



**Universidad
Rey Juan Carlos**

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA**

**MASTER CLOUD APPS
Curso Académico 2019/2020**

Trabajo de Fin de Máster

**ELASTICTRANSCODER: VIDEOTRANSCODING
ENFOCADO A LA ELASTICIDAD**

Autor: Luis Carlos Jiménez López

Tutor: Micael Gallego Carrillo

*«Si tus sueños no te asustan, no son los suficientemente grandes»,
Richard Branson*

Agradecimientos

Gracias a mis padres y a mi hermana por estar ahí siempre detrás remando conmigo.

Gracias Marta Gonzalo por este año que has tenido que aguantar sabiendo que ya no iba a estudiar más, pero como siempre digo, el saber no ocupa lugar.

Gracias a mis compañeros de la universidad, que siempre me están escuchando cuando les cuento mis problemas de servidores, backend y frontend.

Gracias Mica, por otro año más aguantándome. Te seguiré de cerca, como siempre.

Resumen

Vídeo o audio son usados constantemente por millones de personas: Spotify, netflix, HBO o cualquier empresa que se dedique a eso necesita convertir todo ese contenido para que esté disponible en diferentes formatos, resoluciones y adaptado a cualquier tipo de dispositivo. La transcodificación sigue en desarrollo, nuevos códecs, nuevos tamaños de fichero para poder optimizar el tamaño que ocupan en los servidores.

Videotranscoding nació hace tres años y tiene sus limitaciones, como la nula capacidad para recuperar una transcodificación cuando ocurría un error; por ello se ha creado elastictranscoder usando parte del core funcional de videotranscoding¹, adaptándolo a las nuevas tecnologías y enfocado a la elasticidad.

Elastictranscoder resolverá los problemas de este contenido multimedia aportando frescura en lo que a elasticidad se refiere. Es necesario enfocarlo de la mejor manera posible usando buenas prácticas y una buena arquitectura para que se pueda escalar agregando nuevos componentes que mejoren aún más este proyecto.

El estudio de viabilidad para comprobar si es posible la elasticidad por contenedores docker en esta plataforma es lo primero que se debe realizar. Es importante destacar el posible desarrollo para dividir cada fichero en segmentos y conseguir así que la transcodificación sea capaz de ser aún mas rápida.

Utilizando tecnologías punteras como kubernetes, helm, flux o AWS se puede dar respuesta a esta elasticidad de una manera sencilla. Los microservicios usando springboot y la interfaz de usuario implementada con angular completan este proyecto en el que se dedicará el tiempo necesario para obtener los resultados que se establecen por requisitos.

Destacar que en este proyecto no se establecen como prioridades la parte frontal, pero se ha realizado un lavado de cara necesario después de tres años de evolución tecnológica.

Este trabajo contará con una introducción, unos objetivos definidos por requisitos, se explicarán los lenguajes, herramientas y tecnologías que se han usado, una descripción informática extensa de todos los puntos interesantes de este proyecto y como punto final hablaremos de las conclusiones y de los trabajos futuros de los puntos que se quedan abiertos de este trabajo final de máster.

Palabras clave: Contenido multimedia; Kubernetes; Elasticidad.

¹Trabajo de final de grado de Luis Carlos Jiménez López

Índice general

1	Introducción y Motivación	11
2	Objetivos	13
3	Lenguajes, Tecnologías y Herramientas	15
3.1	Lenguajes	15
3.1.1	Java	15
3.1.2	Typescript	16
3.1.3	Yaml	16
3.1.4	Shell	17
3.2	Tecnologías	18
3.2.1	Springboot	18
3.2.2	Angular	19
3.2.3	Docker	19
3.2.4	Helm	20
3.2.5	AWS	21
	SQS	21
	DynamoDB	22
3.2.6	HTML y CSS	22
3.3	Herramientas	23
3.3.1	Repositorios de Código	23
	GitHub	23
3.3.2	Integración continua	24
	Github Actions	24
3.3.3	Despliegue continuo	25
	FluxCD	25
3.3.4	Entornos de desarrollo	25
	VSCode	25
3.3.5	Gestión de video	26
	FFMPEG	26
3.3.6	Gestión de dependencias	27
	Maven	27
	NPM	27

3.3.7	Entorno de pruebas	28
	GlusterFS	28
	Kubernetes	28
4	Descripción informática	31
4.1	Objetivos	31
4.2	Arquitectura y Análisis	33
4.2.1	Amazon Simple Queue Service/SQS	33
4.2.2	Amazon DynamoDB	34
	Flat media	35
	Transcode media	36
4.2.3	GlusterFS	36
4.2.4	Kubernetes	38
4.2.5	Transcoder	38
4.2.6	Transcoder-handler	40
4.2.7	Transcoder-status	40
4.2.8	Media	41
4.2.9	Media-storage	41
4.2.10	Frontend	42
4.3	Diseño e Implementación	43
4.3.1	ProcessBuilder, StreamGobbler y FFMPEG	43
4.3.2	Kubernetes elástico	44
4.3.3	GlusterFS	45
4.3.4	Frontend	46
4.3.5	CloudWatch Adapter	52
4.3.6	Traefik	53
4.3.7	Monitorización del clúster	56
5	Conclusiones	57
	Bibliografía	59

Índice de figuras

2.1	Videotranscoding Dashboard.	13
3.1	Java logo	15
3.2	Typescript logo	16
3.3	Yaml logo	17
3.4	Shell logo	17
3.5	Springboot logo	18
3.6	Angular logo	19
3.7	Docker logo	20
3.8	Helm logo	20
3.9	AWS logo	21
3.10	AWS SQS logo	21
3.11	AWS DynamoDB logo	22
3.12	Html logo	23
3.13	Css logo	23
3.14	Github logo	24
3.15	Github Actions logo	24
3.16	Flux logo	25
3.17	Visual Studio Code logo	25
3.18	FFMPEG logo	26
3.19	Maven logo	27
3.20	NPM logo	27
3.21	Gluster logo	28
3.22	Kubernetes logo	28
4.1	Arquitectura de Elastictranscoder	33
4.2	Colas de trabajo SQS para elastictranscoder	34
4.3	Tablas DynamoDB para elastictranscoder	35
4.4	Comando tree del sistema de ficheros Gluster	37
4.5	Flujo del microservicio transcoder	39
4.6	Flujo del microservicio transcoder-handler	40
4.7	Flujo del microservicio transcoder-status	41
4.8	Flujo del microservicio media	41
4.9	Flujo del microservicio media-storage	42
4.10	Dashboard elastictranscoder sin login	46

4.11 Login on frontend using OKTA 47

4.12 Dashboard elastictranscoder 48

4.13 Componente de carga de contenido 48

4.14 Componente para añadir transcodificaciones 49

4.15 Elastictranscoder con todos sus componentes 50

4.16 Elastictranscoder multitarea 50

4.17 Dashboard Grafana usando datos de prometheus 56

Capítulo 1

Introducción y Motivación

El contenido multimedia está a la orden del día vía streaming en su mayoría, aunque aún se siguen encontrando archivos en un ordenador o móvil que no puedes reproducir debido al formato, códec de vídeo, códec de audio o tasa de bits del fichero demasiado alta para reproducirla. Para esta complejidad se creó Videotranscoding hace 3 años y hoy damos paso a Elastictranscoder, la actualización de la primera versión que con sus bondades y defectos se ha seguido usando en entorno casero.

El problema de todo este contenido sigue residiendo en los tipos de contenedores de los formatos multimedia. En audio está el mp3, muy extendido, pero sin evolución ninguna. Por otra parte, el contenedor de vídeo más extendido es el MP4, con muchos codecs de audio y video soportados; No sin antes mencionar otro contenedor, MKV, capaz de llevar dentro del fichero múltiples subtítulos y pistas de audio.

Existen multitud de aplicaciones en línea que se encargan de transcodificar estos ficheros de audio o vídeo haciendo uso de FFMPEG¹, con muchísima publicidad para poder mantener esas páginas y servidores, pero totalmente lícitas y bastante rápidas. Estas aplicaciones no dan soporte a muchos códecs, o últimos codecs del mercado, debido a que son más lentos en el momento de transcodificar debido a la complejidad de los mismos. Independientemente de esa lentitud, son códecs hechos con una misión, reducir el tamaño del fichero y no obtener pérdida de calidad al reducirla.

El último códec desarrollado que en los últimos años supondrá un avance importante en este área soportado por los reproductores es AV1. Fue lanzado el 28 de Marzo de 2018 y desarrollado por la 'Alliance for Open Media'. Es un códec de código abierto, sin royalty y diseñado para la transmisión de vídeo sobre la red. Todas estas incorporaciones de códecs son bienvenidas, pero es necesario que todos los navegadores y reproductores lo soporten. Es una pena que se creen nuevas formas de reproducir vídeo/audio y aún estemos estancados en el códec de vídeo H264 lanzado hace 17 años o en el MP3 de 1995.

¹Colección de software libre que transcodificar y hace streaming de audio y vídeo.

Videotranscoding tenía varios problemas que se van a resolver en este proyecto:

- Resiliencia cuando ocurría algún error o se paraba el servicio web.
- Escalabilidad: Posibilidad de aumentar el número de instancias para repartir la carga del servicio web.

Por ello, elastictranscoder resolverá estos problemas con una nueva arquitectura, nuevas tecnologías y herramientas que den soporte a un producto fresco y rápido a la hora de transcodificar esos videos.

Capítulo 2

Objetivos

El objetivo del desarrollo de esta aplicación es refactorizar la aplicación Videotranscoding para que se centre en la escalabilidad, resiliencia, simultaneidad y disponibilidad, conceptos que no se aplicaron en esa primera versión.

Para ello se pretende realizar un cambio de monolito¹ a microservicios. De esta manera se actualizarían todas las dependencias y un lavado de cara a la primera interfaz web creada en la figura 2.1.

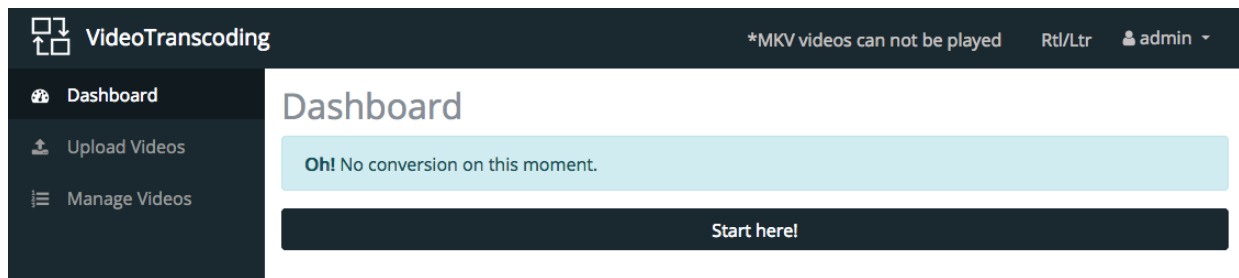


Figura 2.1: Videotranscoding Dashboard.

La metodología que se ha seguido en el desarrollo del software está dividida en las siguientes fases:

- **Fase 1:** Estudio de viabilidad para mejorar el tiempo en el que se transcodifica cada vídeo/audio haciendo uso de segmentación del vídeo o audio.
- **Fase 2:** Estudio de viabilidad de realizar microservicios desgranando cada parte de la aplicación en un módulo..
- **Fase 3:** Realización de los microservicios y refactorización del frontend quitando complejidad.

En toda esa nueva plataforma se implementarán nuevas bases de datos, sistema de colas para gestionar las transcodificaciones, entorno de integración continua y despliegue continuo y un entorno Kubernetes ARM donde se desplegará.

¹ Aplicación software en la que la capa de interfaz de usuario y la capa de acceso a datos están combinadas en un mismo programa y sobre una misma plataforma

Capítulo 3

Lenguajes, Tecnologías y Herramientas

El proyecto se inicia con una base de Java y Angular. Se incorporarán bastantes tecnologías para el desarrollo de esta refactorización a Videotranscoding.

