CHAPTER

6

Deploy

Building an Agile, Resilient, and Scalable Continuous Delivery Pipeline

To keep pace with demands for new features and application updates, digital transformation must be driven by continuous delivery—the ability to rapidly and reliably release software across the pipeline at any time.

By almost every metric, companies that address this imperative create a competitive advantage over those that lag. Yet few companies have actually developed the process maturity and scalable automation needed to deliver applications at the volume, velocity, and quality levels now required to remain competitive.

Before looking the challenges and strategies needed to increase maturity, let's examine how advanced DevOps thinking backed by release automation has helped technology giant Citrix significantly cut deployment times and reduce errors during the release process.[1]

[1]Full story:http://www.ca.com/content/dam/ca/us/files/case-studies/citrix-boosts-business-agility-and-accelerates-devops-adoption-with-ca-release-automation.pdf

A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_6

Case Study: Citrix

Citrix provides a range of virtualization, networking, and cloud solutions to around 400,000 customers worldwide. To help create more productive workspaces both for its customers and own users, Citrix is continually looking at ways to improve its business operations and enable innovation. As part of this drive, Citrix created an Office of IT Delivery Optimization in December 2014, which is tasked with evaluating and improving all aspects of IT. As part of its optimization efforts, the team is embracing DevOps principles. As Eugene Lehenbauer, Worldwide IT Delivery Optimization Group Manager at Citrix, explains, "Adopting a DevOps approach to IT delivery will help us achieve better cross-team collaboration, faster delivery, and greater quality."

Although individual development, architecture, and design teams at Citrix had already embarked on their own DevOps journeys, the company wanted to take a more centralized approach to maximize results and share best practices. "Supporting innovation and free-thinking is really important at Citrix, so we didn't want to impose a specific toolset," says Lehenbauer. "We did, however, want to give teams the option of using a proven enterprise platform for automating application deployments to help free up their people from repetitive and mundane tasks."

During the proof of concept exercise, Citrix moved from manual release processes to fully automated release processes, reducing deployment time by 80 percent. But that was not enough for the Citrix team. Development was inspired to re-architect the one large "MyCitrix" application into many smaller pieces, which, along with release automation, enabled the application deployment time to be reduced further to 94 percent.

"Achieving such impressive and immediate quantifiable results has really helped accelerate the adoption of DevOps principles across the business and inspired development innovation," explains Lehenbauer.

A central dashboard permits everyone involved to view the status of all releases, giving teams the information they need to act quickly and providing an audit trail for development and operational teams alike.

The weekly updates to MyCitrix are managed via a release automation solution. "Since deploying Release Automation, we've achieved faster delivery times and fewer issues. As a result, we now have more developers focused on innovating, rather than reacting," added Lehenbauer.

Release automation has been a catalyst for Citrix's adoption of DevOps principles. It has enabled Citrix to take an enterprise-level approach to application delivery by automating application release tasks and orchestrating its continuous delivery toolchain.

Citrix has been able to significantly accelerate its application delivery and reduce the errors and time required during the release process. This has helped the company to be more responsive to customer needs, ensure compliance and auditability, and focus on innovation instead of repetitive tasks.

Obstacles to Continuous Delivery

As the Citrix story demonstrates, continuously delivering software is an extremely collaborative process that spans multiple departments, from development to test, to release management to operations. With so many stakeholders and motivations, the challenges faced by development-focused teams can be very different than those confronting operations professionals.

Development Challenges

With an emphasis on increasing throughput, major obstacles are delays and release bottlenecks. For example:

- Manual, time-consuming, error-prone environment provisioning and release processes
- Numerous errors happening throughout the application release cycle and lots of detective work to find the source of problems
- Inefficiencies caused by the uncoordinated adoption of open source tools, leading to duplication of effort, redundant solutions, and disjointed integration
- Slow response to customer feedback and market needs impacting customer retention and acquisition

Operations Challenges

With an emphasis on ensuring stability, major challenges involve guaranteeing resilience as the volume and velocity of deployments increases. For example:

- Fractured release processes; managing with spreadsheets, scripts, and tools
- Difficulty managing/tracking the volume of releases as more agile development ensues
- Long weekends, low staff-morale and stress due to problems when finally deploying to production

- Double-digit application outages or downtime happening each month and needing an "all hands on deck" approach to resolve
- Loss of customers and revenue due to downtime/outages or errors in application deployments

Finding Common Ground

Regardless of the issues facing each team, it's important that common ground and consensus is reached by tracking all issues preventing successful business outcomes. This is a shared exercise and involves all stakeholders collectively working to determine where the organization as a whole is on the path to automating software releases that drive a continuous flow of value to the business and its customers.

