Continuous delivery and the world of devops

By Dave Ohara

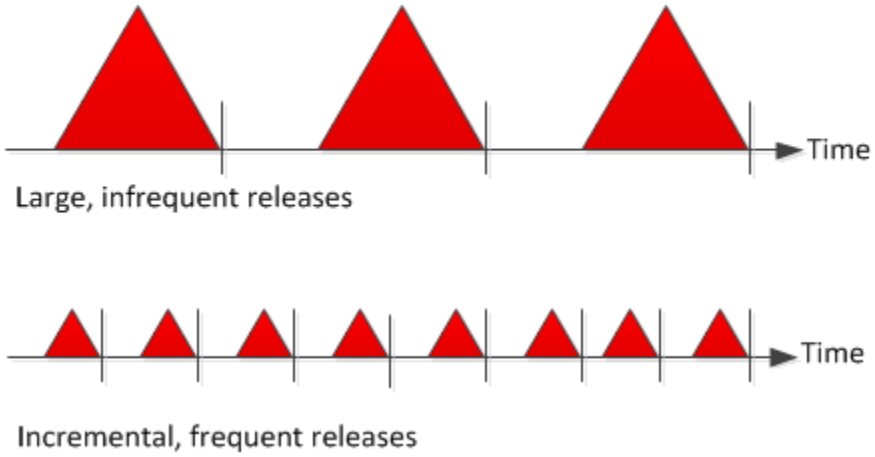
This research was underwritten by Opscode.

Table of contents

Executive summary

The advent of online businesses has created new opportunities and fierce competition. Companies want to get their products and services to market as fast as they can, and releases that occur in periods of months or years are no longer competitive. As a result, the pattern of how to release software is changing from large, infrequent releases of new software to small releases that occur very frequently, as shown in Figure 1. The ultimate goal is the continuous delivery of software updates.

Figure 1. The changing pattern of software releases



Source: GigaOM Pro

This paper explains the world of continuous delivery and its underlying philosophy, devops. Continuous delivery is an automated pipeline constructed with various technologies that allows you to ensure that your code is always ready to be released. It does not mean that you have to release every change you implement: That is a business decision. It does mean that when you choose to release, your code is ready, fully functional, and fully tested.

In conjunction with the technology is the emerging devops methodology, which is an outgrowth of the Agile movement. This movement stresses collaboration among groups that have often found themselves at odds, in particular development teams and

operations teams. This increased level of collaboration blurs the boundaries between infrastructure and code. Looking at application code and infrastructure holistically rather than as separate disciplines and treating them the same in terms of automated delivery provides compelling benefits in terms of time to market and overall stability.

Introduction

The destruction of old business models and the creation of new ones is everywhere. A good example of the transformative power of online business models is the almost complete elimination of brick-and-mortar travel agencies in less than a decade. The internet has enabled many new types of entrepreneurial activity, and as a result traditional businesses such as retailers are in stiff competition with their online counterparts. These enterprises need to implement software solutions that will make them competitive in the virtual space, and the solutions must deliver products and services quickly and allow businesses to respond to feedback in a timely manner.

One approach that addresses these challenges is continuous delivery. Continuous delivery uses the Agile development process, supported by various technologies such as continuous integration tools, to create a pattern that allows you to write and deploy high-quality software at an accelerated rate, as well as make the deployment process automated and predictable. While the traditional release cycle occurs infrequently and delivers many changes and features in each release, continuous delivery has frequent releases with smaller, incremental changes in each release.

Another hallmark of continuous delivery is that it breaks down the barriers between development and operations. Operations teams and software developers are part of the same process. Developers, testers, and systems and database managers as well as product or program managers work together on all aspects of the project. The application and the infrastructure that runs it are not treated as separate, unrelated entities. Neither can exist without the other. This philosophy of close collaboration between traditionally distinct disciplines is called devops.

With continuous delivery, application source code (traditionally created by software developers) and system configuration files and scripts (traditionally created by operations teams, or ops) are all part of the same automated process and subject to the

same quality assurance process. No change is possible without going through a trusted automated testing and deployment system.

In this paper, we will explain the components that make up a continuous-delivery system, such as a continuous-integration system, and automated tools. However, the technologies that enable continuous delivery are only part of the story. To be successful, the transition from major releases to continuous delivery must take human factors into account. It must include a move to devops.

Audience

This paper is intended for executives who determine their organization's business strategies. If you are looking for ways to reduce time to market and are considering a realignment of traditional assumptions about the roles of development and operations, you require knowledge of new tools and new approaches. This paper is meant to help decision makers understand the components that make up a continuous-delivery pipeline as well as the cultural shifts that make software engineers and operations personnel collaborators rather than adversaries. After reading this paper, you will better understand some of the potential benefits and some of the challenges of continuous delivery and the devops philosophy.

What is continuous delivery?

