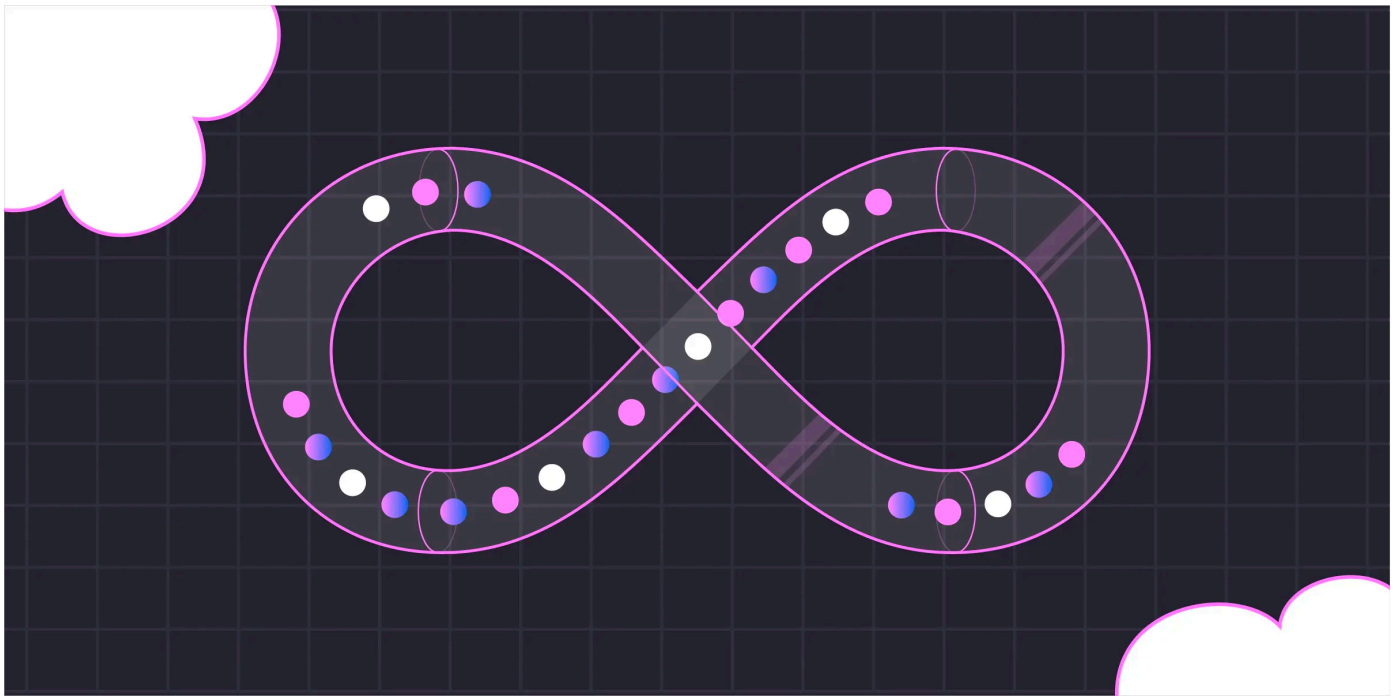
CI/CD Pipeline Security Best Practices

Continuous integration and continuous delivery (CI/CD) have become the backbone of modern software development, enabling rapid, reliable, and consistent delivery of software products. To bolster your CI/CD pipeline, ensuring resilience against ever-evolving threats, follow the best practices in this guide.

 Nicolas Ehrman November 1, 2023 9 minutes read



CI/CD security: A refresher

Continuous integration and continuous delivery (CI/CD) have become the backbone of modern software development, enabling rapid, reliable, and consistent delivery of software products. And with the increasing reliance on CI/CD pipelines, securing them has become non-negotiable. A compromised pipeline can lead to the deployment of vulnerable applications, causing significant damage to organizations and their users.

The refresher section below provides a solid foundation for understanding CI/CD, with overviews of the key components at risk, the shared responsibility model, and the challenges faced in CI/CD security.

Components at risk

In a CI/CD pipeline, several components are susceptible to attacks, including:

- **Source-code repositories:** Where the application code is stored and managed
- **Build servers and workers:** Where the code is compiled into executable artifacts
- **Artifact repositories:** Where the build artifacts are stored for deployment
- **Deployment environments:** Where the application is deployed and runs

Each of these components requires specific security measures to prevent unauthorized access and tampering.

