Best Practices from Real-World DevSecOps Implementations

OTEEMO

CloudBees®

Best Practices from Real-World DevSecOps Implementations

This document compiles seven lessons learned in the field of DevSecOps by leveraging the combined experience from real-world implementations performed by Oteemo and CloudBees consultants. The DevSecOps high-level promises of speeding up innovation, improved quality, and baked-in security throughout can be challenging, though not impossible, to implement well. While this list of lessons learned is not exhaustive, the goal is to inspire the reader in their quest for modernization by providing ideas on best practices. These lessons are the result of directly participating in and observing the DevSecOps journeys of literally hundreds of organizations across government and private sector companies in health care, software, finance, manufacturing, telecom, and more.

We'll share our experience on how to align on goals and later use concrete metrics to demonstrate progress. We'll delve into the need to treat the CI/CD pipelines as a product, the dilemma of standardization vs. innovation, and the benefits of using service catalogs. We'll share some best practices to secure the toolchain and address complex approval processes in a more modern, efficient way.

In general, DevSecOps enhances Agile by increasing reliability of the system, allowing changes to be rapidly developed and deployed, and allowing security testing and patches/updates in a continuous integration environment with the ability to deploy on demand. Successful DevSecOps aims for continuous compliance – however the organization defines that (e.g., government regulations, industry standards, internal policies). Two primary elements of continuous compliance are:

- A professional, safe approach to making changes.
- An audit trail for oversight and problem-finding after a failure

Organizations achieve successful DevSecOps because they put in place two main things:

- **Trustworthy pipelines.** Products coming out of the software factory can be trusted to live in the wild.
- **Evidence.** Being able to prove it, whenever it’s needed, to anyone who needs it (security teams, accreditors, auditors, etc).

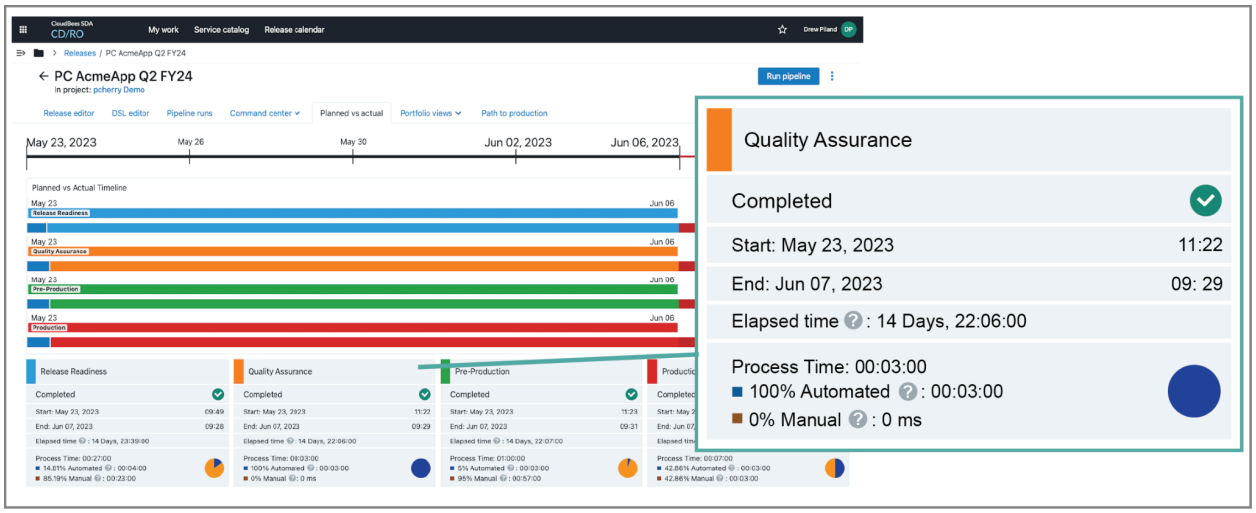
Lesson 1 – Align on Goals and Present Challenges, Show Progress

The old way to build and release software does not work anymore. Customers don't want to wait months for new features to be available, and security issues need to be addressed quickly and efficiently, without taking a toll on the organization. More complex approaches to gate-keeping, like Change Control Boards (CCB), are negatively correlated with both speed and software quality. The traditional ways of CCBs and post-development security audits are antiquated, not just because they are slow, but also because in our experience development teams tend to actively avoid the appearance of the need to trigger those manual reviews and gates. This is typically done by minimizing the scope of the changes performed. Controls and feedback need to be automated as part of the software build. That's a big ask.

