THE LINUX FOUNDATION
TRAINING

DevOps Fundamentals: Understanding the CI/CD Pipeline

Learn How to Optimize Your Software Development Processes

By John Willis and The Linux Foundation Training Staff

Contents

1 Introduction

The DevOps Fundamentals: Implementing Continuous Delivery (LFS261) course from The Linux Foundation provides basic knowledge of the process, patterns, and tools used in building and managing a Continuous Integration/Continuous Delivery (CI/CD) pipeline.

This course aims to help you develop a good working knowledge of the concepts of DevOps, covering the foundation, principles, and practices. Additionally, the course focuses on some successful patterns used by high-performing organizations over the past 10 years.

In this ebook, we'll introduce the basic concepts of CI/CD that are presented in the course. These concepts are important in order to better understand and optimize your software development processes, and they include the following topics:

- High-Performing Organizations
- The Value Stream
- Continuous Delivery and Deployment
- Patterns and Practices
- Consistency in the Pipeline
- Automated Testing

2 High-Performing Organizations

In this chapter, I will briefly talk about some of the principles of DevOps and about the habits of high-performing organizations.

Back in 2012, I wrote a book called The Convergence of DevOps, in which I look at the history of the DevOps movement. And, more recently, I co-authored The DevOps Handbook, with Gene Kim, Jez Humble, and Patrick Debois. These resources will also provide a firm grounding in the principles of DevOps.

Damon Edwards says, “DevOps is continuously looking for new ways to break down silos, eliminate efficiencies, and remove the risks that prevent the rapid and reliable delivery of software based services.” I’m not going to give a formal definition of DevOps. I prefer to say that no one can tell you exactly what it is, but you know it when you see it work.

Collaborative Environments

DevOps has a lot to do with culture. For a successful DevOps flow, you have to foster collaborative environments. And the way you do that is by creating high-trust work environments and by learning how to embrace failure and making failure part of your habits and your culture.

High-performing organizations make work visible. They manage work in process (WIP). And, they manage flow, of course, which is the continuous delivery part of the CI/CD pipeline.

“For a successful DevOps flow, you have to foster collaborative environments.”

The DevOps Survey, which is run by Puppet Labs and the IT Revolution that I work with, has worked out the real science of this approach. The results of the survey found that high-performing organizations were both faster and more resilient, and this was seen in four variables.

SURVEY Results

High-performing organizations tend to **deploy 30 times more** frequently than low-performing organizations.

They have **200 times shorter lead times.**

They have **60 times less failures —** like change failures.

Their mean time to recover (MTTR) is **166 times faster.**

So, we see this kind of continuous delivery where you are fast and reliable, and you have deployment automation, and you version control everything. And, all of this leads to low levels of deployment pain, higher levels of IT performance, higher throughput and stability, lower change failure rates, and higher levels of performance and productivity.

In fact, there is also some data showing that this approach even reduces burnout, so it is really good stuff. In the next chapter, we'll talk about the value stream and lay the groundwork for continuous integration.

3 The Value Stream

In the first chapter, we talked about high-performing organizations and the type of continuous delivery that involves deployment automation and high throughput and stability. But, we can't really talk about continuous delivery without understanding the value stream. So, I will spin through that to make sure we are on the same page.

The value stream is "the sequence of activities an organization undertakes to deliver upon a customer request." That seems pretty obvious. But, if we are going to build a continuous delivery pipeline, or flow, we really need to understand some data points — particularly the difference between lead time and cycle time.

"The value stream is 'the sequence of activities an organization undertakes to deliver upon a customer request.'"

Different authors vary on what lead time means, but here, we'll define it as "what it takes to get a piece of work all the way through the system." Basically, cycle time is "how often a part or product is completed by a process, as timed by observation." The clock starts when the work begins, and it stops when the item is ready for delivery. In other words, cycle time is the more mechanical measure of the process capability.

Focus on What You Can Improve

Deployment lead time is where we really want to focus on the tool chain — the things that we know that we can improve, such as automation, testing, the repeatable functionality, repeatable processes. Process times should be reasonably predictable. So, you really need to figure out your particular lead time or deployment lead time, and how you are going to track that.