3.1. Lenguajes

Las versiones usadas en esta parte del proyecto se han ido modificando debido a actualizaciones del software utilizado, llegando a una versión estable que permite realizar un despliegue de la aplicación con los mínimos errores posibles.

3.1.1. Java

Java es un lenguaje de programación desarrollado por Sun Microsystems; ahora pertenece a Oracle, que permite a los desarrolladores escribir instrucciones en el ordenador usando comandos básicos en inglés en lugar de tener que escribir en códigos numéricos. Es similar a C y C++, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel.



Figura 3.1: Java logo

3.1.2. Typescript

TypeScript es un lenguaje de código abierto que se basa en JavaScript, una de las herramientas más utilizadas del mundo, agregando definiciones de tipos estáticos.

Los tipos proporcionan una forma de describir la forma de un objeto, proporcionando una mejor documentación y permitiendo que TypeScript valide que su código funciona correctamente.

Los tipos de escritura pueden ser opcionales en TypeScript, porque la inferencia de tipos le permite obtener mucha potencia sin escribir código adicional.



Figura 3.2: Typescript logo

3.1.3. Yaml

YAML es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como el formato para correos electrónicos especificado en RFC 2822 (publicaciones RFC). YAML fue propuesto por Clark Evans en 2001, quien lo diseñó junto a Ingy döt Net y Oren Ben-Kiki.

YAML fue creado bajo la creencia de que todos los datos pueden ser representados adecuadamente como combinaciones de listas, hashes (mapeos) y datos escalares (valores simples). La sintaxis es relativamente sencilla y fue diseñada teniendo en cuenta que fuera muy legible pero que a la vez fuese fácilmente mapeable a los tipos de datos más comunes en la mayoría de los lenguajes de alto nivel. Además, YAML utiliza una notación basada en la indentación y/o un conjunto de caracteres Sigil distintos de los que se usan en XML, haciendo que sea fácil componer ambos lenguajes.

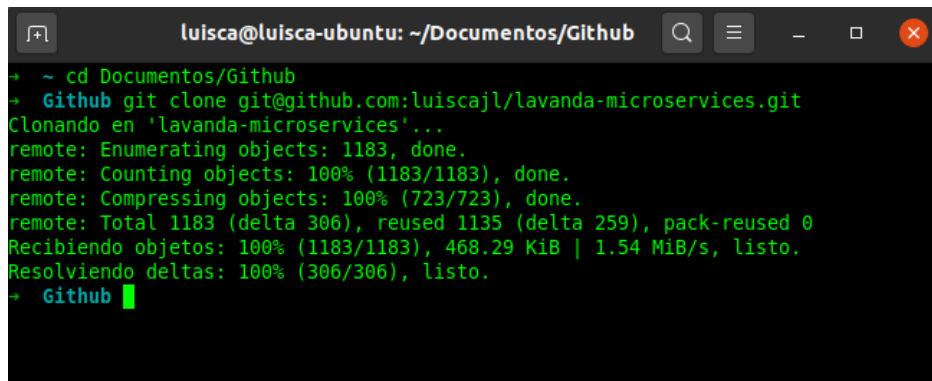


Figura 3.3: Yaml logo

3.1.4. Shell

Una Shell de Unix o también shell, es el término usado en informática para referirse a un intérprete de comandos, el cual consiste en la interfaz de usuario tradicional de los sistemas operativos basados en Unix y similares, como GNU/Linux.

Mediante las instrucciones que aporta el intérprete, el usuario puede comunicarse con el núcleo y por extensión, ejecutar dichas órdenes, así como herramientas que le permiten controlar el funcionamiento de la computadora.

A terminal window with a dark background and light green text. The window title is 'luisca@luisca-ubuntu: ~/Documentos/Github'. The terminal shows the following commands and output:

```
+ ~ cd Documentos/Github
+ Github git clone git@github.com:luiscajl/lavanda-microservices.git
Clonando en 'lavanda-microservices'...
remote: Enumerating objects: 1183, done.
remote: Counting objects: 100% (1183/1183), done.
remote: Compressing objects: 100% (723/723), done.
remote: Total 1183 (delta 306), reused 1135 (delta 259), pack-reused 0
Recibiendo objetos: 100% (1183/1183), 468.29 KiB | 1.54 MiB/s, listo.
Resolviendo deltas: 100% (306/306), listo.
+ Github
```

Figura 3.4: Shell logo

Los comandos que aportan los intérpretes, pueden usarse a modo de guión si se escriben en ficheros ejecutables denominados shell-scripts, de este modo, cuando el usuario necesita hacer uso de varios comandos o combinados de comandos con herramientas, escribe en un fichero de texto, marcado como ejecutable, las operaciones que posteriormente, línea por línea, el intérprete traducirá al núcleo para que las realice. Sin ser un shell estrictamente un lenguaje de programación, al proceso de crear scripts de shell se le denomina programación shell o en inglés, shell programming o shell scripting.

3.2. Tecnologías

3.2.1. Springboot

Spring Boot es una herramienta que nace con la finalidad de simplificar aun más el desarrollo de aplicaciones basadas en el ya popular framework Spring Core. Spring Boot busca que el desarrollador solo se centre en el desarrollo de la solución, olvidándose por completo de la compleja configuración que actualmente tiene Spring Core para poder funcionar.



Figura 3.5: Springboot logo

Spring Boot centra su éxito en las siguientes características que lo hacen extremadamente fácil de utilizar:

- **Configuración:** Spring Boot cuenta con un complejo módulo que autoconfigura todos los aspectos de nuestra aplicación para poder simplemente ejecutar la aplicación, sin tener que definir absolutamente nada.
- **Resolución de dependencias:** Con Spring Boot solo hay que determinar que tipo de proyecto estaremos utilizando y el se encarga de resolver todas las librerías/dependencias para que la aplicación funcione.
- **Despliegue:** Spring Boot se puede ejecutar como una aplicación Stand-alone, pero también es posible ejecutar aplicaciones web, ya que es posible desplegar las aplicaciones mediante un servidor web integrado, como es el caso de Tomcat, Jetty o Undertow.
- **Métricas:** Por defecto, Spring Boot cuenta con servicios que permite consultar el estado de salud de la aplicación, permitiendo saber si la aplicación está prendida o apagada, memoria utilizada y disponible, número y detalle de los Bean's creado por la aplicación, controles para el prendido y apagado, etc.
- **Extensible:** Spring Boot permite la creación de complementos, los cuales ayudan a que la comunidad de Software Libre cree nuevos módulos que faciliten aún más el desarrollo.

3.2.2. Angular

Angular es un framework de desarrollo para JavaScript creado por Google. La finalidad de Angular es facilitarnos el desarrollo de aplicaciones web SPA y además darnos herramientas para trabajar con los elementos de una web de una manera más sencilla y óptima.

Una aplicación web SPA creada con Angular es una web de una sola página, en la cual la navegación entre secciones y páginas de la aplicación, así como la carga de datos, se realiza de manera dinámica, casi instantánea, asincrónamente haciendo llamadas al servidor (backend con un API REST) y sobre todo sin refrescar la página en ningún momento.



Figura 3.6: Angular logo

Es decir las aplicaciones web que podemos hacer con Angular son reactivas y no recargan el navegador, todo es muy dinámico y asíncrono con ajax.

Otro propósito que tiene Angular es la separación completa entre el front-end y el back-end en una aplicación web.

3.2.3. Docker

Docker es una plataforma de software que te permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas las bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, se puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

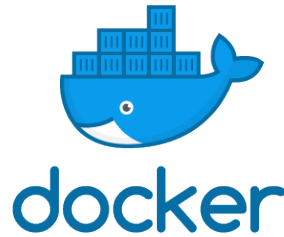


Figura 3.7: Docker logo

Proporciona una manera estándar de ejecutar su código. Docker es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que se pueden utilizar para crear, iniciar o detener contenedores.

3.2.4. Helm

Helm (del término marítimo de timón) es una herramienta para gestionar aplicaciones de Kubernetes. Helm te ayuda a orquestar Kubernetes usando cartas de navegación, que en inglés se conocen como Helm Charts. Aunque nos gustan las referencias de marítimas de Helm y Kubernetes, el término 'carta de navegación' o 'carta' puede ser más difícil de entender que Chart.



Figura 3.8: Helm logo

La principal función de Helm es definir, instalar y actualizar aplicaciones complejas de Kubernetes. Helm es mantenido por la CNCF en colaboración con Microsoft, Google, Bitnami y la comunidad de Helm.

Con Helm Charts es posible crear, versionar y publicar una aplicación Kubernetes. Cuando usamos Helm Charts tenemos un asistente de optimización que facilita la administración e instalación de las aplicaciones Kubernetes y el proceso de empaquetamiento.

3.2.5. AWS

Amazon Web Services (AWS) es la plataforma en la nube más adoptada y completa en el mundo, que ofrece más de 175 servicios integrales de centros de datos a nivel global. Millones de clientes, incluyendo las empresas emergentes que crecen más rápido, las compañías más grandes y los organismos gubernamentales líderes, están utilizando AWS para reducir los costos, aumentar su agilidad e innovar de forma más rápida.



Figura 3.9: AWS logo

Para reducir la complejidad de la aplicación y hacer uso de herramientas que hemos dado en el master vamos a implementar colas de SQS y base de datos usando DynamoDB.

SQS

Amazon Simple Queue Service (SQS) es un servicio de colas de mensajes completamente administrado que permite desacoplar y ajustar la escala de microservicios, sistemas distribuidos y aplicaciones sin servidor. SQS elimina la complejidad y los gastos generales asociados con la gestión y el funcionamiento del middleware orientado a mensajes, y permite a los desarrolladores centrarse en la diferenciación del trabajo. Con SQS se puede enviar, almacenar y recibir mensajes entre componentes de software de cualquier volumen, sin pérdida de mensajes ni la necesidad de que otros servicios estén disponibles.

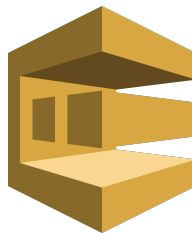


Figura 3.10: AWS SQS logo

SQS ofrece dos tipos de colas de mensajes. Las colas estándar ofrecen una capacidad de procesamiento máxima, un ordenamiento de mejor esfuerzo y una entrega al menos una vez. Las colas

FIFO de SQS están diseñadas para garantizar que los mensajes se procesen exactamente una vez, en el orden exacto en el que se enviaron.

DynamoDB

Amazon DynamoDB es una base de datos de clave-valor y documentos que ofrece rendimiento en milisegundos de un solo dígito a cualquier escala. Se trata de una base de datos duradera de varias regiones y con varios maestros, completamente administrada, que cuenta con copia de seguridad, restauración y seguridad integradas, y almacenamiento de caché en memoria para aplicaciones a escala de Internet. DynamoDB puede gestionar más de 10 billones de solicitudes por día y puede admitir picos de más de 20 millones de solicitudes por segundo.



Figura 3.11: AWS DynamoDB logo

3.2.6. HTML y CSS

HTML es el lenguaje que se emplea para el desarrollo de páginas de internet. Está compuesto por una serie de etiquetas que el navegador interpreta y da forma en la pantalla. HTML dispone de etiquetas para imágenes, hipervínculos que nos permiten dirigirnos a otras páginas, saltos de línea, listas, tablas, etc.

El código HTML es un lenguaje muy simple y fácil de interpretar en términos generales por ejemplo: `` indica que los navegadores web visuales deben mostrar el texto en negrita; entonces podemos decir que estas marcas o etiquetas son como instrucciones a las que obedece el navegador para determinar la forma en la que debe aparecer.



Figura 3.12: Html logo

El CSS (hojas de estilo en cascada) es un lenguaje que define la apariencia de un documento escrito en un lenguaje de marcado como HTML. Así, a los elementos de la página web creados con HTML se les dará la apariencia que se desee utilizando CSS: colores, espacios entre elementos, tipos de letra, separando de esta forma la estructura de la presentación.

