# Informe de Implementación del Ambiente de Monitoreo con Grafana, Prometheus y Loki

Integrantes: Juan David Bahamon y Andrés Pino

## Ambiente configurado para gestionar logs, métricas y alertas de un sistema en ejecución (10%)

## 1. Preparación Inicial

En esta etapa se realizó la instalación y configuración de varias herramientas de monitoreo y visualización de métricas y logs:

- Grafana: Es una herramienta que permite la visualización de datos, y permite integrar y mostrar métricas de distintas fuentes.
- Prometheus (incluyendo Alert Manager y Node Explorer): Es un sistema de monitoreo y base de datos de series temporales, adecuado para obtener métricas de aplicaciones y servicios.
- Grafana Loki (junto con Grafana Agent y Alloy como agentes opcionales): Es un sistema de gestión de logs optimizado para operar en conjunto con Grafana.

## 2. Implementación de la Aplicación de Monitoreo con Spring Boot

Para crear una aplicación que genere métricas personalizadas y logs, se desarrolló una aplicación básica de Spring Boot con una dependencia de Micrometer para integrar métricas en Prometheus.

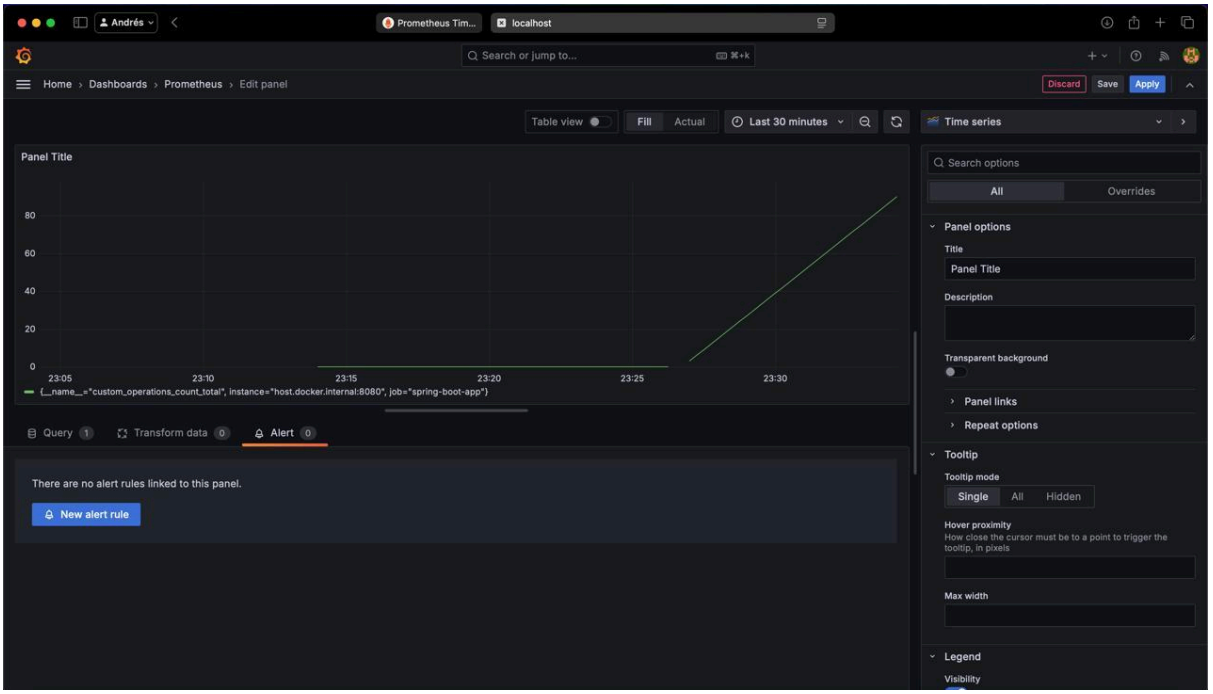
En esta se incluyó la dependencia de Micrometer en la aplicación para monitorear las operaciones simuladas en el código. También, se creó un componente (OperationSimulator) que ejecuta operaciones simuladas y cuenta las operaciones realizadas usando un Counter de Micrometer, el cual se registra en el MeterRegistry para que Prometheus pueda capturar la métrica.

Y por último se habilitaron los endpoints /actuator/prometheus, /actuator/info, y /actuator/health en archivos .yml, lo que permite que Prometheus acceda a las métricas de la aplicación.

# 3. Configuración de Docker y Docker Compose

Para facilitar la gestión y ejecución de todas las herramientas, se creó un archivo docker-compose.yml con los siguientes servicios:

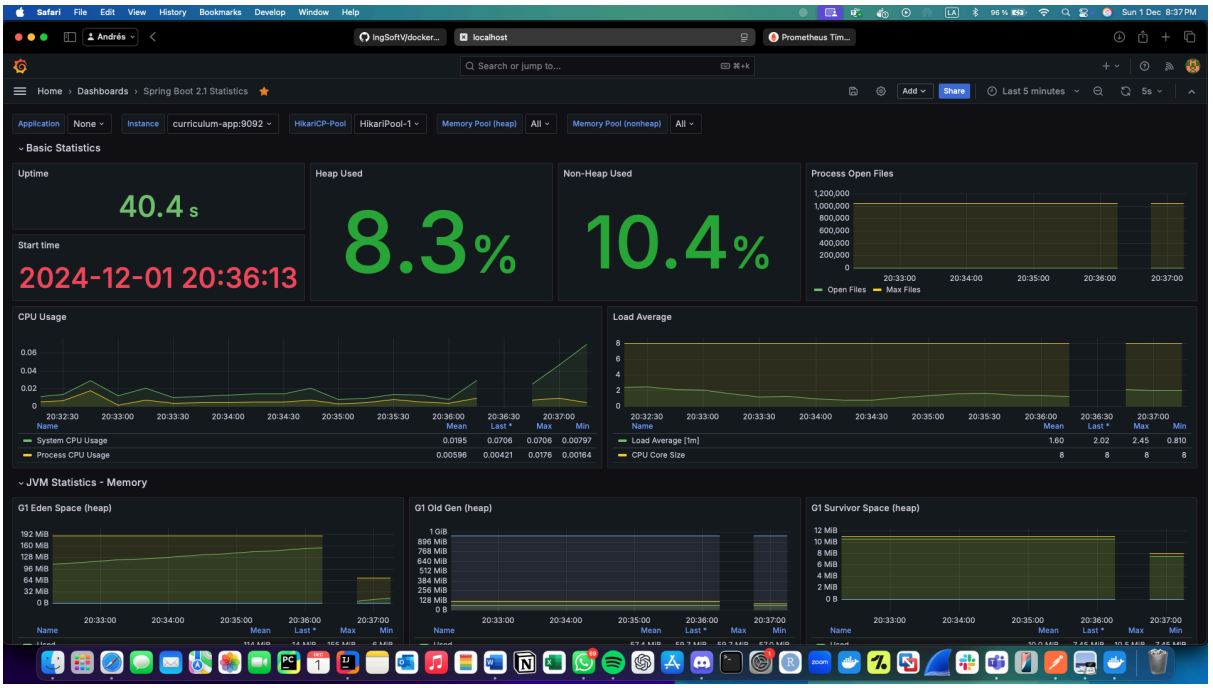
- Grafana: Configurada para conectarse a Prometheus y Loki para visualizar métricas y logs. Expuesta en el puerto 3000.
- Prometheus: Configurado para capturar las métricas de la aplicación de Spring Boot a través del endpoint /actuator/prometheus.
- Loki: Configurado para capturar los logs de la aplicación.
- Promtail: Configurado para recolectar logs locales y enviarlos a Loki.




Grafana: localhost:3000
Prometheus: localhost:9090
Loki: localhost:3100
AlertManager: localhost: 9093
Docker node explorer: localhost: 9100

# 4. Dashboards Predefinidos para Spring Boot:

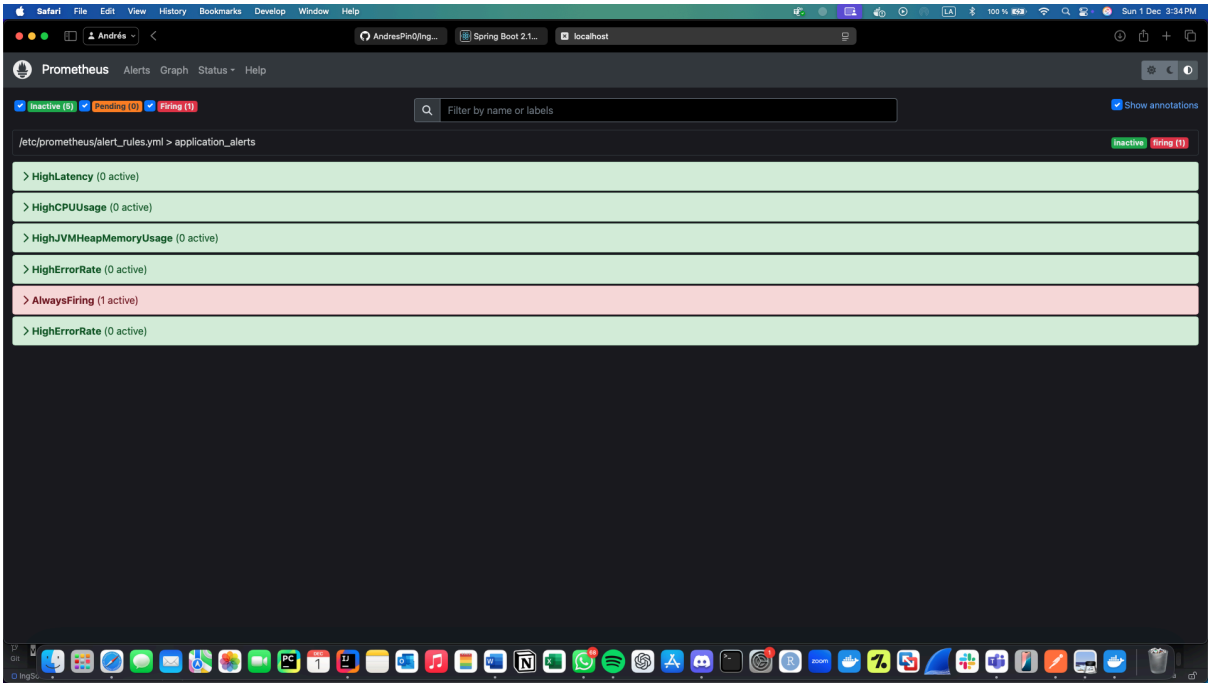
import via grafana.com: Dashboard con ID 10280




Para Node Exporter: ID: 1860

# 5. Alertas adicionales en Prometheus:

http://localhost:9090/alerts



Donde:

## HighLatency

**Esta métrica es un histograma que mide el tiempo de respuesta de las solicitudes HTTP en la aplicación Spring Boot.**

- **expr:** Calcula el percentil 95 de la latencia de las solicitudes HTTP en los últimos 5 minutos, agrupado por método y URI.
- **> 0.5:** La alerta se dispara si la latencia p95 supera los 0.5 segundos. Puedes ajustar este umbral según tus necesidades.
- **for:** La condición debe mantenerse durante 5 minutos para que la alerta se active.

