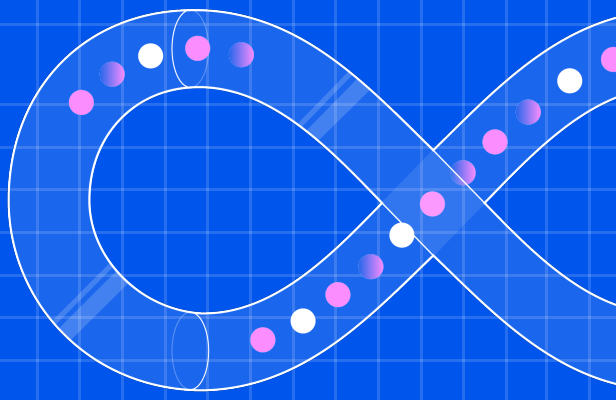
# CI/CD Pipeline Security Best Practices



Continuous integration and continuous deployment (CI/CD) have become essential processes for delivering high-quality software swiftly and efficiently. However, the dynamic nature of CI/CD pipelines also presents evolving security challenges that require specialized solutions to protect applications and infrastructure.

This comprehensive guide provides you with actionable best practices to mitigate CI/CD security risks. Read on to learn about infrastructure security, code security, access, and monitoring. In each section, you'll find technical background information, actionable items, code snippets, and screenshots, empowering you to take a holistic approach to fortifying your CI/CD pipelines.

Let's look at infrastructure security first.

## Infrastructure security

### 1 Best practice: Use container security tools to scan for vulnerabilities

Containers are integral to modern software deployment pipelines thanks to their stateless and portable nature, but they can introduce vulnerabilities if they're not secured. Container security tools scan container images against vulnerability databases and assess compliance with security policies. These tools identify vulnerabilities like outdated libraries, exposed sensitive data, and misconfigurations, providing detailed reports for remediation. Integrating container security tools keeps containers free from known vulnerabilities, leading to a more secure and reliable application environment.

## Actionable items

- **Integrate container security tools:**
  Embed tools like Clair or Anchore into your CI/CD pipeline to automatically scan container images for vulnerabilities and compliance with security policies:

```
# Jenkins pipeline with Clair scanning stage
pipeline {
agent any
stages {
stage('Clair Scan') {
steps {
sh 'docker-compose -f docker-compose.clair.yml up -d'
sh 'clair-scanner --ip YOUR_IP my-image:latest'
}
}
}
}
```

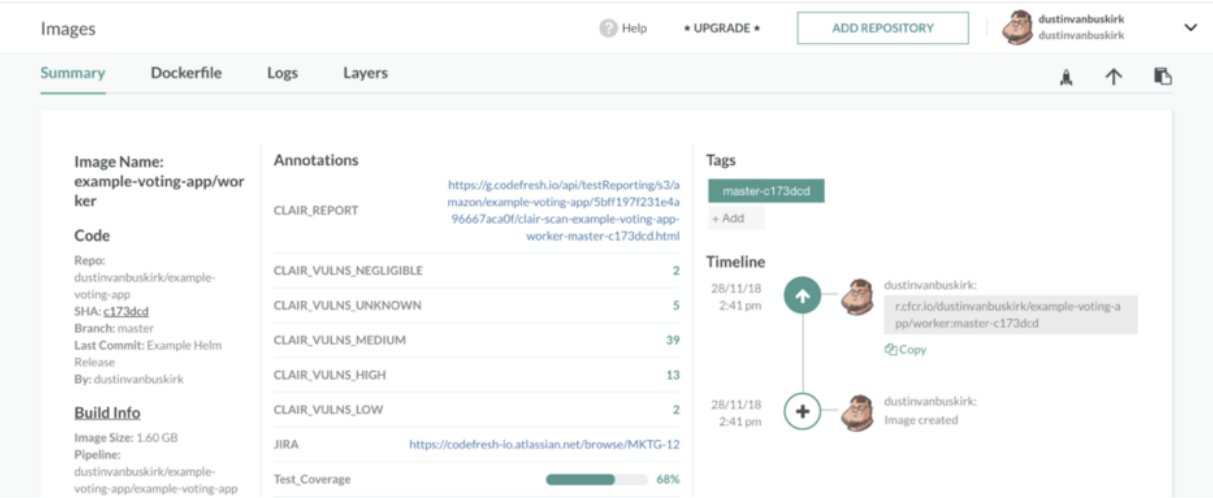



Figure 1: Clair vulnerability report in a Codefresh pipeline (Source: Codefresh blog)

