“UNIVERSIDAD DE LAS FUERZAS ARMADAS”

ESPE

INGENIERÍA DE SOFTWARE

DESARROLLO DE SOFTWARE APLICADO A LA INTERCULTURALIDAD

Tema: Sistema de Monitoreo para Plataforma de Préstamos

INTEGRANTES:

CRISTIAN FELIX

DANNY QUINGALUISA

WENDY QUINTANA

MATTHEW SALAZAR

TUTOR ENCARGADO:

ING. CRISTIAN COLA

NRC:

23398

FECHA DE ENTREGA:

17-06-2025

SEDE MATRIZ SANGOLQUÍ

QUITO-ECUADOR

2025

1. Introducción

Implementación de un sistema integral de monitoreo para una plataforma de microservicios dedicada a la gestión de usuarios, préstamos y pagos. El sistema utiliza herramientas open source como **Prometheus**, **AlertManager**, **Grafana**, **Node Exporter**, **cAdvisor** y **PostgreSQL Exporter**, proporcionando observabilidad completa, alertamiento proactivo y visualización de métricas críticas.

2. Arquitectura General del Monitoreo

La arquitectura está conformada por:

Microservicios:

- Usuarios (puerto 3000)
- Préstamos (puerto 3002)
- Pagos (puerto 3003)

Todos exponen el endpoint /metrics.

- **Prometheus**: Responsable de recolectar y almacenar métricas.
- **AlertManager**: Maneja las alertas disparadas por Prometheus.
- **Grafana**: Plataforma de visualización para dashboards y análisis.
- **Node Exporter**: Exporta métricas del sistema operativo.
- **cAdvisor**: Exporta métricas de contenedores
    - **PostgreSQL Exporter**: Exporta métricas específicas de la base de datos.

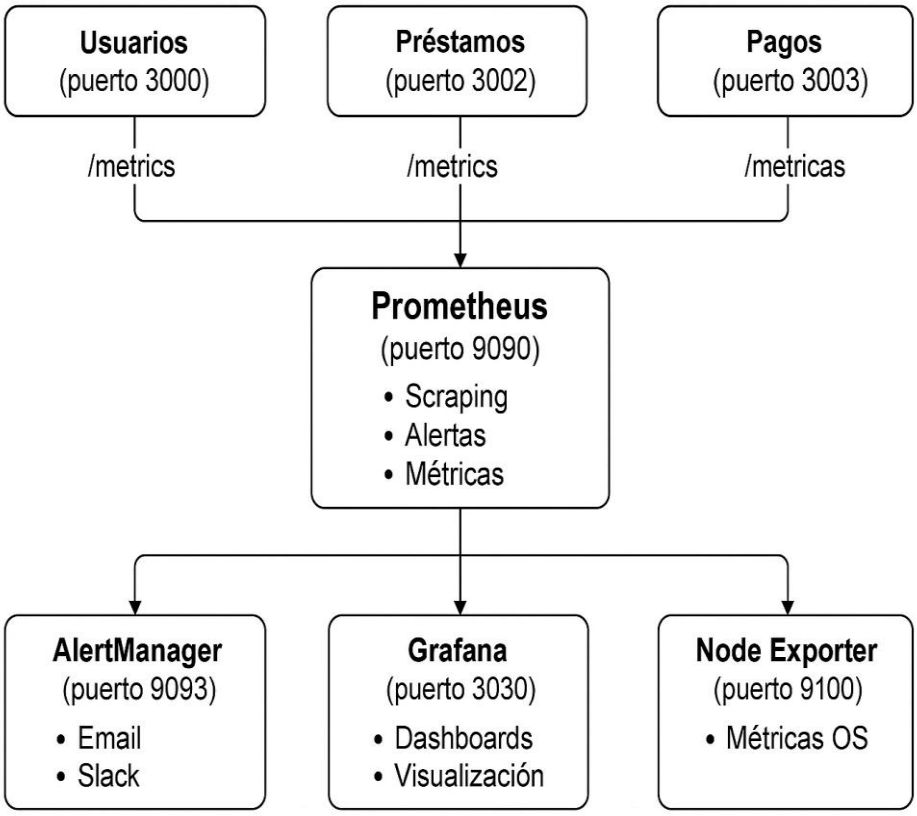

Imagen 1. Arquitectura

3. Proceso de Implementación

3.1. Inicio del stack

El stack puede iniciarse con un script PowerShell:

.\start-monitoring.ps1

Este script inicia **Prometheus** junto con los otros servicios del stack de monitoreo (AlertManager, Grafana, Node Exporter, etc.).

3.2. Arranque de servicios

Los microservicios deben iniciarse manualmente en terminales separadas mediante npm run dev.

3.3. Interfaces disponibles

Servicio	URL
Prometheus	http://localhost:9090
AlertManager	http://localhost:9093
Grafana	http://localhost:3030
Node Exporter	http://localhost:9100
cAdvisor	http://localhost:8080

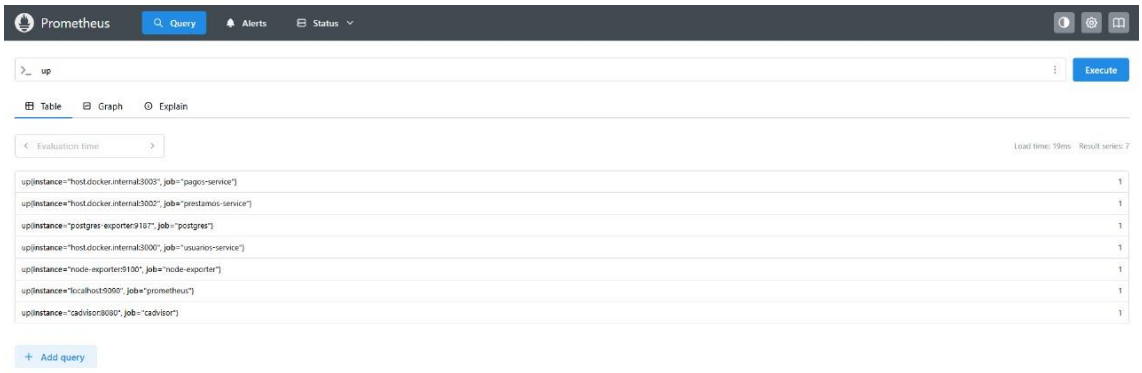

Imagen 1. Prometheus

Aquí se accede a la interfaz de Prometheus para ejecutar consultas PromQL y ver el estado de los targets.