For significant changes to be successful, leadership must explain the vision, why it matters, and the expected outcomes, so the intended objectives and impact are understood. For a successful DevSecOps transformation, there needs to be alignment on the vision and why it is essential. Do we need to release more frequently to improve the citizen experience, empower the warfighter, or get ahead of the competition? Do we need to improve quality to increase customer satisfaction, or is securing the software critical for the success of the mission? Determining how to carry out that vision – the necessary process and supporting tools required to realize the desired change – sometimes results in differences of opinion and even resistance. Some entities in the organization might feel more comfortable with the current processes and status quo rather than the proposed changes.

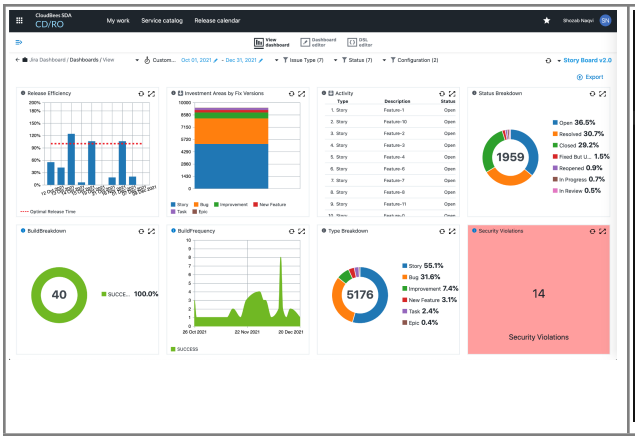
DevSecOps transformations can entail complex environments that need to be modernized iteratively. Therefore, instead of value stream maps that are done statically without getting updated as a project progresses, it is best to map out existing processes and have insights (e.g., time required by the steps in each process) that are surfaced directly in the tool used to coordinate the actual work. There will be a mix of manual and automated stages. What is important here is to keep track of things centrally, have visibility across time and teams, and

make sure the system of record is the same system being used to track the work. This is particularly critical in federal work because of its extensive requirements to keep track of changes and artifacts for security and compliance purposes, often across a variety of projects, networks with different security levels, and contractors. There's a dichotomy between typical government projects and DevOps goals. In our experience, mapping all the steps, including the manual ones, in a single tool dramatically helps to align on the challenges and focus the work where it's needed, as well as provide a way to show progress.

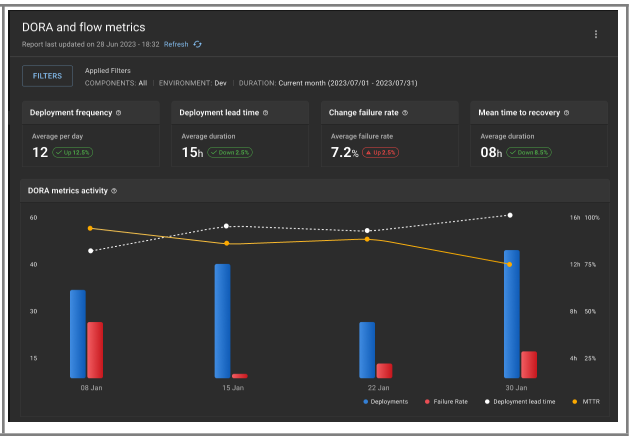


Dashboard visualizing the duration of various steps in a release pipeline

Once the steps are defined and areas for improvement become apparent, effort can be spent to add automation and reduce bottlenecks. While a team of motivated engineers can improve pipelines and gradually increase the automation of each step, doing so without communicating and showcasing their accomplishments may lead to the erosion of management confidence and negatively impact future efforts. Therefore, successful organizations recognize the importance of showing trend data on software delivery performance, both at the single product level and at the organizational level for a portfolio of products.



Application-level dashboard



Portfolio-level DORA metrics

When using software delivery metrics, teams sometimes focus too much on the measurement result they began with (e.g., a low score on code coverage). However, what is important is the trend – the direction in which that code coverage changes over time. Furthermore, metrics should not be feared. Successful organizations do not use metrics for punishment; they use them only for improvement. Their metrics (e.g., deployment frequency) reveal bottlenecks, misaligned resources, and other forms of waste so that they can reprioritize their time and resources. The ultimate goal of their software delivery and DevSecOps is to confidently deliver value to end users.

A word of caution about focusing too much on metrics without conveying the goals underlying them. For example, measuring code coverage does not imply good tests. We have seen tests that had no value beside registering code coverage. Similarly, using story points to measure performance simply inspires more story points, not necessarily value-creating work.

