UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESIS DOCTORAL PARA EL GRADO DE DOCTOR EN CIENCIAS INFORMÁTICAS

PLATAFORMA COLABORATIVA, DISTRIBUIDA, ESCALABLE Y DE BAJO COSTO BASADA EN MICROSERVICIOS, CONTENEDORES, DISPOSITIVOS MÓVILES Y SERVICIOS EN LA NUBE PARA TAREAS DE CÓMPUTO INTENSIVO

AUTOR
Lic. David Marcelo Petrocelli

DIRECTOR
Dr. Marcelo Naiouf

DIRECTOR
Ing. Armando De Giusti

La Plata, 2021

1. Introducción 14
1.1 - Objetivos 18
1.2 - Metodología de trabajo 18
1.3 - Contribuciones 21
1.4 - Infraestructura de experimentación 23
1.5 - Publicaciones 24
1.6 - Organización de la tesis 25
2. Marco Teórico 27
2.1 - Computación de alto rendimiento (HPC) 27
2.2 - Sistemas Distribuidos 28
2.2.1 - Arquitecturas, herramientas y tecnologías para el desarrollo de un Sistema Distribuido para resolver tareas HPC 29
2.3 - Microservicios 30
2.4 - Cultura y Técnicas DevOps 32
2.4.1 - Automatización en DevOps 33
2.4.2 - Infraestructura como código 34
2.4.3 - Integración y despliegue continuo 35
2.4.4 - Monitoreo Continuo 36
2.5 - Computación en la Nube 37
2.5.1 - Modelos de implementación 38
Nube pública 38
Nube privada 39
Nube híbrida 39
2.5.2 - Modelos de Entrega de Servicios 40
Infraestructura como Servicio (IaaS) 40
Plataforma como Servicio (PaaS) 40
Software como Servicio (SaaS) 41
Otros Submodelos: 41
2.6 - Virtualización basada en contenedores 41
2.6.1 - Hipervisores 43
2.6.2 - Contenedores 44
2.6.4 - Docker como plataforma de contenedores 45
Componentes esenciales de Docker 47
Imágenes Docker 47
Dockerfile 47

Contenedor Docker 47
Registro de Docker 48
2.7 - Orquestación de Contenedores 48
2.7.1 - Kubernetes como plataforma de Orquestación de contenedores 49
Capacidades Básicas de Kubernetes: 49
Razones para adoptar Kubernetes 50
Desafíos para la adopción de Kubernetes 51
Objetos básicos de Kubernetes 51
Pod 52
Controlador de replicación (ReplicaSet) 52
Despliegues (Deployments) 52
Servicios (Services) 52
Arquitectura de Kubernetes 53
Componentes del nodo maestro 53
Componentes del nodo trabajador 54
2.8 - Dispositivos móviles: Smart Devices 55
2.8.1 - Introducción 55
2.8.2 - Características básicas de un dispositivo inteligente 56
2.8.3 - Android como Sistema Operativo 57
Arquitectura del SO Android 58
2.8.4 - Arquitectura de procesadores ARM y masividad de dispositivos 60
3. - Diseño e Implementación de la plataforma HPC 62
3.1 - Objetivo 62
3.2 - Descripción general 63
3.3 - Arquitectura de la plataforma HPC 64
3.3.1 - Diseño y características de la plataforma 64
Componentes básicos de la plataforma 66
(a) Usuarios interesados 66
(b) Recursos de administración de la plataforma 66
(c) Recursos de procesamiento 71
3.3.2 - Desarrollo e implementación de la plataforma 73
a) Microservicios Dockerizados para la administración y gestión de la plataforma 74
b) Servicios de colas (tareas) y base de datos (estadísticas) Dockerizadas 84
c) Orquestación, escalado, replicación y automatización de contenedores a través de Kubernetes 86
Alta disponibilidad y portabilidad 88

Gestión de la configuración, automatización y reversiones 91
Mecanismos de Transparencia 106
d) Trabajadores móviles Android ARM y x86 Dockerizados 108
Características Generales 108
Características del nodo trabajador x86 Dockerizado 109
Características del nodo trabajador móvil 112
3.4 - Conclusiones del capítulo 120
4. Caso de aplicación: Compresión de video 121
4.1 - Objetivos 122
4.2 – El Servicio de Streaming 122
4.2.1 - Protocolos de Streaming 123
Streaming Tradicional 124
Hacia Streaming basado en HTTP 125
Descarga Progresiva 125
Streaming de Video Adaptativo (ABR) basado en HTTP 126
4.2.2 - Codificación 128
Digitalización 128
Codificación 132
Configuraciones asociadas para recodificar videos 143
4.3 - Compresión de vídeo utilizando FFmpeg 148
4.3.1 - FFMpeg como herramienta de recodificación de video 151
4.3.2 - Perfiles de codificación H.264 en FFmpeg 152
4.3.3 - Proceso de Fragmentación y Unificación de archivos 156
4.3.4 - Proceso de Transcodificación 158
4.4 - Integración de FFmpeg en microservicios de la plataforma 159
a) Microservicios Dockerizados para la administración y gestión de la plataforma 160
b) Servicios de colas (tareas) y base de datos (estadísticas) Dockerizadas 163
c) Orquestación, escalado, replicación y automatización de contenedores a través de Kubernetes 163
d) Trabajadores móviles Android ARM y x86 Dockerizados. 163
4.5 - Repositorio de código fuente 165
4.6 - Conclusiones del capítulo 165
5 - Modelo de análisis y experimentación 167
5.1 - Objetivos 167
5.2 - Descripción de la fase experimental 167
5.2.1 - Evaluación de rendimiento y consumo energético de los nodos trabajadores 168

Definición de escenarios y equipamiento de prueba 169
Definición de métricas a capturar 171
Ejecución de pruebas y análisis de resultados 173
Análisis de resultados de la arquitectura x86 174
Análisis de resultados de la arquitectura ARM 177
Análisis comparativo entre arquitecturas 180
Conclusiones 181
5.2.2 - Evaluación de escalabilidad y flexibilidad de la plataforma 182
Definición de escenarios y equipamiento de prueba 182
Ejecución de pruebas y análisis de resultados 187
Conclusiones 198
6. Conclusiones 199
7. Trabajos Futuros 200
ANEXO I - Despliegue de la plataforma en Kubernetes 202
Arquitectura básica del proyecto 202
Despliegue de un Cluster de Kubernetes 203
Organizar y establecer el acceso del clúster para kubectl 205
Despliegue de reglas y estructura de servicios en el clúster 206
Despliegue de servicios de soporte 208
Actualizaciones y despliegue de código (funciones) 208
ANEXO II - Dispositivos de medición de consumo energético 212
Medición en arquitecturas x86 212
Componente 1 (C1) - Pinza SCT 013-030 212
Componente 2 (C2) - Módulo Arduino ESP8266 nodemcu v3 213
Componente 3 (C3) - Alargue de tensión 214
Componente 4 (C4) - Multímetro True RMS ProsKit MT-1706 214
Construcción y funcionamiento del prototipo de medición 215
Medición en arquitecturas ARM Android 221
Glosario 222
Referencias 231
Microservicios 231
DevOps 232
Virtualización, Contenedores y Orquestadores de Contenedores 235
Sistemas y plataformas de procesamiento distribuido 240
Arquitecturas y sistemas HPC 243
Análisis de consumo de Energía y Computación Verde 246

Computación en la Nube 249
Computación Móvil y procesamiento basado en ARM 254
Video digital, códecs, streaming y transcodificación de video 257

Palabras clave:
Computación en la Nube; Computación en el borde; Sistemas Distribuidos; Contenedores y Orquestador de Contenedores (Kubernetes); DevOps; ARM; Android; Transcodificación de video

Agradecimientos

Es un orgullo y una gran felicidad para mí saber que hoy lograré uno de mis sueños más grandes, que el esfuerzo que hice cada año al fin tendrá una recompensa. En este largo trayecto, he conocido gente maravillosa con la que he compartido buenos momentos y de quienes he aprendido cosas valiosas. El tiempo ha pasado, fui madurando, aprendiendo cosas extraordinarias, creciendo como ser humano, emocional, física, espiritual y profesionalmente. Por eso, me gustaría que estas líneas sirvan para expresar mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización del presente trabajo.

En primer lugar, doy gracias a mis padres porque me dieron la vida y un encantador hogar. Gracias por brindarme la oportunidad de aprender de su esfuerzo diario y haberlo podido traducir en mis propios deseos y desafíos. Gracias por creer en mí, por brindarme sus consejos, su apoyo incondicional y todo su amor. A mi padre, gracias por haber luchado contra todo lo difícil que te tocó y gracias por estar hoy presente acá conmigo. Te lo prometí y lo estoy compartiendo hoy con vos, me siento inmensamente feliz. A mi madre, gracias por tu presencia incondicional en cada etapa de mi vida. Gracias por recordarme que no aflojara nunca, que lo lograría. Acá estoy y es gracias a vos. A ambos, gracias.

Agradezco a mi persona especial, quien llegó en el 2016 a mi vida y nunca dejó de hacerme compañía. Siempre positiva, con una sonrisa, con un consejo, con la palabra justa y adecuada en el momento preciso, por ser además de mi novia, mi mejor amiga. Gracias por complementarme en los diversos aspectos de la vida y por mostrarme cómo ser una mejor persona y un mejor profesional. Gracias por enseñarme a valorar cada uno de mis logros, por apoyarme todos estos años. Por sobre todas las cosas, gracias por soportar verme tantas horas frente a la computadora y estar ahí conmigo. Esto también es gracias a vos.

Agradezco a mis amigos, que siempre me acompañaron a seguir adelante con el estudio, sin importar lo que se interponga. Agradezco que hayan podido comprender la falta de tiempo en algunas reuniones y cumpleaños, las llegadas tardes a los partidos, retirarse antes de alguna fiesta, y miles de cosas que solo ustedes pueden comprender.

Agradezco a la Universidad Nacional de La Plata por darme la oportunidad de estudiar y ser hoy, después de un arduo camino, un profesional de su casa. En particular, me gustaría agradecer a mis tutores de tesis que me dieron soporte en cada momento para avanzar con todas las fases de este proyecto. Gracias por sus consejos, sus enseñanzas y sobre todo por su preciado tiempo. Su apoyo y confianza en mi trabajo y su capacidad para guiarme ha sido un aporte invaluable, no solamente en el desarrollo de esta tesis, sino también en mi formación como estudiante, docente, e investigador.
Me gustaría también agradecer a un gran compañero, amigo y quien fuera tutor de mi tesis de grado en la Universidad Nacional de Luján, Gabriel Tolosa. No solo me dirigió en ese momento, sino que, durante este ciclo de posgrado, siempre estuvo presente haciendo recomendaciones y críticas constructivas.

¡Para todos aquellos que he mencionado en este apartado muchas gracias!

RESUMEN

A la hora de resolver tareas de cómputo intensivo de manera distribuida y paralela, habitualmente se utilizan recursos de hardware x86 (CPU/GPU) e infraestructura especializada (Grid, Cluster, Nube) para lograr un alto rendimiento. En sus inicios los procesadores, coprocesadores y chips x86 fueron desarrollados para resolver problemas complejos sin tener en cuenta su consumo energético.

Dado su impacto directo en los costos y el medio ambiente, optimizar el uso, refrigeración y gasto energético, así como analizar arquitecturas alternativas, se convirtió en una preocupación principal de las organizaciones. Como resultado, las empresas e instituciones han propuesto diferentes arquitecturas para implementar las características de escalabilidad, flexibilidad y concurrencia.

Con el objetivo de plantear una arquitectura alternativa a los esquemas tradicionales, en esta tesis se propone ejecutar las tareas de procesamiento reutilizando las capacidades ociosas de los dispositivos móviles. Estos equipos integran procesadores ARM los cuales, en contraposición a las arquitecturas tradicionales x86, fueron desarrollados con la eficiencia energética como pilar fundacional, ya que son mayormente alimentados por baterías. Estos dispositivos, en los últimos años, han incrementado su capacidad, eficiencia, estabilidad, potencia, así como también masividad y mercado; mientras conservan un precio, tamaño y consumo energético reducido. A su vez, cuentan con lapsos de ociosidad durante los períodos de carga, lo que representa un gran potencial que puede ser reutilizado.

Para gestionar y explotar adecuadamente estos recursos, y convertirlos en un centro de datos de procesamiento intensivo; se diseñó, desarrolló y evaluó una plataforma distribuida, colaborativa, elástica y de bajo costo basada en una arquitectura compuesta por microservicios y contenedores orquestados con Kubernetes en ambientes de Nube y local, integrada con herramientas, metodologías y prácticas DevOps.

El paradigma de microservicios permitió que las funciones desarrolladas sean fragmentadas en pequeños servicios, con responsabilidades acotadas. Las prácticas DevOps permitieron construir procesos automatizados para la ejecución de pruebas, trazabilidad, monitoreo e integración de modificaciones y desarrollo de nuevas versiones de los servicios. Finalmente, empaquetar las funciones con todas sus dependencias y librerías en contenedores ayudó a mantener servicios pequeños, inmutables, portables, seguros y estandarizados que permiten su ejecución independiente de la arquitectura subyacente. Incluir Kubernetes como Orquestador de contenedores, permitió que los servicios se puedan

administrar, desplegar y escalar de manera integral y transparente, tanto a nivel local como en la Nube, garantizando un uso eficiente de la infraestructura, gastos y energía.

Para validar el rendimiento, escalabilidad, consumo energético y flexibilidad del sistema, se ejecutaron diversos escenarios concurrentes de transcoding de video. De esta manera se pudo probar, por un lado, el comportamiento y rendimiento de diversos dispositivos móviles y x86 bajo diferentes condiciones de estrés. Por otro lado, se pudo mostrar cómo a través de una carga variable de tareas, la arquitectura se ajusta, flexibiliza y escala para dar respuesta a las necesidades de procesamiento.

Los resultados experimentales, sobre la base de los diversos escenarios de rendimiento, carga y saturación planteados, muestran que se obtienen mejoras útiles sobre la línea de base de este estudio y que la arquitectura desarrollada es lo suficientemente robusta para considerarse una alternativa escalable, económica y elástica, respecto a los modelos tradicionales.

SUMMARY

When solving compute-intensive tasks in a distributed and parallel way, x86 hardware resources (CPU / GPU) and specialized infrastructure (Grid, Cluster, Cloud) are commonly used to achieve high performance. In its early days, x86 processors, co-processors, and chips were developed to solve complex problems regardless of their power consumption.

Due to its direct impact on costs and the environment, optimizing use, cooling and energy consumption, as well as analyzing alternative architectures, it has become a relevant concern of organizations. As a result, companies and institutions have proposed different infrastructures to implement scalability, flexibility, and concurrency features.

To propose an alternative architecture to traditional schemes, this thesis aims to execute the processing tasks by reusing the idle capacities of mobile devices. These gadgets integrate ARM processors which, in contrast to traditional x86 architectures, were developed with energy efficiency as their cornerstone, since they are mostly powered by batteries. These devices, in recent years, have increased their capacity, efficiency, stability, processing power, as well as massiveness and market; while maintaining a low price, size and energy consumption. At the same time, they face idle periods during recharging battery lapses, which represents a powerful potential that can be reused.

