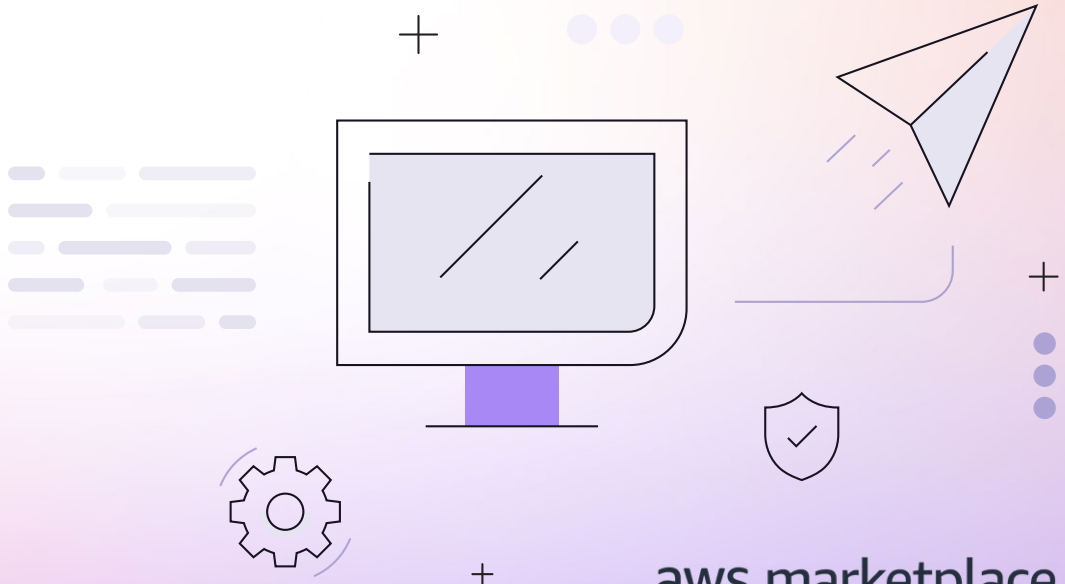
GitLab

# How automated software delivery makes DevSecOps faster and easier



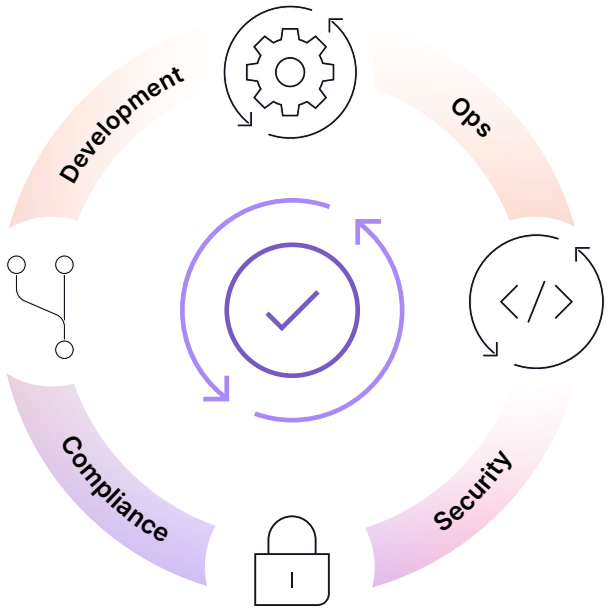
aws marketplace

# Introduction

As organizations race forward in their cloud migration and digital transformation efforts, development and operations teams have come under tremendous pressure. Tasked with innovating to unlock new business opportunities, they often find themselves hampered by large, monolithic, brittle applications that don't talk to each other, siloed teams that don't collaborate, too many manual processes, and multiple complex tools. According to GitLab's 2021 DevSecOps global survey, nearly 50% of operations professionals said their teams use between two and five monitoring tools.[1]

Part of the solution is to adopt a modern DevOps methodology, merging development and operations into a continuous integration/continuous deployment (CI/CD) pipeline. A majority of developers say they're releasing code 2x faster than before, thanks to DevOps — up 25% from (pre-pandemic) 2021.[2] But achieving success with cloud-native transformations and application modernization requires moving away from disconnected, manual processes dependent on legacy source code management (SCM) and CI tools to achieve essential automation that increases speed to value.

To deliver higher-quality applications more quickly and efficiently, your development, ops, and security and compliance teams need to work together in the same tool — one that lets them continuously manage, integrate, verify, and release changes while managing on-demand environments, all through one interface, one platform, and one data model for maximum speed and efficiency. In other words, they need to automate the software delivery lifecycle to speed adoption of cloud-native Kubernetes, achieve faster velocity with lower failures, and improve developer productivity by eliminating repetitive tasks.




