DevOps At Scale: Chapter 6

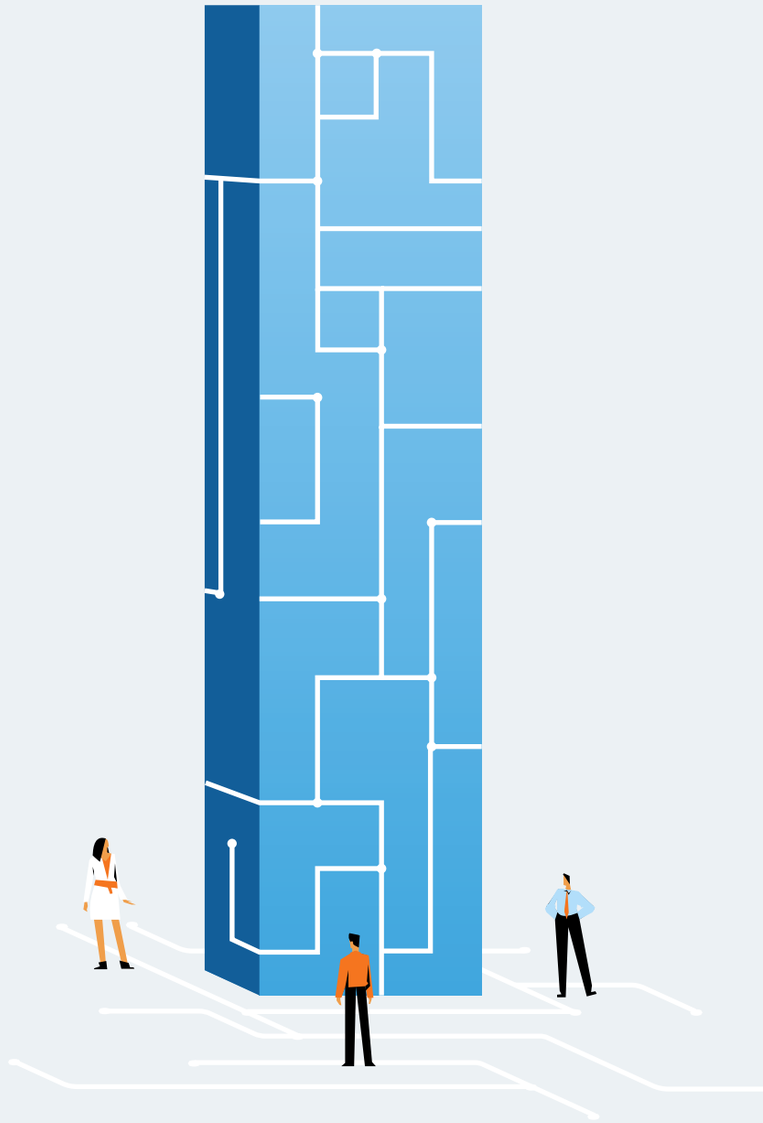
Mastering the DevOps Process

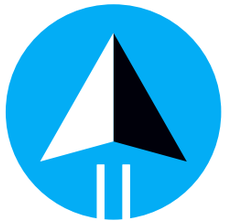
PLUTORA

As a DevOps team matures, they often find that their environment fills with a growing number of moving parts. When all of these pieces are moving in concert, new features are released regularly and smoothly. This isn't only true for technology. Mature DevOps teams have processes that work harmoniously with one another to make shipping code smoother. The same is not necessarily true for immature teams. Immature teams rely on faulty processes and poorly-configured architecture. These delay releases, suck up engineer time with tedious tasks, and cause the team to ship more bugs to customers.

Savvy teams seek ways to avoid these kinds of failures. As a team learns, it progresses through various states of DevOps maturity. Sometimes those tools to avoid failure will be technical in nature, but just as often they'll be changes to process or lines of communication. A team can have the best tech in the world, but if they have bad processes, they're still going to fail as often as they succeed.

In this article, we'll talk about three overarching processes that all mature DevOps teams need to master. Each of these skills is a place where teams must grow from immature to mature. Unless a team is brand new, it's likely that they've already developed a way to handle these processes. But a commitment to good DevOps practices means rethinking how to approach each of these problems.





Release Management

Release management is the process of how a team coordinates the actual release of code to customers. Good code tends to pass through a variety of steps so that stakeholders throughout the business can verify its validity. Beyond simply designing and writing the code, code needs to go through testing, user acceptance, and compliance phases. These steps ensure that the code does what it says it does, that customers are happy with what it does, and that it doesn't expose the company to risk.

How Do Immature Teams Do Release Management?

