Sistemas Distribuidos

Mini Plataforma de Video Streaming P2P con Microservicios

Autores:

Aaron Rodrigo Ramos Reyes

Oscar Martinez Barrales

Oswaldo Mejia Garcia

Universidad Autonoma Metropolitana

Fecha: 13 de agosto de 2025

Introducción

Estructura del Proyecto

El proyecto está organizado en dos microservicios principales:

- **centralservice** - Servicio de registro y coordinación
- **p2pnodo** - Nodos P2P para almacenamiento y distribución

Servicio Central

Modelos de Datos

NodeRegistration.java

```
public class NodeRegistration {
    private String nodeId;
    private String nodeUrl;
    private List<String> fragments;
    // getters y setters
}
```

FragmentEvent.java

```
public class FragmentEvent {
    private String fragmentId;
    private String nodeUrl;
    private Long timestamp;

    public boolean isValid() {
        return fragmentId != null && !fragmentId.isEmpty() &&
               nodeUrl != null && !nodeUrl.isEmpty();
    }
}
```

Controladores REST

El servicio central expone endpoints REST para:

- Registro de nodos
- Consulta de nodos disponibles

- Localización de fragmentos específicos
- Actualización de inventarios de fragmentos

Nodos P2P

Auto-registro

Cada nodo se registra automáticamente al iniciar usando `@PostConstruct`:

```
@PostConstruct
public void registerOnStartup() {
    NodeRegistration registration = new NodeRegistration();
    registration.setNodeId(System.getenv("HOSTNAME"));
    registration.setNodeUrl(NODE_URL);
    registration.setFragments(List.of());

    restTemplate.postForObject(
        CENTRAL_SERVICE_URL + "/api/register",
        registration,
        String.class
    );
}
```

Cliente de Comunicación P2P

La clase `NodeClient` maneja la comunicación entre nodos:

- **downloadFragment()** - Descarga fragmentos de otros nodos
- **uploadFragment()** - Envía fragmentos usando multipart/form-data

Gestión de Fragmentos

Los fragmentos se almacenan como archivos binarios y se transfieren usando:

- HTTP multipart para subida de archivos
- Streaming de bytes para descarga
- Validación de integridad mediante IDs únicos

Configuración con Docker

El sistema utiliza Docker Compose para orquestación:

- **Redis** - Base de datos y Pub/Sub
- **Central Service** - Un contenedor
- **P2P Nodes** - Múltiples contenedores escalables

Comando de despliegue:

```
docker compose up -d --scale p2pnodo=3
```

Manejo de Errores y Logging

El sistema implementa:

- Manejo de excepciones en comunicaciones HTTP
- Validación de datos de entrada
- Timeouts configurables para operaciones de red

Arquitectura del Sistema

La arquitectura del sistema se basa en un modelo híbrido que combina elementos centralizados y distribuidos para optimizar la distribución de contenido de video.

Componentes Principales

Servicio Central (Registry Service)

El servicio central actúa como un registro y coordinador del sistema, implementado como un microservicio Spring Boot que:

- Mantiene un registro de todos los nodos activos en la red
- Almacena información sobre qué fragmentos posee cada nodo

- Proporciona APIs REST para consultas de ubicación de fragmentos
- Gestiona el sistema de notificaciones Pub/Sub con Redis

Endpoints principales:

- `POST /api/register` - Registro de nuevos nodos
- `GET /api/nodes` - Lista de nodos registrados
- `GET /api/fragment/{fragmentId}` - Localización de fragmentos
- `POST /api/updateFragments` - Actualización de inventario de fragmentos

Nodos P2P

Cada nodo P2P es un microservicio independiente que:

- Almacena fragmentos de video localmente
- Sirve fragmentos a otros nodos mediante HTTP
- Se registra automáticamente en el servicio central al iniciar
- Escucha notificaciones de nuevos fragmentos disponibles
- Implementa lógica de descarga automática de fragmentos faltantes

Endpoints principales:

- `GET /fragment/{id}` - Descarga de fragmentos
- `POST /fragment/receive` - Recepción de fragmentos

Sistema Pub/Sub con Redis

Redis actúa como broker de mensajes para:

- Notificar a todos los nodos cuando hay nuevos fragmentos disponibles
- Mantener la sincronización del estado del sistema
- Reducir la carga en el servicio central mediante notificaciones asíncronas

Flujo de Comunicación

1. Los nodos se registran en el servicio central al iniciar
2. Cuando un nodo recibe un nuevo fragmento, notifica al servicio central
3. El servicio central publica un evento en Redis
4. Otros nodos reciben la notificación y pueden solicitar el fragmento
5. La transferencia de fragmentos ocurre directamente entre nodos (P2P)

Escalabilidad y Tolerancia a Fallos

El sistema está diseñado para escalar horizontalmente:

- Nuevos nodos pueden unirse dinámicamente
- La carga se distribuye automáticamente entre nodos
- Si un nodo falla, otros nodos pueden tener copias de sus fragmentos
- El servicio central puede replicarse para alta disponibilidad

Implementación

Estructura del Proyecto

El proyecto está organizado en dos microservicios principales:

- **centralservice** - Servicio de registro y coordinación
- **p2pnodo** - Nodos P2P para almacenamiento y distribución

Servicio Central

Modelos de Datos

NodeRegistration.java

```
public class NodeRegistration {
    private String nodeId;
    private String nodeUrl;
    private List<String> fragments;
    // getters y setters
}
```

FragmentEvent.java

```
public class FragmentEvent {
    private String fragmentId;
    private String nodeUrl;
    private Long timestamp;

    public boolean isValid() {
        return fragmentId != null && !fragmentId.isEmpty() &&
               nodeUrl != null && !nodeUrl.isEmpty();
    }
}
```

Controladores REST

El servicio central expone endpoints REST para:

- Registro de nodos
- Consulta de nodos disponibles
- Localización de fragmentos específicos
- Actualización de inventarios de fragmentos

Nodos P2P

Auto-registro

Cada nodo se registra automáticamente al iniciar usando `@PostConstruct`:

```
@PostConstruct
public void registerOnStartup() {
    NodeRegistration registration = new NodeRegistration();
    registration.setNodeId(System.getenv("HOSTNAME"));
    registration.setNodeUrl(NODE_URL);
    registration.setFragments(List.of());

    restTemplate.postForObject(
        CENTRAL_SERVICE_URL + "/api/register",
        registration,
        String.class
    );
}
```

Cliente de Comunicación P2P

La clase `NodeClient` maneja la comunicación entre nodos:

- **downloadFragment()** - Descarga fragmentos de otros nodos
- **uploadFragment()** - Envía fragmentos usando multipart/form-data

Gestión de Fragmentos

Los fragmentos se almacenan como archivos binarios y se transfieren usando:

- HTTP multipart para subida de archivos
- Streaming de bytes para descarga
- Validación de integridad mediante IDs únicos

Configuración con Docker

El sistema utiliza Docker Compose para orquestación:

- **Redis** - Base de datos y Pub/Sub
- **Central Service** - Un contenedor
- **P2P Nodes** - Múltiples contenedores escalables

Comando de despliegue:

```
docker compose up -d --scale p2pnodo=3
```

Manejo de Errores y Logging

El sistema implementa:

- Manejo de excepciones en comunicaciones HTTP
- Validación de datos de entrada
- Timeouts configurables para operaciones de red

Tecnologías Utilizadas

Backend

Java

Lenguaje de programación principal del proyecto, aprovechando las características más recientes como:

- Records para DTOs inmutables
- Mejor rendimiento y gestión de memoria

Spring Boot

Framework principal para el desarrollo de microservicios:

- **Spring Web** - APIs REST
- **Spring Data Redis** - Integración con Redis
- **Spring Boot Actuator** - Monitoreo y métricas

Maven

Herramienta de gestión de dependencias y construcción:

- Gestión automática de dependencias
- Perfiles de construcción para diferentes entornos
- Plugins para empaquetado Docker

Base de Datos y Mensajería

Redis

Sistema de almacenamiento en memoria que proporciona:

- **Pub/Sub** - Sistema de mensajería para notificaciones
- **Cache** - Almacenamiento rápido de metadatos
- **Persistencia** - Respaldo de datos críticos

Herramientas de Desarrollo

Swagger

Documentación automática de APIs:

- Generación automática de documentación
- Interfaz web interactiva para pruebas

Jackson

Serialización/deserialización JSON:

- Conversión automática de objetos Java a JSON
- Configuración flexible de mapeo
- Soporte para tipos complejos

Containerización y Orquestación

Docker

Containerización de aplicaciones:

- Imágenes ligeras basadas en OpenJDK
- Configuración mediante variables de entorno
- Aislamiento de dependencias

Docker Compose

Orquestación de múltiples servicios:

- Definición declarativa de servicios
- Escalado horizontal automático
- Gestión de redes y volúmenes

Calendario de Actividades

Cronograma del Proyecto

El proyecto se desarrolló en un período intensivo de 5 días (4-8 de agosto de 2025), con una distribución equitativa de responsabilidades entre los tres integrantes del equipo.

