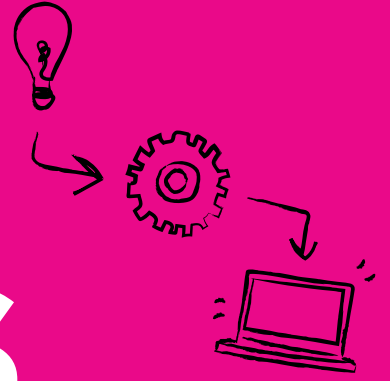


LOW-RISK RELEASES



DevOps is as much a business development strategy as it is a software development strategy. It helps companies solve the most important business challenge: how to move from building good products to creating great ones—faster.

And with the added promise to make it easier and safer for companies to quickly ship products and services, DevOps is becoming the new development standard. When done right, it creates significant competitive advantages. Executed poorly, DevOps can degrade your team's performance and introduce risk.

Set your team up for success. Here's what you need to know to build a leading DevOps practice and ways you can continually optimize it.

5 STEPS TO BUILDING A LEADING DEVOPS PRACTICE

For years, software teams have been mired in delays. And here's the thing: The coding speed of developers hasn't been the main issue. The longest hang ups have always been with getting code through approvals, testing and integration. As a result, companies have been releasing software with massive feature updates in large chunks on a monthly or yearly cycle. That's been the standard, until DevOps came along.

With the constant change in the tech landscape, DevOps methodologies have taken off—and out of necessity. Fast-moving organizations use DevOps to release features and fix bugs faster. Startups leverage these techniques to go to market quicker and catch up to the “big players” in less time. And in that way, DevOps is as much a business development strategy as it is a software development strategy.

Consumers see the direct benefits of DevOps and now expect a flow of small feature releases instead of large overhauls. Because of that, companies feel added pressure to keep release pace with their competitors. The goal of DevOps isn't speed for speed's sake. The goal is production and value above all else. To succeed in DevOps, you need to examine the entire software life cycle and think of it as a system, a factory that produces software features.

Treat your organization as a system

Imagine a factory. Instead of producing a physical product as a factory would, you're producing features. Ideas come in and features go out. Your idea moves down the conveyor belt as it's assembled, tested, packaged and delivered to your customer. Your goal is to optimize the entire production process. But how can you use DevOps (and the Lean principles it's rooted in) to improve your software factory? Here's how:

1. Reduce your batch size: A key principle of the Lean methodology is eliminating waste and reducing work-in-progress. Instead of a large set of features and changes, your releases should be a single, small feature or change. Each update should be as small as possible.

This requires a large culture change, and you may encounter a lot of pushback on it, even from your developers. For decades, a “software release” was synonymous with a large set of drastic changes; the underlying assumption was that the more improvements and changes made through each release, the better the release was. Unfortunately, there are still developers who feel it's a waste of time to go through an integration and deployment process for a single small change. Getting everyone on board with doing something a new way can be hard, so be patient.

2. Create high-fidelity environments: Lean promotes building quality into the process by automating and standardizing tedious, repeatable tasks. Developers and testers must be able to create servers configured exactly as they are in production, and they need to create it from an automated process without permission from IT.

This requires a mindset shift in your IS/IT department. Many IS/IT teams don't like allowing developers to spin up any server they want. Careless creation of machines can cause issues with resource availability and costs. To overcome these roadblocks, you need to establish trust between your departments for this to work. Create standards and procedures that mitigate the risk of developers creating their own environments. Put safeguards in place where you can and let them be autonomous.

3. Automate integration: Continuous integration (CI) gets a lot of attention in the DevOps world because it focuses on delivering fast. Developers should be able to commit their code and have it whisked away, built and pushed to an environment. But even with all the speed it can produce, CI also has enormous potential to be the biggest bottleneck. The resistance you may face in automating integration will likely not come from humans. The biggest challenges lie in technical hurdles and legacy software that doesn't want to

play nice with newer methods like DevOps. Create an integration plan, and start by integrating the smaller applications that have importance to the company but are less effort to integrate. This will help your team learn more about the integration process and perfect it before taking on the large projects.