Anyone who's worked in a big organization before innately understands immature release management. Usually, the process for release management is (in a word) slow. A feature will experience lengthy stops at each gate of the process. Oftentimes, a feature will sit in a queue, meaning that it'll wait days or weeks before someone is able to evaluate the code's fitness. Then, as is common with any bit of code, it's likely that the gatekeeper will discover some issues. The developer responsible for those features is able to begin to correct the problems. Then, they'll send it back through the gates. Again, at each stop along the way, the code will experience lengthy delays before a human actually looks at it.

Just like developers, operations staff suffer from this degree of manual process. Immature teams manually manage their technological resources. If a new feature requires some kind of new service, such as a message queue, operations staff are on the hook to deliver. That doesn't just mean delivering a message queue to production, either. It means a message queue for testing environments, and a message queue for quality assurance. It means going through manual approval by the compliance team. Manual environment management can feel like putting out fires. Just when the team thinks everything is under control, some new environment is needed. The process starts all over again from the beginning.

Once those systems are set up, monitoring them is also a manual task. Operations staff need to manually collect metrics, and when those metrics are outside of acceptable parameters, they report that manually, too.

Immature Teams Don't Think Past the Release

Eventually, the wait between writing the code and releasing it becomes so lengthy that developers will probably need to combine it with some other group of features and bug fixes into a bigger release. This means the process starts all over. The result is significant frustration for technical and project management teams. No one can predict when a new feature will ship to customers. No one is ever sure which features or bug fixes will be part of a release.

What's more, this kind of gating ignores a crucial part of release management: post-launch support. Clever readers will notice that the entirety of the release management process described above focuses on what happens before a release. Obviously, software isn't released into a void. Customers have feedback. Their needs change. Bugs the testing team didn't catch rear their heads. Immature teams have no plans for the way that they'll deal with these issues.

How Do Mature Teams Do Release Management?

Mature teams approach release management from an entirely different perspective. For starters, their focus with release management is just as much on how to support new features after they're released as in shepherding that code before it goes to customers. They have simple, straightforward processes for gathering user feedback and listening to customers about how their needs are changing.

Instead of large, unwieldy manual gates, mature DevOps teams seek to involve stakeholders earlier in the process. Stakeholders test and evaluate code soon after developers write it. This means that developers can create automated tests which help ensure that the code does what it's designed to do as they work on it. Since that manual feedback comes earlier in the process, there is less confusion around the status of a project. Developers spend less time context switching between different features because stakeholder feedback comes quickly and clearly. Because stakeholders are able to confirm the quality of code during the development process, quality gates are now automatic.

Mature Teams Automate As Much As They Can

Operations staff are able to configure CI/CD systems to automatically move code quickly from finished to general availability. Developers don't have to suffer through round after round of feedback before their code ships. What's more, if a developer does launch code and learn only after shipping that there's a flaw customers discovered, there are mechanisms in place to deal with that. Feedback is quickly gathered, and the developer can work on the code—while it's fresh on their mind—to correct the mistake.

While CI/CD is beneficial for deployment, mature DevOps teams move past just building code and running automated tests. They automate everything, including the provisioning of their environments. This is usually done by some sort of solution in which infrastructure is designed through a coding interface. These tools ensure that infrastructure is correctly configured, no matter the environment. So if, for example, a new message queue is needed in four environments, it's only a couple of lines of code away.

These teams also configure effective metric measurement and notification systems. Everything happens automatically, meaning that operations staff are free to work on bigger problems.

Hybrid IT Management

Test environment management is the process of making sure that code is tested in an environment as close to release as possible. This means effectively building a miniature production environment while also ensuring that the security of the organization and its customers remains strong.

How Do Immature Teams Do Test Environment Management?

Test environment management is something that many immature teams don't consider at all. Test environments need to be manually configured by an engineer every time something changes. This means that code can regularly find itself in an environment that looks nothing like the production environment. Critical services might lack important updates. Databases could use a very old schema or severely outdated data.

Some organizations will choose to simply copy their production database straight to their testing environment. This can lead to issues too; these databases will contain personally-identifiable information of customers. Protecting that important information is critical, so exposing it on a testing environment risks disclosing that information to unauthorized employees.

As noted previously, fully-manual testing processes are prone to long delays. People get busy. This is especially true of project gatekeepers in large organizations. Those gatekeepers can only test one thing at a time, which increases the time between writing code and testing it. The problem compounds when some new feature introduces a new dependency to the test environment. These delays pile up on top of each other. By the time code gets back to a developer, they've forgotten most of what they wrote, and the cost for putting that code back in its proper context is high. These costs go straight to the bottom line of the company, resulting in a longer time to develop new features and respond to shifting environments.

