Integrating Security Practices into CI/CD Pipelines in a Backend Development Environment

Romelo Gilbert
CYSE 368 - Internship Reflection #1
Watsco Inc
02/20/2026

Modern software development increasingly relies on Continuous Integration and Continuous Deployment (CI/CD) pipelines to deliver applications efficiently and reliably. As deployment frequency increases, so does the potential attack surface. Integrating security practices into CI/CD pipelines is essential to ensure that applications are not only functional but also secure throughout their lifecycle. During my role as a backend developer at a leading HVAC/R equipment, supplies, and services company. I gained practical experience working with CI/CD pipelines and learned how security considerations must be embedded at every stage of the development and deployment process.

Overview of the CI/CD Pipeline

In my role, the CI/CD pipeline is designed to support microservices and API based architectures. Source code is stored in Bitbucket, where developers collaborate using pull requests (PRs) to review and approve changes. Jenkins is used as the primary CI/CD tool, and builds are triggered manually. The system supports three environments: development (dev), staging (stg), and production (prod).

When a Jenkins build is initiated, the pipeline performs several key actions. First, automated tests are executed during the Docker image build process to ensure code quality and functionality. Once the tests pass, the Docker image is built and pushed to a container registry. The application is then deployed to a Kubernetes cluster, where Helm is used to manage

deployment configurations. This multi environment approach allows for testing and validation before promoting code to production.

Security Practices Implemented

One key practice is the use of pull requests for code changes. PRs enforce peer review, which helps identify potential security issues before code is merged into the main branch. Dependency vulnerability scanning is performed using the Wiz CLI tool. This ensures that known vulnerabilities in third-party libraries are identified early in development. By catching these issues before deployment, we reduce the risk of introducing exploitable code into production.

Secrets management is handled through AWS Secrets Manager. Sensitive information, such as production credentials are not stored directly in the codebase. It is securely retrieved at runtime, reducing the risk of exposure.

At the application and network level, multiple layers of security are enforced. External requests must pass through an API gateway, where valid JSON Web Tokens (JWTs) are required for authentication before traffic is allowed into the Kubernetes cluster. Once inside the cluster, service-to-service communication is restricted to machines operating within the organization's VPN. These internal calls require machine-to-machine authentication using JWT tokens, ensuring that only authorized services can communicate with one another.

In addition to authentication controls, Kubernetes network policies are used to limit communication between pods. This segmentation helps prevent lateral movement within the cluster in the event of a compromise. Production environments also enforce HTTPS and TLS, ensuring secure communication between services.