DevOps Model Defined

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



## How DevOps Works

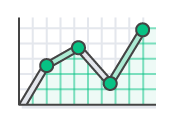
Under a DevOps model, development and operations teams are no longer “siloed.” Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

In some DevOps models, quality assurance and security teams may also become more tightly integrated with development and operations and throughout the application lifecycle. When security is the focus of everyone on a DevOps team, this is sometimes referred to as DevSecOps.

These teams use practices to automate processes that historically have been manual and slow. They use a technology stack and tooling which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks (for example, deploying code or provisioning infrastructure) that normally would have required help from other teams, and this further increases a team’s velocity.

[Learn about AWS DevOps tooling and services »](https://aws.amazon.com/devops/)

## Benefits of DevOps



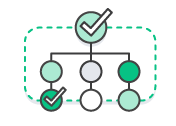
### **Speed**

Move at high velocity so you can innovate for customers faster, adapt to changing markets better, and grow more efficient at driving business results. The DevOps model enables your developers and operations teams to achieve these results. For example, [microservices](https://aws.amazon.com/devops/what-is-devops/" \l "microservices) and [continuous delivery](https://aws.amazon.com/devops/continuous-delivery/) let teams take ownership of services and then release updates to them quicker.



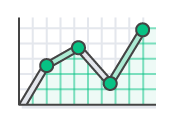
### **Rapid Delivery**

Increase the frequency and pace of releases so you can innovate and improve your product faster. The quicker you can release new features and fix bugs, the faster you can respond to your customers’ needs and build competitive advantage. [Continuous integration](https://aws.amazon.com/devops/continuous-integration/) and [continuous delivery](https://aws.amazon.com/devops/continuous-delivery/) are practices that automate the software release process, from build to deploy.



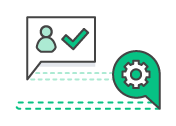
### **Reliability**

Ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users. Use practices like [continuous integration](https://aws.amazon.com/devops/continuous-integration/) and [continuous delivery](https://aws.amazon.com/devops/continuous-delivery/) to test that each change is functional and safe. [Monitoring and logging](https://aws.amazon.com/devops/what-is-devops/#monitoring) practices help you stay informed of performance in real-time.



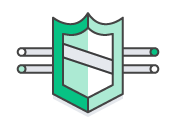
### **Scale**

Operate and manage your infrastructure and development processes at scale. Automation and consistency help you manage complex or changing systems efficiently and with reduced risk. For example, [infrastructure as code](https://aws.amazon.com/devops/what-is-devops/#iac) helps you manage your development, testing, and production environments in a repeatable and more efficient manner.



### **Improved Collaboration**

Build more effective teams under a DevOps cultural model, which emphasizes values such as ownership and accountability. Developers and operations teams [collaborate](https://aws.amazon.com/devops/what-is-devops/#communication) closely, share many responsibilities, and combine their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).



### **Security**

Move quickly while retaining control and preserving compliance. You can adopt a DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and [policy as code](https://aws.amazon.com/devops/what-is-devops/#policyascode), you can define and then track compliance at scale.

## Why DevOps Matters

Software and the Internet have transformed the world and its industries, from shopping to entertainment to banking. Software no longer merely supports a business; rather it becomes an integral component of every part of a business. Companies interact with their customers through software delivered as online services or applications and on all sorts of devices. They also use software to increase operational efficiencies by transforming every part of the value chain, such as logistics, communications, and operations. In a similar way that physical goods companies transformed how they design, build, and deliver products using industrial automation throughout the 20th century, companies in today’s world must transform how they build and deliver software.

## How to Adopt a DevOps Model

## DevOps Cultural Philosophy

Transitioning to DevOps requires a change in culture and mindset. At its simplest, DevOps is about removing the barriers between two traditionally siloed teams, development and operations. In some organizations, there may not even be separate development and operations teams; engineers may do both. With DevOps, the two teams work together to optimize both the productivity of developers and the reliability of operations. They strive to communicate frequently, increase efficiencies, and improve the quality of services they provide to customers. They take full ownership for their services, often beyond where their stated roles or titles have traditionally been scoped by thinking about the end customer’s needs and how they can contribute to solving those needs. Quality assurance and security teams may also become tightly integrated with these teams. Organizations using a DevOps model, regardless of their organizational structure, have teams that view the entire development and infrastructure lifecycle as part of their responsibilities.

## DevOps Practices Explained

There are a few key practices that help organizations innovate faster through automating and streamlining the software development and infrastructure management processes. Most of these practices are accomplished with proper tooling.

One fundamental practice is to perform very frequent but small updates. This is how organizations innovate faster for their customers. These updates are usually more incremental in nature than the occasional updates performed under traditional release practices. Frequent but small updates make each deployment less risky. They help teams address bugs faster because teams can identify the last deployment that caused the error. Although the cadence and size of updates will vary, organizations using a DevOps model deploy updates much more often than organizations using traditional software development practices.

Organizations might also use a microservices architecture to make their applications more flexible and enable quicker innovation. The microservices architecture decouples large, complex systems into simple, independent projects. Applications are broken into many individual components (services) with each service scoped to a single purpose or function and operated independently of its peer services and the application as a whole. This architecture reduces the coordination overhead of updating applications, and when each service is paired with small, agile teams who take ownership of each service, organizations can move more quickly.

However, the combination of microservices and increased release frequency leads to significantly more deployments which can present operational challenges. Thus, DevOps practices like continuous integration and continuous delivery solve these issues and let organizations deliver rapidly in a safe and reliable manner. Infrastructure automation practices, like infrastructure as code and configuration management, help to keep computing resources elastic and responsive to frequent changes. In addition, the use of monitoring and logging helps engineers track the performance of applications and infrastructure so they can react quickly to problems.

Together, these practices help organizations deliver faster, more reliable updates to their customers. Here is an overview of important DevOps practices.

## What is DevOps Not?

### It’s Not NoOps

It is not “they’re taking our jobs!”  Some folks thinks that DevOps means that developers are taking over operations and doing it themselves.  Part of that is true and part of it isn’t.

It’s a misconception that DevOps is coming from the development side of the house to wipe out operations – DevOps, and its antecedents in agile operations, are being initiated out of operations teams more often than not.  This is because operations folks (and I speak for myself here as well) have realized that our existing principles, processes, and practices have not kept pace with what’s needed for success.  As businesses and development teams need more agility as the business climate becomes more fast paced, we’ve often been providing less as we try to solve our problems with more rigidity, and we need a fundamental reorientation to be able to provide systems infrastructure in an effective manner.

Now, as we realize some parts of operations need to be automated, that means that either we ops people do some automation development, or developers are writing “operations” code, or both.  That is scary to some but is part of the value of the overall collaborative approach. All the successful teams I’ve run using this approach have both people with deep dev skill sets and deep ops skill sets working together to create a better overall product. And I have yet to see anyone automate themselves out of a job in high tech – as lower level concerns become more automated, technically skilled staff start solving the higher value problems up one level.

### It’s Not (Just) Tools

DevOps is also not simply implementing a set of tools.  One reason why I feel that a more commonly accepted definition of DevOps is needed is that having various confusing and poorly structured definitions increases the risk that people will pass by the “theory” and implement the processes or tools of DevOps without the principles in mind, which is definitely an antipattern. Automation is just the exercise of power, and unwise automation can do as much damage as wise automation can bring benefit.

Similarly, Agile practitioners would tell you that just starting to work in iterations or adopting other specific practices without initiating meaningful collaboration is likely to not work out real well. There are some teams at companies I’ve worked for that adopted some of the methods and/or tools of agile but not its principles, and the results were suboptimal. Sure, a tool can be useful in Agile (or DevOps), but if you don’t know how to use it then it’s like giving an assault weapon to an untrained person.

