technologyforecast

A quarterly journal

2013
Issue 2

DevOps: Solving the engineering productivity challenge

John Esser
Director of Engineering Productivity and Agile Development, Ancestry.com

pwc

Making DevOps and continuous delivery a reality

Automation, thoughtful choices, and a well-designed workflow are pivotal to making good progress.

By Robert L. Scheier and Alan Morrison

When Chad DePue, founder of Inaka, launched his 30-person software development and hosting company, he was determined to apply the lessons he had learned from standard enterprise software development practices—lessons about what to avoid.

During an earlier period as vice president of product and program management at a secure e-mail and workflow vendor, DePue observed the "tedious, laborious, and slow" way that most enterprises roll out technology infrastructure and applications. For example, stacking hardware in a company-owned data center and plodding through ponderous cycles of as long as one year to develop, test, and deploy each major code release.

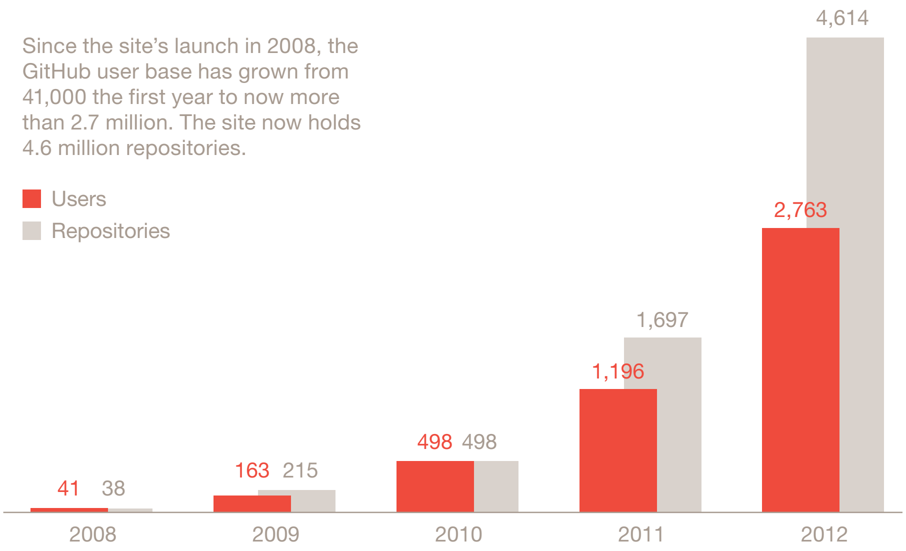
"That's not an environment I find very exciting as a developer," he says. It certainly does not attract today's most skilled and innovative developers, whose work is in high demand by customers. Nor is it conducive to the market demands on Inaka.

At four-year-old Inaka, based in Buenos Aires, DePue is creating what PwC calls a more "antifragile" enterprise, an organization that improves and gets stronger as a result of unpredictable change and environmental volatility.

A DevOps, continuous delivery approach provides a stepping stone to that antifragile state. At one level, DevOps is a working style designed to encourage closer collaboration between developers and operations people: Dev+Ops=DevOps. Historically those groups have been working more or less at cross-purposes. DevOps collaboration seeks to reduce the friction between the two groups by addressing the root causes of the friction, making it possible to increase the volume and flow of production code, to improve the efficiency of developer teams, and to reduce the alienation of the ops people who must guard the stability of the system.

One of those root causes of friction is poor workflow design. DevOps encourages extensive automation and workflow redesign so developers can release small bits of code frequently (in a more or less continuous delivery cycle) and yet not disrupt the operational environment in doing so. The workflow includes buffers, compartmentalization, and extensive monitoring and testing—a very extensive and well-designed

Figure 1: GitHub growth, 2008–2012


Since the site's launch in 2008, the GitHub user base has grown from 41,000 the first year to now more than 2.7 million. The site now holds 4.6 million repositories.

Source: GitHub, 2013

DevOps encourages extensive automation and workflow redesign so developers can release small bits of code frequently, and yet not disrupt the operational environment in doing so.

pipeline, but also a rich feedback loop. It's a very test-driven environment. When the small bits of code get deployed, the individual changes to the user experience tend to be minor or limited to a small subaudience initially.