The shared responsibility model

The **shared responsibility model** is a foundational security framework in cloud computing. It delineates the security responsibilities between cloud service providers and the users of those services. In the context of CI/CD, this model says that while providers are tasked with ensuring the security of the underlying infrastructure and services, the onus is on the users (particularly the development teams) to secure their code, configurations, and data. This division of responsibility ensures that both parties play their part in safeguarding the entire ecosystem. Adhering to this model is crucial for maintaining the integrity and security of the CI/CD pipeline.

Challenges in CI/CD security

Securing CI/CD pipelines has formidable challenges, including:

- **Complexity:** The diverse and interconnected components of CI/CD pipelines make them complex to secure.
- **Speed:** The rapid pace of CI/CD can sometimes outstrip security controls, leading to vulnerabilities.
- **Automation:** While automation is a strength of CI/CD, it can also be a weakness if security checks are not automated as well.
- **Access control:** Managing access to various pipeline components can prove difficult, especially when granting access to large teams.
- **Secrets sprawl:** It's not uncommon for CI/CD solutions to house a plethora of secrets. Combined with subpar access controls, this can often pave the way for secrets leakages.
- **Supply-chain security with SaaS CI/CD tools:** Leveraging software-as-a-service (SaaS) CI/CD tools like GitLab and CircleCI can introduce supply-chain security concerns. Dependence on third-party tools and services might expose organizations to risks if these tools are compromised or suffer from vulnerabilities.

Best practices and recommendations

A report from Cybersecurity Ventures predicts that the cost of cybercrime will soar to $10.5 trillion annually by 2025. This alarming statistic underscores the urgent need for secured CI/CD pipelines. As we delve deeper CI/CD pipelines, we must bear this statistic in mind, understanding its broader implications.

To bolster your CI/CD pipeline, ensuring resilience against ever-evolving threats, follow these best practices:

Automate security scans

Given the complexity of modern software, relying solely on manual security checks is impractical. Automated security scans are indispensable for real-time vulnerability detection, especially with the advent of DevSecOps, which integrates security into the development lifecycle. Automated security scans facilitate continuous security assessments, immediate vulnerability detection, and developer notifications, mitigating the risk of deploying compromised code.

Action items

- **Integrate security tools:** Embed tools like SonarQube or Checkmarx into your CI/CD pipeline. These tools can perform static and dynamic analysis on the application code, identifying vulnerabilities like SQL injection, cross-site scripting (XSS), and insecure object references.

```
# Example of SonarQube integration in a Jenkins pipeline
pipeline {
    agent any
    stages {
        stage('SonarQube Analysis') {
            steps {
                script {
                    def scannerHome = tool 'SonarQube Scanner';
                    withSonarQubeEnv('My SonarQube Server') {
                        sh "${scannerHome}/bin/sonar-scanner"
                    }
                }
            }
        }
    }
}
```

- **Schedule regular scans:** Configure your CI/CD pipeline to trigger security scans after every code commit, utilizing webhook integrations or polling SCM, ensuring real-time vulnerability assessments.
- **Notify developers:** To facilitate immediate remediation, leverage notification channels such as email, Slack, or Microsoft Teams to alert developers about possible vulnerabilities instantly.

Manage secrets effectively

Exposing secrets through hardcoding or improper management can lead to severe security breaches, including unauthorized data access and system compromises. Effective secrets management such as API keys, passwords, and tokens prevents unauthorized access and enhances overall application security.

Action items

- **Use secrets management tools:** Tools like HashiCorp Vault or AWS Secrets Manager provide features including dynamic secrets, secret revocation, and detailed audit logs. Implementing these tools effectively can significantly enhance the security posture of an organization by keeping sensitive data protected and access controlled.

