The enterprise guide to end-to-end CI/CD governance

How to build governance and security into enterprise CI/CD pipelines

WRITTEN BY GITHUB WITH ❤

Preface

Continuous integration and deployment (CI/CD) is one of the most common and powerful ways for organizations to deliver high-quality software faster to users.

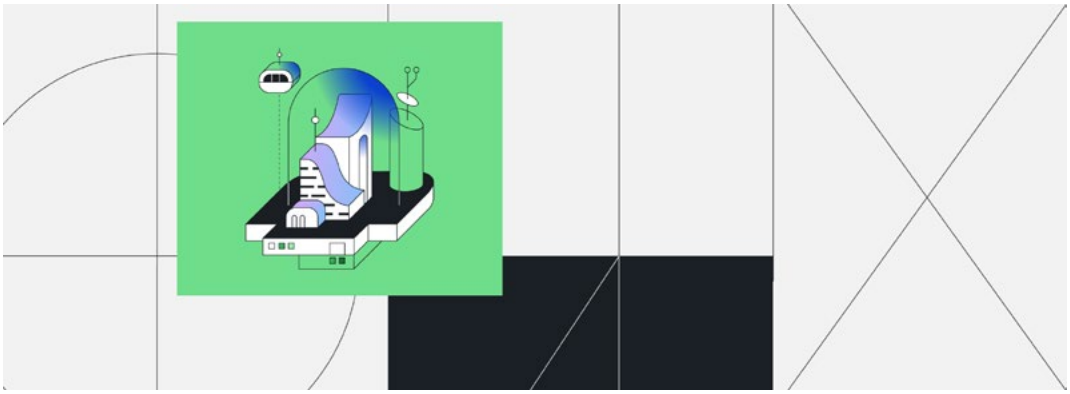
However, CI/CD also introduces challenges when it comes to handling, addressing, and mitigating governance and security risks within a faster-paced environment.

In this guide, we'll explore CI/CD governance strategies to help your organization secure build, test, release, and deployment pipelines without compromising speed.

What's inside

What is CI/CD governance and why is it important?

CI/CD governance refers to a set of processes for designing control points within a CI/CD pipeline. This often includes a combination of access management, policies, automated tests, and human review points. The primary goals of CI/CD governance are to improve operational security and ensure all changes are monitored, approved, and documented for traceability.



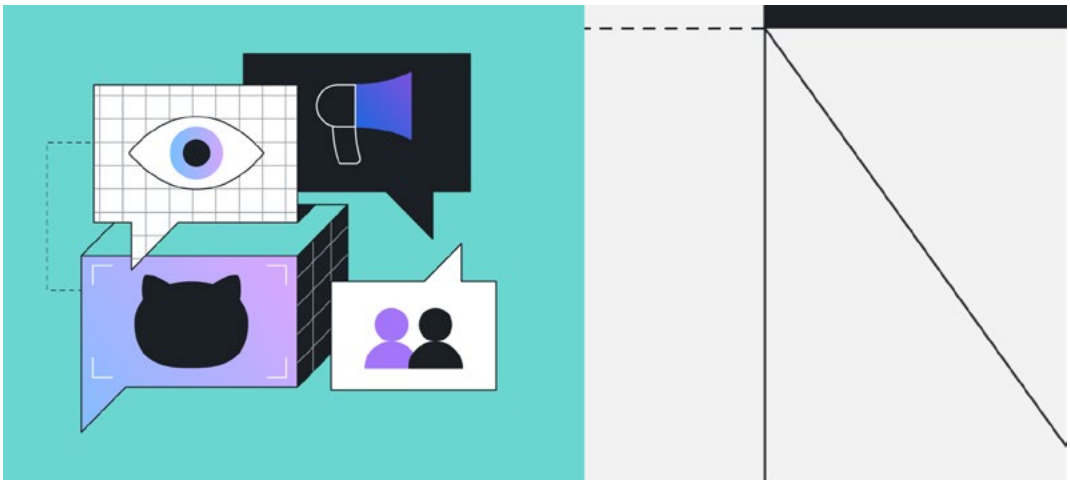
Understanding CI/CD governance starts with having a firm grasp of how CI/CD pipelines work. A CI/CD pipeline consists of a series of automated workflows, systems, and processes designed to help deliver changes from a developer's local environment to production quickly and reliably. This includes:

- **Continuous integration** (**CI**) where automated workflows are used to **build**, **test**, and **integrate** code changes within a shared repository
- **Continuous deployment** (**CD**) which automatically deploys new code updates to production environments

In enterprise settings, this level of automation promises greater speed, better feedback loops, and more operational efficiency. Critically, a CI/CD pipeline that deploys daily or hourly should not have any materially new controls or processes that aren't in place with a traditional monolithic system that deploys quarterly. However, this higher-velocity development model can introduce challenges when it comes to handling governance and security risks.

One common issue stems from development teams' use of open source software (OSS) in their supply chains. Without the right checkpoints, audits, and automated tests, they won't know if or when a vulnerability might impact a key dependency in their code.

The use of OSS in organizations isn't unique to organizations leveraging CI/CD—more than 90% of companies today use open source software. In a CI/CD environment, however, organizations need to deal with open source risks at a faster pace.

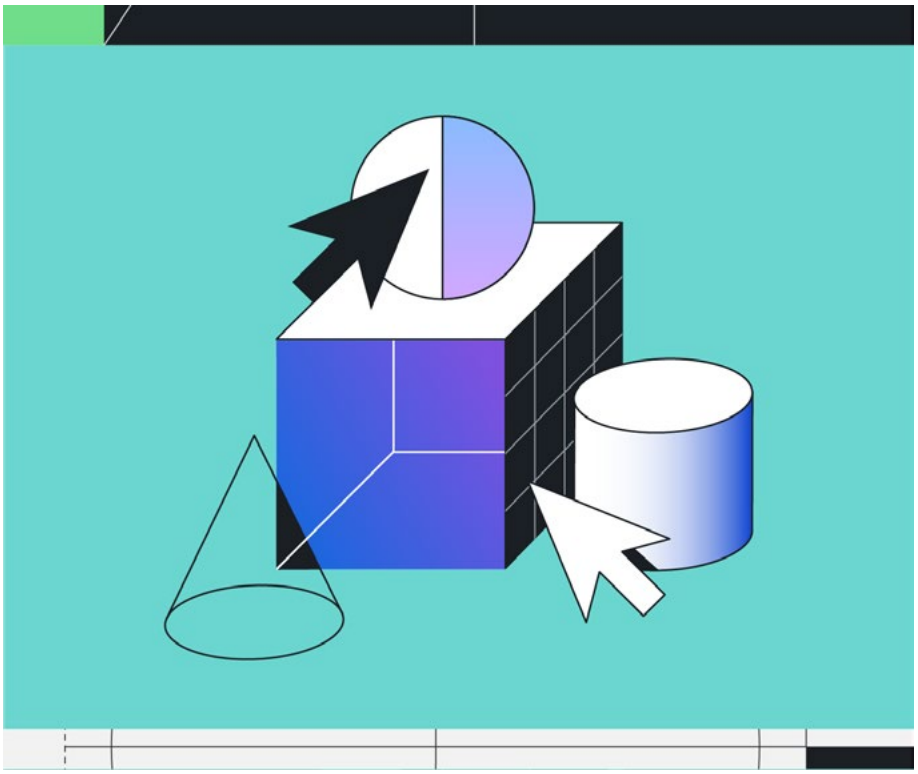


This is exactly what happened to the IT management software company, SolarWinds. Its CI/CD systems were targeted in an attack by nation-state actors, who added a malicious piece of code to a core piece of the company's supply chain software. This was then integrated into the codebase via the company's automated build process. From there, it was distributed to customers for a long period of time before being detected. That hack affected 18,000-plus SolarWinds customers, including key government organizations.

Attacks like this are fueling an increased interest in CI/CD governance and security. The United Kingdom's (UK) National Cyber Security Centre issued guidance for organizations to increase their security posture and the United States (US) has released an executive order to improve supply chain security.