Distribución de Actividades por Día

Lunes 4 de Agosto - Planificación y Diseño

Aaron Rodrigo Ramos Reyes:

- Análisis de requisitos del sistema P2P
- Diseño de la arquitectura general del sistema
- Definición de APIs REST para el servicio central

- Creación de diagramas de arquitectura (PlantUML)

Oscar Martinez Barrales:

- Investigación de tecnologías P2P y microservicios
- Diseño de la estructura de datos para fragmentos
- Definición de protocolos de comunicación entre nodos
- Configuración inicial del entorno de desarrollo

Oswaldo Mejia Garcia:

- Análisis de patrones de distribución de contenido
- Diseño del sistema Pub/Sub con Redis
- Configuración de Docker y Docker Compose

Martes 5 de Agosto - Desarrollo del Servicio Central

Aaron Rodrigo Ramos Reyes:

- Implementación del servicio central (Spring Boot)
- Desarrollo de controladores REST
- Configuración de Spring Data Redis
- Implementación de DTOs (NodeRegistration, FragmentEvent)

Oscar Martinez Barrales:

- Desarrollo de la lógica de registro de nodos
- Implementación del sistema de localización de fragmentos
- Configuración de validaciones con Hibernate Validator
- Desarrollo de servicios de negocio

Oswaldo Mejia Garcia:

- Configuración de Redis Pub/Sub
- Implementación de listeners de eventos
- Desarrollo de la lógica de notificaciones

Miércoles 6 de Agosto - Desarrollo de Nodos P2P

Aaron Rodrigo Ramos Reyes:

- Implementación de la aplicación P2P Node
- Desarrollo de controladores para fragmentos
- Implementación de auto-registro en el servicio central

Oscar Martinez Barrales:

- Desarrollo de NodeClient para comunicación P2P
- Implementación de descarga de fragmentos
- Configuración de RestTemplate para comunicación HTTP

Oswaldo Mejia Garcia:

- Implementación de almacenamiento local de fragmentos
- Desarrollo de la lógica de sincronización
- Configuración de variables de entorno

Jueves 7 de Agosto - Integración y Testing

Aaron Rodrigo Ramos Reyes:

- Integración completa del sistema
- Configuración de Docker Compose

Oscar Martinez Barrales:

- Testing de APIs con Postman
- Validación de transferencia de fragmentos

Oswaldo Mejia Garcia:

- Validación de notificaciones en tiempo real
- Testing de tolerancia a fallos

Viernes 8 de Agosto - Documentación y Entrega

Aaron Rodrigo Ramos Reyes:

- Redacción de documentación técnica
- Creación de instructivos de instalación
- Preparación de la presentación final
- Revisión final del código

Oscar Martinez Barrales:

- Documentación de APIs con OpenAPI/Swagger
- Creación de guías de uso con Postman
- Documentación de arquitectura
- Testing final del sistema completo

Oswaldo Mejia Garcia:

- Documentación de despliegue con Docker
- Creación de scripts de automatización
- Documentación de configuración
- Preparación de entregables finales

Metodología de Trabajo

- **Control de versiones:** Git con ramas por funcionalidad
- **Comunicación:** Discord para coordinación continua
- **Testing continuo:** Validación diaria de integraciones

Conclusiones

El proyecto de Mini Plataforma de Video Streaming P2P con Microservicios ha cumplido exitosamente con todos los objetivos planteados:

- **Arquitectura Distribuida:** Se implementó un sistema híbrido que combina un servicio central de registro con comunicación P2P directa entre nodos
- **Escalabilidad:** El sistema permite agregar nodos dinámicamente usando Docker Compose con escalado horizontal
- **Comunicación Eficiente:** Se estableció un sistema de notificaciones en tiempo real usando Redis Pub/Sub
- **Transferencia de Fragmentos:** Los nodos pueden intercambiar fragmentos de video de manera directa y eficiente

Beneficios del Enfoque P2P

La implementación P2P demostró ventajas significativas sobre modelos centralizados tradicionales:

- **Reducción de Carga Central:** El servidor central solo maneja registro y coordinación, no transferencia de datos
- **Tolerancia a Fallos:** Los fragmentos pueden estar replicados en múltiples nodos
- **Escalabilidad Natural:** Más nodos significan más capacidad total del sistema

Microservicios con Spring Boot

La arquitectura de microservicios permitió:

- Desarrollo independiente de componentes
- Mantenimiento simplificado

Containerización con Docker

El uso de Docker facilitó:

- Despliegue consistente en diferentes entornos
- Escalado automático de nodos
- Aislamiento de dependencias
- Configuración simplificada

Sistema Pub/Sub

Redis Pub/Sub proporcionó:

- Notificaciones en tiempo real
- Desacoplamiento entre componentes
- Sincronización de estado distribuido

Desafíos Superados

Durante el desarrollo se enfrentaron y resolvieron varios desafíos técnicos:

- **Sincronización de Estado:** Implementación de un sistema robusto de notificaciones para mantener consistencia
- **Transferencia de Archivos:** Desarrollo de un mecanismo eficiente para transferir fragmentos entre nodos
- **Auto-registro:** Configuración automática de nodos al iniciar usando variables de entorno
- **Manejo de Errores:** Implementación de tolerancia a fallos en comunicaciones de red

Este proyecto proporcionó experiencia práctica en:

- Desarrollo de sistemas distribuidos
- Arquitectura de microservicios
- Comunicación P2P
- Containerización y orquestación
- Trabajo colaborativo en equipo
- Metodologías ágiles de desarrollo