Container Security & Optimization Engine

Summary

Build a comprehensive container analysis and optimization system that identifies security vulnerabilities, optimization opportunities, and automatically generates improved container configurations. The solution should integrate with CI/CD pipelines to provide continuous feedback and prevent security issues from reaching production.

Objectives/Goals

1. Analyze container images for security vulnerabilities and optimization opportunities
2. Generate optimized container configurations following current best practices
3. Integrate with CI/CD pipelines for continuous security and optimization feedback
4. Provide automated remediation for common container issues
5. Track improvement metrics (size reduction, security score, build time)
6. Block risky deployments based on configurable security policies

Requirements

- Container image analysis engine using vulnerability scanners and optimization tools
- Dockerfile analysis and optimization recommendations
- Security policy enforcement with customizable rules and thresholds
- Automated generation of optimized Dockerfiles and build processes
- CI/CD integration with major platforms (GitHub Actions, GitLab CI, Jenkins)
- Metrics tracking and reporting system
- Multi-stage build optimization and layer analysis

Tech Stack

Security Scanning: Trivy, Grype, or Anchore
Container Tools: Docker, Podman
CI/CD: GitHub Actions, GitLab CI, Jenkins
Backend: Python (FastAPI) or Go
Frontend: React for dashboard and reporting
Metrics: Prometheus + Grafana or custom analytics
Cloud: AWS ECR, Azure ACR, or Google GCR for testing

Deliverables

1. Working container analysis engine with security and optimization scanning
2. Automated Dockerfile optimization system
3. CI/CD integration with security gates and automated blocking
4. Metrics dashboard showing security scores, size reductions, and build performance

5. Policy configuration system for organizational security requirements
6. Documentation with best practices and configuration examples

Evaluation Criteria

- Effectiveness of security vulnerability detection (25%)
- Quality of optimization recommendations and automated fixes (25%)
- CI/CD integration and policy enforcement (20%)
- Measurable improvements in container metrics (15%)
- Performance and scalability of analysis engine (10%)
- User experience and documentation quality (5%)

Additional Guidelines

- Focus on real-world container security issues and optimization opportunities
- Ensure generated optimizations follow current Docker best practices
- Include support for multi-architecture builds as bonus feature
- Test with various base images and application types to ensure broad compatibility

Sample Repositories

Primary Recommendation: DVWA (Damn Vulnerable Web Application)

- Repository: digininja/DVWA
    - Intentionally contains security vulnerabilities
    - Docker setup with multiple containers (PHP app + MySQL)
    - Common web application security issues
    - Clear before/after optimization opportunities

Secondary Options:

1. WordPress with Docker
    - Repository: docker-library/wordpress (official Docker image)
    - Popular PHP application with known optimization patterns
    - Large base images, many optimization opportunities
    - Real production usage patterns
2. Node.js Microservices Example
    - Repository: GoogleCloudPlatform/microservices-demo
    - Multiple microservices with different optimization needs
    - Various languages (Node.js, Python, Go, Java)
    - Real microservices patterns and concerns
3. Spring Boot Sample Applications
    - Repository: spring-projects/spring-boot (samples directory)
    - Java applications with large base images

    - Common enterprise patterns
    - Multiple optimization opportunities

What Teams Can Optimize:

- Security Issues: Running as root, outdated packages, exposed secrets
- Size Optimization: Multi-stage builds, Alpine base images, dependency cleanup
- Performance: Layer optimization, build caching, startup time
- Best Practices: Health checks, non-root users, minimal attack surface