Other potential risks in a CI/CD pipeline include:

- **Resource access management:** Organizations of all sizes need to be cognizant of granting the right access to the right people. Poor identity and access management can create opportunities for attackers to compromise the engineering ecosystem.
- **Dependency chain abuse:** Attackers will often target vulnerabilities in dependency chains across engineering and build environments to insert malicious packages. These supply chain attacks often target known vulnerabilities in first-party and third-party—including open source—dependencies.
- **Poor credential and secret management:** Most CI/CD environments leverage a number of systems, which means automated workflows typically contain access and authentication credentials such as passwords. These are referred to as secrets and are used across CI/CD pipelines. If secrets aren't encrypted or if organizations have overly permissive governance standards for granting access to unencrypted secrets, they can become an effective attack surface for malicious actors.
- **Design and architectural flaws:** CI/CD environments typically include a number of different tools, technologies, and processes—and designing the overall system architectures without addressing security in the pre-code activities, can introduce risk and security flaws. Design and architectural flaws occur when organizations build their overall CI/CD environment without effectively embracing "secure-by-design" principles to protect against potential exposure to security threats. These are distinct from implementation flaws, which occur in the setup and configuration of systems.
- **Ungoverned use of third-party tools:** In a CI/CD environment, organizations will often connect third-party tools and services to their core version control system (VCS), software configuration management (SCM), and CI services. Without proper governance to keep access to these tools secure, malicious entry into one third-party tool could allow an attacker to access the wider technology stack and source code.
- **System configuration security issues:** This refers to practices that could potentially expose systems within a CI/CD environment to security risks, like weak passwords, failing to apply security patches in a timely manner, or failing to properly secure different systems from unauthorized access within a CI/CD environment.
- **Insufficient artifact integrity validation:** The reliance on multiple sources and contributors during the CI/CD process can introduce a number of attack surfaces where code, automated workflows, and third-party packages can be tampered with and infiltrated. If a piece of malicious code makes it into the CI/CD pipeline at any point without detection, it can end up being deployed to end users.
- **Pipeline access management:** Access controls in a CI/CD process center around setting governance standards for what automated workflows can be used, where they can be used, and who can use them; and setting environment-protection rules to determine who can affect changes in which environment (i.e. build, staging, and production environments). Without these types of access controls, organizations open themselves up to additional risks where a non-approved workflow could be used and compromised, or an attacker could access key environments within the CI/CD systems.



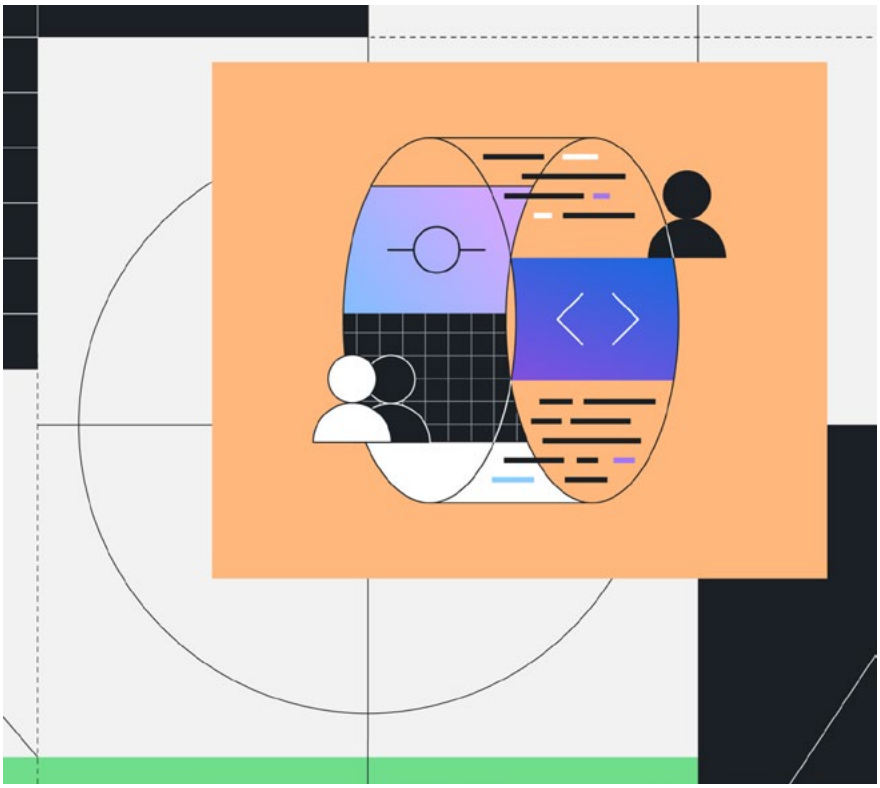
These risks help illustrate the complexity of CI/CD processes and the number of potential attack surfaces they contain.