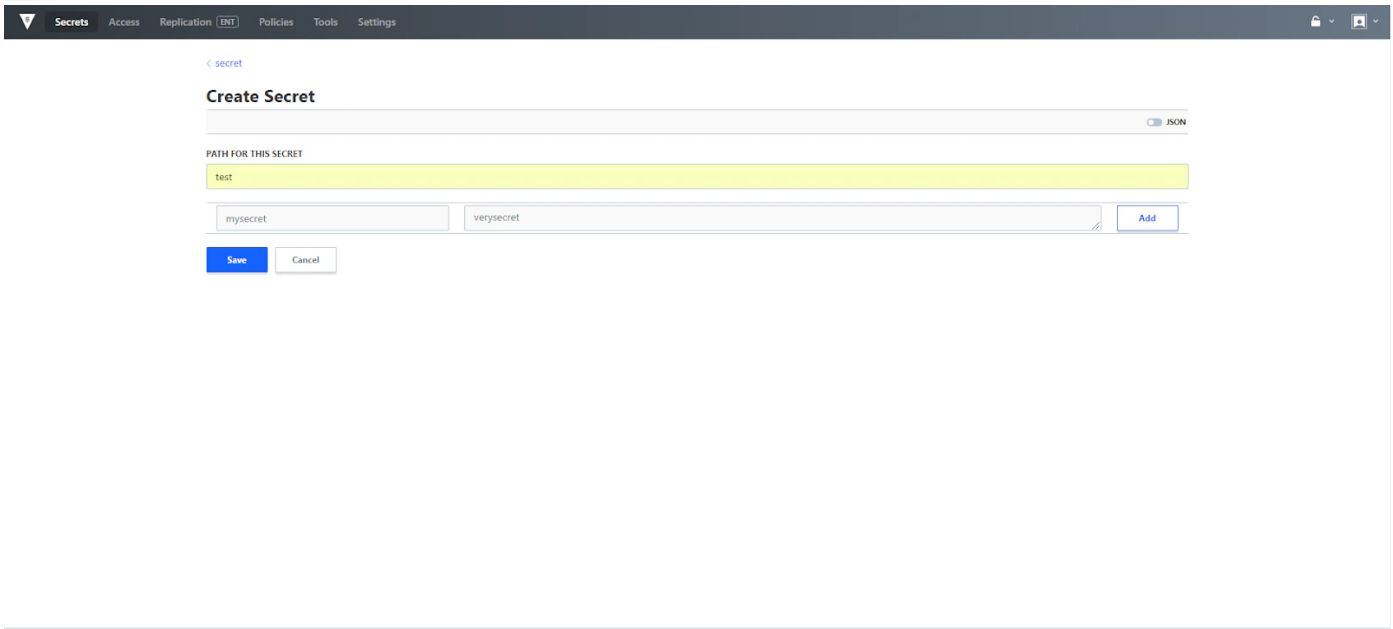


Figure 1: Vault dashboard (Source: Vault)

- **Regularly rotate secrets and keys:** By implementing automated rotation policies for secrets and keys using APIs and SDKs provided by secrets management tools, you can reduce the window of opportunity for malicious actors.
- **Avoid hardcoding secrets:** Use pre-commit hooks and secret-scanning tools to enforce coding standards and perform code reviews to prevent hardcoding of secrets in the codebase.

Immutable infrastructure

Mutable infrastructures are susceptible to inconsistencies and vulnerabilities due to post-deployment alterations. Immutable infrastructure, on the other hand, promotes consistency, reduces attack vectors, and simplifies rollback mechanisms.

Action items

- **Use containerization tools:** Utilize tools such as Docker for containerization and platforms like Kubernetes for orchestration to achieve uniform and unchangeable deployments across different environments.

```
# Example Dockerfile for a Node.js application
FROM node:14
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD ["node", "app.js"]
```

- **Implement infrastructure as code (IaC):** To ensure environmental consistency and immutability, leverage IaC tools like Terraform or AWS CloudFormation to define, version, and provision

infrastructure.

```
# Example Terraform script to create an AWS S3 bucket with versioning enabled
resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-tf-test-bucket"
  acl    = "private"
  versioning {
    enabled = true
  }
}
```

- **Ensure rollbacks re-create the entire environment:** Design rollback strategies to re-create the entire environment from IaC definitions in order to return your environment to a known, secure state.

Regularly update and patch

Utilizing outdated software components exposes systems to known vulnerabilities, which attackers can exploit to compromise CI/CD pipelines and deployed applications. Regular updates and patches secure the CI/CD pipeline against known vulnerabilities by installing the latest features and security enhancements.

