CloudBees

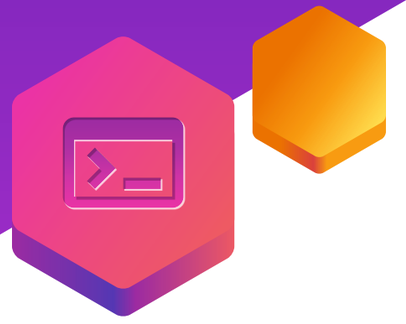
The Definitive Guide to Modern Software Delivery

The demands of today's software economy are rough stuff. The first requirement—innovating at break-neck speed—is hard enough. Add to that fending off disruptive competitors, an ever-growing list of regulatory frameworks, and a worldwide army of cybercriminals. But you're not done! Remember to continuously grow—and satisfy—a user base that expects bug-free-and-perfectly-secure feature rollouts almost *daily*. Don't forget to contain budgets, minimize staff, zero out inefficiencies, drive monetization, and ensure everything you do aligns *perfectly* with your business objectives. In other words, do a million things at once without screwing any of them up. No big deal, right?

So how do you juggle all this without the whole operation collapsing like a house of NFTs? **You need to run a tight ship.** It's your only hope. And to do that, you need a bleeding-edge software delivery life cycle (SDLC).

What Would a Bleeding-Edge SDLC Look Like?

We'll dig into specifics shortly, but to start it's helpful to consider the broad goals of a modern SDLC. What would we need to make the competing demands of software delivery not just *possible*, but (almost) painless?



Working Faster

Speed to innovation, speed to market

The ideal SDLC would empower you to deliver features to your users at the pace your business objectives demand. This means keeping your devs focused on innovation rather than distractions (e.g., bugs, security fixes) so your teams can reliably push rock-solid features into production exactly when you want them to.

Automation

Manual processes are your worst enemy, so you'll want to bake automation (both for processes and testing) into the fabric of your SDLC.

Keep it lean & mean

Also in contention for your worst enemy: inefficiency. You do not have time or budget to waste. You'll need a systemic approach to improving efficiency across your SDLC.

Scalability and resilience

You can't rapidly grow your enterprise if the whole thing collapses under its own weight. Scaling resources inline with your business objectives while avoiding waste, distractions, and points of failure is what it's all about.

Working Smarter

Agile—minus the friction, confusion, & inefficiency

Agile is a must, but we'd want to amplify the good (adaptability, continuous improvement, and customer satisfaction) while avoiding common pitfalls (communication breakdowns, tool/process/staff bloat, and misalignments between DevOps and business teams). Establishing "golden paths" that guide devs through your SDLC is a crucial part of this effort.

Seamless collaboration

Agile is, in part, intended to promote collaboration and communication within and across teams, but it tends to produce the inverse effect. Your people—leadership, devs, operations, practitioners, SREs, front-end and back-end teams—need a low-effort framework for working cohesively toward unified business objectives.

Data-driven decision-making

We're talking about engineers here, so let's use data—monitoring, logging, analytics, etc.—instead of our feelings when we decide what's next!

Painless (but reliable) governance, security, & compliance

Making best practices in these areas ubiquitous, mandatory, and cultural yields a laundry list of advantages over your competitors (who are *extremely* unlikely to have pulled this off).

Bringing it All Together: Maximum Visibility

To get everything above, you're gonna need visibility into *everything*. If you don't have eyes across all your teams, tools, processes, and metrics you can't hope to ensure those things operate in harmony to your benefit.

Your Dream SDLC—Feature by Feature

Now, let's dive into the specific features you'll need to incorporate into your SDLC to reach your modernization goals.

Centralized Visibility & Control

Let's start with the biggest issue: your SDLC is a near-endless array of moving parts that can easily spin out of control or grind to a halt. Your first objective is to make sure those things never happen. But that isn't good enough by itself: you also—on a continuous basis—need to ensure that every component of your SDLC is operating efficiently to the maximum benefit of your business objectives. That means identifying bottlenecks, neutralizing redundancies, accurately predicting release windows, etc.

