOps?

If you want to ship code faster, create a culture of self-service and let your development team own their apps.

We would argue that the goal of DevOps is NoOps. That doesn't mean that nobody is doing operations stuff. It means that the development team is doing all of it for their apps. It means they are self-sufficient and take ownership. The developers are pushing code and are on call. Operations only assist if needed, instead of development only assisting if needed.

DevOps is a natural fit in the cloud. So many barriers that exist in normal data centers are gone. PaaS and application hosting features make it really simple to deploy your applications. It is the future and the way it should be. If you haven't made it to the cloud yet, we would suggest trying to mimic the same sort of agility in your own data center.

How to Divvy Up DevOps Tasks

Since Operations can mean a lot of things and even different things to different people, let's make a list of all the things operations traditionally does and figure out what developers should be doing, and which, if any, responsibilities should be shared.

Operations responsibilities:

- IT buying
- Installation of server hardware and OS
- Configuration of servers, networks, storage, etc.
- Monitoring of servers
- Respond to outages
- IT security
- Managing phone systems, network
- Change control
- Backup and disaster recovery planning
- Manage active directory
- Asset tracking

Shared Development & Operations duties

- Software deployments
- Application monitoring & support
- Server provisioning and configuration

Some of these traditional responsibilities have changed in the last few years. Virtualization and the cloud have greatly simplified buying decisions, installation, and configuration. For example, nobody cares what kind of server we are going to buy anymore for a specific application or project. We buy great big ones, virtualize them, and just carve out what we need and change it on the fly. Cloud hosting simplifies this even more by eliminating the need to buy servers at all.

What part of the "Ops" duties should developers be responsible for?

- Be involved in selecting the application stack
- Configure and deploy virtual or cloud servers (potentially)
- Deploy their applications
- Monitor application and system health
- Respond to applications problems as they arise

Developers who take ownership of these responsibilities can ultimately deploy and support their applications more rapidly. DevOps processes and tools eliminate the walls between the teams and enables more agility for the business. This philosophy can enable the developers to potentially be responsible for the entire application stack from OS level and up in more a self-service mode.

So, what should the operations team do then?

- Manage the hardware infrastructure
- Configure and monitor networking
- Enforce policies around backup, DR, security, compliance, change control, etc.
- Assist in monitoring the systems
- Manage active directory
- Asset tracking
- Other non-production application related tasks
- Manage software not developed in-house

Depending on the company size, the workload of these tasks will vary greatly. In large enterprise companies, these operations tasks become complex enough to require specialization and dedicated personnel for these responsibilities. For small to midsize companies, the IT manager and 1-2 system administrators can typically handle these tasks.

DevOps is evolving into letting the operations team focus on the infrastructure and IT policies while empowering the developers to exercise tremendous ownership from the OS level and up. With a solid infrastructure, developers can own the application stack, build it, deploy it, and cover much if not all of its support. This enables development teams to be more self-service and independent of a busy centralized operations team. DevOps enables more agility, better efficiency, and ultimately a higher level of service to their customers.

Increase Your Security with DevOps

DevOps Does Not Have to be a Security Risk

One of the biggest challenges for development teams is having good visibility into production deployments. It is nearly impossible to track down application problems without access to critical data. Developers need access to a range of things, including application performance reporting, configurations, log files and more.

Possible DevOps Security Issues

DevOps typically refers to topics around application deployment, server provisioning, and application monitoring. All three of these topics have potential security implications.

Application Deployments

One of the best things about using continuous integration and deployment tools is their ability to create a repeatable and dependable way to deploy your application. How you deploy your application is scripted out and works the same way every single time.

From a DevOps security perspective, we see this as a huge upgrade over someone manually pushing code. It allows you to implement controls and security policies into your release process.

We are also starting to see new ways to add security scanning and testing into the build process. Products like Contrast Security are very interesting.

Server Provisioning & Configuration

You have probably heard of infrastructure as code. Similar to scripting application deployments, scripting server deployments allows you to document and control the process.

By scripting out server configurations, you can also easily implement specific company policies. Things like what ports are open, automatic updates, and more.