**Ejemplo de uso:**

Simular una carga en la aplicación que genere alta latencia. Por ejemplo, crear un endpoint que realice operaciones intensivas.

## HighCPUUsage

- **expr:** Calcula el porcentaje de CPU utilizado en los últimos 1 minuto, restando el tiempo en modo idle.
- **> 80:** La alerta se dispara si el uso de CPU supera el 80%.

**Ejemplo de uso:**
Generar cargas en la aplicación para aumentar el consumo de CPU o memoria. Usando herramientas como `stress` o scripts que consuman recursos.

## HighJVMHeapMemoryUsage

- **expr:** Calcula el porcentaje de memoria heap de la JVM utilizada.
- **> 80:** La alerta se dispara si el uso de memoria supera el 80%.

## HighErrorRate

- **expr:** Calcula la tasa de respuestas HTTP con código 5xx (errores del servidor) en el último minuto.
- **> 0.05:** La alerta se dispara si más del 5% de las solicitudes resultan en errores 5xx.

**Ejemplo de uso:**
Provocando errores en la aplicación accediendo a rutas no definidas o generando excepciones.

AlwaysFiring

Aleta configurada para probar el alertmanager y su funcionamiento.

# Modificaciones de aplicación objetivo

Los servicios que se crearán ahora serán: docker-compose up –build

- `curriculum-app`: Aplicación de gestión curricular.
- `prometheus`: Sistema de monitoreo de métricas.
- `loki`: Sistema de recolección de logs.
- `promtail`: Agente que envía los logs a Loki.
- `grafana`: Plataforma de visualización de datos

```
andres@Andress-MacBook-Pro docker % docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                    NAMES
da751fa214ab   grafana/grafana:latest     "/run.sh"                26 seconds ago   Up 23 seconds   0.0.0.0:3000->3000/tcp   docker-grafana-1
aedc308bb646   docker-curriculum-app      "java -jar app.jar"      26 seconds ago   Up 23 seconds   0.0.0.0:9092->9092/tcp   docker-curriculum-app-1
0b21511de428   prom/prometheus:latest     "/bin/prometheus --c…"   26 seconds ago   Up 24 seconds   0.0.0.0:9090->9090/tcp   docker-prometheus-1
bc23909651ae   prom/node-exporter:latest  "/bin/node_exporter …"   26 seconds ago   Up 25 seconds   0.0.0.0:9100->9100/tcp   docker-node_exporter-1
5de881c47a36   grafana/loki:latest        "/usr/bin/loki -conf…"   26 seconds ago   Up 25 seconds   0.0.0.0:3100->3100/tcp   docker-loki-1
andres@Andress-MacBook-Pro docker %
```

## Verificación de Endpoints Expuestos

info, health y prometheus

http://localhost:9092/outcurrapi/actuator/info

```
{}
```

Se le puede agregar info en applications.propperties:

```
info.app.name=Curriculum Management System
info.app.description=Aplicación para gestionar currículos académicos
info.app.version=1.0.0
info.company.name=Universidad ICESI
info.company.department=Ingeniería de Software V
```

http://localhost:9092/outcurrapi/actuator/health


```
{"status":"UP","components":{"db":{"status":"UP","details":{"database":"H2","validationQuery":"isValid()"}},"diskSpace":
{"status":"UP","details":{"total":245107195904,"free":25272827904,"threshold":10485760,"path":"/tmp/05e6bb05-f1fe-48a8-baa2-
6bc74a4951fa/.","exists":true}},"ping":{"status":"UP"}}}
```