To achieve these higher-order objectives, you **must** be able to observe and control your people, tools, processes, and outcomes from a centralized POV. Whether you design your own system or implement a turnkey solution, you'll want your feature set to include:

- A complete set of **organizational controls**: centralized user, team, and org management with role-based access control—and SSO to streamline toolchain access.
- A **command-center view** that allows you to track products or releases through all SDLC phases—you should be able to customize this view to suit specific teams or processes.
- The capacity to **dynamically track release progress**, including the various dependencies, number of known defects, and days to delivery. (This is critical, as it allows teams to know where other teams stand.)
- A **dashboard displaying high-level metrics** for release activities such as planning, continuous integrations, and deployments to various environments. Data should pull from internal CI/CD and Release Orchestration (RO) processes as well as external tools to instantiate a single source of truth for your enterprise.
- **Incident metrics for deployed releases** (pulling data from your preferred incident management system).
- **Insights into success rates** across automation and testing protocols.
- **Feature-flag tracking** (more on this below).

Note: Many of the idealized methodologies, tools, and practices described in the following sections are either impractical or impaired if they aren't integrated with a global visibility/control layer encompassing your SDLC. Developing an "all-seeing eye" for your SDLC activates powerful downstream benefits in areas as diverse as CI/CD, RO, governance, compliance, and value-stream management.

Continuous Integration/ Continuous Delivery (CI/CD)

CI/CD is the one-two punch of modern software delivery.

Continuous integration (CI) involves integrating code changes from multiple devs into a shared repository as often as possible (often multiple times a day) before subjecting them to automated building and testing processes. The goal is to grow your project as quickly as possible while catching defects as early as possible.

Continuous delivery (CD) takes it from there, using automation to usher changes through QA, staging, approvals, and ultimately into production (with or without human intervention, depending on your SDLC). The aim with CD is to ensure new features, updates, and bug fixes make it to your customers ASAP while minimizing manual processes—and the risk of breaking anything.

While you definitely *need* CI/CD, not all solutions are created equal. In particular, there's serious more-harm-than-good potential here, since CI/CD pipelines are notoriously convoluted and prone to breakdown. The following chart provides some guidelines for what you should demand of a best-practices implementation.

Things Every CI/CD Tool Should Have

Continuous Integration	Continuous Delivery
Cloud native (e.g., Kubernetes) or on prem, as needed	
Enterprise-grade governance, compliance, & security	
Scalable across any environment	
Everything as Code (EaC)	
Business-class support & reliability	
Access to key metrics across your entire CI/CD pipeline	
Audit & compliance reports	
Workload isolation for teams	Unified object & data model
Visibility & management across teams for admins	Advanced, model-based deployments
Plugin management	Automatic remediation & rollback

Automation

You are probably aware that there aren't a lot of *looms* in use today. That's because effective automation always beats manual processes. While automation is foundational to any CI/CD pipeline, that's the tip of the iceberg where SDLC efficiency is concerned. As a rule, you'll want to automate just about anything that *can* be safely automated.

While the specifics depend on your SDLC and how your teams work, a couple of guiding principles apply:

1. You'll want the freedom and flexibility to automate processes that are highly tailored to your needs
2. You'll want to do so from a centralized control environment, because automation without an orchestration layer can't touch top-down approaches. Ease of use, breadth of use, and consistency of use are your friends here.

Release Orchestration (RO)

Automation (CI/CD) is like turning your SDLC into a freight train. That's great, provided your train doesn't go to the wrong station—or fly off the tracks! And this is a real risk, because a modern SDLC contains an enormous number of concurrent processes. How those individual processes happen is critical—and that's what we've covered so far. But what's even more important is making sure all these processes operate harmoniously and to your benefit. How do you make sure the right stuff happens at the best possible time? And how do you make sure bad stuff doesn't happen at the worst possible time? This is where release orchestration (RO) comes in. And "orchestration" is apt, because—if you're visualizing a conductor in front of an orchestra—you're not far off.

