GIGAOM RESEARCH

DevOps is Crippled Without Continuous Integration

Report by Chris Riley

This report is underwritten by: Sauce Labs

DevOps is Crippled Without Continuous Integration

02/02/2015

Table of Contents

1 Executive Summary

The idea of shipping code faster has been a priority since the practice of software development began. Therefore it would not come as a surprise that DevOps is about more frequent, higher quality releases. With this as their motivation, modern development shops naturally embrace the ideas behind DevOps that help them achieve these goals.

What makes DevOps different from traditional approaches to development is its focus on people first. It takes people to build a strong software development engine. (On the other hand, people have been the source of many delayed releases.) Only when deliberate communication between teams is established can organizations embrace the modern practices of continuous integration, continuous delivery, and continuous deployment. These processes and tools all share the component of taking code automatically upon a developer's commit and moving it into an environment for testing, integration, staging, or production. Together, they are at the heart of what is called the DevOps practice.

To become a DevOps shop, organizations must first embrace continuous integration (CI). This paper discusses the what, why, and how of CI and its role in the modern development process.

Key Findings:

1. Continuous integration can and should be part of every DevOps team. Unlike continuous delivery and continuous deployment, which may or may not fit into smaller development organizations, CI can fit into and benefit every modern development pipeline, irrespective of size or scale.

2. Continuous integration without a strong quality assurance strategy cannot succeed. Alone, CI is just a set of tools and processes for moving code. In order for it to be successful, it has to be driven by a strong QA strategy.

3. Well-implemented CI opens the doors for more proactive development environments. Most QA efforts fall into the trap of ever-decreasing test coverage and regression-only testing. CI can transform teams; allowing them to expand test coverage and participate in more exploratory testing.

4. Continuous integration is not just for web applications. Originally, it was thought that elements of mobile applications prevented CI from being a part of the development process. Mobile applications do pose a new set of challenges, but CI can and should be used in mobile development as well.

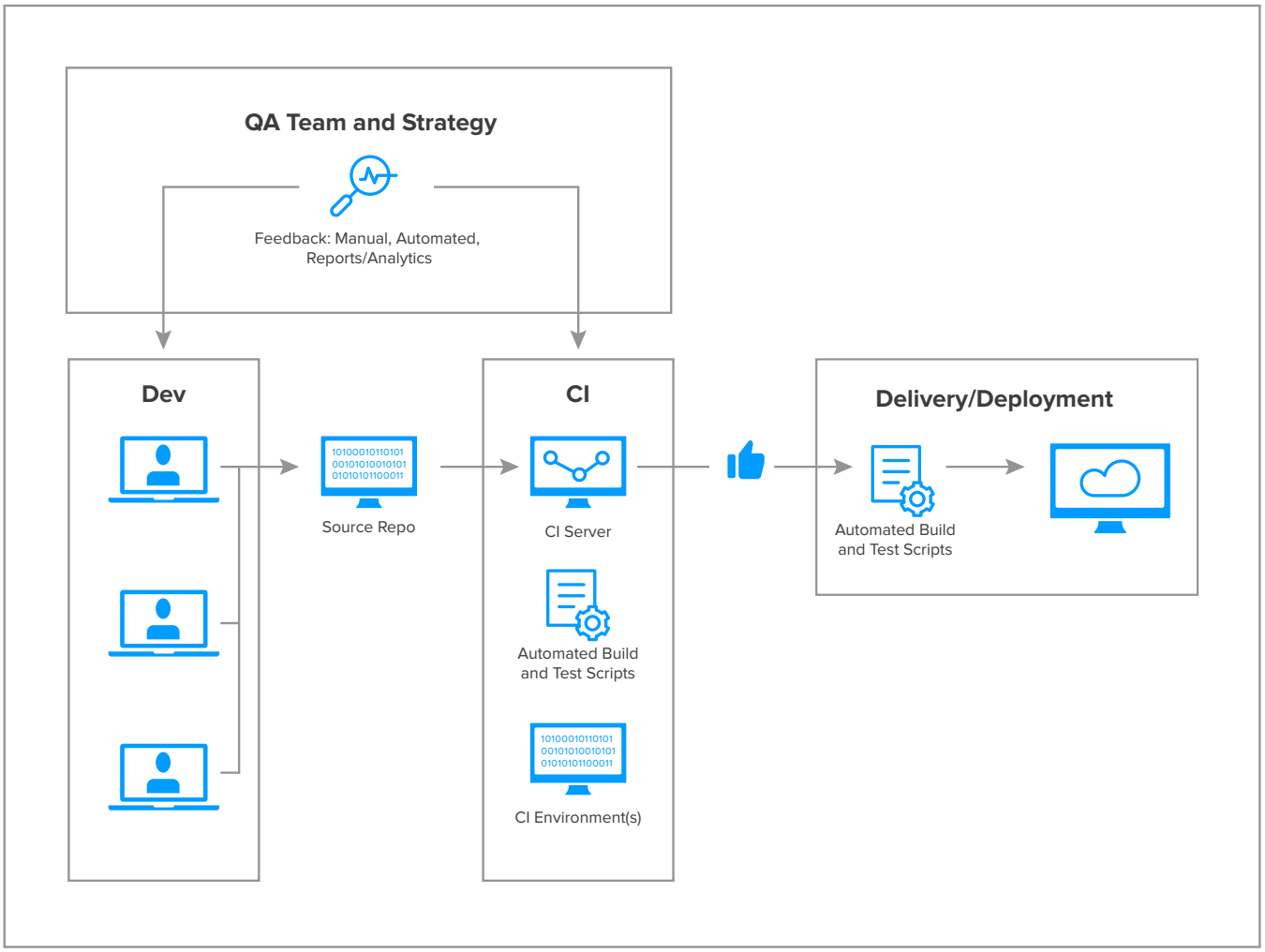
Based on these key findings, it's easy to see that DevOps adoption is crippled without CI. As organizations make the move toward development with greater automation, they cannot jump immediately into delivery and deployment practices. CI is a key element that helps pave the road to DevOps. It is also necessary to ensure this component works correctly before any other aspect of DevOps can be implemented.