- **Regularly update container images and base operating systems:**
  Instantly incorporate the latest security patches (and minimize vulnerabilities) by relying on automation to update container images and their base operating systems.

- **Avoid using containers with root privileges:**

  Configure containers to run with the least privileges necessary and avoid the use of root privileges.

- **Use immutable, on-demand workers:**

  Instead of maintaining long-lived workers that require regular updates and management, opt for immutable, on-demand workers. These workers are spun up for a specific task and discarded after use, ensuring they are always based on the latest secure image without accumulating vulnerabilities over time.

## 2 Best practice: Implement network segmentation in your CI/CD environment

A flat network can expose all assets if just one asset is compromised. Network segmentation combats this issue by dividing your network into segments, allowing you to assign unique communication policies and access lists for each. To put it simply, network segmentation lets you isolate assets, control traffic, and limit the reach of potential attackers.

In CI/CD, it's pivotal to understand the unique roles and potential vulnerabilities of different assets like the build server and worker nodes. By discerning the unique roles and security considerations of each component in your CI/CD setup, you can better tailor your network segmentation strategies and enhance overall security:

- **Build server:** The build server is central to the CI/CD pipeline. It integrates and compiles code, runs initial tests, and acts as the orchestrator of various CI/CD processes. It is crucial that the build server is not exposed to the internet. Doing so can make it a prime target for cyberattacks. A compromised build server can allow malicious actors to introduce malevolent code, alter build processes, or access proprietary information.
- **Worker node:** These are the entities that execute tasks as directed by the build server or the control plane. Worker nodes manage workloads and run applications. Their communication should be limited, especially with the control plane and other services. This way, even if a worker node is compromised, it cannot corrupt other components of the CI/CD pipeline or access restricted data.

### Actionable items

- **Use virtual private clouds (VPCs) and subnets:**

  Implement VPCs and subnets to isolate different parts of your CI/CD environment, restricting communication between segments to only what is necessary.

```
# AWS CLI command to create a new VPC

aws ec2 create-vpc --cidr-block 10.0.0.0/16
```

- **Implement firewalls and access control lists (ACLs):**

  Configure firewalls and ACLs to control traffic between network segments, allowing only authorized communication.