Lesson 2 – Treat Your DevOps Pipelines as Products

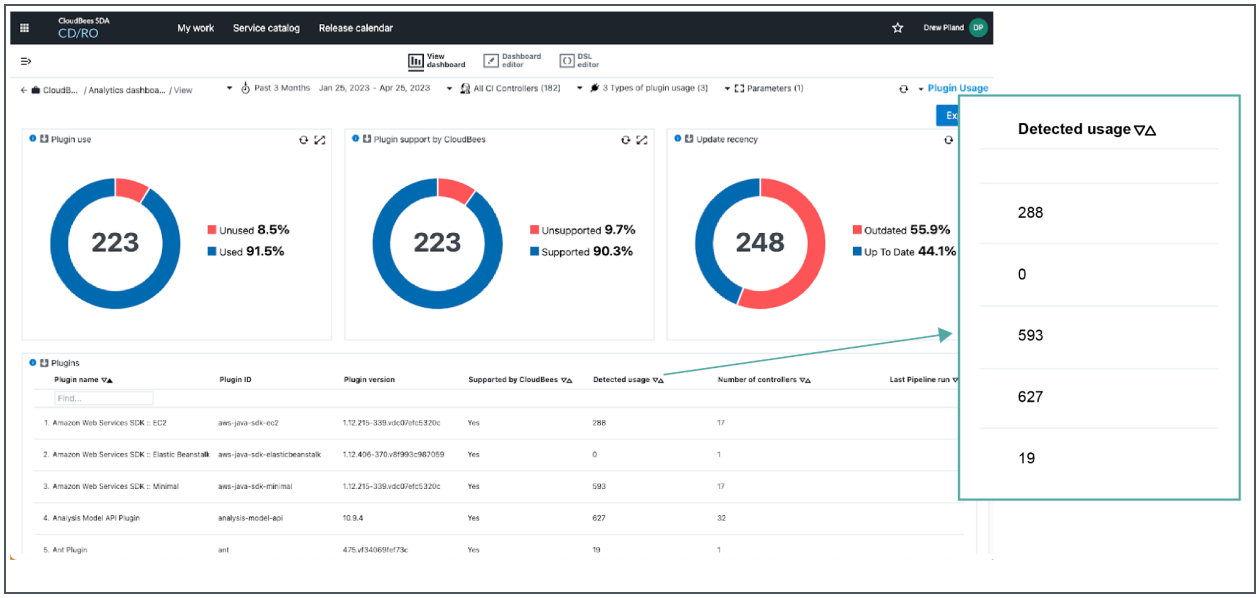
Successful organizations typically establish a group of people to create pipelines and provide features for other teams that need the ability to build, validate, package, and deploy their applications. We have seen this group go by many names: pipeline team, DevOps team, systems team, modern delivery team. Whatever the name, the team's products are software pipelines, the glue between all the components, and the enforcement of best practices for the organization.

Importantly, these pipeline teams consider their pipelines as products. Consider this analogy: a user-facing product involves a product owner who understands end users and actively involves them. It is key to get early feedback and work with users to make sure that the product team prioritizes the right needs and the product is not developed in a vacuum. Similarly, the pipeline team needs to bring its stakeholders together regularly to define the roadmap and refine the backlog.

Another result of treating the pipeline as a product is that the pipeline team assumes proper responsibility for taking care of the technical debt of that product. Like all software products, technical debt keeps accruing unless it is diligently tackled. In the long run, not handling that debt will mean the pipelines run on older technologies, are harder to maintain, and erode the product team's ability to innovate quickly and with confidence.

A specific example we have observed of how technical debt can accumulate is when DevOps teams are so focused on adding new features and improving existing ones that they forget about (or avoid) removing features that are underutilized or whose value has decreased over time. One effective way we have seen teams address this type of technical debt is by tracking the usage of plugins in CI/CD pipelines. If a plugin is seldom used, the pipeline team can reach out to the users of that plugin to understand why and how it's used and explain that there might be newer or more acceptable capabilities to leverage instead.

The screenshot below illustrates the utilization of plugins across various projects. The visual representation highlights a significant disparity in usage rates, indicating an opportunity for streamlining by potentially eliminating underutilized plugins.