Partly, the goal of RO is to ensure different sections of your "orchestra" (DevOps, business units, etc.) work well, both individually and together. But, as with an orchestra, working together serves a higher purpose. In music, the goal is to control the emotional state of the audience. In DevOps, it's to deliver software in a way that furthers your business objectives. CI and CD are already glued together; RO glues CI/CD to all of the other activities involved in getting code from commit to production. To that end, RO can involve a sprawling set of crucial tools, but any good RO implementation should include:

- **End-to-end transparency**—visibility, insights, and control across all SDLC systems
- **Path-to-production view**—displaying deployment status for each stage of the release
- **Connections for "islands of automation"**—orchestration of any tool, for any app, across any environment
- **Visibility into dependencies across the stack**—management of dependencies across applications and services and associated pipelines and releases
- **Real-time evidence collection**—straightforward auditing/reporting across workflows
- **Governed release process**—ability to map out the end-to-end release process, knowing every task will be performed as expected and all relevant data collected automatically
- **Automated testing**—allow testing to be done in parallel

Value Stream Management (VSM)

What's the easiest way to start a fight? Put engineers and business leaders in a room to discuss priorities. And this is why you need value stream management. VSM starts with a shared understanding that the purpose of your SDLC is to maximize business value and customer satisfaction. With those goals in mind, data collected across your delivery pipelines is used to establish success metrics and KPIs for assessing value throughout the end-to-end process (or "value stream") through which a product, feature, or update moves from conception to delivery. Once you have shared goals and agreed-upon measures of success … bingo! No more fights.

VSM is your best bet for making sure engineers aren't left out of the decision-making process, because it gives executives insight into how their decisions impact the SDLC while giving engineering teams the power to show how their goals and priorities deliver concrete value to the enterprise. VSM kicks in at the planning and ideation stage and doesn't stop until you've got metrics describing application performance and customer feedback; it's about making sure stakeholders understand the business context and value of any action that ripples across your SDLC. A solid VSM solution would provide:

- Analytics that reveal bottlenecks, visualize flow, and improve outcomes
- Metrics and KPIs that establish software delivery performance baselines
- Insights into deployment throughput, success rates, and durations over time

Continuous Governance

You can't run a tight ship if everyone onboard does whatever they want. To avoid that, you need two things: a set of rules and a framework for enforcing those rules. Governance is how you bring these things to your SDLC—but this is easier said than done. It requires the ability to map out your entire SDLC while building the controls necessary to ensure the desired procedures are followed, proper approvals obtained, and necessary evidence captured (for auditing purposes) through every stage of your delivery process. If you're thinking you have no hope of achieving seamless governance without a powerful RO implementation, you're spot-on; governance is a subdiscipline of RO. Examples of good governance include:

- Enforcing organization-wide coding standards
- Defining and enforcing a structured change-management process
- Formalized code review and approval processes

Analytics

It's a given that any organization filled with engineers should be data-driven, and one of the primary benefits of RO is that—by tying together all the people, processes, and tools in your SDLC—it provides timely data that can constructively inform decision making. By collecting data on KPIs and other relevant metrics—such as lead time, cycle time, throughput, and work-in-progress—your enterprise can identify areas where processes can be improved, then track the impact of changes (good or bad) over time. A robust analytics platform would include:

- **Comprehensive pipeline visibility**—dashboards and reporting functions to plan, schedule, audit, and track pipelines, releases, and deployments (with the ability to customize as needed)
- **Release command center**—a bird's-eye view of release statuses, milestone dates, dependencies, pending approvals, test results, progress, environments, and teams involved
- **Workload insights**—data covering historical workload growth across demand cycles
- **Plugin usage**—tracking and monitoring of plugin usage across all your pipelines, upgrade prioritization for your most-used plugins, and audit/review processes for unused plugins
- **Developer metrics**—analysis of code-commit trends, investment areas by type/status, etc.