4. Automate deployment: Another way to deliver fast is to automate deployment. Your developers need to deploy approved software changes quickly and without incident. Having smaller batch sizes will help with that, as your batch size has a direct correlation with the deployment risk. Deploying a software change should be a non-event.

The biggest hurdle to deployment automation will be within your infrastructure and security teams. Experienced IT pros struggle with the idea of pushing out changes every day because deployments used to be such a major event (sometimes even filled with cake, pizza and a big celebration). Additionally, automating effectively while maintaining a high standard of security may need a DevSecOps approach, which is also something new to consider for your security team. But by making changes small and frequent over time, it becomes far less scary.

5. Automate as much testing as possible: The goal here, also grounded in Lean philosophy, is to reduce waste. Your testers should spend their time building and refining their tests, not running them. Tests should be automated, and you should treat configuration for these tests as source code. Your tests should run as soon as new artifacts are approved and pushed to test. Certain user interfaces or projects that require unique inputs may still need that human touch, but the majority should be very hands off.

You will encounter the least amount of friction implementing this step because most testers would rather spend their time creatively building and refining tests. Give them time and autonomy to do it.

Sidestep the DevOps adoption struggles

Just because DevOps is a better way of working doesn't mean everyone who's trying it is executing at a high level.

Many companies, for example, take a "tool-first" approach to DevOps transformations. They buy fancy new automation tools or try to fit containers into their existing workflow. But throwing new technology at a poor software development process won't solve any problems. Without drastic process change, you won't get different results.

Software alone can't solve these problems.

Companies struggling with DevOps adoption today are hampered by bottlenecks in their process, usually the same ones that existed before they started DevOps. They include:

- Waiting for the creation of testing or hosting environments
- Approval time (testing, provisioning or business rules)
- Wait times during rework and regressions

Software alone can't solve these problems. Even after the expected delays from introducing, training and integrating new tech, companies that took the "tool-first" approach see similar (or even degraded) performance. The bottom line: No matter how fluid and fast your software infrastructure is, if a code commit is waiting for a week to for testing your pipeline is idle.

Optimize your value stream

To remain competitive in today's market, you need to be moving faster and with fewer errors than your competitor. By examining your value stream as a system focused on output, you can find the true development problems and address them.

DevOps is an all-in approach. Picking one or two things and automating them won't magically increase your output. You need to map your value stream and make adjustments to each system to establish a smooth flow.

The adjustments you make will not be "one and done." Give attention to learning from the process and make changes based on what you learn. Approach the adjustments with a hypothesis and test it, and share the results with the team. The overall goal of Lean and the foundation of a quality DevOps practice, which will allow you to iterate to perfection, is a culture of experimenting and sharing.

CREATING VALUE THROUGH CONTINUOUS DELIVERY

A good development process allows your developers to offload repeatable tasks and focus on more important things, like feature development and bug fixes. And to implement a successful process, you need to think beyond “automate everything!” and look at your entire software life cycle. As you do this, remember: the focus should be equal parts “continuous” and “delivery,” especially the things you do to continuously deliver value to your customers.

At the end of the day, continuous integration and continuous delivery (CI/CD) is all about delivering more value to customers—faster. But, the secondary benefits of a well-designed continuous delivery system are just as appealing, including reduced software risk, fewer broken releases and less stress. (The fact that it happens to make life easier for the engineers doing manual deploys is a nice bonus too.) If you’re ready to capitalize on this, here’s what you need to do.

You need to think beyond “automate everything!” and look at your entire software life cycle.

IMPLEMENT CONTINUOUS DELIVERY

Before you can implement a CI/CD process, you need to examine and establish your value stream—all the steps it takes to deliver value to your customer, starting with the idea. Once you understand your value stream, you’ll be ready to dive into the three steps that will help you implement a CI/CD process aligned to your overarching goal:

Step 1: Configuration

Servers should be created for a predetermined purpose, then destroyed. Create a configuration for each server as a template, and store that template just as you would store code. This enables you to create servers at will for a particular task, with consistency you can rely on.