To properly manage and take advantage of this processing capacity and turn them into a processing-intensive data center; a distributed, collaborative, elastic and low-cost platform was designed, developed and evaluated based on microservices and containers orchestrated (Kubernetes) in Cloud and local environments, integrated with DevOps tools, methodologies and practices architecture.

The microservices paradigm allowed the developed features to be fragmented into smaller services, with limited responsibilities. DevOps practices facilitate the building of automated processes for the execution of tests, traceability, monitoring and integration for changes and new features and improvements. Finally, packaging the features including all their dependencies and libraries in containers helped to keep services small, immutable, portable, secure and standardized that allow their execution independent of the underlying architecture. Including Kubernetes as a Container Orchestrator, allowing services to be managed, deployed and scaled comprehensively and transparently, both locally and in the Cloud, ensuring efficient usage of the infrastructure, costs and energy.

To validate the system’s performance, scalability, power consumption and flexibility, several concurrent video transcoding scenarios were executed. On the one hand, it was

possible to test the behavior and performance of lots of mobiles and x86 devices running tasks under various stress conditions. On the other hand, it was shown how the architecture adjusted and scaled to face the processing needs.

The experimental results, based on the performance, load and saturation scenarios, showed that useful improvements are obtained following the baseline of this study and that the architecture developed is robust enough to be considered as a scalable, elastic, and cheaper alternative compared to traditional models.

1. Introducción

La computación de alto rendimiento (HPC) se refiere a aquellas arquitecturas computacionales que implican la agregación de potencia de cálculo en múltiples nodos para obtener un rendimiento superior al que ofrecen los equipos individuales. De esta manera, las infraestructuras HPC son capaces de procesar grandes cantidades de datos en muy poco tiempo. Esto es ampliamente utilizado en el campo de la investigación y la industria cuando se deben resolver problemas complejos [STE17][BOL12][STR01]. Por lo ejemplo, se utilizan para simulaciones de física y química [OZO17][WOO11], análisis de redes sociales [JIN12], biomedicina [CAL15], salud [COK17] y aplicaciones médicas [ALB19], gestión y mejoras en actividades de cadenas de producción [KES18], inteligencia militar [MOL19], predicciones del clima [WAN18] y desastres ambientales [ROD18][GAU15], optimización de transportes [VOL17] y dinámica de fluidos [THA18][CAN02], entre otros.

Para su construcción, en general, estos sistemas utilizan procesadores (CPU) y aceleradores (GPU) de arquitectura x86 con múltiples núcleos sobre configuraciones de tipo Grid [XUA09][DER07], Cluster [STO18][XIN97], Cloud [MAL19][RUI14], y/o entornos híbridos [PRA18b][EME16] masivamente paralelas y distribuidas.

Problemática del consumo energético:

En su origen, los procesadores x86 fueron desarrollados para resolver problemas complejos con un alto costo por unidad [FLA20], alta disipación de temperatura y sin preocuparse primordialmente en su consumo energético [AND20]. Según los estudios de Nuoa Lei y equipo [NUO20] y Priya Mahadevan [MAH11], en un centro de datos el gasto de energía se debe en mayor proporción a los servidores y los sistemas de refrigeración. Sobre las mismas bases, Zhe Song [SON15] afirma en su investigación que aproximadamente el 40% de la energía total se consume para enfriar los equipos de TI. Por su parte, el estudio de Olle Svanfeldt-Winter y su equipo [SVA11] menciona que el 57% de los costos mensuales de un centro de datos se debe a las facturas de electricidad de los servidores.

Según Jonathan Koomey [KOO11], la electricidad total utilizada por los centros de datos, globalmente en 2010 fue aproximadamente el 1,3% de toda la consumida en el mundo; y cerca del 2% de los Estados Unidos. En ese punto, Arman Shehabi y su equipo [SHE16] explicaron que, si la eficiencia energética de los centros de datos no mejoraba, se predecía que para 2011 el consumo superaría los 100 mil millones de kWh. Además, este incremento sostenido tendría, entonces, un impacto significativo en los millones de toneladas de emisiones de gases generadas por la producción eléctrica.

Afortunadamente durante la última década, con el crecimiento exponencial de los centros de datos, los gigantes de los semiconductores entendieron estos problemas y comenzaron a abordar seriamente la disipación de temperatura (TDP) de sus chips, optimizar sus arquitecturas para otorgar un mejor rendimiento (IPC) y reducir su consumo de energía [FLA20][KOU20][BOU16]. Arman Shehabi y su equipo [SHE18] mostraron que, mientras se mantiene el mismo consumo de energía global de 2010 cómo se muestra en imagen de la derecha de la figura 1.1, las instancias de cómputo de los centros de datos aumentaron en un 550% durante el mismo período de tiempo, cómo se muestra en la imagen de la izquierda de la figura 1.1.

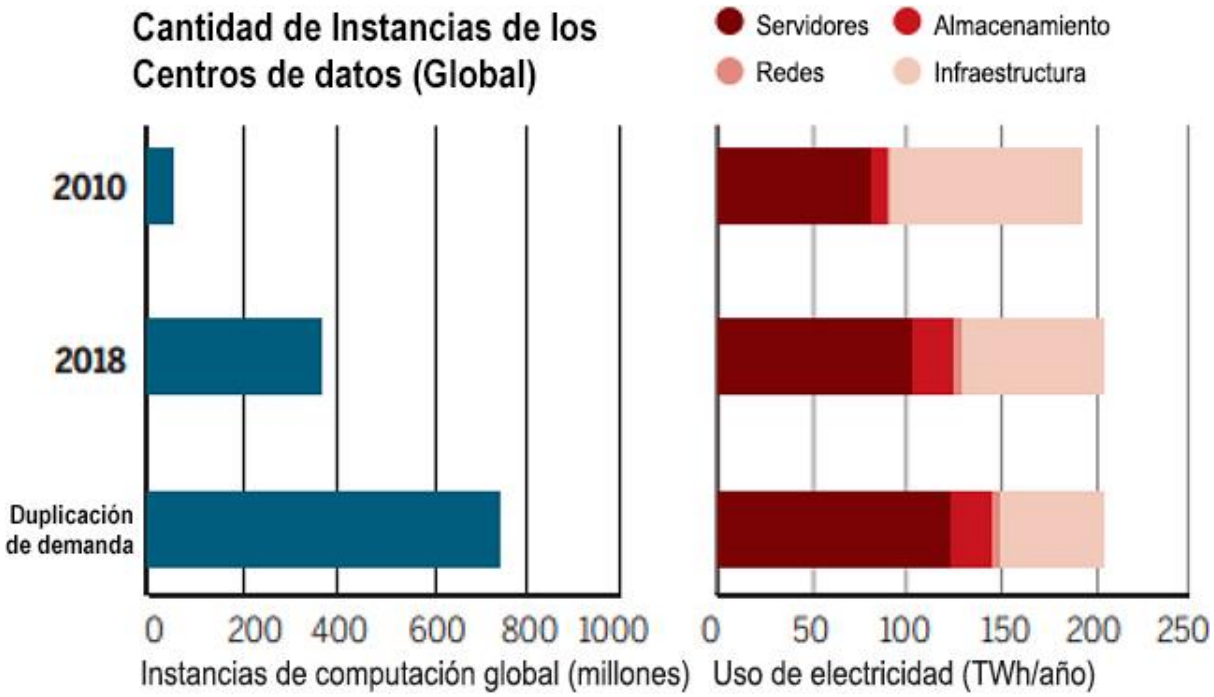

Imagen 1.1 - A la izquierda, instancias de cómputo de los centros de datos a nivel global (2010 a 2018). A la derecha, el consumo de estos centros de datos en el mismo período de tiempo. Fuente: Propia. Versión adaptada de [MAS20].

Sobre estas bases presentadas, se comprende que el **consumo y la eficiencia energética son dos temas clave a la hora de diseñar cualquier arquitectura informática en la actualidad.** Se reconoce que el logro de los objetivos de cualquier compañía o institución debe estar alineado con la optimización de los recursos y el impacto ambiental. Esto no solo es interesante por motivos ecológicos y ambientales (área conocida bajo el término de “Green Computing” [SAH18][MOD15]), sino también para reducir los costos por consumo.

Arquitecturas alternativas al centro de datos tradicional:

Cómo se describió previamente, la composición de las arquitecturas HPC ha estado basada durante un largo tiempo por procesadores x86. Sin embargo, dada la creciente diversidad de cargas de trabajo junto con la evolución de las placas gráficas y la desaceleración de las

mejoras de rendimiento de este tipo de procesadores respecto de los procesadores de su competencia [THE20], han llevado a los diseñadores de aplicaciones y operadores de infraestructura a buscar y utilizar alternativas.

En esta área, a diferencia de los procesadores x86, los chips basados en ARM fueron desarrollados y construidos, desde sus inicios, con la eficiencia energética como prioridad [CRI19][PRA19]. Estos procesadores estaban originalmente orientados a ser utilizados en los dispositivos móviles y micro dispositivos, que funcionan, en su mayor parte, con baterías. Por otro lado, ARM además de ser el principal alimentador de equipos móviles, desde hace algunos años, ha estado avanzando en su participación en los centros de datos, la Nube e incluso en las infraestructuras HPC [MAN20][RUI18][SMI18][BAN17][RIC17]. Varias investigaciones [JAC19][YOK19][KMI18][JAR13][OUZ12], mostraron que el uso de este tipo de procesadores en los centros de datos es una buena opción cuando se necesita eficiencia energética y reducción de costos sin sopesar rendimiento. Por su parte, cómo muestra el sitio oficial de ARM [ARM21] y la publicación digital del New York Times [CLA20], los proveedores de la Nube como Amazon, Microsoft y el productor de equipos Mac (Apple) han comenzado a adoptar arquitecturas ARM y, en algunos casos, a desarrollar sus propias versiones, con la premisa de bajar consumos y costos sin perder performance.

Al mismo tiempo, algunos investigadores comenzaron a estudiar y proponer soluciones alternativas para algunos tipos de problemas, utilizando arquitecturas híbridas. Para ello, reparten las responsabilidades, por un lado, en los centros de datos tradicionales y por otro, en los dispositivos de procesamiento en el borde [LIU19]. Entre las primeras iniciativas, se encuentra el trabajo de Mustafa Arslan y su equipo [ARS12] quienes muestran, que existe una gran cantidad de teléfonos inteligentes inactivos cuando se conectan para recargar baterías y además que cuentan con crecientes capacidades de cómputo. Presentaron un prototipo inicial de la arquitectura distribuida y lo evaluaron a través de un set de pruebas que corrió sobre 18 dispositivos móviles.

Extendiendo el análisis anterior, Pavel Mach y Zdenek Becvar [MAC17], y Giacomo Massari y su equipo [MAS16] estudiaron los casos de aplicación, escenarios y arquitecturas posibles para el aprovechamiento de las capacidades de la computación móvil en el borde.

Mariela Curiel y su equipo [CUR18] presentaron una plataforma para el procesamiento paralelo de imágenes médicas, utilizando cómputo móvil sobre una arquitectura GRID, basada en el proyecto BOINC [AND04]. Más allá de las ventajas, BOINC solo está probado con algunas arquitecturas ARM y no dispone de un cliente nativo para Android, por lo que adicionalmente debieron realizar una adaptación (Android NDK). Por su parte, Erick Lavoie y Laurie Hendren [LAV19] proponen una plataforma alternativa siguiendo el modelo de computación voluntaria. Su trabajo define un sistema que explota la cantidad de dispositivos

disponibles, a través de funciones JavaScript que se ejecutan en los navegadores web y que permite la inclusión de nuevos servicios distribuidos de manera incremental y modular.
Por último, Hend Gedawy y su equipo [GED20] desarrollaron una plataforma abierta que incluye dispositivos IoT y móviles heterogéneos. En cuanto a estructura, plantean dos componentes principales: nodos controladores y de procesamiento, que corren tareas simuladas. Su principal aporte está relacionado con los algoritmos de reducción y minimización de latencia.

Teniendo en cuenta estos antecedentes, en este trabajo se propone una nueva plataforma basada en una arquitectura híbrida, en la que las actividades de administración de la plataforma y gestión de las tareas se realizan utilizando recursos de centros de datos tradicionales y todas las tareas de procesamiento se ejecutan en los dispositivos de borde. De esta manera, la plataforma aprovecha la disponibilidad masiva y potencia de cálculo disponible en los dispositivos móviles de usuarios finales y la robustez ofrecida por servidores corriendo en la Nube, empresa o institución.

Para que el sistema se ajuste a la carga de manera flexible y optimice el manejo de los recursos y los costos, la plataforma se diseñó, desarrolló y desplegó utilizando una arquitectura compuesta por microservicios y contenedores orquestados con Kubernetes, en ambientes de Nube y local, e integrada con herramientas, metodologías y prácticas DevOps. Por su parte, las tareas de cálculo intensivo se ejecutan sobre los dispositivos móviles en desuso (reciclados) y ociosos mientras se encuentran recargando sus baterías. De esta manera, se reducen las tareas que los servidores centrales deben realizar, optimizando los costos y consumo energético, aprovechando y reutilizando las capacidades ya disponibles.

Para la creación de los nodos de procesamiento móviles, en vez de utilizar un navegador como medio de interacción con las tareas de procesamiento y/o versiones adaptadas de un producto de terceros, se desarrolló un módulo nativo Android, que permite optimizar los recursos disponibles en los celulares. Con el objetivo de garantizar la fluidez y no afectar la experiencia del usuario, sólo se utiliza el dispositivo cuando se cumplen dos condiciones: que el equipo se encuentre conectado a la red eléctrica con su carga de batería completa y que el dispositivo esté conectado a la red Wi-Fi.

Para validar las características de rendimiento, disponibilidad, flexibilidad y escalabilidad de la plataforma, se construyeron diversos escenarios de evaluación que permiten corroborar que el sistema responde correctamente ante las necesidades del cómputo HPC. Como tarea de evaluación se adoptó la transcodificación de vídeo. Esto permitió enfrentarse a una actividad exigente, en cuanto a poder de cómputo distribuido, que otros autores [YOO19][LIX18][BAR16][PEN16] ya propusieron, analizaron y utilizaron en arquitecturas similares.

1.1 - Objetivos

El objetivo de esta tesis es diseñar, desarrollar y evaluar una plataforma distribuida, escalable, flexible, tolerante a fallos y de bajo costo para procesar tareas de cómputo intensivo. Sobre estas bases, se pretende avanzar en el campo de las estrategias de resolución de trabajos HPC y probar que el enfoque de esta arquitectura permite conducir a un mejor uso de los recursos computacionales disponibles. Especialmente en entornos donde los costos, el consumo energético y los requerimientos de administración son considerablemente elevados.

A los fines de cumplir con esa meta, este trabajo se basa en la premisa de optimizar y reutilizar la capacidad ociosa, y masivamente disponible de los dispositivos móviles (ARM) y equipos x86, hacia los cuales la plataforma distribuye las tareas de cómputo intensivo. Para ello, el sistema utiliza una arquitectura distribuida basada en microservicios que se estandarizan para correr dentro de contenedores, los cuales se ejecutan en clústeres orquestados por Kubernetes. Las políticas, procesos y técnicas definidas permiten gestionar los recursos de manera eficiente, ofrecer las capacidades de cómputo de manera portable y transparente tanto en ambientes de Nube, locales o híbridos, y ofrecer características de escalabilidad, flexibilidad y alta disponibilidad. De esta manera, se busca que tanto los costos asociados a la infraestructura de administración de la plataforma como los de procesamiento de las tareas se reduzcan considerablemente. Además, al utilizar hardware masivo, distribuido y de bajo consumo energético, que ya se encuentra disponible, la cantidad de energía, requisitos de refrigeración y espacio físico de servidores también se reduce drásticamente.

