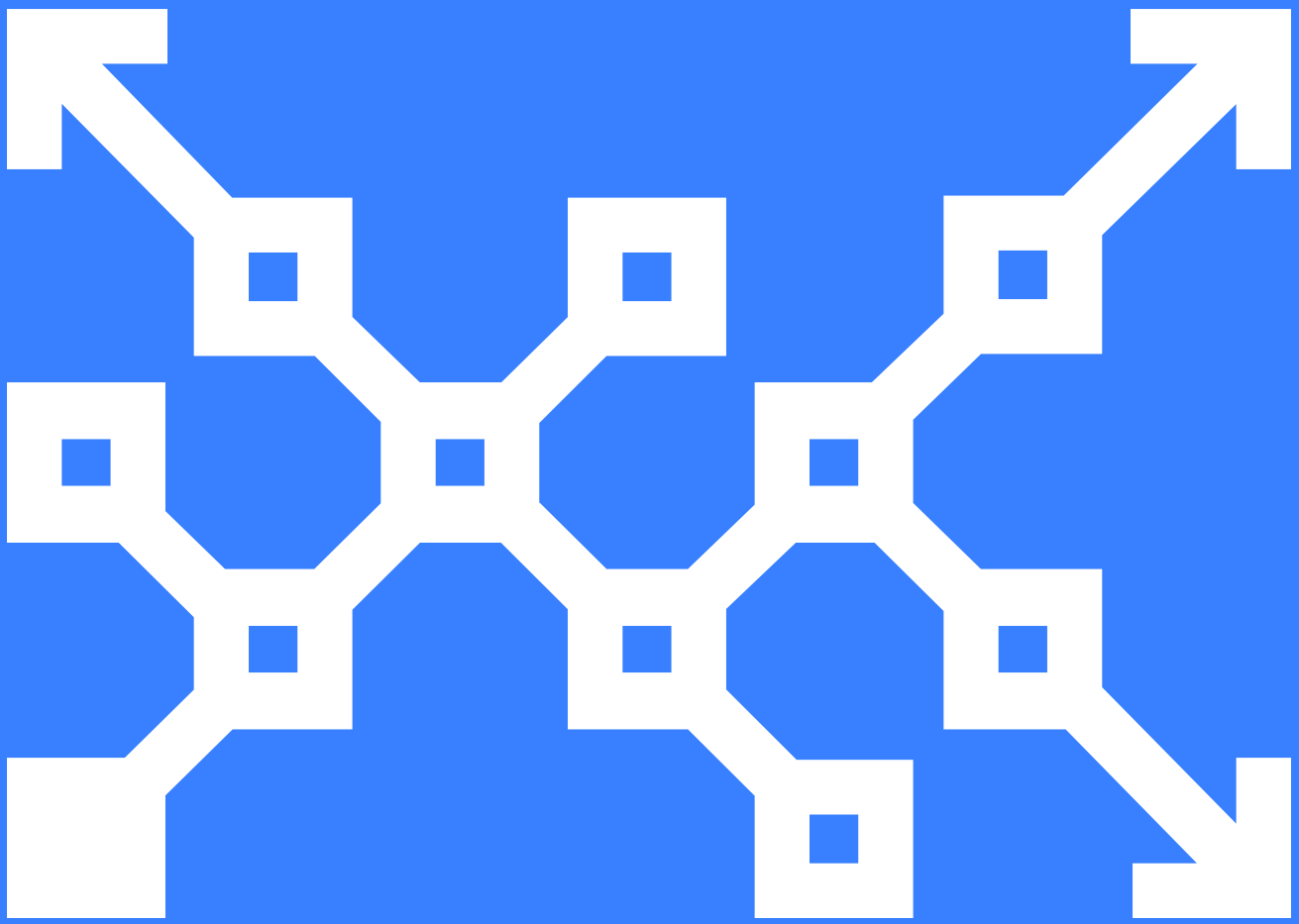
E-Book

Continuous Delivery: What It Is and How to Get Started



puppet

Contents

Introduction

Twenty years into the cultural revolution started by the web, there's still something magical about the way an application written by a few developers and delivered by a handful of sysadmins can delight millions of people.

Whether it's a business-critical procurement application powered by sophisticated algorithms and dozens of data centers, or a deceptively simple to-do app that nails the essence of staying organized, software matters to people in ways that can still surprise anyone who remembers sitting in front of an 8-bit computer and pecking in the BASIC for a checkbook program, or feeding FORTRAN punch cards into a warehouse inventory system.