Deploying to the cloud also changes everything. At Stackify, how we deploy to Azure is part of our application itself. We don't even think about server provisioning or server configurations. Microsoft Azure takes care of securing our servers, Windows Updates, and other common issues.

Scripting server configurations enable security experts to have better visibility and be part of the security conversations throughout the process.

Application Monitoring

When it comes to monitoring and troubleshooting application problems, a DevOps approach solves a lot of security problems. The goal of DevOps is to create collaboration and improve the working relationships between development and operations. Monitoring is a perfect example of where a company can gain efficiency and even security with a DevOps mentality.

By giving developers access to the tools and more data, they no longer need administrator level access to production. You can also get more developers involved in supporting their applications.

Developers Need Data, Not Production Access

In the past, many organizations were forced to give developers administrator level permissions so they could support their apps. It was the only way for them to see if their apps were running, the health of them, and access basic things like log files. This, of course, causes a lot of security concerns.

What developers really need is access to lots of data. Having to log in to servers one by one is not a good solution if your app runs on multiple servers.

Here's what developers really need to support their apps:

- Deployment history: What changed and when?
- Application configurations: Is everything configured correctly?
- Application errors: Is there a critical error going on?
- Application log files: Logs are the eyes and ears for developers.
- Server metrics: Need to double check server CPU, memory, disk, and network performance.
- Application metrics: Do we have issues with garbage collection or other key metrics?
- Application performance: APM tools are invaluable for identifying why an application is slow or not performing correctly.

Developers need tools to aggregate this data together across multiple servers. Traditionally, developers have used multiple monitoring type tools. APM solutions, like Retrace, can help solve this by combining all the data into one place.

Providing your entire development team access to this data fits into the DevOps mentality and solves some security challenges.

By leveraging DevOps best practices, companies can increase the velocity at which they do releases while improving security. DevOps and security issues related to it will continue to be big topics.

Automate Security Testing with DevSecOps

Every company wants to see their company getting press and media attention. Unless it is due to a hacker and a security breach. Every few weeks you see in the media stories of companies who were hacked. Getting a new credit card every few months because the data was hacked has been routine for most of us. The more that our world revolves around the internet and technology, the more cyber security becomes a big deal.

Software applications are complex and can potentially have lots of different types of security issues. The issues range from bad code to misconfigured servers and everything in between. Solving this problem requires everyone to always be thinking about the security implications of what they are working on. DevSecOps is a new movement to do just that. The goal is to get developers to think more about security principles and standards as they are building their applications.

Integrating DevOps + Security = DevSecOps

The goal of DevOps is to give development teams more ownership in deploying and monitoring their applications. Automating how we provision servers and deploy our applications is at the heart of DevOps. Automation helps us move faster and ship higher quality products.

Adding security to this same automation is the goal of DevSecOps. Companies want to create strong security policies and standards without slowing down the development process. Security has to be part of the process and automated to not slow us down.

Things like DevOps and DevSecOps continue to change the meaning of the software development life cycle(SDLC).

Tools for Automating Security Testing

One of the goals of DevSecOps is to build security testing into your development process. There are new tools that can be used to help achieve and automate it across the development lifecycle. Here are some of the types of tools that exist:

- Cloud infrastructure best practices: Tools built into the cloud like Microsoft Azure Advisor and third party tools like evident.io can help scan your configurations for security best practices.
- Automate security tests: You can now create and run automated security tests just like you would unit tests or integration tests. Gauntlt is a popular free framework for automated these types of tests.
- Code Analysis: Tools like Veracode can scan your code to find potential vulnerabilities in your own code and open source libraries.
- Runtime application security: Tools like Contrast Security run within your application in production and can help identify and prevent security issues in real time.

Hopefully, this gives you some ideas of the types of security testing and automation that can be built into your development process. Check out our resources at the end of this article to see the tools mentioned here as well as a huge Github list of tools and resources.

Security Unit Tests

Application security is something that needs to be thought of when we start writing code. Just as we write and run unit tests, running some automated security tests can help ensure new vulnerabilities were not introduced. Gauntlt provides some neat capabilities around this.