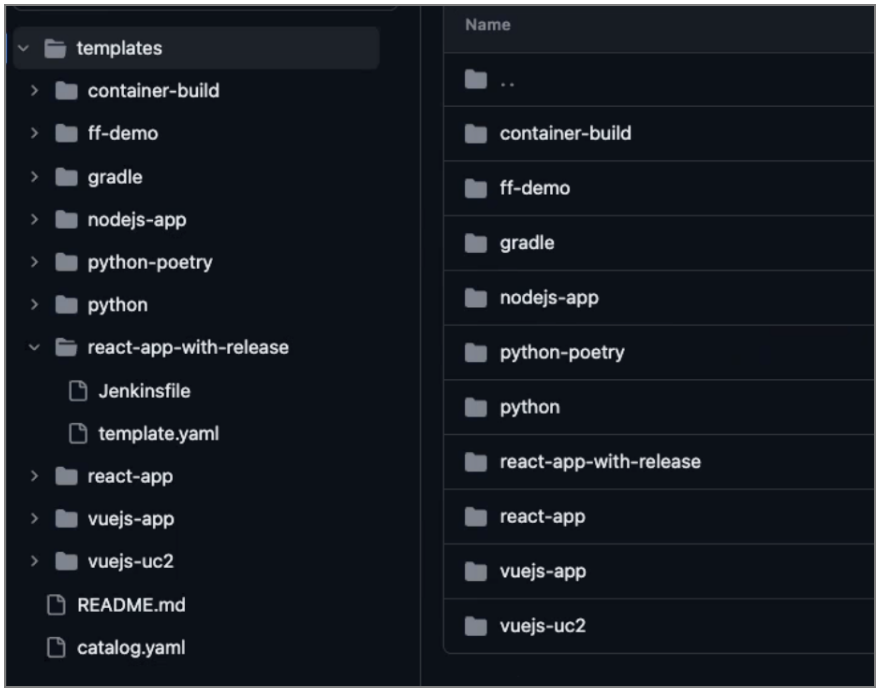
Dashboard showing real-time analysis of plugin usage across projects

Lesson 3 – Balance Innovation With Consistency and Governance

In the previous section, we discussed treating the pipelines and tooling as a product in itself. But it's also a special kind of product, because typically the users of the pipelines are highly technical development teams that can be the source of great ideas and able to make impactful changes. Successful organizations make the most of this innovation by establishing pipelines with appropriate guardrails that still allow the flexibility for diverse teams to be creative.

There are two countervailing forces at play here, between encouraging innovation from development teams to add or change features in the pipelines, and the overall need for governance and consistency at the organization level. In cases where development teams are told to use the pipelines as-given, being overly prescriptive will ultimately limit innovation in the organization by requiring all teams to conform to the existing feature set of the CI/CD solution. The good news is, clear processes and tools can help.

We have seen organizations reach a successful middle ground by providing pre-built CI/CD solutions, having the initial feature set adhere to the internal governance, while also allowing teams the opportunity to make changes for their own use cases. Once the team implements its desired features, it can contribute its findings back to the overall organization. In this model, there must be guardrails in place to protect the organization, so that teams can't remove or disable a stage simply because it's inconvenient. For example, we saw teams in one large organization temporarily disabling security scans because they were failing altogether by exhausting build resources. The security team was unaware and the situation lasted for months.



The screenshot above offers a comprehensive view of a central, self-service repository housing an extensive collection of continuous integration templates. These templates are available as “grab and go” for development teams to incorporate into their workflows, thereby reducing rework and boosting efficiency.

In this model, there is a mix of centralization and autonomy given to the teams. Teams can innovate by submitting pull requests or merge requests for the features they need. The pipeline team can review the changes, propose modifications, and provide it as an approved template for everybody else to use. Tools like CloudBees CI enable the definition of workflows leveraging GitOps principles and defined as code. CI/CD can be created from a catalog of templates, providing rapid delivery and consistency for product teams, with the confidence that organizational standards are being followed.