**TIP:** To deliver higher-quality applications more quickly and efficiently, your development, ops, and security and compliance teams need to work together in the same tool.

# The path to CI/CD is paved with speed bumps

Those are the goals, anyway. But how to get there? DevOps teams face a number of challenges that make it difficult to achieve the kind of seamless CI/CD pipeline that organizations need to succeed. These include:

**Complexity:**

Developers have to learn multiple languages and multiple tools, as well as work in multiple environments. They really just want to focus on their primary task — coding — and not have to switch between tools.

**Cloud migration:**

Taking applications to the cloud requires rethinking how your applications are going to work in a new paradigm and how they will need to be secured.

**Time to market pressures:**

There's no time for monolithic updates or slow, manual processes. Meeting high customer expectations requires that features and fixes arrive on a frequent basis.

**Accuracy and best practices:**

Developers who are pressed for time might not perform sufficient testing, resulting in errors and delays.

**Governance:**

All too often, manual processes are not documented, so there's no audit trail and no easy way to reverse deployments when something goes wrong.

**Developer experience:**

Development talent is costly and difficult to come by, especially for legacy industries. This means that smaller teams have to be more productive. At the same time, developers don't want to be burdened with rote tasks.

**Security risks:**

As IT environments become more complex, the risk surface grows, making it difficult for developers to secure data and applications.

**Configuration:**

Downstream configurations (e.g., firewall, policy) are frequently stored and managed in separate systems, not always accessible to developers who may not even know their apps can be affected or outright blocked.

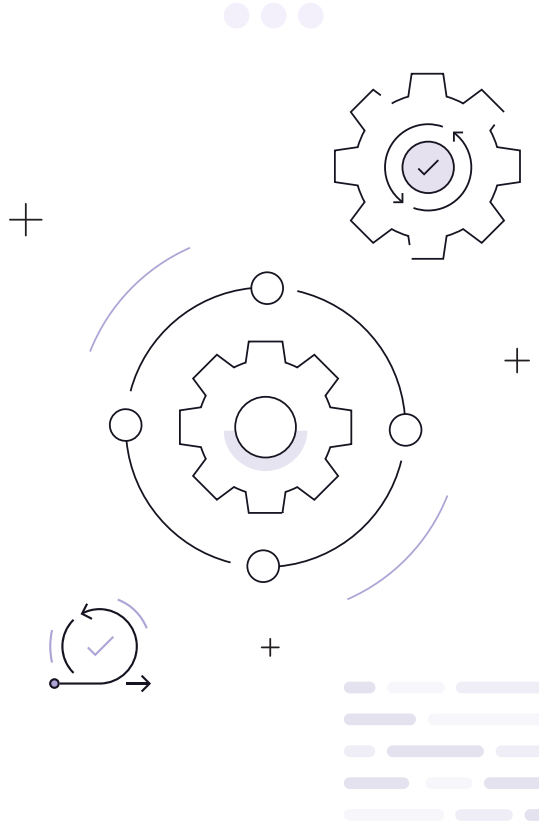
# How automated software delivery overcomes these challenges

The benefits of automated software delivery — which involves automating as many processes within the software development pipeline as possible — are multifold. It helps you speed up your cloud-native Kubernetes adoption by eliminating click-ops and introducing controls essential for cloud-native adoption. It makes every change shippable with more testing so that errors are detected earlier, reducing risk. And, most important, it unleashes team productivity from manual work streams, minimizing repetitive tasks so they can focus on value-generating work. Adopting automated software delivery can help DevOps teams overcome their most complex challenges in three important ways:

01

**It provides continuous integration and verification.**

Automating code, build, and testing to incrementally update for every change accelerates digital transformation by allowing teams to build high-quality applications, at scale. Automation reduces risk by detecting errors early, and it facilitates parallel builds and merge trains to improve developer productivity while maintaining governance.



02

**It reduces complexity.**

The ability to create repeatable and stable environments automatically can greatly reduce the complexity of working in the cloud and minimize the risk associated with manual infrastructure configurations and click-ops. Automating infrastructure helps you release faster and recover from errors more quickly. It also allows for modularizing and simplifying the configuration of add-ons and security policies through git-ops. Importantly, automating change auditing and governance guardrails provides a higher level of security, compliance, and accountability.