Una vez desarrollada la plataforma y definidos los escenarios de experimentación, utilizando el servicio de transcodificación de video como tarea de cálculo intensivo, se evalúa, por un lado, el rendimiento y consumo energético de los dispositivos trabajadores móviles; y por otro, las capacidades de escalabilidad, flexibilidad y disponibilidad de la plataforma. Con los resultados obtenidos, se pretende mostrar que la plataforma es lo suficientemente robusta para presentarla como una alternativa escalable, de bajo costo y consumo energético para resolver tareas de cómputo intensivo.

1.2 - Metodología de trabajo

Para poder cumplir con los objetivos del trabajo, se llevaron a cabo las siguientes actividades:

- Realizar una revisión bibliográfica de diferentes autores, trabajos y editoriales (trabajos de investigación, tesis de posgrado, artículos de empresas, reportes técnicos, publicaciones, entre otros). Para ello, debido a la constante evolución de los temas tratados, se releva material, proyectos y referencias bibliográficas

contemporáneas con el objetivo de que las estadísticas y los datos obtenidos sean compatibles con las necesidades actuales del mercado. Se consultaron diferentes fuentes reconocidas como: IEEE, Springer, Gartner, BBC, Cisco, The Economist, Forbes, Pew Research, Statista, The Guardian, Telegraph, entre otros.

- Realizar un análisis pormenorizado de las problemáticas e implicancias que conllevan el desarrollo de la plataforma definida. Esto implica conocer y estudiar el estado actual de las herramientas, técnicas y tecnologías de desarrollo, así como también las mejores prácticas y recomendaciones del área de estudio para el despliegue y administración de sistemas de carácter distribuido.

- Diseñar, desarrollar e implementar las funciones del sistema HPC. Para ello fue necesario construir los siguientes componentes:

    - Módulos para el procesamiento de tareas de cómputo intensivo. Esto implica la construcción de la aplicación móvil para los nodos Android (.apk) y la versión Dockerizada para los dispositivos x86.

    - Microservicios Dockerizados[1] para la administración y gestión de la plataforma. Esto implica la codificación de los servicios web, divisor y unificador de tarea, integración con motor de base de datos y sistema de colas, entre otros, a través de un paradigma de microservicios y despliegue de funciones basadas en procesos y cultura DevOps.

    - Implementación y despliegue de los servicios de colas y base de datos Dockerizadas, para registrar tareas de procesamiento e información estadística, respectivamente.

    - Utilizar el orquestador de contenedores (Kubernetes) para manejar, implementar y automatizar los despliegues de la plataforma de manera flexible y optimizada. Estas actividades permiten ejecutar el sistema tanto a nivel local como en la Nube.
- Definir y analizar las características más relevantes para dar sustento a la integración de la plataforma con el caso de aplicación particular de tarea HPC (compresión de video) de manera distribuida. Para cumplir con esos objetivos se realizaron las siguientes actividades:

[1] Dockerizado: No es una palabra válida en el diccionario español. Esta descripción hace referencia al proceso de estandarizar y convertir una aplicación a un formato de contenedores, específicamente Docker. En adelante, se utiliza esta escritura por simplicidad.

- Estudiar las características del video digital, los códecs, formatos y calidades, las plataformas de videostreaming, los mecanismos de transcodificación de manera distribuida, entre otros. Esto permite sentar las bases y el conocimiento necesario para incluir este servicio como tarea HPC.

- Analizar las soluciones de código abierto que están disponibles para realizar este tipo de tareas. Una vez seleccionada FFMpeg como herramienta básica de trabajo, se realizaron los ajustes necesarios para poder integrar sus funciones dentro de los componentes del sistema.

- Analizar y seleccionar recursos audiovisuales que sean representativos para ejecutar y evaluar la plataforma a través de la compresión de video distribuida.

- Estudiar y definir diversos códecs y perfiles de compresión de video, para generar material audiovisual con una calidad adecuada para las necesidades de compatibilidad y condiciones de ambiente de los diferentes dispositivos destino. A su vez, estas tareas generan trabajos que requieren un gran poder computacional para ser resueltas.

- Definición de métricas y escenarios de experimentación y evaluación de la plataforma. A través de las tareas de compresión definidas fue posible verificar que las características y funciones desarrolladas cumplieran con las necesidades requeridas para el procesamiento de la tarea HPC. Para ello fue necesario:

  - Analizar y estudiar trabajos relacionados y definir las métricas y estadísticas a registrar teniendo en cuenta los objetivos de este trabajo (performance, escalabilidad y consumo energético). Para el caso particular de la medición de energía, se incluyen dispositivos y mecanismos de captura acordes a las características de los dispositivos físicos (x86) y los nodos móviles (Android).

  - Analizar y seleccionar plataformas x86 y dispositivos ARM para ejecutar los escenarios de experimentación, teniendo en cuenta las necesidades de la tarea y las condiciones de despliegue de la plataforma.

  - Ejecución de los escenarios de prueba y análisis de resultados para dar soporte a la hipótesis de este trabajo. Cómo resultado, se evalúa el rendimiento, escalabilidad, flexibilidad de la plataforma y se mide la performance y la eficiencia energética de los dispositivos utilizados.

1.3 - Contribuciones

En esta tesis se analizó, diseñó, desarrolló y evaluó una plataforma para ejecutar tareas de cómputo intensivo, la cual tiene en cuenta características de escalabilidad, flexibilidad, alta disponibilidad, a la vez que intenta mantener un costo reducido de implementación y mantenimiento. Para ello se utilizan componentes de desarrollo y despliegue de aplicaciones basados en DevOps, microservicios, contenedores orquestados y recursos en la Nube; así como también se reutiliza el hardware de equipos móviles y tradicionales para la resolución de las tareas HPC. Cómo resultado del trabajo realizado, de las pruebas ejecutadas y de los resultados presentados, se derivan los siguientes aportes:

- Haber obtenido una descripción global del estado del arte para la implementación de algoritmos distribuidos teniendo en cuenta las tecnologías, técnicas y herramientas actuales. Toda la bibliografía relevada se encuentra organizada por unidad temática y se presenta al final de este trabajo.

- Haber probado que el uso eficiente y planificado de los recursos y tecnologías disponibles (Nube, móviles, microservicios y contenedores) permite construir una plataforma distribuida escalable, eficiente, y flexible, que optimiza la infraestructura requerida, dependiendo de la carga de trabajo y disponibilidad de nodos trabajadores, permitiendo optimizar la gestión de costos. Para ello, fue necesario realizar una definición de métricas para la evaluación de la plataforma, construcción y ejecución de escenarios de prueba, y su posterior análisis de resultados, a través de los cuales fuera posible evaluar:

    - Rendimiento, compatibilidad, comportamiento y consumo energético de los nodos trabajadores.

    - Comportamiento, flexibilidad y escalabilidad de la plataforma.

- Haber comprobado que los dispositivos móviles deben considerarse como un equipamiento con ventajas competitiva en lo que respecta a capacidad, rendimiento y consumo energético:

    - Sobre la base del relevamiento, análisis y estudio del estado actual de los dispositivos, conocer que estos equipos presentan una masividad y penetración en el mercado en constante crecimiento, al mismo tiempo que su capacidad de cálculo, estabilidad de software y funciones agregadas se encuentran transitando una mejora continua y sostenida. También conocer

que presentan largos períodos de ociosidad mientras estos dispositivos se encuentran recargando sus baterías.

- A partir de la experimentación desarrollada, haber validado que la reutilización de estos recursos (masivos y ociosos) es beneficioso para procesar tareas de manera distribuida y paralela, ya que son dispositivos competitivos en cuanto a rendimiento y consumo energético respecto de las arquitecturas tradicionales x86, a la hora de resolver tareas HPC. Para verificar estas condiciones, en el caso de equipos x86 se construyó un dispositivo (hardware) de medición y en el caso de los equipos móviles se desarrolló un módulo de software que accede a la información de consumo energético del sistema operativo Android. Ambas herramientas también se brindan a la comunidad interesada.

- Haber desarrollado una aplicación móvil simple en lenguaje nativo para Android la cual es compatible con cualquier dispositivo que cumpla con las condiciones definidas. Además, el sistema funciona independientemente de la arquitectura del procesador (32 o 64 bits). Por último, el trabajo se encuentra publicado en un repositorio abierto donde se indica y explica cómo puede extenderse su funcionalidad incorporando librerías, funciones y binarios que den soporte a las actividades HPC.

- Ofrecer a la comunidad educativa y profesional un sistema de cómputo distribuido y paralelo para resolver tareas que requieran capacidad de cálculo escalable. Es importante destacar que este aumento en capacidad se da a través de la integración de equipamiento de hardware heterogéneo, ocioso, masivamente distribuido, disponible y en constante crecimiento. Además, es relevante haber vinculado estas características con el servicio de compresión de video, mostrando cómo puede integrarse una nueva tarea HPC y ser correctamente ejecutada.

1.4 - Infraestructura de experimentación

Para realizar la experimentación y evaluación de la plataforma se plantearon un conjunto de condiciones que se describen a continuación:

Consideraciones sobre la evaluación de los nodos trabajadores:

La ejecución de tareas de experimentación y evaluación sobre nodos trabajadores Android se realizó con éxito sobre un conjunto amplio y heterogéneo de equipos. El objetivo, por un lado, era ejecutar las pruebas de compatibilidad, estabilidad y rendimiento sobre cada uno de los dispositivos, y por el otro comprobar que, con los requisitos y condiciones mínimas establecidas en la aplicación, el nodo trabajador móvil puede instalarse y realizar sus operaciones sin ningún problema.

Por otra parte, a la hora de realizar pruebas de carga concurrente, estrés y escalabilidad, se utilizaron arquitecturas virtualizadas, limitando sus recursos en base a las definiciones de hardware disponible en los dispositivos móviles. Esta restricción, aplicada en los escenarios de prueba, se debe a que no era posible contar con grandes cantidades de cada uno de los modelos de los celulares para la evaluación. Teniendo en cuenta la capacidad del laboratorio de pruebas (arquitectura empresarial de 9 Servidores tipo Blade HP ProLiant BL460c G8) y las definiciones de hardware establecidas en el trabajo, fue posible definir una cantidad de 32 nodos trabajadores para cada una de las arquitecturas de evaluación. De esta manera, se logró realizar extrapolaciones y análisis de comportamiento sobre diversas condiciones y escenarios.

Consideraciones sobre los despliegues de la plataforma en los ambientes de Kubernetes:

Debido a que los servicios se desarrollaron para ser ejecutados y desplegados en clústeres del orquestador Kubernetes, la plataforma puede ser montada sobre cualquier infraestructura local o Nube que sea compatible con este orquestador. Sin embargo, en la experimentación realizada, se limitó a desplegar la plataforma y ejecutar los escenarios de prueba a los siguientes casos:

- Proveedores de Nube: Se desplegó, ejecutó y validó en los dos proveedores de Nube pública más utilizados en este momento (Amazon Web Services y Microsoft Azure), con sus servicios de almacenamiento (Amazon S3 y Azure Blob Storage).
- Ambientes locales: Se desplegó, ejecutó y validó en una arquitectura de Servidores tipo Blade (9 unidades Blade HP ProLiant BL460c G8), corriendo sobre un esquema de virtualización VMware + VCenter 6.5.

Más allá de las condiciones mencionadas, los casos analizados son significativos y los resultados son representativos para la validación de la plataforma desarrollada.

1.5 - Publicaciones

A lo largo del desarrollo de esta tesis se han publicado los avances y resultados obtenidos en diferentes congresos científicos. Cómo resultado, se presentaron cuatro artículos que cubren el análisis del marco actual, el desarrollo y la evaluación de la plataforma diseñada.

Sobre la base de haber analizado la bibliografía y el estado del arte de los dispositivos móviles, las arquitecturas distribuidas de procesamiento paralelo y distribuido, y estudiar diversas tareas HPC, se sentaron las bases para el diseño de la arquitectura propuesta. Además, se estudió la factibilidad de implementar la tarea de compresión de video como actividad HPC y se realizaron las primeras pruebas sobre los dispositivos móviles. Este trabajo dio como resultado la presentación del primer artículo:

- [PET17] Petrocelli, D., De Giusti, A. E. & Naiouf, M. “Procesamiento distribuido y paralelo de bajo costo basado en Cloud & móvil”. XXIII Congreso Argentino de Ciencias de la Computación, XVIII Workshop de Procesamiento Distribuido y Paralelo (WPDP), 216–225 (2017).

Una vez validada la propuesta y sobre las bases definidas, se desarrolló y evaluó la primera versión de la arquitectura del prototipo de procesamiento HPC. Se integró la compresión de video bajo demanda como tarea de cómputo intensivo sobre la arquitectura distribuida.

En este trabajo, se estudió, evaluó y documentó la capacidad de procesamiento y la ventaja competitiva, en cuanto a consumo energético, de los dispositivos móviles respecto de las arquitecturas tradicionales. Para obtener estas mediciones fue necesario incluir y desarrollar herramientas de medición (de hardware y software). Además, se evaluó el funcionamiento de la plataforma distribuida y la integración de los dispositivos trabajadores (móviles y tradicionales) para la resolución de las tareas.

Este apartado sentó las bases para el segundo trabajo publicado:

- [PET19] Petrocelli, D., De Giusti, A. E. & Naiouf, M. “Hybrid Elastic ARM&Cloud HPC Collaborative Platform for generic tasks”. VII Conference Cloud Computing and Big Data (La Plata, 2019), Instituto de Investigación en Informática, ISBN: 978-3-030-27713-0, Páginas: 16-27 (2019).

Con objeto de otorgar flexibilidad, escalabilidad, alta disponibilidad y optimización de la infraestructura, la plataforma adoptó herramientas y tecnologías modernas, se mejoró el desarrollo del sistema y se generó una segunda versión de la plataforma. En particular, se integraron las prácticas DevOps, microservicios y contenedores orquestados. Gracias a su inclusión, las funciones se fragmentaron en servicios más pequeños y con responsabilidades acotadas, las cuales se empaquetaron con todas sus dependencias y librerías en contenedores (Docker), resultando en servicios más pequeños, inmutables, portables, seguros y estandarizados, para ejecutarse independientemente de la arquitectura subyacente; los cuales se administran, despliegan y escalan para brindar un uso eficiente de recursos, costos y energía.

Con la plataforma definitiva desarrollada, se incluyeron un conjunto de pruebas y escenarios que permitieron validar la compatibilidad, rendimiento, escalabilidad, flexibilidad y tolerancia a fallos del sistema.

Cómo resultado de este proceso, finalmente, se desarrollaron tres publicaciones:

- [PET20a] Petrocelli, D., De Giusti, A. E. & Naiouf, M. “Collaborative, distributed and scalable platform based on mobile, cloud, micro services and containers for intensive computing tasks” Short Papers of the 8th Conference on Cloud Computing Conference, Big Data & Emerging Topics (JCC-BD&ET 2020).
- [PET20b] Petrocelli, D., De Giusti, A. E. & Naiouf, M. “Plataforma colaborativa, elástica, de bajo costo y consumo basada en recursos de la Nube, contenedores y móviles para HPC” 7ª Conferencia Ibero Americana Computação Aplicada - 2020 Lisboa, Portugal
- [PET21] Petrocelli, D., De Giusti, A. E. & Naiouf, M. “Collaborative, distributed, scalable and low-cost plat-form based on microservices, containers, mobile devices and Cloud services to solve compute-intensive tasks” Euro-Par 2021 PhD Symposium (Parallel and Distributed Processing) . Lisboa, Portugal.