But in the end, fretting about “tools shouldn’t be called DevOps” is misplaced. Is poker planning “agile” in the sense that doing it magically gets you Agile?  No.  But it is a common tool used in various agile methodologies, so calling it an “agile tool” is appropriate. Similarly, just because DevOps is not just a sum of the tools doesn’t mean that tools specifically designed to run systems in accordance with a DevOps mindset aren’t valuable. (There are certainly a bunch of tools I’ve used that seem specifically designed to prevent it!)

### It’s Not (Just) Culture

Many people insist that DevOps “is just culture” and you can’t apply the word to a given principle or practice, but I feel like this is overblown and incorrect. Agile has not helped thousands of dev shops because the work on it stopped at “culture,” with admonitions to hug coworkers and the lead practitioners that identified the best practices simply declaring it was all self-evident and refusing to be any more prescriptive. (Though there is some of that). DevOps consists of items at all the levels I list above, and is largely useless without the tangible body of practice that has emerged around it. You might be able to figure out all those best practices yourself given the cultural direction and lots of time to experiment, but sharing information is why we have the Internet (and printing press for that matter).

### It’s Not (Just) Devs and Ops

And in the end, it’s not exclusionary.  Some people have complained “What about security people!  And network admins!  Why leave us out!?!”  The point is that all the participants in creating a product or system should collaborate from the beginning – business folks of various stripes, developers of various stripes, and operations folks of various stripes, and all this includes security, network, and whoever else.  There’s a lot of different kinds of business and developer stakeholders as well; just because everyone doesn’t get a specific call-out (“Don’t forget the icon designers!”) doesn’t mean that they aren’t included.   The original agile development guys were mostly thinking about “biz + dev” collaboration, and DevOps is pointing out issues and solutions around “dev + ops” collaboration, but the mature result of all this is “everyone collaborating”. In that sense, DevOps is just a major step for one discipline to join in on the overall culture of agile collaboration that should involve all disciplines in an organization. So whoever is participating in the delivery of the software or service is part of DevOps.

### It’s Not (Just) A Job Title

Simply taking an existing ops team and calling them “The DevOps Team” doesn’t actually help anything by itself.  Nor does changing a job title to “DevOps Engineer.” If you don’t adopt the values and principles above, which require change at an overall system level not simply within a given team, you won’t get all the benefits.

However, I’m not in the camp that rails that you ‘can’t have DevOps in a job title.” It is often used in a job title as a way to distinguish “new style DevOps-thinking, automation-first, dev-collaborating, CI-running, etc. sysadmin” from “grouchy back room person who aggressively doesn’t care what your company does for a living.” Some people find value in that, others don’t, and that’s fine. As a hiring manager myself, I see a clear difference in the fit of applicants when I put it on a job posting for a systems engineer, which provides an incentive for me to keep doing so…

### It’s Not Everything

Sometimes, DevOps people get carried away and make grandiose claims that DevOps is about “everything everywhere!” Since DevOps plugs into the overall structure of a lot of lean and agile thinking, and there are opportunities for that kind of collaboration throughout an organization, it’s nice to see all the parallels, but going and reengineering your business processes isn’t really DevOps per se.  It is part of an overall, hopefully collaborative and agile corporate culture, but DevOps is specifically about how operations plugs into that.  Some folks overreach and end up turning DevOps into a super watered down version of Lean, Agile, or just love for everyone. Which is great at the vision level, but as you march down the hierarchy of granularity, you end up mostly dealing with operational integration – other efforts are worrying about the other parts (you can personally too of course). But there are still a lot of unsolved problems around the delivery of software and maintenance of services and making it fast, reliable, secure, et al. – if someone wants to use what they’ve learned from DevOps to go be a larger scope corporate consultant that’s fine, but most people involved in DevOps are technical practitioners who are looking for better ways to do their job, not someone else’s.  In Agile there is “Agile Software Development” and then there’s the larger Agile organization work. I think DevOps is best defined as “Agile Software Delivery and Operations,” which should similarly work in concert with others working on larger organizational initiatives, but without losing sight of its primary value proposition for the organization.

## Full-stack developers are using DevOps tools

I’ve always seen them as manifestations of similar ideas in different technical areas. However, when you look at the data we’ve collected in our survey, alongside some wider research, the relationship between the DevOps engineer and the Full-Stack developer might possibly be more than purely conceptual. ‘Full-Stack’ and ‘DevOps’ are both terms that blur the lines between developer and engineer, and both are two sides of an intriguing form of cross-pollination; technologies more commonly used for deployment and automation. [Docker](https://subscription.packtpub.com/tech/docker" \o "Docker eBooks, Courses & Videos" \t "_blank) and Vagrant were the most notable, highlighting the impact of [containerization](https://www.packtpub.com/bundles/containerization) and virtualization on [web development](https://www.packtpub.com/web-development), but we also found a number of references to the Microsoft automation tool PowerShell – a distinctly DevOps-esque tool if ever there was one.

Perhaps there’s a danger of overstating my point – surely we shouldn’t be surprised if web developers are using these tools – it’s not that strange, right? Maybe, but the fact that tools such as these are being used by web developers in their day-to-day work suggests that they are no longer simply expected to develop: they also need to deploy and configure their projects. Indeed, it’s worth noting that across all our web development respondents, a large number plan on learning [Docker](https://subscription.packtpub.com/tech/docker" \o "Docker eBooks, Courses & Videos" \t "_blank) over the next 12 months.

## DevOps engineers use a huge range of tools

DevOps Engineers were even more eclectic in their tool-usage than full-stack developers.[Python](https://subscribe.packtpub.com/learn-python/) is the language of-choice and Puppet the go-to configuration management tool, but web tools such as [JavaScript](https://www.packtpub.com/tech/JavaScript) and [PHP](https://www.packtpub.com/tech/PHP) are also being used. References to Flask, for example, the [Python](https://subscribe.packtpub.com/learn-python/) microframework, emphasise the way in which DevOps Engineers have an eye on web development while they’re automating your infrastructure.

Taken alone, these responses might not fully evidence the relationship between DevOps engineers and Full-Stack developers. However, there are jobs out there asking for a combination of both skillsets. One, posted by a recruiter working for a nameless ‘creative media house’ in London, was looking for someone to become ‘a key member of multi-party cloud research projects, helping to bring a microservices-based video automation system to life, integrate development and developed systems into onside and global infrastructure’. The tools being asked for were very varied indeed. From a high-level language, such as[JavaScript](https://www.packtpub.com/tech/JavaScript), to scripting languages such as Bash, Python and Perl, to continuous integration tools,[configuration management tools](https://www.packtpub.com/bundles/configmanagement) and containerization technologies, whoever eventually gets the job certainly deserves to be called a polyglot.

## Blurring the line between full-stack and DevOps

A further indication of the blurred line between engineers and developers can be found in [this article from computing.co.uk](http://www.computing.co.uk/ctg/feature/2414761/the-rise-of-devops-necessitates-full-stack-developers-at-the-washington-post). It’s an interesting tale of how working practices develop according to necessity and how methodologies and ideas interact with the practical details of a given situation. It tells the story of how the Washington Post went about building its submission platform, and how the way in which the project was resourced and managed changed according to certain pressures – internal and external. The title might actually be misleading – if you read it, it’s not so much that DevOps necessitates full-stack development, more that each thing grows out of the next. It might even be said that the reverse is true – that full-stack development necessitates DevOps thinking.

The relationship between DevOps and full-stack development gives a real indication of the state of the tech world in 2015. Within a tech landscape of increasing complexity and cross-pollination there are going to be opportunities for developers and engineers to significantly drive their value as technical professionals. It’s simply a question of learning more, and of being open to new challenges and ideas about how to work effectively. It probably won’t be easy, but it might just be a fun journey.