Tip Consider organizing a continuous delivery "current state" workshop that brings all stakeholders together. These may include application owners, developers, enterprise architects, security managers, change managers, release managers, operations, and support.

To facilitate open discussion, some good conversation-starters include:

- In terms of continuous delivery, what are our agreed business goals and metrics?
- How are we managing and executing application deployments? What elements are heavily scripted and rely on manual intervention?
- How are we configuring environments from development through to production? Are different teams using different processes?
- Across the software pipeline, what are the readily visible bottlenecks in the application release process?
- What automation tools are currently leveraged (e.g., continuous integration and configuration management)? Are teams using different tools?

By jointly answering questions like these, teams can develop a structured understanding of where there may be weaknesses across the entire release pipeline and identify opportunities to automate and improve processes.

Tip Try not to restrict analysis to release teams and processes only. Look for opportunities where automation can help drive improvements in development and testing. Careful attention should be given to how the "current state" affects the work of others. For example, if there are release delays, how does this impact development? What processes are they using to circumvent?

For example, are development teams being pulled off important refactoring work because of delays? Is testing being pushed late in the cycle or neglected because teams think they have time to do it later?

Continuous Delivery Maturity

As with all new technologies and best practices, organizations will be at different points on the journey to continuous delivery (see Figure 6-1). Some will have already begun, often by adopting facets of agile or even DevOps, while others will just be starting out. In fact, it's not uncommon for different teams across IT to be at different points in adoption.

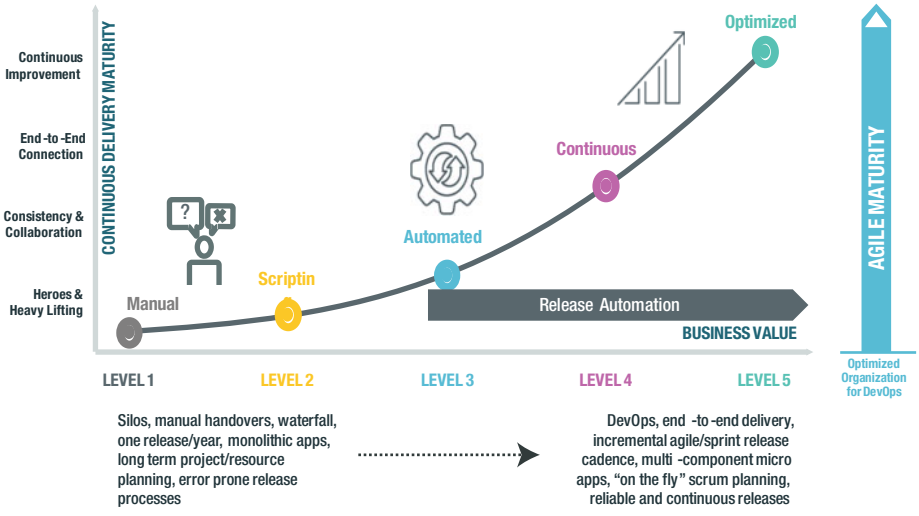

Figure 6-1. Continuous delivery maturity levels

Level 1: (Manual)

At this level, success depends on the competence and heroics of the people doing the delivery. Teams are very much operating in silos. Application releases are error-prone and infrequent and the business is badly positioned to act quickly on new opportunities, defend market position, or retain customers.

Level 2: (Scripting)

Deployment processes are planned per release, and status is managed and tracked. Automation may exist for some deployment capabilities (e.g., scripts). Teams are probably using version control/repositories (e.g., Nexus) and doing automated builds using tools like Jenkins/CloudBees. It's likely that provisioning or configuration management tools like Chef or Puppet are used to help with delivery, but no application-centric end-to-end release orchestration is employed.

Level 3: (Automated)

Here, there are common, reusable, automated application delivery processes established across environments and releases. Release processes, release metadata, and release artifacts are monitored and tracked under full lifecycle control. Delivery automation exists at the environment release level, which may include leveraging existing provisioning and deployment automation capabilities (which may not be scalable).

Level 4: (Continuous)

At this stage, release automation orchestrates application release promotion, enabling predictable, monitored, and measurable continuous delivery from development to production. Organizations can deploy applications consistently across different types of environments and releasing software is a routine and relatively low-risk event.