1.6 - Organización de la tesis

En el Capítulo 2, se define el marco teórico de la tesis. Se presenta una definición de los Sistemas distribuidos y HPC, así como también su evolución y conceptos más importantes. Se investigan, también, las arquitecturas, modelos y tecnologías actuales más relevantes, relacionados con el desarrollo de software para este paradigma. Por último, se estudian las características, oportunidades y evolución de los dispositivos móviles desde el punto de vista

de su capacidad como equipamiento portátil, económico, masivo y potente para ser utilizado en ambientes de procesamiento HPC.

En el Capítulo 3, se describen los detalles del modelo de arquitectura del sistema y sus componentes. Se detalla de qué manera se cumplen los objetivos planteados. Para ello, se presentan las entidades que se interrelacionan, así como también cada una de sus responsabilidades. Se explican las funciones desarrolladas y los aspectos tecnológicos, patrones y arquitecturas utilizadas para la construcción, despliegue e integración de la plataforma. Esto permite a los interesados tomar la arquitectura desarrollada o las ideas planteadas para ser adaptadas a sus propios proyectos.

En el Capítulo 4, se define el caso de estudio HPC: Compresión distribuida de video bajo demanda. Se presentan las características básicas de la composición de un video, los tamaños y formatos de archivo, los códecs de video y la complejidad que requiere realizar procesos de transcodificación sobre este material. Luego, se explica por qué se integra este servicio como tarea HPC, para la plataforma antes definida. Se describe, también, cómo la plataforma realiza el proceso de transcodificación de vídeo utilizando la herramienta de código libre FFMpeg.

En el Capítulo 5, se presentan las métricas de evaluación definidas, los modelos de evaluación, pruebas, escenarios y experimentación. Se define un coeficiente de rendimiento, el cual correlaciona tiempos de respuesta y consumo energético; así como también métricas de escalabilidad y comportamiento. Sobre estas bases, se diseñaron los escenarios de pruebas para validar, tanto el comportamiento, performance de la plataforma y los nodos trabajadores; como las cargas de trabajo concurrente que permitieron validar la escalabilidad y flexibilidad del sistema. Más tarde, luego de haber ejecutado los escenarios de experimentación, se describen y analizan los resultados obtenidos. Estos, sustentan a la arquitectura desarrollada y muestran las ventajas obtenidas.

En el Capítulo 6, finalmente, se presentan las conclusiones abordadas y trabajos futuros.

Al final del documento se incluye un glosario y las referencias bibliográficas relacionadas con este trabajo. Para contar con una organización más sencilla y fácil de seguir, la bibliografía se estructura por temática. De esta manera, la persona interesada puede acceder de manera más simple a los contenidos citados**.** A su vez, cómo capítulos anexos al final del trabajo, se presentan los protocolos, servicios y herramientas adicionales utilizadas.

2. Marco Teórico

El propósito de este capítulo es proporcionar una descripción de los principales conceptos y tecnologías relevantes para el desarrollo de este trabajo de investigación. Las primeras dos secciones proporcionan conceptos básicos y sientan las bases de lo que implica un sistema HPC y su relación directa con los Sistemas Distribuidos. El resto de los apartados explican los conceptos, terminologías, estructuras y tecnologías relevantes que luego se interrelacionan y se aplican para el diseño, construcción, implementación y despliegue de la plataforma HPC planteada en este trabajo.

2.1 - Computación de alto rendimiento (HPC)

La historia de los sistemas HPC se remonta a la década de 1960, cuando se utilizó la computación distribuida y paralela para lograr un alto rendimiento computacional [STE17][BOL12].

La computación en paralelo generalmente utiliza memoria compartida para intercambiar información entre procesadores mientras que la computación distribuida usa memoria distribuida, con información compartida entre procesadores mediante el paso de mensajes. Por las características del entorno HPC, se ha vuelto difícil distinguir entre sistemas paralelos y distribuidos, ya que los sistemas de cómputo paralelo a menudo integran características y funciones distribuidas.

En términos generales, el término computación de alto rendimiento (HPC) o supercomputación se refiere a cualquier implementación computacional que implique la agregación de potencia para obtener un rendimiento superior al que ofrecen los equipos individuales [YOU17]. HPC es una tecnología ampliamente utilizada para la investigación y la industria cuando se necesita una gran potencia de cálculo. Por lo general, se utilizan para simulaciones de física y química, pronóstico del tiempo, accidentes en aeronaves, dinámica de fluidos, diseño de materiales, farmacéutica, análisis y modelado de proteínas y bioinformática, entre otras [KUR19] [FIS19].

Las supercomputadoras modernas son una integración de diferentes diseños, arquitecturas y plataformas de hardware. Sin embargo, algunos de ellos están más extendidos y son más dominantes que otros. En particular, estos suelen ser sistemas de clústeres de productos básicos modularizados, heterogéneos e inherentemente paralelos con procesadores de múltiples núcleos y aceleradores opcionales. Los sistemas HPC pueden estar conformados por entornos Grid [XUA09][DER07], Cluster [STO18][XIN97], Cloud [MAL19][RUI14] o

entornos híbridos [PRA18][EME16], y correr sobre hardware masivamente paralelo y distribuido, donde cada nodo, generalmente contiene múltiples CPU con varios núcleos, aceleradores gráficos (GPU) y memoria RAM. Esos nodos suelen estar interconectados por interconexiones Gigabit o 10-Gigabit Ethernet [PAN07] [WEN00] e InfiniBand [RUH20][NAR08].

Optimizar el rendimiento de las aplicaciones científicas en entornos HPC es una tarea compleja que ha motivado el desarrollo de muchas herramientas de análisis de rendimiento durante las últimas décadas. Esta infraestructura de múltiples núcleos, multiprocesadores y múltiples nodos se explota de la mejor manera usando generalmente los frameworks de computación distribuida y paralela Open Multiprocessing (OpenMP) [BAR07], CUDA [FAT08] y Message Passing Interface (MPI) [GRO14] de manera híbrida. La combinación de MPI y OpenMP/CUDA brinda la posibilidad de instanciar múltiples tareas paralelas en diferentes nodos (gracias a MPI), y en cada uno múltiples subprocesos (gracias a OpenMP y CUDA). Sin embargo, para seleccionar los recursos correctos y ejecutar los trabajos, se necesita un software de programación y despacho.

Las aplicaciones de computación intensiva muchas veces se denominan aplicaciones de ejecución prolongada y, con mayor frecuencia, son cálculos científicos que utilizan modelos matemáticos y técnicas de análisis cuantitativo que se ejecutan en sistemas HPC para analizar y resolver problemas científicos grandes y complejos. Algunos de ellos habrían sido demasiado difíciles de realizar sin los sistemas HPC, debido al capital necesario, la complejidad o el riesgo financiero involucrado (por ejemplo, en experimentos con armas nucleares).

2.2 - Sistemas Distribuidos

La informática ha pasado por muchas transformaciones desde el nacimiento de las primeras computadoras. Los avances tecnológicos han dado como resultado la disponibilidad de procesadores rápidos y económicos, y la evolución en la tecnología de comunicaciones ha devenido en la disponibilidad de redes lucrativas y altamente competentes [DAS20][ERC19]. La integración de las tecnologías informáticas y de redes dio origen, a finales de la década de 1970, al paradigma de la computación distribuida [BRE18].

Cuando un puñado de computadoras potentes se conectan y se comunican entre sí, la capacidad de cálculo global disponible resulta mayor. Como resultado, el sistema puede tener mayor rendimiento que una sola supercomputadora [CZA18]. Por lo tanto, la computación distribuida se puede aproximar a través del siguiente concepto: un enfoque de descentralización para la computación permite acceder a grandes cantidades de potencia

computacional donde el objetivo es minimizar el costo de comunicación y de unidades de hardware individuales [DAS20].

En los sistemas distribuidos, los procesos de trabajo de la aplicación se dividen entre los nodos participantes [GRU10]. El aspecto básico en todas las arquitecturas informáticas distribuidas es la noción de comunicación entre computadoras.

Por lo tanto, un sistema distribuido es una aplicación que se ejecuta bajo la utilización de una colección de protocolos para coordinar las acciones de múltiples procesos en una red de comunicación, de modo que todos los componentes cooperan para realizar un conjunto único o pequeño de tareas relacionadas. Los equipos pueden acceder a recursos remotos, así como a recursos locales en el sistema distribuido a través de la red de comunicación. La existencia de múltiples recursos autónomos es transparente para el usuario en un sistema distribuido [ERC19]. El usuario no sabe que los trabajos son ejecutados por múltiples computadoras que subsisten en ubicaciones remotas [KUM14][PUT93][GUY90]. Esto significa que, a la inversa de los sistemas centralizados, ninguna computadora en el sistema lleva toda la carga de recursos del sistema que usualmente requería ejecutar un programa de computadora.

2.2.1 - Arquitecturas, herramientas y tecnologías para el desarrollo de un Sistema Distribuido para resolver tareas HPC

El desarrollo de sistemas HPC modernos requiere la adopción de técnicas, tecnologías, herramientas e infraestructura de vanguardia. Aunque hay muchas opciones para elegir en cada categoría, las características más relevantes son:

- El desarrollo y despliegue de aplicaciones rápidas, ligeras y escalables, buscando siempre garantizar la confidencialidad de los datos y la garantía del servicio.
- Optimizar la infraestructura necesaria para cada trabajo.
- Reducir los costos asociados.

En la actualidad, el tiempo de desarrollo, despliegue, aprovisionamiento y la respuesta a los nuevos requisitos se han vuelto críticos. La escalabilidad ha alcanzado límites que nunca antes fueron posibles y requeridos, con cientos de millones de usuarios concurrentes que acceden al mismo servicio al mismo tiempo. Esto implica que para lograr que la implementación de los servicios sea exitosa y disponga de atributos de escalabilidad, alta disponibilidad, auditabilidad, monitorización y trazabilidad que explote todos los beneficios de

la computación actual (On-Premise + Nube) se deben aplicar, respetar y promover las prácticas DevOps y el desarrollo basado en microservicios.

Además, en los últimos años, la automatización de las pruebas, despliegues y operaciones se han vuelto centrales en el aprovisionamiento y control de las nuevas arquitecturas IT y el desarrollo de software. Los administradores crean procesos automatizados (scripts) y otras herramientas de automatización para completar tareas de forma rápida, segura y sin errores. Esto permite a los equipos IT responder de forma más eficaz a las necesidades de las compañías e instituciones, a la vez que concede una mayor eficiencia y escalabilidad del sistema. Especialmente si se explotan las características disponibles por la computación en la Nube, y también las tecnologías de virtualización basada en contenedores orquestados

A continuación, se presentan seis apartados que describen los modelos, arquitecturas y aspectos tecnológicos que fueron integrados de manera conjunta para construir la plataforma definida en la hipótesis de este trabajo.

2.3 - Microservicios

El término "micro servicios web" fue utilizado por primera vez por el Dr. Peter Rogers durante una conferencia sobre computación en la Nube en 2005. Netflix y Amazon estuvieron entre los primeros pioneros que implementaron microservicios a gran escala [MAU19].

Una primera aproximación a la definición de microservicios podría ser: un estilo arquitectónico que estructura una aplicación como una colección de servicios acoplados libremente, que implementan capacidades comerciales [RIC18][RIC18b][DRA17]. Los microservicios representan un claro ejemplo del refrán “divide y vencerás” [MER20][TAB19], ya que se basan en la descomposición de un sistema, en unidades de servicio desplegables e independientes; de este modo, cada servicio tiene un contexto limitado basado en un dominio acotado específico.

Este modelo se ha introducido y evolucionado debido a los logros de la ingeniería de software en los últimos años, con respecto a las infraestructuras de computación distribuida en la nube, las mejoras de la interfaz de programación de aplicaciones (API), las metodologías de desarrollo ágil y el surgimiento de los recientes fenómenos de aplicaciones en contenedores [FET16]. Un contenedor es una unidad estándar de software, que empaqueta el código y todas sus dependencias, para que la aplicación se ejecute de manera rápida y confiable, desde un entorno informático a otro, comunicándose con otros a través de una API [PAH17]. Un resumen de la evolución de estas arquitecturas de virtualización se presenta en la figura 2.1.

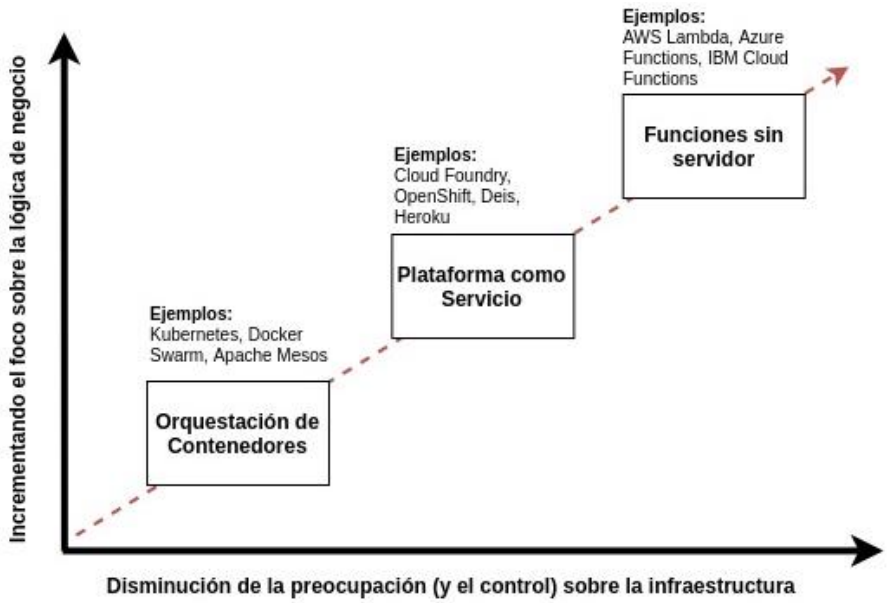

Figura 2.1 - Modelos y evolución para la implementación de microservicios. Fuente: Propia. Versión adaptada de [WIN19].

En este modelo, los servicios se desarrollan para ser autónomos, confiables, pequeños, resistentes y pensados para realizar una única función. Esto provee una cultura de iteración rápida, automatización, pruebas e implementación continua; permitiendo a los equipos crear productos e implementar código, exponencialmente más rápido que nunca [NEW14].

El desacoplamiento y las interfaces bien definidas de los microservicios brindan a las aplicaciones la capacidad de escalar sin problemas, y permite a los desarrolladores realizar actualizaciones mediante la implementación de nuevas versiones de servicio, sin detenerlo [FET16]. También permite que se desarrollen microservicios utilizando diferentes tecnologías o cambiando las mismas a lo largo de la vida útil del software. Los equipos de desarrollo pueden usar o agregar nuevas tecnologías para cumplir mejor los requisitos de la aplicación, mejorando su seguridad y confiabilidad a través de actualizaciones frecuentes (por ejemplo, aplicando correcciones de errores en el producto elegido), lo que resulta en implementaciones frecuentes de nuevos y mejorados componentes. Entonces, los microservicios son una parte crítica de una serie de avances significativos que están cambiando la naturaleza de cómo se trabaja y desarrolla en el área de la tecnología de la información.