It would be nice to believe that we — developers, sysadmins and managers — are somehow keeping up, delighting our internal and external customers with things that offer real value.

It's not enough to have a great idea and execute on it once. You have to execute, get feedback, refine, and execute again — and again and again. To keep competitors from grabbing a piece of your market, you need to cycle with ever-increasing speed and agility.

Continuous delivery is about this cycling process. In this white paper, we'll explain:

- What continuous delivery is (and isn't).
- How important continuous delivery is to your business.
- How to get started with the cultural and technological changes required to practice continuous delivery.

We hope you'll enjoy it.

What is continuous delivery?

There's plenty of confusion as to what continuous delivery actually is. Arguably, Martin Fowler has offered the best definition. (See sidebar.) Fowler's definition is based on the idea that software is created to fill a business need, and that it must be delivered more frequently and reliably so customers — whether they are internal or external — can start getting its benefits. For a company that ships software, or that depends on software to deliver its products or services, faster delivery of working software is critical to becoming or staying competitive.

The problem is, trying to speed software cycles through traditional waterfall development techniques usually results in late, buggy software. In fact, most developers and IT professionals know this, and are justifiably leery of speeding things up. With traditional methods, they simply aren't equipped to test software frequently or thoroughly enough to release high-quality code.

Continuous delivery works because it incorporates automation, frequent code releases, testing at every stage of the process, and a pull-based architecture that permits only successful releases to move to the next stage. All of these reduce errors and make it easier to improve the software delivery process.

Automation allows you to make successful processes repeatable. When you decide to introduce a new feature, make a change to a service or system underlying your application or an adjustment to your infrastructure, automation lets you make the change quickly and safely, without introducing the human errors that would result from repeating a process manually.

Continuous Delivery According to Martin Fowler

You're doing continuous delivery when:

- Your software is deployable throughout its lifecycle.
- Your team prioritizes keeping the software deployable over working on new features.
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them.
- You can perform push-button deployments of any version of the software to any environment on demand.

More from Martin Fowler on continuous delivery

Releasing code frequently, rather than the traditional model of shipping big releases once or twice a year, means you're testing more often against existing code and systems. There's less change in each release, so it's easier to isolate and fix problems. It's also easier to roll back when needed.

Automated testing enables you to catch errors early in the process and **pull-based architecture** prevents you from inadvertently passing code that fails automated tests to the next stage of development. This prevents compounding errors and making them harder to diagnose.

Continuous delivery also works because teams share responsibility for the process, putting an end to siloed teams handing work off to each other and then walking away. Because you're working together as a single team made up of people with different skills and responsibilities, errors and issues become an opportunity to collaborate on improving the process — not an excuse to blame someone else.

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

"First Principle," *Agile Manifesto*. 2001

Continuous delivery builds on agile practices

Continuous delivery is the natural outgrowth of the Agile movement. Agile seeks to correct the problem of late, large, buggy software releases by promoting iterative, incremental changes to code, and collaboration between teams. Agile's benefits include the ability to adapt quickly to change and lower risk to the business.

Continuous delivery requires a cultural shift

Continuous delivery is a set of practices enabled by a toolset. It's also a shift in how people think about delivering software, and a shift in work culture. Instead of delivering software every six months to a year, and spending long periods of time fixing the things the software breaks, teams deliver smaller code changes more frequently, so smaller problems can be fixed more quickly. Instead of developers crafting code and then tossing it over the metaphorical wall to IT operations to deploy, teams agree that creating, testing and deploying quality code is a shared responsibility. Everyone is on the same team, and the goals of the business are shared goals.

Continuous delivery and DevOps

Continuous delivery and DevOps are closely related in that each supports the other in its own way. DevOps is the practice of ensuring that an organization's development and physical environments, as well as processes, are set up to deliver new builds into production as rapidly as possible. It requires a tight integration between Dev and Ops. Successfully implementing DevOps in your organization puts you one step closer to continuous delivery.

A key point of DevOps is to understand that we're all trying to achieve the same thing — creating business value through software applications. With DevOps, developers need to learn how to build high-quality, production-ready software and Ops needs to learn that the Agile techniques developers have been using for years are powerful and enable effective, low-risk change management. At the end of the day, it is all about getting better at working together and focusing on creating business value, rather than trying to optimize our own domains.