How Do Mature Teams Manage Test Environment Management?

Mature teams, like with release management, automate as much as they can. For test environments, this means automating just about everything. Mature DevOps organizations leverage their CI/CD system to make setting up new testing environments a simple process. The same goes for tearing them down. This means that test environments are cheap. If a gatekeeper has the bandwidth to look at four different features in one day, operations can provide them four testing environments, one for each feature. The whole thing might take a few minutes.

Instead of using outdated databases, or leaving in rows which contain sensitive information, a mature DevOps team uses an automated script to create new testing databases. This script anonymizes data before it moves to the pre-production server, so

sensitive information doesn't leak. Much like test environments, these databases are trivial to stand up and tear down. Operations teams easily build new test environments, including a completely separate database, for every feature.

Deployment Planning

Deployment planning is about making sure that a release has all the necessary resources, both at launch and after release.

How Do Immature Teams Do Deployment Planning?

Once again, this is something that immature teams rarely do. The development and project management teams will regularly toss new requirements over the wall to operations staff. They won't discuss the feasibility or efficiency of architectural decisions. Decision-makers finalize decisions before consulting all the team's technical experts on a topic. This means that it's often difficult to correctly configure production environments for a release. An entirely manual configuration management strategy compounds these challenges.

The slow, painful process for getting a feature to release adds another layer of challenge to operational work. Because testing and user acceptance and compliance are manual and take unknown amounts of time, operations staff don't know which features will be in a release. This makes it difficult to prepare an environment for a feature. It's no good updating software packages or library versions if the new code to take advantage of those changes isn't in the release.

All of this adds up to stressful deployment planning. A hallmark of this kind of disjointed planning is operations staff regularly needing to play the hero. Immature DevOps teams regularly see their operations staff working long hours or needing to manually prepare releases for hours ahead of time.

How Do Mature Teams Do Deployment Planning?

A mature DevOps team turns that kind of thinking on its head. Their attitude is that operations staff should not need to be manually involved in a release. Instead, continuous delivery systems mean that as developers finish features, they're deployed automatically. Planning phases for new features involve operations experts, so they're never surprised by what architecture they need for a feature.

As features and releases become smaller, releases become more frequent. This is a boon to deployment planning, too. Because the releases are small and regular, there's no need to do the hard work of a big, last-second integration of disparate features. The quick-release cadence also means that features themselves can be smaller. Project management doesn't need to shove a dozen

features into the same release when releases happen every week instead of every quarter.

Finally, moving more testing into automated tests means that the team ships fewer bugs to customers. This decreases long-term support costs and makes the lives of both developers and operations staff less stressful as a result.

Managing DevOps Means Thinking Smaller

Much like an orchestra, managing DevOps is about balance. It means finding a way to get dozens of individuals on the same page at the same time. Just like with an orchestra, the key is to make the parts small and simple. Each player only needs to understand how to perform their part of the overall performance.

Mature DevOps teams do this by making each part of the release small and manageable. Features are smaller and released more quickly. Operations staff automate as many parts of the process as they possibly can. Project management design the minimum number of viable products, then ship them as quickly as possible. Testing staff test small features quickly and provide feedback as soon as possible.

As with every part of the DevOps culture, mature processes don't happen overnight. They're the product of years of hard work and constant, small improvements to the team. Each step of the way makes things a little better. The reward at the end of the road is a more functional business and happier staff, which is well worth the journey.

Want to learn more about DevOps?

Check our series of white papers about DevOps to learn
from the foundations, to the cultural change and maturity model.

1. What is DevOps?
2. The Benefits of DevOps Strategy
3. DevOps Methodology: Aligning your Organization
4. DevOps Tools: Why You Need Them
5. The DevOps Maturity Model
6. DevOps Pipeline: The Functional Building Blocks
7. Mastering the DevOps Process
8. Making your DevOps + Agile Transformation a Success

Visit www.plutora.com/devops to learn more.

About Plutora

Plutora, the market leader of value stream management solutions for enterprise IT, improves the speed and quality of software creation by capturing, visualizing and analyzing critical indicators of every aspect of the delivery process. Plutora orchestrates release pipelines across a diverse ecosystem of development methodologies, manages hybrid test environments, correlates data from existing toolchains, and incorporates test metrics gathered at every step. The Plutora Platform ensures organizational alignment of software development with business strategy and provides visibility, analytics and a system of insights into the entire value stream, guiding continuous improvement through the measured outcomes of each effort.

PLUTORA®

Learn more: www.plutora.com
Email: contact@plutora.com