http://localhost:9092/outcurrapi/actuator/prometheus:

```
# HELP tomcat_sessions_rejected_sessions_total
# TYPE tomcat_sessions_rejected_sessions_total counter
tomcat_sessions_rejected_sessions_total 0.0
# HELP executor_active_threads The approximate number of threads that are actively executing tasks
# TYPE executor_active_threads gauge
executor_active_threads{name="applicationTaskExecutor",} 0.0
# HELP spring_security_filterchains_SaamfiAuthenticationFilter_after_total
# TYPE spring_security_filterchains_SaamfiAuthenticationFilter_after_total counter
spring_security_filterchains_SaamfiAuthenticationFilter_after_total{security_security_reached_filter_section="after",spring_security_filte
rchain_position="0",spring_security_filterchain_size="0",spring_security_reached_filter_name="none",} 3.0
# HELP tomcat_sessions_expired_sessions_total
# TYPE tomcat_sessions_expired_sessions_total counter
tomcat_sessions_expired_sessions_total 0.0
# HELP spring_security_filterchains_SaamfiAuthenticationFilter_before_total
# TYPE spring_security_filterchains_SaamfiAuthenticationFilter_before_total counter
spring_security_filterchains_SaamfiAuthenticationFilter_before_total{security_security_reached_filter_section="before",spring_security_fil
terchain_position="0",spring_security_filterchain_size="0",spring_security_reached_filter_name="none",} 4.0
# HELP jvm_compilation_time_ms_total The approximate accumulated elapsed time spent in compilation
# TYPE jvm_compilation_time_ms_total counter
jvm_compilation_time_ms_total{compiler="HotSpot 64-Bit Tiered Compilers",} 1939.0
# HELP jdbc_connections_max Maximum number of active connections that can be allocated at the same time.
# TYPE jdbc_connections_max gauge
jdbc_connections_max{name="dataSource",} 10.0
# HELP executor_completed_tasks_total The approximate total number of tasks that have completed execution
# TYPE executor_completed_tasks_total counter
executor_completed_tasks_total{name="applicationTaskExecutor",} 0.0
# HELP spring_security_http_secured_requests_active_seconds
# TYPE spring_security_http_secured_requests_active_seconds summary
spring_security_http_secured_requests_active_seconds_active_count 1.0
spring_security_http_secured_requests_active_seconds_duration_sum 0.00719425
# HELP spring_security_http_secured_requests_active_seconds_max
# TYPE spring_security_http_secured_requests_active_seconds_max gauge
spring_security_http_secured_requests_active_seconds_max 0.007218125
# HELP hikaricp_connections_creation_seconds_max Connection creation time
# TYPE hikaricp_connections_creation_seconds_max gauge
hikaricp_connections_creation_seconds_max{pool="HikariPool-1",} 0.0
# HELP hikaricp_connections_creation_seconds Connection creation time
# TYPE hikaricp_connections_creation_seconds summary
hikaricp_connections_creation_seconds_count{pool="HikariPool-1",} 0.0
hikaricp_connections_creation_seconds_sum{pool="HikariPool-1",} 0.0
# HELP jdbc_connections_min Minimum number of idle connections in the pool.
# TYPE jdbc_connections_min gauge
jdbc_connections_min{name="dataSource",} 10.0
# HELP jvm_gc_pause_seconds Time spent in GC pause
# TYPE jvm_gc_pause_seconds summary
jvm_gc_pause_seconds_count{action="end of minor GC",cause="G1 Humongous Allocation",gc="G1 Young Generation",} 1.0
jvm_gc_pause_seconds_sum{action="end of minor GC",cause="G1 Humongous Allocation",gc="G1 Young Generation",} 0.021
# HELP jvm_gc_pause_seconds_max Time spent in GC pause
# TYPE jvm_gc_pause_seconds_max gauge
jvm_gc_pause_seconds_max{action="end of minor GC",cause="G1 Humongous Allocation",gc="G1 Young Generation",} 0.021
# HELP spring_security_filterchains_authorization_before_total
# TYPE spring_security_filterchains_authorization_before_total counter
spring_security_filterchains_authorization_before_total{security_security_reached_filter_section="before",spring_security_filterchain_posi
tion="0",spring_security_filterchain_size="0",spring_security_reached_filter_name="none",} 4.0
# HELP spring_security_filterchains_session_url_encoding_after_total
# TYPE spring_security_filterchains_session_url_encoding_after_total counter
spring_security_filterchains_session_url_encoding_after_total{security_security_reached_filter_section="after",spring_security_filterchain
_position="0",spring_security_filterchain_size="0",spring_security_reached_filter_name="none",} 3.0
# HELP jvm_memory_usage_after_gc_percent The percentage of long-lived heap pool used after the last GC event, in the range [0..1]
# TYPE jvm_memory_usage_after_gc_percent gauge
```

Prueba de las métricas implementadas

# Rutas (Endpoints) de Cada Servicio Modificado

### a. Servicio `CourseService`

http:localhost:9092/v1/acadprog/{acadProgId}/faculty/{facultyId}/acadprogcurr/{acadProgCurrId}/courses

- **Endpoint:** `GET /v1/acadprog/{acadProgId}/faculty/{facultyId}/acadprogcurr/{acadProgCurrId}/courses`
- **Descripción:** Obtiene todos los cursos asociados a un programa académico, facultad y currículo académico específicos.