Continuous delivery means that software is always production-ready. Any build can be published to the production environment and it will run correctly. Developers begin writing releasable code from the first day of the project, and that code meets the requirements for code correctness and satisfies all functional tests. Continuous delivery does not dictate when to release software, as that's a business decision. It means that whenever the release date occurs, code is ready for your customers.

A continuous-delivery pipeline is a push-button system that depends on automation. In order to ensure production-ready code, you need continuous integration, automated testing, automated configuration, and automated deployment. Once someone commits a change, the automated system takes over.

An implication of continuous delivery is that you treat your infrastructure as code and it is handled no differently than an application. Just as every change to an application is immediately integrated into the build, tested, and, if it passes, deployed, so are changes to the infrastructure. For example, if a system administrator changes a configuration script, that script change is treated just as if it were an application change, including checking that change into the version-control system and running automated tests. From the viewpoint of the delivery pipeline, there is no difference between the two.

Finally, continuous delivery is not only about technology. It is about a culture of collaboration among groups that, traditionally, have often found themselves at odds with one another. Developers, testers, operations personnel, and managers are all members of the same team, and they work together to produce high-quality software that serves the needs of the customers and that is deployed in a reliable and predictable fashion.

Trends driving continuous delivery

The past decade has been marked by the emerging dominance of large consumer-facing web properties such as Amazon, Google, and Facebook. These companies have focused on delivering a wide range of applications to a large-scale audience over a broad range of devices. In order to do that, they have invested heavily in scalable, flexible infrastructures. As a consequence, the internet is no longer an exotic environment used by the technologically enlightened. Everyday activities such as shopping and socializing routinely occur online. For many consumers the virtual experience is at least as desirable as a trip to the mall.

The next step, then, is for traditional enterprises, such as banks and retailers, to increase their presence in the virtual world. Examples include a bank that wants to write a mobile application for depositing checks or a retail business that wants an online store that offers at least the same level of service as its brick-and-mortar counterpart. For these results, these businesses must begin by accomplishing what their precursors such as Amazon and Netflix have done, which is to build an infrastructure that relies on virtualization that enables dynamism and flexibility.

Here are some of the trends that are fueling the drive toward continuous delivery. Remember that these trends apply to infrastructure as well as to applications.

Traditional businesses are under pressure

Companies that have taken a standard business model, such as the retail store, and converted it to a software-based model are highly successful, and their traditional counterparts are feeling the competitive pressure. For example, Etsy is an e-commerce website that took the idea of a crafts fair and turned it into a thriving virtual marketplace. Etsy uses continuous delivery. In an interview with Jez Humble, Michael Rembetsy, Etsy's director of Operations and Engineering, estimates the company deploys 25 to 50 code changes per day.

Continuous delivery is part of an overall solution for building a responsive online presence. It allows businesses to release new features at an accelerated rate and to assure that the quality of those features is high.

Many companies that have been in the vanguard of the movement toward continuous delivery use the iterative approach. In Facebook's S1 filing, CEO Mark Zuckerberg describes what he calls "the hacker way." He writes:

> "Hackers try to build the best services over the long term by quickly releasing and learning from smaller iterations rather than trying to get everything right all at once. To support this, we have built a testing framework that at any given time can try out thousands of versions of Facebook. We have the words 'Done is better than perfect' painted on our walls to remind ourselves to always keep shipping."

Businesses want immediate value

If you have a good idea, you want to get it to market as quickly as possible. That's due not only to competitors but also to the importance of making money on your innovations as soon as possible. However, many organizations find that their software-release processes are cumbersome and ineffective. In its 2011 global CIO study called "The Essential CIO," IBM found that 50 percent of deployed applications must be rolled back and that rework accounts for more than 30 percent of costs.

Continuous delivery allows you to release new features quickly. For example, John Esser, the director of Agile Development at Ancestry.com, says in his presentation "Continuous Delivery @ Ancestry.com" that releases that once took a few weeks now take a few days and that a single new feature can go from check-in to live in as little as 30 minutes.

Businesses need timely feedback

Timely feedback is not only about customer service. Feedback serves as a valuable metric that lets you know if you are putting your efforts into features that make money for your business. When releases occur infrequently, knowing if you are investing in features customers actually want is difficult. It is possible that there is no real feedback for months, or even years. One consequence is that the success of a project is often measured in terms of meeting schedules and budgets rather than on whether the release makes money or not. Another is that those features are generally very complicated by the time they are released, and many people have invested large amounts of time, money, and their own blood, sweat, and tears into those features. The stakes are very high.

With continuous delivery, the success or failure of a service is soon evident. When the changes are small, features can be revised or even withdrawn without much pain.

Businesses need to reduce risk

Contrary to what many people believe, continuous delivery decreases risks. Small, frequent changes are much more manageable than a few large ones, where many things can go wrong. There is no need for a white-knuckled release process anymore.