**best DevOps tools for 2019**

**1. Gradle**

Your DevOps tool stack will need a reliable build tool. Apache Ant and Maven dominated the automated build tools market for many years, but [Gradle](https://gradle.org/) showed up on the scene in 2009, and its popularity has steadily grown since then. Gradle is an incredibly versatile tool which allows you to write your code in Java, C++, Python, or other languages. Gradle is also supported by popular IDEs such as Netbeans, Eclipse, and IntelliJ IDEA. If that doesn’t convince you, it might help to know that Google also chose it as the [official build tool](https://developer.android.com/studio/build/) for Android Studio.

While Maven and Ant use XML for configuration, Gradle introduces a Groovy-based DSL for describing builds. In 2016, the Gradle team also released a [Kotlin-based DSL](https://github.com/gradle/kotlin-dsl), so now you can write your build scripts in Kotlin as well. This means that Gradle does have some learning curves, so it can help a lot if you have used Groovy, Kotlin or another JVM language before. Besides, Gradle uses Maven’s repository format, so dependency management will be familiar if you have prior experience with Maven. You can also [import your Ant builds](https://docs.gradle.org/current/userguide/ant.html)into Gradle.

The best thing about Gradle is [incremental builds](https://blog.gradle.org/introducing-incremental-build-support), as they save a nice amount of compile time. According to Gradle’s [performance measurements](https://gradle.org/gradle-vs-maven-performance/), it’s up to 100 times faster than Maven. This is in part because of incrementality, but also due to Gradle’s [build cache](https://blog.gradle.org/introducing-gradle-build-cache) and [daemon](https://docs.gradle.org/current/userguide/gradle_daemon.html). The build cache reuses task outputs, while the Gradle Daemon keeps build information hot in memory in-between builds.

All in all, Gradle allows faster shipping and comes with a lot of configuration possibilities.

**2. Git**

[Git](https://git-scm.com/) is one of the most popular DevOps tools, widely used across the software industry. It’s a distributed SCM (source code management) tool, loved by remote teams and open source contributors. Git allows you to track the progress of your development work. You can save different versions of your source code and return to a previous version when necessary. It’s also great for experimenting, as you can create separate branches and merge new features only when they’re ready to go.

To integrate Git with your DevOps workflow, you also need to host repositories where your team members can push their work. Currently, the two best online Git repo hosting services are [GitHub](https://github.com/) and [Bitbucket](https://bitbucket.org/). GitHub is more well-known, but Bitbucket comes with free unlimited private repos for small teams (up to five team members). With GitHub, you get access only to public repos for free—which is still a great solution for many projects.

Both GitHub and Bitbucket have fantastic integrations. For example, you can integrate them with Slack, so everyone on your team gets notified whenever someone makes a new commit.

**3. Jenkins**

[Jenkins](https://jenkins.io/) is the go-to DevOps automation tool for many software development teams. It’s an open source CI/CD server that allows you to automate the different stages of your delivery pipeline. The main reason for Jenkins’ popularity is its huge plugin ecosystem. Currently, it offers [more than 1,000 plugins](https://plugins.jenkins.io/), so it integrates with almost all DevOps tools, from Docker to Puppet.

With Jenkins, you can set up and customize your CI/CD pipeline according to your own needs. I found the following example in the [Jenkins Docs](https://jenkins.io/doc/book/pipeline/). And, this is just one of the possibilities. Nice, isn’t it?

It’s easy to [get started with Jenkins](https://jenkins.io/download/), as it runs out-of-the-box on Windows, Mac OS X, and Linux. You can also easily [install it with Docker](https://wiki.jenkins.io/display/JENKINS/Installing+Jenkins+with+Docker). You can set up and configure your Jenkins server through a web interface. If you are a first-time user, you can choose to install it with frequently used plugins. However, you can create your own custom config as well.

With Jenkins, you can iterate and deploy new code as quickly as possible. It also allows you to measure the success of each step of your pipeline. I’ve heard people complaining about Jenkins’ “ugly” and non-intuitive UI. However, I could still find everything I wanted without any problem.

**4. Bamboo**

[Bamboo](https://www.atlassian.com/software/bamboo) is Atlassian’s CI/CD server solution that has many similar features to Jenkins. Both are popular DevOps tools that allow you to automate your delivery pipeline, from builds to deployment. However, while Jenkins is open source, Bamboo comes with a price tag. So, here’s the eternal question: is it worth choosing proprietary software if there’s a free alternative? It depends on your budget and goals.

Bamboo has many pre-built functionalities that you have to set up manually in Jenkins. This is also the reason why Bamboo has fewer plugins (around 100 compared to Jenkins’ 1000+). In fact, you don’t need that many plugins with Bamboo, as it does many things out-of-the-box.

Bamboo seamlessly integrates with other Atlassian products such as Jira and Bitbucket. You also have access to built-in Git and Mercurial branching workflows and test environments. All in all, Bamboo can save you a lot of configuration time. It also comes with a more intuitive UI with tooltips, auto-completion, and other handy features.

**5. Docker**

[Docker](https://www.docker.com/) has been the number one container platform since its launch in 2013 and continues to improve. It’s also thought of as one of the most important DevOps tools out there. Docker has made containerization popular in the tech world, mainly because it makes distributed development possible and automates the deployment of your apps. It isolates applications into separate containers, so they become portable and more secure. Docker apps are also OS and platform independent. You can use Docker containers instead of virtual machines such as VirtualBox.

What I like the most about Docker is that you don’t have to worry about dependency management. You can package all dependencies within the app’s container and ship the whole thing as an independent unit. Then, you can run the app on any machine or platform without a headache.

Docker integrates with [Jenkins](https://jenkins.io/solutions/docker/) and [Bamboo](https://confluence.atlassian.com/bamboo/getting-started-with-docker-and-bamboo-687213473.html), too. If you use it together with one of these automation servers, you can further improve your delivery workflow. Besides, Docker is also great for cloud computing. In recent years, all major cloud providers such as AWS and Google Cloud added support for Docker. So, if you are planning a cloud migration, Docker can ease the process for you.

**6. Kubernetes**

This year, everyone is talking about [Kubernetes](https://kubernetes.io/). It’s a container orchestration platform that takes containerization to the next level. It works well with Docker or any of its alternatives. Kubernetes is still very new; its first release came out in 2015. It was founded by a couple of Google engineers who wanted to find a solution to manage containers at scale. With Kubernetes, you can group your containers into logical units.

You may not need a container orchestration platform if you have just a few containers. However, it’s the next logical step when you reach a certain level of complexity and need to scale your resources. Kubernetes allows you to automate the process of managing hundreds of containers.

With Kubernetes, you don’t have to tie your containerized apps to a single machine. Instead, you can deploy it to a cluster of computers. Kubernetes automates the distribution and scheduling of containers across the whole cluster.

A Kubernetes cluster consists of one master and several worker nodes. The master node implements your pre-defined rules and deploys the containers to the worker nodes. Kubernetes pays attention to everything. For instance, it notices when a worker node is down and redistributes the containers whenever it’s necessary.

**7. Puppet Enterprise**

[Puppet Enterprise](https://puppet.com/products/puppet-enterprise) is a cross-platform configuration management platform. It allows you to manage your infrastructure as code. As it automates infrastructure management, you can deliver software faster and more securely. Puppet also provides developers with an [open-source tool](https://puppet.com/download-open-source-puppet) for smaller projects. However, if you are dealing with a larger infrastructure, you may find value in Puppet Enterprise’s [extra features](https://puppet.com/products/why-puppet/puppet-enterprise-and-open-source-puppet), such as:

* Real-time reports
* Role-based access control
* Node management

With Puppet Enterprise, you can manage multiple teams and thousands of resources. It automatically understands relationships within your infrastructure. It deals with dependencies and handles failures smartly. When it encounters a failed configuration, it skips all the dependent configurations as well. The best thing about Puppet is that it has [more than 5,000 modules](https://forge.puppet.com/) and integrates with [many popular DevOps tools](https://puppet.com/products/managed-technology).