For example, as part of your deployment process perhaps you provision new servers or deploy some Docker containers. You could then automatically run some various basic security tests.

- Scan for open ports on your server
- Test to see if your server responds to pings or not
- Do an HTTP request and validate the cookies in the response
- Test various HTTP verbs. Is it supposed to support DELETE, PATCH, etc?

Conclusion

Software and automation continue to change our world. Automation within the software development lifecycle helps us ship our code faster and at a higher quality. Adding security testing into that automation will also help us create more secure applications. DevSecOps is still a new thing and is evolving quickly. Hopefully, we've given you a few ideas you can use in the future to improve the security of your apps.

Recommended Resources:

- Shifting Security to the Left
- Awesome DevSecOps on GitHub
- DevSecOps: 9 ways DevOps and automation bolster security, compliance
- OWASP Top 10 Application Security Risks
- Microsoft Azure Advisor
- Evident.io
- Gauntlt
- Veracode

Support Production Applications the DevOps Way

Long-term success of your application depends on proper maintenance and monitoring of your applications as you continue to roll out new features. Thanks to agile development, that is usually a weekly or monthly cycle at most companies. A constant stream of new features is awesome, but with that comes some risk in introducing new bugs, performance problems, or instability. This constant stream of change coupled with the overall complexity of today's applications makes them difficult to support.

Application Support: "It Takes a Village"

Collaboration between developers and IT operations is needed to keep your applications running and your customers happy. If you only have a couple senior developers who handle all the production support issues they will have little time to deal with more important issues like architecting new features.

The key to supporting production applications is removing the bottlenecks and getting more people involved in application support. Operations needs to make sure that the development team has visibility of basic troubleshooting information like errors, logs, and key metrics while ensuring security. Without this information, it is hard for developers to fix basic bugs and improve application performance. They must contact other developers or system administrators for help which creates a huge bottleneck. You could say, "It takes a village."

As we have detailed, granting access to production data raises both security concerns and fear of unauthorized changes to the servers without going through proper release processes. But again, developers just need visibility to the aggregated information. This will allow them to find the source of the problem fast, while not creating any security risks or letting developers create new issues while fixing others.

DevOps Should Include Monitoring & Production Application Support

The goal of DevOps is to create collaboration and improve the working relationships between development and operations. The DevOps movement initially started focusing on software deployment and continuous delivery. However, support of production applications is the second and equally important component in the application lifecycle that is sometimes overlooked. To solve

production problems quickly, developers and operations cannot be pointing fingers at each other. They need to work together as a team.

To help support production applications, developers need:

- Visibility and notifications around all application errors
- Access to centralized aggregated log files for viewing and searching
- Basic server utilization trends and stats (e.g. CPU, memory, etc.)
- Recording and alerting of key application metrics (KPIs)
- Tracking of application web page load times
- Ability to access the application database and run test queries

Most companies utilize several different tools to track all of this data. This makes it difficult to correlate and identify issues. It is also very hard to train all team members on how to effectively use all the available tools, not to mention the economic inefficiency. Companies should look for solutions that combine these features into one dashboard so that solving issues can be done quickly and with minimal training.

Stackify gives developers access to all the data they need to monitor and troubleshoot production applications, without giving them administrator type access to all the servers. From one application dashboard you can see where the app is deployed, the current health, performance stats, recent errors, full logs, and key metrics. Stackify makes it easy for companies to embrace DevOps when it comes to supporting production applications.

3 Key DevOps Needs for Every Development Team

What do development teams need to embrace DevOps? DevOps should be about empowering your development team to be able to do their job and support their apps from A to Z. There are the 3 critical things every development team needs.

1. Automate Builds and Deployments

If you want to deploy software quickly and consistently, you need a consistent build process. How you consistently do builds is up to you. Your tech stack and preferences will dictate that to some degree.