Inaka's cloud-based service offerings are never finished. In the cycle of collaborative development, deployment, testing, and revision, the services are repeatedly reinvented through ongoing and fast-paced experimentation, deployment, and tweaking. The services are more stable because of the pace of that reinvention. When something breaks, it is fixed within a short period of time, and when something is improved, that improvement arrives quickly.

To match this continuous delivery style of today's best developers and to meet ever-changing market demands, DePue relied heavily on open source software. "Inaka's main goal is to be the new outsourcing stack," he says.

It might be tempting to dismiss Inaka as a greenfield example, but the demands on it are similar to those on larger, older, and more traditional businesses: fast-changing technology and customer requirements, plus unrelenting pressure to cut costs and speed delivery. Although it does not face the pressures of operating in a heavily regulated industry, Inaka is a serious competitor in its field. Clients range from web startups to Fox News Network and Viacom, VH1, and MTV. One app it developed and hosts, Whisper (a pseudo-anonymous social network for college students), made the top 100 in the Apple App Store.

Startups and established enterprises alike are using open source software to become antifragile organizations capable of meeting unpredictable business needs more quickly and inexpensively through continuous delivery supported by the tools described in this article.

Tools for the open source IT stack

From the smallest social media startup to the oldest traditional financial exchange or manufacturer, the business environment is changing far too quickly for the software deployment cycles of the past to be effective. When a new mobile or social platform surges into popularity, or a natural resources boom creates an emerging market, enterprises must quickly tap that potential with new products and applications.

Increasingly, these enterprises will need to become more antifragile, as explained in the article, "The evolution from lean and agile to antifragile," on page 06. For the IT department, that means adopting a continuous delivery software stack to enable rapid, collaborative, and massively scalable application deployment, testing, and monitoring in an ongoing and increasingly automated process.

Neither open source approaches nor agile development methods are new. But standalone open source tools and entire frameworks for application development and deployment are new. Also new is the extent to which these tools are being adopted along with the related process of agile development to bring more speed, flexibility, and collaboration to enterprise IT.

For example, GitHub (the commercial version of the open source Git, a version control system) now has more than 2.7 million users and 4.6 million repositories. (See Figure 1.) More details about Git and GitHub can be found later in this article in the section "Requirement 2: Collaborative, iterative development."

In these continuous delivery environments, the organization's processes and tools must support a deploy-and-fail-fast mindset that constantly reacts to market changes. The tools in this stack must do the following:

- Provide not just point capabilities but frameworks that encompass multiple parts of the continuous delivery cycle

- Interact and share information to foster real-time responsiveness and collaboration across business and geographic boundaries

- Automate manual functions wherever possible to increase speed and reduce costs

- Support rapid, iterative, and continuous development, deployment, and testing

In this "new age" IT, what previously was a physical infrastructure is more a virtual entity to be automatically deployed, tested, reconfigured, and reused through scripts. The traditional long, complex sequence of gathering requirements, developing code, testing, deployment, and monitoring becomes a collaborative, iterative, and never-ending process.

Commercial and open source vendors offer tools to automate, integrate, and increase the speed of each function. However, the market is young, and many of these tools are still immature. Veterans warn that IT organizations may need to write some of their

From the smallest social media startup to the oldest traditional financial exchange or manufacturer, the business environment is changing far too quickly for the software deployment cycles of the past to be effective.

own tools or adopt different tools for different needs. More cautious organizations may also want to keep some work manual until they fully trust automation or can see a suitable return on their development effort.

The rest of this article examines the main requirements for continuous delivery and some of the tools available.

Commercial and open source vendors offer tools to automate, integrate, and increase the speed of each function. However, the market is young, and many of these tools are still immature.

Requirement 1: Become antifragile

Conventional planning for robust systems usually involves avoiding equipment failure at all cost, buying the most expensive (and supposedly most reliable) components, and clustering them so a backup can take over for a failed component. An antifragile approach requires a continuous delivery software stack that turns these traditional processes upside down.