2 What is Continuous Integration?

Continuous integration is a process. It takes developers' code and deploys it upon every commit to what is called an integration environment — a set of infrastructure where the entire application stack is running. It is a place where all automated unit and functional testing can be run that is non-production and has no impact on the user base. The integration environment is the key to succeeding at CI and embracing the DevOps framework.

The "Integration" portion of continuous integration means that all components of the application come together in one location; a build. "Continuous" means that these builds happen both regularly and automatically. A build acts as the process of putting source code together and verifying that the software works as a cohesive unit.

The following illustration shows the entire process of CI, and how it integrates the efforts of development and testing teams.



The steps in a typical CI process are as follows:

1. First, developers commit code to the source repository. Meanwhile, the CI server on the integration build machine is polling this repository for changes (e.g., every few minutes).
2. Soon after a commit occurs, the CI server detects changes in the source repository, so it retrieves the latest copy of the code and then executes a build script, which integrates the software.
3. The CI server generates feedback through various mechanisms. It can email project members, it can be viewed manually by developers and testers, and automated scripts can compile test results and analytics for review.
4. The CI server continues to poll for changes in the version control repository.
5. One of the key purposes of CI is to produce feedback on an integration build, because ideally everyone should know as quickly as possible if there is a problem. By receiving this information promptly, developers can address bugs before delivery and reduce time, cost, and risk.

There is some confusion about the term 'continuous integration' as far as its scope of testing and use cases goes. In this paper, we are discussing a comprehensive definition and framework for CI which includes:

1. Unit testing
2. Component testing
3. Functional testing
4. Manual testing

In the early days, CI was most associated with unit and component testing, but today functional testing is the largest and arguably most complex component.

3 CI Is for Everyone

Because CI is not a threat to production, it helps increase overall release volume, and is a safe place to test even unstable versions of the application. It becomes a convenient place for any team approaching DevOps, a modern software delivery pipeline. When you look at the spectrum of available tools and process in the modern development world, CI is particularly effective at onboarding organizations to a DevOps practice.