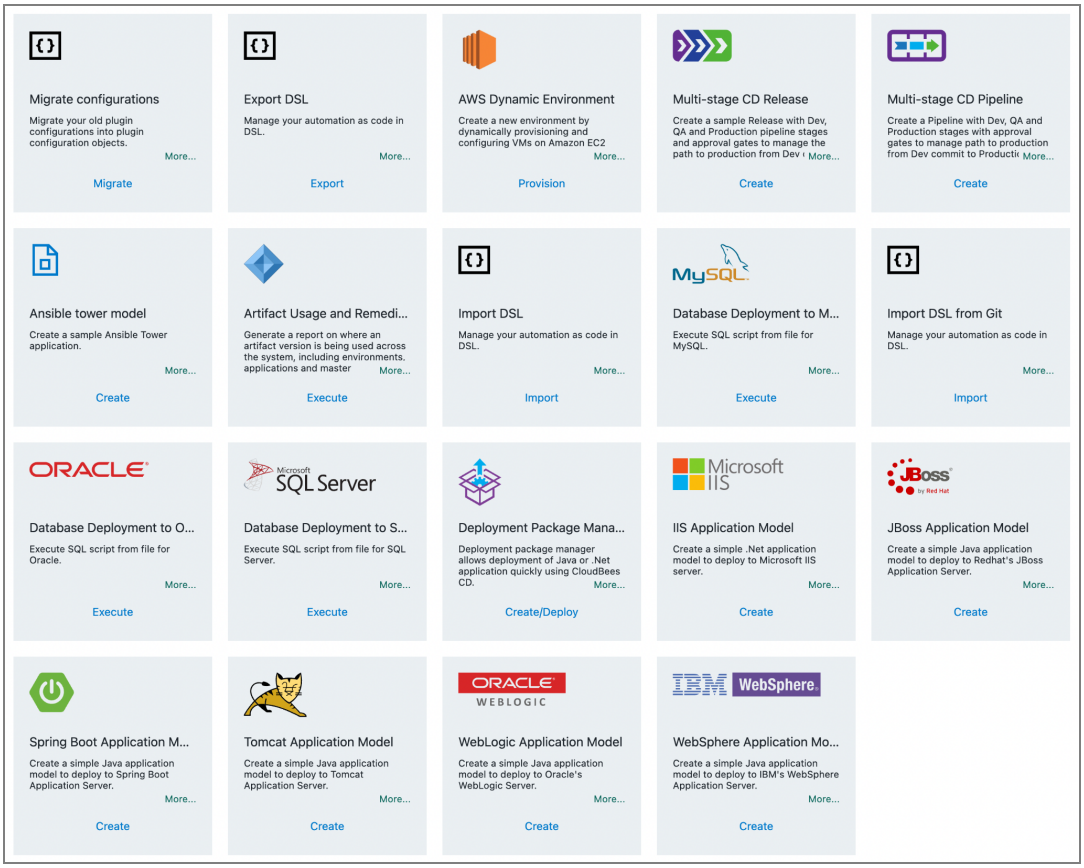
Lesson 4 – Leverage Service Catalogs

In order to increase speed to market, responsibilities that were historically the purview of specific teams need to be achievable by the product teams. For example, provisioning a database used to be the responsibility of the DBA team, pipeline provisioning fell to the DevOps team, and new machines were the domain of Operations. This model provides control and consistency, but it is inefficient, is detrimental to the speed of innovation, and can also demotivate teams by having them wait for some other team to process their request (when they're not ignored or lost altogether).

In large organizations with complex requirements, it is nonetheless unrealistic for product teams to have cross-functional knowledge in all these areas, and adhere to all the non-functional requirements.

A better solution is to provide service catalogs as an approved self-service offering to development teams. Development teams can then instantiate those services autonomously and confidently, knowing that the resulting entities follow the necessary guardrails. In the case of pipeline creations from a self-service model, the security teams know that the development teams have all the traceability required and needed to support approvals.

Displayed below is an example of a comprehensive service catalog designed using CloudBees CDRO. The various services are meticulously templated, cataloged, and organized for optimal accessibility (self-service). In this way, development and release teams do not have to build and rebuild from scratch, and they have the confidence that the templates are already vetted and have the proper configurations and other guardrails required by the organization.



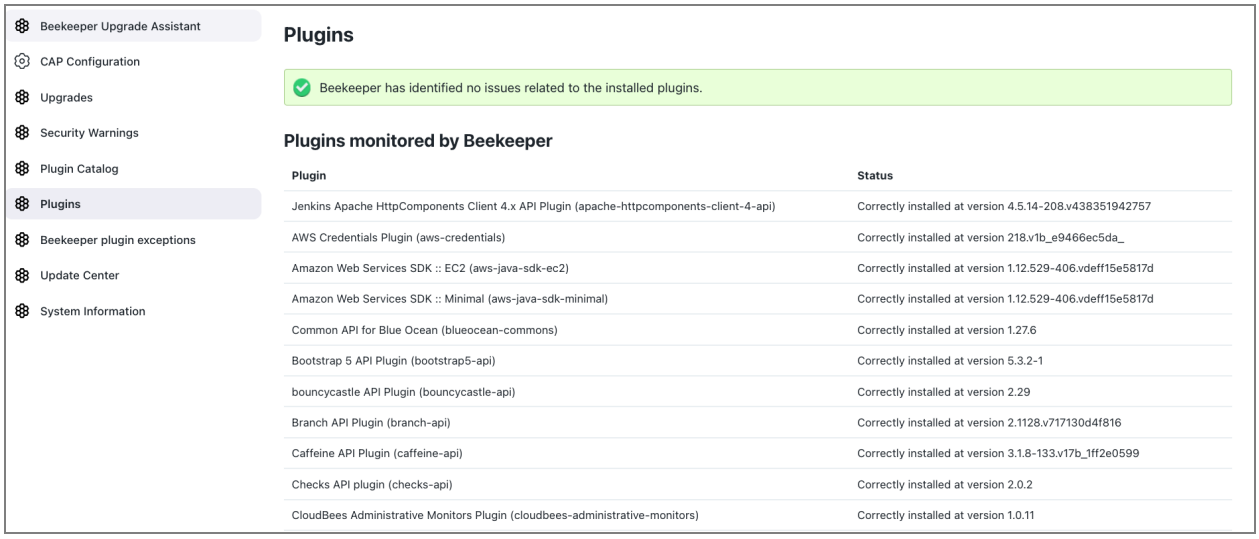
Lesson 5 – Secure the Toolchain Through Careful Tooling Choices