Esta separación entre la estructura y la presentación es muy importante, ya que permite que solo cambiando los CSS se modifique completamente el aspecto de una página web. Esto posibilita, entre otras cosas, que los usuarios puedan usar hojas de estilo personalizadas (como hojas de estilo de alto contraste o de accesibilidad).



Figura 3.13: Css logo

3.3. Herramientas

3.3.1. Repositorios de Código

Para guardar el código que se va a realizar es necesario que sea guardado en aplicaciones externas, ya que están dedicadas para esos menesteres. Por ello, se utilizará Github como repositorio de código.

GitHub

GitHub es un servicio de alojamiento web, propiedad de Microsoft, para control de versiones que usa Git. En él se aloja principalmente código fuente. Ofrece todas las funciones de control

de versiones distribuidas y administración de código fuente (SCM) de Git, además de agregar sus propias características. Proporciona control de acceso y varias funciones de colaboración, como seguimiento de errores, solicitudes de funciones, gestión de tareas y wikis para cada proyecto.



Figura 3.14: Github logo

3.3.2. Integración continua

La integración continua es necesario en cualquier proyecto, se requiere de programas para que compilen el código fuente y creen en este caso imágenes de docker que serán usadas en el entorno de pruebas.

Github Actions

Github Actions permite crear flujos de trabajo (workflows) que se pueden utilizar para compilar, probar y desplegar código, dando la posibilidad de crear flujos de integración y despliegue continuo dentro del propio repositorio de git.

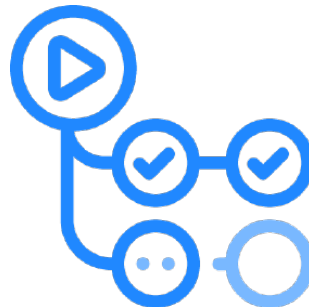


Figura 3.15: Github Actions logo

Los flujos de trabajo tienen que contener al menos un job. Estos incluyen una serie de pasos que ejecutan tareas individuales que pueden ser acciones o comandos. Un flujo de trabajo puede comenzar por distintos eventos que suceden dentro de GitHub, como un push al repositorio o un pull request.

3.3.3. Despliegue continuo

FluxCD

Flux es una herramienta que garantiza automáticamente que el estado de su clúster de Kubernetes coincida con la configuración que proporcionó en Git. Utiliza un operador en el clúster para activar implementaciones dentro de Kubernetes, lo que significa que no necesita una herramienta de entrega continua separada.



Figura 3.16: Flux logo

3.3.4. Entornos de desarrollo

VSCode

Visual Studio Code o Vs Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y Mac OS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, aunque la descarga oficial está bajo software propietario.



Figura 3.17: Visual Studio Code logo

Visual Studio Code se basa en Electron, un framework que se utiliza para implementar aplicaciones Node.js para el escritorio que se ejecuta en el motor de diseño Blink. Aunque utiliza el framework Electron, el software no usa Atom y en su lugar emplea el mismo componente editor

("Monaco") utilizado en Visual Studio Team Services (anteriormente llamado Visual Studio Online).

3.3.5. Gestión de video

FFMPEG

FFmpeg es una aplicación líder *open source* capaz de decodificar, codificar, transcodificar, mux, demux, transmitir, filtrar y reproducir casi cualquier formato de vídeo o audio. Es compatible con los formatos antiguos o los más nuevos. Es multiplataforma: Desde Linux, pasando por Windows, hasta Mac OS. Funciona bajo una amplia variedad de entornos de construcción, arquitecturas de máquinas y configuraciones.



Figura 3.18: FFMPEG logo

El proyecto completo está compuesto por:

- Ffmpeg: es una herramienta de línea de comandos (que se usará para este proyecto) para convertir audio o vídeo de un formato a otro. También puede capturar y codificar en tiempo real desde DirectShow, una tarjeta de televisión u otro dispositivo compatible.
- Ffserver: es un servidor de streaming multimedia de emisiones en directo que soporta HTTP (la compatibilidad con RTSP está en desarrollo). Todavía no está en fase estable, y de momento no está disponible para Windows.
- Ffplay: es un reproductor multimedia basado en SDL y las bibliotecas FFmpeg.
- Libavcodec: es una biblioteca que contiene todos los códecs de FFmpeg. Muchos de ellos fueron desarrollados desde cero para asegurar una mayor eficiencia y un código altamente reutilizable.
- Libavformat: es una biblioteca que contiene los multiplexadores/demultiplexadores para los archivos contenedores multimedia.
- Libavutil: es una biblioteca de apoyo que contiene todas las rutinas comunes en las diferentes partes de FFmpeg.

- Libpostproc: es una biblioteca de funciones de postproceso de vídeo.
- Libswscale: es la biblioteca de escalado de vídeo.

3.3.6. Gestión de dependencias

La gestión de dependencias facilita el desarrollo a través de la automatización de descargas de nuevos bloques que sean necesarios para el código que se esté realizando. En este caso vamos a tratar dos que son las que se van a utilizar:

Maven

Maven es un software de gestión de proyectos. Se basa en un fichero llamado pom.xml donde se define todo lo que necesita el proyecto. Se encarga de descargar las dependencias necesarias, compilar, empaquetar y ejecutar los test del proyecto. Maven trabaja con grupos y artefactos. Un artefacto es un componente software que podemos incluir en nuestro proyecto como una dependencia. Normalmente será un JAR. Un grupo es un conjunto de artefactos. Un ejemplo sería el grupo del framework Spring.



Figura 3.19: Maven logo

NPM

Para hacer uso de paquetes para angular se usa NPM, el administrador del paquete Node. Este instala los paquetes que se quieren usar y proporciona una interfaz útil para trabajar con ellos.



Figura 3.20: NPM logo

3.3.7. Entorno de pruebas

Estos entornos se usarán para poder desarrollar la aplicación en un entorno parejo al que se usaría en un entorno de producción.

GlusterFS

GlusterFS es un sistema multiescalable de archivos para NAS desarrollado inicialmente por Gluster Inc. Este permite agregar varios servidores de archivos sobre Ethernet o interconexiones Infiniband RDMA en un gran entorno de archivos de red en paralelo. El diseño del GlusterFS se basa en la utilización del espacio de usuario y de esta manera no compromete el rendimiento. Se pueden encontrar siendo utilizado en una gran variedad de entornos y aplicaciones como computación en nube, ciencias biomédicas y almacenamiento de archivos. El GlusterFS está licenciado bajo la licencia GNU General Public License versión 3.



Figura 3.21: Gluster logo

Gluster Inc fue el principal patrocinador comercial del GlusterFS, el cual ofrece tanto productos comerciales como apoyo para desarrollo de soluciones libres basadas en el GlusterFS. En octubre de 2011, fue anunciada la adquisición de Gluster Inc por Red Hat Inc.

Kubernetes

Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento. El soporte, las herramientas y los servicios para Kubernetes están ampliamente disponibles.

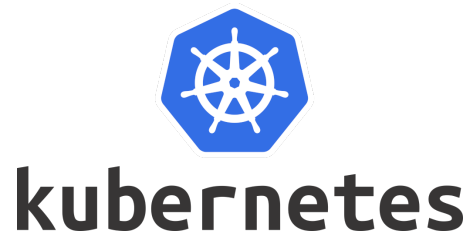


Figura 3.22: Kubernetes logo

Kubernetes puede ser utilizado de difentes formas:

- Plataforma de contenedores
- Plataforma de microservicios
- Plataforma portable de cloud

Kubernetes ofrece un entorno de administración centrado en contenedores. Kubernetes orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo. Esto ofrece la simplicidad de las Plataformas como Servicio (PaaS) con la flexibilidad de la Infraestructura como Servicio (IaaS) y permite la portabilidad entre proveedores de infraestructura.

Una vez explicadas los lenguajes, tecnologías y herramientas que se va a usar en este proyecto, damos paso al capítulo 4 en el que detallaremos el proceso que se ha seguido para el mismo.

Capítulo 4

Descripción informática

Para lograr este proyecto de la mejor manera posible se deben seguir unos estándares:

- Análisis de objetivos y pruebas de las mejoras a implementar en entorno local.
- Esquema de la arquitectura e implementación de herramientas externas.
- Desarrollo y pruebas.

Se explicará como se ha realizado el proyecto internamente a nivel de código, haciendo hincapié en las partes más complejas y que merezca la pena dar una aclaración que ayude a su comprensión.

Por último, se comprobarán los resultados obtenidos con las pruebas que se han realizado, los tiempos medidos, la rapidez de los mismos y la cantidad de código que se ha examinado. Además se darán unas conclusiones de la solución escogida.

4.1. Objetivos

La definición de objetivos de la aplicación, ayudan al desarrollador a fijar unas metas del producto que se quiere implementar. Con ellos podemos abordar el desarrollo fijando la visión del producto final.

- Conversión de diferentes vídeos usando contenedores en diferentes nodos.
- Guardar las conversiones y ficheros originales en un sistema de ficheros que se pueda acceder desde diferentes nodos.
- Implementación sistema de colas para gestionar las transcodificaciones.
- Gestionar los datos de las transcodificaciones y su estado usando una base de datos no relacional.

- Implementación de un websocket para comunicar con el frontend el estado de las transcodificaciones.
- Obtención de la información del fichero que se ha transcodificado para obtener metadatos.
- Escalabilidad de contenedores de transcodificación dependiendo del tamaño de la cola.
- Capa REST de la media convertida y la original.
- Autenticación usando OKTA.
- Interfaz Web mejorada con respecto a videotranscoding.
- Reproductor flotante que no quite importancia a las transcodificaciones en el frontend.

Con estos objetivos podemos empezar a desarrollar toda la arquitectura del proyecto y su organización.

4.2. Arquitectura y Análisis

La arquitectura se ha dividido en varias partes. Se han desarrollado de manera que una vez un microservicio está terminado se puede subir, probar y no tener que realizar otro despliegue.

Todos los microservicios creados guardan el estado de la aplicación y por lo tanto son replicables, se pueden crear mas instancias del mismo y así poder repartir la carga entre ellos.