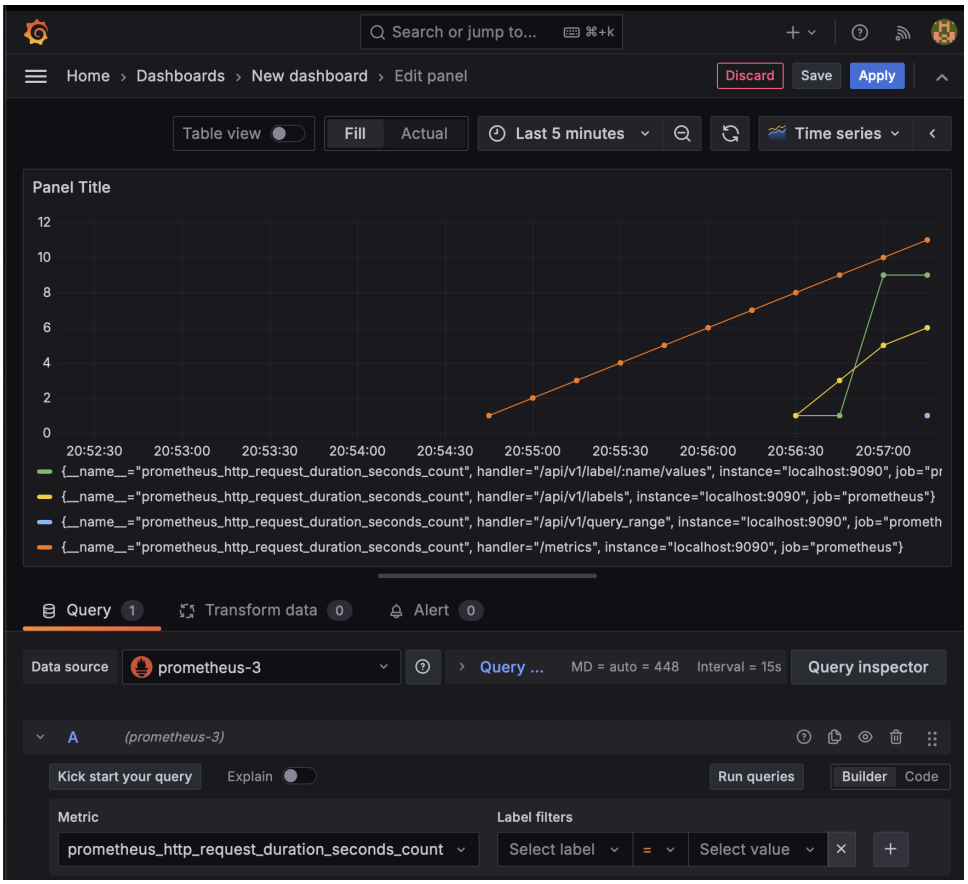
### b. Servicio `FacultyService`

- **Crear Facultad:**
  - **Endpoint:** `POST /v1/faculties`
  - **Descripción:** Crea una nueva facultad.
- **Obtener Facultad por ID:**
  - **Endpoint:** `GET /v1/faculties/{facId}`
  - **Descripción:** Obtiene los detalles de una facultad por su ID.
- **Actualizar Facultad:**
  - **Endpoint:** `PUT /v1/faculties/{facId}`
  - **Descripción:** Actualiza los datos de una facultad existente.
- **Eliminar Facultad:**
  - **Endpoint:** `DELETE /v1/faculties/{facId}`
  - **Descripción:** Elimina una facultad si no tiene asociaciones.
- **Listar Facultades:**
  - **Endpoint:** `GET /v1/faculties`
  - **Descripción:** Obtiene una lista de todas las facultades.

Despliegue y enlace de servicios 10%
Despliegue la aplicación de gestión curricular para utilizar las métricas en prometheus y logs con Grafana Alloy

```
andres@Andress-MacBook-Pro docker % docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED              STATUS              PORTS                    NAMES
38b9c6f707b1   grafana/grafana:latest     "/run.sh"                About a minute ago   Up About a minute   0.0.0.0:3000->3000/tcp   docker-grafana-1
2f11f2a15325   grafana/loki:latest        "/usr/bin/loki -conf…"   About a minute ago   Up About a minute   0.0.0.0:3100->3100/tcp   docker-loki-1
19f9808eab18   docker-curriculum-app      "java -jar app.jar"      About a minute ago   Up About a minute   0.0.0.0:9092->9092/tcp   docker-curriculum-app-1
ee862b7daf48   prom/prometheus:latest     "/bin/prometheus --c…"   17 minutes ago       Up 17 minutes       0.0.0.0:9090->9090/tcp   docker-prometheus-1
bd6f6c8f7cf3   prom/node-exporter:latest  "/bin/node_exporter …"   17 minutes ago       Up 17 minutes       0.0.0.0:9100->9100/tcp   docker-node_exporter-1
662576661cf4   prom/alertmanager:latest   "/bin/alertmanager -…"   17 minutes ago       Up 17 minutes       0.0.0.0:9093->9093/tcp   docker-alertmanager-1
9f9c5f4fec85   grafana/promtail:latest    "/usr/bin/promtail -…"   17 minutes ago       Up 17 minutes                                docker-promtail-1
```

Verificar métricas:

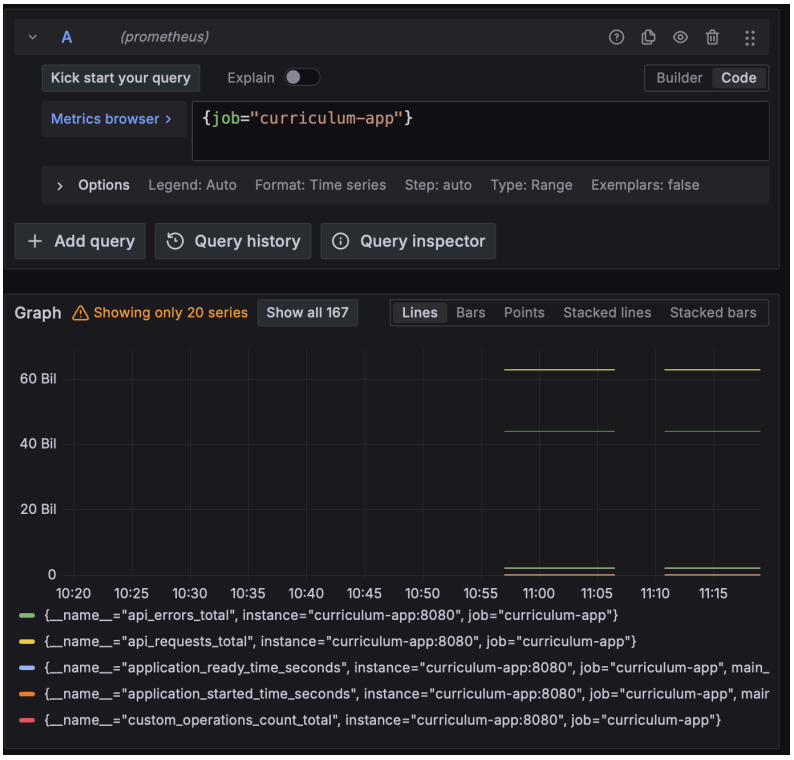


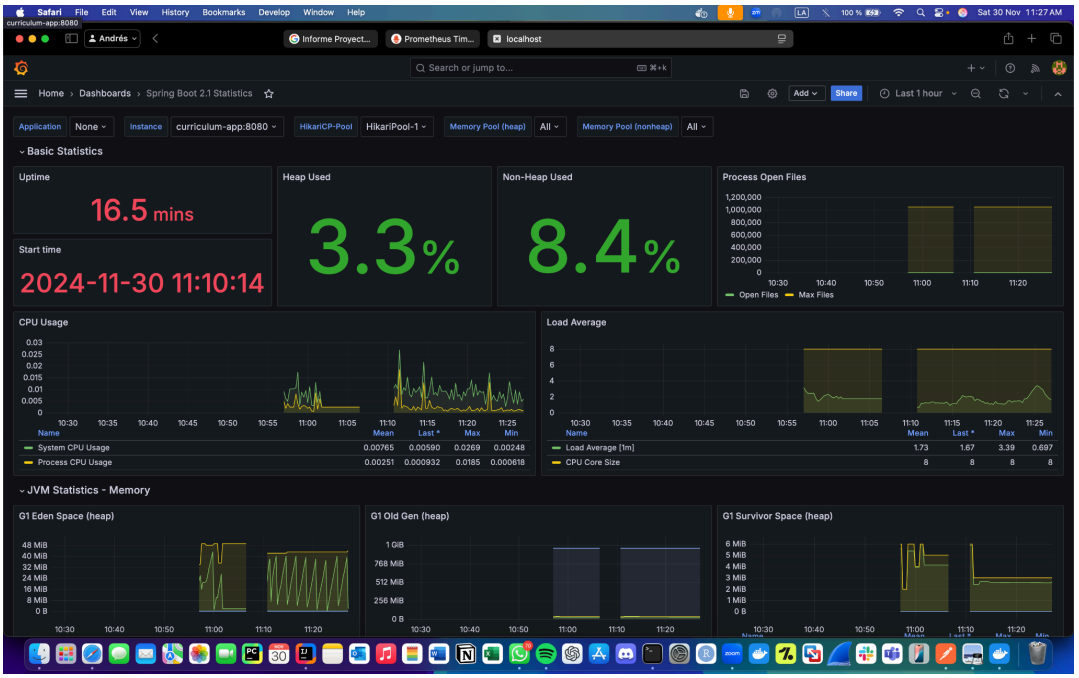
# Prueba de Logs estructurados

Esto está en el archivo application.log, hasta ahora:
https://github.com/AndresPin0/Project_IngSoft_V/blob/master/demo/logs/application.log

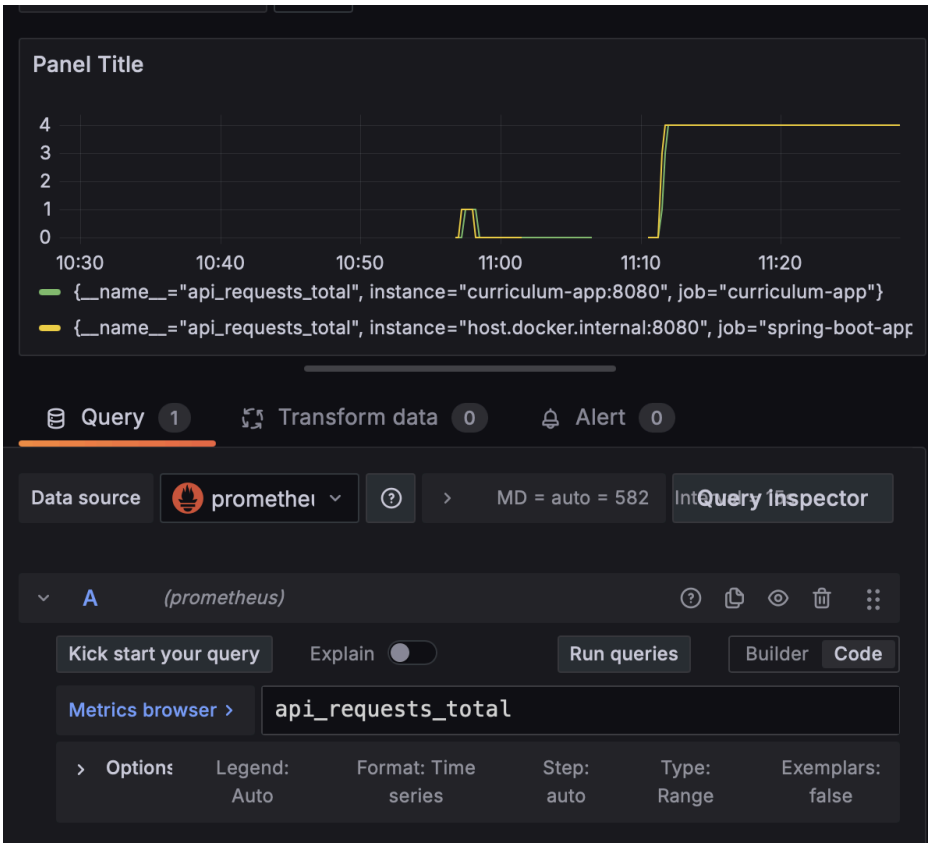
Ahí se puede verificar los logs de operaciones exitosas y con errores.