Businesses need a transparent and accountable process

To improve quality you need a fully auditable process. Continuous delivery encourages transparency throughout the entire deployment pipeline. Everyone on the team is familiar with the release. Automation ensures that every change is tracked as it makes its way from development to production. The results are there for everyone to see. If you need to satisfy compliance and regulatory committees, using continuous delivery ensures that your records are complete.

What is devops?

Moving to continuous delivery involves new technology, but there are organizational and cultural aspects as well. In particular, you must address the longstanding tension between development and operations. A development team values innovation. It is measured on how well it delivers new products to market. An operations team values stability. It is measured on how well it keeps the infrastructure running.

Quite often the risk to existing infrastructure is borne by the operations team deploying the code and not by the developers who produced it under stress to meet challenging deadlines. Deploying a new product and being responsible for it once it is released is filled with risks.

A 2011 paper from IBM called "Collaborative DevOps with Rational and Tivoli" reports that "overcoming complexity requires a tremendous amount of coordination and communication, yet the two teams who need to collaborate the most — development and operations — are typically not even managed within the same reporting structure."

Devops is a methodology for resolving these tensions with both a technological and collaborative component. Devops is an outgrowth of the Agile method and relies on the Agile process for developing code. It also aims to improve the first steps of creating a component, such as developing stories and requirements, and the last stage of the process, which is releasing the code. Although it is still evolving, one of its basic devops tenets is to break down the barriers between infrastructure and code and to blur or even eradicate the boundaries between development and operations.

Although the term "devops" may be new, at least some of its fundamental practices are not. The CTO of Amazon, Werner Vogels, outlined Amazon's design philosophy, which has evolved over years, in his talk "Amazon and the Lean Cloud." He notes that all team members at Amazon share all responsibilities. There are no operations teams

and no development teams. Instead, Amazon's philosophy is "You build it, you run it." Every service has a small team dedicated to it, and that team supports the service throughout its life cycle. This includes interactions with customers. These teams are called "two-pizza teams." In other words, the optimal number on a team is about 8 to 10 people, or the number of people it takes to eat two pizzas. Vogels says that larger teams need too many meetings and smaller ones don't function well.

John Willis, a longtime proponent of continuous delivery and devops, uses the acronym CAMS to highlight its key components. CAMS stands for culture, automation, metrics, and sharing.

Technologies

An effective devops strategy relies on the technologies that make up a continuous-delivery pipeline. In his lecture "Camp DevOps 2011," as well as in his book *Continuous Delivery*, Jez Humble separates these technologies into three categories: configuration management, continuous integration, and automated testing. This section gives a brief description of these three areas. For more information, see the section below called "Constructing a continuous-delivery pipeline."

Configuration management

Humble says that for configuration management you should try for the level of complete automation. You should be able to plug in a machine and then run an automated process that completely provisions it and deploys and configures any applications without manual intervention. Developers should be able to use a single command to build and test software and another single command to deploy the software to any environment.

As an example of how to prove that your automatic configuration is robust, Humble cites Netflix's Chaos Monkey. Netflix's philosophy is that the best defense against

failure is to fail often. By deliberately causing individual services to fail, Netflix forces its developers to build networks of services in a resilient fashion. To that end, it created Chaos Monkey, which runs on Amazon Web Services (AWS) and randomly terminates virtual machines. The system should be able to automatically provision new machines to compensate so that no one on the outside knows there were problems. To learn about Chaos Monkey, see the blog post "Chaos Monkey Released Into the Wild."

Continuous integration

On his blog, Martin Fowler of ThoughtWorks defines continuous integration as "a software development practice where members of a team integrate their work frequently, leading to multiple integrations per day." In effect, every time someone commits a change to the revision-control system, it is integrated into the build. Each change is then automatically tested in order to detect errors as quickly as possible. The point of continuous integration is to prove your code still works after every change. If there are problems, you can address them immediately. When changes are small, the connection between the failure and the code change that induced it is apparent.

Automated testing

Testing is not an activity that is the sole responsibility of testers. It involves every member of the team. Tests are performed automatically as part of the continuous-integration process. Team members collaborate from the start of the project to write automated tests before anyone begins writing the actual code. There should be tests that operate at different levels, such as unit tests that test a single feature, component tests that test an entire component, and acceptance tests that test the entire system. Taken together, these tests constitute an executable specification, and they prove that the code does what it is supposed to do.

Metrics

Another point that Humble makes in his talks is the need for good metrics. These include standard operational metrics such as "time to detect" (TTD) and "time to

resolve" (TTR), but an often overlooked set of metrics are those that apply to business goals. Examples of business metrics are the amount of revenue you're generating, the number of orders that have been placed, and the number of users you have. Business metrics can give other measures a context that they otherwise lack. Your operational metrics may register that your CPU usage is low. But is this because the thread pool is depleted or because there simply aren't very many customers?