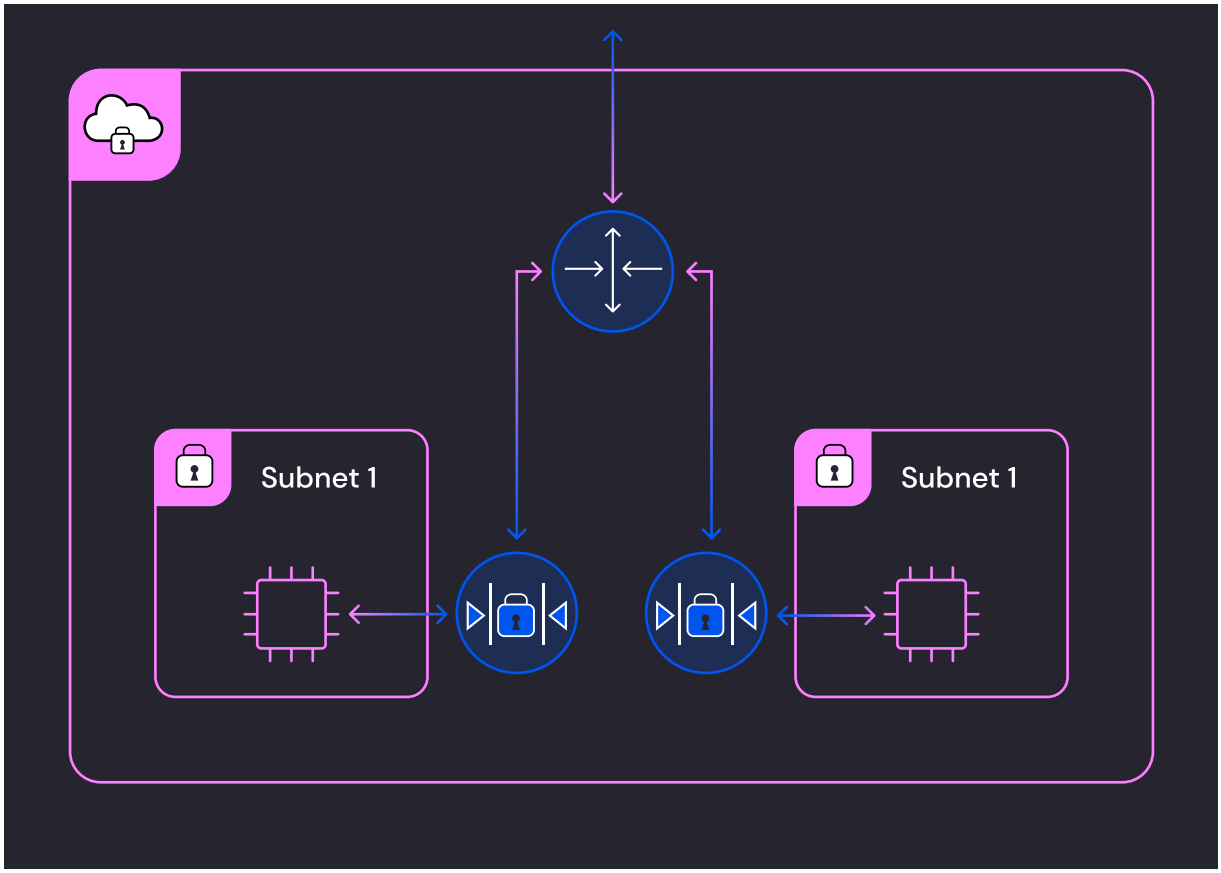



Figure 2: VPC with two subnets and ACLs (Source: AWS Docs)

- **Regularly audit network configurations:**
  Use network configuration and compliance assessment tools to conduct regular audits of network configurations, ensuring that segmentation rules are up to date and effective.

# Code security

## 1 Best practice: Integrate static application security testing (SAST) tools

SAST tools perform static analysis of the source code, bytecode, or binary code to identify vulnerabilities such as SQL injection, cross-site scripting (XSS), and buffer overflows. These tools integrate with CI/CD pipelines to provide immediate feedback to developers. Incorporating SAST tools into the CI/CD process allows developers to spot and tackle security flaws early on, making risk mitigation more cost effective and easier to handle.

## Actionable items

- **Embed SAST tools:**

  To automatically analyze code for vulnerabilities, integrate Veracode or Fortify into your CI/CD pipeline:

```
# Jenkins pipeline to integrate Fortify

pipeline {

agent any

stages {

stage('Fortify Scan') {

steps {

sh 'sourceanalyzer -b my_build_id -clean'

sh 'sourceanalyzer -b my_build_id ./path_to_source_code'

sh 'sourceanalyzer -b my_build_id -scan -f results.fpr'

}

}

}

}
```