Las técnicas ágiles de desarrollo de software, el traslado de aplicaciones a la nube, la cultura DevOps, la integración y la implementación continuas (CI/CD) [CHE18][CHE16]; y el uso de contenedores se está utilizando junto con los microservicios para revolucionar el desarrollo y la entrega de aplicaciones.

2.4 - Cultura y Técnicas DevOps

El término DevOps fue introducido en 2009 por Patrick Debois, Gene Kim y John Willis [KIM16], y representa la combinación de Desarrollo (Dev) y Operaciones (Ops). Esto permite ofrecer valor comercial agregado a los usuarios más rápidamente y, por lo tanto, ser más competitivos en el mercado. Este proceso de liberar software a un ritmo más ágil viene acompañado con la necesidad de alinear mejor las preocupaciones de las partes interesadas que residen en la cadena de desarrollo de software [MAC20]. Puntualmente, las áreas de desarrollo y operaciones deberían estar alineadas. Sin embargo, estos sectores han tradicionalmente organizado sus procesos de manera diferente, y muchas veces trabajan aislados unas de las otras [AGR19]. Por esta razón, con el fin de crear un flujo de extremo a extremo rápido y sin problemas, DevOps proporciona una manera de lidiar con lo antes mencionado y toma en consideración no sólo el desarrollo y las operaciones, sino también otras partes interesadas, como el aseguramiento de la calidad (QA), la gestión de productos y seguridad de la información. A continuación, se presenta una gráfica (figura 2.2) que muestra la evolución de este concepto a lo largo de los últimos 5 años.

Figura 2.2 - Aumento de la tendencia del interés del término de DevOps en los últimos 5 años. Fuente: Google Trends, consultado el 15 de enero de 2021. Recurso: https://trends.google.com.ar/trends/explore?date=today%205-y&q=devops

Por lo tanto, DevOps no es un proceso, herramienta o tecnología fija, sino una mentalidad en la que la confianza y el respeto entre las partes interesadas constituyen un hecho esencial; y la colaboración entre dichas partes interesadas se lleva a cabo en perfecto orden, para lograr un lanzamiento de productos de manera cooperativa [DAV16]. Para ese objetivo, DevOps

aplica prácticas relacionadas con la cultura (por ejemplo, cultivar la confianza y el respeto [DAV16]); la colaboración (como ser, adoptar equipos multifuncionales [KIM16]); el pensamiento intercultural (entre ellos, utilizar la técnica de mapeo de flujo de valor para optimizar el movimiento DevOps [KIM15]); y la automatización (tales como automatizar compilaciones, pruebas e implementaciones para lanzar software más rápidamente [HUT12]). Además, DevOps pretende medir el efecto del cambio técnico y social [DAV16]. Esto implica que también abarca prácticas que se vinculan con el monitoreo y la medición, lo que genera una mejora continua [FIT17] [PLW15][HUT12].

2.4.1 - Automatización en DevOps

La automatización es una herramienta utilizada para ahorrar tiempo, recursos y disminuir significativamente los errores cuando un proceso, dentro de la cadena de producción del software, se completa en varias etapas. Por ejemplo, las grandes fábricas implementaron la automatización cuando un producto debía fabricarse exactamente igual y repetitivamente. Para esos casos, utilizaron un robot industrial automatizado [GRO15][BRO07][LOO11]. En el caso de los sistemas, existen diferentes herramientas o soluciones que se ocupan de implementar procesos automáticos [ZAM12][HEN12][SUD12]. Cómo se describió, estos procesos de automatización pueden integrarse en cualquier tipo de organización y sobre cualquier tipo de arquitectura. Por ejemplo, redes, infraestructura, recursos en la Nube, gestión de la configuración, despliegue de aplicaciones, computación de borde, aspectos de seguridad y monitoreo continuo.

DevOps permite obtener una mayor velocidad en los procesos, escalabilidad, consistencia y retroalimentación. Como se describe en la figura 2.3, existen 4 características básicas provistas gracias al uso de esta sistematización (velocidad, consistencia, escalabilidad y retroalimentación). Cada una de estas cualidades depende de las otras tres. Por ejemplo, no puede pensarse en escalar un sistema a menos que puedan agregarse rápidamente recursos con configuraciones consistentes. No es posible recibir retroalimentación (feedback) automáticamente sin disponer de técnicas de prueba y monitoreo en tiempo real. Tampoco es posible responder a los comentarios de manera efectiva, a menos que tenga una forma rápida para realizar cambios incrementales de manera segura.

Figura 2.3. Características provistas gracias al uso de procesos de automatización. Fuente: Propia. Versión adaptada de [CHE15].

2.4.2 - Infraestructura como código

La infraestructura como código (IaC) [KLE19] [RAH18] [ART17] se ha impulsado durante la última década y se ha vuelto muy popular, para automatizar la administración de los centros de datos. Por lo tanto, IaC es un mecanismo para sistematizar casi cualquier recurso en un centro de datos, al producir una “receta” de los aspectos y pasos que se desean ejecutar. Para ello, se define un conjunto de prácticas que utilizan "código” en lugar de comandos manuales para configurar recursos, despliegues y redes virtuales, instalar paquetes y configurar el entorno para la aplicación de interés [DES19]. Esto puede incluir: equipos físicos ("bare metal"), máquinas virtuales, contenedores y redes definidas por software. En la figura 2.4 se presentan las herramientas más utilizadas para realizar este tipo de operaciones.

Figura 2.4 - Herramientas más utilizadas a la hora de realizar tareas de IaC en la actualidad. Fuente: Propia.

El código debe desarrollarse y administrarse utilizando los mismos procesos que se aplican sobre cualquier otro desarrollo de software; por ejemplo, debe diseñarse, documentarse, probarse y almacenarse en un repositorio de control de versiones [RAH18b] [AWS17].

Los operadores de sistemas de tecnología de la información (Ops) han empleado durante mucho tiempo este tipo de tareas a través de secuencias de comandos ad hoc de tareas,

tecnología y prácticas de IaC. Sin embargo, con la introducción de la computación en la Nube, y particularmente la tecnología de infraestructura como servicio (IaaS) [PAT20] [AWS17][BAS15] esto se ha vuelto una práctica indispensable para mantener las infraestructuras adecuadas y adaptadas a las necesidades de un mercado en constante cambio y expansión.

Como se señaló, el código utilizado para las tareas de IaC debe almacenarse en un repositorio con control de versiones, lo que permite una arquitectura robusta sobre la infraestructura desplegada. Así, cualquier versión de la infraestructura se puede crear utilizando el código IaC correspondiente a la versión de interés [PEA19][DRI19]. Conjuntamente, la automatización y el control de versiones proporcionan la capacidad de crear y recrear de manera eficiente y confiable una configuración particular. Esto se puede utilizar para revertir un cambio realizado durante el desarrollo, la integración o incluso en el entorno productivo, así como también para respaldar la recreación y depuración de problemas.

2.4.3 - Integración y despliegue continuo

En la actualidad los sistemas de software se están volviendo cada vez más complejos y distribuidos, a menudo involucrando equipos de desarrollo multidisciplinarios que trabajan simultáneamente en diferentes componentes de un proyecto modular común [DEJ17]. Por lo que es de suma importancia garantizar que las piezas individuales del sistema general puedan integrarse sin alterar la estabilidad y el funcionamiento del proyecto de software general. A partir de la introducción del concepto y prácticas de CI, en 1991 [FOW06], se motiva a los desarrolladores a realizar confirmaciones más frecuentes, regulares y detalladas que se vuelcan en un repositorio compartido, utilizando un sistema de control de versiones (VCS) como Git, Mercurial, Subversion (SVN), Sistema de versiones concurrentes (CVS) o Team Foundation Server (TFS). Disponer de estos repositorios es un elemento fundamental para lograr la entrega continua. El VCS actuará como una única fuente de verdad, y cada desarrollador debe contribuir al mismo repositorio. Con este modelo, es posible hacer un seguimiento de qué versión se implementa y dónde, facilitando así la colaboración entre los desarrolladores.

La planificación, integración, implementación, pruebas, entrega y retroalimentación continua son prácticas de desarrollo y operaciones ampliamente ejercidas y promovidas en las empresas modernas de desarrollo de software, basadas en procesos de desarrollo ágil [CHE18]. Como sugieren sus nombres, las operaciones continuas e incrementales de todos los equipos y activos involucrados, se encuentran en el núcleo de las prácticas CI y tienen como objetivo disminuir el tiempo entre la aplicación de un cambio en un sistema, y que este,

esté disponible en el entorno de producción. Además, el mantenimiento de la calidad del software, en términos tanto del código como de los mecanismos de entrega, son elementos clave en el proceso de desarrollo de software [BAS15]. En la figura 2.5, se describe la cantidad y frecuencia de despliegues automatizados que utilizan diversas compañías para cumplir con sus necesidades y objetivos.

Compañía	Frecuencia de Despligues	Tiempos de Entrega	Fiabilidad	Capacidad de respuesta del cliente
Amazon	23000 / día	minutos	alta	alta
Google	5500 / día	minutos	alta	alta
Netflix	500 / día	minutos	alta	alta
Facebook	1 / día	minutos	alta	alta
Twitter	3 / semana	minutos	alta	alta
Empresa Típica	una cada 9 meses	meses o cuatrimestres	baja / media	baja / media

Figura 2.5 - Frecuencia en despliegues de ajustes y nuevas funcionalidades a través de procesos DevOps (CI y CD) en las grandes compañías. Fuente: Propia. Versión adaptada de [ITR19].

Los procesos de CI generalmente se acompañan de tareas de entrega continua (CD). Este concepto se refiere a la liberación constante de cada desarrollo correcto (porción de código que pasa las pruebas establecidas) a producción [CAU13]. Ambos son términos generales, y constan de muchas tecnologías diferentes para formar un enfoque unificado para la entrega de software. El objetivo de estas tecnologías es lograr que se trabaje en conjunto para que la implementación de nuevos artefactos sea rápida y transparente al tener la base de código en un estado de producción constante [HUM10].

Al adoptar CI y CD, los desarrolladores pueden reducir el tiempo dedicado a corregir errores y aumentar el esfuerzo destinado a desarrollar nuevas funciones o mejorar las existentes. La implementación de esas acciones crea un mayor impacto en el valor comercial [ARA18], donde se potencia la automatización en cada paso del ciclo de desarrollo, incluida la creación de la infraestructura. La automatización ha evolucionado debido a la disponibilidad y madurez de las tecnologías en la nube y las herramientas de CI y CD.

2.4.4 - Monitoreo Continuo

El monitoreo continuo (CM) es un proceso constante, en el que el software se cambia y ajusta permanentemente. Su raíz es la automatización. La visibilidad es el paso final en la línea de desarrollo de DevOps, y es un elemento crucial para obtener verdadera eficiencia y escalabilidad [KAR19].

El monitoreo debe automatizarse de la misma manera que la integración, las pruebas y la implementación. En entornos altamente dinámicos y escalables, el proceso de observación

de microservicios debe adaptarse a los cambios sin intervención manual y configuraciones complejas [NIE17][FAR14].

Las capacidades críticas son las siguientes:

- Descubrimiento y visualización automática de todos los componentes y sus dependencias.
- Instrumentación y seguimiento de los componentes de la aplicación.
- Alerta inmediata de problemas de infraestructura y rendimiento de aplicaciones.

2.5 - Computación en la Nube

La computación en la Nube (Cloud Computing) es un paradigma donde un gran conjunto de sistemas están conectados en redes públicas o privadas, para proporcionar una infraestructura escalable dinámicamente, para el almacenamiento de aplicaciones, datos y archivos [PRA18][PAT12]. Con el advenimiento de esta tecnología, el costo de cómputo, alojamiento de aplicaciones, almacenamiento de contenido y entrega se redujo significativamente.

El Instituto Nacional de Estándares y Tecnología (NIST), describe la computación en la Nube como un modelo que permite un acceso a la red ubicuo, conveniente y bajo demanda; con un grupo de recursos de computación compartidos, configurables (redes, servidores, almacenamiento, aplicaciones y servicios, entre otros) que se pueden aprovisionar y liberar rápidamente con mínimo esfuerzo o interacción del proveedor de servicios [NIS11].

Según el NIST, la computación en la nube abarca las siguientes 5 características esenciales: a) Pool de recursos disponibles; b) acceso a los servicios desde Internet; c) Rápida elasticidad; d) Servicio medido; e) Autoservicio a demanda. Esto se muestra en la figura 2.6.

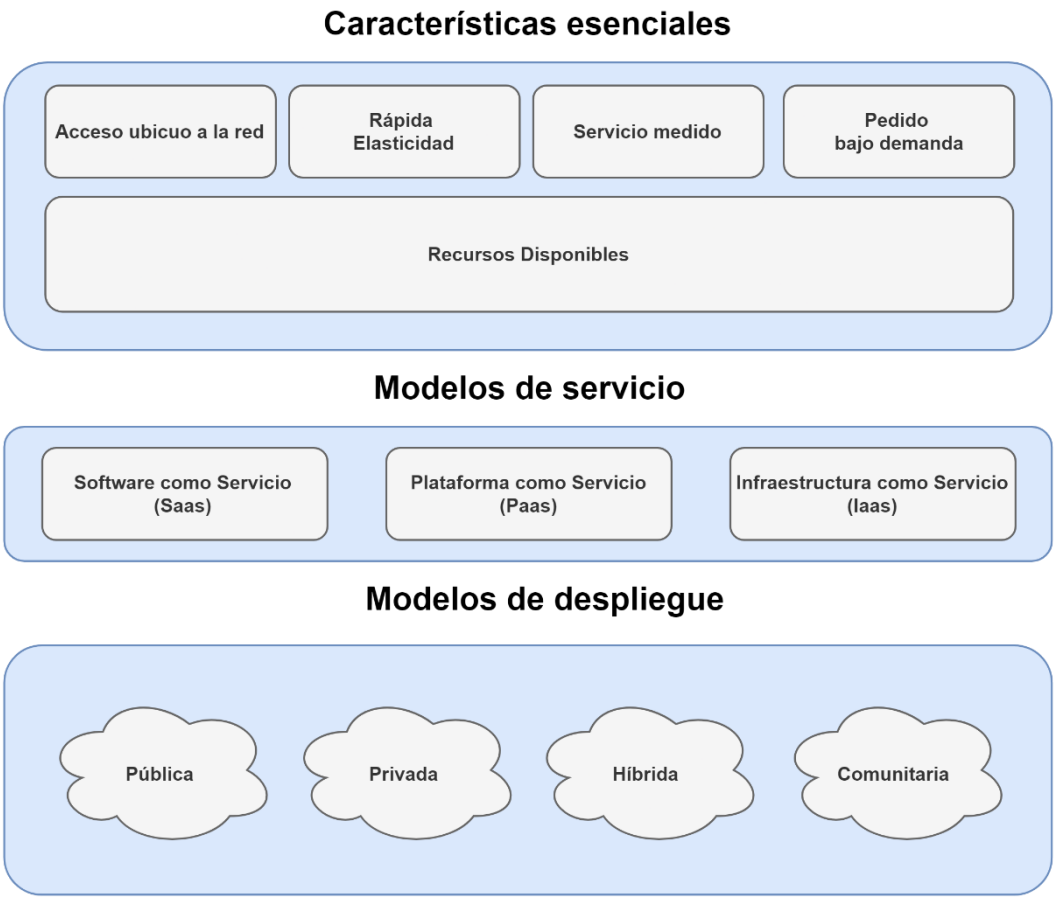

Figura 2.6 - Características esenciales de la computación en la Nube. Fuente: Propia. Versión adaptada de: https://www.researchgate.net/profile/Young-Chan_Lee/publication/259972968/figure/fig1/AS:297209158488073@1447871593349/Visual-model-of-NIST-working-definition-of-cloud-computing.png

2.5.1 - Modelos de implementación

La computación en la Nube se puede utilizar a través de diferentes modelos de implementación, en función del público objetivo [FAT15]. Estos modelos son (i) Nube pública, (ii) nube privada, (iii) nube híbrida.