In Effective DevOps — which is a really good book — Jennifer Davis and Katherine Daniels say "Continuous integration is the process of integrating new code written by developers with a mainline or "master" branch frequently throughout the day. This is in contrast to having developers work on independent feature branches for weeks or months at a time, only merging their code back to the master branch when it is completely finished."

And, there are tools to allow people to be much more effective, to be doing parallel work, creating branches and feature branches. The key points here are:

- Integration
- Testing
- Automation
- Fast feedback
- Multiple developers

You cannot really talk about continuous anything — but certainly not about continuous integration — without quoting Martin Fowler, who is one of the original Agile Manifesto signers.

Fowler says:

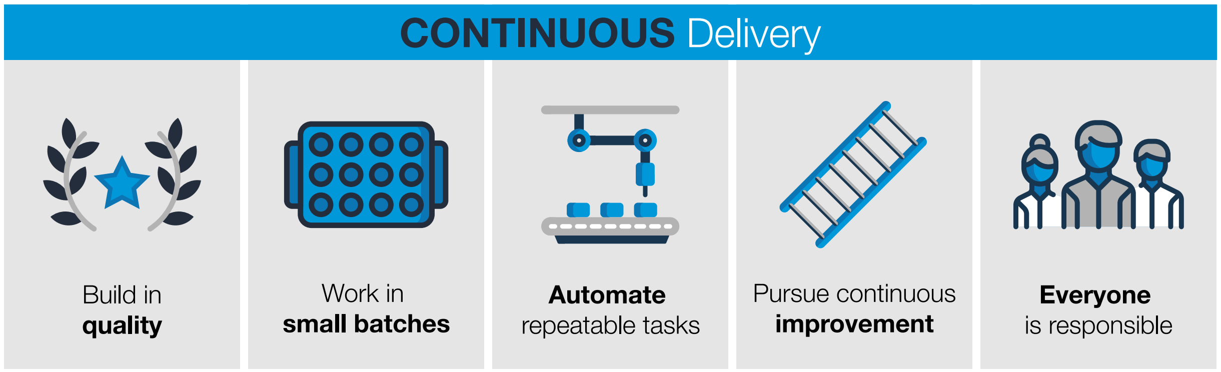
> *"Continuous integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily -- leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."*

In the next chapter, we'll take these ideas a step further and look the difference between continuous delivery and continuous deployment.

4 Continuous Delivery and Deployment

In the previous chapters, we've looked at high-performing organizations and then discussed the value stream. Now, we'll move along to continuous delivery and deployment.

Continuous delivery means the following:



Continuous delivery basically includes continuous integration. It is mandatory to have continuous Integration to get continuous delivery. Consider the following definition from Effective DevOps by Jennifer Davis and Katherine Daniels.

> *"Continuous delivery is the process of releasing new software frequently through the use of automated testing and continuous integration…"*

In other words, continuous integration is required.

Additionally, continuous delivery *"is closely related to CI, and is often thought of as taking CI one step further, so that beyond simply making sure that new changes are able to be integrated without causing regressions to automated tests, continuous delivery means that these changes are able to be deployed."*

Basically, this definition shows what we want to accomplish with the process of continuous delivery, which is that someone checks in code, version control is in place, it runs the build and tests, it fails, it kicks it back, and so on.

Benefits

The various benefits of continuous delivery include:

- **Low-risk releases:** The primary goal of continuous delivery is to make software deployments painless, low-risk events that can be performed at any time, on demand.
- **Faster time to market:** It's not uncommon for integration and test/fix phase of traditional software delivery lifecycle to consume weeks or even months. When teams work together to automate the build and deployment, environment provisioning, and regression testing processes, developers can incorporate integration and regression testing into their daily work and completely remove these phases.
- **Higher quality:** When developers have automated tools that discover regressions within minutes, teams are freed to focus their efforts on user research and higher level testing activities. By building a deployment pipeline, these activities can be performed continuously throughout the delivery process, ensuring that quality it built into products and services from the beginning.
- **Lower costs:** Any successful software product or service will evolve significantly over the course of its lifetime. By inverting in build, test, deployment, and automation, we substantially reduce the cost of making and delivering incremental changes to software by eliminating many of the fixed costs associated with the release process.