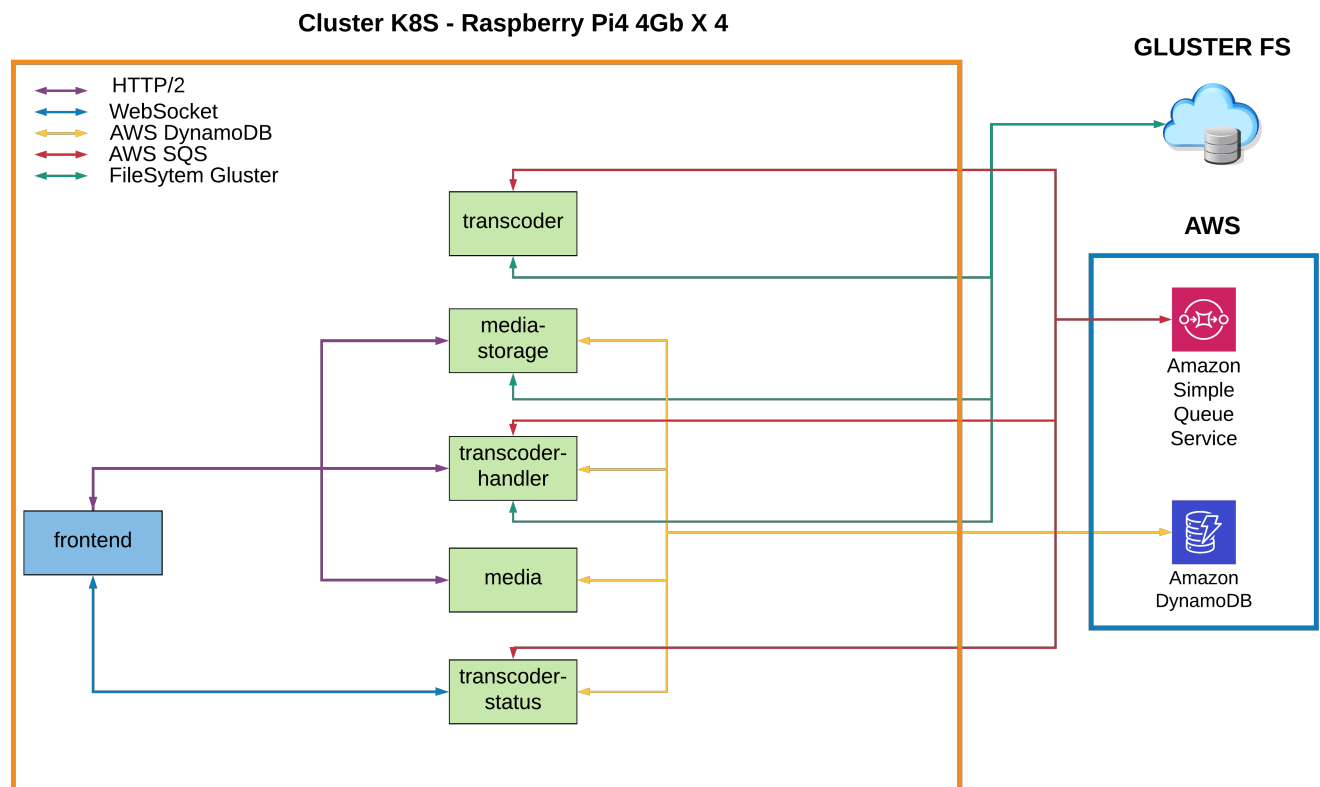
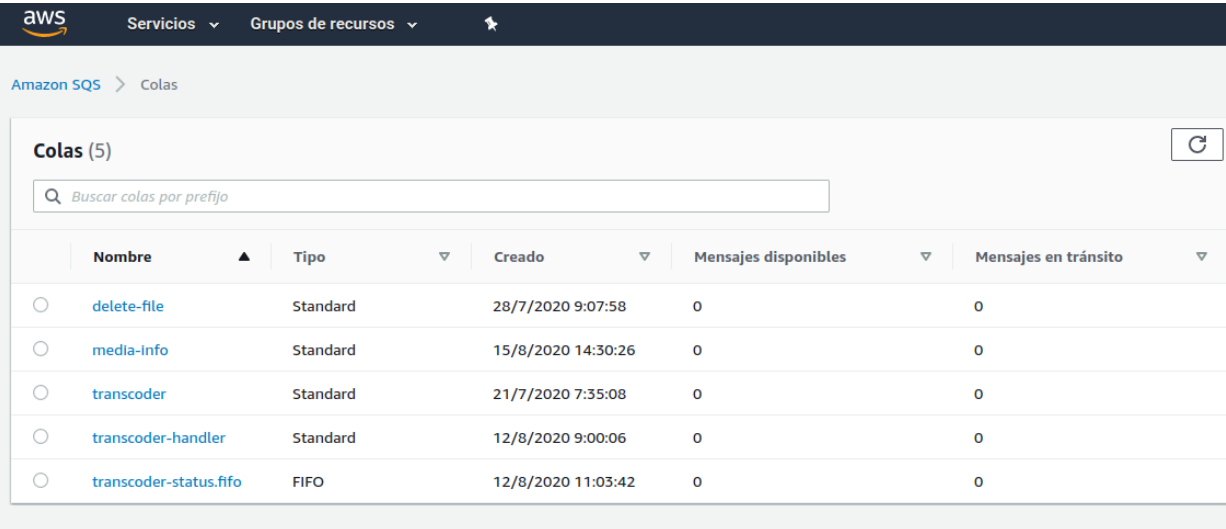


Figura 4.1: Arquitectura de Elastictranscoder

Vamos a desgranar la Figura 4.1 y realizar la explicación de cada parte de la aplicación.

4.2.1. Amazon Simple Queue Service/SQS

Para poder obtener elasticidad y comunicar los microservicios encargados de transcodificar es necesario implementar colas de trabajo. La sencillez y autonomía que ofrece amazon web services permite al usuario realizar otras tareas que requieran mas atención.



	Nombre ▲	Tipo ▼	Creado ▼	Mensajes disponibles ▼	Mensajes en tránsito ▼
<input type="radio"/>	delete-file	Standard	28/7/2020 9:07:58	0	0
<input type="radio"/>	media-info	Standard	15/8/2020 14:30:26	0	0
<input type="radio"/>	transcoder	Standard	21/7/2020 7:35:08	0	0
<input type="radio"/>	transcoder-handler	Standard	12/8/2020 9:00:06	0	0
<input type="radio"/>	transcoder-status.fifo	FIFO	12/8/2020 11:03:42	0	0

Figura 4.2: Colas de trabajo SQS para elastictranscoder

Las colas usadas en la Figura 4.2 son las siguientes:

- **Transcoder:** Cola principal de la aplicación. Contiene las transcodificaciones que no se han realizado. Dispone de un periodo de retención de la transcodificación. Tiempo suficiente para que no se pierda y algún microservicio de transcoder sea capaz de realizar la transcodificación.
- **Transcoder-handler:** Cola cuya función es obtener los datos finales del fichero ya transcodificado.
- **Transcoder-status.fifo:** Cola FIFO. Mantiene el orden de entrada y salida, el primero que entra es el primero que sale. Encargado de enviar el estado que es enviado desde el microservicio de transacoder a transcoder-status, para registrar el estado de la transcodificación que se esté realizando en ese servicio-
- **Media-info:** Cola que se ejecuta cuando un contenido es subido. Analiza los metadatos y los guarda en dynamoDB.
- **Delete-file:** Cola encargada de no dejar rastro de ficheros. Cuando llega una petición de borrar al microservicio de media, este envía un mensaje a esta cola para que media-storage se encargue de eliminar los ficheros padre e hijos que se haya solicitado.

4.2.2. Amazon DynamoDB

DynamoDB, una base de datos no relacional que permite una fácil implementación en cualquier tipo de entorno. Pasamos a detallar la estructura de estos 2 únicos objetos que han sido necesarios



Figura 4.3: Tablas DynamoDB para elastictranscoder

Flat media

Objeto que permite mantener la metainformación de la media original, antes de ser transcodificado. Es el objeto sobre el que se van a realizar las transcodificaciones. Los campos necesarios para tener un control sobre este tipo de media son los siguientes:

- Active: Estado del objeto, 1 si se está realizando una transcodificación de él.
- AudioCodec: Códec del contenido multimedia.
- Bitrate: Bitrate del contenido multimedia.
- Container: Extensión del fichero.
- Duration: Duración del fichero.
- Filesize: Tamaño de fichero.
- Id: Id autoasignado por el servicio.
- Name: Nombre original fichero, sin extensión.
- Path: Ruta donde se ha almacenado el fichero.
- Resolution: Resolución del vídeo si procede.
- Type: Tipo de contenido, distingue entre audio y vídeo.
- User: Usuario que ha realizado la subida del fichero.
- VideoCodec: Códec del vídeo si procede.

Transcode media

- Active: Estado del objeto, 1 si se está realizando una transcodificación de el.
- AudioCodec: Códec del contenido multimedia.
- Bitrate: Bitrate del contenido multimedia.
- Command: Comando Bash que es ejecutado por el microservicio transcoder.
- Container: Extensión del fichero.
- Duration: Duración del fichero.
- Filesize: Tamaño de fichero.
- FlatMediaId: Id de flat media, es decir, del fichero original.
- Id: Id autoasignado por el servicio.
- Name: Nombre original fichero, sin extensión.
- Path: Ruta donde se ha almacenado el fichero.
- Preset: Tipo de transcodificación de FFMPEG.
- Processed: Número que contiene el proceso de la transcodificación.
- Resolution: Resolución del vídeo si procede.
- Type: Tipo de contenido, distingue entre audio y vídeo.
- User: Usuario que ha realizado la subida del fichero.
- VideoCodec: Códec del vídeo si procede.

4.2.3. GlusterFS

GlusterFS es un sistema de ficheros, en este caso se ha usado para guardar los ficheros originales y sus transcodificaciones de manera que sean accesibles a través de la red.

```

→ elastictranscoder-data tree -L 2
.
├── FlatMedia
│   ├── AFTER-EARTH_TLR-D_GEN_EN-XX_INTL_51-4K-HDTN_1599233339359
│   ├── DARK-KNIGHT-RISES_TLR-F1_51-4K-HDTN_1599233232985
│   ├── Elysium_trailer_1-4K-HDTN_1599233152340
│   ├── PuppierBath4K_1599233124697
│   ├── StarWarsTrailer2_1598779090292
│   ├── StarWarsTrailer2_1599232988796
│   ├── file_example_MP4_1920_18MG_1598779213662
│   ├── file_example_MP4_1920_18MG_1599232911427
│   ├── sample_3840x2160_1598779226184
│   ├── sample_3840x2160_1599232998368
│   └── small_1599663723421
└── TranscodeMedia
    ├── DARK-KNIGHT-RISES_TLR-F1_51-4K-HDTN_1599233232985
    ├── Elysium_trailer_1-4K-HDTN_1599233152340
    ├── StarWarsTrailer2_1598779090292
    ├── StarWarsTrailer2_1599232988796
    ├── file_example_MP4_1920_18MG_1598779213662
    ├── file_example_MP4_1920_18MG_1599232911427
    ├── sample_3840x2160_1598779226184
    └── small_1599663723421

21 directories, 0 files
→ elastictranscoder-data █

```

Figura 4.4: Comando tree del sistema de ficheros Gluster

El sistema de ficheros está dividido en 2, y a su vez en subcarpetas.

- FlatMedia:
 - /NOMBRE_DEL_FICHERO_TIMESTAMP
 - /NOMBRE_DEL_FICHERO_.EXTENSION
- TranscodeMedia:
 - /NOMBRE_DEL_FICHERO_TIMESTAMP
 - /NOMBRE_DEL_FICHERO_.EXTENSION
 - /CODECVIDEO_RESOLUCION_CODECAUDIO_PRESET_EXTENSION
 - /NOMBRE_DEL_FICHERO.EXTENSION

4.2.4. Kubernetes

Kubernetes dispone de diferentes manifiestos para poder organizar el clúster:

- **Service:** Forma abstracta de exponer una aplicación que se ejecuta en un conjunto de Pods como un servicio de red.
- **Deployment:** Un controlador de Deployment proporciona actualizaciones declarativas para los Pods y los ReplicaSets.
- **Pod:** Los Pods son las unidades de computación desplegadas más pequeñas que se pueden crear y gestionar en Kubernetes
- **Configmap:** Objeto de la API utilizado para almacenar datos no confidenciales en el formato clave-valor. Los Pods pueden utilizar los ConfigMaps como variables de entorno, argumentos de la línea de comandos o como ficheros de configuración en un Volumen .
- **Secret:** Objetos seguros que almacenan datos sensibles, como contraseñas, tokens o llaves SSH.

En caso de que estos manifiestos no cumplan nuestras necesidades podemos hacer usos de CRDs(Custom resource definition) en el que se declaran para ese cluster un nuevo tipo de manifiesto que cumpla lo que sea necesario. En este caso se han usado CRDs como HelmRelease, manifiesto creado por The Flux maintainers para el despliegue automatizado de Flux, o IngressRoute, un CRD usado para el uso de Traefik¹ e ingress².

El despliegue de la aplicación se ha realizado usando un clúster con 4 nodos con ubuntu server como sistema operativo host. A través de Helm y flux se realizan los despliegues automatizados sin necesidad de intervenir en los hosts.