Collaboration

The technologies to implement continuous delivery are available. The greater challenges may exist within the structure of the organization. As stated elsewhere in this report, devops depends on cross-functional teams where people work together, even people who traditionally have viewed one another with suspicion because of conflicting goals.

In devops, everyone involved in a project is accountable and a full participant in the project's life cycle. There is no point where one group of people is done with the project and it becomes the sole responsibility of another group. System administrators join in feature discussions, architectural-planning meetings, and postmortems. Their knowledge of the infrastructure can be useful to application developers. For example, if data is going to be spread across multiple databases, developers can plan for this when they write their code with information about network latency and bandwidth limits from operations. In return, developers wear pagers when code is being released and are on call until everyone is assured that the release was successful. The released code is no longer the responsibility solely of operations.

Practitioners of continuous delivery and the devops philosophy stress the importance of collaboration. In an interview, Michael Rembetsy described how Etsy became compliant with the Payment Card Industry Data Security Standard (PCI DSS) and how the company implemented a continuous delivery system that observed all the associated regulations. The predominant theme was not technology and review

committees; instead, it was the need for collaboration, communication, and a transparent process.

Jez Humble's lecture "Camp DevOps 2011" makes a similar point. He says that meetings where people are forced to sit in a room together will not solve the problems. Simple things can be more powerful. If the developers have a release party, the operations team should be invited. In essence, try inviting one another to lunch.

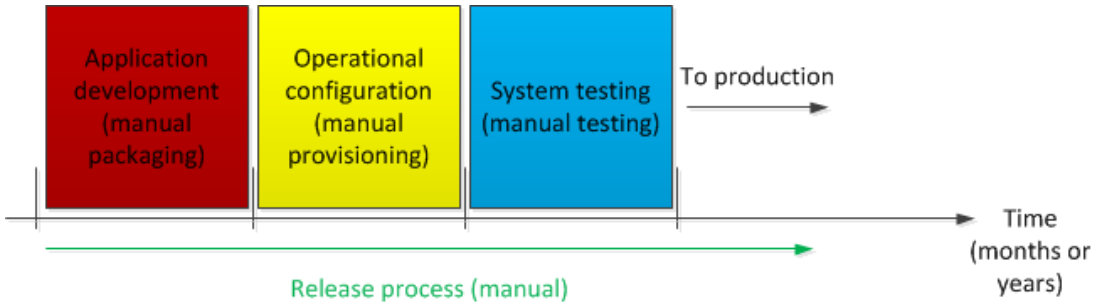
Ben Rockwood, the director of Systems Engineering at Joyent, echoed these sentiments in his keynote address at the twenty-fifth Large Installation System Administration Conference (LISA 2011). (Independently, both Rockwood and Humble cite beer and chips as the most important tools for implementing a successful devops group.)

Making smart use of smart people

Some employees fear that devops and continuous delivery are really just ways to cut down on personnel and simply make fewer people do more work, but this is not the case. People's specialties remain valuable. Their knowledge is not something that's easy to replace. There are changes to the process, but these are because people's efforts are concentrated at the beginning of the project rather than distributed throughout the release process.

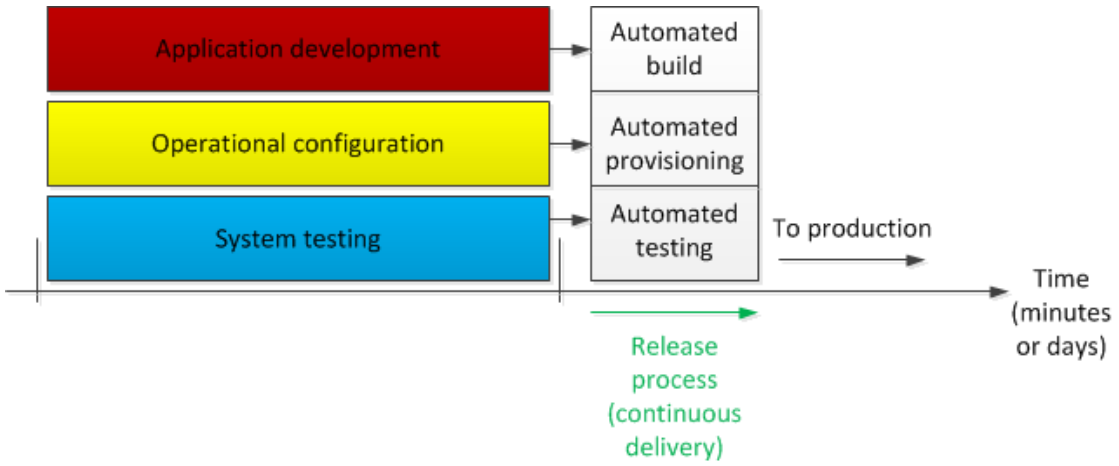
Figure 2 shows a traditional release process, where efforts are serial in nature and the release process is long. Figure 3 shows a process that uses continuous delivery.