At its core, a CI/CD governance model maps these risk areas and introduces a series of policies, standards, and procedures to ensure processes are followed and the quality of code—and integrity of the systems—is up to standards. And this work will impact the business as a whole. With better governance, not only are your security risks reduced, but your pipeline will be optimized and your developers will be able to stay in their flow—addressing your business needs faster (and more affordably).

Balancing CI/CD governance with speed to market

Balancing CI/CD governance with speed can be a challenge, as overly lax governance can lead to subpar code quality and security vulnerabilities, while overly strict governance can slow down the deployment process and hinder innovation.

To be efficient and secure, it is important to establish clear policies and procedures, and ensure that they are consistently followed. Strategies should include code reviews, automated testing, and deployment approvals, so code changes pass quality gates and do not introduce security vulnerabilities. It is also important to regularly review and update governance standards to ensure that they support the organization's goals.



At GitHub, we see two common routes in CI/CD governance:

- **Complete standardization.** Some organizations will elect to standardize the entire CI/CD process—and that includes fully governing every automated test, workflow, quality gate, third-party service or dependency, and so on. This typically is done by either a central governing body or an outside provider. This strategy takes a lot of planning, but can help organizations move faster. The downside of this model, however, is a lack of flexibility. Your teams may have edge cases that you may not account for while planning and developing your CI/CD processes and procedures.

- **Flexibility with strict controls.** Other organizations will decide to grant limited access to a small number of people who can augment CI/CD processes and the pipeline as needed. This typically involves strict controls with a multi-tiered approval board that evaluates and tests any new workflows, processes, or third-party services. While this leads to more flexibility, it inhibits speed.

In our experience, there's no one-size-fits-all approach to effective CI/CD governance. In fact, plenty of organizations will often opt for a compromise approach where they balance rigidity with flexibility.

Pro tip: An area of importance in CI/CD governance is immutability, a key DevOps best practice and GitOps principle that prioritizes creating and deploying software and infrastructure that cannot be modified once it has been created. This is typically accomplished via "immutable artifacts," which can be metadata, log files, or software packages and configurations. Critically, these artifacts are all stored in a version control system that can be easily referenced, replicated, and deployed across environments.

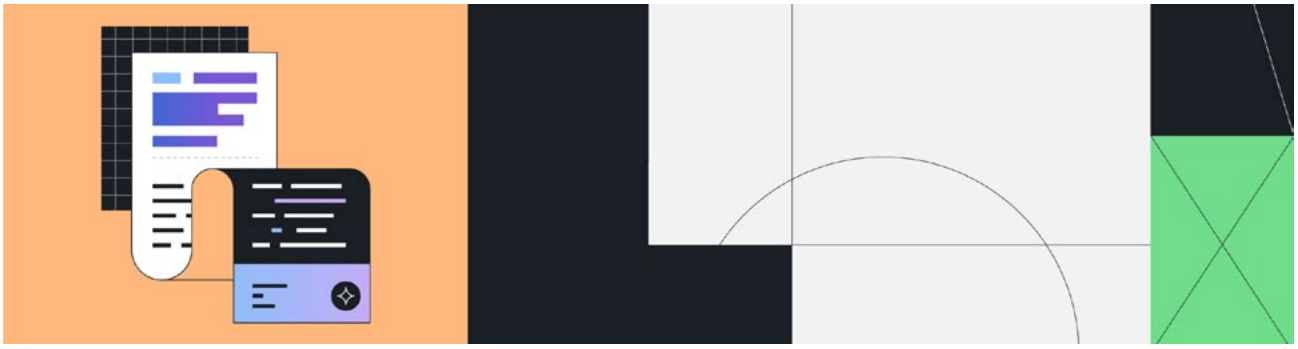
From a governance perspective, being able to "show" immutable artifacts that describe how you've addressed required policies and procedures can be critical. These artifacts enable organizations to avoid having to manually show stakeholders that governance policies and processes have been followed and move faster.