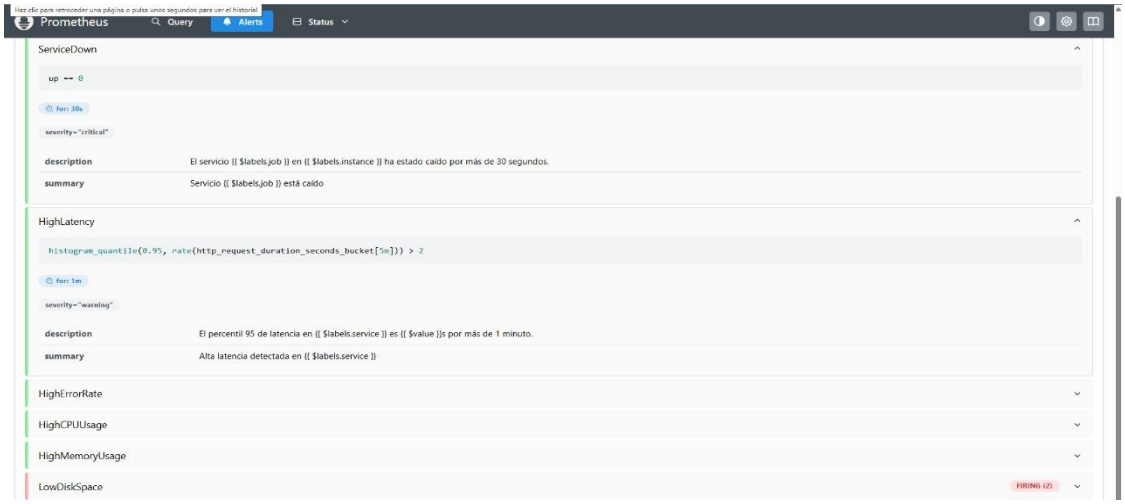

Imagen 2. estadísticas prometheus

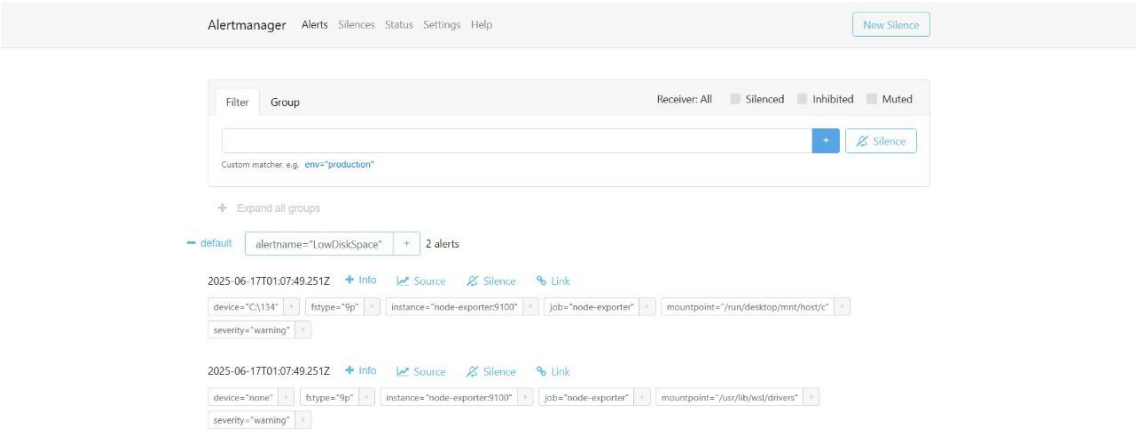

Imagen 3. Alert manager

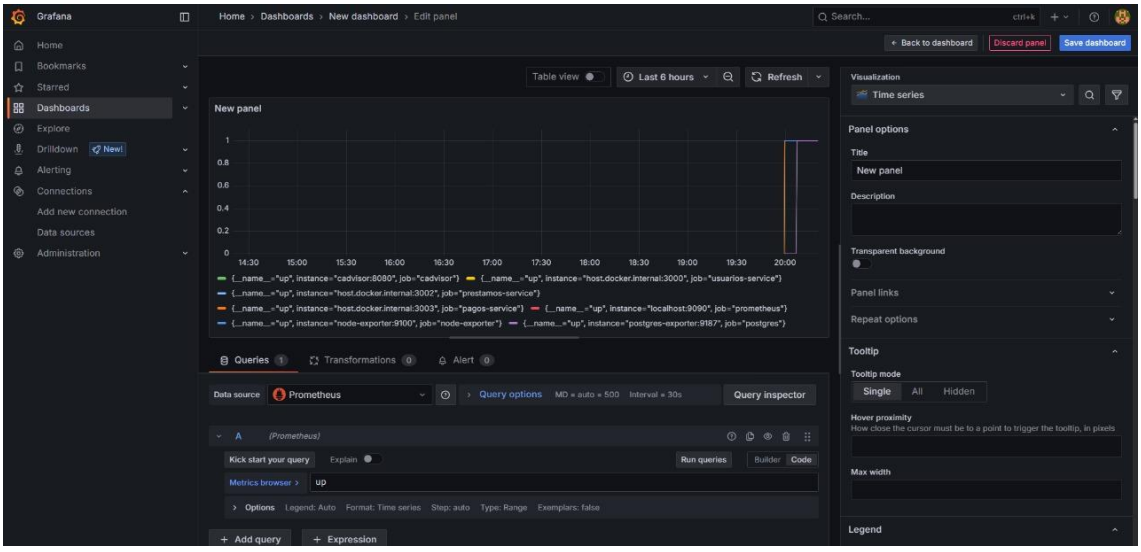

Imagen 4. Grafana estadísticas

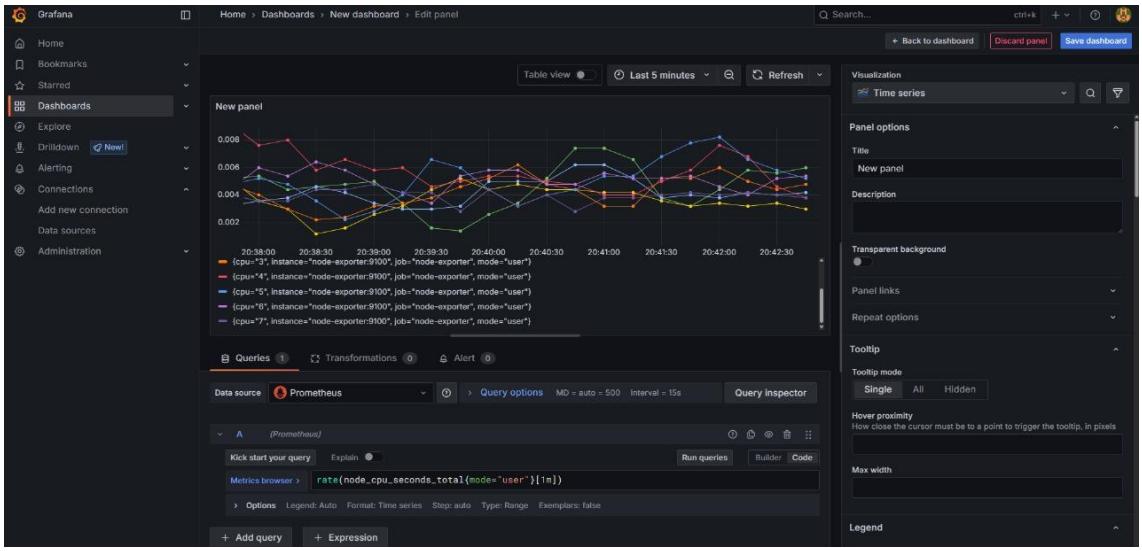

Imagen 5. Grafana estadísticas

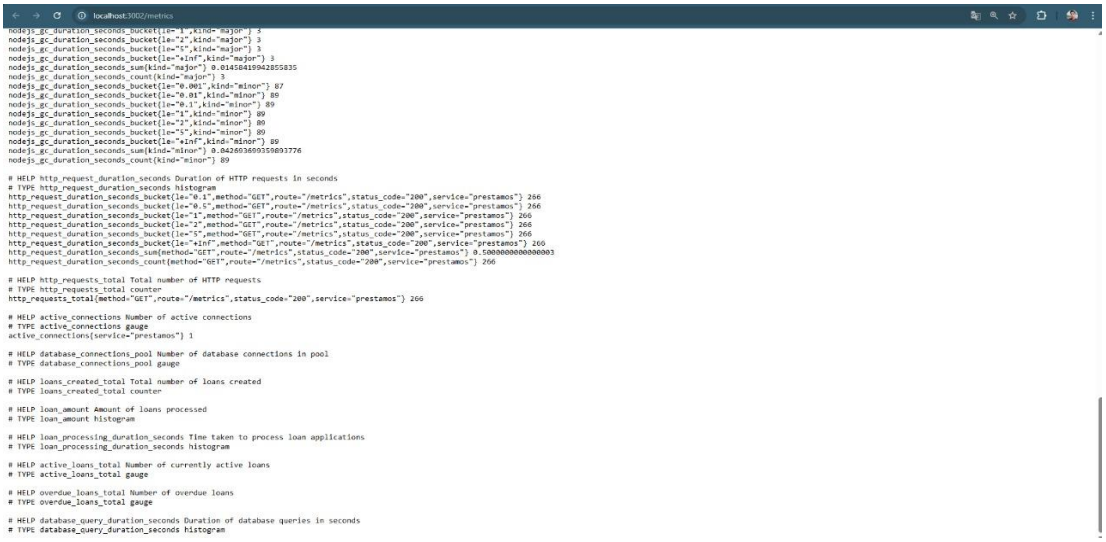

Imagen 6. Métricas de los microservicios

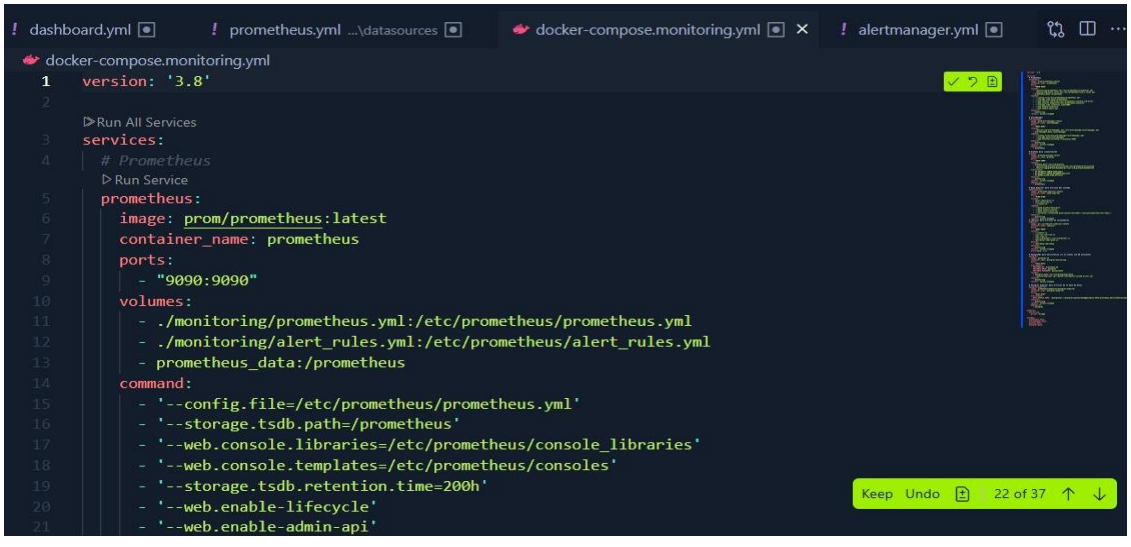

Imagen 7. Archivo de configuración docker

4. Métricas Recolectadas

4.1. Métricas HTTP

- http_requests_total
- http_request_duration_seconds
- active_connections

4.2. Métricas del Sistema (Node Exporter)

- node_cpu_seconds_total
- node_memory_*
- node_filesystem_*
- database_connections_pool

4.3. Métricas de Negocio

Servicio	Métricas Clave
Usuarios	Registros y logins
Pagos	Pagos procesados, montos y duración
Préstamos	Préstamos creados, activos, vencidos

5. Sistema de Alertas

5.1. Alertas de Infraestructura

- Servicios caídos (ServiceDown)
- Alta latencia (HighLatency)
- Errores HTTP 5xx (HighErrorRate)
- Alto uso de CPU, memoria o disco

5.2. Alertas de Negocio

- Alta tasa de fallos en pagos o registros
- Préstamos vencidos en exceso
- Consultas lentas en la base de datos

6. Configuración de Notificaciones

- Email: A través de SMTP en alertmanager.yml
- Slack: Mediante Webhooks configurables

7. Visualización de Datos

Se utiliza **Grafana** para la visualización de métricas con dashboards preconfigurados y consultas personalizadas usando **PromQL**.

- Estado de servicios (up)
- Latencia P95 (histogram_quantile)
- Métricas de sistema
- Métricas de pagos y préstamos

8. Git hub Actions

GitHub Actions es un sistema de automatización basado en eventos. Te permite escribir workflows (flujos de trabajo) en un archivo YAML para que se ejecuten automáticamente en función de eventos como:

- Push a un branch
- Pull request
- Creación de tags o releases
- Cron (ejecuciones programadas)

Componentes clave

Componente	Descripción
Workflow	Archivo YAML que define los pasos a ejecutar. Está en .github/workflows/.
Job	Conjunto de pasos que se ejecutan en un mismo runner.
Step	Acción individual dentro de un job (puede ser un comando o una acción).
Runner	Máquina que ejecuta los jobs (GitHub-hosted o self-hosted).
Action	Componente reutilizable que realiza una tarea específica (como checkout).