Nube pública

Los servicios de la nube pública están abiertos y disponibles a los usuarios en general, a través del modelo de pago por uso. Un ejemplo son los proveedores que permiten a las empresas implementar sus aplicaciones de Internet y respaldar la escalabilidad y el monitoreo rápido [MEL11].

La Nube pública está actualmente presentando un gran margen de ahorro en costos, siendo uno de los motivos más destacados de este tipo de nube. Sin embargo, es necesario tener

ciertas precauciones y comprender cuándo es conveniente utilizar esta Nube y aprovechar sus características diferenciales y también cuando no debe aplicarse.

Nube privada

La Nube privada es uno o varios centros de datos, provisionados y utilizados por una sola corporación [HUM16] y los usuarios de este modelo pueden ser internos o externos de la organización. De hecho, este modelo es una arquitectura muy similar a un centro de datos tradicional en las instalaciones. Sin embargo, la utilización de tecnologías de computación en la Nube (por ejemplo, virtualización), la flexibilidad, escalabilidad y la posibilidad de ser ejecutadas y gestionadas por agrupaciones de terceros, son los aspectos diferenciadores [MEL11] [CHA11].

Para construir una plataforma de calidad, se requiere experiencia interna, compartimiento de las responsabilidades entre el personal de la organización y sobre los servicios a ofrecer. En la Nube privada, la organización es responsable de todos los aspectos, de extremo a extremo.

Nube híbrida

La infraestructura de Nube híbrida es una composición de dos o más Nubes (privadas, públicas o comunitarias), que siguen siendo entidades únicas, pero están unidas por una tecnología estandarizada o patentada que permite la portabilidad de datos y aplicaciones (por ejemplo, escalabilidad en modo ráfaga hacia la nube pública en caso de que ocurra un evento que requiere una alta demanda de servicios de forma transitoria o esporádica) [CHO14][AHR10]. La Nube híbrida trajo consigo flexibilidad y beneficios de cada uno de los modelos de Nube [FEN12]. Existen diferentes reglas y procedimientos en cada modelo de implementación en la Nube.

Este entorno es el más complejo de las soluciones de Nube para implementar. El uso de datos dentro de la nube híbrida puede tener desafíos. Los archivos de registro de eventos de auditoría deben tomarse del sistema interno y externo, debiendo controlarlos y auditarlos correctamente. Para eso, la transferencia de datos debe ser segura.

Ambos entornos de nube deben admitir los mismos protocolos de seguridad y ser compatibles. Además, el uso del ancho de banda requerido durante los movimientos de datos debe ser considerado y estimado [DER14]

2.5.2 - Modelos de Entrega de Servicios

La computación en la Nube proporciona sus características en diferentes modelos de prestación de servicios y tiene tres predominantes modelos de aplicación. Si bien existen especializaciones y ramificaciones de las tareas en la Nube, se presentan los modelos propuestos por el NIST: Infraestructura como servicio (IaaS), Plataforma como servicio (PaaS), Software como servicio (SaaS) [MEL11]) y una extensión a modelos extendidos.

Infraestructura como Servicio (IaaS)

El modelo de infraestructura como servicio (IaaS, por sus siglas en inglés), significa que las máquinas virtuales simplemente se trasladan de las instalaciones a la Nube [BAR17].

Es decir, IaaS alquila servicios tales como computación, almacenamiento, redes y cualquier otra infraestructura fundamental; además de una pequeña cantidad de software (el hipervisor) para alojar máquinas virtuales (VM). Cada VM consiste en el sistema operativo, el software del sistema asociado y la aplicación en sí. Sin embargo, los usuarios no tienen la capacidad de administrar la infraestructura subyacente de la Nube. Es la estrategia de migración más fácil y rápida, ya que ofrece muchos beneficios, incluido el ahorro de costos. Sin embargo, todavía significa que su personal de operaciones necesitará realizar las siguientes operaciones: administración de parches, actualizaciones y mejoras. El uso de IaaS es uno de los patrones de implementación en la Nube más comunes hasta la fecha, porque reduce significativamente el tiempo entre la compra y la implementación. Además, debido a que su modo de operar es el más similar al de cualquier departamento de informática. Hoy en día, resulta de fácil incorporación para transformar la cultura y los procesos actuales de TI. La mayor parte de la migración, especialmente en las primeras fases de la adopción de la Nube, es hacia IaaS [RIG19] [GOR13].

Plataforma como Servicio (PaaS)

En la parte superior de IaaS, se proporciona la Plataforma como servicio (PaaS). En este caso, el proveedor de la Nube mantiene todo el software del sistema, eliminando la carga de actualizaciones y parches del departamento de operaciones. En un modelo de implementación de PaaS, la empresa necesita enfocarse en implementar su código en los recursos PaaS del proveedor. Este último se asegura que los sistemas operativos, el software de la base de datos, el software de integración y otras características estén disponibles y actualizadas y logren un alto nivel de SLA [RIG19].

PaaS proporciona a los departamentos de operaciones importantes beneficios, entre los que destacan: los ahorros de costos asociados con el mantenimiento reducido o nulo del software del sistema, y otras funciones de optimización. Sin embargo, PaaS generalmente implica cierto rediseño de la aplicación para un mejor aprovechamiento del modelo.

Como se podría esperar, los usuarios de PaaS no tienen ningún control sobre la plataforma o la infraestructura de la Nube, pero sí sobre la gestión de sus aplicaciones [RIG19][BAR17][GOR13].

Software como Servicio (SaaS)

En el modelo software como servicio (SaaS), simplemente se alquila o utiliza una aplicación de un proveedor. Algunos ejemplos de SaaS son los sistemas de CRM, correo electrónico o los escritorios virtuales [GOR13]. Es la opción más rentable de todas porque, por lo general, el único trabajo involucrado para el departamento de TI es aprovisionar usuarios y datos y, en ocasiones, integrar la aplicación con inicio de sesión único (SSO, Single Sign On). Generalmente, las aplicaciones SaaS se utilizan para funciones que no se consideran diferenciadoras o centrales del proceso de negocio; permitiendo que los esfuerzos de la empresa y sus trabajadores se orienten al desarrollo de las plataformas necesarias [ARM09].

En este modelo, los usuarios de SaaS no pueden controlar la infraestructura subyacente, la plataforma o las aplicaciones [MEL11][GUO07].

Otros Submodelos:

Más allá de las definiciones estándares definidas por NIST, existen especialmente en la industria, otras ramificaciones de servicios, tales como Contenedores como servicio o Kubernetes como Servicio [FLE19]; Pruebas como servicio; Base de datos como servicio; Seguridad como servicio e incluso Metadatos como servicio [DEY15]. A veces, estos tipos de tareas se agrupan en cuanto a XaaS, donde X implica una función en particular ofrecida como servicio [ARM09][RIM09].

2.6 - Virtualización basada en contenedores

La tecnología de virtualización existe desde hace más de 50 años [NEC15]. El término fue utilizado por primera vez por IBM durante la investigación de métodos eficientes de tiempo compartido, para el uso de recursos de hardware. Sin embargo, se volvió popular alrededor del año 2000, cuando comenzaron a surgir múltiples soluciones para recursos x86 (Servidores, computadoras de escritorio, etc.). Su principal motivación fue la necesidad de

ejecutar diversas instancias independientes, con sus propios sistemas operativos (SO) y bibliotecas, en un único hardware físico.

La mayor ventaja de este enfoque consiste en el ahorro de costos, que se obtiene al ejecutar múltiples instancias virtuales en el mismo hardware, permitiendo una mejora en la utilización de sus recursos [YEO06]. Otra ventaja es el aislamiento que ofrece. Las aplicaciones pueden requerir diferentes versiones de sus dependencias y bibliotecas. Sin esta tecnología, las aplicaciones entrarían en conflicto, no se podrían ejecutar en el mismo host y requerirían recursos (hosts) dedicados. Entonces, la aplicación podría usar solo una fracción del hardware, significando un desperdicio de capacidades computacionales.

Existen dos tipos básicos de mecanismos de virtualización: a) a través de hipervisores b) a nivel de sistema operativo, cómo se detalla en la figura 2.8.

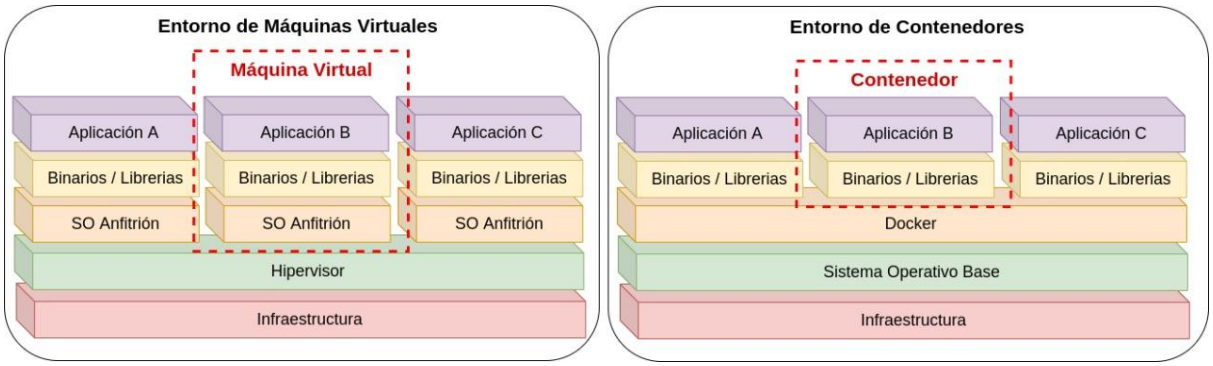

Figura 2.8 - Arquitectura de virtualización basada en hipervisores y contenedores (Docker). Fuente: Propia. Versión adaptada de [PON17].

El método tradicional se llama “virtualización basada en hipervisor” [SAN18]. Este es una pieza de software (o combinación de software y hardware) que ejecuta y administra máquinas virtuales. Las instrucciones sensibles del sistema, como las llamadas al sistema de entrada/salida (E/S) son atrapadas y traducidas por el hipervisor [ROB00]. Además de la tecnología basada en hipervisores existe una nueva alternativa que utiliza grupos de control de Linux y espacios de nombres, para generar recursos aislados en el sistema operativo host (contenedores). Los procesos dentro de los contenedores no conocen otros procesos externos. También pueden tener un árbol de directorios dedicado. Más importante aún, los objetivos fundamentales de las VM (*Virtual Machine*) y los contenedores son diferentes: el propósito de una VM es emular completamente un entorno; mientras que el de un contenedor es hacer que las aplicaciones sean portátiles y autónomas. A continuación, en la Tabla 2.1 se presenta un resumen de estas características diferenciales entre arquitecturas.

Tabla 2.1 - Comparativa rápida de las características más relevantes de ambos esquemas de virtualización. Fuente: Propia.

Criterio	Máquinas Virtuales	Contenedores
Tamaño de imagen	4x / 3x	x
Tiempo de inicialización	>10 segundos	**1 segundo**
Overhead de CPU	>15%	**<5%**
Overhead de I/O en Disco	>50%	**Despreciable**
Aislamiento de recursos	Bueno	Bueno (en avance)
Seguridad	Baja-Media	Media-Alta
Flexibilidad del SO	Excelente	Buena (en avance)
Administración	Madura	Buena - Madura (en avance)
Migración de aplicaciones	Baja - Media	**Excelente**

2.6.1 - Hipervisores

Los hipervisores permiten que varias máquinas virtuales se ejecuten simultáneamente en un mismo host. Ofrecen una plataforma donde las VM pueden ejecutarse aisladas unas de otras y del host. El hipervisor ofrece una ilusión de hardware nativo dedicado a las VM. Se realiza atrapando una variedad de instrucciones sensibles entre la máquina virtual y el hardware del host. Las directrices que acceden a los recursos del host, como los discos físicos, se consideran confidenciales [ROB00]; y están atrapadas y traducidas, ya que de lo contrario podrían romper la ilusión de que un sistema operativo invitado no sea el único en el sistema. Este tipo de virtualización se llama *virtualización completa*. Sin embargo, el rendimiento de E/S de la virtualización completa se reduce debido a la captura y traducción de las llamadas. Para mejorar el rendimiento y la comunicación entre la VM y el SO del equipo anfitrión, se utiliza la para virtualización. La para virtualización es una técnica en la que la computadora virtual es consciente de que se está ejecutando en un entorno virtualizado [SHA08]. Al usar la para virtualización, no hay necesidad de capturar y traducir, porque tanto el sistema operativo host como el invitado pueden cooperar. Todos los sistemas operativos modernos disponen de esta característica habilitada, lo que se considera una funcionalidad madura e implementada en el despliegue de VM del mundo real [PEN10].

2.6.2 - Contenedores

Una alternativa a la virtualización tradicional basada en hipervisor es la virtualización a nivel de sistema operativo. Este tipo de virtualización existe desde hace muchos años [SOL07]. Chroot [CHR21], desarrollado en 1982, se considera la primera herramienta de virtualización a nivel de sistema operativo. Otras herramientas similares, como Linux-Vserver [POT07][POT05] y OpenVZ [VAS16] [SWS16], han surgido con posterioridad. Además, existen algunas herramientas de virtualización a nivel de sistema operativo basadas en Unix, como las jaulas (*jails*) de FreeBSD [KAM00] y las zonas de seguridad de Solaris [PRI04]. Sin embargo, sus desarrolladores no trabajaron de manera integrada en sus herramientas para incluir estas características en el núcleo principal de Linux.

La diferencia entre una máquina virtual basada en hipervisor y un contenedor es que este último no ejecuta un sistema operativo invitado dentro de él [XAV13][SOL07]. En cambio, los contenedores comparten partes del núcleo del host para proporcionar una funcionalidad similar a la que ofrece el sistema operativo a las aplicaciones.

El aislamiento de la virtualización basada en contenedores se logra mediante el uso de espacios de nombres de Linux. Estos se utilizan, por ejemplo, para controlar qué parte del sistema de archivos del host puede ver el contenedor y qué procesos son administrados en el árbol de procesos del mismo. La gestión de recursos de los contenedores es manejada por grupos de control de Linux (CPU, memoria y red) [FEL15]. Los contenedores se utilizan para ejecutar aplicaciones donde el contenedor solo contiene binarios, bibliotecas y otras dependencias de la aplicación que la aplicación requiere.

En la actualidad, el despliegue de las nuevas funcionalidades y versiones de una aplicación son cada vez más veloces. Sin embargo, el software en sí no se vuelve más simple. Por el contrario, los proyectos de software aumentan en complejidad.