**8. Ansible**

[Ansible](https://www.ansible.com/) is a configuration management tool, similar to Puppet and Chef. You can use it to configure your infrastructure and automate deployment. Its main selling points compared to other similar DevOps tools are simplicity and ease of use. Ansible follows the same Infrastructure As Code (IAC) approach as Puppet. However, it uses the super simple YAML syntax. With Ansible, you can define tasks in YAML, while Puppet has its own declarative language.

Agentless architecture is another frequently mentioned feature of Ansible. As no daemons or agents run in the background, Ansible is a secure and lightweight solution for configuration management automation. Similar to Puppet, Ansible also has [several modules](https://docs.ansible.com/ansible/latest/modules/modules_by_category.html).

If you want to better understand how Ansible fits into the DevOps workflow take a look at [this post](https://www.redhat.com/en/blog/integrating-ansible-jenkins-cicd-process) by the Red Hat Blog. It shows how to use Ansible for environment provisioning and application deployment within a Jenkins pipeline.

**9. Nagios**

[Nagios](https://www.nagios.org/) is one of the most popular free and open source DevOps monitoring tools. It allows you to monitor your infrastructure so that you can find and fix problems. With Nagios, you can keep records of events, outages, and failures. You can also keep an eye on trends with the help of Nagios’ graphs and reports. This way, you can forecast outages and errors and detect security threats.

Although there are many DevOps tools for infrastructure monitoring, Nagios stands out due to its rich [plugin ecosystem](https://exchange.nagios.org/). As Nagios has been around for a while (since 2002), there’s a vast community around it. Besides plugins, they also make add-ons, tutorials, translations, and other goodies—all for free.

Nagios offers four open source monitoring solutions:

1. Nagios Core
2. Nagios XI
3. Nagios Log Server
4. Nagios Fusion

**Nagios Core** is a command line tool, with all the basic functionalities. You can also opt for **Nagios XI** that comes with a web-based GUI and monitoring wizard. Nagios writes a handy [comparison of their capabilities](https://www.nagios.org/downloads/nagios-core/).

**Nagios Log Server** lets you search log data and set up alerts about potential threats. And, **Nagios Fusion**allows you to monitor multiple networks at the same time.

On the whole, Nagios provides DevOps teams with an infrastructure monitoring solution. However, it can take a while to set it up and make it compatible with your environment.

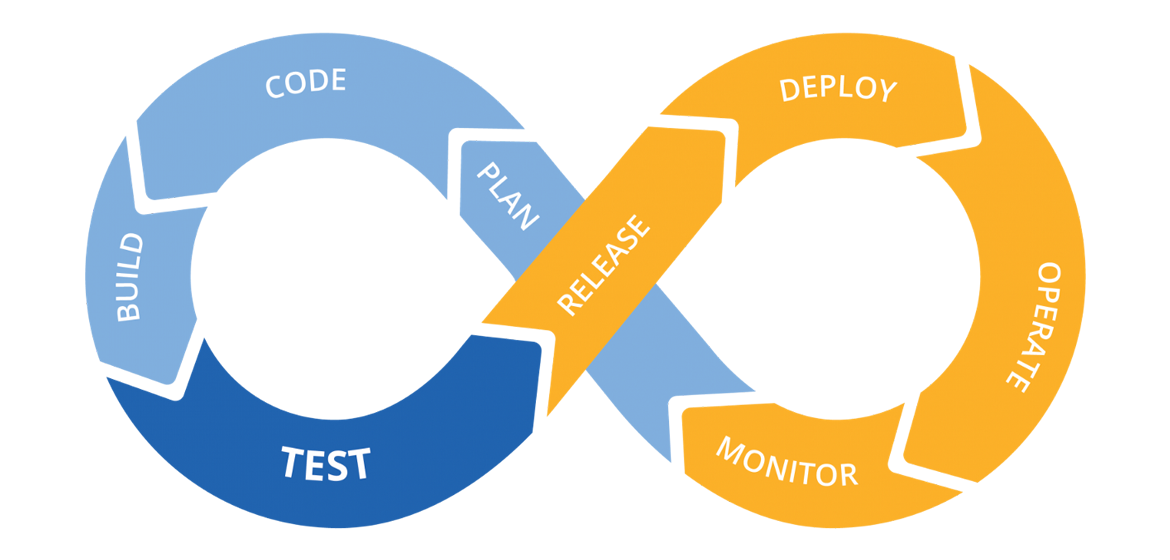
**10. Raygun**

[Raygun](https://raygun.com/) is a world-class error monitoring and crash reporting platform. [Application performance monitoring](https://raygun.com/platform/apm)(APM) is its most recent product. Raygun’s DevOps tool helps you diagnose performance issues and tracking them back to the exact line of code, function, or API call. The APM tool also fits well with Raygun’s error management workflow. For example, it automatically identifies your highest priority problems and creates issues for you.

Raygun APM can help you make the most out of other DevOps tools, as you are always notified about the problems. Since it automatically links errors back to the source code, Raygun brings Development and Operations together by providing one source of truth for the whole team the cause of errors and performance problems.

# **An Introduction To CI/CD**

The adoption of CI/CD has changed how developers and testers ship software. This blog is about [this transition to CI/CD pipelines](https://www.mabl.com/testing-in-the-development-phase-cicd-pipeline) that will provide insights into different tools and process changes which can help developers be more successful. Developer approaches are always changing: long ago, we had [Waterfall](https://airbrake.io/blog/sdlc/waterfall-model), then it was [Agile](https://www.seguetech.com/waterfall-vs-agile-methodology/), and now it's [DevOps](https://theagileadmin.com/what-is-devops/" \t "_blank). DevOps is how modern developers are building great products. The new methods of Continuous Integration, Continuous Delivery (CI/CD), and Continuous Deployment have come with the rise of DevOps. Conventional software development and delivery methods are rapidly becoming obsolete as deployment frequency increases.



How the CI/CD pipeline flows 

Before DevOps, most companies would deploy/ship software in monthly, quarterly, bi-annual, or even annual releases (also know as the Agile days). In the DevOps era, weekly, daily, and even multiple daily deployments are the norm. With SaaS taking over the development world, you can easily update applications on the fly without forcing customers to download new components. If your CI/CD pipeline is working right, they shouldn’t even realize an update has happened.

Development teams have adapted to the shortened delivery cycles by embracing automation across their software delivery pipeline. Most teams have automated processes to check in code and deploy to new environments. These adaptations have birthed the CI/CD process and have been coupled with a focus on automating the testing process, which we cover [in this article on unit testing](https://www.mabl.com/blog/unit-testing-in-the-development-phase-of-the-cicd-pipeline). Today, we’ll cover what CI/CD is and how modern software companies are using CI/CD tools to automate the process of shipping new code.