2. Provisioning Servers

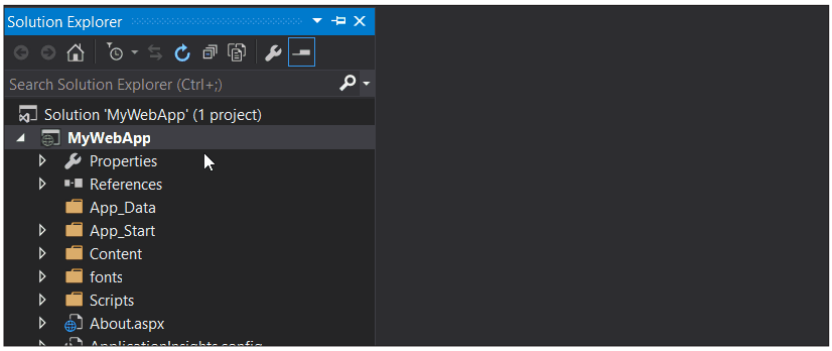
Technically, you should probably provision servers before you try to deploy to them. But as a developer, you might think about getting a successful build done first. So, this could certainly be Step 2.

Provisioning servers is a very broad topic that varies wildly based on the type of app and how you host it.

In a lot of organizations, they tend to deploy to the same servers all the time and probably don't set up new servers very often. Most apps never run on more than a few servers. Their apps just don't get enough traffic. In those cases, provisioning new servers are less of an issue. At Stackify, we add new servers all the time due to our massive scale. It is a big deal for us!

Option 1: Use an Application Hosting Engine (Public or Private Cloud)

If you have your choice, we would highly recommend deploying your applications in a way that automatically provisions the servers for you. This helps to ensure that developers don't have to get involved in the actual creation of the servers and that it is done consistently every time.



For example, with Visual Studio and Azure, it can be done in a couple clicks. Here's an example below. You should really do this as part of your build server and deployment process. That will give you better tracking of when deployments are done, what was in them, etc.

Cloud application hosting options:

- Azure App Services & Cloud Services
- AWS Elastic Beanstalk
- Google App Engine
- Heroku
- … and more

On-premise private cloud options:

- Cloud Foundry
- Azure Stack

Cloud computing is not an option for some organizations. However, developers still need the same sort of PaaS environment and tools to make them more productive. Containers might help with this.

Option 2: Containers

Another good way to solve this problem is Docker containers. Instead of provisioning lots of virtual machines, you could pack a bunch of lightweight Docker containers together. Although, you still need an easy way to manage how many instances of them there should be, deploying new versions of them, etc.

Docker management tools:

- Docker
- Rancher
- Cloud66

Option 3: Configuring Servers

If you have to provision actual servers, first off, you have our sympathies.

We still have nightmares about having 15 web servers that do the same thing and frantically needing to set up more of them. Today, we solve that problem by clicking a button in Azure. Back then, we had a VMWare image saved off that had to be cloned. Inherently, 10 things changed since the image was taken and nobody knew what they were.

The problem with servers is knowing the exact configuration needed. This problem varies wildly based on your application. This involves things like installing your web server, application framework, OS updates, security permissions and other dependencies.

To solve this challenge, configuration tools like Puppet and Chef have become really popular.

3. Optimizing and Monitoring Application Performance

The DevOps community has always focused on deploying applications. We have always felt like the monitoring side of it gets forgotten. Developers need visibility on data about how their apps are performing to troubleshoot problems quickly. It is an essential component of DevOps.