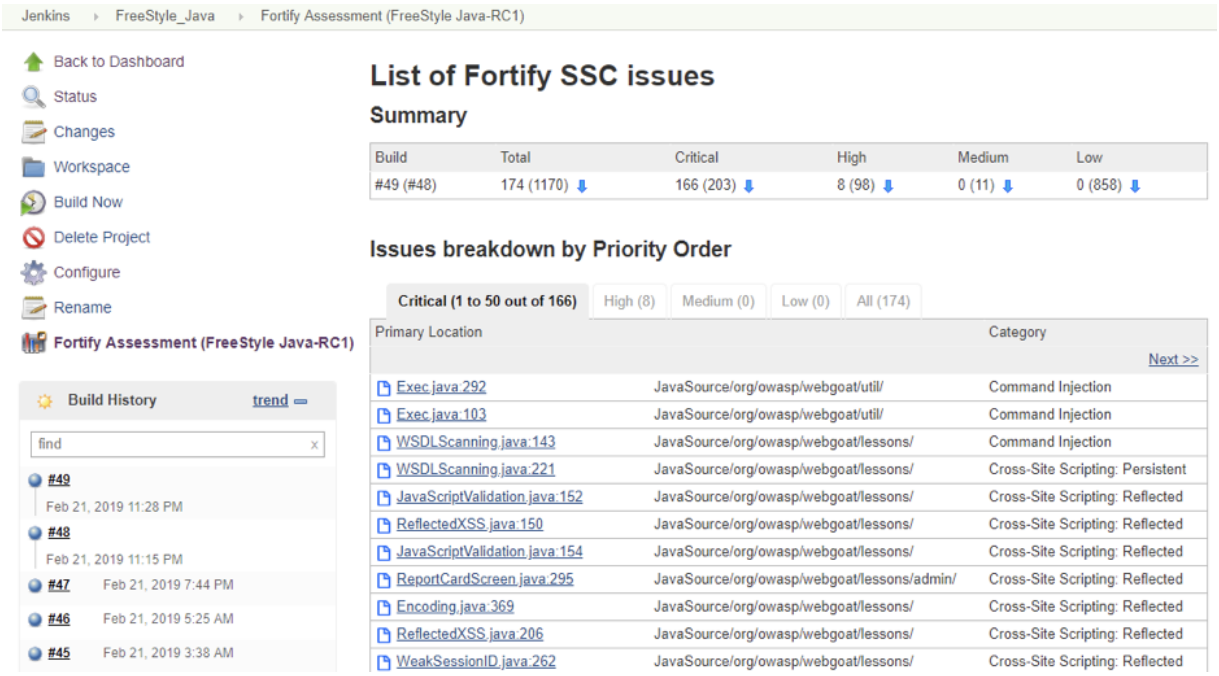



Figure 3: Fortify assessment in Jenkins (Source: Jenkins Plugins Index)

- **Train developers to interpret and act on SAST results:**

  Conduct regular training sessions for developers that cover how to interpret SAST results and remediate identified vulnerabilities effectively.

- **Incorporate SAST results into sprint reviews:**

  During sprint evaluations, analyze SAST outcomes to emphasize security priorities and guarantee that any identified vulnerabilities are tackled right away.

## 2 Best practice: Use dependency-checking tools to scan for vulnerable libraries

Third-party libraries can introduce vulnerabilities if they are outdated or poorly maintained. Dependency-checking tools, like OWASP Dependency-Check, scan project dependencies against known vulnerability databases. They identify outdated or vulnerable libraries and recommend updating to secure versions. Using dependency-checking tools ensures that the application is not exposed to known vulnerabilities present in its dependencies, enhancing the overall security of the application.

### Actionable items

- **Embed dependency-checking tools:**

  Take advantage of tools like OWASP Dependency-Check to automatically scan project dependencies against vulnerability databases:

```
# Example of integrating OWASP Dependency-Check with Maven

<!-- pom.xml -->

<plugin>

<groupId>org.owasp</groupId>

<artifactId>dependency-check-maven</artifactId>

<version>VERSION</version>

<executions>

<execution>

<goals>

<goal>check</goal>

</goals>

</execution>

</executions>

</plugin>
```