The other benefit of CI over deployment and delivery processes is that it can fit into organizations with minimal disruption. Continuous deployment requires far greater considerations. Most of the time continuous delivery is relevant only for applications that have a significant number of users, have a high transaction volume, and/or are geographically distributed. Thus, this process does not apply to everyone.

Not New; Refined and Automated

The idea of automation and testing is not new. Application Lifecycle Management (ALM), and Information Technology Infrastructure Library (ITIL), have been pushing the idea for several years. But what is different is who owns it, its fluidity, and idea that if you can't automate a portion, there is a problem.

CI infrastructure used to be the machine under a developer's desk, or a VM provisioned and eventually abandoned in AWS or other Cloud, for quick concept or code tests. The problem with these environments was that they were personal and absent from all the other parts of the application. CI creates new flexibility for developers to test without fear and allow failure to be a way to improve faster. CI also establishes consistency on quality of releases, and process.

Rather than developers creating personal testing environments or performing quick tests on their machines, the entire team can benefit from the daily stream of commits into clean testing environments.

What is interesting about CI is that if you were ignorant to the fact that this is even a practice, your development shop, if it is results-oriented, will naturally land there. Going after faster releases with greater quality can only be done with highly effective team communication, automation and early detection of bugs. Thus any push in the direction of modern development will have an outcome that contains all the building blocks of CI.

Continuous integration allows you to look at the application as the entire stack. This includes the VM/container, OS, configuration, web server, backend, and frontend. Driven by orchestration scripts, the entire application and infrastructure with code is laid out in front of each build; therefore you are testing against the entire stack, also called end-to-end testing, not just the application. This helps identify issues in the orchestration and deployment process itself.

CI facilitates faster releases without having a negative impact on software quality and efficiency. It helps meet business goals such as faster-moving roadmaps, increasing customer demand and sensitivity, and applications that are ever-growing in complexity. CI also has the residual benefit of enabling more exploratory functional testing, and therefore helps businesses stay ahead of competitors.

Challenges and Solutions

Despite these benefits, there are clear and present obstacles to the adoption of CI in organizations, including the following:

1. **Test strategy:** The biggest mistake an organization can make is to begin the CI process without a strategy. The speed at which CI should move, and the increase in commits and pushes to CI environments can get confusing if it is not backed by a clear results-oriented strategy.

    a) What should QA and developers be focusing on?
    b) What should they test?
    c) When should they test?
    d) When should automated tests be run?

    It is easy to forget about the business drivers and focus only on methods for utilizing CI.

2. **Team culture:** The most common obstacle to CI is the change in habits, workflow, and team culture that come with its adoption. An easy way to get started is to start small. By taking on a small automation project that is an easy win, the organization can catch the automation "bug", and move more easily to CI. It builds confidence in the process, gives a taste of the value, and drives enthusiasm amongst the entire team, not just a small portion for the broader solution.

3. **Complex infrastructure:** Due to the sheer complexity of managing a test grid in-house, organizations often resolve to perform testing on only the latest and most common browser or operating systems. This eventually backfires. The smarter option is to avoid this administration all together. It is not one or the other. You could instead leverage a cloud-based functional test solution that has a comprehensive grid and a flexible infrastructure. Human error or pressing schedules will result in dated functional test grids 9 out of 10 times. For every new version of a browser, device, or OS, your IT team needs to make that available

in the CI environment. (Just because they can do it, does not mean they should.) Thus, keeping backend and unit testing in-house or IaaS, and off-loading the functional test grid to a testing PaaS is a good combination.

4. **Buggy functional tests:** If an aging test suite is in place, more test failures are likely to occur. This is particularly true in the case of functional testing, which often involves long, automated tests. The biggest loss is the time cost associated with failures. This can be addressed by making infrastructure flexible, creating small, atomic tests when and where possible and having enough CI environments to test in parallel.

5. **Test automation is slow:** The test suite should run quickly if it's to be run frequently. This can be achieved using an infrastructure tool that allows parallel testing. Parallel tests of separate test cases, or even the same test run simultaneously on different environments, can drastically improve the speed of test automation. This way, even if one test run crashes, it can continue to run somewhere else. This helps to avoid re-runs all together, and speeds up the entire suite dramatically.