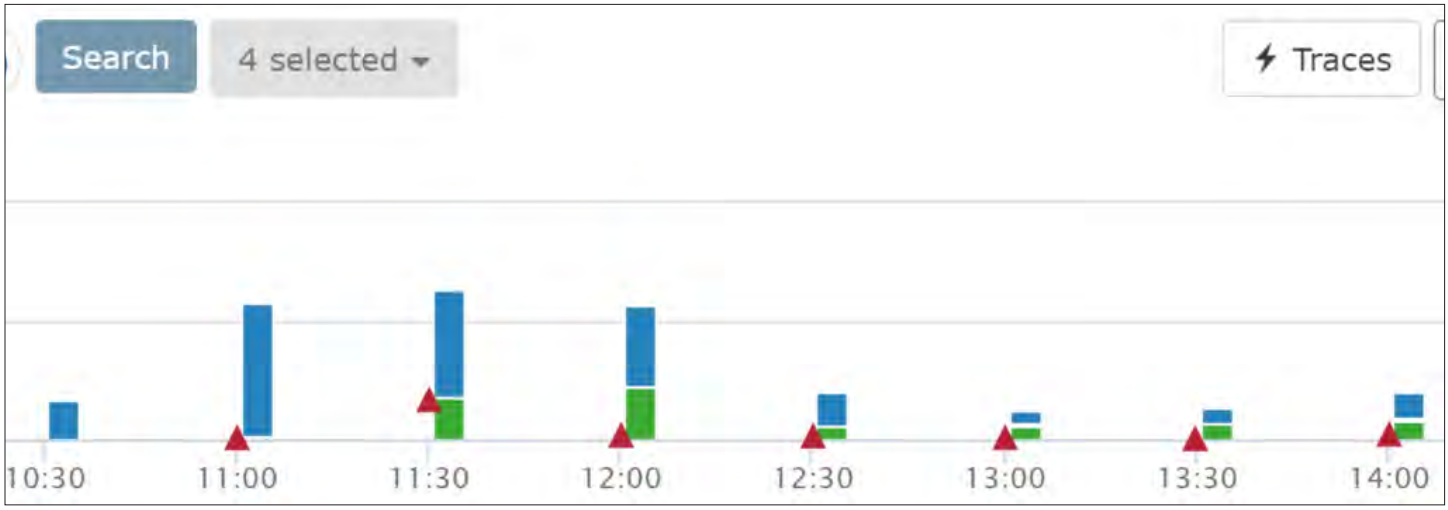
Application monitoring is a completely different beast than server monitoring. 10 years ago, we monitored if a server was up or down and the CPU. We monitored Exchange and our critical web servers the same way. Obviously, that didn't provide much insight into how our applications were performing.

Today, application monitoring is an umbrella term that could mean a lot of different things. There are 6 specific things that we feel developers need:

- APM: Code-level application performance visibility
- Transaction Tracing: Code-level traces of what your code is doing
- Metrics: Server, application, and custom metrics monitoring
- Logs: Aggregation, searching, and management
- Errors: Aggregation, reporting & alerting
- Alerts: Robust monitoring and alert capabilities

If you want to understand the performance of your application, you need all of these things. Each of them tells a small part of the story.

Let's take errors as an example. While doing any software deployment, monitoring for errors is the first line of defense for finding problems. Odds are, the new release will likely cause some new errors they have never seen. It is just the nature of software development. Developers need to be able to quickly see these new errors so they can rapidly hot fix them.



Application monitoring tools are the eyes and ears of the development team in production. In lots of companies, developers do not have access to production servers. That is easy to understand for multiple reasons. However, it is very difficult for developers to troubleshoot problems if they can't get access to log files and other data.

Companies can enforce strict security rules in production and give developers everything they need by leveraging the right types of tools.

Stackify was founded because we were frustrated that our development team didn't have the tools that they needed. We had Nagios, Splunk, and multiple other tools. None of the developers had access to them. Even if they did, the tools only told part of the story and were hard to use.

Developers love and need tools. If you want to embrace DevOps, you need tools to help them build, deploy, and monitor their applications. DevOps is about removing barriers. These essential functions can help developers take ownership of their apps.

Deploy Your Software with More Confidence

Everyone wants to ship their code faster. Agile development and all the variants of it have helped companies release software more often and spend less time in large, waterfall planning and project management. Agile development still has one big problem… confidence in software deployments.

Developers have no idea if they are really ready to ship their new version.

Does this happen in your office?

You just finished a sprint and you are ready to push it to production. You ask everyone on the team, including the development manager, how they feel about pushing to production and you get a long, uncomfortable silence.

The problem is that the tasks are all done, and "testing" is finished, but nobody has any real idea if the code is going to blow up spectacularly in production, or work just fine. How often do you delay releases because you are nervous about potential problems? What issues are lurking right beneath the surface?

