Microservicio Beats Upload

> **Nota**: Todas las referencias a archivos de código incluyen hipervínculos directos al repositorio de GitHub para facilitar su verificación.

Autores: Daniel Vela Camacho y Miguel Encina Martínez
Nivel objetivo: 10 puntos
Repositorio: github.com/SocialBeats/beats-upload
Docker Hub: hub.docker.com/r/socialbeats/beats-upload

---

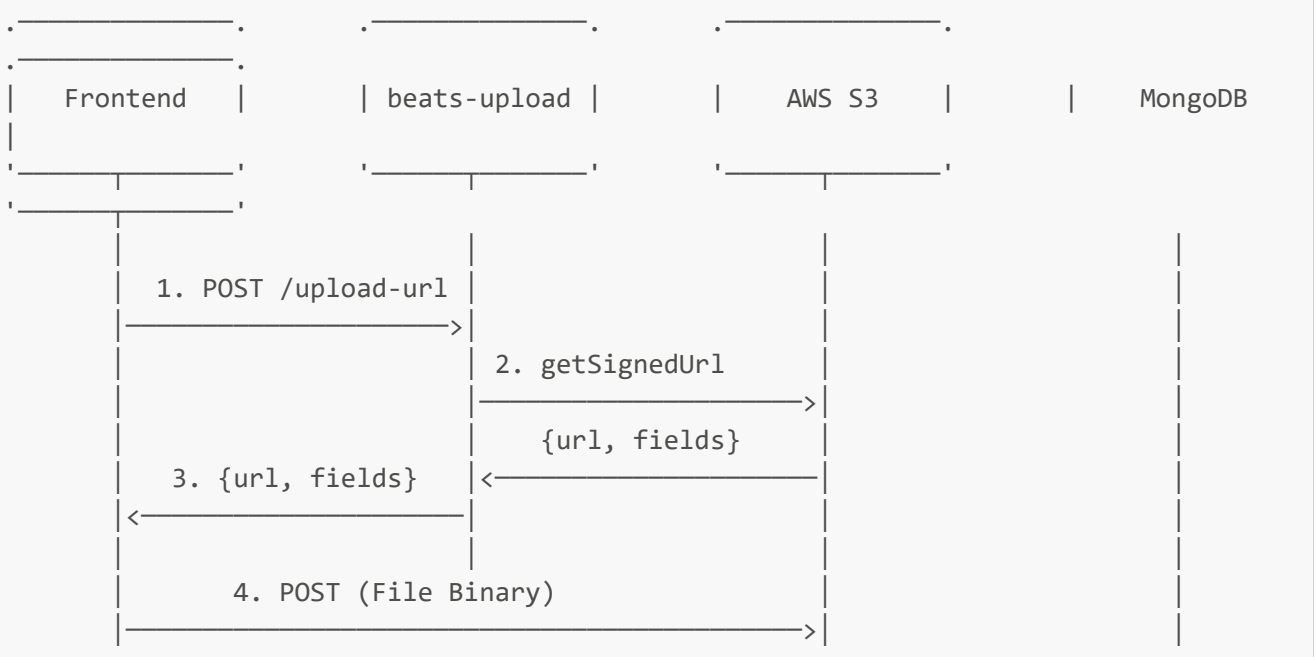
1. Resumen

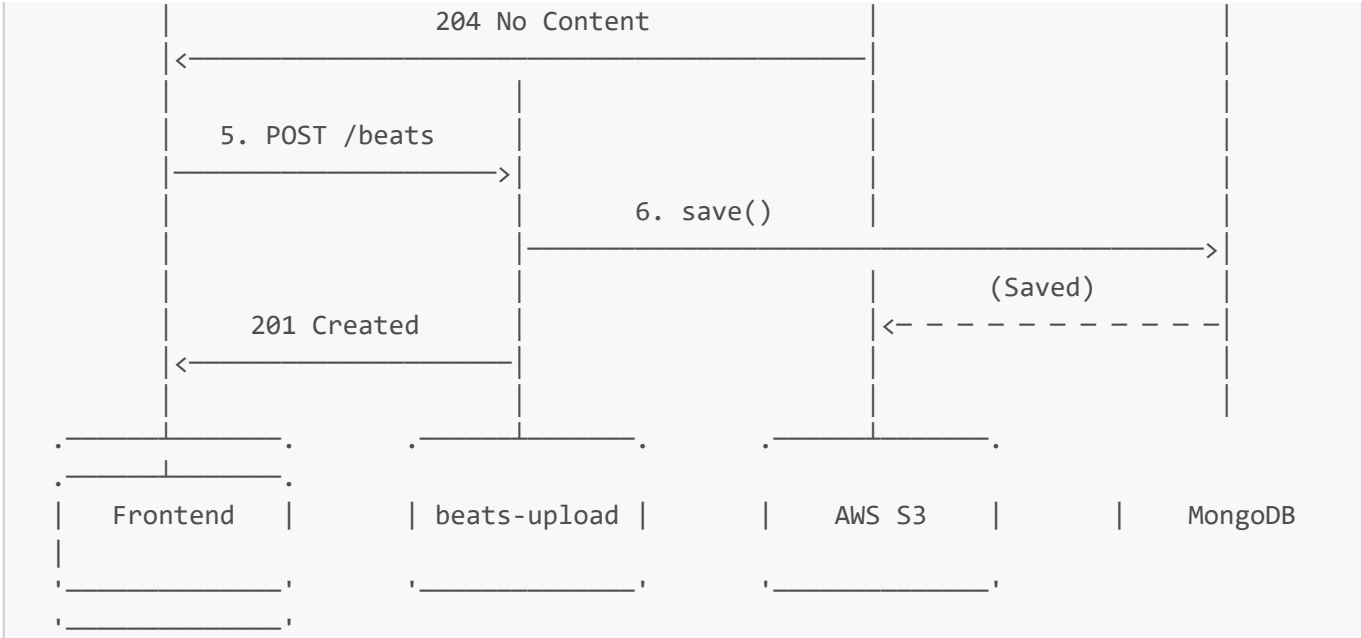
Beats-upload es el microservicio responsable de la **subida, almacenamiento y gestión del ciclo de vida de los beats** (instrumentales) en la plataforma SocialBeats.

Responsabilidades

Función	Descripción
Gestión de Beats	API REST completa (CRUD) para subir, listar, buscar, editar y eliminar beats
Almacenamiento Cloud	Integración con **AWS S3** para archivos de audio (MP3, WAV, FLAC, AAC) e imágenes
CDN	**CloudFront** con URLs firmadas para streaming optimizado y baja latencia
Event-Driven	Productor y consumidor **Kafka** para sincronización con otros microservicios
Pricing	Validación de límites del plan de usuario vía **SPACE**

Flujo principal





1. Usuario solicita URL prefirmada → Sistema valida plan
2. Frontend sube archivo directo a S3 (sin pasar por backend)
3. Usuario registra beat con metadatos → Persistencia en MongoDB