Organizations that are aware of these challenges and solutions are better prepared to make the transition to a DevOps culture.

4 Making CI Work

Implementing CI is a three part process. It starts with process design, followed by QA strategy, and finally, tool implementation. Many organizations may find that they already have the tools, and only the processes have to be revisited. For years organizations have been using release automation in the practices created by ITIL and ALM. However, CI is unique in that CI as described in this document can slip into existing environments without negatively impacting or going through a complete overhaul of existing teams.

Designing the CI process is not determined top-down. It is a collaboration from several groups that includes R&D Management, Development, Release Management, QA, and DevOps. Because CI is the connection to the first half and the last quarter of the software delivery chain, it cannot live in a silo. Handoffs between each portion of the chain need to be fluid and seamless.

This starts with the developers. Developers first have to be open to the idea of working in a way that facilitates a CI environment. Considerations such as how often they should commit code, what their engagement level with the CI environment is, and what they are responsible for should be taken into account before moving forward.

Next is IT. IT often will deliver the infrastructure required for CI, or become facilitators for some portion of its setup. Because of their involvement, they also need to understand its value. They need to move quickly. Moving quickly is only possible by allowing developers and QA to provision and deprovision infrastructure as needed.

QA usually takes the majority of ownership of the CI environment. However, this is highly dependent on the organization's definition of what QA is. In some organizations it is not quality assurance, but quality engineering (QE), which has a greater technical focus and strategy. In others, QA is executed by non-technical manual testers. In this case, the CI environment would most likely be controlled by IT operations or the developers themselves.

This is in reference to QA/QE in the context of a DevOps mindset which is automation and technology-focused. In this scenario, the QA personnel are mostly working on testing strategy and test case design. They are not running tests themselves.

In the DevOps mindset, the QA/QE team will be very active in the orchestration, utilization, and maintenance of CI environments. If they are leveraging a cloud-based testing tool, they will likely fully own its procurement and setup. They are also the gatekeepers of what is tested, how it is tested, and the creation and maintenance of the entire test suite.

In a world where application volume and complexity are increasing, there is a lot of pressure on teams to test more often and catch more bugs, faster. Because of this pressure, they have a lot to gain from robust CI processes and tools. More specifically, a comprehensive and flexible testing grid is of the utmost value.

Finally, there are automated processes and individuals that make some declaration about the quality of a release in a CI environment. This decision will ultimately determine if it reverts or moves on to the delivery stages of the development chain.

The key elements to CI's success include both automation and flexible infrastructure. Let's take a deeper look into these key components.

5 A Strong QA/QE Strategy

Many believe that CI without automated, continuous testing is not CI. In truth, without automated tests, it is difficult for developers and other project stakeholders to have confidence in software changes. But with automation comes a new set of challenges that need to be addressed, as the complexity of automated systems can increase at an unmanageable rate.

There are three types of tests that are prime candidates for automation — unit tests, integration tests, and functional tests.

Automating Unit, Integration, and Functional Tests

If we are to build software systems that are truly reliable, we have to ensure reliability at the object level, which can only be achieved through successful unit testing. Unit testing is the process of testing individual units of source code to determine whether they are fit for use.

Unit tests can be created and run early in the development cycle (i.e., day one, usually) by the developer who writes the feature test before the feature itself (test-driven development). Because of the rapid time between coding and testing the results, unit tests are an efficient means of debugging. They should be run each time someone checks in code. This process is called the commit build. There is little configuration cost, and the resource cost to run them is negligible.

Integration tests verify portions of a system. They may require a fully installed system or some external dependencies such as databases, file systems, or network endpoints, to name a few. These tests verify that components interact to produce the expected aggregate behavior of the application. Integration tests exercise the business layer under the application web pages. Because larger amounts of code are exercised by each test case, more code coverage is obtained per test; therefore, these tests tend to run longer than unit tests. As such, they should be run as part of secondary builds or on a set periodic basis.