Los contenedores son una tecnología increíblemente poderosa que proporciona a los desarrolladores y a los administradores de sistemas (DevOps) grandes ganancias a la hora de pensar en rendimiento y productividad ya que mediante el uso de contenedores es posible:

- Implementar software de manera ágil
- Realizar copias de seguridad más simples
- Replicar y mover aplicaciones y sus dependencias de forma rápida y sencilla.

Esta tecnología está cambiando fundamentalmente la forma en que se desarrolla, distribuye y ejecuta el software. Los desarrolladores pueden crear y desplegar su software localmente,

sabiendo que se ejecutará de manera idéntica independientemente del entorno del host, ya sea un servidor tradicional, un equipo personal, o un recurso en la nube [MOU16]. Los ingenieros de operaciones pueden concentrarse en la creación de redes, los recursos y el tiempo de actividad, y dedicar menos tiempo a configurar entornos y luchar contra las dependencias del sistema. El uso y la penetración de este paradigma está aumentando a un ritmo exponencial en toda la industria, desde las empresas más pequeñas hasta las de gran escala. En la figura 2.9 se compara, teniendo en cuenta los últimos diez años, la popularidad y adopción de la tecnología de contenedores respecto de las máquinas virtuales tradicionales.

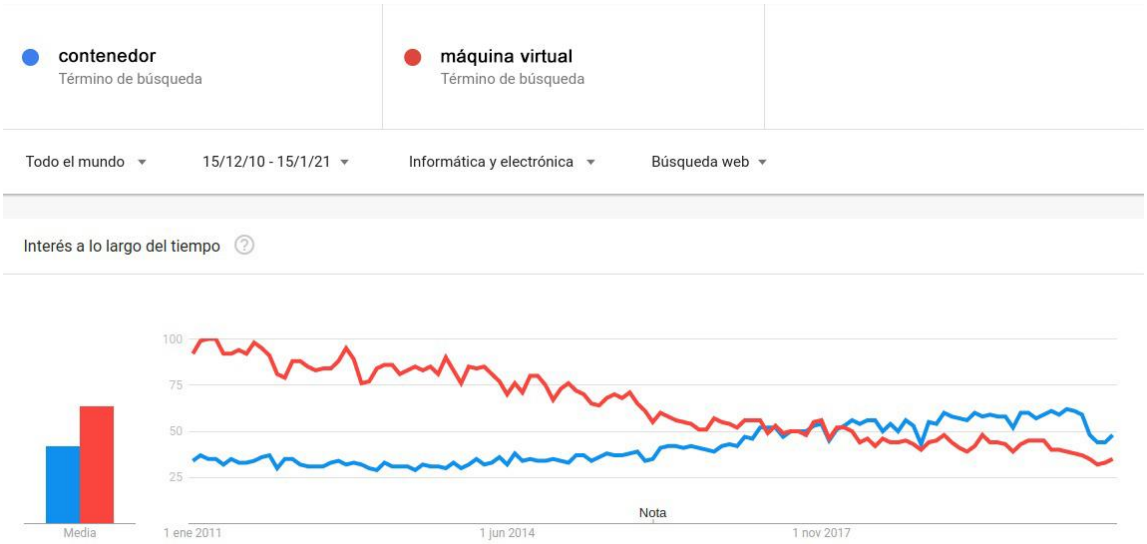

Figura 2.9 - Popularidad de contenedores Docker y máquinas virtuales en los últimos 10 años. Fuente: Google Trends. Recurso: https://trends.google.com.ar/trends/explore?cat=5&date=2010-12-15%202021-01-15&q=container,virtual%20machine

2.6.4 - Docker como plataforma de contenedores

Docker se basa en un modelo Cliente/Servidor (C/S) que se ejecuta dentro de un host (equipo anfitrión) [JAR16]. En ese ejecutable central (core), funcionan muchos procesos complejos, como varios controladores de sistemas de archivos, traducciones físicas/virtuales, cgroups, espacios de nombres y otros mecanismos del kernel básicos [MAT18]. Estas características básicas se presentan, de forma gráfica en la figura 2.10.

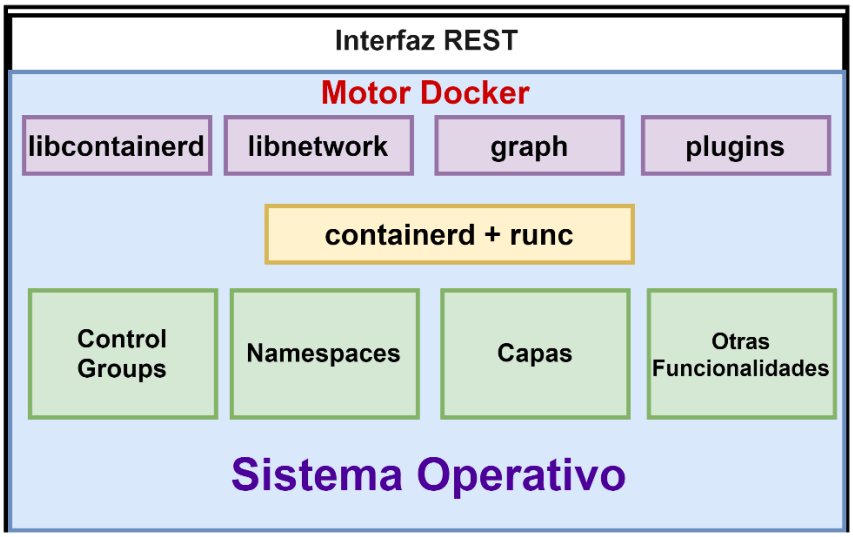

Figura 2.10. Arquitectura de Docker simplificada. Fuente: Propia. Versión adaptada de [SCH20].

Para describir la arquitectura de Docker se pueden mencionar los siguientes elementos esenciales:

- Capa base: Sistema operativo Linux con sus Cgroups, espacios de nombres y capacidades de capas (layers), y otras funciones propias del SO.
- Capa Docker Engine: Por encima de la capa base, se encuentra el motor Docker. Ofrece una interfaz RESTful para el mundo exterior a la que se puede acceder mediante cualquier herramienta compatible.
- Capa de objetos Docker: Imágenes y Registros en Docker.

Docker proporciona un enfoque rápido y automatizado para implementar una aplicación dentro de contenedores portables. De esta manera, la aplicación sería adecuada para correr en cualquier entorno y cuenta con capacidades para escalar e interactuar con el mundo exterior. Esta función resuelve el problema incipiente de todo desarrollo, la integración de librerías, código y dependencia para los desarrolladores y hace que la implementación, el despliegue y la prueba de las aplicaciones sean más fáciles y rápidas. Esto es especialmente beneficioso para los flujos de trabajo de CI/CD [CHE18][SHA18].

Docker además brinda la posibilidad de configurar componentes de infraestructura, como CPU, memoria o redes, al usuario a través de sus archivos de configuración. Por lo tanto, los desarrolladores pueden administrar la infraestructura de la misma manera que se tratarán en el entorno de ejecución real. Para organizar tales características, Docker introduce un kernel y una API a nivel de aplicación en el contenedor de Linux, que ejecutará procesos como CPU, E/S y memoria de forma aislada [MAT18][BHA17]. Además, para implementar y ejecutar

aplicaciones contenidas, Docker utiliza dos de las características más importantes del kernel de Linux: Cgroups y Namespaces. Esto permite, como se explicó, ejecutar las aplicaciones en un espacio seguro y aislado dentro de los contenedores de Docker. Por lo tanto, al usar Docker, se pueden ejecutar múltiples aplicaciones o microservicios simultáneamente en la parte superior del mismo host físico. De acuerdo con esta posibilidad, los desarrolladores pueden dividir un sistema grande en un conjunto más pequeño de servicios, cada uno de los cuales se puede implementar como un contenedor Docker separado. De esta forma, la depuración, gestión y actualización de cada componente resulta más fácil. Al ser más liviano que los hipervisores convencionales, Docker puede utilizar los recursos de hardware de manera más eficiente y puede administrar su carga de trabajo de forma dinámica.

Componentes esenciales de Docker

Imágenes Docker

Proporciona el código fuente para crear y ejecutar contenedores. Esta imagen contiene todas las bibliotecas y binarios necesarios para construir y ejecutar una aplicación. Hay recursos de Docker preconstruidas, que los desarrolladores pueden descargar y usar. Sin embargo, también se pueden crear imágenes personalizadas.

Una imagen personalizada siempre comienza con una imagen base. Para combinar diferentes capas de una imagen y tratarlas como una sola, Docker utiliza un sistema de archivos especial llamado Union File System (UnionFS) [QUI06]. Esto permite combinar archivos y directorios de diferentes sistemas en un único recurso consistente [QUI06]. Al aplicar cualquier cambio a la imagen, se agrega una nueva capa encima de las existentes.

Dockerfile

Es una plantilla que contiene todas las instrucciones necesarias para crear una imagen de Docker. El motor de Docker adquiere la información necesaria para configurar un contenedor, o ejecutar las aplicaciones en un contenedor desde este archivo. Estas instrucciones se ejecutarán en orden, mediante la ejecución de un comando de "compilación de Docker" (docker build) presentado por el usuario. En caso de que alguna de esas instrucciones resulte en un cambio en el contenido actual de la imagen, este se aplicará a una nueva capa basada siguiendo el enfoque de múltiples capas (UnionFS), cómo ya se describió.

Contenedor Docker

Los contenedores Docker son las instancias en ejecución creadas e implementadas a partir de imágenes de Docker y los recursos del sistema asignados. Cabe recordar que se pueden

lanzar varios contenedores en el mismo host, y todos están aislados entre sí y actúan por separado. Al iniciar un contenedor, se crea una nueva capa de escritura, a saber, "capa contenedora", sobre la imagen utilizada [MAT18]. Cualquier cambio que se produzca dentro del contenedor en ejecución solo se aplica a la capa de lectura/escritura del contenedor actual. Cuando se ejecutan diferentes contenedores sobre la misma imagen, escriben los cambios en sus propias capas de lectura/escritura. Cuando el contenedor se detiene, los cambios realizados se pierden, salvo que se utilice un almacenamiento externo.

Registro de Docker

El registro de Docker es donde se pueden almacenar las imágenes de Docker. El objetivo principal de un registro es simplificar la distribución de imágenes. Por lo tanto, los desarrolladores pueden crear imágenes de Docker y almacenarlas en los registros. Luego, al acceder a ellos mediante el cliente Docker, otros usuarios pueden descargar y usar las imágenes disponibles. Podemos considerar estos registros similares a los repositorios de código fuente [BHA17]. El registro de Docker puede ser público o privado. Docker Hub [MAT18] es el registro público más popular, donde puede registrarse libremente, y obtener acceso a un enorme sistema de carga y descarga de imágenes. Docker Registry [MAT18], un proyecto de código abierto liderado por Docker, proporciona un repositorio privado para organizaciones con acceso autenticado, que les permite controlar el acceso a sus imágenes.

2.7 - Orquestación de Contenedores

Con el crecimiento de las tecnologías de virtualización se ha vuelto muy popular [GER15], especialmente con el lanzamiento de Docker, y con el creciente interés de los proveedores de infraestructura IT y servicios PaaS en la virtualización de aplicaciones. Teniendo en cuenta los beneficios antes mencionados, la industria de IT ya ha adoptado cómo un estándar de facto las tecnologías de contenedores [MAT18].

Sin embargo, operar con contenedores a una escala de producción con múltiples servicios y proyectos, requiere características más avanzadas que las ofrecidas nativamente por los motores de virtualización basada en sistemas operativos. Estas cargas de trabajo deben estar preparadas para atender despliegues de aplicaciones de carácter dinámico, procesos de automatización, gestión, escalado y supervisión de aplicaciones empaquetadas de varios contenedores. Para ello, es necesario contar con un producto de orquestación que permita realizar esos procesos de forma ordenada, segura y automatizada [PAN19].

Un sistema de orquestación debe implementar y distribuir procesos en varios hosts físicos, monitorear y realizar un seguimiento del estado de los servicios, recursos físicos y

contenedores. Para cumplir con este objetivo, un framework de administración de contenedores emplea capas de software, las cuales se encargan de abstraer la complejidad de los equipos físicos y mostrar todo el clúster como un solo grupo de recursos. Además, los contenedores dentro de un clúster pueden comunicarse entre sí, independientemente del host físico en el que estén implementados [JAW19]. Para ello, es necesario también que el sistema de gestión cree redes virtuales, mecanismos de detección y descubrimiento de servicios, y balanceo de carga entre equipos y servicios.

2.7.1 - Kubernetes como plataforma de Orquestación de contenedores

Kubernetes es un administrador de clústeres de código abierto, que inicialmente fue desarrollado e introducido por Google en su Foro de desarrolladores en junio de 2014. Es un sucesor de Borg [BUR16][VER15], el sistema de orquestación interno de Google que acumuló más de una década de la experiencia del gigante tecnológico en la ejecución de grandes cargas de trabajo empresariales en producción. En 2014, Google decidió ampliar el ecosistema de contenedores al compartir Kubernetes con la comunidad nativa de la nube. Kubernetes se convirtió en el primer proyecto graduado de la Cloud Native Community Foundation (CNCF) [CNC20] [VAY19], una organización concebida por Google y Linux Foundation como el principal impulsor del emergente movimiento nativo de la nube.

El objetivo principal de la plataforma es automatizar la implementación y la administración (por ejemplo, actualización, escalado, seguridad, redes) de aplicaciones en contenedores en grandes grupos de computadoras distribuidas [ROS20][PER19][ABD19]. Con este fin, la plataforma ofrece una serie de primitivas (API), opciones de implementación, redes [DEW19], interfaces de almacenamiento y contenedores, seguridad incorporada y otras características útiles.

En estos seis años de vida, Kubernetes ha logrado posicionarse fuertemente en el mercado debido a su versatilidad para la administración de microservicios y en cualquier tipo de infraestructura [KAP20]. Tanto así que compañías reconocidas como Spotify, Pokemon Go, Huawei, IBM, eBay, Nokia, SAP y Bose son un ejemplo de casos de éxito por la aplicación de este tipo de arquitectura.

Capacidades Básicas de Kubernetes:

- Gestión de grupos de contenedores.
- Proporcionar herramientas para implementar aplicaciones.
- Escalar aplicaciones cuando es necesario.

- Gestión de cambios en las aplicaciones existentes.
- Ayudar a optimizar el uso del hardware subyacente debajo de los contenedores.
- Permitir que los componentes de la aplicación se reinicien y se muevan por el sistema cuando es necesario.