The #1 problem with agile development is confidence in your releases.

There is a lot of risk anytime you do a deployment. How do you confidently measure that risk?

Since nobody knows, eventually someone says, "Let's just ship it."

Then the next day is a scramble between dev and ops to deliver an ad hoc release while fielding frustrated calls and messages from your customers.

Is agile working well for you?

Velocity is not the goal of agile development. Shipping new value to your customers as soon as possible is the goal. However, no part of that means to ship sloppy code as quickly as possible. Going fast in the wrong direction is not the right kind of velocity you are after.

In your last sprint, what % of the work items were bug fixes?

If you spend a lot of time on bug fixes and not on new features or improvements, then your current velocity is not working. You need to put more focus on quality and less time on velocity.

One of the great things about agile is shipping things quickly and being able to take feedback about it to quickly make changes. This feedback from your users is critical to guide what development work should be done. But don't make the mistake of only listening to your users for feedback.

Listen to your users and your code for feedback.

What developers need is feedback loops at every step of the development cycle. They need it while writing their code, testing it, building it, deploying it, in QA, and in production. Utilizing this constant feedback can help you quickly identify problems before they get to production.

Integrate feedback into your development process.

Before starting the next step in the development process, you need to review feedback from the previous step. It all starts with the start of the process, planning your next sprint!

Before you finish planning the next sprint… how are things going in production?

Performance review: First thing you need to do is review how things are working in production. Here are some things you should do before every sprint planning meeting:

- Review application errors that are occurring
- Ensure most important web requests are performing well
- Look for poor performing SQL queries
- Decide if any web requests need any performance tuning
- Verify if all changes from the previous release are performing well

Reviewing your APM solution to see how things are going in production is a critical feedback loop. You want to find potential performance problems before your customers do. It is also important to understand performance before your next release so you have some sort of baseline to compare to.

The best time to find and prevent bugs is while creating them. As you start working on new work items, here are some suggestions on how to improve code quality.

IDE plug-ins: After using tools like Resharper, it is hard to live without them. They are awesome at helping point out possible places in your code where common exceptions or problems could occur.

Code level transaction tracing: Depending on which programming language you are using, there are amazing tools now that can show you most key things about what your code is doing and how long it takes. These tools are great at helping you understand what SQL queries, web services, and other things your code is calling.

Code reviews: Having a second set of eyes look at your changes is never a bad thing, especially if it is a critical part of the code or complicated. Don't nitpick over silly things like how people named variables. Focus on the things that matter and ask questions.

Automate your builds and deployments to remove human error.

Build server: Having an automated build and deployment process is critical. If your current process involves asking Bob (what else would you name the builder?) to do a build and push code manually, your process is full of human error and too dependent on Bob. You should be using tools like Octopus Deploy, Jenkins, Continuum, TFS, or others to automate your build and deploy process to make sure it is done the same way every single time and Bob doesn't skip step #3.

Unit tests: We don't believe that you need unit tests for everything. They shouldn't be a replacement for a compiler. But we do think they are highly valuable for testing complex scenarios, business rules, etc. Have your build server run your unit tests to help validate the build before it deploys.

Review code commits: One of our favorite things about a good build and deployment tool is its ability to show you exactly what code commits were included in the new build. This is a good way to verify if anything has sneaked its way in that you weren't aware of.

Your build server is a good first line of defense to make sure nobody checked in bad code. The last thing you want to do is go to push your code to production and find out you can't even compile.

Collaborate with QA and do performance reviews.

Clearly stipulate to QA what to test. It may seem obvious, but nobody does it. If you want your QA team to find problems efficiently, it is critical that you tell them as many details as possible about what they should be testing for. We suggest that there should be a separate and required field in your ALM tool for this.

Performance review: Earlier we mentioned important things you should be reviewing in production to assist in planning your next sprint. You should be doing the same sort of things in QA to see how performance changes between builds and looking for new errors.

Synthetic tests: Use selenium or some other tool to set up and run automated synthetic tests for basic and key parts of your application. Make sure a user can login, navigate to key pages, etc.

After your software deployment, quickly check these things!