Continuous integration is part of continuous delivery

One frequent confusion is equating continuous delivery and continuous integration. Continuous integration is the practice of integrating and testing new code against the existing code base with every change, and it's a necessary part of the continuous delivery process. Tools for continuous integration are discussed later in the chapter, **The tools of continuous delivery**.

Continuous delivery is not the same as continuous deployment

People sometimes use the terms continuous delivery and continuous deployment interchangeably. They are not the same thing. Where continuous delivery is a set of practices that ensure code can be deployed to production at any time, continuous deployment takes it one step further by automatically deploying code that has successfully passed through the testing stage.

> You can't achieve really high levels of quality and verifiability in my experience by throwing things over the wall. The teams need to be working very, very closely together, communicating on a daily basis, interacting on tasks. So cross-functional teams all focused on delivering high quality software is the way to go.
>
> **Dave Farley** talking to Peter Bell of InfoQ

Why continuous delivery matters

Companies that depend on software turn to continuous delivery to get better quality code released quicker, and more dependably. But different people have different reasons for wanting it. Here's a quick rundown of continuous delivery's importance to systems administrators, software developers and business managers.

Why does continuous delivery matter to sysadmins?

System administrators are rewarded for keeping the IT infrastructure running reliably to serve business needs. That means limiting the impact of any changes. This conflicts with the needs of software developers, who are rewarded for delivering new code — which means making lots of changes that can bring systems down.

Continuous delivery practices help sysadmins keep things running smoothly while giving software developers the ability to test code in an environment that closely resembles production conditions.

Automated configuration — an important part of continuous delivery — makes it possible for sysadmins to provide developers with the ability to turn on their own testing environments. Developers don't have to wait for weeks to test their code and use the results to move on, so they don't see sysadmins as standing in their way.

Deployment becomes much less stressful, too. Again, configuration management lets you make sure the development, testing and production environments are closely matched, so any errors that new code could cause in production are discovered — and corrected — long before deployment.

Practicing continuous delivery permits much more flexibility to experiment and learn from those experiments, in a low-risk way. IT operations people who enable that flexibility are providing real value to the business.

Why does continuous delivery matter to developers?

Software developers are rewarded for delivering quality software that addresses business needs, on schedule. That's where the conflict often arises between developers and sysadmins: Developers want to release code to production, and sysadmins worry about that code breaking the systems they manage. Developers also want test environments when they need them, and if they're stuck filing requests that take weeks to fulfill, their work is slowed down.

Continuous delivery practices give software developers the ability to provision themselves with production-like stacks and push-button deployment so they can run automated tests. Instead of standing in their way, the ops team helps developers get their work done.

Continuous delivery depends on continuous integration, which means every change is merged into and tested against the main code base, reducing the opportunity for long-standing feature branches and large merge windows that can lead to serious errors. Deployment becomes much less stressful when changes are small and tested at every step. And if you need to, it's easier to roll back changes to your code, changes to the environment, or more importantly, both together.

Practicing continuous delivery permits much more flexibility to experiment and learn from each experiment, in a low-risk way. Creating software in these conditions is less stressful, which makes it easier for developers to work creatively and productively, and deliver higher quality code, more frequently, to meet business goals.

Until your code is in production making money or doing what it is meant to do, you have simply wasted your time.

Chris Read, quoted on ***Jamie's Blog***

Why does continuous delivery matter to the business?

You'll face lower risk from deployments. Continuous delivery, once processes are in place and regularly used, gives you the option of much faster software development cycles. Instead of releasing code once or twice a year, companies have the option of releasing multiple times per day. When you're releasing at that rate, each release is small — maybe only a single line of code — so the risk to system stability and customer service is much lower. Every change is easier to roll back, easier to test — and if there's a failure, easier to diagnose, because so little was actually changed. A company still may not want to release new code as frequently as multiple times per day, but what really matters is that every piece of code that's checked in is ready to deploy.

This makes it much more viable to continually test small changes on your systems and on your customers. For example, you might want to see if a big blue "buy now" button on the home page makes people act more quickly than your existing green button. With continuous delivery practices in place, you can test that change to see if it breaks anything before rolling it out. And you can limit the change to just a small percentage of website visitors to see how they respond, or see if visitors arriving at various times of day, or on different days of the week, react any differently. The feedback from these experiments helps business managers make better decisions.