Level 5: (Optimized)

Now all elements are working in a fully orchestrated fashion to provide a zero-touch deployment—from planning to production. Continuous optimization of end-to-end application delivery processes through feedback loops, with deployment patterns, scenario simulation, and analysis of operational release data are used to continuously improve cost-performance of application delivery. Teams manage multiple applications (multi-services) through the continuous delivery pipeline, which becomes a single point of control. There is also a strong focus on resilience and continuous availability.

No matter where you sit on the continuous delivery maturity curve, one thing is clear—every new level provides tangible benefits. Processes become more automated and standardized. And teams become more productive, focusing on delivering differentiating features rather than managing unplanned work and maintenance tasks. They can handle the growing tempo and complexity of applications, while still ensuring quality and resilience.

Accelerating Maturity: Three Ways

As Figure 6-1 illustrates, the adoption of automated release processes and tools typically drives major inflection points in a continuous delivery journey. There are three important considerations.

The First Way: Connect End-to-End Release Management

Scripting to Automated

Taking an end-to-end release automation approach is essential in order to execute a successful continuous delivery strategy. Key to this is the ability to automate and standardize application releases all the way from development through to production, combined with capabilities to plan, manage, and optimize the release pipeline to improve quality and processes. Rather than act in isolation, release automation must easily integrate with other processes and tools (e.g., continuous integration, provisioning, and configuration management) across the continuous delivery toolchain; seamlessly scaling as the volume, velocity, and complexity of applications grow.

Taking this step to end-to-end release automation also supports DevOps adoption. It becomes easier for teams to have the release transparency, communication, and consistency needed for more purposeful collaboration.

More importantly, cross-functional teams gain control and visibility of the entire release pipeline, looking at the release process systematically versus in silos.

At this and any stage it's important to measure how improvements are helping support the business goals of continuous delivery that were identified before any toolset implementation. For a large Fortune 100 financial services company participating in a release automation ROI study, this involved increasing application release rates across the software lifecycle. As stated by the manager of DevOps enterprise release and deployment, "One of our core application deployments was done twice a week due to lack of automation, intensive manpower, and complicated deployment procedure. After automating this application deployment with release automation, the application is being deployed at least 50 times in a week, all the way from continuous integration to production."[2]

[2]The Total Economic Impact™ of CA Release Automation, December 2015: http://www.ca.com/content/dam/ca/us/files/industry-analyst-report/the-total-economic-impact-of-ca-release-automation.pdf

■ **Tip** Never underestimate the people impact when introducing new automated release methods. Rather than enforcing enterprise adoption, consider small but important projects where benefits can be quickly demonstrated. This can become the catalyst for wider support.

The Second Way: Operationalize Feedback Loops

Automated to Continuous

While automation is essential for continuous delivery, it's only the start of the journey. As automated end-to-end release processes become firmly entrenched, many new release challenges emerge. Taken individually or as a whole (as illustrated in Table 6-1), these pressure points drive a shift to better pipeline management.

Table 6-1. Pressure Points Increase the Need for Advanced Release Automation

Application Content Complexity	Infusing releases with feedback more quickly
	Prioritizing deployment of the right content
	Demonstrating implementation against business requirements
	Preventing "polluted" content from reaching production
The Pipeline Multiplier Effect	Planning, tracking, and prioritizing many complex multi-level applications and independently developed services
	Managing dependencies and avoiding conflicts
	Sharing resources between multiple teams, projects, and timelines
Pipeline Tooling Expansion	Juggling a growing breadth of open source, home-grown, and third-party commercial tools used across the enterprise by different teams

As these pressure points intensify, organizations need to consider processes for executing multi-team, cross-app, composite releases, while ensuring all dependencies are handled. The proliferation of moving parts requires a "big picture" view of the pipeline to maintain throughput, contain issues, and ensure fast feedback.

More importantly, the continuous delivery pipeline is becoming the single control point and application delivery is becoming streamlined, predictable, and risk-free. At this stage, release automation is orchestrating tools and processes beyond deployment, including application lifecycle management (ALM) and service management processes (e.g., change management). This is essential for DevOps, since it strengthens feedback loops and better informs decision-making.

Many more teams within in the enterprise should now be running apps through the single control point. If they are, they are better equipped to establish a framework of continuous delivery best practice that's valuable across the organization.

■ **Note** According to the 2016 State of DevOps report, high-performing IT organizations deploy 200 times more frequently than low performers, with 2,555 times faster lead times.[3]

The Third Way: Optimize the Continuous Delivery Pipeline

Continuous to Optimized

Although few departments are operating at this level, it is the pinnacle toward which all teams should aspire.