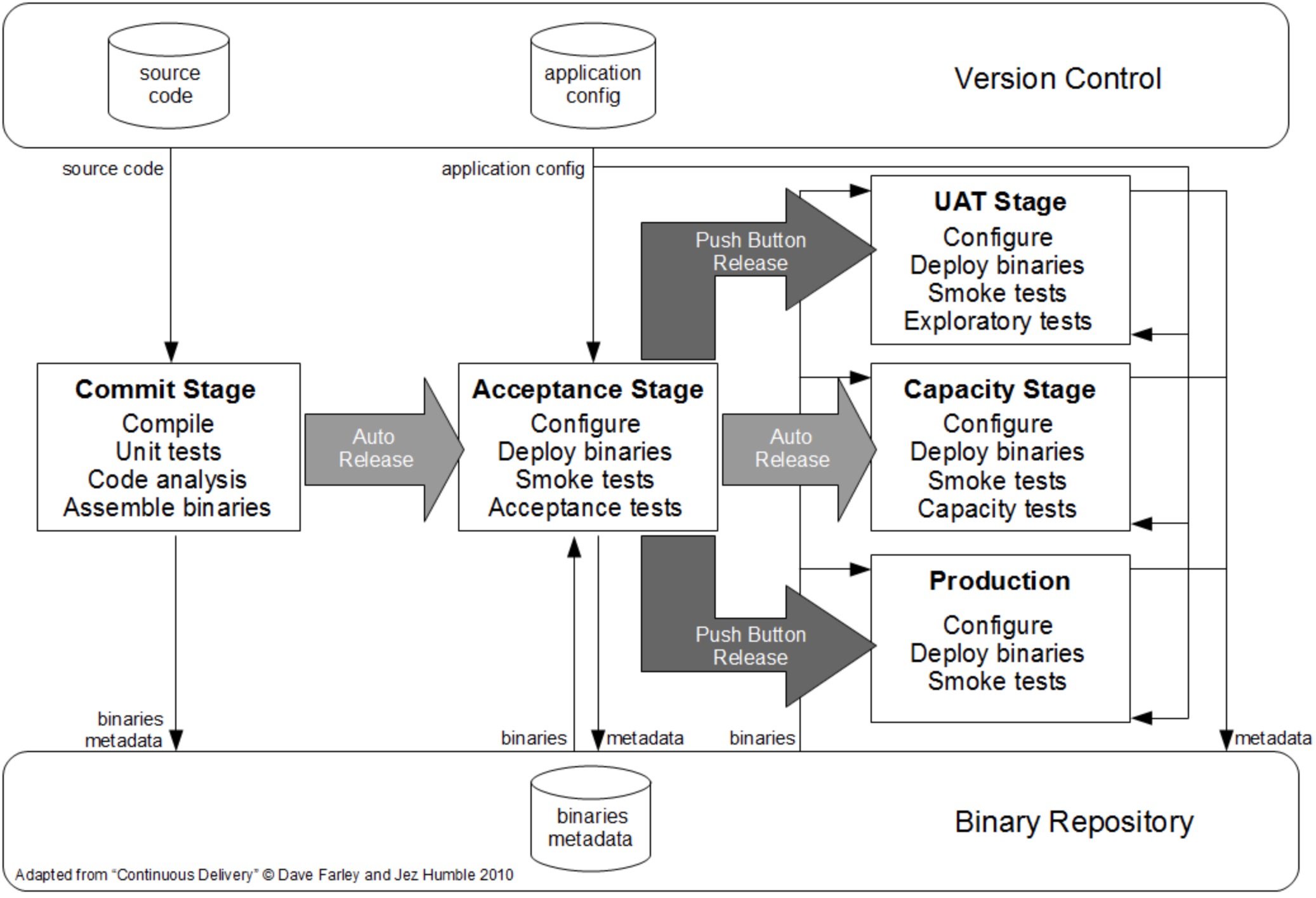
# The Terms

**Continuous integration (CI)**focuses on blending the software work products of individual developers together into a repository. This can be done several times a day, with the primary purpose being to enable early detection of integration bugs while also allowing for tighter cohesion and more development collaboration.

The aim of **continuous delivery (CD)**is to minimize the friction points that are inherent in the deployment or release processes. Typically, a team's implementation involves automating each of the steps for build deployments so that a safe code release can be done at any moment in time.

***Continuous deployment***is a higher degree of automation, in which a build/deployment occurs automatically whenever a major change is made to the code.

Each of these stages is part of a **deployment (or development) pipeline**. In their book [***Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation***](https://www.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/)**,** Jez Humble and David Farley explain how every change to your software[“](http://www.informit.com/articles/article.aspx?p=1621865&seqNum=2)goes through a complex process on its way to being released.[That process involves building the software, followed by the progress of these builds through multiple stages of testing and deployment.](http://www.informit.com/articles/article.aspx?p=1621865&seqNum=2)This, in turn, requires collaboration between many individuals and possibly several teams.[The deployment pipeline is a model of this process, and its incarnation in a continuous integration and release management tool is what allows you to see and control the progress of each change as it moves from version control through various sets of tests and deployments to release to users.”](http://www.informit.com/articles/article.aspx?p=1621865&seqNum=2)



A basic deployment pipeline 

# **Continuous Integration (CI)**

When practicing continuous integration, developers frequently integrate their code into a main branch of a common repository. Rather than building features in isolation and submitting each of them at the end of the cycle, a developer is able to [contribute software work products to the repository](https://www.atlassian.com/git/tutorials/using-branches/git-merge) several times on any given day.

The main idea with CI is to reduce integration costs by having developers do it more frequently and sooner than they normally would. In practice, a developer will often discover boundary conflicts between new and existing code at the time of integration. If it’s done early and often, the expectation is that resolving these conflicts will be easier and less costly to perform.

Of course, there are trade offs; namely, this process change does not provide any additional quality assurances. Many organizations find that such an integration can become more costly in terms of time [since they rely on manual procedures to ensure that new code doesn’t introduce new bugs and doesn’t break existing code](https://fortegrp.com/blog/quality-assurance-and-testing/continuous-delivery-world-is-there-a-place-for-manual-testing/). To reduce friction during integration tasks, continuous integration relies on test suites and an automated test execution. It’s important to realize that automated testing is quite different from continuous testing.

The goal of CI is to refine integration into a simple, easily-repeatable everyday development task that reduces overall build costs and reveals defects early in the cycle. Success with CI will depend on the culture of the development team, specifically if there is incentive for frequent and iterative builds and an eagerness to deal with bugs when they are found frequently. You may have to make necessary cultural changes to the team to ensure these facets are sustainable.

# **Continuous Delivery (CD)**

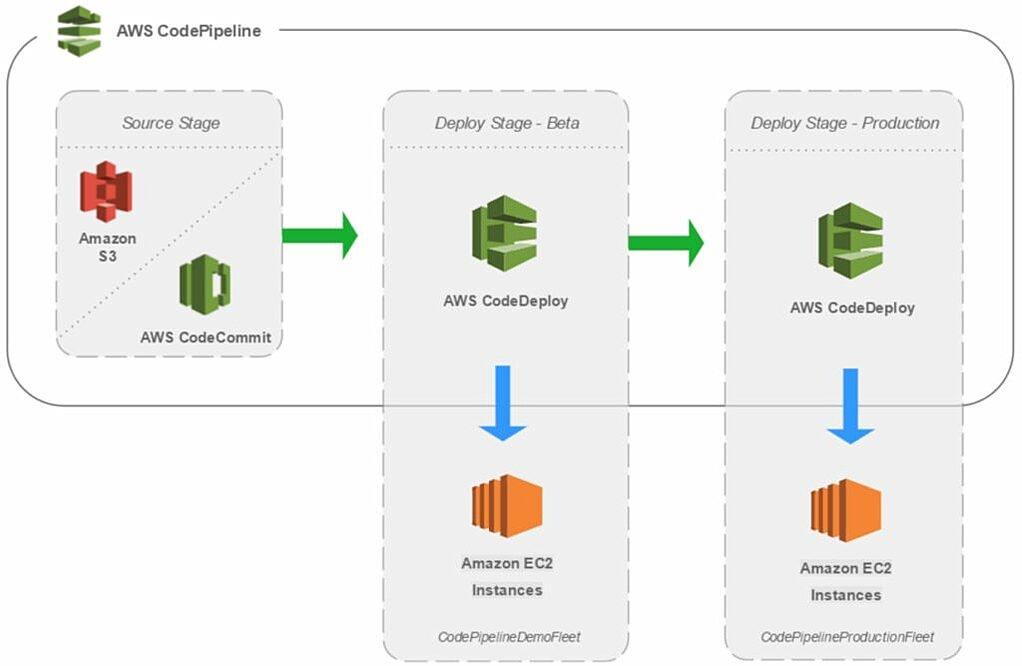
Continuous delivery is actually an extension of CI, in which the software delivery process is automated further to enable easy and confident deployments into production at any time.