Figure 2. The traditional release process



Source: GigaOM Pro

Figure 3. The continuous-delivery process



Source: GigaOM Pro

With continuous delivery, efforts are concentrated at the beginning of the process, with the goal being the creation of a fully automated delivery pipeline. The automated release process itself is short. Note, however, that the level of effort hasn't changed. Instead, it has moved forward in time. For example, instead of performing manual tests, testers write automated tests. Instead of performing manual provisioning steps,

the operations team creates automated provisioning scripts. These activities occur in parallel with application development.

Who is devops?

As we have said, devops is an emerging pattern that encompasses both technology and mindset. Many people would argue there is no such thing as a devops title, but the word is beginning to appear in job descriptions. For example, Wildfire titled an operations engineering job description as "devops." It says, in part, "As part of the Ops team, devops Engineers work with the Dev Team to provide high availability services to our clients. Ops Engineers provide input on and execute security and deployment practices, scaling and metrics, as well as running general day-to-day server management."

Constructing a continuous-delivery pipeline

To understand continuous delivery in concrete terms, let's look in more detail at the component technologies that are required to implement it. Particular tools mentioned in this section are only a few of the available options. Also, many of the companies listed have end-to-end solutions you may want to examine. The same principles apply to the infrastructure as they do to applications.

Version-control system

Version-control systems allow you to keep multiple versions of files so that when you modify something you can still access previous versions. A good version-control system stores and gives access to all the files that have ever been stored in it. It also allows people who may be distributed across the world to collaborate.

Keep every relevant artifact in version control, not just source code. Examples include tests, database scripts, build and deployment scripts, documentation, libraries,

configuration files, and metadata that describes stored data. If you do this, you control every aspect of the project's environment. A new team member should be able to use the version-control system to recreate the environment from scratch. In keeping with the precepts of continuous delivery, team members should commit their changes to the version-control system frequently.

Examples of popular source-control systems include GitHub's Git, Apache's Subversion, and Microsoft's Team Foundation Server.

Automated build system

An automated build system allows you to write scripts that describe the entire build process. For example, you may specify the sequence of tasks in the build process, prepare a build environment, validate the source code, compile the sources, run system tests, build the documentation, produce packages that contain all the artifacts, and release builds to a package repository.

Popular tools include Apache Maven, Ant, Gradle, uBuild, and Microsoft Team Build, which is part of Microsoft Visual Studio.

Package repository

A package contains everything necessary to install your application as well as information such as the package's version, documentation, data, and configuration information. A package repository stores packages that you want to make available for deployment. Package-management systems (or repositories) allow you to perform tasks such as installing, uninstalling, verifying, querying, and updating software packages.

By providing an authoritative source for release artifacts, a well-managed package repository can help address issues of compliance, traceability, and audit while enabling releases to proceed more quickly. A good package repository can enable faster build times, facilitate more-efficient collaboration among teams, and reduce the learning curve for new developers.

Package repositories are often overlooked. To learn more about them, see UrbanCode's webinar on the subject. NuGet and RPM are examples of package repositories.

Continuous integration and delivery server

Continuous integration (CI) means that every time someone commits a change, the entire application or environment is rebuilt and a series of automated tests are run against it. Problems are fixed before any other work is done. Continuous integration requires version control, automated build, and commitment from the team.

Continuous integration requires:

- **Version control.** Everything must be checked in, no matter how small the project or the change.
- **Automated build.** It must be possible for either a person or a computer to initiate the build process in an automated fashion.
- **Commitment from the team.** Everyone must agree to check-in small, incremental changes frequently. The highest priority is to fix any change that breaks the build. This is not trivial. It requires discipline and a willingness to break what may be long-standing habits.

Some open-source options include CruiseControl and Jenkins. Commercial options include products from ThoughtWorks, JetBrains, UrbanCode, and Microsoft.

Automated testing

Automated test suites should test the functional aspects of the system as well as nonfunctional requirements such as capacity and security. Testing involves the entire team and should be done continuously from the beginning of the project. The goal is to catch problems early, when they are easiest to fix. Automated tests should include unit tests, functional tests, and build-verification tests (BVTs).

Benefits of automated testing include:

- Developers get quick feedback on their work.
- Testers do not need to perform repetitive tasks and can concentrate on more-complex activities.
- Tests can be used as a regression test suite to prevent the reintroduction of previous defects.
- Tests can be used to automatically generate requirements documentation.

There are many tools available for automated testing. Two examples are Microsoft Visual Studio and Selenium Grid. Tools such as Cucumber and Twist allow you to write requirements as executable test scripts.

