Universal GitLab CI/CD Pipeline

1 Overview

This CI/CD pipeline is a universal, technology-agnostic solution built with GitLab CI/CD that works with any containerizable application. The pipeline integrates Docker builds, security scanning with Trivy, and comprehensive monitoring using Prometheus and Grafana.

2 Pipeline Stages

The pipeline consists of 6 sequential stages:

1. **Setup** - Install dependencies using `install-deps.sh`
2. **Test** - Run tests and build using `build.sh` and `test.sh`
3. **Build** - Create Docker images with Kaniko and push to GitLab registry
4. **Security Scan** - Vulnerability scanning with Trivy using `scan-docker.sh`
5. **Deploy** - Deploy containers to VM using `deploy.sh`
6. **Monitoring** - Setup Prometheus/Grafana stack using `monitoring.sh`

3 Key Features

3.1 Universal Design

- Supports any technology (Node.js, Python, Java, Go) through Docker
- Automatic detection of project type in test stage
- Modular shell scripts for maintainability

3.2 Security Integration

- Trivy scanner for vulnerability assessment
- Configurable security failure thresholds
- SARIF output for compliance reporting
- Manual nginx-certbot script for SSL setup

3.3 Monitoring Stack

- Prometheus for metrics collection (15s intervals)
- Grafana for visualization
- Node Exporter for system metrics
- cAdvisor for container metrics

4 Configuration

Key configurable variables:

```
# Application settings
APP_NAME: "my-app"
CONTAINER_PORT: "80"
HOST_PORT: "5004"

# Security settings
SECURITY_FAIL_ON: "CRITICAL,HIGH"
TRIVY_VERSION: "0.48.3"

# Docker settings
IMAGE_TAG_SHORT: "${CI_COMMIT_SHORT_SHA}"
DOCKER_RESTART_POLICY: "unless-stopped"
```

5 Advantages

- **Technology Agnostic**: Works with any containerizable app
- **Security First**: Automated vulnerability scanning
- **Complete Monitoring**: Built-in observability stack
- **Modular Design**: Easy to maintain and modify
- **Multi-environment**: Handles dev/main branches differently
- **Simple Architecture**: Uses Docker and shell scripts only

6 Limitations

- **Docker Required**: Applications must be containerizable
- **VM Deployment Only**: Not designed for Kubernetes/cloud-native
- **Sequential Stages**: No parallel execution
- **Runner Requirements**: Needs both Docker and Shell runners
- **Manual SSL Setup**: nginx-certbot script requires manual execution

7 Important Notes

- **Universality**: As a universal pipeline, occasional adjustments may be required for specific applications
- **Environment Scope**: Designed for single-machine or VM deployments only
- **Database Connection**: The pipeline currently does **not automatically connect to the database**. Any database setup or connection must be performed manually
- **Architecture**: Uses Docker and shell scripts only - no Docker Compose, Ansible, or Kubernetes
- **Manual SSL Step**: The nginx-certbot script must be run manually with proper domain and email configuration
- **Architecture**: Uses Docker and shell scripts only - no Docker Compose, Ansible, or Kubernetes

8 Future Enhancements

- **Docker Compose**: Container orchestration for easier management
- **Advanced Trivy**: Custom security policies and deeper vulnerability analysis
- **Enhanced Testing**: Code coverage, linting, and security testing
- **ModSecurity WAF**: Web Application Firewall integration
- **Automated SSL**: Automatic Let's Encrypt certificate management
- **Advanced Monitoring**: Custom dashboards and alerting

9 Architecture

9.1 Script Organization

All logic is externalized to `./scripts/` directory:

- `install-deps.sh`, `build.sh`, `test.sh`
- `scan-docker.sh`, `deploy.sh`, `cleanup.sh`
- `monitoring.sh`, `nginx-certbot.sh` (manual)

10 Conclusion

This pipeline provides a complete DevOps solution combining universal compatibility, integrated security scanning, comprehensive monitoring, and production-ready deployment automation using simple Docker and shell script architecture.

The modular design ensures maintainability while the security-first approach makes it suitable for enterprise environments.