You'll be able to respond to the market more quickly. Markets change all the time: Regulations get modified, commodity prices go up and down, safety warnings go out, celebrities launch fads. With faster cycle times, you can respond much more quickly to those changes.

Something you thought was profitable may turn out to be a loser. Website analytics may show that people visiting your site on mobile devices are buying more than those using desktop computers. Whatever decision you need to make, you can implement it faster if you're already practicing continuous delivery.

Once the whole organization gets comfortable with making more changes more often, you'll have a distinct edge over competitors whose deployments are infrequent, chaotic and error-prone. And the more you practice frequent code release, the better you get at it.

> It's virtually impossible for us to practice continuous improvement, to learn how to get better as teams or as individuals, and to acquire the skills that enable the successful creation of great products and services — unless we focus on getting that feedback loop as short as possible so we can actually detect correlations, and discern cause and effect.
>
> **Jez Humble, *Why Software Development Methodologies Suck***

You'll have happier, more productive people. Workplace satisfaction matters. There are just too many options out there for talented technology people; they don't have to stick around if they're getting burned out. Continuous delivery is known to reduce stress on technical teams, ending the 2:00 AM pager calls and reducing the last-minute technical fixes that themselves add to technical debt, and slow future releases.

Continuous delivery also fixes another source of developer dissatisfaction: writing code that never gets released. It's not at all uncommon for blocked development pipelines to swallow code forever, and no developer likes working on a product that no one ever gets to use.

Bonus: When there's less stress, there's more room to be creative. Technical teams who aren't fighting fires and fixing bugs all the time have the bandwidth to innovate in ways that benefit the business.

The practice of continuous delivery

Continuous delivery is, for many people, a kind of ideal, something that many people believe can be achieved only by small startups, or conversely, only by large resource-rich enterprise companies. In fact, all kinds and sizes of companies are implementing practices that are part of continuous delivery — and realizing great benefits, even if they don't implement it all.

Where is continuous delivery practiced today?

Continuous deployment is the ultimate version of continuous delivery, in which every change that makes it through automated tests is automatically deployed to production. Most experts will tell you this is not practiced by many companies.

More do eventually arrive at continuous delivery, but it can take a year or more of diligent and committed effort to transition from traditional waterfall software development and release practices to continuous delivery. Many companies benefit from implementing even one or two continuous delivery practices. Frankly, you don't have to do it all to create code with fewer bugs and improve your release cycle time.

Early leaders in this practice were (naturally enough) companies whose products and services are actually software. Other leading-edge adopters are sophisticated technology companies that work in the software space, such as GitHub. Continuous delivery has also been adopted by companies whose businesses are dependent on large-scale web operations. A few well-known examples of these are Salesforce, the popular customer relationship management tool; Box, the file sharing solution for businesses; Netflix, which streams its videos from Amazon's cloud infrastructure; and of course, Amazon itself.

The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win

Gene Kim's ***The Phoenix Project*** is a funny and realistic depiction of a normal dysfunctional company and IT environment, and does a good job of describing how to transition to a high-functioning and strategic IT team.

But companies that aren't your typical website-based businesses are also turning to continuous delivery to speed up delivery of high-quality software that helps them deliver their products and services. Examples include HR management solutions company ADP; manufacturing conglomerate GM; Pennsylvania State University; business liability insurance provider Hiscox; payment processing company Heartland Payment Systems; and a host of others in various industries.

Organizations that commit to continuous delivery see impressive improvements in their release cycles. AOL, for example, **went from six-hour deployment cycles to 45 minutes**, according to Gene Kim, DevOps advocate and author of *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*.

Don't forget continuous delivery is as much a cultural shift as it is a technical one

For most teams, the biggest shift is from separate teams dealing with the writing, testing and deployment of software to a single team that is responsible for the successful deployment of quality software — albeit one staffed by people who have specialized skills and are tasked with specific responsibilities.

Developers will still write code, QA people will still manage testing, and IT operations will still configure and manage infrastructure, but everyone shares ownership of the software development and release process. That means when something goes wrong, it's important to refrain from finger-pointing. Instead, get the delivery pipeline moving again, and then analyze the situation for a blameless postmortem afterwards. The object: to treat each such event as an opportunity to learn and make the process better.

Continuous delivery also requires that testing and IT operations people get involved earlier in the software design process. When all parties talk over what the new application will need, in terms of validation and infrastructure, the IT and testing people will be better prepared to test and deploy, and developers will be better equipped to make sure the code they write is testable and deployable.

