|  |  |
| --- | --- |
| **NOMBRES ESTUDIANTES:** | Sebastián Donoso  Ismael Toala |
| **FECHA:** | 16-06-2025 |
|  |  |
|  |  |
| **TEMA:** | Lista de Procesos y Herramientas |
|  |  |

**Proceso y Herramientas de Configuración del Software**

**1. Uso de GIT**

* Se utilizará **Git** como sistema de control de versiones distribuido.
* El flujo de trabajo será basado en **Git Flow**, con ramas definidas como:
  + main: rama de producción (estable).
  + develop: rama de integración continua.
  + feature/\*: ramas para desarrollo de nuevas funcionalidades.
  + release/\*: ramas para preparación de versiones.
  + hotfix/\*: correcciones urgentes sobre producción.
* Se trabajará mediante comandos estándar:
  + git checkout -b feature/nombre
  + git merge
  + git pull, git push
  + git rebase (cuando aplique)

**2. Herramientas para Pipelines y Repositorio**

* **Repositorio principal**: GitHub (privado o público según necesidad).
* **Herramientas CI/CD** (para integración y entrega continua):
  + **GitHub Actions**: automatización nativa del repositorio.
  + Alternativas: GitLab CI, Jenkins, CircleCI (según el contexto del proyecto).
* Las pipelines incluirán:
  + Disparo automático por push o pull request.
  + Etapas configurables: build, test, lint, análisis, despliegue.

**3. Proceso de Análisis de Código**

* Automatizado mediante las siguientes etapas:

**a. Instalación y Build**

* + Instalación de dependencias (npm install, pip install, etc.).
  + Compilación del proyecto, si aplica.

**b. Linting y Formato**

* + **JavaScript/TypeScript**: ESLint + Prettier
  + **Python**: Pylint o Black
  + **.NET**: StyleCop

**c. Análisis Estático**

* + **SonarQube** para detectar vulnerabilidades, duplicación y code smells.
  + **Snyk** para análisis de dependencias vulnerables.

**d. Seguridad**

* + **OWASP Dependency-Check** para escaneo de paquetes.

**e. Pruebas**

* + Unitarias: Jest, PyTest, JUnit.
  + Integración: según arquitectura (Postman/Newman, Cypress, etc.)

**f. Cobertura**

* + Codecov, Coveralls (publicación del porcentaje de líneas cubiertas).

**4. Pautas para la Revisión de Código**

* Todo cambio debe pasar por una **pull request (PR)** a develop.
* La PR debe cumplir con lo siguiente:
  + Título y descripción clara de los cambios.
  + Tests cubriendo nuevas funcionalidades o correcciones.
  + Sin errores de lint, build ni pruebas fallidas.
  + Aprobación mínima de **2 revisores**.
  + Uso de herramientas de revisión automática (opcional):
    - **CodeRabbit** para sugerencias automáticas.
    - **SonarQube** embebido en PR para control de calidad.
* Guía de estilo:
  + Seguir las convenciones del lenguaje usado.
  + Documentación mínima para métodos y clases nuevas.
  + Evitar código duplicado o comentarios innecesarios.

**5. Flujo de Despliegue del Aplicativo**

* El flujo completo se activa cuando se realiza merge a la rama main.
* Flujo general:

**a. Post-merge**

* + Al hacer merge a main, se dispara automáticamente la **pipeline de entrega continua (CD)**.

**b. Construcción de artefactos**

* + Se genera una imagen Docker si corresponde (Dockerfile).
  + Se publica en un contenedor (Azure Container Registry o DockerHub).

**c. Despliegue**

* + **Entorno de producción**:
    - Azure App Service (web apps)
    - Azure Kubernetes Service (AKS)
    - Azure Functions (para microservicios serverless)
  + **Infraestructura como Código**:
    - Terraform
    - ARM Templates o Bicep
  + El despliegue se realiza vía scripts automatizados o pipelines CD (GitHub Actions u otro).

**d. Validación post-despliegue**

* + Validaciones automáticas de disponibilidad, logs y monitoreo.
  + En caso de fallo: rollback o alerta a DevOps/Cloud Engineer.

**Diagrama del Proceso y Herramientas de Configuración del Software**