Action items

- **Set up automated update checks:** Subscribe to vulnerability databases like NVD, and implement automated update checks for all components, taking advantage of package managers like npm, pip, and apt-get.

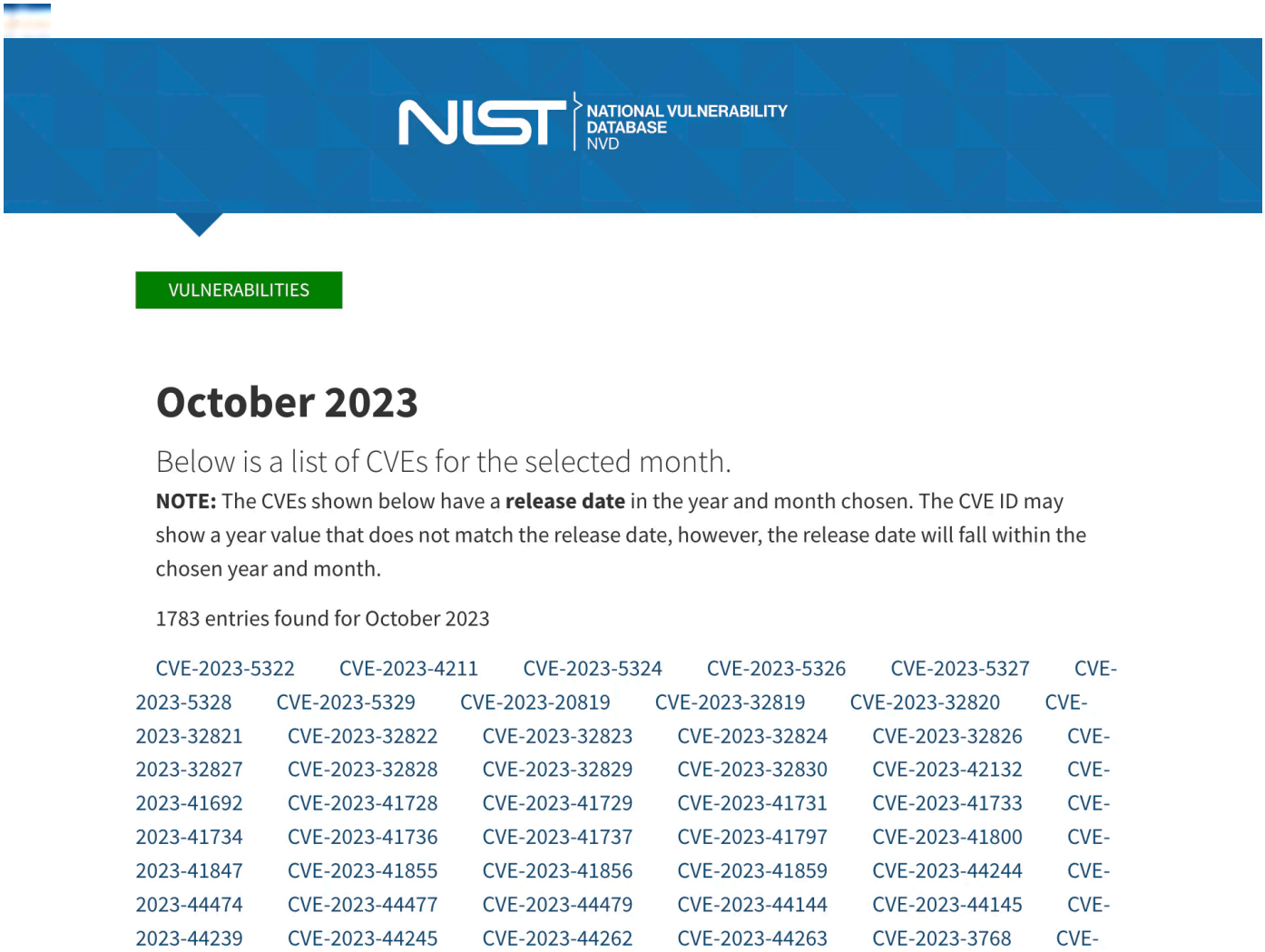


Figure 2: National Vulnerability Database

- **Implement a patch-management strategy:** A comprehensive patch-management strategy incorporates risk assessment, testing, and phased deployment to prevent introducing new issues.
- **Regularly review and apply security advisories:** Subscribe to security advisories from software vendors and open-source communities, and promptly apply recommended patches and updates.