4.2.5. Transcoder

Core de la aplicación. Encargado de ejecutar en un solo hilo la petición de transcodificación que le llega por una cola SQS. Haciendo uso de programación podemos conectarnos a la terminal del host en el que nos encontremos y convertir un contenido multimedia usando FFMPEG. Por ello, cada línea que la terminal nos devuelve la analizamos y si son datos nuevos, como el progreso de la conversión, es enviado a la cola de estado de SQS.

¹Controlador parecido a nginx

²Controlador nginx oficial de kubernetes

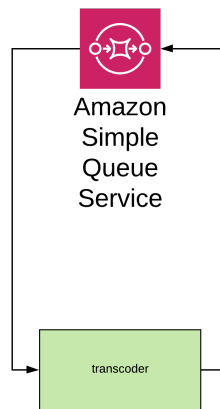


Figura 4.5: Flujo del microservicio transcoder

Como es un proceso largo se realiza en un solo thread, que es especificado por los límites del deployment de kubernetes. Más adelante se detallará el comportamiento de estos límites.

4.2.6. Transcoder-handler

Encargado de manejar las conversiones que se van a realizar y de los tipos de conversión disponibles a través de una API rest con el frontend. Este microservicio contiene la lógica que se va a aplicar para que FFMPEG pueda convertir la media dependiendo del tipo de conversión que se requiera. Dispone de una api rest con un método POST para crear esas nuevas transcodificaciones y que posteriormente sean enviadas a la cola de trabajo.

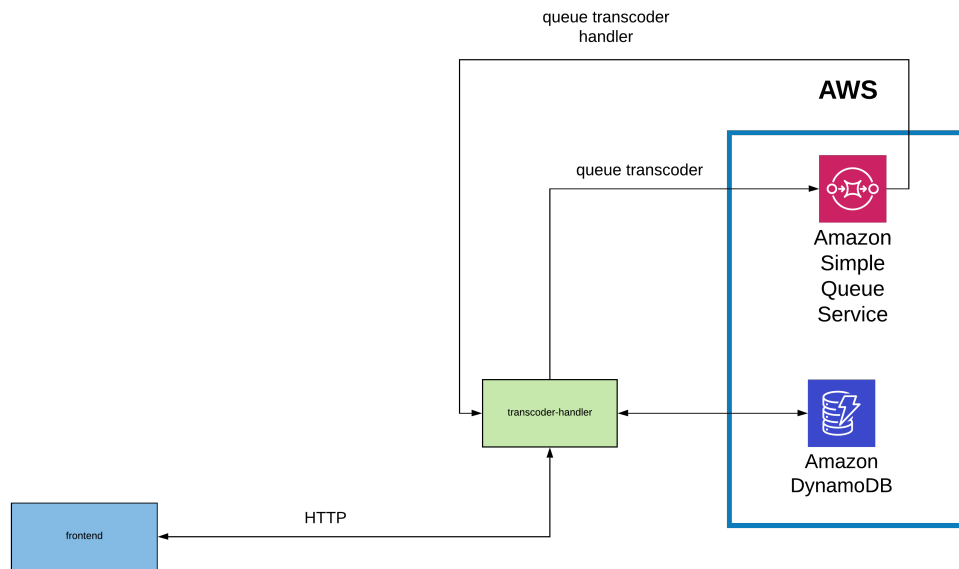


Figura 4.6: Flujo del microservicio transcoder-handler

Se comunica con las colas SQS, como se ve en la Figura 4.6, para enviar las nuevas transcodificaciones o obtener los metadatos de los ficheros convertidos o de los originales.

4.2.7. Transcoder-status

Microservicio que tiene una conexión websocket para comunicarse con el frontend y enviar el estado de las transcodificaciones en vivo. Para ello lee de la BBDD, que veremos en el siguiente microservicio y envía las actualizaciones al frontend.

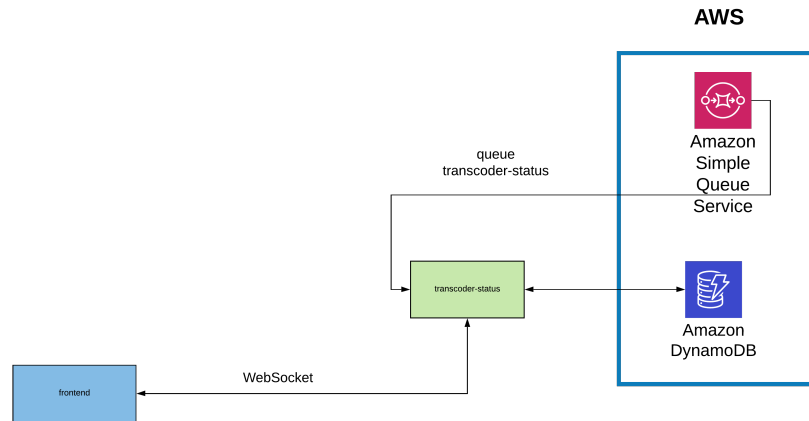


Figura 4.7: Flujo del microservicio transcoder-status

Además, lee de la cola de estado y guarda en base de datos el estado actual de las transcodificaciones.

4.2.8. Media

Microservicio que tiene conexión con DynamoDB para obtener los datos del servicio web de AWS y así retornarlos en forma de json a través de una api rest. Se exponen diferentes métodos para obtener o eliminar el contenido, tanto si es contenido sin modificar o las transcodificaciones.

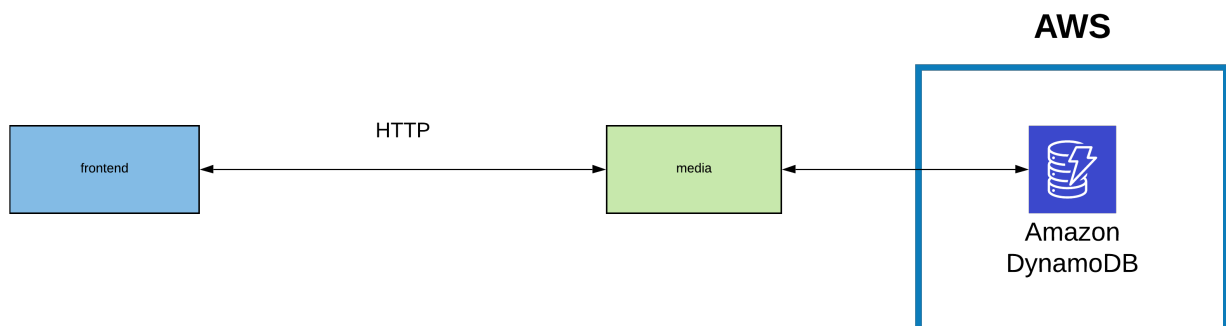


Figura 4.8: Flujo del microservicio media

4.2.9. Media-storage

Microservicio encargado de subir, descargar o reproducir todo el contenido multimedia. Tiene acceso al sistema de ficheros para poder realizar el streaming, la descarga o la subida. La comuni-

cación entre el frontend y el microservicio es vía api rest.

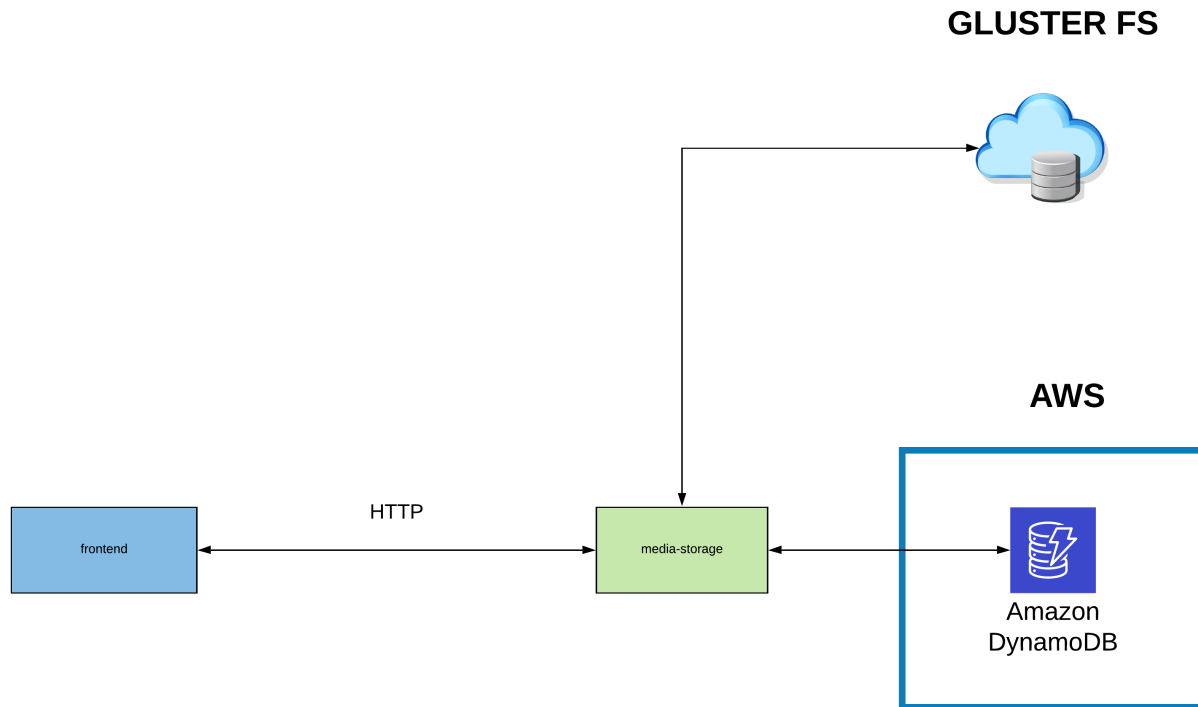


Figura 4.9: Flujo del microservicio media-storage

4.2.10. Frontend

Aplicación angular 10, diseñada usando angular material. Securizada con OKTA. Tiene acceso a los microservicios junto con el token JWT de okta haciendo uso de http. También tiene acceso al microservicio de transcoder-status via websocket para obtener el estado actual de las transcodificaciones.

Una vez visto la arquitectura y los servicios que se van a usar vamos a dar un repaso a la implementación que se ha llevado a cabo.

4.3. Diseño e Implementación

En esta parte detallaremos las partes más importantes de la aplicación y como ha sido su desarrollo. Por ponernos un poco en contexto las partes a detallar serán las siguientes:

- ProcessBuilder, StreamGobbler y FFMPEG: Detallaremos como funciona este proceso y el tratamiento de errores
- Kubernetes elástico: Proceso en el que el clúster ajusta su número de pods dependiendo de la carga de transcodificación.
- GlusterFS: Almacenamiento entre clústeres
- Frontend: Angular material
- CloudWatch Adapter: CRD de kubernetes para obtener datos de las colas SQS
- Traefik: Ingress de kubernetes
- Cluster Monitoring: Grafana y prometheus.