A mature continuous delivery process exhibits a codebase that is always deployable. With CD, software release becomes a routine and no frills event without anxiety or urgency. Teams are able to proceed with daily development tasks with the confidence that they can build a production-grade release ready to be deployed at any time without elaborate orchestration or special late-game testing.

CD depends on a central deployment pipeline in which the team automates the testing and deployment processes. This pipeline is an automated system that executes a progressive set of test suites against the build. CD is highly automatable, and, in some cloud-computing environments, easily configurable.

In each segment of the pipeline, the build may fail a critical test, in which case the pipeline will alert the team. Otherwise, the build continues on to the next test suite, with successive test passes resulting in automatic promotion to the next segment in the pipeline. The last segment in the pipeline will deploy the build to a production-equivalent environment. This is a comprehensive activity, since the build, the deployment, and the environment are all exercised and tested together. The result is a build that is confidently deployable and verifiable in an actual production environment.

A solid exhibit of a modern CI/CD pipeline can be seen with AWS. Amazon is one of the cloud-computing providers that offers an impressive [CI/CD pipeline environment](https://www.mabl.com/blog/how-to-setup-ci/cd-with-aws-codepipeline), where you can choose from among its many development resources and link your choices together in a pipeline that is readily configurable and easily monitored.



Many consider continuous delivery attractive primarily because it automates all the steps from submitting code into the repository to releasing fully-tested, properly-functional builds that are ready for production. CD is an elaborate automation of the build and testing processes, but decisions about when, how, and what should be released remain a manual process. Continuous deployment can free up time for those discussions by automating all the other steps.

# **Continuous Deployment**

Continuous deployment extends continuous delivery so that the software build will automatically deploy if it passes all tests. In such a process, there is no need for a person to decide when and what goes into production. The last step in a CI/CD system with continuous deployment will automatically deploy whatever build components/packages successfully exit the delivery pipeline. Such automatic deployments can be configured to quickly distribute components, features, and fixes to customers, and provide clarity on precisely what has been pushed to production.

Organizations that employ continuous deployment will benefit immensely from the ability of users to give quick feedback on new deployments. Quick user response on unhelpful or misunderstood features will help the team refocus and avoid devoting more effort into functional areas that are unlikely to produce a good return on their investment. However, as features are being quickly delivered to users, any defects that become evident should be handled promptly or the team risks getting overloaded with trying to fix the latest bugs and release new features.

# **CI/CD Tools**

With the move to DevOps, there has been a surge of new automation tools to help with the CI/CD pipeline. These automation tools typically integrate with various established popular developer tools, including code repository systems like GitHub and bug tracking systems like Jira. As SaaS has become a more popular delivery model, many of these tools are running in the cloud, the same place where many modern developers are running their apps ([including the ones at mabl!](https://www.mabl.com/blog/why-we-chose-google-cloud-platform-over-aws-at-mabl)).

The most popular automation tool is [Jenkins](https://jenkins.io/) (formerly Hudson), which is an open source project supported by hundreds of contributors as well as a commercial company, [Cloudbees](https://www.cloudbees.com/" \t "_blank). Cloudbees even offers [several different Jenkins training programs](https://www.cloudbees.com/jenkins/training) and [product add-ons](https://go.cloudbees.com/plugins/).

Besides open source projects, there are several modern commercial software automation products available including [CircleCI](https://circleci.com/" \t "_blank), [Codeship](https://codeship.com/" \t "_blank), and [Shippable](https://www.shippable.com/). Each of these has several different advantages and disadvantages for specific workflows. To really understand which will work for you, I’d encourage trying each of them specifically within your developer workflow to see how they work in your environment (how they work with your tools, your cloud platform, your container system, etc.).

We build mabl on Google Cloud Platform, so we were looking for a product that was compatible and preferably integrated with GCP. We took a look at CircleCI, Codeship, and Shippable to compare their strengths and weaknesses:

# **9 things I wish I knew about CI/CD pipelines during first year of my start-up**

We’ve been installing some upgrades to our release pipeline here at Test Collab. Such upgrades give you a good opportunity to find your past mistakes and eliminate them altogether. While doing so, I thought it’ll be good to share what I learned about continuous integration during lifetime of Test Collab and before.

Continuous Integration is a key element of your software. If not planned carefully and correctly, you’ll most likely have higher risk exposure to bad releases and bugs, resulting more costs on your business.

CI is a big investment but good for us that you can fix your screw ups easily. So here are some of the things I learned in past few years about CI/CD pipelines, feel free to agree, disagree or share your comments below (the list is randomly ordered):

# **1. Tools are temporary, do not get tied up to one tool**

Avoid marrying a tool for tools are temporary. Well ideally speaking, CI tool should be.

Why? It might not seem much now, but you’ll be stuck for years with that old tool because migrating to a new one is huge job.

Storing long shell commands in Jenkins? You’re doing it wrong!

Instead, try:

Create small .sh files or .bat files, commit to your repo and call them when needed. Use environment variables for saving credentials.

This’ll help you to migrate to new tools very easily. Not only that, but you can run these files anywhere to carry out small tests.

# **2. Save build artifacts on a remote location (disc space finishes up really really fast)**

If you’re spending a minute cleaning disc space, you did something wrong.

See what Joel Spolsky [wrote](https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/) about disc space 17 years ago:

“

At my last job, the system administrator kept sending me automated spam complaining that I was using more than … get this … 220 megabytes of hard drive space on the server. I pointed out that given the price of hard drives these days, the cost of this space was significantly less than the cost of the toilet paper I used. Spending even 10 minutes cleaning up my directory would be a fabulous waste of productivity.

https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/

Today with platforms like AWS you never have to worry about disc space. Your artifacts should go automatically to a cloud storage service or your enterprise server directly instead of filling up your CI server.

# **3. Package first, then run jobs**

We made this mistake while building our first pipeline few years ago. It was a four-stage pipeline, and packages were generated late in pipeline, as late as third phase. As a result, we couldn’t trigger automated jobs and manually test builds until whole one hour was passed.

So, lesson learned? Generate packages first.

Sooner the packages are ready, dependent jobs can be called upon in any order.

Exception here is unit testing. So packaging job should:

run your unit tests -> package -> publish to private space

# **4. Use separate VM or containers for each job**

Not doing so will cost you lot of debugging hours, it’ll fail your build randomly and will leave you with unexpected unseen bugs, resolving which will be equivalent to working Sherlock Holmes’ cases.

What really happens if you don’t do it?

Your previously ran jobs alters a lot of stuff on system: packages, log files, system settings and what not. Maybe more stuff than you can keep track of.

So it is not possible to have a clean slate for a new job unless you’re using a fresh VM or a container. These modifications conflict with future builds and produces unexpected issues or random failures.

In our case, this mistake might’ve been the biggest expense from this list. Good thing is today we have [so](https://semaphoreci.com/) [many](http://circleci.com/) [build](https://app.shippable.com/) [tools](http://codeship.com/) that enforces you to have separate containers for each build from day 1 and they’re really easy to implement too.