When it comes time to deploy, most errors will have (hopefully) been worked out already. If there are errors, each deployment should be a small enough change that it's easy to roll back to the last known good state.

How do i get started?

Continuous delivery is not a thing — it's a process. Getting to where you're doing continuous delivery is itself a process. That's because it requires changes to tooling, to processes, and most important, to how people work together, and who works together.

Continuous delivery can be approached in small steps. The first real step is to follow your build process and write it all down. Then focus on two things:

- Which steps take the most time?
- Which steps are the most error-prone and/or require the most human intervention?

These are the steps to automate first. That yields immediate improvement, and gives you back some time to improve the rest of your process. Choose the next steps to automate by prioritizing things that are a headache or a time suck, improving your process incrementally. You can facilitate your early automation steps by:

- **Putting everything into version control.** You'll find a few popular tools listed in the next chapter, **The tools of continuous delivery**. If you need to track something that isn't in a file, you can log versions manually for now.
- **Adding tests to verify your code works.** Start with a small group of tests — don't try to achieve 100 percent test coverage right away.
- **Managing servers with configuration management tools.** Again, start with the servers that are involved with the steps you want to automate first (see the next chapter for a list of steps).
- **Monitor everything.** You can't improve what you don't measure. You need to know how long a step takes to start; how long it takes to complete; and how many resources it takes. You must continue to monitor and measure over time, to see if you are improving your process.
- **Be prepared for the early steps to be messy.** Getting good at writing code that doesn't fail is like training a muscle to be strong and flexible. Initially, when you start doing continuous delivery, the build may break often. Over time everyone gets good at writing code that doesn't break the build, because they can see what the sources of the breakages are. And from there, the process is re-tuned.

The tools of continuous delivery

Continuous delivery is a set of practices, rather than a set of tools. Nevertheless, assembling the right tools is absolutely necessary to enable both the process and the communication between developers, test and QA staff, and sysadmins.

The tools that enable continuous delivery have one thing in common: they treat everything, including IT infrastructure, as code. That makes it possible to automate every part of the process, and it also means that developers and IT operations people can communicate in the same language, through the same tools.

Adopting one tool, and the practices that go with it, can make a significant improvement in your software delivery process. Once your team is comfortable with the first tool and the changes in workflow around it, you can move on to the next.

Software-defined infrastructure

Applications shouldn't be hampered by poor coordination between computing, network and storage resources. Your infrastructure should be able to respond automatically to whatever an application needs to run efficiently.

A software-defined approach to infrastructure configuration enables that. Think of an application as a spider web of many different systems and devices, relying on a multitude of components and services. The application requires computing resources, has storage requirements, and will interact with databases and other services, such as a payment or verification system. Then there's network architecture to consider, and the dependencies between all these elements.

You can configure everything that supports the application with tools that treat infrastructure as code — the same tools your development team is already using. And when you adopt these tools to manage the entire software supply chain, everyone working to create the application — dev, QA, and ops — has the power to provision the environment the application requires to progress through each stage of the software delivery process.

The Puppet DSL and continuous delivery

Puppet's declarative configuration language supports describing and managing every environment in your continuous delivery pipeline.

By using a declarative approach, the Puppet DSL enables you to model applications and how they relate to the broader infrastructure in a reusable way, accounting for dependencies without tying application components to nodes.

Once you've written that declarative Puppet code, you can automatically deploy it to every environment in the right order on the right machines as the application is deployed, updated or destroyed. Consistency between environments supports the continuous delivery process.

Monitoring

Continuous delivery is about shortening your release cycles, and that means shortening your test cycles. Doing that is a process of continuous improvement, so you need to continually monitor your testing environment. That lets you figure out how to improve it.

Infrastructure monitoring tools enable you to collect and analyze data on your testing environments, as well as on other parts of the infrastructure. Rather than talking about whether it feels like testing is going better, monitoring gives the team data that show whether performance is improving — or deteriorating.

Some well-regarded tools for collecting software release data: Graphite, logstash, Nagios, Splunk

Continuous integration

Continuous integration (CI) is the practice of frequently checking code in with the main code base, triggering automated tests. It's a critical part of continuous delivery. Without continuous integration, you run the risk of discovering that code that ran fine in its own branch collides with code residing in other branches, and breaks the application. The huge advantage here is that bugs get caught while they're small, and easier to trace and fix.