For instance, if you have a web application that requires a particular configuration, you should be able to pull that configuration from source control and create a server that mirrors that same application server in testing, staging and production. Once you’re done with that server, you can discard it.

How to get started with configuration management

Document the plan for creating configurations of known servers in the production environment. Then, create a template that recreates the configuration of each server. Since this can be difficult and time consuming, here are some methodical steps:

- Create a server with a suitable hardware profile
- Create a script to “provision” and set it up
- Push the software to the server
- Ensure every change you make to get it running is reflected in your setup script
- Test the script to make sure it’s repeatable and predictable

You’ll need virtualization for this; you won’t want to provision a physical server each time. That would take a lot of effort and get expensive fast. Instead, create a “server” in a hosted VM environment.

Step 2: Optimization

Most slow build systems have similarities, including common roadblocks that are preventable with a change in process. Here are ones to watch for:

- **Long-lived branches:** When developers create a branch and stay in it for long periods of time without committing, it creates merge problems. Their code is changing while the rest of the code in the system is changing in parallel. Merging them together becomes time consuming and uncovers regressions.
- **Manual processes:** Manual processes create a bottleneck when a human is required to move code forward in a process. The flow of the system becomes dependent on that person to move the code forward, otherwise the flow stops.
- **“Works on my machine:”** When developers build software on their local machine for long periods of time without deploying/testing, problems arise. You’ll face merge conflicts and dependency issues that take time to solve and slow the overall flow.

How to start optimizing the build process

The most important thing to do is focus on removing bottlenecks and preventing work in progress from entering your development workflow. You can do that by:

- **Moving to a “trunk-based” development model:** This model, also called “mainline development,” is where developers collaborate on a single branch (usually called “master” in git) and build the software from that branch. In this model, every developer commits at least daily.
- **Automating tests:** This effort is a big lift, as it requires significant time and problem solving, but getting everything automated ensures that software can move freely throughout your value stream.
- **Automating builds:** Every commit from a developer should trigger an automatic build. This completely removes that step from the developer’s workload, freeing them to focus on the software itself rather than its implementation.

Step 3: Automation

Automation of manual processes reduces friction and improves your overall flow. It also improves quality as you reduce the possibility of human error.

How to get started with automated deploys

Examine the flow outside of the deployment process to make sure artifacts are moving in and out of the deployment process uninhibited. Any improvements to the deploy process itself will be ineffective if your inputs aren't coming in quickly or if your outputs are stalled upon deployment.

Deployment is the “bridge” between developers and operations, so it relies on several moving parts to be successful:

- **Consistent configurations:** See step one. If your configurations are different, your deployment will fail immediately.
- **Good automated testing:** This will ensure that changes aren't held up in testing before they're deployed.
- **Automated steps:** This goes for each step within the deployment process. A single push of a button should move software from one stage to the next.

Once you're happy with the input and output flow, focus on the parts of the process that require manual work, and prioritize them according to which ones can be automated with the lowest effort. It may be tempting to prioritize the biggest process but by improving on the smaller ones first, you get faster flow immediately. What you learn from solving the smaller problems can then be applied when you tackle the larger automation efforts.

Implementing a CI/CD practice in your organization is a major undertaking. It can be difficult to start. You'll have to make lots of changes and you'll likely be met with a lot of resistance. But it's a change you need to make to continually deliver value to your customers. You can optimize your systems, configurations and frameworks to death and get nowhere, so get clear on the real goal. Optimize to make the human experience of your product better. When you step back and view your entire system from that lens and locate and resolve bottlenecks, you'll see faster flow.

As you improve your systems through configuration, optimization and automation, you'll also see its effect on your team. Developers will be able to focus on feature enhancement and bug fixes. Operations will be less concerned about deploying new software and less reserved about changing their infrastructure. When your process provides predictability and faster flow, you have happier engineers—ones delivering more customer value than ever before.

REMOVING 7 COMMON WASTES FROM YOUR SOFTWARE DEVELOPMENT LIFE CYCLE

Even a well-functioning software development process is subject to waste. The same bottlenecks, inefficiencies and issues that present themselves on a manufacturing production line will exist in your software development “factory.” For your organization to become a highly-efficient software factory, you need to address waste in a systematic way.