Certain technologies will lend themselves better to immutability than others—and just because your tools may not support immutable artifacts, it doesn't mean that you're not correctly adhering to a CI/CD governance model.

Common pain points in developing a CI/CD governance model

Beyond the challenge of balancing stability and reliability with speed in CI/CD governance, there are several other common pain points enterprise companies can experience, including governing CI/CD processes and systems at scale.

Enterprise companies, after all, have large workforces, more complex organizational structures, and sizable codebases with unique demands.



At GitHub, we typically see four recurring pain points among enterprise companies developing a CI/CD governance model. These include:

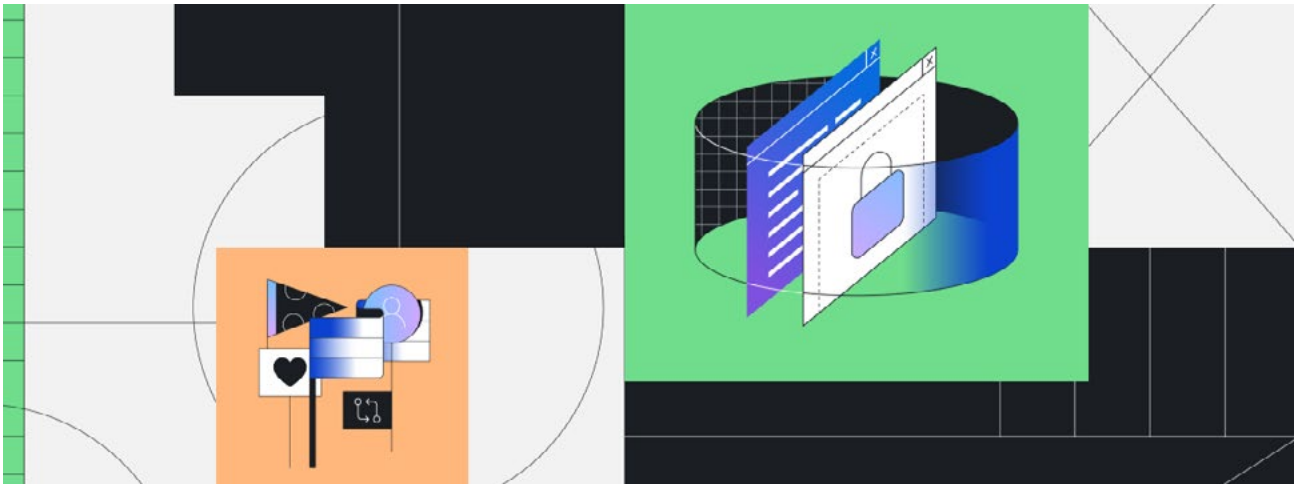
- **The proliferation of tooling.** The complexity of an organization's technology stack is often a key challenge when it comes to implementing CI/CD governance models. In most environments, development teams will leverage a number of different languages, frameworks, productivity tools, and core systems—and all of this tool sprawl increases the difficulty of applying consistent governance practices and processes.
- **User access control and permissioning.** One of the biggest challenges that organizations face is governing access and permissioning for users. Having a proper set of tooling that can automate a lot of that process so that the right users have the right at the right times is critical. Tools exist to help govern and manage user access control, but some don't have the capabilities required to control granular-level permissioning.
- **Systems access management.** CI/CD processes almost always involve more than one system—and keeping multiple systems securely connected to one another is a common pain point for organizations. Key strategies to ensure security include encrypting secrets, limiting who can configure secrets, using PATs (personal access tokens) as an alternative to passwords, and ephemeral, single-use tokens. Additional tooling exists that rotates secrets and automatically updates credentials.
- **Traceability.** In highly regulated industries, traceability and auditability are often core requirements. But regardless of your regulatory status, traceability is a key practice. The goal is to know what's in your software and what it's built upon, and be able to identify whether or not the things you expect to be in production are actually in production. This becomes even more important in the event of a security incident.