- **Better products:** Continuous delivery makes it economic to work in small batches. This means we can get feedback from users throughout the delivery lifecycle based on working software.
- **Happier teams:** Peer-reviewed research has shown continuous delivery makes releases less painful and reduces team burnout. Furthermore, when we release more frequently, software delivery teams engage more actively with users, learn which ideas work and which don't, and see immediate outcomes of their work. By removing the low-value painful activities associated with software delivery, we can focus on what we care about most — continuously delighting our users.

This then, brings us to continuous deployment. The difference between delivery and deployment is that the deployment is actually automated.

"The difference between delivery and deployment is that the deployment is actually automated."

Again, let's check the definition from Effective DevOps because the authors are DevOps leaders in every sense of the word.

> *"Continuous deployment is the process of deploying changes to production through the engineering of application deployment that has defined tests and validations to minimize risk. While continuous delivery makes sure that the changes are able to be deployed, continuous deployment means that they get deployed into production."*

The key points here are that code is deployed, and continuous deployment includes both continuous integration and continuous delivery. There is mix and match; some things we can automatically deploy, and some things can be deployed while others can be delivered.

The main point is that continuous deployment is all automated. You hit the button, you commit, and it is gone.

In the next chapter, we'll look at patterns and practices in the CI/CD pipeline.

5 Patterns and Practices

In this ebook, we're providing a preview of the DevOps Fundamentals: Implementing Continuous Delivery (LFS261) course from The Linux Foundation and discussing the elements of a successful CI/CD pipeline. In this chapter, we will cover patterns and practices. We will also look at the deployment pipeline, consistency in the pipeline, and automated testing at a high level.

To start, let's look at the concept of The Three Ways of DevOps, which is also covered extensively in *The DevOps Handbook.*

The First Way

The First Way is really continuous delivery, but it is about the flow. It involves thinking about a left-to-right flow, automated supply, a software delivery supply chain, the commit, and the whole continuous delivery process.

"All stages of the pipeline are visible to everyone responsible for the delivery."

The Second Way is about monitoring and feedback. And, the Third Way is continuous learning.

For the discussion here, everything is about the First Way. We are talking about the pipeline. And, the pipeline requires you to think about visibility. All stages of the pipeline are visible to everyone responsible for the delivery. That is a key point.

CD Patterns & Practices

Continuous Integration
The process of integrating components of a feature, application or service

Continuous Delivery
Uses continuous integration to create installable artifacts (packages) that can be deployed

Continuous Deployment
The process of deploying a feature, application or service to production

That is where we need to be. The point is that everybody is responsible; everybody should see the pipeline.

There are five steps to increase value and flow:

1. Define value precisely from the perspective of the end customer in terms of a specific product with specific capabilities offered at a specific price and time.
2. Identify the entire value stream for each product and eliminate waste.
3. Make the remaining value-creating steps flow.
4. Design and provide what the customer wants only when the customer wants it.
5. Pursue perfection.

Additionally, feedback loops should be designed to create and eliminate downstream defects. You find things downstream. You move those checks early. You have test-driven development, and behavior-driven development, and smoke tests. Again, at the end of the day, this allows you to start building incredibly robust chains of delivery.

And, you are continually deploying. Now you've got it. The pipeline should be such that any patch, any update, any new feature can be automated and deployed for release. And that means whether you are Dev, or Ops, or Sec, you are basically putting everything into source control.

The next chapter will build on these ideas and discuss how to create consistency in the pipeline.

6 Consistency in the Pipeline

In this chapter, I will quickly review some of the various tools to consider and then discuss how to achieve consistency in the pipeline.

To start, we need to have source control, and for that Git is one of the more popular tools to use. But in the Microsoft world, you have Team Foundation Server, Perforce, and SVN, which is a bit older. Then, there is also the really old CVS. There are also some SaaS-based source control systems like GitHub, Bitbucket, and GitLab.

In terms of what we call the build console or the Continuous Integration server, we have Jenkins, Bamboo, Team City, as well as Travis CI, Circle CI, and Shippable.

For repository managers, we use Nexus and Artifactory in this course, but you can also consider Docker Trusted Registry, Docker Hub, and Google Container Registry.

In terms of operations consoles, there's Rundeck and Marathon. Also, Asgard is interesting; it is part of Netflix's open source tooling and particularly works with Amazon. There's also Spinnaker and WeaveScope.

For automation, we have Cfengine, Chef, Puppet, Ansible, Docker Compose, Cloud Formation, and Terraform. This is in no way the entire list. There are new products every day. These are just some of the most commonly used tools that you should consider to help you achieve your goals.