4.3.1. ProcessBuilder, StreamGobbler y FFMPEG

Transcodificar es muy sencillo usando ffmpeg:

```
ffmpeg -i input -c:v libx265 -crf 26 -preset fast -c:a aac -b:a 128k output.mp4
```

Como se puede ver en el código anterior es bash, por lo que es necesario usar una terminal para ejecutarlo. ProcessBuilder es una de las mejores formas usando java para llamar a esa terminal y poder ejecutar código bash.

```
public int transcode(TranscodeMediaDTO transcodeMediaDTO) throws TranscoderException {
    log.info("Transcoding file {}", transcodeMediaDTO.toString());
    transcodeMediaDTO.setActive(true);
    producerService.sendStatus(transcodeMediaDTO);
    try {
        finalTime = 0;
        Process process = new ProcessBuilder("bash", "-c",
            transcodeMediaDTO.getCommand()).redirectErrorStream(true)
            .start();
        StreamGobbler streamGobbler = new StreamGobbler(process.getInputStream(), line -> {
            log.debug("FFMPEG commandline: {}", line);
            Matcher progressMatcher = progressVideoPattern.matcher(line);
            Matcher durationVideoMatcher = durationVideoPattern.matcher(line);
            if (progressMatcher.find()) {
                double difference = getDifference(finalTime, progressMatcher.group(0));
                transcodeMediaDTO.setProcessed(difference);
                if ((Math.round(transcodeMediaDTO.getProcessed() * 100.0) / 100.0) != 100) {
                    try {
                        producerService.sendStatus(transcodeMediaDTO);
                    } catch (TranscoderException e) {
                        log.error("Runtime transcoderException", e);
                        throw new TranscoderRuntimeException("Runtime transcoderException", e);
                    }
                }
            }
        })
    } catch (Exception e) {
        log.error("Exception in transcode", e);
        throw new TranscoderException(e);
    }
}
```

```

        if (durationVideoMatcher.find()) {
            finalTime = getDuration(durationVideoMatcher.group(0));
        }
    });
    executorService.submit(streamGobbler);
    int status = process.waitFor();
    if (status != 0) {
        transcodeMediaDTO.setProcessed(0);
        transcodeMediaDTO.setError(true);
    } else {
        transcodeMediaDTO.setProcessed(100);
    }
    transcodeMediaDTO.setActive(false);
    producerService.sendStatus(transcodeMediaDTO);
    log.info("File {} transcode with status {}", transcodeMediaDTO.toString(), status);
    return status;
} catch (InterruptedException | IOException e) {
    log.error("Exception on command line transcode", e);
    transcodeMediaDTO.setProcessed(0);
    transcodeMediaDTO.setError(true);
    transcodeMediaDTO.setActive(false);
    producerService.sendStatus(transcodeMediaDTO);
    Thread.currentThread().interrupt();
    throw new TranscoderException("Exception on command line transcode", e);
}
}

```

Cuando se prepara el comando que se va a ejecutar necesitamos leer la entrada de errores, por ello se hace uso de ‘redirectErrorStream(true)’.

StreamGobbler se encarga de recoger las líneas que la terminal devuelve para que así sean analizadas usando regex y poder actualizar el estado de la transcodificación. Cuando una transcodificación es terminada, su estado final es procesado a 100, error en false y active en false. En caso de que exista un fallo, estas variables son inversas y la cola SQS no podrá borrar el mensaje y se apilará en la cola para que más adelante vuelva a ser transcodificado.

4.3.2. Kubernetes elástico

Cuando un pod transcoder de kubernetes está ocupado y existen más transcodificaciones para realizar, hay un recurso de kubernetes, HorizontalPodAutoScaler, para aumentar el número de pods/réplicas

```

{{- if .Values.autoscaling.enabled }}
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2beta2
metadata:
  name: transcoder-scaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: transcoder
  minReplicas: {{ .Values.autoscaling.minReplicas }}
  maxReplicas: {{ .Values.autoscaling.maxReplicas }}
  metrics:
    - type: External

```

```

external:
  metric:
    name: sqs-transcoder-length
  target:
    type: AverageValue
    averageValue: "1"
{{- end }}

```

Las métricas de las que se alimenta el código anterior son externas. Existe una relación unilateral entre el pod de transcoder y la transcodificación. Por tanto, un mensaje en la cola SQS de transcoder creará un nuevo pod para que se ejecute la tarea. Automáticamente este tipo de manifiesto se encarga de aumentar o disminuir la cantidad de réplicas/pods.

```

resources:
  limits:
    cpu: 1000m
    memory: 1000Mi
  requests:
    cpu: 200m
    memory: 128Mi

autoscaling:
  enabled: true
  minReplicas: 1
  maxReplicas: 10

```

Cada pod tiene unos límites y unos recursos de los que siempre tiene disponibles. En este caso, el pod siempre dispone de 200m, unidad de cpu para kubernetes que significa un 20 % de un core de la máquina y 120 megas de RAM. Debido al uso de FFMPEG y la escalabilidad hemos ajustado los límites máximos a los que puede llegar usando solo un core de la máquina, 1000m y 1000 megas de RAM. Los megas de RAM son bastantes debido a que conversiones de un fichero en resolución 4K consumen muchos recursos.

4.3.3. GlusterFS

Esta aplicación es de fácil implementación y uso entre pods de kubernetes para gestionar el sistema de ficheros entre los diferentes nodos de las máquinas. Estas deben tener instalado glusterfs-server para poder usarlo en cada nodo de kubernetes.

```

apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster
  namespace: elastic-transcoder
subsets:
- addresses:
  - ip: 192.168.3.145
  ports:
  - port: 1

-----
apiVersion: v1
kind: Service
metadata:

```

```

name: glusterfs-cluster
namespace: elastic-transcoder
spec:
  ports:
    - port: 1

```

Con estos 2 manifiestos se declaran los endpoints a los que debe acceder el servicio de glusterfs para tener acceso al volumen que hemos creado. En este caso 192.168.3.145 corresponde con una máquina con un disco duro externo conectado.

```

volumes:
  - name: elastictranscoder-data
    glusterfs:
      endpoints: glusterfs-cluster
      path: /elastictranscoder-data
      readOnly: false
-----
volumeMounts:
  - name: elastictranscoder-data
    mountPath: /media/elastictranscoder-data

```

Para hacer uso de el volumen creado en glusterfs debemos invocarlo en el manifiesto anterior accediendo al servicio y montar el volumen en cada pod.

4.3.4. Frontend

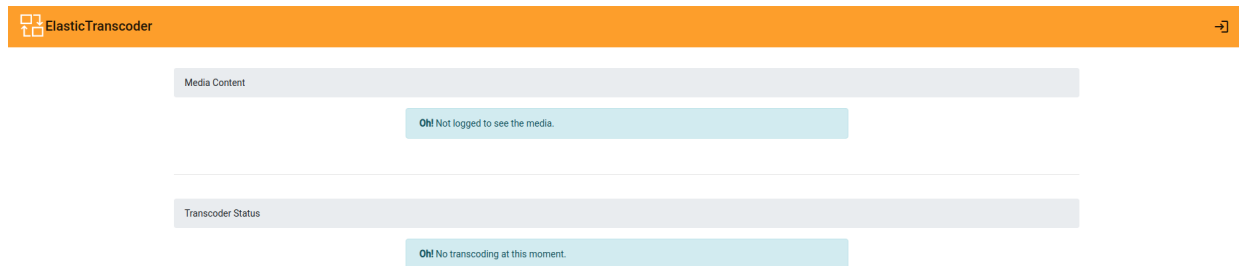


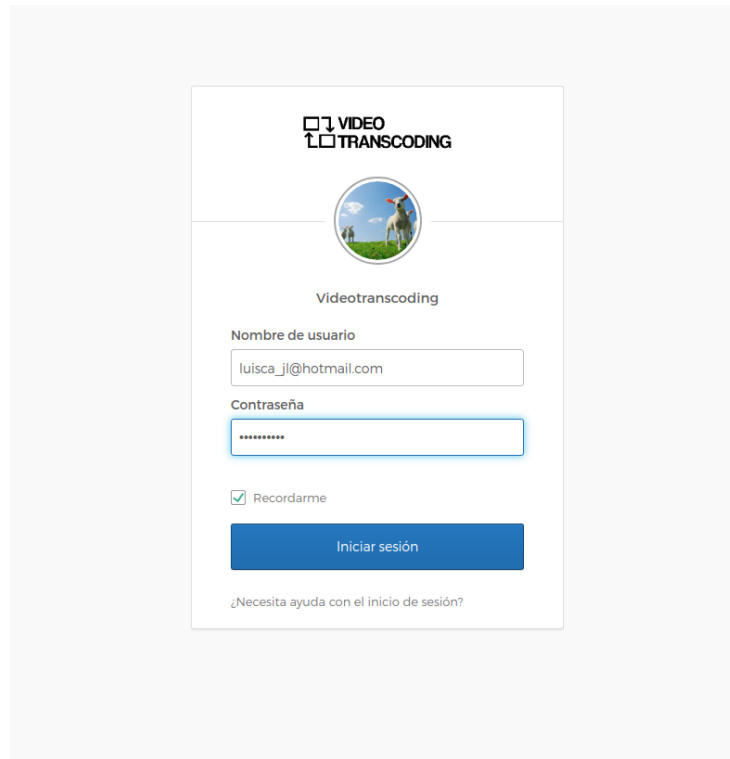
Figura 4.10: Dashboard elastictranscoder sin login

Como se puede ver en la figura 4.10, en la página principal se distinguen 2 componentes:

- **MediaContent:** Componente al que se tiene acceso al contenido propio, por lo tanto se debe estar logueado para acceder a esos datos.
- **TranscoderStatus:** Componente que muestra el estado actual de las transcodificaciones usando websocket.

Se ha usado el servicio externo de Okta para poder realizar la autorización y así poder acceder al backend con ese acceso. El frontend se comunica con el backend usando un header en cada petición

que contiene la autorización. De esta manera cuando una petición llega al backend este pregunta a Okta si este header es válido y así dar paso a la petición.



The image shows a login form for 'Videotranscoding' using OKTA. At the top, there is a logo with the text 'VIDEO' and 'TRANSCODING' flanking a central circular image of a cow in a field. Below the logo, the text 'Videotranscoding' is displayed. The form contains two input fields: 'Nombre de usuario' with the value 'luisca_jl@hotmail.com' and 'Contraseña' with masked characters. A checkbox labeled 'Recordarme' is checked. A blue button labeled 'Iniciar sesión' is positioned below the password field. At the bottom, there is a link that says '¿Necesita ayuda con el inicio de sesión?'.

Figura 4.11: Login on frontend using OKTA

OKTA provee una configuración sencilla a través de su dashboard, el cual se accede con usuario/contraseña como puede verse en la Figura 4.11, en el que modificar y añadir otros servicios de autenticación como google, github o facebook.

The screenshot shows the ElasticTranscoder dashboard. At the top, there's an orange header with the ElasticTranscoder logo and the user name 'Luis Carlos Jimenez Lopez'. Below the header, the 'Media Content' section features two buttons: 'Upload Content' and 'Reload media table'. A table lists media content with columns: Name, Type, Filesize, Duration, Container, Video Codec, Audio Codec, Resolution, and Bitrate. The table contains five rows of video files. Below the table, the 'Transcoder Status' section shows a message: 'Oh! No transcoding at this moment.'

Name	Type	Filesize	Duration	Container	Video Codec	Audio Codec	Resolution	Bitrate
file_example_MP4_1920_18MG	video	17.01 MB	30.53	MP4	H264	AAC	1920x1080	4675 kb/s
Elysium_trailer_1-4K-HDTN	video	420.27 MB	132.4	MP4	H264			26628 kb/s
PuppierBath4K	video	378.93 MB	149.63	MP4	H264	AAC		21244 kb/s
DARK-KNIGHT-RISES_TLR-F1_51-4K-HDTN	video	645.70 MB	88.8	MP4				60996 kb/s
StarWarsTrailer2	video	117.21 MB	130.55	MP4	H264	AAC	1920x816	7531 kb/s

Figura 4.12: Dashboard elastictranscoder

Una vez estamos logueados en la página principal, figura 4.12, muestra el contenido del usuario. En la esquina superior derecha podemos ver el usuario logueado. Ahí mismo puede pulsarse para desloguearse.

Disponemos de varios botones sencillos para actualizar o para subir nuevo contenido.

The screenshot shows the 'Upload Files' modal. It has a title 'Upload Files' and a yellow 'Add Files' button. Below the button, there are three file upload progress bars with their names: 'DARK-KNIGHT-RISES_TLR-F1_51-4K-HDTN', 'Elysium_trailer_1-4K-HDTN', and 'file_example_MP4_1920_18MG'. At the bottom left, there is an 'Upload' button.

Figura 4.13: Componente de carga de contenido

Como podemos ver en la figura 4.13 se puede subir varios ficheros a la vez para así poder agili-

zar esa carga. Estos ficheros los recibe el microservicio de media-storage que a su vez lo guarda en el sistema de ficheros de glusterfs. La velocidad de subida depende de los servidores donde estén alojados los servicios web y de su almacenamiento.