With the continuous delivery pipeline being too important to fail, attention should become focused toward making the pipeline (so many teams depend upon) as efficient, stable, and resilient as possible.

This involves shifting toward mastering the art of releasing multi-app, cross-app, multi-team applications and making deployments more predictable and efficient. Improving business execution through accelerated feedback loops will be another benefit, and by establishing a culture of continuous improvement, teams will embrace a "fail fast" culture and then apply the lessons learned within their release processes to prevent future problems.

This notion of continuous improvement is well illustrated by a director of DevOps tools management at a leading Fortune 100 financial services company, who stated, "Agile and continuous delivery can be nothing but a journey. You are never done; you are constantly moving the needle. There is always something you can do."[4]

Essential Toolchain Integrations

While it's important to review functional aspects of release automation solutions, what's more important is examining a solution in terms of how it helps organizations increase continuous delivery maturity.

---

[3]2016 State of DevOps Report: https://puppet.com/resources/white-paper/2016-state-of-devops-report

[4]The Total Economic Impact™ of CA Release Automation , December 2015: http://www.ca.com/content/dam/ca/us/files/industry-analyst-report/the-total-economic-impact-of-ca-release-automation.pdf

No release automation tool will work in isolation. More advanced solutions will serve as an integration hub, orchestrating many activities across the pipeline. At a simple level this could involve application-centric release automation to configure all the resources needed to support a new build (e.g., allocating server and storage capacity and ensuring an appropriate platform is in place to receive the build).

Beyond addressing immediate operational requirements, advanced solutions will work in concert with many other processes to build a continuous delivery ecosystem that helps IT achieve the most advanced levels of maturity. What distinguishes capabilities here isn't just strong integration, but the flexibility needed to support a more adaptive toolchain - one where new technologies can be quickly and easily incorporated to strengthen the continuous delivery model.

Tip To avoid vendor lock-in, ensure release automation tools provide an open and scalable platform, integrating easily with any continuous delivery toolchain for end-to-end visibility and orchestration of releases.

Figure 6-2 and the section that follows illustrate and describe essential release automation toolchain integrations needed to optimize continuous delivery.

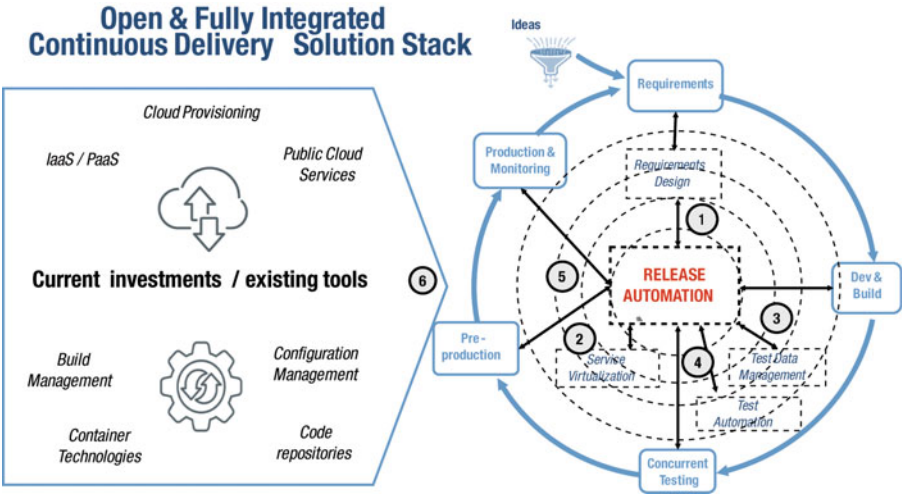

Figure 6-2. Release automation: toolchain integration

1. Requirements Design

This integration allows agile teams to track multi-application release content through the software lifecycle and establish critical feedbacks loops for faster problem resolution and application delivery.

With a real-time dashboard for managing and monitoring multi-application release content (user stories, features, and bug fixes) through the release pipeline, agile teams gain complete visibility of release progress, more easily reconcile dependencies, and can map to business requirements.

Without this integration, agile teams would have to manually track and report on business-level user stories, features, or fixes to specific application releases moving through the pipeline.

2. Service Virtualization

This integration automates the launch of virtual services as part of a deployment to optimize resources and speed testing.