# **5. Parallel jobs are good, but implement only when needed**

When I read [Continuous Delivery](https://www.amazon.com/Continuous-Delivery-Deployment-Automation-Signature/dp/0321601912) for the first time (read it twice actually – it’s a good book) I was suddenly overwhelmed with the things we weren’t doing in our CI process. I immediately started putting in more effort to make things better and implemented more jobs and parallelization. Only I didn’t quite gauge of what future costs of hosting and maintenance would be, or why I was doing so.

Result? A lesson learned the hard-way, servers can easily cost more than your apartment rent. And only implement parallel jobs where it makes sense, where it actually saves time.

Not to mention, you have to maintain all these jobs in years to come.

# **6. CI/CD doesn’t mean no manual testing**

Do not think that having world-class CI/CD pipeline would replace manual testing ever. Because it won’t. It may come close but it won’t.

Instead of thinking about elimination, integrate manual testing with your pipeline. Here’s one method we use:

We use [Test Collab](http://testcollab.com/) (yes, our own product) for our manual testing. We’ve tagged test cases which needs to run during different stage of the pipelines.

Since most of our projects have fairly simple pipeline as of now, process is simple:

As soon as staging setup job finishes up, we call Test Collab API to delegate test cases tagged with ‘pre-launch’ to our tester. This informs our testers that they’re expected to run ‘pre-launch’ test cases on a particular URL. After they wrap up testing, an email is sent to manager (me, in this case) with full report. I manually presses “OK” button to trigger “launch” job – which deploys everything and then once again Test Collab is called to create manual test execution request with ‘post-launch’ tag.

# **7. Unit tests are big investment when your project is new**

I suggest to avoid it entirely until your code stabilizes. Unit tests are brittle and many times they can be replaced by automated acceptance tests.

When your project is in beginning phase, it might see a lot of rewrites, new libraries etc. In such cases, unit tests can be a huge overhead and there’s little-to-no ROI there.

Acceptance tests on the other hand are better candidates.

While building [Load Xen](http://testcollab.com/loadxen/) we followed Test-driven-development and it didn’t really work for us, well for some parts at least. The primary reason was that major parts of code were being rewritten every week to counter new problem.

It’s still a debatable thought, but you’ve been warned! 

# **8. Build all branches, not just trunk/master**

Another hugely important pointer here. I urge you, read this title once more, please. Thanks!

So what’s the problem if I just build master? All other branch’s failures are hidden from you before you merge.

Ideally we would like to keep all branches close to release-ready state. Without the pipeline, branches are an element of surprise for you. On the other hand, if you’re building all branches, you are confident about their quality.

This also saves time as you’ll see builds go green smoothly, one after another, even after merge!

Changes in branches, also affects manual testing, see [how branches and manual testing can be handled in Test Collab](http://testcollab.com/blog/testing-feature-branches/).

# **9. Pipelines should be matured slowly instead of in one-go**

Remember my mistake from #5? Chasing perfection can be good, and harmful too. Do not over-invest in your CI/CD pipeline. Solve current problems, leave the theoretical problems for future.

Here’s an excellent representation from [Shippable](http://shippable.com/) team:

I suggest you to slowly mature into CI – CD model instead in one go. Assess where your project stands now, and what problems you’re facing today- and then move up the model.

Because to move up from one stage to another, will cost you and things will go wrong. You have to have good rationale behind the move.

One more thing: not all your projects will need to move up after certain stage.

**Top 8 Continuous Integration Tools**

Posted by [Vladimir Pecanac](https://code-maze.com/author/codemaze_blog/) | Feb 20, 2016 | [17](https://code-maze.com/top-8-continuous-integration-tools/#comments)



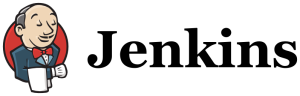
If you are familiar with the concept of [Continuous Integration](https://code-maze.com/what-is-continuous-integration/) we can agree that using it has become mandatory. There are **many Continuous Integration tools** out there, and in this article, we will go through some of the greatest tools available on the market and see how they fare.

After reading this list you will have a better understanding what tools are available to you and choose the perfect tool for yourself.

On the other hand, if you are interested in mobile apps, there is a specific subset of tools that might suit you better, so check out our [Top Mobile Continuous Integration Tools List](https://code-maze.com/top-mobile-continuous-integration-tools/).

So without further ado and in no particular order of importance, I present you the list of the top 8 Continuous Integration tools:

**Jenkins**

[](https://jenkins-ci.org/)

Jenkins is an open-source CI tool written in Java. It originated as the [fork of Hudson](https://jenkins-ci.org/blog/2011/01/11/hudsons-future/) when the Oracle bought the Sun Microsystems. Jenkins is a cross-platform CI tool and it offers configuration both through GUI interface and console commands.

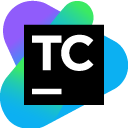
What makes Jenkins **very flexible** is the feature extension through plugins. [Jenkins plugin list](https://wiki.jenkins-ci.org/display/JENKINS/Plugins) is very comprehensive and you can easily add your own. Besides extensibility, Jenkins prides itself on distributing builds and test loads on multiple machines. It is published under MIT license so it is free to use and distribute.

Cloudbees also offers hosted solution in the form of the [Jenkins in the Cloud](https://www.cloudbees.com/products/jenkins-cloud).

**Verdict:**One of the best solutions out there, both powerful and flexible at the same time. The learning curve could be a bit steep, but if you need flexibility it very well pays off to learn how to use it.

**Official website:** [Jenkins](https://jenkins-ci.org/)  
**Availability:** Free  
**Platform:** Cross-platform

**TeamCity**

[](https://www.jetbrains.com/teamcity/?utm_source=code-maze&utm_medium=cpc&utm_campaign=teamcity)

TeamCity is the mature CI server, coming from the labs of the JetBrains company. JetBrains has established authority in the software development world, and developers all over the world use their tools like WebStorm and ReSharper.

TeamCity offers all the features in its free version, but it is **limited to the**[**100 build configurations and 3 build agent**](https://www.jetbrains.com/teamcity/?utm_source=code-maze&utm_medium=cpc&utm_campaign=teamcity)[**s**](https://code-maze.com/continuous-integration-with-teamcity/#basicconcepts). Additional build agents and build configurations need to be purchased. Recently JetBrains started offering a **cloud trial** of TeamCity where you can try it out for one project without a hassle of having to install it on-premises. It lasts 60 days and you can export the project afterward.

Out of the box, TeamCity works on many different platforms and has the support for wide variety of tools and frameworks. There are many publicly available plugins, developed both by JetBrains and third parties.

Despite being the Java-based solution, TeamCity offers **the best .NET support** among the tools on this list. There are also different enterprise packages, that scale by the number of agents your company needs.

You can find a great [in-depth case study on TeamCity](https://code-maze.com/continuous-integration-with-teamcity/) here on our blog.

**Verdict:**Great solution overall, but due to its complexity and price, better suited for enterprise needs.

**Official website:** [TeamCity](https://www.jetbrains.com/teamcity/?utm_source=code-maze&utm_medium=cpc&utm_campaign=teamcity" \t "_blank)  
**Availability:** Free for 3 agents and 100 build configurations and paid enterprise edition  
**Platform:** Servlet container (On-premises), cloud trial

**Travis CI**

[https://code-maze.com/wp-content/uploads/2016/02/TravisCI-logo-gray.png](https://travis-ci.org/)

Travis CI is one of the oldest hosted solutions out there and it has won the trust of many people. Although it’s mostly known for the hosted solution, it offers the on-premise version too in a form of [enterprise package](https://enterprise.travis-ci.com/).

Travis CI is **free for all open source projects** hosted on the GitHub and for the first 100 builds otherwise. There are a few [pricing plans](https://travis-ci.com/plans) you can choose from, the main difference being the number of concurrent builds you can run.