Finally, there is functional testing. Older definitions of CI ignored functional testing due to the manual nature of this form of testing at that time. But today, functional testing makes up the majority of the overall implementation effort of CI, and ultimately contributes the most to its value.

Functional testing is when the application is tested from a user's perspective. It includes all typical user flows. Each one is called a test case, and testing runs include the entire test suite. To ensure that the application behaves well under a wide range of scenarios, functional tests should be run against the most comprehensive grid of operating systems and browser types with multiple versions of those browsers.

Functional tests can be done manually, but in a modern software delivery pipeline they should be executed using automation tools such as Selenium for scripting, and appropriate web drivers to run the scripts. This is the point at in which automated testing dramatically increases in complexity. For this reason, many organizations are forced to limit functional testing to regression testing on basic features only, and only test on a limited matrix of devices, browsers, operating systems, etc.

Flexible Infrastructure

Infrastructure is not what defines CI. Often, depending on the application, there can be multiple CI environments. For example, functional testing can be parallelized by having multiple environments per test on a specific OS/browser combination. Collectively, the process, tools, and environments are considered continuous integration. When thinking of the ideal infrastructure for CI, the concept of 'flexible infrastructure' is most relevant.

Flexible infrastructure means containers or VMs are used, and can be spun up and down rapidly, and do not become the bottleneck or source of contamination for testing processes. This matters because infrastructure has a bearing on the type of tests, their complexity, the number of test runs, and who has access to the environments. Any hindrance to these activities will result in huge limitations to the possibilities of CI.

Some organizations set up their own CI labs, preferring to have maximum control over the environments. However, they often end up using the same infrastructure and the same host OS for each deployment. This can cause a buildup of system variables that will impact testing results, as well as the reliability of the CI environment and process. In an ideal scenario, infrastructure used to deploy applications and run tests is brand new in each and every build. The use of virtualization or containers alongside robust configuration management tools will help ensure that each run is performed on pristine VMs. While CI labs are tedious to maintain in-house, QA/QE-tailored cloud services make it easy to automatically provision and deprovision infrastructure based on the volume and types of tests.

The biggest advantage of adopting CI labs with a flexible infrastructure is that everyone can participate in quality. Once the application is deployed, it is run through some or all of the unit, integration, and functional tests. The environments where this automation is run frequently are then made available for quick manual tests by developers and QA/QE. Easily accessible environments with the entire stack make it easy for front end developers, back end developers, and QA/QE to quickly review application functionality and quality. It also allows QA/QE teams to focus their efforts on automation rather than running tests. This shift brings broader testing coverage that includes bringing exploratory testing into the environment faster.

CI introduces the idea that quality starts with developers and exists in every step in the delivery chain. It illustrates the notion that quality is everyone's responsibility. When CI is in place, developers suddenly have an interest in seeing their code in action with all other components. It also allows them to spend more time on feature creation, and less time chasing down bugs reported in production. This can be a huge motivator for developers.

6 Beyond Regression

If and when an organization adopts CI, and the net result is a modern and faster version of what has always been done, something has gone wrong. CI should extend test coverage and become a central point for testing the entire stack. It allows teams to expand what they test, not just to look at existing functionality, but what could be.

Organizations that want to test functionality in ways previously thought impossible would leverage service virtualization. This method emulates the behavior of specific components that are unavailable, yet are required for certain integration and functional tests, such as a backend, or sample customer data. For example, functionality that requires testing live data and user interaction with that data is made possible using service virtualization.

Similarly, a mock environment is another type of service virtualization that emulates not just certain components, but the entire back end. This method helps to test user-facing features using the front end, as automated tests can run extremely fast, allowing testing of all functionality of the application very early on.

Along the same lines, there are elements of visual testing, such as unmanned recorded test videos, which are made possible with certain cloud tools. These videos can be flagged when the test result returns a bug, and the bug can be automatically logged in the ticketing system with a direct link to the video or screenshots. Visual testing makes it possible to compare screenshots between the current version and previous versions of the application, giving 100% test coverage on single page functionality. (This is called perceptual diffs.) It can't be used to test user flows, but it tests functionality for which one might never consider writing a test case.