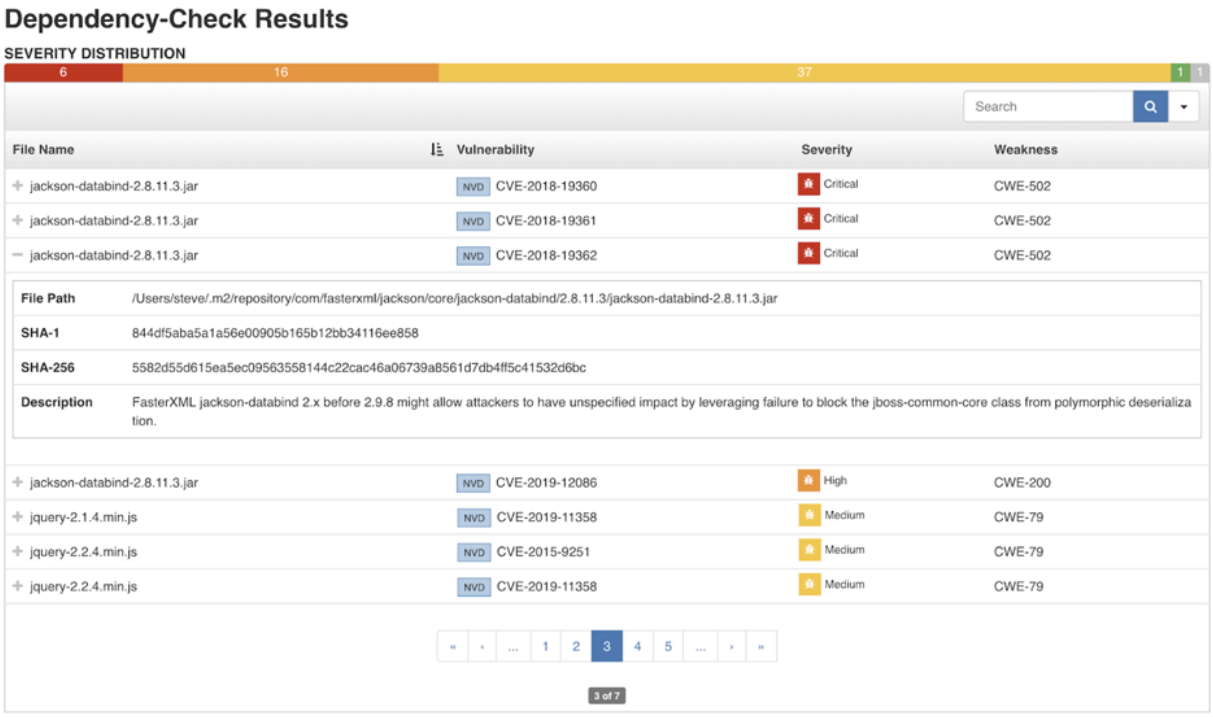



Figure 4: OWASP Dependency-Check results in Jenkins (Source: Jenkins Plugins Index)

- **Regularly update libraries to their latest secure versions:**

  Automate the process of updating libraries to their latest secure versions, utilizing package managers and dependency bots.

- **Maintain an inventory of approved libraries:**

  Keep a curated list of security-vetted libraries and make sure developers adhere to this list.

# Secrets management

## 1 Best practice: Ensure robust secrets management

Secrets, such as API keys, tokens, and passwords, are critical components that require the utmost protection. Storing secrets within CI/CD tools or as environment variables can introduce security risks.

### Actionable items

- **Use secure vaults to store secrets:**

  Always store secrets in specialized secret management tools, such as Vault, which are designed to encrypt and securely manage them.