The purpose of software factories is to package source artifacts into a runnable product while ensuring that quality and security of the solution are in line with organizational requirements before deploying them in the different environments. These tasks are typically done by leveraging numerous tools. Instead of an all-in-one CI/CD platform, good software factories require multiple tools, tools that can be swapped in and out. They use the tool that is best-in-class for its function. They value the ability to exchange and/or add a tool as necessary. However, the more tools that are used (security scanners, code coverage reporting tools,

GitOps plugins to help update manifests), the more opportunities for a component to become a vector for a bad actor. Even with a vetting process for every tool and command line utility, the addition of each tool increases the likelihood of problems. To increase the trustworthiness of the toolchain, successful organizations improve the intake vetting process and make sure to perform proper ongoing maintenance of the toolchain.

A tool like Jenkins is recognized as being highly customizable with an enormous variety of plugins. The diversity of plugins can be both a blessing and a challenge. In the federal government, the Authority to Operate (ATO) process imposes the same level of Certificate to Field (CtF) rigor on plugins as it does on any add-on software component. All plugins that are to be deployed, used, or integrated into a restricted environment (e.g., related to the Department of Defense or FedRAMP) have to pass code quality and security scans, software bill of materials (SBOM) vetting, and provenance checks before they can be certified for use. Furthermore, any integration "glue" code to integrate/activate a plugin for the parent software also needs to be part of the CtF equation. To solve the challenges around vetting and maintaining the plugins, CloudBees has created a list of pre-vetted plugins to improve trust, streamline adoption, and simplify maintenance efforts. The pre-vetted plugins provide a secure tooling solution for the CI/CD orchestrator, which can be leveraged to comply with the DoD DevSecOps reference architecture (section 4.2.1). Such trusted tooling packages significantly limit plugin risk.

Below are some examples of the plugins that CloudBees verifies to ensure both functionality and security.



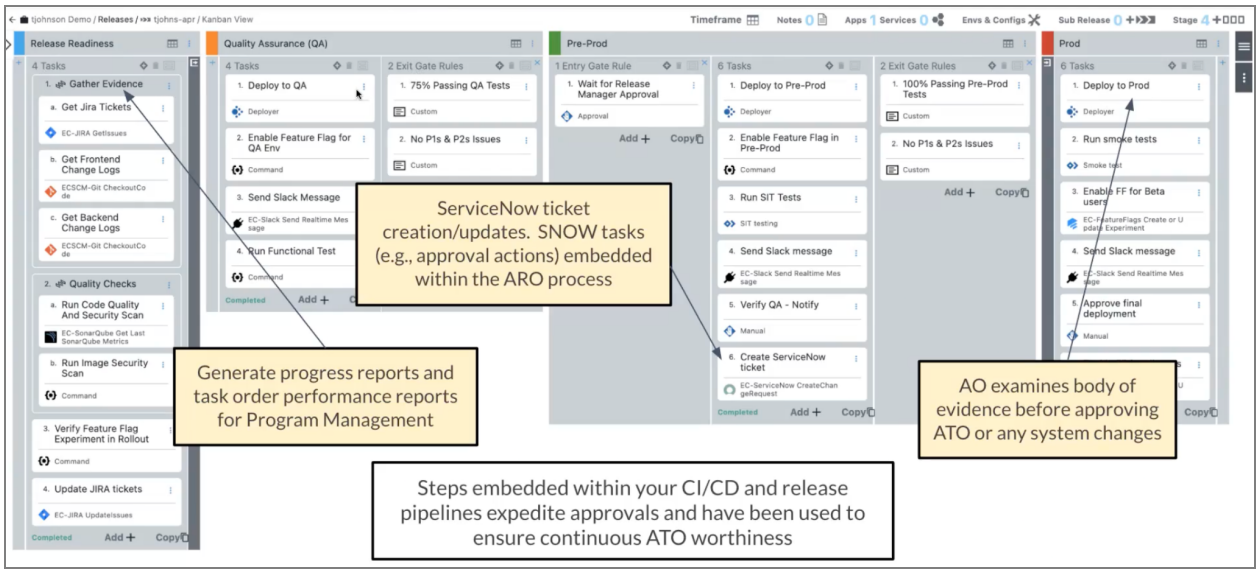
Lesson 6 – Traceability, Certification, and Convergence