If back end functionality is combined with service virtualization, then later in the testing lifecycle, teams have an environment to automate API testing as well. API testing has traditionally been difficult to implement as it requires near-real data and strong environments with which it cross-communicates.

After feature creation, CI is the most opportunistic element of software development. It empowers testing teams to go beyond the standard test suite of typical, common functionality, and grows their test cases to include very recent functionality, as well as exploratory testing. This is a process of simultaneous testing, and test design, testing that evolves with the application, and results in expanded test coverage. It also gives the opportunity for developers to implement features they've always wanted, but have been afraid to try due to the risk of failure. Now developers can quickly try functionality against the entire stack without worrying about it slowing down releases, or creating issues that impact production.

7 Introducing CI to Mobile

Most people associate CI with web applications. As the world becomes increasingly "mobile first", the same practices we use to accelerate web apps must be used for mobile applications as well. And all the benefits of CI extend to mobile application development, with some additional considerations:

1. **App stores:** The approval process of getting code to a device involves many steps and approvals that are not in the control of developers.
2. **More complex grid:** The testing matrix of devices, operating systems, and browsers is many times larger.
3. **Workflow:** Builds and deploys are handled differently.
4. **Intimacy of devices:** Testing applications is more personal and less of a team task.

However, there are some things that do not change. For example, if a web application has a mobile version, then some of that functionality can be tested in a traditional CI environment. On the other hand, native mobile applications need to be tested on devices or simulators/emulators. Also, like web apps, the back end portion of code makes up a large part of most mobile applications as well, which means unit and integration tests are still relevant and run separately, but nearly synchronously.

Where web and mobile applications are just variants of each other, one of the worst things that can happen is a complete disconnect between each other's testing. If there is a considerable rift between the two, the teams handling each type of testing will not learn from each other or benefit from test cases that span both application types. This eventually means that the quality of the application on different platforms would widely vary. And because users often leverage both, it could create frustration across the user base.

This is why the tools for integration environments are so important. Ideally, the tool responsible for the functional testing grid and the infrastructure for both mobile and web applications should be the same. This is especially true because of the sheer size of a mobile testing matrix; rather than relying on the testing team, it is beneficial to use a cloud service hosted by a vendor that manages and maintains the simulators/emulators and/or physical devices.

8 Conclusion

Key Takeaways

1. Continuous integration can and should be part of every DevOps shop, irrespective of size or scale.
2. Continuous Integration has to be driven by a strong QA/QE strategy.
3. Well implemented CI opens the doors for wider test coverage and more exploratory testing.
4. Continuous Integration can and should be used in mobile development.

Based on these key findings, it is clear that DevOps adoption is crippled without CI.

As organizations make the move to development with greater automation, they cannot jump immediately into delivery and deployment practices. CI is on the path towards DevOps, and is a necessary component that must be executed correctly before any other aspect of DevOps can be implemented.

9 About Chris Riley

Chris Riley is a bad coder turned market analyst. After he founded his IT firm in 2000, he began to analyze that gap between technologies and their adoption. He is a technologist who has spent 15 years helping organizations transition from traditional development practices to a modern set of culture, processes and tooling. In addition to being a Gigaom Research analyst, he is an O'Reilly author, regular speaker, and subject matter expert in the areas of DevOps strategy and culture and Enterprise content management. Chris believes the biggest challenges in the tech market are not tools, but rather people and planning.

Throughout Chris' career he has crossed the roles of marketing, product management, and engineering to gain a unique perspective on how the deeply technical is used to solve real-world problems. By working with both early adopters and late, he has watched technologies mature from rough solutions to essential and transparent. In addition to spending his time understanding the market, he helps ISVs selling B2D and practitioners of DevOps strategy. He is interested in machine-learning, and the intersection of Big Data and Information Management.

10 About Gigaom

Gigaom Research gives you insider access to expert industry insights on emerging markets. Focused on delivering highly relevant and timely research to the people who need it most, our analysis, reports, and original research come from the most respected voices in the industry. Whether you're beginning to learn about a new market or are an industry insider, Gigaom Research addresses the need for relevant, illuminating insights into the industry's most dynamic markets.

Visit us at: research.gigaom.com.