Infrastructure as code

This section deals specifically with ways to manage your infrastructure. The most obvious point about virtually any company's infrastructure is that it has grown increasingly complex over the years. People are constantly looking for ways to improve scalability and manageability. How do you build and manage systems that can handle

millions of users and have complex environments that can include tens of thousands or even millions of machines? You need:

- Virtualization, which allows isolation, dynamism, and better resource utilization and is most often the underpinning of cloud computing
- Cloud computing, which provides scalable computing resources and a pay-as-you-go model
- Continuous delivery, where infrastructure is treated as code and goes through the same automated pipeline as application source code

Managing the infrastructure

Now that you have a scalable, responsive infrastructure in place, how do you manage it? Here is a simple example to demonstrate how quickly complexity sets in. Suppose you have a standard three-tier application that uses:

- Web servers that send content to users
- Application servers that contain the logic
- Database servers with information

Your application is becoming popular, and you want to add a server to the application layer. To do this, you may need to change the configurations of all the servers, which means you may need to restart them. In other words, adding even a single server can mean many changes to the existing infrastructure. This can take hours, and if it is a manual process, it can be extremely error prone. Now imagine having to do this across an entire cloud.

Large-scale buildouts and the necessity to manage complex environments present a challenge to manual approaches. Also, ideally, the infrastructure is dynamic and

changes according to the loads it carries. This means that golden images, which are snapshots of the system's state, are not the answer: A virtual hard disk or any other kind of packaging is static and will quickly become out of date. Automation is the key to solving this problem.

Automation is essential

Many businesses use a team of operations people to manage their environments. Changes, such as reconfiguring a server, are performed manually, and records of the changes are kept in a database. However, humans tend to be bad at repetitive tasks. They become bored or sloppy and begin to cut corners. Computers, on the other hand, excel at these tasks. Automated configuration management allows you to:

- Manage complexity
- Create a repeatable and scalable infrastructure
- Gain a coherent understanding of your infrastructure

There are tools available to help you treat your infrastructure as code. The next section discusses some of them.

Automation tools for infrastructure

Opscode helps companies develop fully automated infrastructures. Adam Jacob, a co-founder and the chief customer officer, discussed the problems he faced as a consultant in his speech "Opscode Chef: State of the Union, Part 1." He recalled that his initial impulse was to find a one-size-fits-all collection of standardized approaches that he could apply in all situations. The reality, he found, is that no single infrastructure is identical to another, and every infrastructure has its idiosyncrasies, with its own set of tweaks and tunings.

When defining a way to approach his problem, he felt that a good tool would allow you to make easy things easy. The tool should allow you to do routine tasks without a lot of work on your part. For example, you should be able to install a package or write a configuration file and tell it to run. The tool should make it possible to think at the level of a single system or across large numbers of servers. Also, the tool should be:

- Idempotent
- Convergent
- Predictable
- Order preserving

"Idempotent" means that once a component is in its desired state, reapplying the configuration operation should not change it. "Convergent" refers to the fact that you can never guarantee the initial state of any resource, so the tool must be able to bring a system that is in an unknown state into one that is known and verifiable. "Predictable" means that when something fails, the tool must react in a predetermined and well-understood way. An order-preserving tool lets you establish dependencies in a particular sequence so that operations occur in the required order.

There are various tools on the market that can help you, and each has its own approach. Here are descriptions of some of them. A detailed comparison of these tools is outside the scope of this paper.

Opscode Chef

Opscode's Chef provides an extensible set of resources, which are descriptions of packages and services that need to be configured on a node. In Chef, a node is a data structure that describes the entity under configuration management. It is usually a server. Resources are deployed on nodes as parts of dynamic, search-driven recipes.

Chef is very flexible in how it lets you manage configuration. For example, roles express common configurations that are shared by a group of nodes. The role of web server might be applied to a particular set of servers in order to install an Apache server in the same fashion on all of them.

Chef abstracts away the commonalities of the infrastructure while giving you the ability to describe features that are particular to your unique system environment. Chef is not limited to a fixed schema of resources and node types. Opscode believes that with the many legacy systems that are part of almost every company's infrastructure, there is no single vocabulary that can describe all existing systems. Consequently, Chef allows you to define your own resources and modes for the legacy components it does not cover.

Chef components, including user-created extensions, are written in the Ruby programming language.

Puppet Enterprise

Puppet Labs' product Puppet Enterprise (see disclosure at the end of this section) is a tool that emphasizes a model-driven, declarative interface for server management. Puppet Enterprise creates a catalog of all entities that are being managed and allows larger logical constructs to be built from smaller ones. The catalog is a virtual representation of the data center environment to be managed. System administrators can perform operations on the catalog to gauge the impact those operations would have on the physical environment.

Disclosure: *Puppet Labs is backed by True Ventures, a venture capital firm that is an investor in the parent company of this site, Giga Omni Media. Om Malik, the founder of Giga Omni Media, is also a venture partner at True.*

CFEngine

CFEngine provides its own programming language with a compact runtime that is implemented in C. Policy is described using promises: Every statement in CFEngine 3 is a promise to be kept at some time or location. Every promise you make in CFEngine is continuously verified and maintained. It is not a one-off operation but an encapsulated process that repairs itself should anything deviate from the policy. You use CFEngine to express high-level intentions about the system, and the runtime determines the algorithms needed to implement the result.