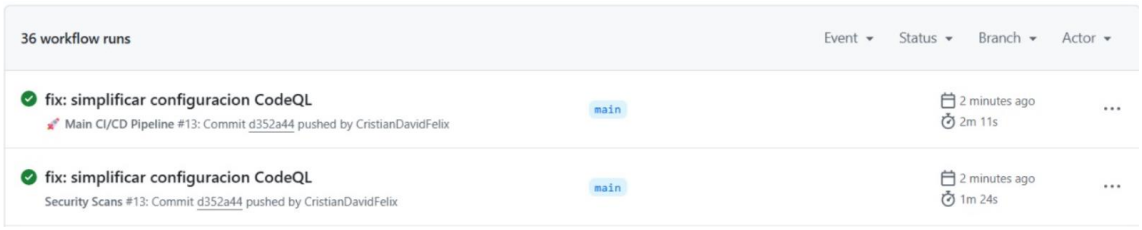

Imagen 8. Workflow

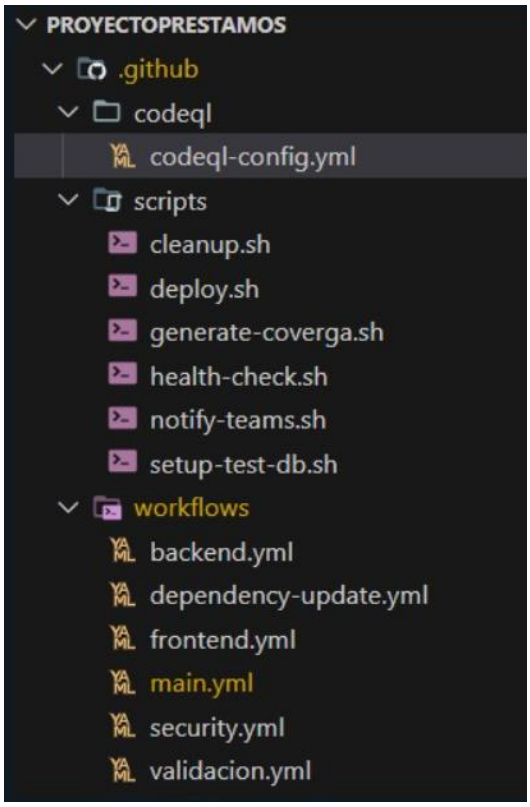

Imagen 9. GitHub

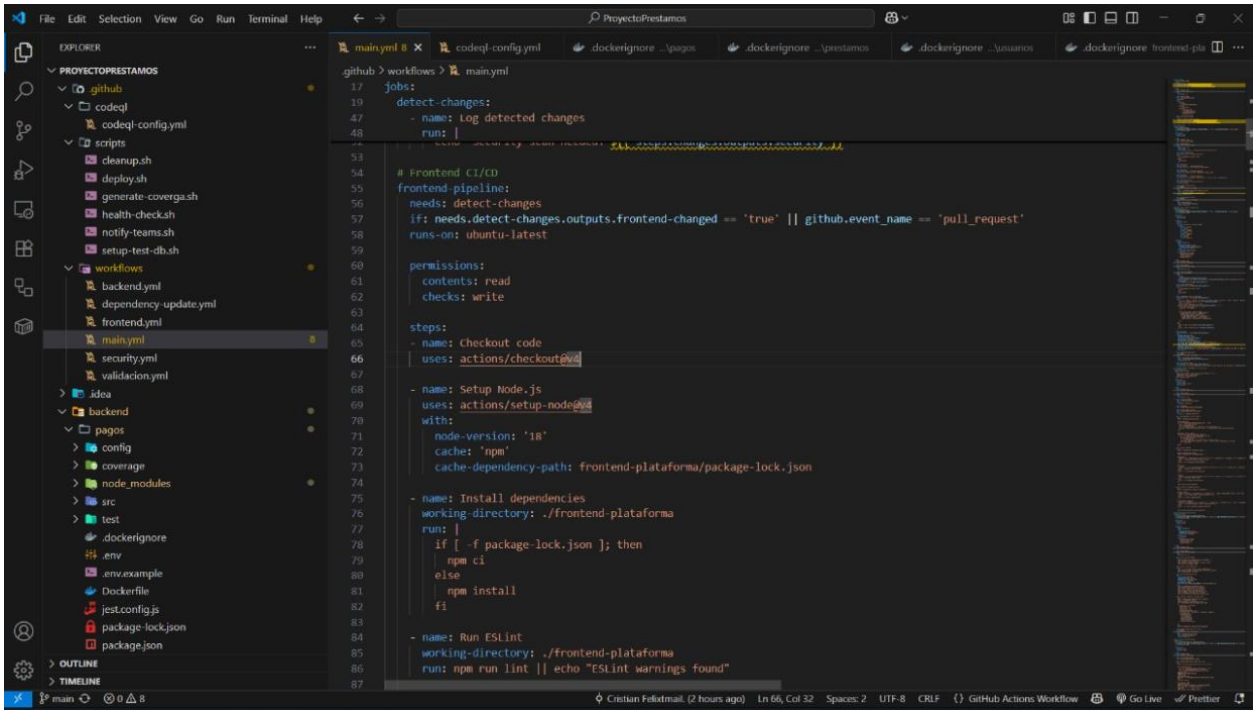

Imagen 10. Main.yml

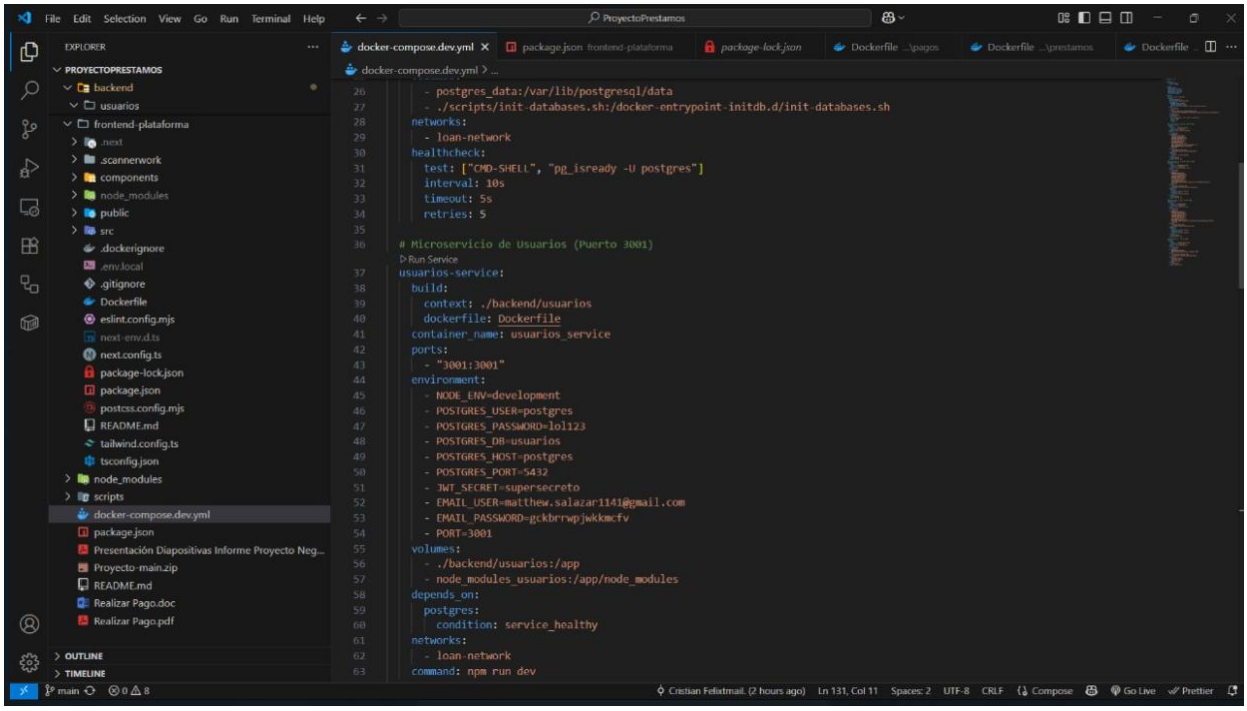

Imagen 11. Docker-Compose.dev.yml

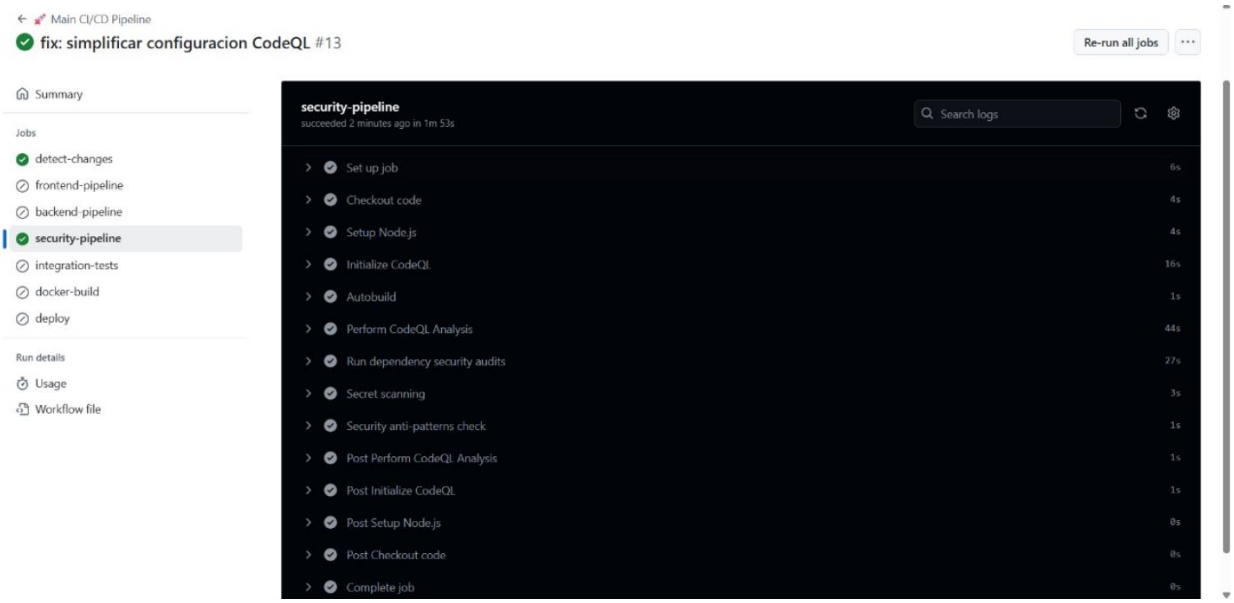

Imagen 12. Action

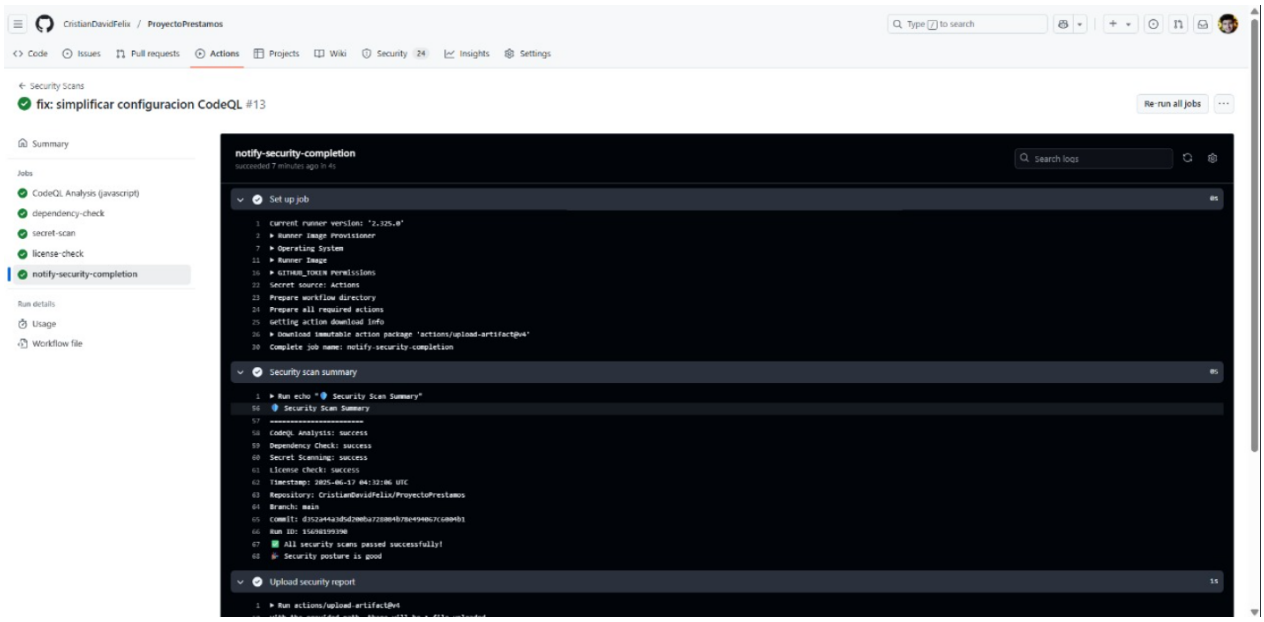

Imagen 13. Action

8. Implementación de la Interculturalidad

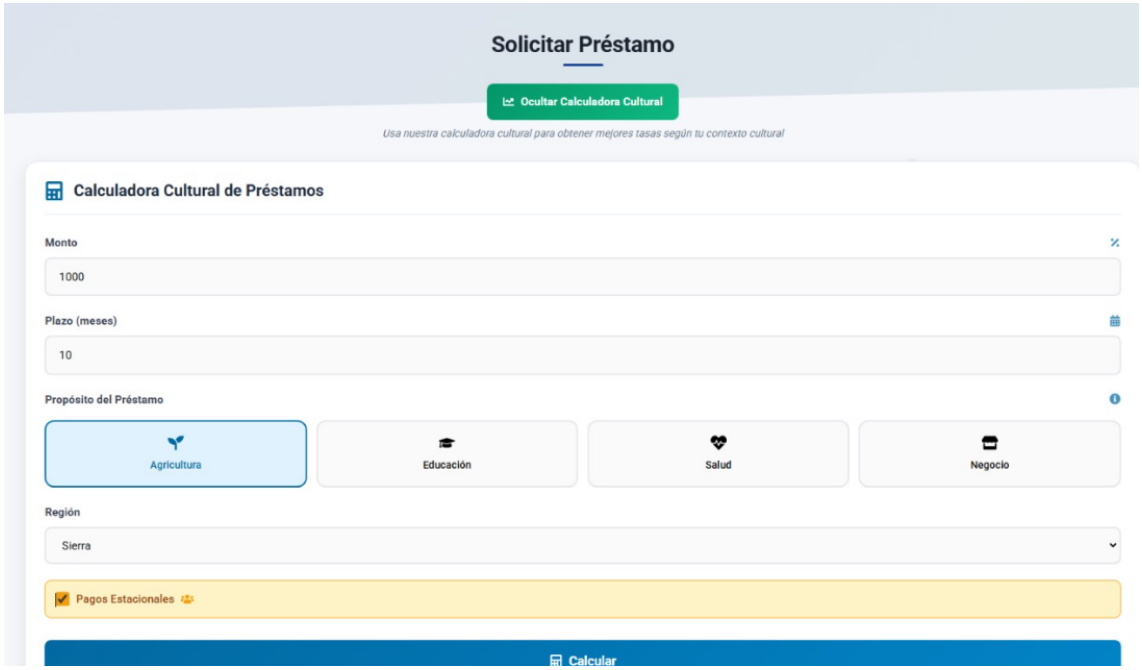

Imagen 14. Implementación de interculturalidad

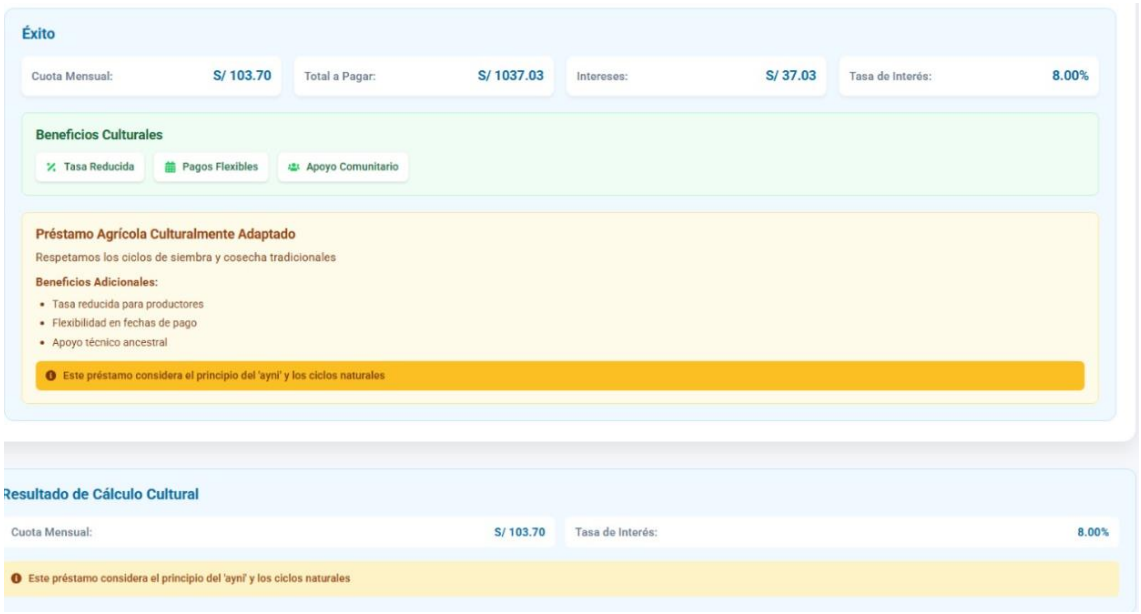

Imagen 15. Implementación de interculturalidad