In a DevSecOps environment, security teams' time and effort are best spent defining acceptable processes and automated security gates, versus checking about-to-be-released software and being a bottleneck to throughput. That's the concept with shifting security left: it starts with a mindset change that security is the responsibility of everybody in the product team. This

includes incorporating the constant practice of scanning work very early in the development cycle to avoid poor quality code from progressing further down the pipeline undetected. This reduces resolution costs, as findings can be linked to recent changes rather than buried in multiple days (or weeks) of coding that makes changes and accountability more difficult. Additionally, testing for security early means minimizing future rework because of early detection at the point the vulnerability is created.

While the shift-left principle and automated gates are gaining credence as best practices and becoming more mainstream, certain systems and environments, such as those requiring an Authority to Operate (ATO), require particular extra scrutiny. Federal agencies and contractors that successfully navigate the ATO process typically establish “audit-ready pipelines” that provide visibility across their software delivery function, from ideation to release to monitoring. While the goal is to reach a capability of continuous ATO (cATO) – which requires more than the software factory piece but also real-time monitoring and the ability to respond to auditable events that match a compliance ruleset – the software factory pipeline needs to be traceable and auditable to make manual approvals much easier. Since not all manual approvals can be fully removed immediately (for pragmatic or regulatory reasons), it becomes imperative to embed the required manual approval gates and reviews into the workflow of the overall processes, and to generate – directly from the tooling – the appropriate evidence that people need for approval decisions.

The dashboard visual below was taken from CloudBees CDRO. It shows that having real-time insights into the throughput of release pipelines is extremely important to making decisions about the readiness of applications and systems to advance into Production.

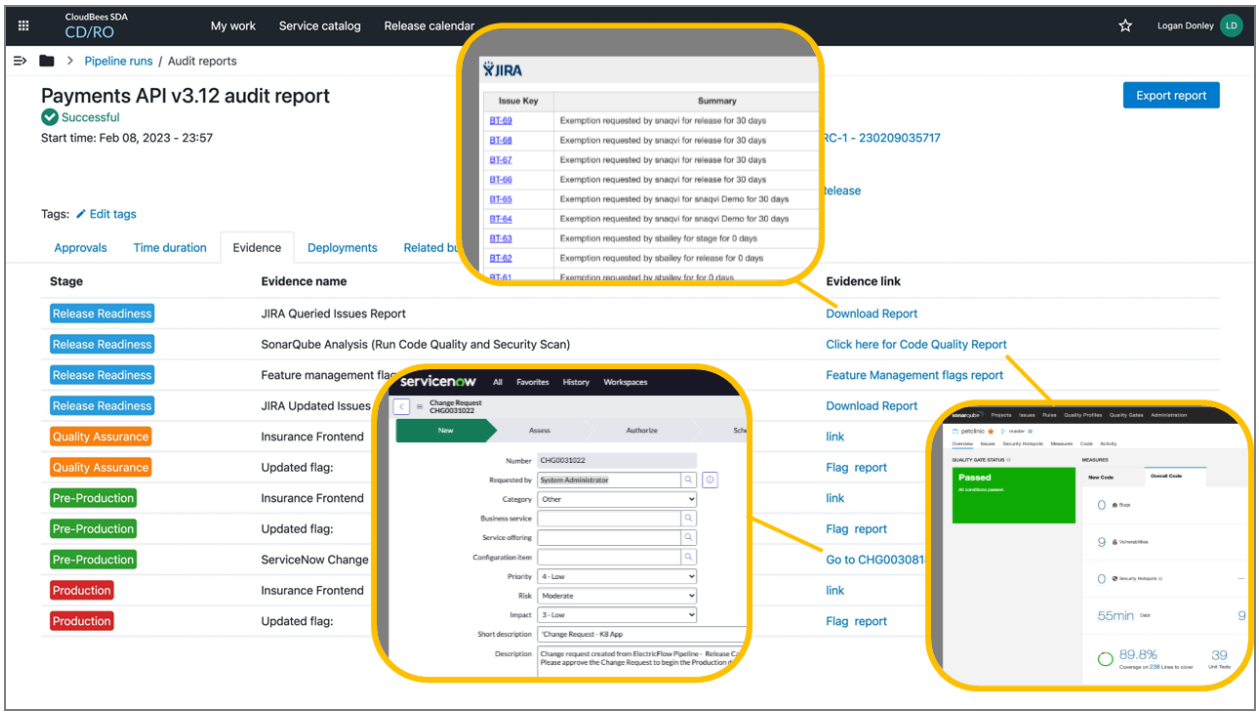


The automatic collection of all the evidence and artifacts in a single location enables security experts to focus on reviewing the data efficiently and confidently, instead of spending time hunting down information and being presented with partial data from product teams who simply

want to pass a perceived deployment hurdle. The product teams also have more time to focus on productive work, thereby boosting productivity and morale.