As practiced in the open source community, antifragile thinking assumes that systems comprise large numbers of lower-cost commodity hardware nodes, some of which are bound to fail unpredictably. An antifragile approach also can involve deliberately causing unpredictable failures in components within a cloud infrastructure to detect and remove vulnerabilities.

Some of these failure-causing tools, once used only by cloud-based service providers, are now available as open source software. They include Netflix Chaos Monkey and Stelligent Havoc, which randomly shut down Amazon EC2 instances to help developers plan for such disruptions and fix weak points in their infrastructures.

To cope with the expected failures these tools uncover, organizations can use clustering and other traditional measures. But organizations also must use newer techniques, such as:

- Providing a scaled-down but adequate version of a service in case of a component failure. For example, if an e-commerce site suddenly lacks the compute power to provide personalized choices for each customer, it could still provide a list of the most popular choices for all customers that require less processing.
- Removing as many dependencies as possible so the failure of one component has the smallest possible impact.
- Creating workarounds that keep the enterprise operating even if services once thought essential are no longer available. These workarounds include ensuring that traffic load balancers correctly detect and route requests around instances that go offline, reliably rebuilding instances, and ensuring that patches made to one instance are committed to the source repository. Continuous delivery often allows rapid fixes.

Requirement 2: Collaborative, iterative development

Web-based, open source version control has transformed what once was the routine housekeeping function of storing code under development. Such version control is now the center of dynamic and creative development environments. Traditional source code repositories commonly stored work from internal developers who wrote proprietary apps using commercial integrated development environments. Today's web-based repositories let developers around the world develop and share multiple branches of code, reuse code, and easily combine branches. They also make it easier for developers to share code libraries for commonly used functions, reducing the cost and time required to bring new services to market.

GitHub, the commercial version of the open source Git developed by Linux spearhead Linus Torvalds, is one of the hottest open source version control systems. Part of its appeal is its social features, which promote collaboration, and with more than

3.5 million users, GitHub is gaining share on Apache Subversion, a more centralized version control system.

“This is the language of open source now, and it’s all about source control and how you manage changes to code,” says Charles Oppenheimer, founder of Prizzm, which has developed the MightBuy shopping app. “If you aren’t coding this way, you aren’t speaking the new language. I don’t hire anybody who doesn’t have GitHub contributions, so I can see their code.”

GitHub Inc. has launched GitHub Enterprise, a version of Git for an organization’s internal network, which is protected by the customer’s existing authentication. Stash from Atlassian also provides behind-the-firewall Git repository management for enterprise teams. Microsoft has announced support for Git in its own Team Foundation Server (TFS) version control system and in its Visual Studio development tool.

Requirement 3: Rapid coding

Speed and simplicity are essential when business models and delivery channels (such as mobile) change constantly. This necessity requires programming models that expect data to be presented on different devices and that promote code reuse across platforms.

One of the most popular is JavaScript. Its support for user-defined functions allows the development of a library of reusable code that can be pasted into future applications and called through a statement. Rated most popular in a first quarter 2013 ranking of development languages according to activity on GitHub and the Stack Overflow developer discussion site, JavaScript is widely used to speed the performance of web pages and to embed capabilities that make websites more interactive and compelling.[1]

[1] Stephen O’Grady, “The RedMonk Programming Language Rankings: January 2013,” February 28, 2013, http://redmonk.com/sogrady/2013/02/28/language-rankings-1-13/#ixzz2UFTwNmli, accessed May 24, 2013.

Today's web-based repositories let developers around the world develop and share multiple branches of code, reuse code, and easily combine branches.

A popular tool that complements JavaScript is Joyent's Node.js, which is server-side software used to create scalable Internet applications, minimize overhead, and maximize scalability while allowing developers to create the server and client side of an application using JavaScript. A major retailer, for example, chose Node.js for mobile application development because of its scalability and ability to communicate with other services, such as the retailer's application programming interface (API) and database.

Another challenge is linking modern web-based or mobile applications to existing databases, transaction processing systems, and other legacy services. To ease such connections, eBay released ql.io, a data-retrieval and aggregation gateway for HTTP APIs.