Here, DevOps practitioners can provision virtual services and execute test suites across multiple virtual environments directly within a deployment workflow. By deploying into any testing environment, teams are freed from constraints (e.g., waiting for physical hardware environments to be built and made ready for testing). This improves productivity and speeds time-to-value.

Without this integration the release process could be interrupted. Manual requests would be needed to provision physical systems and virtualized services separate from the automated deployment workflow. This impedes the flow of value and ties up resources on repetitive and error-prone tasks.

3. Test Data Management

This integration automates the generation of accurate test data based on proper test cases within a release workflow.

Without this integration, manual requests are needed to generate the proper test data separate from the automated deployment workflow. Again, this results in release interrupts, delays, and slower delivery.

Note Integrating test data management with release automation should be considered a DevOps automation best practice. Not only does it ensure teams have ready access to accurate test, it also helps establish compliance (e.g., generating synthetic data to protect customer information) into the release process itself and avoid the delays associated with lengthy auditing checks at the end of each cycle.

---

4. Test Automation

This integration automatically starts the test case process and ties the results back into the release to determine and confirm readiness for promotion.

Here the test case process would be automatically initiated with the results linked back to the release. This is essential in order to determine go/no for automated promotion—enabling faster, higher quality deployments.

Without this integration it would be necessary to manually determine if the application has sufficiently passed a testing stage in order to move forward to the next stage and then manually promote the application. Again, this is time consuming.

5. Performance Monitoring

This integration establishes monitoring earlier in the software lifecycle in order to feedback critical information needed to improve quality.

Release automation can coordinate the installation and activation of monitoring in pre-production. The technique of "shift left" monitoring (discussed in Chapter 7) enables teams to see the performance impact of releases and compare it against production baselines. This provides development with earlier warning on code-related performance issues and operations earlier guidance on service-level requirements.

6. Existing Toolchain Investments

A fully integrated continuous delivery toolchain solution will be open and scalable, coordinating the application of any existing products within standard and reusable release processes. Some important integrations include:

- *Continuous integration*—Automatically kick off an application deployment upon the immediate completion of a software build in Jenkins.
- *Configuration management*—Combine release automation with solutions like Chef and Puppet to solve the problem of attempting a deployment when the target environment is not in a good known state. Integration here can be used to enforce specific environment configurations prior to deployment and manage configuration drift.
- *Cloud provisioning*—Enable users to build workflows that provision, configure, and tear down cloud environments within a deployment workflow.

Release Automation: Capability Checklist

With release automation playing such a central role in integrating tools and processes across the toolchain, solutions in this category should at a minimum deliver a deployment engine capable of supporting:

- *Artifact management*—The ability to deploy many different components and configurations of applications on physical, virtual, and public or private clouds.
- *Configurable deployment options*—A powerful, visual workflow engine to easily create standard, reusable deployment processes to promote apps from one environment to the next.
- *Reusable deployment best practices*—Shared components, allowing teams to leverage and reuse deployment logic across different projects and applications.
- *Orchestration of preferred tools*—As discussed, solutions should leverage existing tool and technology investments to automate deployments by using out-of-box action packs or through a software development kit.
- *Deployment remediation and auditing*—Using a visual dashboard teams can track and record configurations, artifacts, and release progress for improvement and auditing.

To develop, plan, manage, and optimize the continuous delivery pipeline, release automation should also scale to helping teams:

- *Design a shared pipeline*—Orchestrate manual and automated tasks within the continuous delivery pipeline.
- *Execute many complex releases*—Run through all the release phases—development to production for multi-app, multi-team releases. Iterate and improve failed content.
- *Plan and manage the timeline*—Schedule and manage apps through multiple phases using a visual calendar. Provide immediate notification of conflicts and maintenance windows.
- *Improve collaboration*—Assign owners to tasks and use an activity feed to share comments.

- *Manage and track content*—Track features as they proceed to production. Provide full insight when prioritizing and ensure the business implications of delays are clear.
- *Optimize releases*—Detect problems in real-time, recognize bottlenecks, and improve processes and team activities.

As maturity increases, release automation should cater to more advanced requirements. This may include:

Dependency Management

When building multi-component/multi-application systems, there will be complex dependencies between applications or different versions of an application. This may include a mix of release, content, application, and application version level dependencies. The knowledge of which application version depends on which is critical and often only known to a small number of experts within the department. Systems should be able to establish the definition of these dependencies with automatic alerts when dependency conditions are not met.