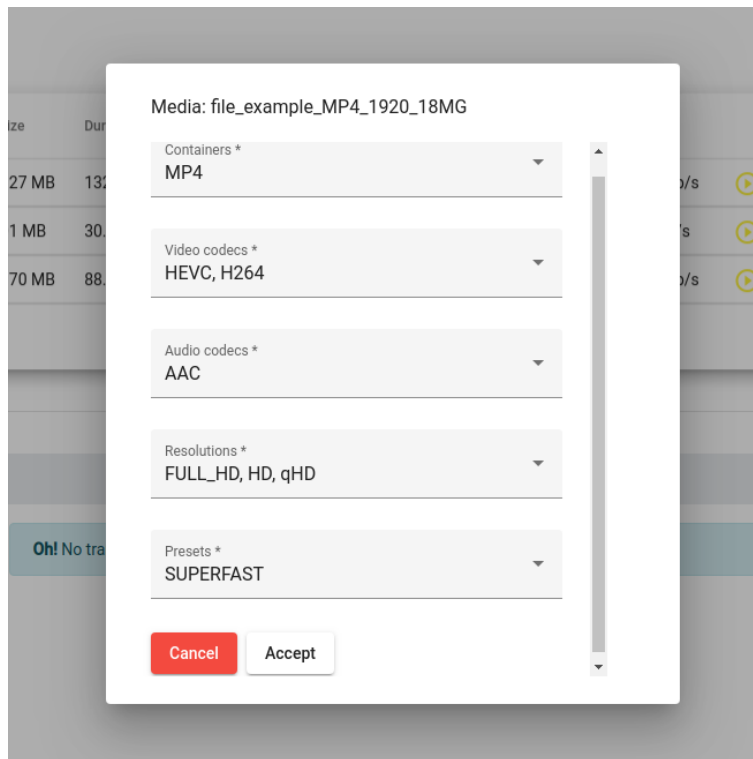
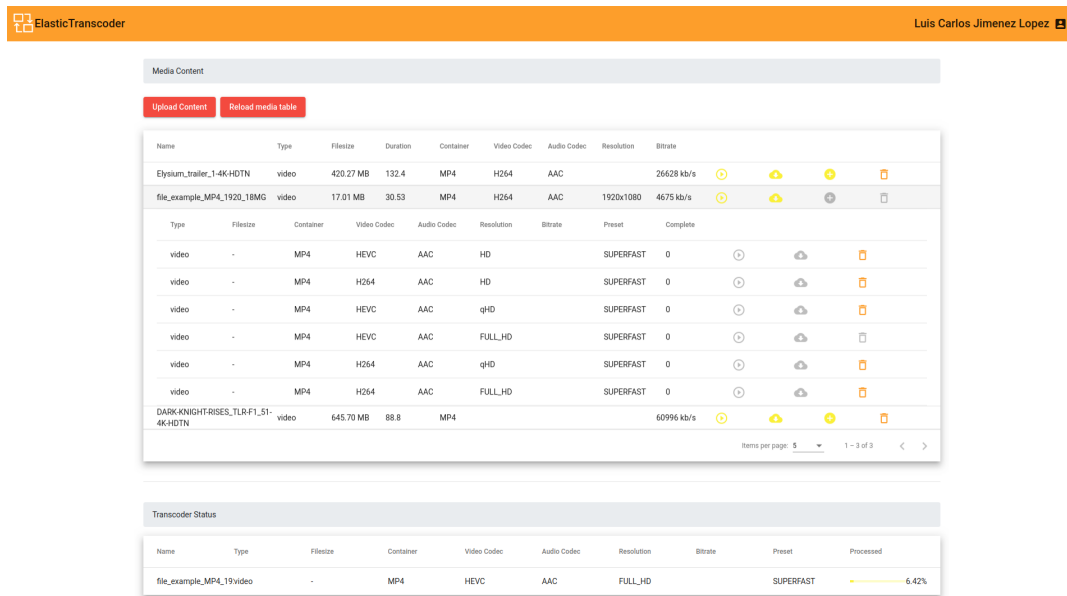


Figura 4.14: Componente para añadir transcodificaciones

Cuando se dispone de contenido multimedia al que convertir podemos hacer uso de este componente, figura 4.14, en el que podemos seleccionar el tipo de conversión a aplicar.



The screenshot shows the ElasticTranscoder web interface. At the top, there's an orange header with the ElasticTranscoder logo and the user name 'Luis Carlos Jimenez Lopez'. Below the header, there are two main sections: 'Media Content' and 'Transcoder Status'.

Media Content Section:

Buttons: Upload Content, Reload media table

Name	Type	Filesize	Duration	Container	Video Codec	Audio Codec	Resolution	Bitrate				
Elysium_trailer_1-4K-HDTN	video	420.27 MB	132.4	MP4	H264	AAC		26628 kb/s				
file_example_MP4_1920_18MG	video	17.01 MB	30.53	MP4	H264	AAC	1920x1080	4675 kb/s				

Below the table, there's a detailed view for the selected item 'DARK-KNIGHT-RISES_TLR-F1_51-4K-HDTN'.

Type	Filesize	Container	Video Codec	Audio Codec	Resolution	Bitrate	Preset	Complete			
video	-	MP4	HEVC	AAC	HD		SUPERFAST	0			
video	-	MP4	H264	AAC	HD		SUPERFAST	0			
video	-	MP4	HEVC	AAC	qHD		SUPERFAST	0			
video	-	MP4	HEVC	AAC	FULL_HD		SUPERFAST	0			
video	-	MP4	H264	AAC	qHD		SUPERFAST	0			
video	-	MP4	H264	AAC	FULL_HD		SUPERFAST	0			

Items per page: 5 | 1 - 3 of 3

Transcoder Status Section:

Name	Type	Filesize	Container	Video Codec	Audio Codec	Resolution	Bitrate	Preset	Processed
file_example_MP4_19video	-	-	MP4	HEVC	AAC	FULL_HD		SUPERFAST	6.42%

Figura 4.15: Elastictranscoder con todos sus componentes

Una vez el websocket recibe datos de la cola de transcoder-status significa que se está transcodificando contenido como puede verse en la parte inferior de la figura 4.15.

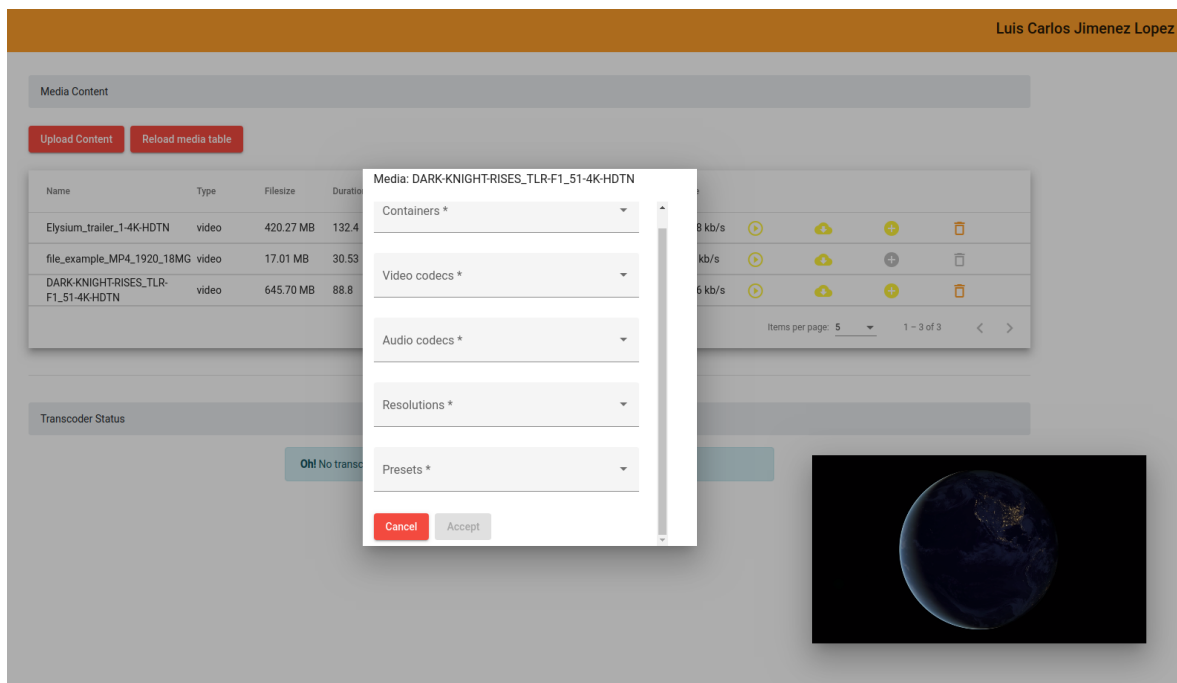


Figura 4.16: Elastictranscoder multitarea

Debido a la facilidad de angular material es sencillo implementar modales, como puede verse en la figura 4.16, que se puedan mover por toda la pantalla y que no bloqueen el acceso a otros componentes. Por ello, podemos reproducir contenido y además seguir realizando otras operaciones.

4.3.5. CloudWatch Adapter

Custom Resource Definition creados por aws labs en github y necesarios para acceder a AWS y obtener las métricas de cloudwatch³ y así conocer el tamaño de la cola SQS.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: k8s-cloudwatch-adapter
    name: k8s-cloudwatch-adapter
    namespace: custom-metrics
spec:
  replicas: 1
  selector:
    matchLabels:
      app: k8s-cloudwatch-adapter
  template:
    metadata:
      labels:
        app: k8s-cloudwatch-adapter
        name: k8s-cloudwatch-adapter
    spec:
      serviceAccountName: k8s-cloudwatch-adapter
      securityContext:
        fsGroup: 65534
      containers:
        - name: k8s-cloudwatch-adapter
          image: chankh/k8s-cloudwatch-adapter:latest
          args:
            - /adapter
            - --cert-dir=/tmp
            - --secure-port=6443
            - --logtostderr=true
            - --v=2
          ports:
            - containerPort: 6443
              name: https
            - containerPort: 8080
              name: http
          volumeMounts:
            - mountPath: /tmp
              name: temp-vol
          env:
            - name: AWS_REGION
              value: eu-west-1
            - name: AWS_ACCESS_KEY_ID
              value: {{ .Values.global.aws.accessKeyId }}
            - name: AWS_SECRET_ACCESS_KEY
              value: {{ .Values.global.aws.secretAccessKey }}
          volumes:
            - name: temp-vol
              emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: k8s-cloudwatch-adapter
  namespace: custom-metrics
spec:
  ports:

```

³Amazon CloudWatch es un servicio de monitorización y observación creado para ingenieros de DevOps, desarrolladores, ingenieros de fiabilidad de sitio (SRE) y administradores de TI.

```

- name: https
  port: 443
  targetPort: 6443
- name: http
  port: 80
  targetPort: 8080
selector:
  app: k8s-cloudwatch-adapter

```

Para poder escuchar y actualizar el estado de AWS se debe tener un pod y un servicio que sea el encargado de conectarse a AWS y obtener las métricas que se requieran. En este caso ha sido necesario la métrica sqs-transcoder-length.