Apache Cordova (formerly PhoneGap) is a set of JavaScript libraries that developers can invoke to run device-specific native code for each platform and utilize native device capabilities such as the accelerometer, camera, or contact list.

When developers combine Cordova with a user interface framework, such as jQuery Mobile, Dojo Mobile, or Sencha Touch, developers can use HTML, CSS, JavaScript, or other familiar web languages to create apps for multiple platforms, rather than using separate native code for each platform. Cordova also bundles the application assets into a package ready for distribution through each platform's app store. Disney World, Stanford University, and SlideShare are among the many organizations that have used the HTML5-based jQuery Mobile to build mobile applications to run on multiple platforms.

The open source IT stack must also be built for high performance and scalability. Erlang, an open source programming language whose runtime system supports concurrency, distribution, and fault tolerance, is designed for massively scalable real-time systems that have high-availability requirements. It is used in telecom, messaging, banking, finance, gaming, and embedded systems.

Requirement 4: Automated, rapid configuration and deployment

Users chronically complain about IT's inability to rapidly configure servers, applications, and other resources. For that reason, the ability to instantly—and even automatically—create infrastructures from scripts is one of the key goals of continuous delivery.

"Dependency management is a very hard problem to solve," says a software development manager at a major smartphone handset vendor. "If we change one component or the internals of one of the components, we need to figure out the effect that would have on every other component."

Rather than rely on complex dependency management software, his organization uses a manual process of releasing one changed component at a time to a "near live" environment and then putting it in production only if it passes rigorous tests. He trusts this manual process, discipline, and "the fact that our developers know what they are doing." For a larger organization that has more services to track, he says, a deployment tool probably would make better sense. These tools now are more common and more advanced than when he began the move toward continuous delivery, he notes.

Open source automated configuration tools, such as Puppet and Chef, replace older, slower methods, such as structured scripting and image cloning. Rather than rely on "golden images," these tools allow organizations to express their infrastructure as code and to share configurations through GitHub or other version control systems.

Being first to market, Puppet has wider platform support and more users than Chef. System administrators often find Puppet's model-driven approach easier to use because it relies on data structures created with the JSON (JavaScript Object Notation) data interchange format. Chef's procedural approach, which requires writing configuration "recipes" in Ruby, often makes it a better match for developers.

Inaka uses Puppet and Chef to automatically and quickly provision servers "depending on the application's needs," DePue says. "The closest we get to a physical server are companies like Liquid Web," which custom provisions servers similar to Amazon Web Services servers so DePue's customers get dedicated higher-performance hardware at the price of lower-performing public-cloud multitenant servers.

The need to train users in Chef and to maintain those skills over time can be a risk, but the tradeoff is increased power and flexibility through the use of its command-line interface, called Knife. At first glance, Puppet's model-driven approach means less fine-grained configuration control, but the latest version also allows developers to craft configurations using Ruby.

John Esser, director of engineering productivity and agile development at Ancestry.com, has been using Chef in a three-year effort to move the online research site toward agile development and more frequent code deployments. Before Chef, he says, "even replicating a given server's configuration was very challenging, if not impossible."

He likes Chef because of its Windows support and its use of Ruby rather than a customer-defined domain-specific language. He can easily create code to handle unique configuration requirements.

Users chronically complain about IT's inability to rapidly configure servers, applications, and other resources. For that reason, the ability to instantly—and even automatically—create infrastructures from scripts is one of the key goals of continuous delivery.

Along with Chef, Esser is using Go, an agile release management tool from ThoughtWorks Studios, as a "continuous delivery dashboard that indicates the status of development and lets the team push the button when things are deemed ready" to deploy. These tools help his developers and operations staff to leave their traditional sequential approach and adopt continuous fail-fast deployment, enforcing standards "through tooling and automation," he says.

Niek Bartholomeus, an independent software architect in Antwerp who recently helped a European bank move toward more frequent delivery, says Chef and Puppet "are declarative in the sense that you give them a desired end state and they will make sure the infrastructure is aligned properly."