4. Evento `BEAT_CREATED` emitido a Kafka

---

2. Arquitectura

Stack tecnológico

Categoría	Tecnología
Runtime	Node.js + Express
Base de Datos	MongoDB + Mongoose
Almacenamiento	AWS S3 + CloudFront CDN
Mensajería	Apache Kafka (KafkaJS)
Resiliencia	`bottleneck` (*rate limit*), `toobusy-js` (*backpressure*)
Testing	Vitest
CI/CD	GitHub Actions → Docker Hub
API Docs	OpenAPI 3.0 + Swagger UI

Estructura de carpetas

```
beats-upload/
├── .github/workflows/       # CI/CD: tests, linter, releases
├── spec/oas.yaml            # OpenAPI Specification
├── src/
```

```
│   ├── config/s3.js          # Cliente S3 + Bottleneck + toobusy-js
│   ├── controllers/          # Manejo de requests/responses
│   ├── middlewares/          # Auth, Authorization, Validation
│   ├── models/Beat.js        # Schema Mongoose
│   ├── routes/               # Definición de endpoints (+ Swagger docs)
│   ├── services/             # Lógica de negocio: S3, Kafka, CRUD
│   └── utils/                # CloudFront signer, SPACE client
├── tests/
│   ├── unit/                 # Tests in-process (mocks)
│   └── integration/          # Tests out-of-process
├── main.js                   # Entry point
├── Dockerfile                # Imagen producción
└── docker-compose.yml
```