# Verificar logs con grafana http://localhost:3000

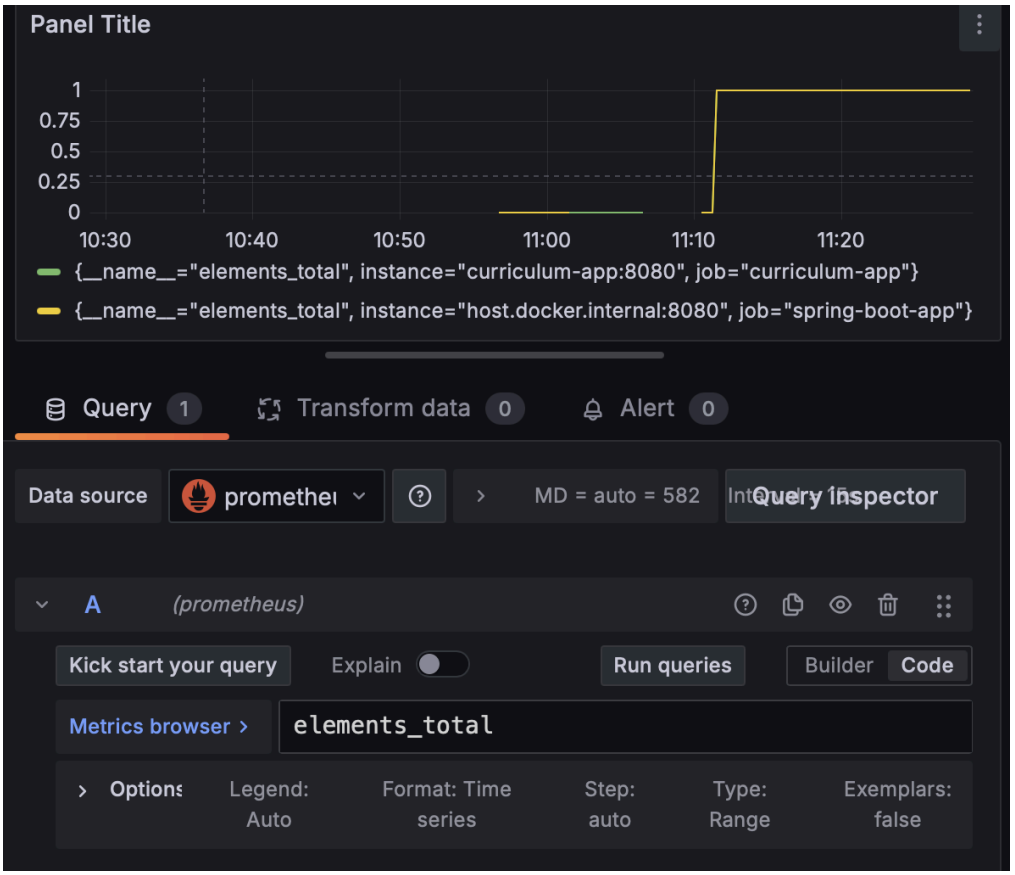


api_requests_total




Elements_total



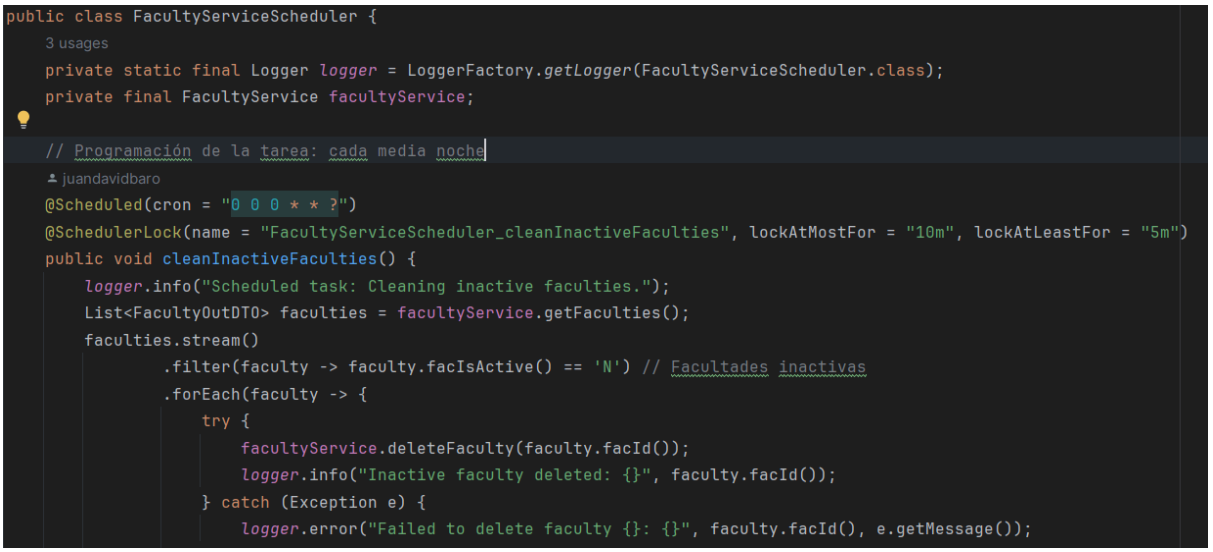
# Pruebas incluyendo “schedulers” para que las funcionalidades se llamen periódicamente:

## FacultyServiceScheduler

Este código define un servicio programado en Spring que **limpia automáticamente las facultades inactivas de la base de datos** todos los días a la medianoche. Utiliza la anotación @Scheduled para configurar la periodicidad de la ejecución, y SLF4J para registrar los eventos ocurridos durante la ejecución de la tarea.