Creating an end-to-end CI/CD governance model: 11 enterprise strategies

No two CI/CD governance models look exactly the same. That's because each model reflects the particular needs, regulations, compliance standards, and industry requirements of the organization it serves.

However, there are strategies that enterprise organizations can pursue to develop and maintain an effective, end-to-end CI/CD governance model.

At GitHub, we often advise organizations to consider the following 11 strategies when planning or auditing their CI/CD governance standards.



1. **Map your CI/CD processes and systems from beginning to end.**

   Mapping your CI/CD processes and systems provides a clear overview of your entire CI/CD pipeline and the stages at which you are most vulnerable to security threats. It can also reveal opportunities to improve your processes, systems, and security posture.

   The goal is to create a value stream map of your CI/CD processes, systems, and tooling to understand the handoff points, and connect them to enterprise key controls, regulatory requirements, and industry controls. This allows you to both optimize existing processes, build a governance model, and position your organization to conduct audits.

2. **Describe your CI/CD pipeline declaratively as code.**

   Often referred to as "pipeline as code," defining your CI/CD pipeline through code starts with using a declarative approach using YAML configuration files you can leverage through a version control system. The benefit of being able to describe your CI/CD pipeline as code is that you can build in controls, gates, and processes (ie, governance practices and processes) and then easily apply them across your environments. It also creates an audit trail that organizations can use to ensure governance standards are being met.

3. **Define clear roles and responsibilities for each team involved in the CI/CD process.**

   The best way to think about building a CI/CD governance model is to look at the stages of your CI/CD pipeline and determine the role and responsibilities of each team and person that engages with it. (For instance, your developers will almost never be the ones building and maintaining a CI/CD pipeline, since they're responsible for developing code.)

   This will help you decide what level of access you need to grant each team to your underlying systems, and what processes you need to implement to ensure proper governance.

4. **Control and regularly audit access and permissioning for your teams and systems.**

   Managing access and permissioning effectively is difficult, but critical.

   We recommend developing a golden master for all access and permissioning through an identity provider (IDP) like Azure Active Directory.

   Regardless of what platform you use, you should define your principle sources and make sure every other system is subservient to it. This helps stop the inevitable entropy.

5. **Give teams flexibility—but not too much.**

   At GitHub, we know that people will often set up their own tools, scripts, and automations no matter how many safeguards you put in place to prevent that from happening. Part of effective CI/CD governance revolves around implementing guardrails to prevent people from doing that, or making it so they can set up their own tools and instances in a visible and sanctioned way.

   The best way to do this is to give people the flexibility they need to do their jobs, and regularly audit processes and to see where additional flexibility is needed—or, inversely, where more rigidity is necessary.

6. **Invest time in planning your automated testing requirements.**

   Effective testing suites are a critical part of effective CI/CD governance models—and especially automated tests that help teams "shift left," or prioritize security and functionality as early as possible within the SDLC.

   At the beginning of the software development lifecycle (SDLC), we recommend first prioritizing fast, efficient tests that are quick to turn around or at least execute within timeframes where developers are not going off to do another task. That way if it fails, your developers will know and won't accidentally ignore it. The further through your SDLC you get, the more complex your testing requirements should become.

   It is also important to have traceability. If you know the point of failure, you'll be able to diagnose a problem and solve it faster.

7. **Standardize all practices and protocols for code review and testing.**

   Effective CI/CD governance means one person can never write code, initiate a build, and deploy that code without any additional checks. At GitHub, we recommend the "four-eyes" approach when planning your policies. This means any change should be signed off on by at least two people. Critically, you don't always need to have a second person sign off on all changes—an automated test may provide you with sufficient confidence to move forward.