He likes the idea of using such tools to automate the provisioning of operating systems and middleware. For the bank, however, he chose StreamStep's SmartRelease (since acquired by BMC Software and renamed Release Process Management) because it requires either notification of a staff member or a custom-developed script before deployment. "Chef or Puppet would be too big of a leap in one step" from the bank's formerly highly manual application deployment processes, he says.

Others have chosen configuration tools from major vendors. For example,Vijay Gopal, vice president and enterprise chief architect at Nationwide Insurance, uses a configuration platform from Hewlett-Packard.

When speed is imperative, testing is a necessary evil—but a very necessary evil. In a 24/7 social world, and especially for web-facing applications, a slow or failed site or application can sink a business.

Requirement 5: Ongoing, automated, and proactive testing

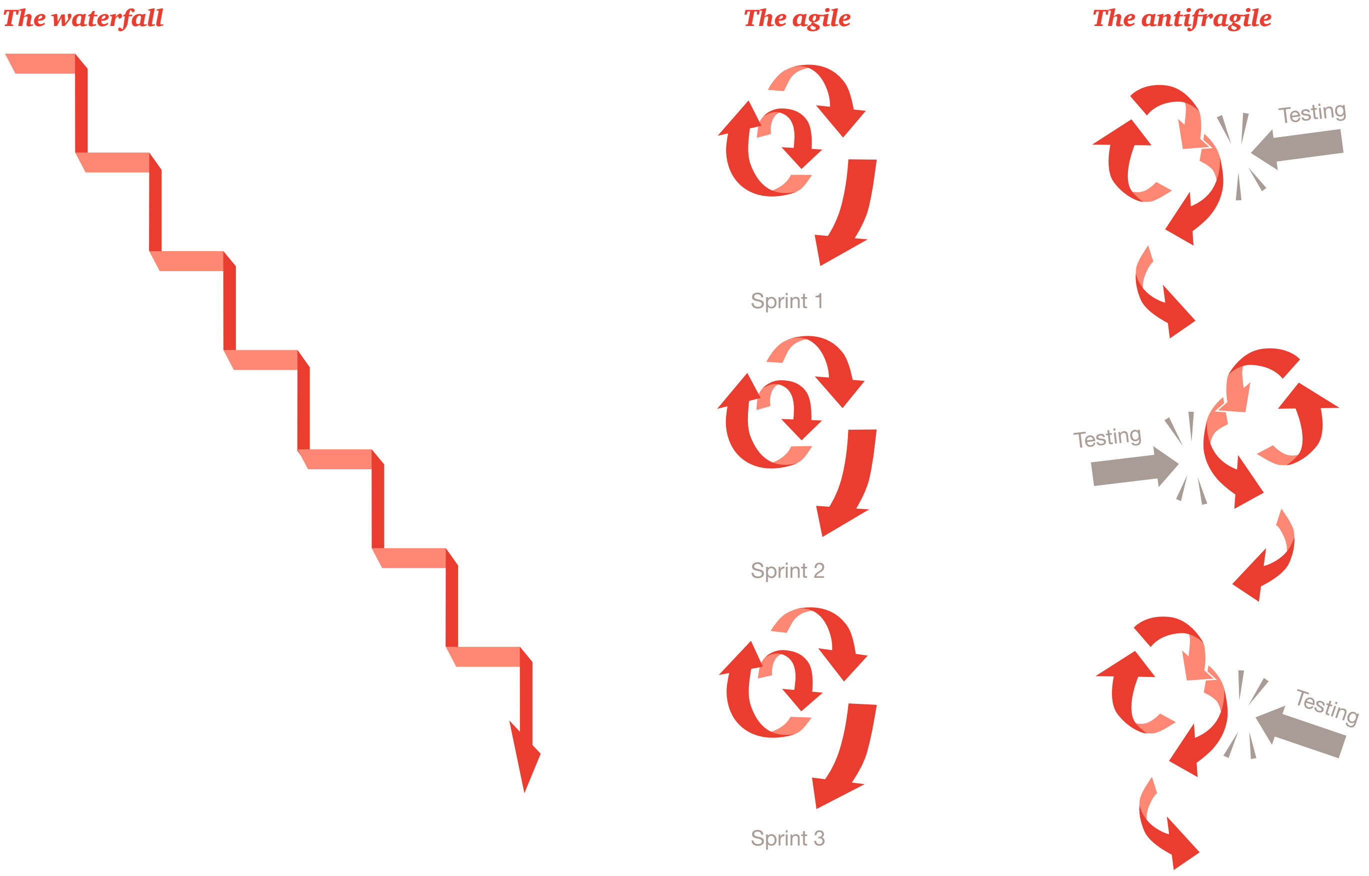
When speed is imperative, testing is a necessary evil—but a very necessary evil. In a 24/7 social world, and especially for web-facing applications, a slow or failed site or application can sink a business. The challenge is to repeatedly and thoroughly ensure that code and the entire IT infrastructure can meet surges in demand without delaying time to market. This goal requires automated testing of everything—from the creation of the infrastructure through unit and integration testing.