```

apiVersion: metrics.aws/v1alpha1
kind: ExternalMetric
metadata:
  name: sqs-transcoder-length
spec:
  name: sqs-transcoder-length
  resource:
    resource: "deployment"
  queries:
    - id: sqs_transcoder_length
      metricStat:
        metric:
          namespace: "AWS/SQS"
          metricName: "ApproximateNumberOfMessagesVisible"
          dimensions:
            - name: QueueName
              value: "transcoder"
          period: 60
          stat: Sum
          unit: Count
    - id: sqs_transcoder_length_not_visible
      metricStat:
        metric:
          namespace: "AWS/SQS"
          metricName: "ApproximateNumberOfMessagesNotVisible"
          dimensions:
            - name: QueueName
              value: "transcoder"
          period: 60
          stat: Sum
          unit: Count
    - id: sqs_transcoder_length_sum
      expression: "sqs_transcoder_length + sqs_transcoder_length_not_visible"
      returnData: true

```

ApproximateNumberOfMessagesVisible y ApproximateNumberOfMessagesNotVisible son el nombre de las 2 métricas de las que recogemos los datos. Una vez obtenidos devolvemos la suma que es recogida por el HPA(Horizontal Pod Autoescaler) para escalar dependiendo del numero de mensajes.

4.3.6. Traefik

Traefik es un loadbalancer para kubernetes. Necesita de CRDs para funcionar correctamente ya que emplea manifiestos que no están disponibles nativamente.

```

---
apiVersion: helm.fluxcd.io/v1
kind: HelmRelease
metadata:
  name: traefik
  namespace: traefik-v2
  annotations:
    fluxcd.io/automated: "true"
spec:
  releaseName: traefik
  chart:
    git: ssh://git@github.com/luiscajl/flux-get-started
    ref: master
    path: charts/traefik
  values:
    image:
      name: traefik
      tag: 2.2.8
    deployment:
      enabled: true
      replicas: 1
      annotations: {}
      podAnnotations: {}
    ingressRoute:
      dashboard:
        enabled: true
        annotations: {}
        labels: {}
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
    additionalArguments:
      - --certificatesresolvers.myresolver.acme.tlschallenge=true
      - --certificatesresolvers.myresolver.acme.email=luisca_jl@hotmail.com
      - --certificatesresolvers.myresolver.acme.storage=/data/acme.json
    persistence:
      ## Recuerda habilitar en caso de microk8s enable storage o crea un storageclass
      enabled: true
    service:
      enabled: true
      type: LoadBalancer
      annotations: {}
      spec: {}
      loadBalancerSourceRanges: []
      externalIPs:
        - 192.168.3.150
    ports:
      traefik:
        port: 9000
        expose: false
        exposedPort: 9000
      web:
        port: 8000
        expose: true
        exposedPort: 80
      websecure:
        port: 8443
        expose: true
        exposedPort: 443
    podSecurityContext: {}

```

Este HelmRelease es el encargado de desplegar traefik a través de flux en el cluster de kubernetes y disponer de un load balancer con https a través de un DNS. Con él tenemos además la redirección a los diferentes servicios de una manera rápida y sencilla haciendo uso de IngressRoute,

CRD creado por traefik con reglas para usar https como puede verse en el siguiente código.

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: ingress-route-https
  namespace: elastic-transcoder
spec:
  entryPoints:
    - websecure
  routes:
    - match: Host('api.videotranscoding.es') && PathPrefix('/files')
      kind: Rule
      services:
        - name: media-storage
          port: 8080
    - match: Host('api.videotranscoding.es') && PathPrefix('/media')
      kind: Rule
      services:
        - name: media
          port: 8080
    - match: Host('api.videotranscoding.es') && PathPrefix('/status')
      kind: Rule
      services:
        - name: transcoder-status
          port: 8080
    - match: Host('api.videotranscoding.es') && PathPrefix('/transcodes')
      kind: Rule
      services:
        - name: transcoder-handler
          port: 8080
    - match: Host('videotranscoding.es') && PathPrefix('/')
      kind: Rule
      services:
        - name: frontend
          port: 80
  tls:
    certResolver: myresolver
```

4.3.7. Monitorización del clúster

Prometheus es una software que recoge métricas del cluster y que son recogidas en la interfaz gráfica de grafana. Con ello podemos ver valores de los diferentes nodos entre los que destacar el sistema de ficheros, el uso de CPU por cada pod o la RAM que está consumiendo cada pod. Pueden verse estos datos en la figura 4.17.

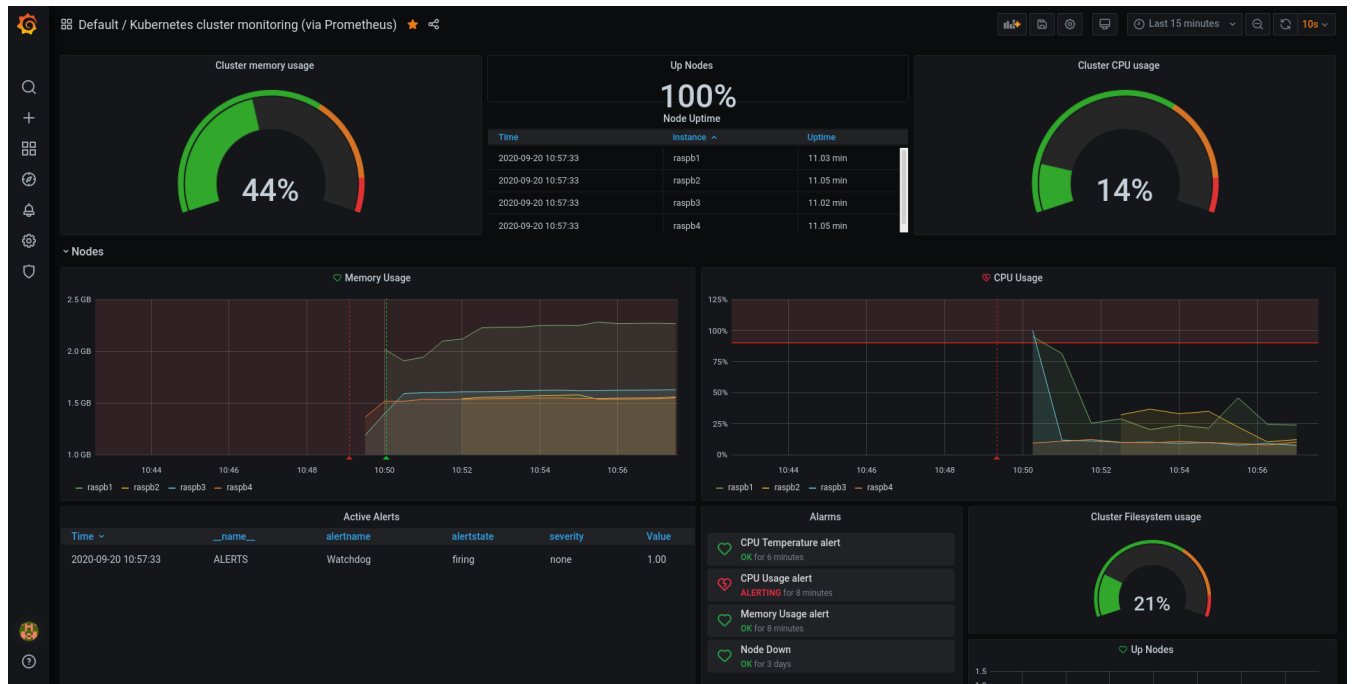


Figura 4.17: Dashboard Grafana usando datos de prometheus

Para poder configurar todos estos parámetros y poder mostrar estas gráficas se ha usado un proyecto open source en github, cluster-monitoring, que es un fork de kube-prometheus.

Capítulo 5

Conclusiones

Actualmente las aplicaciones que existen en el mercado presentan numerosas dificultades para los usuarios finales. Aquellos que tienen pocos conocimientos sobre metadatos de video se ven bastante imposibilitados a la hora de convertir un fichero de video o audio debido a la multitud de elecciones que deben hacer. Además, estas aplicaciones, presentan problemas de compatibilidad con otros dispositivos a causa de los diferentes códecs disponibles. En este sentido, lo que consigue nuestra aplicación es un proceso elástico y rápido, ya que las máquinas se adaptan a la carga de trabajo. Mejoramos la experiencia del usuario con una interfaz gráfica fácil de usar, reduciendo los tiempos, con más tolerancia a los fallos y con menos elecciones de metadatos a la hora de convertir un fichero.

El objetivo principal de la aplicación era poder transcodificar en un entorno elástico, que se ha cumplido superando las expectativas. El resultado es un código legible, con una buena automatización y escalabilidad. Además, se ha externalizado los mecanismos de autenticación de forma que no se tiene que invertir tiempo en realizar uno propio. La transcodificación de audio ha quedado en un segundo plano, debido a que los tiempos eran ajustados. Una de las principales dificultades que hemos encontrado, era que los videos de alta calidad contienen varias pistas de audio en diferentes idiomas. Debemos tratar cada una de ellas por separado, convirtiéndolas todas al mismo formato, para finalmente asignárselo al video transcodificado.

El grado de aspectos que se han utilizado para este proyecto han sido altos tanto en backend como frontend acabando con el despliegue de las aplicaciones en máquinas físicas. Al tratarse de una aplicación elástica tienen que colaborar estrechamente todas las partes de la aplicación. Por consiguiente, se ha escogido el uso de kubernetes para levantar diferentes instancias de una misma parte de la aplicación y así conseguir la elasticidad.

El desarrollo de este proyecto ha tenido como consecuencia el aprendizaje de tecnologías como Kubernetes, Traefik o Glusterfs. Kubernetes, por su parte, está muy asentado en el mercado y está considerado como el futuro para desarrollos escalables y tolerantes a fallos. Por otra parte, Traefik, es otro punto de vista con respecto a Ingress y es interesante conocer su funcionamiento y la au-

to creación de certificados SSL usando LetsEncrypt¹. GlusterFs da el total soporte a este proyecto que acaba con el total acceso desde cualquier nodo al sistema de ficheros.

¹ Autoridad de Certificación gratuita, automatizada, y abierta

Bibliografía

- [1] What Is Java?
<https://www.thoughtco.com/what-is-java-2034117>
- [2] Shell de Unix
<https://bit.ly/2uTGNsJ>
- [3] Spring Framework - Overview
<https://bit.ly/2LuWwlg>
- [4] Spring Boot – Simplifying Spring for Everyone
<https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone>
- [5] Typescript
<https://bit.ly/2sMvwpz>
- [6] ¿Que es Angular y para que sirve?, *Victor Robles*
<https://victorroblesweb.es/2017/08/05/que-es-angular-y-para-que-sirve/>
- [7] ¿Qué es y para qué sirve HTML?
<https://bit.ly/2Jg2cCV>
- [8] CSS, ¿qué es?
<http://www.arumeinformatica.es/dudas/css/>
- [9] Acerca del control de versiones
<https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones>
- [10] Github
<https://en.wikipedia.org/wiki/GitHub>
- [11] FFMPEG
<https://www.ffmpeg.org/about.htmlb>
- [12] Visual Studio Code
<https://bit.ly/2JKi78H>
- [13] Visual Studio Code
<https://www.genbetadev.com/java-j2ee/introduccion-a-maven>

- [14] Package Manager
<https://www.sitepoint.com/beginners-guide-node-package-manager/>
- [15] Explicación de la integración continua
<https://aws.amazon.com/es/devops/continuous-integration/>
- [16] ¿Qué es docker?
<https://aws.amazon.com/es/docker/>
- [17] Comparison of video containers, *Wikipedia*
<https://bit.ly/31YppRY>
- [18] HLS, Streaming de tasa de bits adaptable *Wikipedia*
<https://bit.ly/33NkKSD>
- [19] Que es Spring Boot y su relación con los microservicios *Oscar Blancarte*
<https://bit.ly/33NkKSD>
- [20] Traefik *Traefik*
<https://docs.traefik.io/>
- [21] Kubernetes *Kubernetes*
<https://kubernetes.io/>
- [22] Flux *Flux*
<https://fluxcd.io/>
- [23] Grafana *Grafana*
<https://grafana.com/>
- [24] Okta *Okta*
okta.com
- [25] Cluster monitoring using grafana and prometheus
<https://github.com/carlosedp/cluster-monitoring>, *Carlos Eduardo*
- [26] Proyecto Github
<https://github.com/MasterCloudApps-Projects/ElasticTranscoder>, *Luis Carlos Jiménez López*