Devops and ITIL

The Information Technology Infrastructure Library (ITIL) is a set of practices for IT service management (ITSM) that focuses on aligning IT services with business needs. ITIL describes procedures, tasks, and checklists that are not organization-specific and that an organization can use to establish a minimum level of competence. It allows an organization to establish a baseline from which it can plan, implement, and measure. It is used to demonstrate compliance and to measure improvement.

There is controversy about the compatibility between ITIL and devops, and the subject is beyond the scope of this paper. However, to begin, you might want to consult Jez Humble's book *Continuous Delivery*, which does address the issue to some extent. You might also want to look at Eric Minick's blog post "Gartner also observes DevOps mixing with ITIL." Minick is the lead consultant at UrbanCode. In this post, he cites a Gartner research note written by Ronni Colville and George Spafford that says, in part:

> "With the increased demand for more frequent releases and predictable outcomes, IT organizations are beginning to realize that they can no longer focus on production-only processes. By factoring in the devops philosophy and using a more integrative, holistic approach toward the release of new and changed services into production, IT management can tailor their efforts to

create solutions that address the need to increase the velocity and quality of releases from service design, development and transition into production."

Tips for success

Here are some quick tips to help you be successful when implementing a continuous-delivery solution.

Use the Agile development process. Use the Agile development process to structure your teams. Think about training scrum masters for every team. Time box your iterations to make sure they remain manageable. Only consider software to be done if it's functional, production-ready, and deployed.

Be realistic: Legacy systems always exist. Most organizations have existing systems that rely on legacy hardware and software. These systems don't go away overnight. For example, you can put the continuous-delivery pipeline in place and drop code quickly to production, but operations won't be ready to handle it unless you take specific steps to account for legacy systems. One approach is to put controls in place in the pipeline to deal with legacy problems and remove them when they're no longer necessary.

Use automated testing. Use a completely automated test suite to ensure that your results are predictable and auditable. Examples of the types of tests you should include are unit tests, functional tests, and build-verification tests (BVTs). Consider having a commit test suite that runs in less than 10 minutes. This test suite occurs before check-in and ensures that you haven't introduced any obvious regressions.

Aim for complete automation. A content-delivery pipeline is fully automated, and every change triggers that process. If you have a manual step that, for example, promotes the change from QA to production, you aren't fully automated. However, you can gradually introduce steps toward this goal. You don't have to do everything at

once. Most businesses have legacy systems and firmly entrenched policies and attitudes that have to be taken into account.

Increase transparency. Developers should understand the deployment process and configuration. Operations personnel should understand application development. All information about a project should be visible to everyone. Transparency allows you to catch errors much earlier in the process.

Invest in monitoring. You need automated tools to have confidence that the application is running as it should. Instrument your applications and infrastructure, store the data, create dashboards to aggregate and present the data, and set up notifications so people can find out about events.

Adopt IaaS solutions. Infrastructure-as-a-Service (IaaS) products mean delivering virtual servers, desktop computers, or remote storage from the cloud. An IaaS approach to infrastructure can offer:

- Faster responses to changing business conditions or internal needs, enabled by the ability to rapidly provision a system and by scalability, both up and down
- Productivity increases, because you can access applications and data from anywhere and because of the reliability of a distributed-computing model
- Reduced capital outlay for hardware acquisitions, maintenance, data center real estate, and power and cooling, because of the cloud's pay-as-you-go model

Develop a culture of cooperation. The barriers between developers and operations need to be dismantled. Just as applications and infrastructure should be looked at holistically — one cannot exist without the other — so should the teams that create and manage them.

Recognize that stuff happens. Be realistic about outages: They are a fact of life. Systems will always fail. It's a question of when and not if. Train your people to expect

outages and to react to them as routine occurrences, not crises. As an analogy, no one would want firefighters whose approach to learning about putting out fires was to wait for buildings to burn down. Train for failures.

Ancestry.com: a case study

Ancestry.com is the world's largest family-history resource. It operates a network of genealogical- and historical-record websites focused on the U.S. and nine foreign countries, develops and markets genealogical software, and offers a wide array of genealogically related services. As of June 2012, the company provided access to more than 10 billion records and 38 million family trees, and it had 2 million paying subscribers. It recently finished implementing a continuous-delivery platform. This is an overview of how it did it. For the complete details, see John Esser's presentation "Continuous Delivery @ Ancestry.com."

Timeline

John Esser was hired by Ancestry.com in March 2010 to institute Agile practices at the company. He was its first scrum master. (The scrum master is accountable for removing impediments that interfere with the team's ability to deliver the sprint goal and deliverables. The scrum master is not the team leader but acts as a buffer between the team and any distracting influences.)