Continuous delivery tools aim to meet this challenge not only by automating much of the test process, but also by turning conventional test methodologies on their head. Rather than making test a separate and lengthy sequence in the larger deployment process, continuous delivery practitioners roll out small upgrades almost constantly, measure their performance, and quickly roll them back as needed.

Automated testing tools include the Ruby-based Cucumber, which allows development teams to describe the desired behavior of an application in a business-readable, domain-specific language. For unit testing, QUnit is a framework for testing any generic JavaScript code. JUnit is a framework for writing repeatable tests for Java applications.

Twist, a tool from ThoughtWorks Studios, assists development teams in creating and maintaining test suites that serve as a bridge from manual to automated testing. CruiseControl is also used as a continuous integration tool and extensible framework to integrate the continuous building and testing of software projects.

Figure 2: Ongoing testing and improvement anticipate and encourage antifragility.



The waterfall

The agile

The antifragile

Sources: David S. Maibor, "The DOD Life Cycle Model," Christine Anderson, Ed., *Aerospace Software Engineering* (Washington, DC: American Institute of Aeronautics and Astronautics, 1991), 33–49. Michael Reich, "Agile vs. Waterfall: How to Approach Your Web Development Project," September 10, 2012, http://www.commonplaces.com/blog/agile-vs-waterfall-how-approach-your-web-development-project, accessed May 14, 2013.

Inaka uses the Keep It Functional (KIF) open source test framework from Square for iOS apps, JUnit for testing Android apps, and Erlang's Common Test framework built into Ruby.

At Ancestry.com, Esser avoids automated test tools, especially those that record and play back interactions with an application. He sees them as "not conducive to agile." Instead, he uses test engineers that "move from team to team to help the developers build and own their testing."

Ancestry is just one of several companies interviewed who are moving beyond agile by placing more emphasis on a test-driven, continuous change environment. (See Figure 2.)

Requirement 6: Real-time monitoring and performance management

Through monitoring and performance management, administrators receive and act on real-time feedback to be sure each new function works properly and to release new code that fixes anything from performance to usability issues.

Some deployment tools, such as Jenkins, use plug-ins to monitor the executions of repeated jobs—for example, building a software project or jobs run by cron and other schedulers. Jenkins can also monitor the execution of job runs on remote machines, capturing their output for analysis by administrators.

Some corporate cultures may not be comfortable immediately abandoning long-standing habits of separate requirements, development, testing, and deployment cycles.

Boundary, an application performance-monitoring provider, uses a software-as-a-service (SaaS) delivery model. Other commercial tools include Nagios and Librato, Amazon CloudWatch (provided for and by the AWS environment), and Splunk. Open source tools include Graphite, Kibana, logstash, and more.

Says Inaka's DePue, "For the physical monitoring—'Are the servers OK?'—we're using the Nagios monitoring suite," which provides basic data for all servers on an in-house dashboard. To track the "missing piece" of server and network performance across the cloud, Inaka uses Boundary.

A real-time dashboard and visualization capabilities drew DePue to Boundary. "We rolled out our system for a client using a popular open source database, but we had some real scalability problems with the database and couldn't figure out what was going on. We finally took some screen shots of the graph of network usage and immediately saw that our network interface cards couldn't handle the load, as traffic across the NIC would collapse and then slowly recover," DePue says.