Though continuous integration is really a practice — not a tool — there are certainly plenty of tools available to help you implement that practice. A CI tool regularly checks the version control system for changes to the application, builds the application and runs automated tests on each build. It also provides reports on whether each build passed or failed the tests.

The most commonly used CI tools are: Jenkins, Hudson, Atlassian Bamboo, CruiseControl

Version control

This is the heart of continuous integration. Development teams use version control to keep a record of every version of every feature, add-on or other change to the code base.

Version control is important for everyone on the team, not just developers. It should be used to keep a record of all tests, scripts, documentation and configuration files — in fact, everything to do with your software development work. It should also be used by your operations team to record the configuration of your infrastructure across different environments.

Once a common version control tool is being used by everyone on the team — developers, QA, IT operations — everyone can see the state of the build. It becomes the single source of truth for the entire software workflow. You can also restore anything that you want to roll back, because you have a version of every piece of the build, including system configurations.

A few of the best known version control tools are: Git, Subversion, Perforce, Mercurial

Code review

You need a tool that enables you to step through a proposed change to your codebase and see what the differences are. It allows you to note which changes are acceptable, and which are not, before merging the changes with the main code base.

Some version control tools — **Git**, for example — include the ability to review code. So does **Stash**. **Gerrit** is a free code review tool that integrates with Git. And **GitHub** is, of course, a Git repository hosting service that's also a well-loved collaboration environment.

Configuration management

Continuous delivery requires that the developer's environment, the test environment and the production environment be configured the same way. For example, a new version of your application may require changes to the settings on an Apache server. So you have to make sure those changes are made to the test environment to make the tests meaningful.

In many technical teams, different people (or groups of people) take responsibility for configuring development machines, test servers, and production servers. If these different teams are all using different tools, or hand-configuring their machines, the development, test and production environments won't be consistent.

Inconsistent environments make it very difficult to determine why an application breaks when it's promoted from one environment to the next. Is it because there's a fault in the source code itself? Or is it because each environment is configured differently? Now it's really hard to figure out where the flaw is, and how to fix it. And because you now have to check so much more — the code itself, the environments it runs properly in, and the environment that breaks it — you're eating up time.

A configuration management tool enables you to keep environments consistent throughout the software development process, from the developer's laptop to production. Furthermore, a good configuration management tool will enforce configurations, resetting resources to their correct configurations if someone (or some process) makes a change.

As you test the application at scale — and once you deploy it — there could be hundreds of servers and services that must be configured correctly, and you might be doing it again and again, as you run through iterative changes and tests. A configuration management tool allows you to set the configuration for every resource the application will use, and copy those configurations automatically to more servers, virtual machines, switches, routers and storage servers as you scale.

Configuration management tools based on infrastructurem as code offer a big benefit: the ability to version-control configuration of environments along with the application itself. That allows developers to work in an extremely realistic environment, often identical to the environment that will manage the application in production. Most importantly, any changes to an environment can be immediately reflected across all copies of it, through the use of the version control system — a huge boost for enabling and supporting continuous delivery.

The bigger the difference between development and production environments, the less realistic are the assumptions that have to be made during development. This can be difficult to quantify, but it's a good bet that if you're developing on a Windows machine and deploying to a Solaris cluster, you are in for some surprises.

Continuous Delivery by Jez Humble and Dave Farley

Orchestration

Once your environment is configured, you may need to roll out changes, updates or complete applications in a specific order. That's orchestration. As with any task, automating orchestration vastly reduces the possibility of human error, and makes it possible to scale far beyond what people could do manually.

An application orchestration solution should ensure that the right things happen in the right order on the right machines as the application is deployed, updated or destroyed. It should also leverage a model-driven approach, enabling you to describe the relationship between application components like API services, the data layers behind those, and load balancers serving customer requests.

Dashboards

Continuous delivery isn't a one-person show. The entire team needs to see, from moment to moment, if the build is red (broken) or green (working), and what's actually been released into each environment.

A dashboard should display the status of your test environment and production environment. It should show the status of every node, both physical servers and virtual machines.

Bamboo, Jenkins and Go are all dashboards of choice for many sysadmins.

Summing up continuous delivery

It's all about software quality, faster cycles, and people.

Despite the list of tools above, continuous delivery really isn't a toolset; It's a process that you implement via tools. The purpose of continuous delivery is to deliver less buggy software, and faster, so people can start using it.