During your deployment can be a nervous time. The last thing you want to do is push code, do a quick test to make sure your app loads, and call it done. If you want to sleep at night, you need to do a few more things!

Watching production after your software deployment is a critical part of the feedback you need to ensure that your release was a success. If things don't look right, you can quickly fix the problem or decide if you have to roll back the deployment. Utilize your various monitoring and APM tools to verify these items.

Look for new errors: After almost every single release, you are likely to see some new errors being thrown by your code. Be sure to check your error tracking tool to see if anything new is happening.

Check your error rates: A certain amount of errors is normal in most apps. A certain amount of noise, goofy errors, transient errors, etc. Ensure that the overall level of errors is consistent with normal and hasn't rapidly increased.

Watch requests per minute: Make sure that the overall traffic rate on your site looks normal. If traffic drastically goes up or down either way, something could be wrong.

Review top requests: Double check that your key web requests that get accessed a lot still look normal.

Database performance: It is also a good idea to make sure your database and potentially other dependencies still look normal. If you pushed any SQL schema changes, there is definitely a chance things could be better or worse.

Utilize feedback to build release confidence.

Every step of the development process can provide a lot of feedback about the quality of your code and software. Utilizing these quick feedback loops can help you deploy more often and with more confidence.

Is Your Developer Team Designing for a Disaster?

Development teams work at top speed, and the environment in which they work is demanding and complex. Software is no longer considered done until it's shipped, and there's been a tendency to view shipped software as preferable to perfect software.

These philosophies, created in large part by agile and lean methodologies, celebrate deployments and meeting release cycle deadlines. But have our standards of working software quality been trumped by the rapid pace of delivery? We may be designing faster, but are we designing disaster?

We practice Agile development at Stackify, and are advocates of the methodology as a system to build and deploy better software more frequently. What we don't subscribe to is the notion that the process we take to create better performing, high-quality software is more important than the software itself. If the process gets in the way of the product, it's time to re-evaluate it.

The beauty of a process like Agile is that you can modify it to suit your team and what's most important in your delivery. Here's a bit of what we've done to optimize, and some of the things we've learned along the way.

Don't let quality draw the short straw.

We've yet to see a project that doesn't have a problem when it gets past development and heads into the final push towards "done." Primarily, we see one of two things happen: testing cycles are compressed or incomplete due to time constraints. A major contributor to this is that code often isn't ready to be tested until it's all ready. As the sprint burns down, more and more code tasks are complete but they've yet to be reviewed, merged, and deployed to test environments. The code is rushed through QA, and ultimately, issues are found in production. Fixing those issues robs time from the next sprint.

Testing doesn't get compressed, but it extends the sprint in order to be complete and/or fix problems. In a scenario where sprints overlap (i.e. once dev is complete for Sprint A, developers begin picking up tasks for Sprint B), this has a domino effect throughout the entire schedule of releases. We commonly joke about this being a "death march," as it creates wider deltas of code diffs, more complex merges, and a general log jam of productivity.

We are no stranger to these phenomena ourselves. Like any other dev shop, we have testing tools that help out, running automated UI tests, unit testing around core functions, and automated/ manual integration tests of complex system functions. But, it's still really hard to get through everything we'd like, at the level of detail we'd like, in a reasonable time frame. There are two criteria we ultimately base a go/no-go release decision on:

- Confidence: Have we accomplished our goals for the release while also improving (or at least maintaining) our application's overall performance, stability, and reliability?
- Risk: What have we changed, what can it impact, and do we fully know the scope and scale of that impact?

Throughout our development process, we use Prefix and Retrace to help us build confidence toward the next release and to assess how much risk we have. Our developers run Prefix on their development machines, finding and eliminating bugs, bad code patterns, and performance issues before they commit code.

Our developers, QA team, and management all use Retrace to look at overall performance and new and regressed errors in each one of our pre-production environments at each stage of our dev lifecycle. We know, from build to build, if we have introduced new problems and moved the platform forward or backward. It's a tangible measurement of the overall health of our release.

Don't fear a punch.