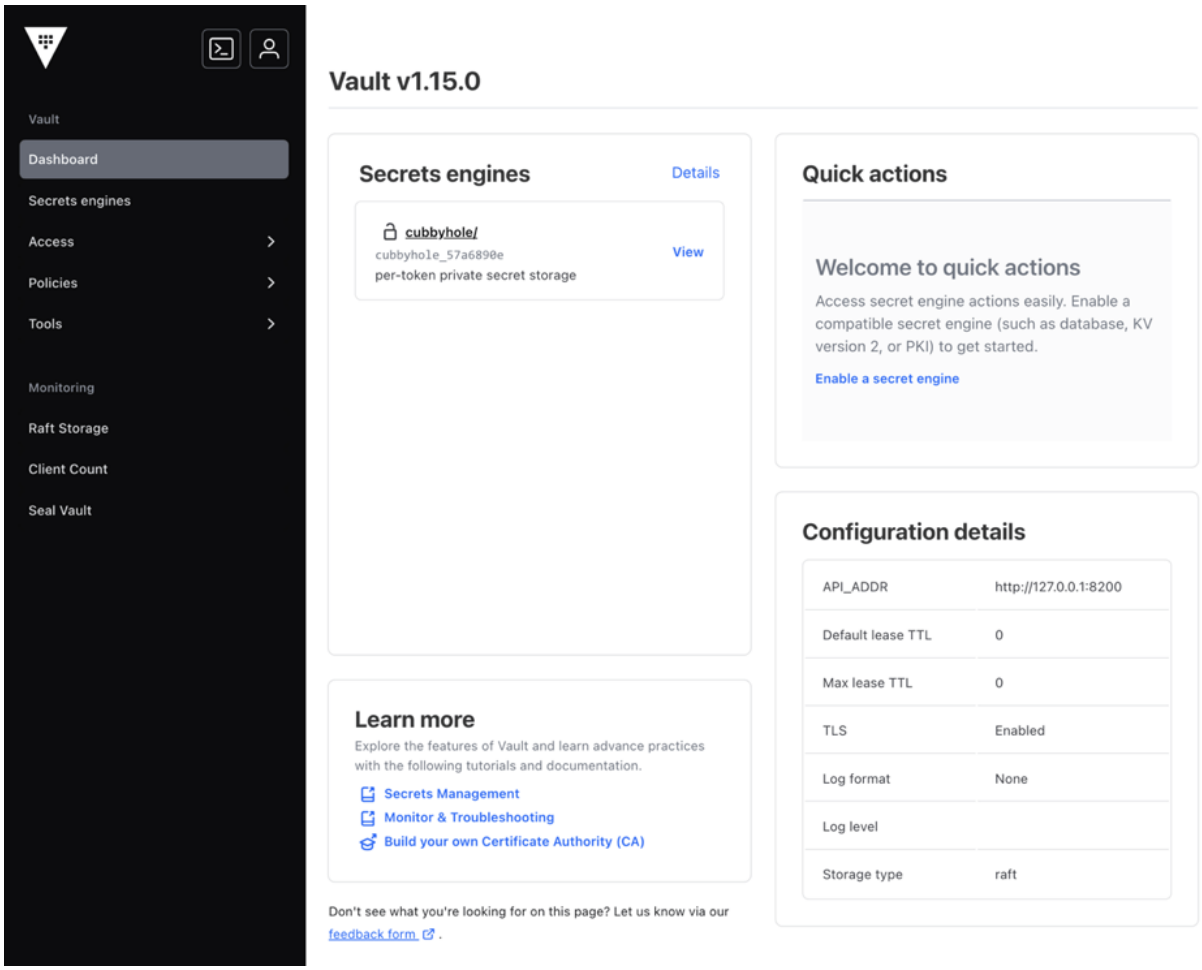



Figure 5: Vault UI dashboard (Source: HashiCorp documentation)

- **Restrict access:**

  Grant access to secrets on a need-to-know basis, ensuring that only specific roles or services can retrieve them.

- **Avoid exposing secrets:**

  Never store secrets in source code, CI/CD tools, or as plaintext environment variables. Instead, fetch them dynamically from secure vaults when needed.

- **Regularly rotate secrets:**

  Periodically update and change secrets to minimize the risks associated with any potential leaks or exposures.

# Access and authentication

## 1 Best practice: Implement multi-factor authentication (MFA) for all CI/CD tools

As cyberattacks become increasingly sophisticated, relying solely on passwords isn't enough. MFA boosts security because users must offer several verification methods, like a known password, a possessed security token, and a unique biometric identifier. Implementing MFA across all CI/CD tools significantly enhances account security, preventing unauthorized access and protecting sensitive resources.