Role-based access control (RBAC)

Implementing RBAC is an essential means of assigning access based on user roles, which minimizes the risk of accidental or malicious changes and limits the potential damage from compromised accounts. RBAC implementation secures the CI/CD pipeline by granting users the minimum necessary access to perform their duties, reducing the chances of unauthorized access and modifications.

Action items

- **Define clear roles and responsibilities:** Define and document roles and responsibilities within your organization, assigning permissions based on the principle of least privilege.
- **Implement RBAC in CI/CD tools and environments:** Configure RBAC settings in CI/CD tools like Jenkins, GitLab, and Kubernetes, leveraging built-in RBAC mechanisms and plugins.

```yaml
# Example of defining roles and permissions in Kubernetes RBAC
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: database
  name: pod-viewer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
kind: User
name: "db-admin"
apiGroup: rbac.authorization.k8s.io
roleRef:
kind: Role
name: pod-viewer
apiGroup: rbac.authorization.k8s.io
```

- **Regularly review and update roles and permissions:** Take advantage of auditing tools and logs to conduct regular audits and reviews of roles and permissions, checking to make sure access is granted on a need-to-know basis and that there aren't any discrepancies.

Monitor and alert

- Implementing comprehensive monitoring and alerting mechanisms enhances the visibility of the CI/CD pipeline's health and security, enabling immediate detection of and response to any anomalies or security incidents. Such proactive measures help teams swiftly address potential threats, minimizing risks and potential downtimes.

Action items

- **Integrate monitoring tools:** Incorporate advanced monitoring tools like Splunk or the ELK Stack into your CI/CD pipeline. These tools provide real-time insights, log analytics, and generate visualizations to help you identify suspicious activities.