Here are the seven most common areas you’ll find waste in your software development life cycle (SDLC) and how you can solve them with a DevOps approach.

The same bottlenecks, inefficiencies and issues that present themselves on a manufacturing production line will exist in your software development “factory.”

1. *Overproduction*

In your SDLC, overproduction might come from your team producing a large set of bug fixes or features in staging that are not ready to deploy. Your process keeps humming along and the team keeps building, growing the pool of updates sitting idle. If the team were to deploy everything in staging all at once, you’d risk overwhelming your users with excessive feature additions and changes.

How to solve overproduction with DevOps

If you get into a position where your team is producing features at high velocity, look into your overall flow and reconsider shifting focus from quantity to quality. Shifting some time and effort to addressing bug fixes faster, for example, could create better quality software with less defects and a better pace for delivering features to the customer.

2. *Stagnation*

While it may not technically cost much to store stagnant software “inventory,” it’s still unfinished work the organization paid for but is not seeing returns on. And the more partially completed work tickets that start piling up, the greater your potential cost when you try to address them later.

How to solve inventory issues with DevOps

Examine how you’re prioritizing work. If you find, for example, that developers have multiple unfinished projects, and none of them are getting done, their scope of work is too broad. Devote resources and time to proper backlog refinement and story adjustment.

3. Motion

Motion is the movement between phases, transferring software from one stage of the software development life cycle to another. Each transition is an opportunity for a roadblock to introduce waste into your process.

How to solve motion roadblocks with DevOps

Identify common (and anticipated) bottlenecks and be mindful of how they would affect the flow of software. Prioritize removing those bottlenecks so you don't have developers sitting around waiting for something else to happen before they can begin working.

4. Preventable defects

You can never truly be “defect-free” in product development cycles. There is no “end state” of your software; it's constantly evolving to respond to technology innovations, changing threat landscapes and more. In a year, your software won't look the same as it does right now, and that's a good thing. But that doesn't mean you can't avoid preventable defects that create rework and increase customer frustration.

How to address defects in DevOps

By rigorously applying a process to improve software flow, you'll build in quality and see fewer defects. A primary driver of quality is small units of work and automated testing processes. A well-built testing process will catch defects quickly, and if the unit of work is small, the changes needed to fix it will require minimal effort.

5. Feature bloat

Too many features will make your software overwhelming to use in any practical way. The more feature bloat you have, the greater the chances of something going wrong.

How to solve over-processing with DevOps

Proper backlog and task grooming can prevent runaway feature sets from being deployed. If your team has time to produce new features the customer doesn't need, they have time to work on identifying and preventing defects. While this work may not be as fun for a developer, it's a better use of time.

6. *Wait time*

Waiting is waste in its purest form. Wait time is any bottleneck in the process (such as integration, testing or deployment) that will cause others to delay or slow their work process.

How to decrease wait time in DevOps

Focus on creating and maintaining your value stream map by looking at the time between each step in the process. (For example, the gap between deploying from stage to production.) Get a plan together to use automation to reduce wait times between those steps, prioritizing the gaps that require the least amount of effort to solve. Most issues with wait time will be removed with automation or a policy change in a process. For example, removing a human approval process for something that can be checked by other means.

7. *Transport*

No value is added in the transport phase. It's simply moving the created product to its destination. But, the opportunity for waste is high. In software, "transport" encompasses the time between software being built, tested and approved for release and the moment when the customer uses it.

How to solve transportation issues with DevOps

Transport issues with software usually point to issues with integration, where artifacts are in motion. Implementing automated and painless integration processes will allow you to push software into an environment for test and deliver it as soon as its deemed customer-ready.

Now that you know the common areas you may find waste, keep a lookout and take action when you see the warning signs. Map your waste against your SDLC to help you take a structured approach to analyzing your flow and address inefficiencies with purpose as they arise. Reducing waste within your process will help you increase velocity, the value you deliver to customers and your company's bottom line.

