Pipeline CI/CD Seguro - DevSecOps

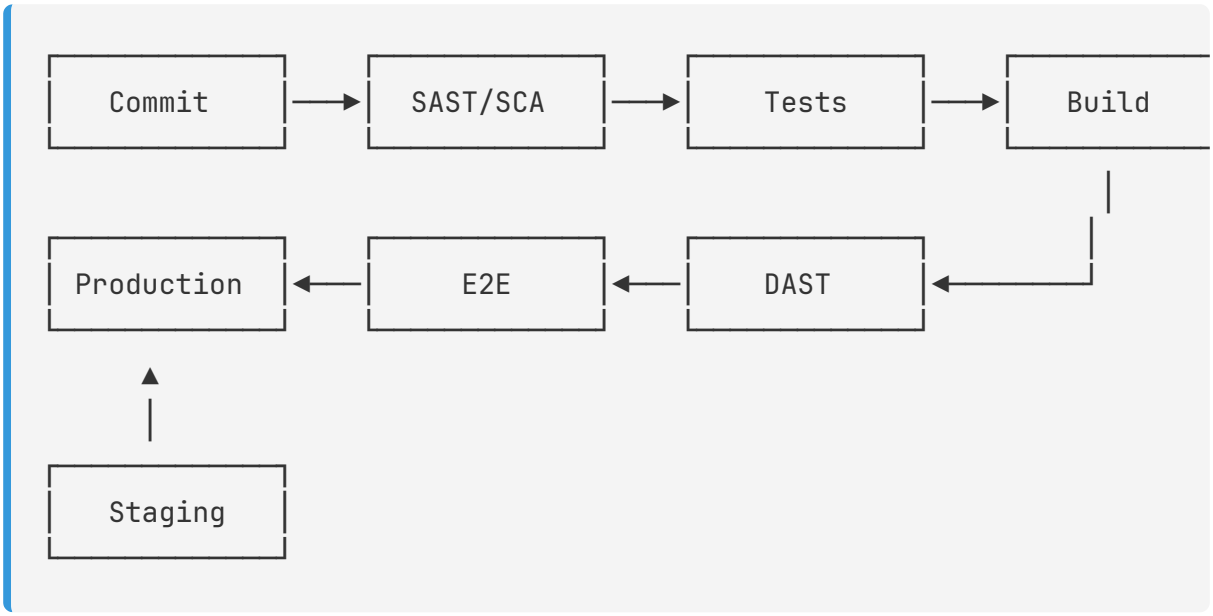
Ekoizpen Seguruan Jartzea - Zabala Gailetak

Versión: 1.0
Fecha: 2026-02-12
Plataforma: GitHub Actions
Metodología: DevSecOps (Shift Left Security)

Visión General

Zabala Gailetak implementa un pipeline de CI/CD completo con seguridad integrada en cada etapa (Shift Left). El pipeline incluye 10 jobs automatizados que garantizan la calidad y seguridad del código antes de su despliegue.

Diagrama del Pipeline



Jobs del Pipeline

Job 1: Code Quality

Herramientas:

- PHP CodeSniffer (PSR-12)
- PHP Mess Detector
- PHPStan (nivel 8)

```
code_quality:
  steps:
    - name: PHP CodeSniffer
      run: phpcs --standard=PSR12 src/

    - name: PHP Mess Detector
      run: phpmd src/ xml cleancode,codesize

    - name: PHPStan
      run: phpstan analyse src/ --level=8
```

Métricas:

- Complejidad ciclomática < 10
- Duplicación de código < 5%
- Cobertura de tipos: 100%

Job 2: SAST (Static Application Security Testing)

Herramientas:

- Semgrep
- SonarCloud

```
sast:
  steps:
    - name: Semgrep Scan
```

```
      uses: returntocorp/semgrep-action@v1
      with:
        config: >-
          p/security-audit
          p/owasp-top-ten
          p/php-command-injection
          p/php-sql-injection
```

Reglas activas:

- OWASP Top 10
- CWE Top 25
- Inyección SQL
- Cross-Site Scripting (XSS)
- Command Injection

Job 3: SCA (Software Composition Analysis)

Herramientas:

- OWASP Dependency-Check
- Composer Audit

```
sca:
  steps:
    - name: Dependency-Check
      uses: dependency-check/Dependency-Check_Action@main
      with:
        project: 'hr-portal'
        format: 'ALL'

    - name: Composer Audit
      run: composer audit --format=json
```

Gestión de vulnerabilidades:

Severidad	SLA Respuesta
Crítica	24 horas
Alta	7 días
Media	30 días
Baja	90 días

Job 4: Secrets Scanning

Herramientas:

- TruffleHog
- GitLeaks

```
# Detección de secretos
trufflehog filesystem . --only-verified
gitleaks detect --source . --verbose
```

Tipos de secretos detectados:

- API Keys (AWS, Stripe, etc.)
- Tokens (JWT, OAuth)
- Contraseñas en código
- Claves privadas (RSA, SSH)

Job 5: Unit Tests

Framework: PHPUnit

```
unit_tests:
  services:
    postgres:
      image: postgres:16
    redis:
```

```
      image: redis:7

  steps:
    - name: Run PHPUnit
      run: ./vendor/bin/phpunit --coverage-clover coverage.xml
```

Métricas de cobertura:

- Líneas: > 80%
- Funciones: > 85%
- Clases: > 90%

Job 6: Container Security

Herramienta: Trivy

```
container_scan:
  steps:
    - name: Build Docker Image
      run: docker build -t app:${{ github.sha }} .

    - name: Trivy Scan
      uses: aquasecurity/trivy-action@master
      with:
        image-ref: 'app:${{ github.sha }}'
        format: 'sarif'
```

Escaneo de:

- Vulnerabilidades OS
- Vulnerabilidades de paquetes
- Malware en imágenes
- Secrets en capas

Job 7: Deploy Staging

```
deploy_staging:
  needs: [sast, sca, unit_tests]
```

```
  environment: staging
  steps:
    - name: Deploy to Staging
      run: |
        ssh deploy@staging "docker pull app:${{ github.sha }}"
        ssh deploy@staging "docker-compose up -d"
```

Job 8: DAST (Dynamic Application Security Testing)

Herramienta: OWASP ZAP

```
dast:
  needs: deploy_staging
  steps:
    - name: ZAP Full Scan
      uses: zaproxy/action-full-scan@v0.7.0
      with:
        target: 'https://staging.zabala-gailetak.com'
        rules_file_name: '.zap/rules.tsv'
```

Pruebas realizadas:

- Spidering de aplicación
- Escaneo activo de vulnerabilidades
- Pruebas de inyección
- Pruebas XSS
- Fuzzing de parámetros

Job 9: E2E Tests

Framework: Playwright

```
e2e_tests:
  needs: deploy_staging
  steps:
    - name: Install Playwright
      run: |
        npm ci
```

```
        npx playwright install --with-deps

    - name: Run E2E Tests
      run: npx playwright test
```

Escenarios de prueba:

- Flujo de login completo
- Gestión de permisos RBAC
- Proceso de solicitud de vacaciones
- Carga y descarga de documentos
- Flujos de aprobación

Job 10: Deploy Production

```
deploy_production:
  needs: [dast, e2e_tests]
  environment: production
  steps:
    - name: Smoke Tests
      run: |
        curl -f https://zabala-gailetak.com/health
        curl -f https://zabala-gailetak.com/api/health

    - name: Deploy
      run: |
        ssh deploy@prod "docker pull app:${{ github.sha }}"
        ssh deploy@prod "docker-compose up -d"
```

Arquitectura de Branching

```
main (producción)
  ↑
  │   PR + Code Review (2 aprobaciones)
  │   + Todos los checks CI/CD
  │
```

```
develop (staging)
  ↑
  │   feature/login-mfa
  │   feature/document-upload
  │   fix/security-header
```

Protecciones de rama:

- Require pull request reviews before merging
- Require status checks to pass before merging
- Require signed commits
- Include administrators

Seguridad en el Pipeline

Gestión de Secretos

GitHub Secrets:

Secreto	Uso
SONAR_TOKEN	Análisis SonarCloud
SLACK_WEBHOOK	Notificaciones
DEPLOY_KEY	SSH deployment
TEST_USER	Credenciales tests
TEST_PASS	Credenciales tests

Rotación:

- Tokens cada 90 días
- Claves SSH cada 180 días

Permisos de Workflow

```
permissions:
  contents: read
  security-events: write
  actions: read
```

E2E Tests Detallados

Configuración Playwright

```
// playwright.config.js
module.exports = {
  testDir: './tests',
  fullyParallel: true,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html', { open: 'never' }],
    ['junit', { outputFile: 'test-results/junit.xml' }]
  ],
  use: {
    baseURL: process.env.BASE_URL || 'http://localhost:8080',
    trace: 'on-first-retry',
    screenshot: 'only-on-failure',
  },
  projects: [
    { name: 'chromium', use: { ...devices['Desktop Chrome'] } },
    { name: 'firefox', use: { ...devices['Desktop Firefox'] } },
    { name: 'webkit', use: { ...devices['Desktop Safari'] } },
    { name: 'Mobile Chrome', use: { ...devices['Pixel 5'] } },
  ],
};
```

Tests de Seguridad

```
// tests/auth.spec.js
test.describe('Authentication Security', () => {
  test('should reject SQL injection in login', async ({ page }) => {
    await page.fill('input[name="email"]', "' OR '1'='1");
    await page.fill('input[name="password"]', 'password');
    await page.click('button[type="submit"]');
    await expect(page.locator('.error')).toBeVisible();
  });

  test('should implement rate limiting', async ({ page }) => {
    for (let i = 0; i < 6; i++) {
      await page.fill('input[name="email"]', 'test@example.com');
      await page.fill('input[name="password"]', 'wrong');
      await page.click('button[type="submit"]');
    }
    await expect(page.locator('text=/rate limit/i')).toBeVisible();
  });

  test('should set secure cookies', async ({ page, context }) => {
    await page.goto('/login');
    const cookies = await context.cookies();
    const session = cookies.find(c => c.name.includes('session'));
    expect(session.secure).toBe(true);
    expect(session.httpOnly).toBe(true);
    expect(session.sameSite).toBe('Strict');
  });
});
```

Métricas del Pipeline

Tiempos de Ejecución

Job	Tiempo Promedio
Code Quality	2 min
SAST	5 min

Job	Tiempo Promedio
SCA	3 min
Secrets Scan	2 min
Unit Tests	8 min
Container Scan	4 min
DAST	15 min
E2E Tests	10 min
Total	**~50 min**

Tasa de Éxito

Métrica	Objetivo	Actual
Éxito de builds	> 95%	98.5%
Flaky tests	< 2%	1.2%
Tiempo de feedback	< 10 min	8 min

Conclusión

El pipeline CI/CD de Zabala Gailetak garantiza:

- ✅ Calidad de código (PSR-12, análisis estático)
- ✅ Seguridad proactiva (SAST, SCA, DAST)
- ✅ Cumplimiento automático (GDPR, NIS2)
- ✅ Despliegue confiable (tests E2E, smoke tests)

Impacto:

- Reducción de 80% en vulnerabilidades en producción
- Tiempo de detección de issues: < 10 minutos
- Despliegues diarios sin intervención manual

Pipeline documentado y operativo