---

3. API Interface

Base URL: `/api/v1`
Documentación interactiva: api.socialbeats.es/socialbeats-api/api-docs/

Endpoints Principales

Método	Endpoint	Descripción	Auth
POST	/beats/upload-url	Genera URL prefirmada para S3	☑
POST	/beats	Crea nuevo beat	☑
GET	/beats	Lista paginada (filtros: genre, bpm, tags)	✗
GET	/beats/search?q=	Búsqueda por título/artista/tags	✗
GET	/beats/my-beats	Beats del usuario (incluye privados)	☑
GET	/beats/{id}	Detalle de beat	✗*
PUT	/beats/{id}	Actualiza metadatos	☑
DELETE	/beats/{id}	Elimina beat (S3 + DB)	☑
GET	/beats/{id}/audio	Redirige a URL streaming	✗*
POST	/beats/{id}/play	Incrementa reproducciones	☑
GET	/beats/{id}/download	URL de descarga	☑
PATCH	/beats/{id}/promote	Activa/desactiva promoción	☑
GET	/health	Healthcheck	✗

*Requiere auth si el beat es privado

Códigos de estado

Código	Uso

Código	Uso
200	OK
201	Beat creado
302	Redirect a streaming/download
400	Validación fallida
401	No autenticado
403	Sin permisos (no owner, plan insuficiente)
404	Beat no encontrado
422	Límites del plan excedidos
503	Servidor sobrecargado (toobusy-js)

4. Detalles de implementación

4.1 Integración con AWS S3

Estrategia: Subida directa desde frontend → El backend solo genera URLs y registra metadatos.

Aspecto	Implementación
Bucket	Privado, acceso solo vía URLs firmadas
Estructura	`users/{userId}/{timestamp}-{filename}.{ext}`
Tipos permitidos	`mp3`, `wav`, `flac`, `aac`, `jpg`, `png`, `webp`
Límite de tamaño	15MB (definido en la politica de presigned URL)
Código	`src/config/s3.js`

4.2 CDN con CloudFront

Aspecto	Implementación
Firma de URLs	`@aws-sdk/cloudfront-signer`
Expiración	1h (acceso normal), 2h (streaming)
Range Requests	Soportados (seek/scrubbing en player)
Código	`src/utils/cloudfrontSigner.js`

4.3 Kafka (Event-Driven)

Eventos consumidos:

Evento	Trigger	Acción

Evento	Trigger	Acción
`USER_DELETED`	Usuario eliminado en `user-auth`	Elimina todos los beats del usuario (cascade en S3 + MongoDB)

Eventos producidos:

Evento	Trigger	Descripción
`BEAT_CREATED`	`POST /beats` exitoso	Notifica creación para vistas materializadas
`BEAT_UPDATED`	`PUT /beats/{id}` exitoso	Notifica cambio de metadatos
`BEAT_DELETED`	`DELETE /beats/{id}` exitoso	Notifica eliminación para limpiar referencias
`BEAT_PLAYS_INCREMENTED`	`POST /beats/{id}/play`	Notifica reproducción para estadísticas
`BEAT_DOWNLOADS_INCREMENTED`	`GET /beats/{id}/download`	Notifica descarga para estadísticas

Código: `src/services/kafkaConsumer.js`

4.4 Mecanismos de resiliencia

Mecanismo	Librería	Función
Rate Limiting	`bottleneck`	Limita a 5 operaciones simultáneas contra S3
Protección por sobrecarga	`toobusy-js`	Rechaza peticiones (503) cuando el servidor está saturado
Circuit Breaker	Custom	Reintentos con espera incremental ante fallos en Kafka
Graceful Shutdown	Custom	Cierre ordenado de conexiones antes de apagar

4.5 Seguridad

Capa	Implementación
Autenticación	Headers del Gateway: `x-gateway-authenticated`, `x-user-id`, `x-roles`
Autorización	`requireAuth`, `requireOwnership`, `requireBeatAccess`
Validación	`express-validator` + Mongoose schema validators
Uploads	Presigned URLs con políticas restrictivas (tamaño, MIME)