Progressive Delivery via Feature Management

The point release represents the old way of updating software; you add an entire set of features/fixes to your application then cross your fingers that, on the whole, you're doing more good than bad (in the form of bugs, conflicts, or security flaws). This method has fallen out of favor for two primary reasons: users don't benefit from production-ready features until other unrelated features are ready to go, and a single point of failure can invalidate a whole host of valuable improvements. Progressive delivery, made possible through use of feature flags, eliminates both problems.

Progressive delivery is the process of deploying new features, updates, or fixes individually. By attaching feature flags to individual code changes, improvements can be individually activated (or deactivated) from a centralized control scheme. Critically, rollout can be limited to a *subset of your users*. This unlocks powerful release strategies:

- **Canary releases**—deploy new features to a small subset of users so you can monitor impact and address issues prior to progressively scaling availability to the rest of your users.
- **Blue-green deployment**—run two production environments: one with the current version and one with an updated version. Gradually shift users from the former to the latter, rolling them back if any issues arise.
- **A/B testing**—deploy multiple variations of a feature to different user subsets to compare performance and feedback.

Continuous Security & Compliance

No matter how good your pipeline is, if your approach to assessing, asserting, and evidencing compliance and security are manual, disconnected events, you're going nowhere fast. And the common practice of "shift left"—asking your dev teams to handle compliance-related tasks—doesn't make sense as a solution. Your developers are there to drive innovation; compliance obligations, while necessary, are a distraction from that purpose.

The solution is to integrate security and compliance tools (and checks) into every stage of your SDLC, baking them into Release Orchestration so you default to secure and compliant through every DevOps process. In this scenario, compliance protocols ensure that pipelines comply with relevant standards (GDPR, NIST, etc.) while RO-fueled governance guarantees those protocols are followed. Combining these complimentary solutions unlocks serious advantages over traditional approaches:

1. Continuously assess, assert, and evidence security and compliance across your entire SDLC
2. Automatically contextualize scanning results and provide all teams with a clear list of required actions based on *actual* risk

Most importantly, built-in compliance relieves your dev teams of having to know security and compliance in depth, while minimizing distractions from noncritical issues. They can concentrate on high-priority threats while making defensible decisions about what to fix—or not—based on contextual information.

Living the Dream

The SDLC your teams need to be their best *is* attainable, but how do they get it?

OPTION A

Roll your own using an assortment of open source or standalone tools. Many enterprises find out the hard way that this isn't the optimal approach; you'll have difficulty achieving centralized visibility/control and you run the risk of making your SDLC more rather than less complicated. (Because you're ultimately growing your tool chain using methods that lack enterprise-class support, reliability, or scalability.)