Around December 2010, Ancestry.com instituted an enterprise-level Agile implementation to be able to work across groups for planning and coordination. By May 2011 the Agile process was working well and it was releasing new features every two weeks.

At that time Esser read Humble's book *Continuous Delivery* and was inspired to implement a continuous-delivery platform in Ancestry.com's largely Windows-based environment. To that end, Ancestry.com adopted Go from ThoughtWorks, which is an

Agile-release-management package, and Chef from Opscode, for automated configuration and deployment.

It completed its end-to-end continuous-delivery platform in October 2011. This included transforming a bare-metal data center into a virtualized one.

The acid test

The United States census from the 1940s was released in March 2011. Ancestry.com wanted to be able to deploy new services and features for it using the continuous-delivery platform. It was so successful that some teams were deploying once every hour or two. Currently it may only take a few minutes to go through the entire deployment pipeline. The average is around 30 minutes. Esser also found that people liked the platform and process so much that developers were willing to be much more involved in the deployment process.

Elements of the Ancestry.com process

This is a brief description of the steps Ancestry.com follows to deploy code. (The relevant technologies are included in parentheses.) The basic tenets are:

- Use an Agile development process. Keep changes small with time-boxed iterations.
- Use a version-control system for all application and configuration artifacts.
- A commit triggers the continuous-delivery pipeline (Go).

Here is what occurs once the pipeline is triggered:

1. The code is built, and unit tests run in parallel across the build grid.
2. On success, a deployable package is uploaded to the package repository (NuGet, RPM).

3. Configuration changes are applied to any application servers in the integration environment (Chef).
4. The code is deployed to the integration environment (Chef).
5. Integration tests are run (Selenium Grid).
6. On success, configuration changes are applied to preproduction servers (Chef).
7. The code is deployed in a preproduction environment (Chef).
8. Preproduction tests are run to verify the deployment was successful (Go).
9. On success, a push-button approval process releases the code and orchestrates the deployment into a production environment (Go).
10. The code is deployed into production (Chef).
11. Smoke tests are run to verify the code is operating as it should (Go).
12. On success, everyone is done!

The results

Ancestry.com can now provision a continuous-delivery pipeline in a few days. Initially it took a few weeks. The company can go from a check-in to deployment to live in as little as 30 minutes. A typical deployment once took hours. Before continuous delivery, deployments might have happened once every two weeks, but a particular team might not have deployed for months. Now teams often deploy multiple times per day.

Key takeaways

- The internet, virtualization, and cloud computing are enabling the development of new business models. These new models are putting traditional business models, such as brick-and-mortar retail, under stress.
- The growing complexity of code and the infrastructure can make it difficult to deliver software in a timely manner.
- People are looking for ways to release code quickly.
- Use the Agile method to develop code.
- Make incremental changes frequently, not huge changes infrequently.
- A continuous-delivery pipeline can help you deliver code. This is an automated implementation of the build, deploy, test, and release processes.
- Every committed change made to configuration, source code, the environment, or data triggers the pipeline.
- Treat your infrastructure as code. Changes in infrastructure go through the same automated pipeline as changes to source code.
- Use automated tools to manage your infrastructure.
- Devops is an outgrowth of the Agile movement. It espouses close collaboration between development teams and operations teams.
- Try to foster a spirit of cooperation among all team members. Representatives of all software disciplines should attend meetings, from the very beginning of the project through the postmortem.
- Developers are responsible for their code until everyone knows the release was successful. Equip them with pagers.
- There is a social dimension to successful collaboration between dev and ops: Beer helps.

List of resources

This section includes links to the technologies mentioned in this document.

Release management

Microsoft

ThoughtWorks

UrbanCode

Version control

Git

Microsoft Team Foundation Server 2012

Subversion

Automated builds

Apache Ant

Apache Maven

Gradle

Microsoft Visual Studio

UBuild

Package repositories

NuGet

RPM

Continuous integration

CruiseControl

Hudson

JetBrains

Microsoft Visual Studio

ThoughtWorksUrbanCode

Testing platforms

Cucumber

Microsoft Visual Studio

Selenium Grid

Twist

Infrastructure as code

CF Engine — CFEngine 2 and CFEngine 3 Nova

Opscode — Opscode Chef

Puppet Labs — Puppet

Books

Continuous Delivery, by Jez Humble and David Farley, is a good starting point. It contains a bibliography that lists many other useful books and articles.

About Dave Ohara

Dave Ohara's corporate career began at HP and continued with Apple and Microsoft. After over 20 years of developing technology, he switched to the data center industry to better understand the challenges in going to market with online services. He now consults with companies in the data center community. He holds a degree in Industrial Engineering and Operations from the University of California, Berkeley.

About GigaOM Pro

GigaOM Pro 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 Pro addresses the need for relevant, illuminating insights into the industry's most dynamic markets.

Visit us at: pro.gigaom.com