Builds are**configured using .travis.yml** file which contains the build tasks that will be executed on running the build. It supports a variety of different languages and a **good documentation** to back them up.

**Verdict:** A Mature solution that offers both hosted and On-premises variants, loved and used by many teams, very well documented.

**Official website:** [Travis CI](https://travis-ci.org/)  
**Availability:** Free for open source plans and first 100 builds, paid plans for everything else  
**Platform:** Hosted and On-premises

**Go CD**

[](https://www.go.cd/)

Go is the [newest Cruise Control incarnation](http://build-doctor.com/2010/06/25/cruise-go/) from the ThoughtWorks company. Excluding the commercial support that ThoughtWorks offers, Go is free of charge. It is available for Windows, Mac, and various Linux distributions.

What makes Go stand out from the crowd is **the concept of pipelines** which makes the modeling of the complex build workflows easy. On the pipeline concept, how it can help with Continuous Delivery and how it compares to Jenkins pipelines you can read [here](https://highops.com/insights/continuous-delivery-pipelines-gocd-vs-jenkins). Go CD supports pipelines from scratch and eliminates build process bottlenecks with the parallel execution of the tasks.

**Verdict:** On-premises solution, great for complex scenarios, free of charge with paid support.

**Official website:** [Go CD](https://www.go.cd/)  
**Availability:** Free with a paid support  
**Platform:** On-premises for Windows, Mac and some Linux distributions

**Bamboo**

[bamboo logo](https://www.atlassian.com/software/bamboo)

Atlassian is the company focused on providing tools for software development teams and you might know them by their tools like [JIRA](https://www.atlassian.com/software/jira) and [Bitbucket](https://www.atlassian.com/software/bitbucket). Bamboo originally offered both cloud and On-premises solutions, but in the May 2016 the cloud version was discontinued in the favor of the Bitbucket pipelines (accessible through the left panel of your Bitbucket account).

By utilizing the power of Docker, Bitbucket Pipelines is offering very efficient and fast builds that and is rapidly growing and becoming a worthy successor to the Bamboo Cloud.

Bamboo is free to try for 30 days, and after that, there are two plans for small and growing teams. Being the Atlassian tool, it has the native support for JIRA and BitBucket and you can even import your Jenkins configurations into the Bamboo easily.

**Verdict:** Great On-premises CI tool that originally offered Cloud solution too. Bitbucket Pipelines replaced the cloud solution. Pipelines is a modern and fast cloud CI tool integrated into Bitbucket. Has a free trial for 30 days, and paid plans after that.

**Official website:** [Bamboo](https://www.atlassian.com/software/bamboo)  
**Availability:** Paid with a free trial  
**Platform:** On-premises

**GitLab CI**

[](https://about.gitlab.com/)

GitLab CI is an integral part of the open-source Rails project GitLab, which was brought to light by the company GitLab inc. It is hosted on GitLab.com, a free hosted service and it provides detailed git repository management with features like access control, issue tracking, code reviews and much more.

GitLab CI integrates seamlessly with GitLab and it can easily hook projects using the GitLab API. GitLab runners that process builds are written in Go language and can run on Windows, Linux, OSX, FreeBSD, and Docker.

[The official Go runner](https://gitlab.com/gitlab-org/gitlab-ci-multi-runner#features) can run multiple jobs concurrently and has inbuilt Docker support. Gitlab CI comes with both the open-source GitLab Community Edition and with the [GitLab Enterprise Edition](https://about.gitlab.com/pricing/).

**Verdict:**APhenomenal hosted tool with impressive list of features, offers both free and enterprise solutions.

**Official website:** [GitLab CI](https://about.gitlab.com/)  
**Availability:** Free and paid with trial  
**Platform:** Hosted (can be hosted for you on Gitlab.com)

**CircleCI**

[](https://circleci.com/)

Another cloud alternative that comes from the company with the same name. CircleCI currently only supports GitHub and the list of supported languages includes Java, Ruby/Rails, Python, Node.js, PHP, Haskell, and Scala.

What separates CircleCI from the other tools is the way they offer services. The main [pricing](https://circleci.com/pricing/) block for the CircleCI is the “container”. One container is free and you can build as many projects on it as you need. Once you start adding more containers (at a fixed price each) you can choose the level of parallelization that suits your needs.

There are 5 levels of parallelization (1x, 4x, 8x, 12x and 16x). So, starting with the 16 containers, you can achieve maximum parallelization of 16x on one build. Or you can run 4 builds on 16 containers with 4x parallelization. It is up to you.

And did I mention CircleCI supports Docker?

**Verdict:** Flexible cloud CI tool that offers parallelization up to 16x. Excellent if you need something built fast and money is not the biggest issue (can reach up to $3150/mo).

**Official website:** [CircleCI](https://circleci.com/)  
**Availability:** Free and paid with trial  
**Platform:** Hosted

**Codeship**

[](https://codeship.com/)

If you haven’t had enough hosted solutions up until now, here is another one.

Codeship comes in two different versions: Basic and Pro. Basic version offers out-of-the-box Continuous Integration service but doesn’t have docker support and its main purpose is to build applications with common workflows through the UI. Pro version offers **more flexibility and docker support**.

The basic version comes in several paid packages, where the more expensive ones have more parallelization power. In the pro version, you get to choose your instance type and the amount of parallelization up to 20x). It can get a bit pricey, but some teams may need that kind of power.

**Verdict:** Powerful hosted solution with docker support, flexible plans suited both for small teams and enterprises alike.

**Official website:** [Codeship](https://codeship.com/)  
**Availability:** Free for 100 builds per month and paid for more than that  
**Platform:** Hosted

**Honorable mention: Codefresh**

[codefresh logo](https://g.codefresh.io/signup?ref=BJV2J4zib)Many tools on this list have Docker support, but Codefresh was designed and built from the ground up specifically with the containers in mind.

Docker can be a bit overwhelming to figure out at first, and the guys from the Codefresh inc. are well aware of that. In addition to working with existing docker files, you can choose from**several different templates** to ease the migration of your project to Docker containers. UI is clean and intuitive, there is almost no need to parse through the documentation to start using it.

The reason this CI tool deserves to be on the list lies in a feature that surprised me a bit. And that feature is **launching your images to a stage-like environment**. When the build finishes, you can launch the image to see if it works! That effectively means you get a staging environment without a need to provision additional virtual machines or deploy anything. And that’s great!

Codefresh is still very young and has room for improvement and new features, (eg. .NET core template and more deployment options), but it treats containers as a first class citizen and that makes it an ideal solution for any team that plans to utilize Docker.

**Verdict:** Easy to use tool with Docker containers at its core and very nice feature of launching the built Docker images to the hosted environment.

**Official website:** [Codefresh](https://g.codefresh.io/signup?ref=BJV2J4zib" \t "_blank)  
**Availability:** Free for 200 builds per month, 5 concurrent builds, and 1 hosted environment, paid for additional stuff  
**Platform:** Hosted and On-Premises via Kubernetes and Helm Charts

**So, What Is the Perfect Continuous Integration Tool for You and Your Team?**

There are several things to keep in mind when choosing the right CI tool for your projects.

On-premises solutions offer a great deal of **build process** **flexibility** and **store the** **artifacts locally**. This may or may not be important to you, but in some cases and for some companies, it might be mandatory.

On the other hand, the hosted solutions offer **no hassle setup and greater scalability** since you don’t need hardware to host them.

Another important thing is the**Docker support**. Docker revolutionalized the way we distribute our apps and has become something you should not ignore. Although the vast majority of the tools support Docker, some take it more seriously than others.

And the last and often neglected aspect is the **user interface**. I found some of the tools from the list much easier to use than others. You cannot say with the clear conscious that the UI is not important because one of the main roles of any good CI tool is to make a build process easier. It should not be hard or complicated.