### Actionable items

- **Enable MFA on all CI/CD-related platforms and tools:**

  Configure MFA settings on all CI/CD platforms and tools, leveraging authentication apps, hardware tokens, or SMS verification:

```
# Example of enabling MFA in GitLab

gitlab-rails runner "user = User.find_by(username: 'username');
user.require_two_factor_authentication = true; user.save!"
```

- **Educate team members about the importance of MFA:**

  Conduct awareness sessions and provide guidelines for setting up and using MFA, emphasizing its importance in safeguarding accounts.

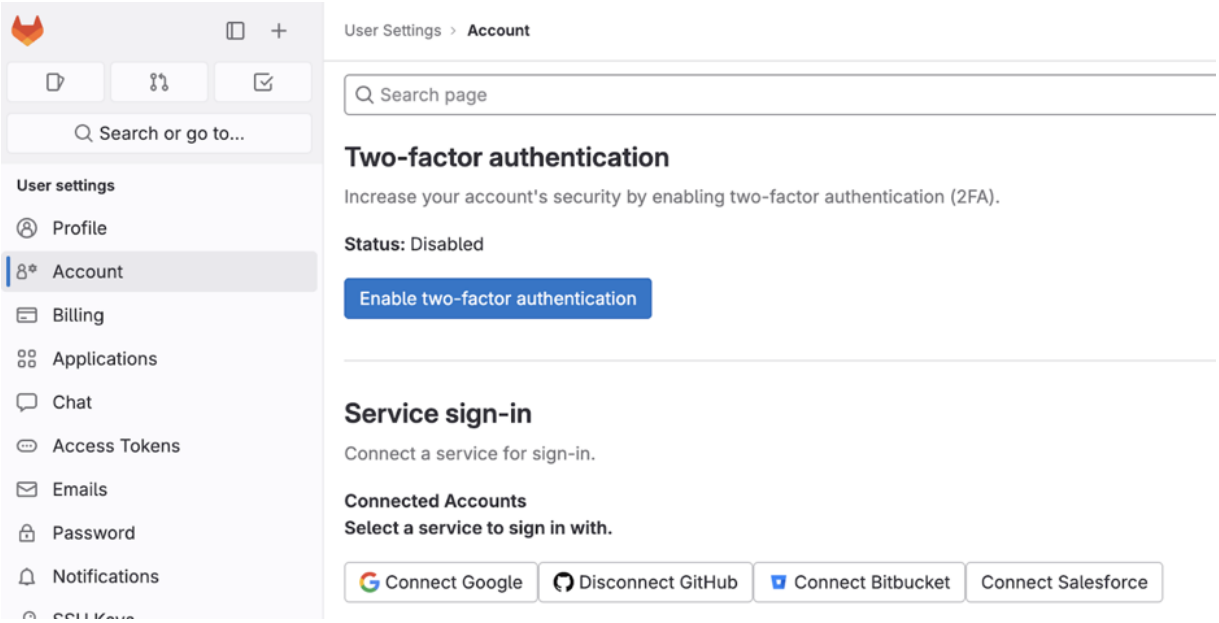



Figure 6: MFA configuration in GitLab

- **Educate team members about the importance of MFA:**
  Conduct awareness sessions and provide guidelines for setting up and using MFA, emphasizing its importance in safeguarding accounts.

- **Regularly audit MFA configurations:**
  Perform periodic audits of MFA configurations and usage logs to ensure compliance and identify any accounts that may be at risk.

## 2 Best practice: Adopt role-based access control (RBAC) with a least-privilege approach

RBAC is an approach to access management that assigns users to roles based on their job responsibilities. Each role is granted the permissions necessary to fulfill its duties. The least-privilege principle, when applied to RBAC, means assigning only the minimum necessary permissions for a role.

### Actionable items

- **Define clear roles:**
  Establish specific roles within the organization, each with a clear set of responsibilities and corresponding permissions.

- **Assign users to roles, not permissions:**
  When onboarding or modifying user permissions, assign them to roles rather than giving direct permissions. This ensures consistency and simplifies management.