Similarly, we see federal agencies (smartly) move away from the traditional CCB model to one where approvers have a single pane of glass to review evidence, artifacts, and audit information as soon as it's generated. Therefore, instead of reviewing proposed changes at biweekly CCB meetings, which often get sidetracked or postponed, they can make approval decisions at any time because the required information is continuously available. Security teams become enablers of "trust at speed" instead of being a drag on software delivery.

Below is an example of a comprehensive evidence portal within CloudBees CDRO that collects and makes available the necessary artifacts, such as a list of Jira tickets requesting exemptions, a code quality report from SonarQube, or a record of ServiceNow change requests.



Typically for DoD programs, an Information System Security Manager/Officer would require a CtF for any capability built through pipelines deploying into a DoD environment. Sometimes the development cycle and footprint of certain capabilities (e.g., a collection of multiple services) may span both unclassified and classified environments. This places additional burden on the DevOps teams responsible for providing pipelines as a service. These teams are now burdened with tracking the SBOMs of the build tools (FOSS libraries, dependencies) on top of the direct dependencies for the applications/services. All this adds to the complexity of maintaining and providing platform engineering services that could span multiple product teams. It is also important to note that there may be some variances between the body of evidence requirements across programs, which may require these teams some customizations to comply with these requirements. Given these unique challenges in the federal/DoD environments, a platform that

has been built to cope with these compliance requirements can provide significant value to these engineering teams. The CDRO platform addresses these challenges by codifying processes into the pipeline and including monitoring and tracking dashboards that put product owners, cyber, platform engineering, and development teams on the same page by providing a homogenized view of the delivery layer.

Lesson 7 – Be Wary of Over-Engineered Custom Solutions

Teams have typically been responsible for reliably delivering value to stakeholders and been evaluated based on system performance, not the CI/CD and processes to get there. To make delivery reliable and repeatable, and to overcome challenges they encounter, teams commonly develop custom solutions targeting specific problems they confront and the specific systems they maintain over the years. However, we've seen this practice regularly lead to one-of-a-kind system creation (snowflakes) that rely on individuals very skilled in the customizations introduced over those years.

An example we frequently come across is when teams design the CD side of pipelines to be highly customized to a specific team. Though well-intentioned, this can become a bottleneck for innovation and hamstring the overall solution by being challenging to scale and maintain. The best CD solutions use GitOps, automate the deployments, and are non-opinionated on how to handle things like configuration management, resource management, or secrets.

Another example of over-engineering comes from organizations using advanced deployment strategies such as blue-green, and routing specific users to different versions of an application. We have seen clients engineer in-house solutions, relying on complex custom setup, that take way too long for teams to understand and drastically increase the likelihood of deployment errors. While it checks the box for supporting blue-green deployments, it lacks simplicity and elegance. It's better to adopt industry standard components and processes, which decrease internal long-term maintenance cost and make it easier to benefit from ongoing industry innovation.

Sometimes the perspective of an outside consultant is the best way to illuminate the art of the possible to an existing team. Their expertise can bring experience in the form of best practices and thought leadership to provoke thinking in areas that the organization's internal teams haven't been required to consider.

Conclusion

The lessons we've laid out above were gleaned from DevSecOps deployments by real-world customers. Even though their individual circumstances were diverse, they share certain ingredients of success. We hope that we've illustrated them clearly and in ways that can be adapted to your own pathways to success.

Links to Additional Resources

- Definitive Guide to Modern Software Delivery
- Accelerating the ATO Process with DevOps
- The Engine Inside Successful Software Factories
- Seven Tips for Creating Audit-Ready Pipelines
- Additional CloudBees Whitepapers
- Oteemo Resource Center
- Additional Customer Case Studies
- DoD DevSecOps Reference Architecture

Contact Us

We'd love to hear from you, so please don't hesitate to reach out with questions or to talk with our DevSecOps professionals.

OTEEMO

oteemo.com/contact-us

or

Brice Dardel

CloudBees.

cloudbees.com/contact-us

or

Ben Chicoski

Thank You!

OTEEMO

CloudBees®