Most important of all, continuous delivery is about ongoing collaboration between the people who are part of the software creation and release process. It's about getting everyone to feel like they're on the same team; that their concerns about process and quality are being heard; and to free people from spending nights and weekends at work, when they could be relaxing with family and friends.

Continuous delivery Is...

Incremental. You can implement continuous delivery practices just a step at a time and still get great benefits.

Automated. By making successful processes repeatable, you can make changes more quickly and safely, with less likelihood of error.

Fast. By making frequent releases of smaller changes, you can isolate and fix problems more quickly, or roll back to a previous working state more easily.

You're doing continuous delivery when...

- You can perform push-button deployments of any version of the software to any environment on demand.
- You're leveraging automation, making frequent releases, testing at every stage of the process, and using a pull-based architecture that permits only successful releases to move to the next stage.
- You work together as a single team, with everyone — developer, QA engineer and sysadmin — responsible for delivering quality code.

Puppet Enterprise: Automate and make your life better

At Puppet, we make software that makes peoples' lives easier. We believe ops, dev, and QA are happier and more productive when they work together as a single team to create software they're proud of.

Puppet Enterprise, our flagship product, enables continuous delivery. Its declarative language tells your entire application stack and its underlying services and infrastructure what they should look like and how they should relate to each other. Puppet Enterprise facilitates collaboration between the people who share responsibility for creating great software.

Download and try it out on 10 nodes today - for free.

Appendix: Resources for continuous delivery

There's a wealth of other resources available for anyone who wants to implement continuous delivery — or elements of it — in the workplace. Here are a few additional tools and educational resources that can help.

Desktop workflows for developers

Developers can benefit from extending automation to their own desktop workflows. Here are two tools that can help with that process.

Vagrant. Automates the management and provisioning of development and test environments. It provides a way for sysadmins to share production configurations with developers in a desktop virtual environment, using a tool that's simple enough to manage just that desktop use case. Developers and QA people can also use it easily, eliminating a well-known hassle for sysadmins: trying to get dev and QA up and running on VM solutions, and then keeping them in sync with production.

Bonus: Vagrant lets you test your environment even if your organization doesn't have the resources to set up a full-fledged QA environment.

Boxen. A tool available from GitHub, Boxen allows you to install on your local workstation — usually with a single command — tools like the "hub" Ruby Gem that interacts with your continuous delivery and continuous integration workflows. Similar command-line tools for Jenkins, Vagrant, Amazon Web Services and orchestration tools like MCollective are easy to install with Boxen, too.

Puppet modules

You'll find more than 3,700 modules on the **Puppet Forge** to help you automate just about anything. These are written by both Puppet employees and Puppet community members. Many are helpful for continuous delivery, and a few are highlighted here.

puppetlabs/mcollective installs, configures and manages the MCollective agents, clients and middleware of an MCollective cluster across a range of operating systems and Linux distributions. MCollective powers the orchestration engine in Puppet Enterprise.

fiddyspence/mconotify is a report processor that sends MCollective RPC messages to nodes or classes of nodes to trigger configuration updates for dependent resource sets.

rtyler/jenkins is a plugin for managing the Jenkins continuous integration tool.

maestrodev/maestro_nodes is a plugin for deploying and configuring remote nodes.

maestrodev/maven allows you to download artifacts from a Maven repository.

Educational resources

Git workflows with Puppet and r10k. Puppet segments infrastructure configuration into environments, and r10k maps Git branches to Puppet environments so you can automate your entire workflow — from dev to production to patch, and back again.

Test-driven Infrastructure Development. Tomas Doran of TIM Group talks about how his team developed end-to-end testing of its infrastructure to facilitate continuous deployment.

Bootstrapping Puppet and Application Deployment. Robert de Macedo Soares of BusinessWire discusses solutions to issues people can face when they first launch Puppet across existing heterogeneous servers, based on his team's experience.

Testing for Ops: Going Beyond the Manifest. Christopher Webber of Demand Media talks about the value of rspec-puppet for people from an operations background, including how to test for baseline security, differences between development and production environments, and more.

Continuous delivery in a .NET shop with Puppet Enterprise. Application architect Jason Moorehead details the continuous delivery workflow at IP Commerce, and lists the tools, including Puppet Enterprise. He also talks about the business results (and they're good).

Releasing Puppet: Automating Packaging for Many Platforms. Moses Mendoza and Matthaus Owens, who both work in release engineering at Puppet, show how Puppet has fully automated diverse, difficult packaging workflows.