   Whatever path you take, you should set and define policies at the organizational level to ensure your teams are following a standardized set of practices and protocols. The goal is to help your developers more effectively identify and fix problems with the code, prevent defects from being introduced, and ensure that the final product meets the necessary requirements. This in turn can help to improve the overall efficiency and effectiveness of the development process, and help teams to deliver high-quality software to their users more quickly.

8. **Set deployment environment rules to ensure successful ships.**

   Typically, the deployment stage in a CI/CD process occurs in a separate system from the core code, build, test, and release stages. Oftentimes, you'll potentially be pulling from multiple sources to orchestrate a deployment. You might, for instance, be using version control to leverage infrastructure as code files.

   A best practice is that you should have environment consistency and traceability across the build, stage, and deployment parts of your SDLC. The worst thing you can do is add conditionality in one environment and not your other environments. This makes it a quick path to testing software in production, since your production environment may have certain conditions that aren't present in your staging environment. It's also important to set deployment environment rules so that when something makes it to production, it has been fully vetted for security, quality, and overall functionality.

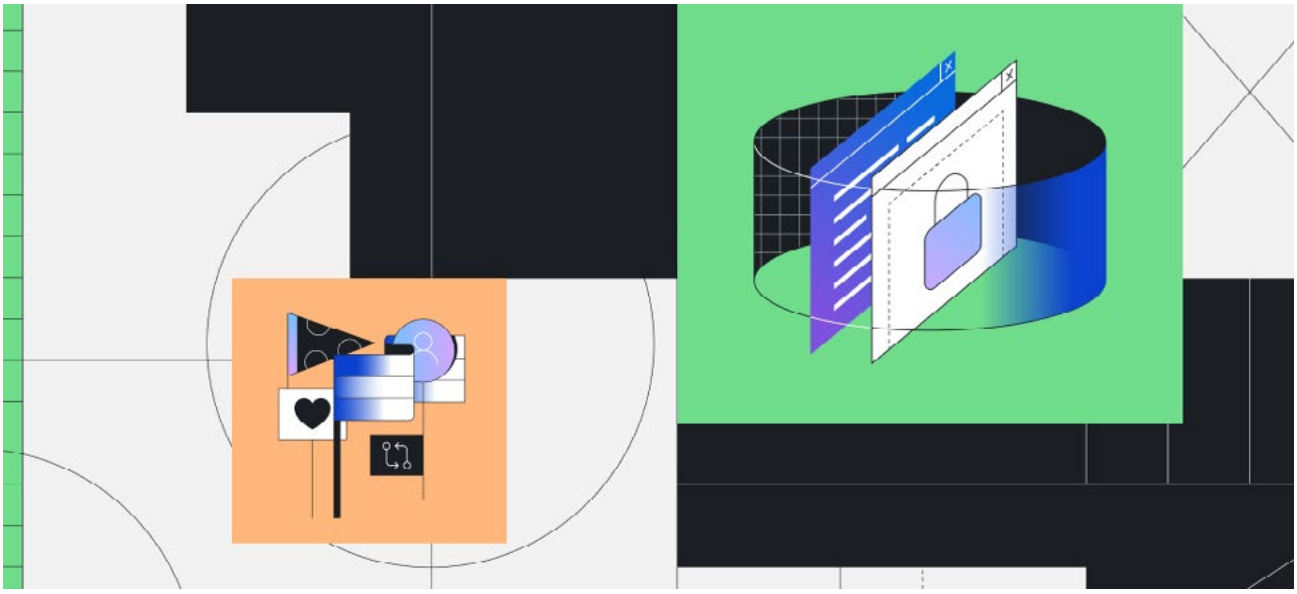
   Another strategy organizations should consider is treating environments as "parameters" in their CI/CD pipelines via declarative code in a YAML configuration file. By parameterizing your environments as code, you can better ensure that all required conditions are met consistently all the way to production.