Version Control Is Key

Let's get back to consistency in the pipeline. Here is the thing: when you get into containers, consistency is even more important, because you are running hundreds, maybe thousands of containers; you're running containers in a cluster.

So, the goal is to create consistency. In the early days, this was all checklist-based or relied on somebody's shell script, and that was somewhat inconsistent. Then, Chef and Puppet came in and created a high level of consistency, where all the environments would get built at every level through some domain specific language (DSL).

At the end of the day, all elements of the pipeline should be disposable and reproducible. All environments should look like production, from laptop, to integration, to any type of testing. You want to decrease variability between elements in the pipeline. Repeatability increases speed in rebuilding environments. Improved consistency also results in reduced errors and increased security.

"All elements of the pipeline should be disposable and reproducible."

Creating consistency in the pipeline means:

- Version control everything.
- Version control keeps a history of all changes.
- You can easily check differences between versions.
- You can restore and rebuild all elements.
- Everything can be versioned and tagged.
- All changes are visible and audited for everyone.
- Changes can be automated.

The magic really starts when everything is in version control and you can track where everything came from. Basically everything should be in version control.

7 Automated Testing

This is the last chapter in our ebook covering the basic principles of DevOps and CI/CD. This time, we'll cover automated testing. There are nine principles; these are based on work by Elisabeth Hendrickson, Pivotal's Vice President of Data R&D, who presented a brilliant overview on Agile testing.

These principles are an important part of the Continuous Delivery pipeline and an important way of thinking about how things need to be done.

Nine principles:

- Change the adversarial mindset to a collaboration mindset.
- Change the dev-then-test mindset to test-and-dev mindset.
- Instead of having a test team, you have an "everyone tests" team.
- Test early, test often, and shorten the feedback loop.
- Tests should have reasonable expectations.
- Fix bugs when you find them.
- Reduce test documentation, automate the expectation.
- Done means released and tested.
- Test implement vs. implement test.

Always Be Testing

Tests should always be running. Every commit, nightly functional testing, smoke tests, stability testing, performance testing, zero configuration, zero downtime.

Now let's see how automated testing plays out. In continuous delivery, we have:

"Tests should always be running."

- Test-driven development (TDD)
- Acceptance test-driven development (ATDD)
- Behavior-driven development (BDD)
- Automated tests (build and deploy)
- Tools

In the full training course, we talk about system testing, performance testing, load testing, security testing, even configuration management, like behavior-driven testing. Test-driven development means you write tests before you write the code. The point is that there are lots of opportunities for testing, and you should be testing all the time.

DevSecOps

Another area that is really getting hot is DevSecOps. There is a lot going on here: security testing, SQL injection, cross-site scripting, unprotected redirects, remote code execution — all the things that need to be caught in a post- or pre-deployment. For example, you can:

DevSec**OPS**

Create **security related stories**.

Add **security unit tests**.

Create a library of **reusable test case patterns**.

Add **vulnerability scans** into the pipeline.

It used to be that software bugs were evaluated by one group, and another group evaluated security bugs, or security vulnerabilities, or security defects. Now, everything is in the same place. It includes both the delivery of software and the security in the pipeline as it is happening. DevSecOps is hot, and there's some fascinating stuff going on with this topic.

Conclusion

This wraps up our ebook previewing the DevOps Fundamentals training course and the principles of the CI/CD pipeline. Much more information is available in the sample videos for the training course, which you can access now. Or, you can sign up for the full training course here.

THE LINUX FOUNDATION

Learn More about the Principles of DevOps and CI/CD

The Linux Foundation is the go-to place for training and certification on some of the hottest and most important software technologies. As the home to 50+ leading open source projects—including Linux, Cloud Foundry, OpenDaylight, Kubernetes, and many others, The Linux Foundation offers:

- Detailed, hands-on training from true industry experts that you just can't find anywhere else.
- Certifications created by collaborating with some of the top companies and subject-matter experts in the world. And they're available online anytime, anywhere, saving you a trip to the testing center.
- Unparallelled expertise and experience in teaching people to use and profit from open source technology.

So take your expertise to the next level with training straight from the source. For more information on The Linux Foundation's training and certification programs, please visit: http://training.linuxfoundation.org.