Pipeline Visibility with Notifications

Systems should provide a clear view of the release pipeline, including all phases and all tasks within each phase. Each phase should show list of tasks, the order of their execution, and whether it is to be run sequentially or in parallel. To support continuous delivery, each release should trigger a new build and promote this build through the pipeline, from the test phase and all the way to production. Automation should reiterate phases until all tasks pass predefined criteria. If something goes wrong with a release, delays and idle time should be reduced through automated notifications.

Flexible Approval Processes

For sensitive phases such as production deployment, full governance may be required. To support these cases, systems should prevent mistakes by allowing only the permitted users to approve the execution of these phases.

Recommendations and Action Plan

To attain the continuous delivery best practices described in this chapter, organizations need to ensure they apply due diligence when adopting an automated approach.

As suggested, the best way to start is by assessing processes, culture, and tools currently in use. This way a clearer picture emerges of where businesses are today in comparison to where they need to be to support agreed goals and objectives.

At the start of its journey, City Index used manual processes to deploy application code from development to production. Value chain analysis showed that moving code through development environments to quality assurance, then pre-production before finally going live, made up 50 percent of the delivery effort.[5]

Tip When assessing capabilities, don't limit analysis to one element (e.g., test lab provisioning or configuration management). Take a system-level approach to understanding the flow of value and inhibitor across every stage—involving people, processes, and technology.

Demonstrate Business Benefits and ROI

Stakeholders, influencers, and decision makers need to understand the underlying business benefits of adopting release automation tools to support continuous delivery.

Two key metric categories that are used to indicate IT performance and can be useful in supporting a case include the speed or throughput with which applications are delivered and the quality or stability of the releases.

Tip Seek out real-world customer examples from companies that have achieved significant improvements in both release throughput and quality. ING is one such example. They increased release frequency to over 12,000 a month, achieving faster time-to-market with less than six weeks cycle time, but with a greater than 50 percent reduction in incidents.[6]

Any solution must also demonstrate positive economic impact to the department and business—both short term and long term. To support this, CA Technologies commissioned Forrester Consulting to conduct a Total Economic Impact™ (TEI) study and examine the potential return on investment (ROI) that enterprises may realize by implementing CA Release Automation.[7]

---

[5]Full story: http://www.ca.com/content/dam/ca/us/files/case-studies/city-index-bets-on-ca-release-automation-for-it-operations.PDF
[6]http://www.slideshare.net/CAinc/case-study-ing-builds-highly-available-continuous-delivery-pipeline-with-microservices-and-containers
[7]http://www.ca.com/au/collateral/industry-analyst-report/the-total-economic-impact-of-ca-release-automation.html

To better understand the benefits, costs, and risks associated with an implementation, Forrester interviewed five organizations that had implemented this solution in their enterprise. Taken as a whole, this composite company reported a 389 percent return on investment, $8.44 million net present value, with a 2.8-month payback. The study also illustrated that the composite organization's configuration management and testing team saved time and effort on deployments, with savings of six FTEs quantified at $1.22 million over three years.

Note To help determine business benefits and ROI, seek out tools that calculate the full economic impact of release automation. Comprehensive tools provide total benefit analysis, assessing metrics such as increased staff productivity, reduced release errors, improved time-to-value, and reduced auditing and compliance costs.

Execute Tactically, Grow Strategically

Starting small is okay: It's not in anyone's interests to embark on a lengthy company-wide committee to investigate introducing release automation. It's often easier to showcase business value through a pilot.

With this is mind, consider selecting a suitable project to act as your pilot. Many departments start small with a low-risk application that is important but not business critical. The aim is to start a groundswell of support, gather compelling metrics, and then apply lessons learned across larger teams and projects. The Western Union Shared-Service Enterprise IT Operations team took a grass roots approach to DevOps adoption, starting small and measurable. The team used release automation tools to release software into production and then used this as a lever to open the door for broader conversations with its development partners.[8]

Summary

In this chapter, we discussed the automated methods needed to advance continuous delivery maturity—taking teams from manual, scripted processes to more automated, standardized, efficient and agile methods, all while continuously improving both the quality of releases and the applications they deliver.

---

[8]Presentation - https://www.youtube.com/watch?v=JW2eukJuOqw

We also described how as the lynchpin in a continuous delivery ecosystem, release automation solutions must be capable of orchestrating many processes and tools. The ultimate goal is complete continuous delivery optimization across the enterprise and zero-touch deployments, from planning to production.

In the next chapter, we'll examine the DevOps strategies needed to build more agile operations—strategies that extend beyond basic monitoring of applications and infrastructure toward optimizing the all-important customer experience.