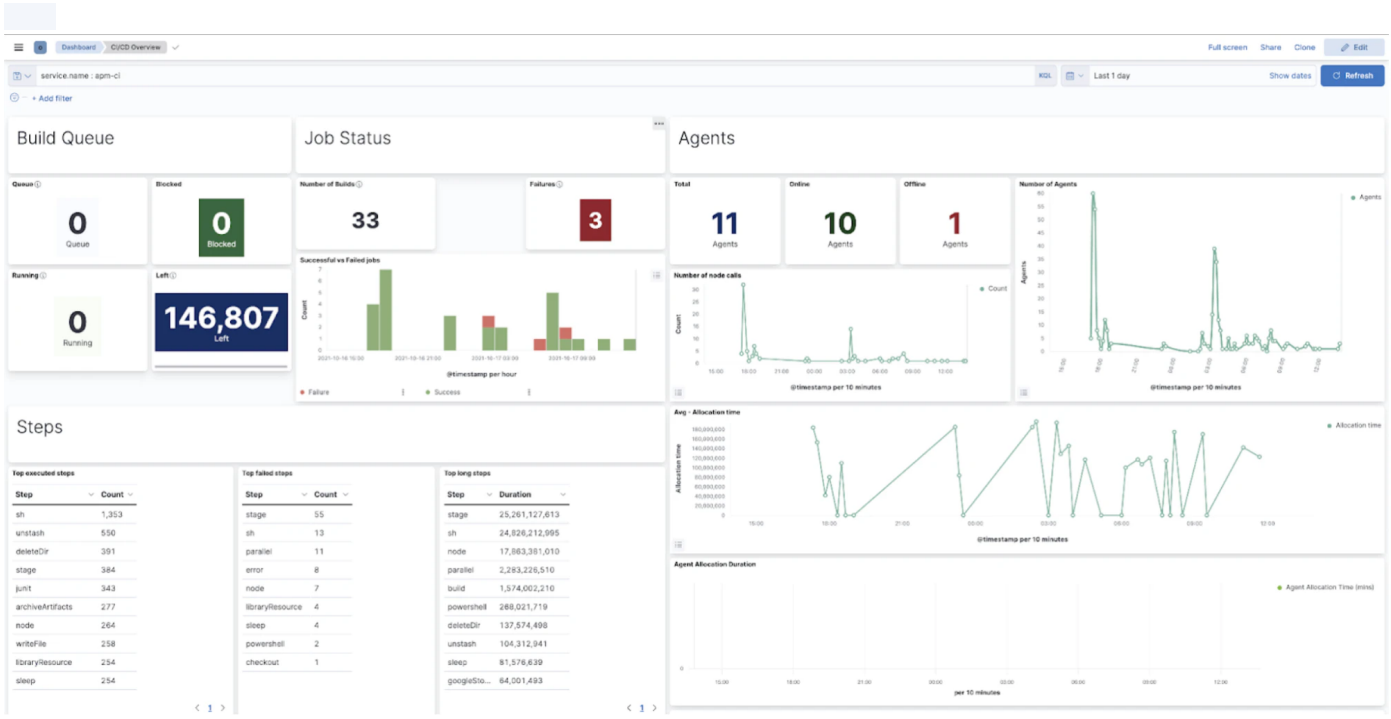


Figure 3: Jenkins KPIs in a Kibana dashboard (Source: ELK Guide)

- **Set up real-time alerts:** Configure your monitoring tools to send real-time alerts for suspicious activities or anomalies. Define alerting thresholds and double-check that the relevant team members are notified immediately.

```
# Alert manager rule to check the connection status of Jenkins instances. Offline
instances can be hijacked or used for malicious software.
- alert: JenkinsNodeOffline
    expr: jenkins_node_offline_value > 1
    for: 0m
    labels:
      severity: critical
    annotations:
      summary: Jenkins node offline (instance {{ $labels.instance }})
      description: "Jenkins node offline: `{{$labels.instance}}` in
{{$labels.realm}}/{{$labels.env}} ({{$labels.region}})\n  VALUE = {{ $value }}\n  LABELS =
{{ $labels }}"
```

- **Regularly review monitoring dashboards and update alerting thresholds:** Periodically review monitoring dashboards to ensure that they reflect the current state of your CI/CD pipeline. Adjust alerting thresholds as needed to avoid false positives and facilitate the timely detection of issues.

Going beyond the basics

CI/CD security is a necessity for organizations who want to build and deploy reliable and secure applications. The strategies and best practices outlined above lay a strong foundation for securing CI/CD pipelines. However, achieving a scalable and robust pipeline is a continuous process that requires you to go beyond the basics.

Immediate recommendations

- **Educate and train teams:** Regularly educate and train development and operations teams on security best practices and emerging threats.
- **Conduct security audits:** Regularly schedule security assessments for your CI/CD pipeline to detect and address potential vulnerabilities or configuration errors.
- **Stay informed:** Read up on the latest security trends, vulnerabilities, and patches to keep your CI/CD pipeline secure.

Wiz's approach to CI/CD Security

Wiz Code enhances CI/CD security by embedding security checks directly into the development pipeline, ensuring vulnerabilities are caught early and continuously throughout the software lifecycle.

- **Seamless CI/CD Integration**: Wiz Code integrates with popular CI/CD platforms like Jenkins, GitLab, and CircleCI, ensuring security scans run automatically at every stage of the CI/CD pipeline. This allows developers to identify and address issues, such as misconfigurations and vulnerabilities, before the code reaches production.
- **Shift-Left Approach**: Wiz Code promotes a shift-left security model by embedding security checks early in the development process. Developers receive instant feedback on security risks in their Infrastructure as Code (IaC) templates and dependencies, preventing the introduction of vulnerabilities before they become embedded in later stages of development.
- **Automated IaC Security Scans**: Wiz Code automatically scans IaC files like Terraform and CloudFormation within the CI/CD pipeline, detecting potential security risks such as misconfigurations or exposed secrets. These scans ensure that infrastructure remains secure as code changes are continuously made.
- **Continuous Monitoring and Feedback**: Wiz Code delivers real-time, continuous feedback on security risks, enabling teams to take action on vulnerabilities throughout the CI/CD process. This reduces the risk of shipping insecure code or configurations by addressing risks proactively.
- **Policy Enforcement**: It enforces predefined security policies within the CI/CD pipeline, ensuring that all code meets an organization's security standards before moving to production. This consistent policy enforcement minimizes the risk of human error and misconfigurations.

In this article, we've talked about various tools, each focusing on a different part of the wide-ranging landscape of DevOps security. Schedule a Wiz demo, and learn about our advanced features like container and Kubernetes security and IaC scanning on an easy-to-use, unified platform.