   A sizable number of enterprise organizations also elect to use OpenID Connect (OIDC) at the deployment stage for additional security. OIDC uses ephemeral tokens to authenticate each deployment, which are automatically rotated after each usage. This helps improve overall security and reduce attack surfaces for malicious actors.

9. **Implement robust security measures to protect against unauthorized access to the CI/CD pipeline and the systems it interacts with.**

   CI/CD pipelines can involve multiple systems (version control, a CI tool, security tools, etc.) and making sure all those systems are integrated is critical to prevent any unauthorized access to your systems and code. Here are some security measures we recommend:

   - **Always practice least privilege.** Users, tools, and services should only have the minimum level of access and permissions required to do their jobs. This ensures that individuals, teams, and systems don't have unauthorized access to sensitive data and resources.
   - **Encrypt sensitive data.** One best practice is leveraging ephemeral, single-use tokens (found in systems like GitHub Actions and OIDC) which are automatically rotated after each job.
   - **Automate dependency testing for security vulnerabilities.** From your CI/CD workflows to your code base to your underlying systems such as containers, you should always be conducting regular security tests on all of your dependencies with a static application security testing (SAST) tool. Tools such as Dependabot can be incorporated into a CI/CD pipeline to automatically test your codebase for any known vulnerabilities while sending automatic alerts to your teams if anything is discovered.
   - **Conduct regular security audits.** The goal of these audits is to document any findings, including any issues or weaknesses, and produce recommendations for improving your overall security posture.



10. **Monitor the performance of your CI/CD pipeline.**

    Common indicators of CI/CD performance include deployment frequency, lead time for changes, and failure rates. Measuring these metrics can help organizations identify potential bottlenecks or inefficiencies in their wider CI/CD pipeline. They can also be used to track how governance policies and procedures are impacting the delivery of new code to end users.

    One way to monitor CI/CD pipeline performance is via dedicated monitoring tools, which provide visibility into overall performance and can help identify trends over a period of time. But tools alone aren't enough. It's also important to establish key metrics to track the performance of your pipeline and governance policies and procedures (e.g. the time it takes to deploy a new build or the reliability of deployments). These metrics offer observable signals that, at a high level, provide a more holistic view of how a pipeline and system are performing.

11. **Regularly review and update CI/CD governance policies and procedures.** Make sure they remain aligned to your organization's goals and demands.

    Here's a sample workflow for a review:

    - Create a review panel with cross-functional members from your development, security, operations, and IT teams.
    - Examine existing policies and procedures for any gaps or improvement areas.
    - Convene your review panel to offer feedback on existing policies and procedures from their individual areas of expertise.
    - Update existing policies and procedures to reflect the feedback from your review panel.
    - Review your updates to ensure they align with your organizational needs.
    - Implement them with clear and comprehensive communications about them to impacted teams.
    - Monitor and measure the efficacy of your new policies, and make any needed adjustments.

    CI/CD governance and the priorities that matter most to your business will change over time, so regular reviews should be a key business function to support the success of your organization.

Take this with you

The goal of CI/CD governance is to establish policies and procedures to help ensure that a CI/CD pipeline is efficient, secure, and compliant with industry standards and regulations—and effective CI/CD governance ensures that all shipped code is high quality, secure, and traceable.

At the enterprise level, organizations often struggle to implement the right tooling, processes, and procedures to effectively govern their CI/CD workflows—and that's especially true when they're first starting to develop a governance model. This comes down to scale: there are a lot more people, tools, systems, and users at the enterprise level, making effective governance more complex.

But when your teams are following your processes, and your CI/CD practices are governed appropriately, you can be confident that everything you're releasing has satisfied all of your requirements and quality gates, and has the necessary markers to track back if needed in the future. This makes the upfront investment all the more important.

GitHub offers the complete platform to deliver secure software at scale. Trusted by more than 94 million developers and 90% of the Fortune 100, GitHub helps DevOps teams of every size collaborate securely and deliver better customer experiences, faster.

To start your free trial or learn more about GitHub's DevOps, CI/CD, and security solutions, visit https://github.com/enterprise.

WRITTEN BY GITHUB WITH ❤