03

**It ensures continuous delivery.**

Automating the application release process makes software delivery repeatable and on-demand. Every change is “releasable,” so your teams can progressively deploy changes to minimize disruption and get feedback faster by testing changes on a subset of users.

# What can be automated?

On a state-of-the-art DevOps platform, just about any process can be automated, from build through production. Here are just a few examples:

**At build:** Automation can take care of the basics of configuring the build environment and repository, so developers can focus on what they do best: coding apps.

- Build automations include:
- Server and storage configuration
- Infrastructure code and GitOps
- Infrastructure configuration policies for multiple environments

**During testing:** For the past three years, a majority of respondents to our DevSecOps survey have resoundingly pointed to testing as the area most likely to cause delays.[3] Testing benefits tremendously from automation, which makes it faster, more accurate, and more thorough while reducing swivel-chair context switching. Quality assurance (QA) and compliance testing can be programmed once and performed over and over. AI can even auto-discover source code to check for errors. Automated testing jobs should include:

- Static analysis and automated code reviews
- Automatically scan for code quality and security with every commit
- Identifying security issues in containers
- Analyzing project dependencies and security issues
- Scanning license dependencies
- Detecting credentials and secrets exposure
- Running security analysis of multiple programming languages
- Specific unit tests for the language and framework
- Integration tests
- Performance testing

**Reviews and approvals:** Once testing is complete, you can easily automate the process of spinning up an ephemeral environment to run application security testing, analyze code, and check for potential security issues. For example:

- Automated code review
- Reviewer assignment and routing
- Approval and validation rules and enforcement

**At production:** In this final stage, you can automate incremental deployment to avoid disrupting the user experience during application rollout.

- Incremental canary or blue/green deployment
- Automated rollback

# How GitLab and AWS work together to automate software delivery

As a certified AWS Advanced Technology Partner with DevSecOps Competency, GitLab CI/CD is a proven model for customer success with the leading cloud platform.

GitLab offers a number of platform-specific implementation patterns to ensure a consistent and reliable continuous deployment on AWS, including site reliability engineering, AWS Elastic Kubernetes Service (EKS) cluster provisioning, and more. GitLab's close integrations with AWS enable workflows for every workload to help you develop better cloud-native applications faster. Some examples include:

## Virtual machines:

Amazon Elastic Compute Cloud (EC2) provides scalable AWS cloud computing capacity, allowing GitLab to scale jobs across multiple machines. When used together, GitLab on Amazon Graviton 2 instances can significantly reduce infrastructure costs.

## Serverless:

With one click on GitLab, AWS Fargate enables scalable serverless container deployments. Migrating to AWS Fargate can help organizations reduce effort in managing infrastructure, which helps to reduce administrative burden, optimize compute resources, and save on infrastructure costs.

## Kubernetes:

GitLab CI/CD offers integrated cluster creation for AWS EKS, the only Kubernetes service that lets existing AWS users take advantage of the tight integration with other AWS services and features. GitLab also supports Amazon EKS-D in hybrid environments.

## Event-driven compute:

AWS Lambda is a computing service that runs code in response to events and automatically manages the computing resources required by that code. GitLab supports the development of AWS Lambda functions and serverless application switch AWS Serverless Application Model (AWS SAM) and GitLab CI/CD.

## Containers:

Save time when you run AWS commands from Gitlab CI/CD and automate docker deployments with GitLab's CI templates on AWS Elastic Container Services (ECS).

## .Net:

GitLab enables CI/CD for .Net applications on AWS. You can automatically deploy containerized applications, including serverless resources, with GitLab on AWS Lambda or AWS Fargate.

# GitLab

AWS customers can choose from two deployment options: Install, administer, and maintain your own GitLab instance that runs on everything from bare metal, VMs, and containers on AWS with GitLab self-managed; or choose GitLab SaaS, which requires no installation, so you can sign up and get started quickly.

Automated Software Delivery with GitLab on AWS enables your teams to eliminate manual and repetitive tasks to improve the velocity of your software development lifecycle, deliver high-quality applications at scale, and improve collaboration between development, operations, and security. GitLab provides Auto DevOps, which are prescribed, out-of-the-box CI/CD templates that auto-discover the source code you have. Based on best practices, it automatically detects, builds, tests, deploys, and monitors your applications so your DevOps team can get more done faster, more efficiently, and more cost-effectively.

**To learn more about automating CI/CD with Gitlab on AWS, visit Gitlab in AWS Marketplace.**

[1, 2, 3] GitLab, 2021 Global DevSecOpsSurvey


GitLab