When Boundary revealed that some programming Inaka hosted for a customer was running up excessive network charges, DePue moved it to CloudFlare, a content delivery network. CloudFlare routes customer traffic through its global network, optimizing delivery for the fastest load times and best performance while blocking threats. While the basic CloudFlare services are free, additional offerings—such as faster performance, 100 percent uptime guarantees, and advanced security protection—incur a fee.

"CloudFlare is like an insurance policy, where you pay them only if somebody attacks your service or you suddenly need more capacity," DePue says. By dynamically assigning more servers to counter the attack, "they help you weather the storm. Ten years ago, you would have needed to buy a bunch of extra hardware for that protection."

Splunk complements Boundary's network monitoring SaaS application, says Cliff Moon, Boundary's founder and CTO: "Splunk focuses heavily on log-based data and has a great analysis engine."

Esser of Ancestry.com notes, "We view monitoring at two levels: the hardware/virtualization system level, such as CPU, memory, throughput, etc., and the business cluster level—that is, each key service is in a cluster that must meet a service level agreement. At the hardware/virtualization level, we use the monitoring tools provided by Microsoft HyperV. At the business cluster level, we have a custom-built solution that integrates and monitors each business level cluster."

Unified tools for unified work

Properly used, the new generation of automation and scripting tools described in this article are turning app development, deployment, testing, and management from a series of discrete, cumbersome events into a nimble recurring process that speeds business agility and success in fast-changing markets—and that supports more antifragile organizations.

However, the market is still young. "All the capabilities needed for a continuous delivery stack are largely immature," Esser warns. "You have a few gems like Chef, Go, Puppet, or others, but organizations are left to build their own solutions or piece together various tools." The choice of tool also depends on scale. "Large websites like Ancestry.com, Amazon, and Netflix employ scaling techniques different from those used by smaller organizations," he says. "It also depends on your core technology. Linux is more mature and offers more open source solutions, while Windows is a tough road."

Monitoring is "still a very thorny problem," says Jez Humble, a principal at ThoughtWorks Studios. "A lot of companies have built their own stuff to do this. We're seeing a whole new area [of] application release automation emerge," Humble says, but most of the tools work only in greenfield environments—not with the legacy applications that exist in most established organizations.

The best tools mirror the ongoing and parallel development, test, and deployment activities at the core of continuous delivery. "Look for something that integrates the information that comes from various functions, such as change management, version control, issue tracking, and monitoring, and that can be used by developers, operators, testers, and others," Bartholomeus says. "For me, that's an ideal start for one team to get insight in the other team's work and to help bridge the gap between them."

Conclusion: Start now

Whatever an enterprise's size, age, or market, today's ever-changing business requirements demand a continuous delivery software stack that enables rapid, collaborative, and massively scalable application deployment. The organization's tools and processes must support a deploy-and-fail-fast antifragile mindset that constantly reacts to market changes.

Rather than wait for commercial software vendors to meet all the tool needs, open source developers are building on each other's work to repeatedly refine tools across the continuous delivery spectrum. This open source stack manages a virtual, cloud-based infrastructure that is automatically deployed, tested, reconfigured, and reused through the use of scripts.

Not every enterprise needs—or is ready—to move completely to a continuous delivery stack and a continuous delivery way of working. Some legacy apps and systems of record may not be suited to the cloud. Some highly regulated industries may not be comfortable fully automating the deployment of sensitive applications. Some corporate cultures may not be comfortable immediately abandoning long-standing habits of separate requirements, development, testing, and deployment cycles.

But the urgent business needs that drive continuous delivery—speed, quality, efficiency, agility, reuse, and low cost—are intensifying. CIOs should educate themselves and their staffs in this new IT mindset and the tool stack that enables it, so they're ready to move their organizations toward a continuous delivery culture when—not if—it becomes urgent.

The urgent business needs that drive continuous delivery—speed, quality, efficiency, agility, reuse, and low cost—are intensifying.