## 3 Best practice: Regularly audit user access and permissions

Over time, users/roles may accumulate permissions they no longer need, or employees may leave your organization. Regularly auditing user access and permissions is an essential means of maintaining the principle of least privilege and ensuring that only authorized individuals have access to your applications.

### Actionable items

- **Schedule quarterly access reviews:**
  Schedule and conduct access reviews each quarter to be sure least privilege is in effect.

- **Use tools to automate access reviews and audits:**
  Use access-review tools and reports to validate user access and permissions. By automating the review process, you ensure accuracy and efficiency:

```
# Configuring RBAC in Argo CD

apiVersion: v1

kind: ConfigMap
```

```
metadata:
name: argocd-rbac-configmap
namespace: argo
data:
policy.default: role:readonly
policy.csv: |
p, role:org-admin, applications, *, */*, allow
p, role:org-admin, clusters, get, *, allow
p, role:org-admin, repositories, get, *, allow
p, role:org-admin, repositories, create, *, allow
...
```

- **Revoke access for users who no longer need it:**
  Promptly revoke access and permissions for users who no longer require them, such as employees who are leaving the organization or changing roles.

# Monitoring and response

## 1 Best practice: Set up anomaly detection for your CI/CD pipelines

Anomalies in CI/CD pipelines can indicate unauthorized changes or potential security incidents. Setting up anomaly detection provides early warning of unusual activities, allowing for quicker response. Anomaly detection tools use AI and machine learning to monitor CI/CD pipelines for unusual activities or deviations from normal behavior, providing alerts that can be acted on immediately.

### Actionable items

- **Use monitoring tools with AI capabilities:**
  Integrate advanced monitoring tools like Datadog or New Relic:

```
# Example of setting up an anomaly detection monitor in Datadog

api.Event.create(title="Anomaly Detected", text="An unusual
activity detected in the CI/CD pipeline.", priority="normal",
tags=["anomaly", "CI/CD"])
```

- **Train the model to recognize normal CI/CD behavior:**

  Configure and train the anomaly detection model to recognize normal pipeline behavior and identify deviations.

- **Set up alerts for detected anomalies:**

  Configure alerting mechanisms to notify relevant team members immediately when anomalies are detected, enabling prompt investigation and response.

## 2 Best practice: Have an incident-response plan specifically for CI/CD breaches

A dedicated incident-response plan for CI/CD breaches guarantees your team knows how to act swiftly and effectively in case of a security incident, which reduces potential damage and downtime.

### Actionable items

- **Draft a CI/CD-specific incident-response plan:**

  Develop a comprehensive incident-response plan that outlines the procedures, roles, and communication strategies to be followed in the event of a CI/CD breach:

```
# Example of an automated script to lock down CI/CD tools in case
of a breach

gitlab-rails runner "Feature.disable(:ci_cd)"
```

- **Conduct regular drills, such as simulated cyberattacks or data breach scenarios, to ensure your team is familiar with the plan:**

  Schedule and conduct these specific incident-response drills to make sure team members are well-versed in security procedures and can act swiftly when faced with a real incident.

- **Review and update the plan after any incident or significant change in the CI/CD setup:**

  Regularly review the incident-response plan and update it to reflect any changes in the CI/CD setup or lessons learned from previous incidents.

# Comprehensive CI/CD protection with Wiz

Comprehensive CI/CD protection with Wiz

CI/CD security is constantly evolving, so staying informed and prepared is a critical part of mitigating risks. The advanced best practices outlined in this cheat sheet serve as a roadmap for achieving a secure and resilient CI/CD pipeline. One key takeaway? Automation is your ally in maintaining security at the highest level.

Wiz's automated tools, such as real-time threat detection, IaC scanning, and container security provide robust solutions for ensuring your CI/CD pipelines remain secure. Integrating Wiz early in your development workflows allows for the detection of vulnerabilities, ensuring a fortified and resilient CI/CD process.

For comprehensive protection, discover how Wiz's cutting-edge security solutions are specifically tailored for CI/CD pipelines. Request a Wiz demo today and see how you can seamlessly integrate CI/CD pipeline security best practices with our unified platform.