As Mike Tyson once said, "Everybody has a plan until they get punched in the face." You're going to get punched in the face. It's inevitable. There will always be a problem, an unexpected server or cloud failure, a critical bug uncovered, support requests, or an "urgent" need from someone else in the company to get something that isn't in the plan done ASAP.

If you know you are going to get punched, your process must plan for it. At a minimum, you must be leaving some capacity to deal with it. If you're doing it well, you should have an "expedite" lane on your planning board (along with an established process) to deal with items that come out of nowhere with a lot of urgency.

Be wary of the tendency to let too many tasks fall into the "urgent" category. It happens sometimes due to pressure from somewhere in your organization, or just because of fallout from a chaotic application. By its very nature, urgent work will always have higher risk and the introduction of an opportunity for shrinking quality. If it's all urgent, nothing is, and you're just sacrificing the quality of your product.

Don't be a process zealot.

In theory, having an Agile development shop sounds great. In practice, it can be great.

A trap that many fall into is trying to carry out a textbook implementation of agile, often times attempting to cram what really happens into what should be happening. But it's not a one-size-fits-all process.

Craft your Agile implementation around the way your business needs to deliver software. A great example of this is the concept of "scrumban" that many teams have started to adopt. The focus is on limiting the work in progress, but still having timeboxed sprints and releases.

SCRUM would dictate that you release at the end of each sprint. If this could have a negative impact on your customers, would be difficult to manage because of frequency, or for any other reason doesn't fit your needs, then simply modify the process to work well for your specific project.

Make agile work for you.

Agile is meant to help teams get on a cycle of continuous learning and deployment, and if systems are not in place to check for success or quality, you'll have a lot of problems. Any amount of downtime or rework is too much for software teams and businesses that rely on applications as their primary revenue source. Efficiency, in regards to the software development lifecycle, should not be married to "faster" deployments, but stabler, higher quality deployments. Successful deployments should adhere to the rule of "working code." Too many times, "done code" is not "complete code," and then it turns into an emergency. Agile teams must build a better criteria for success to accomplish the company's objectives for the code, and not just adhere to a time-boxed deployment schedule.

Many companies wait until there's a problem to make an adjustment to their team processes and products, but we think time is better spent preventing problems and avoiding work that interrupts from the greater business objective. Prefix and Retrace are more than pieces of software, they're comprehensive app performance tools that work alongside your agile team to help you plan ahead, build proactively, and deploy without interruption or emergency.

There's nothing like a few emergencies to challenge your team's morale and focus. Agile is a great start, but building in support structures of purpose and product means that your developers can focus on their code performance and your company can focus on building better products. It's a win-win for everybody.

8900 State Line Rd. STE 100
Leawood, KS 66206

Stackify exists to increase developers' ability to create amazing applications as quickly as possible. We believe that the ideal development team, today and in the future, is consistently optimizing its output across the entire lifecycle of an application; from development to testing to production. Stackify's mission is to give developer teams easy access to powerful tools, which enable them to take the lead in delivering the best applications as quickly as possible.

If you're a developer, team lead, or architect, Stackify's tools were built for you. In fact, we have two game-changing, code performance products no developer or dev team should ever be without.

Prefix FREE

Prefix is a popular developer tool for finding and fixing bugs while you write your code. You have profilers and debuggers, but nothing is like Prefix, and it's free!

Retrace FREE TRIAL. STARTS AT $99/MO

Retrace is an APM tool built specifically for developers and dev teams. Our agent can install on pre-prod or production servers to make black boxes transparent. Try Retrace free for 14-days.

"We're using Stackify to find potential misbehaviour in our software and to improve the performance of our support team. Most of our incoming tickets can be solved or least linked to our trace/error logs in Retrace.

As Stackify supports diagnostic contexts, it is easy for us to filter the logs by LEVEL or customer and to retrieve valuable metadata about the log context.

– Julian Neuhaus

There is good documentation how to integrate Stackify in your business applications. The 'Error View' is one of our most used features and helps a lot to find, analyze and keep track of bugs.

If you're using services/tools like Docker, AWS, Azure or Bluemix, it is a must have."