Código: `src/middlewares/authMiddlewares.js`

---

5. Matriz de cumplimiento (Nivel de acabado)

Microservicio Básico

Requisito	Estado	Detalle / Enlace
API REST con CRUD (GET, POST, PUT, DELETE)	☑	`beatController.js`, `beatRoutes.js`
Mecanismo de autenticación	☑	`authMiddlewares.js` - Headers del Gateway
Frontend integrado	☑	Frontend común - `beatsService.js`
Desplegado en la nube	☑	Kubernetes (DigitalOcean) vía API Gateway
API versionada (`/api/v1`)	☑	`main.js`, `src/routes/`
Documentación OpenAPI (Swagger)	☑	`spec/oas.yaml`, Swagger UI
Persistencia MongoDB	☑	`db.js`, `Beat.js`
Validación de datos (Mongoose)	☑	`validationMiddleware.js`, `Beat.js`
Imagen Docker	☑	`Dockerfile`, Docker Hub
GitHub Flow	☑	Repositorio, Metodología
CI/CD (GitHub Actions)	☑	`.github/workflows/`
Tests de componente (Vitest)	☑	`tests/` - unitarios e integración

Microservicio Avanzado

Requisito	Estado	Detalle / Enlace
Frontend con rutas (React Router)	☑	Frontend común
CDN/Caché (CloudFront)	☑	`cloudfrontSigner.js`
API externa (AWS S3)	☑	`s3.js`
Rate Limit (Bottleneck)	☑	`s3.js`
Autenticación JWT	☑	`authMiddlewares.js`
Circuit Breaker (Kafka)	☑	`kafkaConsumer.js`
Gestión de capacidad (toobusy-js)	☑	`s3.js`

Total características avanzadas: **7** (requisito: 6)

Niveles de puntuación

Nivel	Requisitos	Estado
Hasta 5 pts	Microservicio básico + Documento	☑
Hasta 7 pts	+ App microservicios + 3 características avanzadas	☑
Hasta 9 pts	+ 20 tests + OpenAPI + 5 características avanzadas	☑
Hasta 10 pts	+ 6 características avanzadas + Documento IA	☑

6. Tests

```
npm test              # Tests unitarios
npm run test:watch    # Modo watch
npm run test:coverage # Cobertura
```

Tipo	Ubicación	Cobertura
Unit	`tests/unit/`	Controllers, Services, Middlewares, Models, Utils
Integration	`tests/integration/`	Flujos completos con DB real

Resumen

Métrica	Valor
Framework	Vitest
Total tests	**287**
Tests unitarios	273 (14 archivos)
Tests integración	14 (1 archivo)

Tests Unitarios (273)

Archivo	Tests	Escenarios
`validationMiddleware.test.js`	56	Validación entrada, errores 422
`beatController.test.js`	51	CRUD, errores 4xx/5xx, ServerOverload
`beatService.test.js`	50	S3, CloudFront, límites plan
`aboutRoutes.test.js`	30	Docs, README, Swagger, Changelog
`authorizationMiddleware.test.js`	20	Ownership, permisos, 403
`s3.test.js`	12	Bottleneck, toobusy-js, presigned URLs
`kafkaConsumer.test.js`	12	Eventos, circuit breaker, DLQ
`cloudfrontSigner.test.js`	9	Firma URLs, expiración, config
`authMiddlewares.test.js`	8	Auth headers, rutas públicas, 401/400
`beat.test.js`	8	Schema, validación Mongoose, virtuals
`waveformService.test.js`	7	Generación de waveforms
`beatService.download.test.js`	5	Descargas, permisos, URLs firmadas
`versionUtils.test.js`	4	Changelog, versión
`health.test.js`	1	Healthcheck

Tests Integración (14)

Archivo	Tests	Escenarios
`beatService.test.js`	14	CRUD con MongoDB real, S3 mocks, cleanup

Tipos de escenarios

Tipo	Ejemplos
Positivos	Crear beat exitoso, URL válida, auth correcta
Negativos	Validación (400), sin auth (401), sin permisos (403)
Corner cases	Servidor sobrecargado (503), Kafka down, límites excedidos

---

7. Anexos

A. Uso de IA

Documentado en `docs/ai_usage.md`:

- Generación de código boilerplate
- Generación de tests
- Documentación

B. Documentación Externa

Documento	Enlace
Customer Agreement	Documentación global del proyecto
Análisis de Esfuerzos	Documentación global del proyecto
Ficha Técnica APIs	`docs/external_apis/`
Metodología de Trabajo	work_methodology.md