INCREASING VELOCITY THROUGH SKILL DEVELOPMENT

Deciding whether or not to implement DevOps might seem like a no brainer. What company doesn't want to get their innovative ideas into the hands of their customers faster, and with less risk? That's why many companies are jumping on the DevOps bandwagon. Unfortunately, not all companies—regardless of how well they problem solve for process inefficiencies—will successfully implement DevOps or increase velocity. This is because many of them fail to invest in technology skill development.

DevOps is highly focused on continuous improvement, and if you want to increase velocity and create a competitive advantage within your practice, you need to have this same mindset with your team's tech skills. In the DevOps world, your developers need a variety of skills, far more than ever before. Gone are the days of needing just “an SQL engineer” or a “front-end engineer.” You need people who can reasonably do both, and who have the skills to introduce the latest technologies into your org. You can't hire new engineers to solve for this; you need to invest in your current talent.

For DevOps to enable the workflow velocity your company and customers demand, you need to have a strong understanding of the existing skills on your team and a plan to level them up. Your plan should build up the skills of each individual on the team as well as the team as a whole. Here's how to get started.

Step 1: Understand your skill profile

Before you start building a skill strategy, you must first lay the foundation. Take time to understand the project at hand and what successful velocity requires. Examine which skills or technologies the work falls into. Some projects will be all backend/database driven, while others may be focused on the UI. Most projects, however, will be a mix of both.

From there, you'll divide the project into the skills and technologies needed to complete it. Assess your team's skills and then match them to the work that needs to be done. This is your skill profile. In an ideal scenario, your team members' skills match the requirements perfectly.

However, this may not be the case. You may see gaps or underserved skill areas in your skills profile. Take note of them. These are the bottlenecks in your process caused by skill deficiencies, and you'll want to address them before the project begins. For example, if your project consists mostly of UI development and you only have one person on the team skilled at UI, you've got a workflow efficiency (and possibly a burned out employee) problem on the horizon.

Step 2: Level up the team

Once you've established a baseline of the skills you need and the skills you have, create a specific plan for each team member to level up. Empowering your team to gain new skills helps spread the workload and improves velocity. Start by:

- Discussing how much time engineers can dedicate to learning
- Setting a short-term goal for improvement
- Checking back to measure progress
- Create a long-term goal for skill adoption
- Reviewing the learning plan frequently and matching it against your workflow data to gauge improvement

Your plan needs actionable items and a solid metric to measure improvement. And just like DevOps, this is not a one and done rollout. It's an ongoing process that is continually refined.

Step 3: Maintain and adapt skills

Building a highly-proficient team requires a long-term plan to ensure team members stay skilled. They need to keep their current skills sharp and acquire future skills as needed. This plan should include:

- **Measurable objectives:** *I will be at an expert level of SQL Querying by Q3*
- **Study goals:** *I will read one book on advanced SQL per quarter, and share my learnings with the team*
- **Knowledge sharing targets:** *I will present on one topic per quarter related to advanced SQL*

As the work and business needs change, you should adjust your team's skill development priorities. If no new technologies are coming, have them invest their skill development time to deepen their depth of knowledge on the technologies they're currently using. As new technologies start gaining relevance, reprioritize their focus.

Software development and deployment looks nothing like it did five years ago, and is a world away from where it was 10 years ago, especially with the rise of DevOps. As you build out your DevOps practice, apply the same principles of continuous improvement to your team itself. Be intentional about keeping up with the latest technology and trends, and make sure your team is gaining the skills that will fuel innovation and give you the first mover advantage. A team that develops their technology skills and shares domain knowledge will be limitless in velocity.

Building a leading DevOps practice starts with skills.

Talk to us about starting a pilot.

sales@pluralsight.com | 1.888.368.1240 | 1.801.784.9007



Jeremy Morgan is a tech blogger, speaker, and author who loves technology. He has been a developer for nearly two decades and has worked with a variety of companies—from the Fortune 100 to shoestring startups. Jeremy has spent the last several years as a DevOps consultant, helping organizations move code faster through automated pipelines.