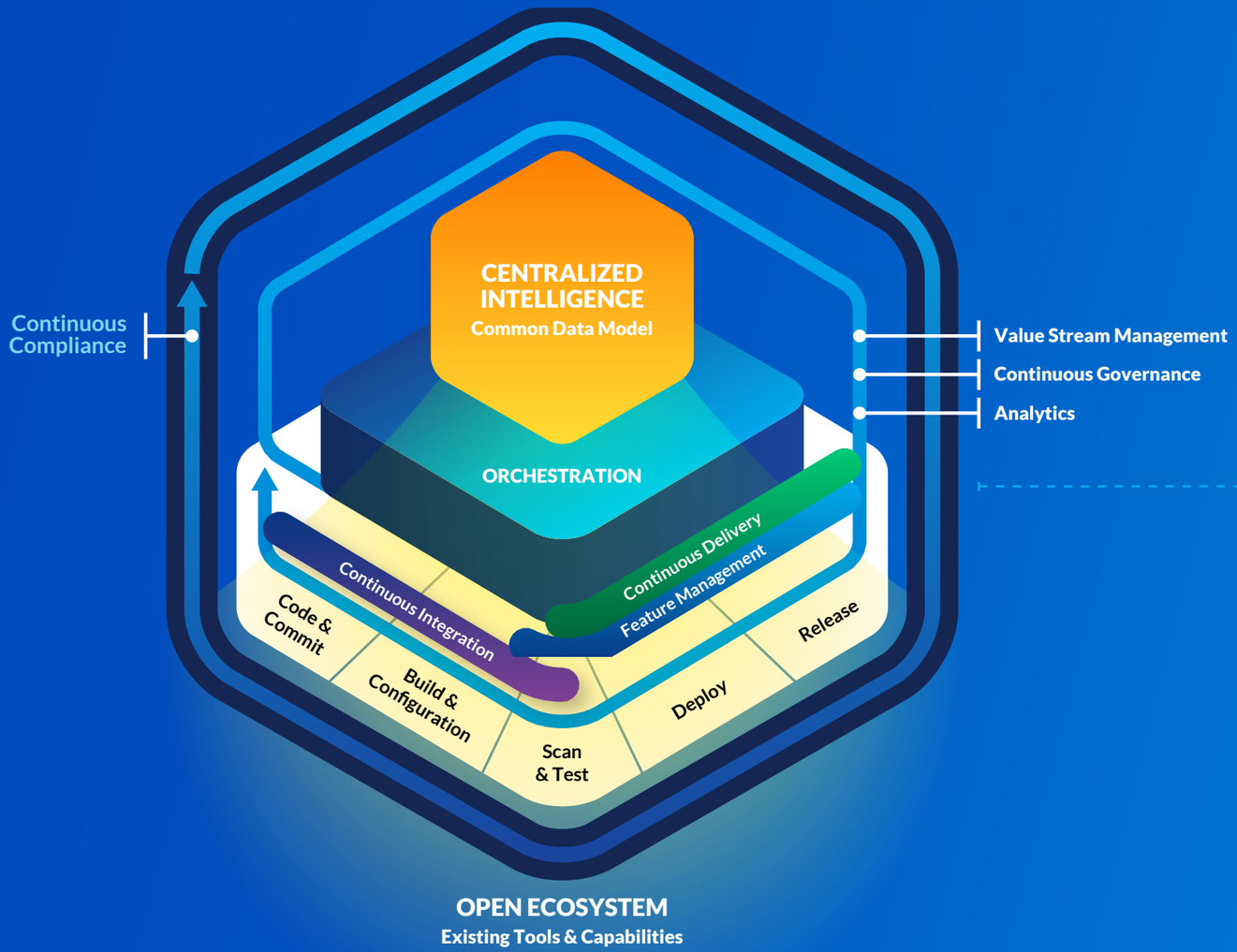
OPTION B

Implement a mature platform that supports your SDLC. Adopting a flexible, customizable platform solution designed to optimize your unique software delivery environment helps teams onboard and innovate faster; work more autonomously; and maintain real-time visibility, governance, and security—while keeping your ROI on a much shorter time horizon.

You want to deliver products to your customers *faster* and *safer*. Implementing the ideal software delivery platform to support best practices will get you there.

The CloudBees Platform

CloudBees provides an enterprise-class experience for all of the features discussed in this guide, so you can run a tight ship with minimal time, effort, and budget. Critically, the platform provides compelling ROI that you should consider before building anything in-house.



Estimated Yearly Savings with the CloudBees Platform*

$5K/user
conservatively, during initial phase of CloudBees rollout

$5M/org
based on industry average of 1,000 devs organized optimally across 142 teams (~7 users/team)

*Results are based on an internal study based on business value assessments.

CloudBees®

Reach out to us to discuss how the CloudBees Platform can transform your SDLC *and* your business.

CloudBees®