Agregamos este decorador:
@SchedulerLock(name = "FacultyServiceScheduler_cleanInactiveFaculties", lockAtMostFor = "10m", lockAtLeastFor = "5m")

Para que la tarea cleanInactiveFaculties se ejecute solo una vez a la vez.
El bloqueo persiste al menos 5 minutos (por seguridad) y no más de 10 minutos (en caso de fallos).



```
public class FacultyServiceScheduler {
    3 usages
    private static final Logger logger = LoggerFactory.getLogger(FacultyServiceScheduler.class);
    private final FacultyService facultyService;

    // Programación de la tarea: cada media noche
    juandavidbaro
    @Scheduled(cron = "0 0 0 * * ?")
    @SchedulerLock(name = "FacultyServiceScheduler_cleanInactiveFaculties", lockAtMostFor = "10m", lockAtLeastFor = "5m")
    public void cleanInactiveFaculties() {
        logger.info("Scheduled task: Cleaning inactive faculties.");
        List<FacultyOutDTO> faculties = facultyService.getFaculties();
        faculties.stream()
                .filter(faculty -> faculty.facIsActive() == 'N') // Facultades inactivas
                .forEach(faculty -> {
                    try {
                        facultyService.deleteFaculty(faculty.facId());
                        logger.info("Inactive faculty deleted: {}", faculty.facId());
                    } catch (Exception e) {
                        logger.error("Failed to delete faculty {}: {}", faculty.facId(), e.getMessage());
```

La configuración usa **ShedLock** para evitar duplicación de tareas programadas en múltiples instancias, garantizando que solo una las ejecute mediante bloqueos en la base de datos.

```java
@Configuration
@EnableScheduling
public class SchedulerConfig {
    juandavidbaro
    @Bean
    public LockProvider lockProvider(DataSource dataSource) {
        return new JdbcTemplateLockProvider(
                JdbcTemplateLockProvider.Configuration.builder()
                        .withJdbcTemplate(new JdbcTemplate(dataSource))
                        .build()
        );
    }
}
```

```java
void testCleanInactiveFaculties() {
    // Crear facultades simuladas
    FacultyOutDTO activeFaculty = FacultyOutDTO.builder()
            .facId(1L)
            .facIsActive('Y')
            .facNameEng("Engineering Faculty")
            .facNameSpa("Facultad de Ingeniería")
            .acadPrograms(List.of())
            .build();

    FacultyOutDTO inactiveFaculty = FacultyOutDTO.builder()
            .facId(2L)
            .facIsActive('N')
            .facNameEng("Arts Faculty")
            .facNameSpa("Facultad de Artes")
            .acadPrograms(List.of())
            .build();

    // Configurar el mock para devolver las facultades simuladas
    Mockito.when(facultyService.getFaculties()).thenReturn(List.of(activeFaculty, inactiveFaculty));
    // Ejecutar el scheduler
    scheduler.cleanInactiveFaculties();
    // Verificar que solo la facultad inactiva sea eliminada
    Mockito.verify(facultyService, Mockito.times( wantedNumberOfInvocations: 1)).deleteFaculty( facId: 2L);
    Mockito.verify(facultyService, Mockito.never()).deleteFaculty( facId: 1L);
```

En este test se asegura que el método cleanInactiveFaculties() del FacultyServiceScheduler:

- Elimine correctamente solo las facultades inactivas.
- No elimina las facultades activas.
- Que funcione correctamente con el bloqueo de ejecución, evitando que se ejecuten múltiples instancias de la tarea programada simultáneamente.

Pipeline para clonar el repositorio, construir el proyecto, ejecutar las pruebas unitarias en una máquina con Windows y desplegar el docker.

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            agent { label 'build-node' }
            steps {
                git url: 'https://github.com/AndresPin0/IngSoftV.git', branch: 'master'
            }
        }
        stage('Build') {
            agent { label 'build-node' }
            steps {
                script {
                    bat "mvn clean install -DskipTests"
                }
            }
        }
        stage('Unit Test') {
            agent { label 'build-node' }
            steps {
                script {
                    bat "mvn test"
                }
            }
        }
        stage('Start Docker Containers') {
            agent { label 'build-node' }
            steps {
                script {
                    // Cambiar al directorio docker y ejecutar docker-compose
                    bat "cd docker && docker-compose up -d"
                }
            }
        }
    }
}
```

10% Teniendo en cuenta la estrategia de despliegue mencionada y los resultados vistos en las métricas, proponga las modificaciones necesarias para el despliegue, la arquitectura y el código que se podrían implementar para mejorar requerimientos no funcionales del sistema de software.

### 1. Optimización del Rendimiento

- **Caching de Consultas:** Si las métricas muestran latencia en las respuestas debido a consultas repetidas a la base de datos, implementar un sistema de caching puede reducir los tiempos de respuesta y disminuir la carga sobre la base de datos.
- **Balanceo de Carga:** Distribuir las solicitudes entrantes entre múltiples instancias del servicio puede mejorar el rendimiento y la disponibilidad. Usa un balanceador de carga para distribuir el tráfico entre varias instancias de tu aplicación.

### 2. Escalabilidad

- **Autoescalado:** Basado en las métricas de uso de CPU y memoria, implementar soluciones de autoescalado para los contenedores puede ayudar a manejar mejor los picos de demanda.

### 3. Seguridad

- **Auditorías de Seguridad:** Realizar auditorías de seguridad regulares y actualizar las dependencias y componentes a sus versiones más seguras.
- **Mejoras en la autenticación y Autorización:** Reforzar los mecanismos de control de acceso, especialmente si detectas intentos de accesos no autorizados en las métricas.

### 4. Despliegue Continuo y Automatización

- **Integración y Despliegue Continuos (CI/CD):** Automatizar los procesos de integración y despliegue para asegurar despliegues suaves y consistentes. Esto también facilita la implementación rápida de cambios y mejoras.

### 5. Despliegue Continuo y Automatización

- **Integración y Despliegue Continuos (CI/CD):** Automatiza los procesos de integración y despliegue para asegurar despliegues suaves y consistentes. Esto también facilita la implementación rápida de cambios y mejoras.