9. Buenas prácticas en la documentación y trazabilidad de requisitos en el Desarrollo de Software Global (DSG)

Paso 1: Definición de la ERS (Especificación de Requerimientos)

Este sistema web debe:

Para el cliente:

- Registrarse e iniciar sesión de forma segura.
- Solicitar préstamos desde la plataforma.
- Ver sus préstamos activos y su historial crediticio.
- Realizar pagos de cuotas y visualizar su estado.
- Consultar su información financiera personal.
- Visualizar una tabla de amortización por préstamo.

Para el administrador:

- Aprobar o rechazar solicitudes de préstamos.
- Registrar manualmente información en el historial crediticio del cliente.
- Gestionar usuarios y sus datos financieros.

Paso 2: Identificación y clasificación de componentes (PF)

Frontend (React)

Tipo de Componente	Descripción	Cantidad	Complejidad	PF / unidad	Total PF
Entradas Externas (EE)	Login, registro, solicitud de préstamo, pagos, edición de datos financieros	5	Media	4	20

Salidas Externas (SE)	Tablas de amortización, confirmaciones, historial crediticio, notificaciones	3	Media	5	15
Consultas Externas (CE)	Consultas a datos de préstamos, estado de pagos, perfil financiero	2	Baja	3	6
Archivos Lógicos Internos (ALI)	Carrito de datos, datos del préstamo temporal en cliente	2	Media	10	20
Total Frontend PF					**61 PF**

KLOC estimado frontend:
KLOC_frontend = (61 × 53) / 1000 ≈ 3.23 KLOC

Backend (Node.js + PostgreSQL)

Tipo de Componente	Descripción	Cantidad	Complejidad	PF / unidad	Total PF
Entradas Externas (EE)	CRUD de usuarios, préstamos, pagos, historial crediticio	5	Media	4	20
Salidas Externas (SE)	Reportes de pagos, tablas de amortización, dashboards para el admin	2	Media	5	10
Interfaces Externas (IE)	Posible integración futura con burós de crédito o sistemas bancarios	1	Media	7	7
Archivos Lógicos Internos (ALI)	Usuarios, préstamos, pagos, historial, amortizaciones	5	Media	10	50
Consultas Externas (CE)	Filtros por fecha, tipo de préstamo, estado de pagos	2	Media	4	8
Total Backend PF					**95 PF**

KLOC estimado backend:
KLOC_backend = (95 × 53) / 1000 ≈ 5.04 KLOC

Resultado Final del Sistema

Parte del sistema	Puntos de Función	LOC estimado	KLOC
Frontend (React)	61 PF	3233 LOC	3.23
Backend (Node.js)	95 PF	5015 LOC	5.04
TOTAL SISTEMA	**156 PF**	**8248 LOC**	**8.25**

Aplicación del Modelo COCOMO

Datos base:

- KLOC estimado: 8.25
- Tipo de proyecto: Orgánico
- Coeficientes: a = 2.4, b = 1.05, c = 2.5, d = 0.38

1. Esfuerzo (persona-mes):
E = 2.4 × (8.25)^1.05 ≈ 2.4 × 9.35 ≈ **22.44 PM**

2. Tiempo (meses calendario):
T = 2.5 × (22.44)^0.38 ≈ 2.5 × 3.12 ≈ **7.80 meses**

3. Personal requerido:
P = 22.44 / 7.80 ≈ **2.88 personas**

4. Productividad:
PROD = 8.25 / 22.44 ≈ **0.37 KLOC/persona-mes**

5. Costo estimado:
CT = 22.44 × $200 = **$4,488 USD**

Link de Git Hub:

https://github.com/CristianDavidFelix/ProyectoPrestamos

¿Cómo se resolvieron los conflictos de timezone? (Ej: horarios de merge)

Los conflictos de zona horaria se resolvieron mediante:

- Uso de UTC como estándar en los workflows de GitHub Actions, lo que permite uniformidad en la ejecución de jobs, sin importar la ubicación de los colaboradores.
- Sincronización del equipo con herramientas colaborativas como Slack y Google Calendar, donde se agendaron actividades críticas (como merges o despliegues) en franjas horarias comunes para ambos equipos.
- Automatización de merges y releases mediante GitHub Actions programados con `cron`, eliminando la dependencia de la presencia humana en diferentes husos horarios.
- Revisión de Pull Requests de forma asincrónica, con reglas claras sobre quién aprueba y en qué tiempo, permitiendo continuidad del trabajo sin importar la zona horaria.

¿Qué pasaría si el equipo en India no tiene acceso a Docker Hub?

Si el equipo en India no tiene acceso a Docker Hub, podrían surgir varios problemas:

- Imposibilidad de obtener imágenes base para desarrollo, pruebas o despliegue automático.
- Fallos en los workflows de CI/CD configurados para extraer imágenes desde Docker Hub.
- Pérdida de sincronización en entornos de desarrollo, lo que puede generar errores inesperados al ejecutar contenedores.

Soluciones:

- Configurar un mirror privado de Docker Hub en una red local o en la nube accesible desde India.
- 2Subir las imágenes necesarias a un registro alternativo como GitHub Container Registry (GHCR), Amazon ECR o un registro privado en Azure o GCP.
- Construcción local de las imágenes y compartición mediante archivos `.tar` o repositorios de artefactos.

Reflexión basada en su trabajo como desarrolladores

Trabajar en un equipo distribuido, con diferentes zonas horarias y acceso desigual a herramientas como Docker Hub, nos ha enseñado el verdadero valor de la automatización, estandarización y comunicación clara.

GitHub Actions no solo nos permitió automatizar tareas repetitivas, sino que nos brindó una plataforma de confianza para coordinar despliegues y pruebas sin importar la hora ni el lugar. Pero también aprendimos que la tecnología no lo es todo: los procesos humanos, como definir políticas de revisión o consensuar flujos de trabajo, son igual de importantes.

Finalmente, enfrentarnos a limitaciones como el acceso restringido a servicios globales nos hizo más resilientes y creativos. Hoy entendemos que ser desarrollador no es solo escribir código, sino resolver problemas reales con empatía, colaboración y adaptabilidad.

8. Conclusiones

- Se logró establecer un sistema de monitoreo robusto que permite observar en tiempo real el estado de los microservicios de usuarios, préstamos y pagos, gracias a la integración de Prometheus y Grafana.
- Con el uso de AlertManager, se automatizó el envío de notificaciones ante condiciones críticas, mejorando la capacidad de respuesta ante caídas o comportamientos anómalos del sistema.
- La recopilación de métricas tanto técnicas (CPU, memoria, conexiones) como de negocio (pagos procesados, préstamos activos) mejora la visibilidad y toma de decisiones en base a datos reales.
- La arquitectura basada en microservicios permite escalar cada componente de forma independiente, y el sistema de monitoreo se adapta fácilmente al crecimiento del ecosistema.
- La documentación y estimación técnica basada en Puntos de Función y COCOMO proveen una base sólida para planificar, medir esfuerzo y anticipar costos.

9. Recomendaciones

- Incorporar certificados TLS y autenticación en Prometheus, Grafana y AlertManager para asegurar la comunicación en entornos reales.
- Configurar almacenamiento persistente para Prometheus y Grafana, asegurando la disponibilidad histórica de las métricas.
- Incluir pruebas automatizadas para validar el disparo de alertas ante fallos simulados y garantizar su funcionamiento.
- Entrenar al equipo de desarrollo y operaciones en el uso de PromQL y la lectura de dashboards para maximizar el aprovechamiento del monitoreo.
- Incluir PostgreSQL Exporter para un monitoreo más profundo del rendimiento de la base de datos.

10. Bibliografía (APA 7)

- Bartholomew, D., & Braun, J. (2021). Monitoring with Prometheus and Grafana: Performance metrics and alerting. O'Reilly Media.

- Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). *Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade*. ACM

Queue, 14(1), 70–93. https://doi.org/10.1145/2890784.2890797

- Grafana Labs. (2025). *Grafana documentation*. https://grafana.com/docs/
- Prometheus Authors. (2025). *Prometheus documentation*. https://prometheus.io/docs/
- Red Hat. (2023). *Alertmanager*. https://access.redhat.com/documentation/en-us/red_hat_openshift/4.12/html/monitoring/using-alertmanager-monitoring
- Turnbull, J. (2018). *The Prometheus monitoring system and time series database*. https://book.prometheus.io