Razones para adoptar Kubernetes

Kubernetes tiene varias características relevantes que le permiten implementar aplicaciones más rápido teniendo en cuenta la escalabilidad [ABD19]:

- Escalado horizontal de la infraestructura: se pueden agregar o eliminar nuevos servidores y servicios fácilmente.
- Autoescalado: permite, automáticamente, cambiar el número de contenedores en ejecución, según la utilización de la CPU u otras métricas proporcionadas por la aplicación.
- Escalado manual: escale manualmente el número de contenedores en ejecución mediante un comando o la interfaz.
- Controlador de replicación: el controlador de replicación se asegura de que su clúster tenga la misma cantidad de pods (contenedores) en ejecución que los que se especificaron. Si hay demasiados pods, el controlador de replicación finaliza los pods adicionales. Si hay muy pocos, se inician más instancias de contenedores.
- Verificaciones de estado y autorreparación: Kubernetes puede verificar el estado de los nodos y contenedores para asegurarse de que su aplicación no tenga fallas. Kubernetes también ofrece autorreparación y reemplazo automático para que no tenga que preocuparse por si falla un contenedor o un pod.
- Enrutamiento de tráfico y equilibrio de carga: el enrutamiento de tráfico envía solicitudes a los contenedores adecuados. Kubernetes también viene con balanceadores de carga integrados para que pueda equilibrar los recursos para responder a interrupciones o períodos de alto tráfico.
- Implementaciones y reversiones automatizadas: Kubernetes maneja las implementaciones de nuevas versiones o actualizaciones sin tiempo de inactividad mientras monitorea el estado de los contenedores. En caso de que la implementación no funcione bien, se revierte automáticamente.

- Implementaciones Canary: las implementaciones Canary le permiten probar la nueva implementación en producción en paralelo con la versión anterior.
- Kuberntetes es independiente del proveedor. Ser independiente del proveedor permite a los operadores diseñar, construir y administrar plataformas de nube híbrida y de múltiples nubes de manera fácil y segura sin riesgo de dependencia del proveedor. Kubernetes también elimina las preocupaciones del equipo de operaciones sobre una estrategia compleja de nube híbrida / múltiple.

Desafíos para la adopción de Kubernetes

Kubernetes ofrece infinidades de ventajas. Sin embargo, la transición a Kubernetes se asocia con numerosos desafíos que deben ser abordados por las empresas e instituciones que buscan adoptar esta tecnología. Los puntos más importantes se asocian a las siguientes características:

- Curva de aprendizaje pronunciada.
- Migración de aplicación a contenedores.
- Configuración, instalación e integración de herramientas.
- Configuración de alta disponibilidad.
- El personal de administración debe estar muy bien capacitado y tiende a ser costoso.

A pesar de las desventajas, Kubernetes sigue siendo una excelente opción para la orquestación de contenedores.

Objetos básicos de Kubernetes

Kubernetes dispone de varios objetos que permiten representar y definir el estado de un sistema. Dentro de las más importantes se encuentra:

- Aplicaciones contenerizadas desplegadas y cargas de trabajo
- Sus recursos de red y almacenamiento asociados
- Información adicional acerca de lo que el clúster está haciendo en un momento dado.

A continuación, se detallan los objetos más relevantes de Kubernetes.

Pod

La unidad de computación más pequeña implementable en Kubernetes se llama pod [PHI21]. Cada pod consta de uno o más contenedores de aplicaciones, que se implementan en el mismo host físico y comparten el conjunto de recursos (como almacenamiento y redes). En otras palabras, un pod modela un "host lógico" específico de la aplicación e incluye diferentes contenedores que están estrechamente acoplados [PHI21], por lo que se colocan y operan en un nivel superior que los contenedores individuales [RAU21].

Kubernetes aplica sus mecanismos de programación y orquestación en la parte superior de los pods, en lugar de los contenedores. Esto permite a los desarrolladores implementar varios contenedores de sus microservicios fuertemente relacionados, y luego empaquetarlos en un único módulo para que actúen como una sola aplicación.

Kubernetes asigna una dirección IP particular a cada pod. Además, los contenedores del mismo pod comparten directorios y recursos.

Controlador de replicación (ReplicaSet)

Los controladores de replicación comprueban el estado de los pods y mantienen el número deseado de ellos, en un momento dado [REE20].

Despliegues (Deployments)

El despliegue es un concepto de nivel superior que el replicaSet y pod. Al definir una implementación, podemos declarar actualizaciones para Pods y ReplicaSets [SHA20]. En otras palabras, se describe un estado deseado para que el controlador de implementación compare el estado real con el deseado.

Servicios (Services)

Un servicio es una forma abstracta de exponer a los usuarios las aplicaciones que se ejecutan en los pods. Como se mencionó, se asigna una dirección IP única a cada pod dentro de un clúster. Sin embargo, dado que los pods pueden crearse o eliminarse mediante el controlador de réplicas, sus direcciones IP no son estables. Cada pod recién creado recibirá una nueva IP; por lo tanto, no sería un enfoque adecuado que los clientes se conecten a las aplicaciones a través de las direcciones IP cambiantes. Aquí es donde los servicios de Kubernetes resuelven el problema ofreciendo una API con puntos de acceso (endpoints) que permite que los servicios sean accesibles externamente. Además, al asignar un único nombre DNS a cada servicio, Kubernetes proporciona un mecanismo de descubrimiento de servicios interno y los desarrolladores no necesitan utilizar un enfoque externo para ello [RAU21][SHA20].

Arquitectura de Kubernetes

Kubernetes está compuesto de varios componentes interrelacionados que permiten brindar las funciones a los nodos que los componen. Desde una perspectiva muy general, Kubernetes tiene los siguientes componentes principales:

- Uno o más "Nodos Master".
- Uno o más "Nodos Worker".
- Un almacén de valores key-value, distribuido (etcd).

En la figura 2.11, se presenta una arquitectura básica de este orquestador.

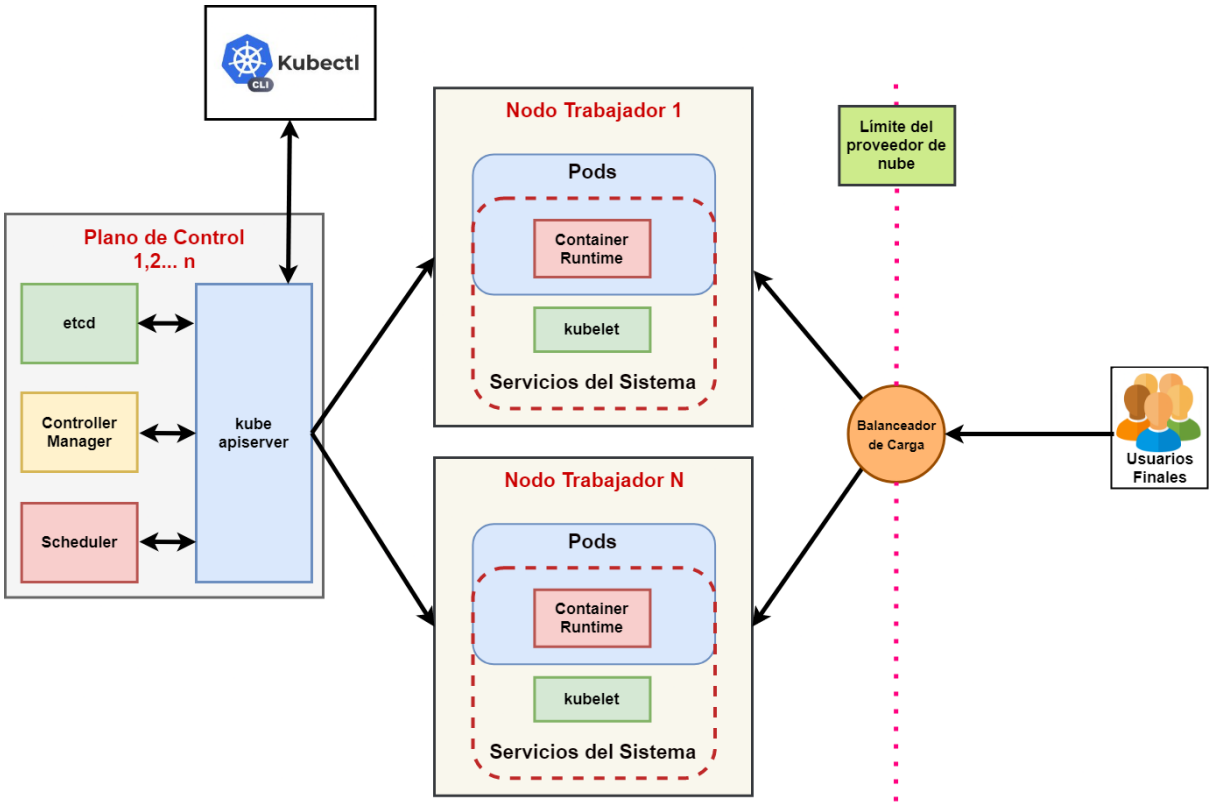

Figura 2.11 - Arquitectura básica del orquestador Kubernetes (Estructura Maestro-Esclavo). Fuente: Propia. Versión adaptada de https://platform9.com/wp-content/uploads/2019/05/kubernetes-constructs-concepts-architecture-610x476.jpg

A continuación, se detallan brevemente los elementos más relevantes de cada tipo de nodo.

Componentes del nodo maestro

El nodo maestro, es decir la unidad de control del clúster de Kubernetes, es responsable de mantener y administrar el registro de estado de todos los objetos del sistema.

Para responder a los cambios de estado, realiza ciclos de control continuos y se asegura de llevar el estado real del clúster al deseado, según lo descrito por los usuarios [PHI21]. Con objeto de cumplir este objetivo, el nodo principal de Kubernetes adopta una colección de servicios que administran el estado del clúster, que se describen a continuación:

Servidor API

El servidor API expone una API REST la cual hace posible la comunicación de todos los componentes del clúster. También permite a los usuarios configurar y validar todos los objetos del sistema, como pods, controladores de replicación, servicios, etc. [DOB20]. Además, el servidor API maneja las comunicaciones entre los nodos maestros y trabajadores.

Controlador de Administración

Es un bucle continuo de control que se ejecuta verificando el estado del clúster. Si surge algún cambio, el administrador del controlador mueve el estado actual del clúster hacia el deseado. Hay diferentes controladores en Kubernetes, incluido el de replicación, de espacio de nombres, de puntos de acceso y de cuentas de servicio.

Programador

El programador actúa como un controlador de recursos y maneja la carga de trabajo de un clúster. Más precisamente, es responsable de asignar pods a los diferentes nodos trabajadores en función de varias métricas, como los recursos informáticos de los nodos, las restricciones de política de los pods y los requisitos de calidad del servicio. Además, se encarga de realizar un seguimiento de la descripción general de los recursos y observar cuáles están libres u ocupados.

Base de datos de estados (etcd)

Etcd es un almacén de datos de tipo clave/valor distribuido, uniforme, de alta disponibilidad y ligero, que se utiliza para almacenar todos los datos del clúster, incluidas las configuraciones y la información de estado. Para operar como un almacén de datos, etcd actúa basándose en el algoritmo de consenso de Raft [PHI21][LAR20].

Componentes del nodo trabajador

Los nodos trabajadores son los equipos en los que se ejecutarán finalmente las aplicaciones. El usuario a menudo no tiene interacción con estos nodos. Es el nodo maestro el que controla a cada uno de los trabajadores. Cada uno de ellos consta de algunos componentes que se describen a continuación:

Container Runtime

Como primer componente, se requiere instalar un “tiempo de ejecución de contenedor” en cada nodo trabajador. Este es el software responsable de ejecutar los contenedores. Kubernetes admite varios tiempos de ejecución; sin embargo, Docker, Containerd y CRI-O son los más populares [ESP20].

Kubelet

Kubelet es el componente más importante en este tipo de nodos, ya que es el responsable de la gestión de los pods y los contenedores en ejecución. Este servicio recibe las instrucciones y notificaciones del nodo maestro e interactuando con etcd, actualiza las configuraciones. Según la información adquirida, se asegura de que los pods estén en buen estado y funcionen correctamente. Luego, también informa el estado de los nodos al clúster.

Proxy (Kube-Proxy)

Kube-Proxy es un proxy de red que se encarga de las reglas de red en cada nodo trabajador. Estas permiten las comunicaciones de red con los pods desde dentro o fuera del clúster de Kubernetes, proporcionando acceso y balanceo de carga por los protocolos TCP, UDP y asociados.

2.8 - Dispositivos móviles: Smart Devices

2.8.1 - Introducción

Los avances en la tecnología informática y las comunicaciones han impulsado un nuevo concepto de informática llamado computación móvil, en el que los usuarios que llevan dispositivos de este tipo tienen acceso a una infraestructura compartida, independientemente de su ubicación física [LIH16][ZHA16]. Esto proporciona una comunicación flexible entre personas e, idealmente, acceso continuo a los servicios en red [PRI14].

El estudio de esta nueva área de la informática ha motivado la necesidad de repensar detenidamente la forma en que se conciben las redes y los sistemas portátiles. Aunque los sistemas distribuidos, tradicionales y móviles, pueden parecer estrechamente relacionados, hay una serie de factores que los diferencian, especialmente en términos de tipo de dispositivo (fijo/móvil), conexión de red (permanente/intermitente) y contexto de ejecución (estático/dinámico).

Con los avances recientes en procesadores de baja potencia, los dispositivos portátiles pueden realizar operaciones computacionalmente intensivas. Esto hace que, por ejemplo, los

teléfonos inteligentes se vuelvan cada vez más populares y generan mayor atracción hacia las personas, porque crean nuevas áreas de aplicación y aumentan su capacidad, en términos de potencia computacional, servicios, sensores y comunicación [SAR13].

Otra de las ventajas de estos equipos, es que permiten a los usuarios transportar la oficina a cualquier lugar, en cualquier momento [SAR13].

A pesar de sus diferentes capacidades de hardware (por ejemplo, resolución de pantalla, calidad de la cámara, disponibilidad de los sensores) pueden compartir el mismo sistema operativo (por ejemplo, Android), de modo que es posible tener la misma pila de software en una amplia variedad de dispositivos, lo que permite implementar aplicaciones sin problemas en un dispositivo u otro.

Estas consideraciones permiten observar que un conjunto de dispositivos móviles puede, potencialmente, explotarse como un sistema informático distribuido, donde cada nodo está representado por un artefacto, y la infraestructura de interconexión se basa en tecnologías de comunicación inalámbrica.

2.8.2 - Características básicas de un dispositivo inteligente

Dentro de las características más destacadas de los dispositivos móviles [SAR19][BIC13][KRO08], es posible encontrar:

- Pantalla táctil, ideal para lograr un fácil uso y navegación.
- Capacidad de realizar múltiples tareas en simultáneo.
- Cámara digital integrada, que permite tomar capturas instantáneas y participar de videollamadas.
- Conexiones inalámbricas: Conexión Bluetooth, WiFi y antenas 4/5G.
- Agendas y notas electrónicas, permitiendo administrar contactos y eventos.
- Sensores y actuadores:
    - Acelerómetros.
    - Lectores de huellas dactilares para seguridad y desbloqueos.
    - Sensores básicos:
        - Giroscopio.

        - Proximidad.
        - Brújula.
        - Barómetro.
    - GPS, lo que permite disponer de un equipamiento potente tanto para localización como para navegación en diversos medios de transporte.
    - NFC, para realizar pagos y recargas.
- Instalación de aplicaciones adicionales. Esto permite incrementar las funcionalidades, el procesamiento de datos y la conectividad. Estas aplicaciones pueden ser desarrolladas por el fabricante del dispositivo, por el operador o por un tercero.

2.8.3 - Android como Sistema Operativo

Android es uno de los sistemas operativos más nuevo, adoptado y diseminado alrededor del mundo para el uso de los dispositivos inteligentes [SAR19]. Android ha sido desarrollado y construido en conjunto por muchas empresas comerciales como Google, Samsung, Motorola y Dell y es de código abierto.

Los dispositivos inteligentes basados en Android brindan conectividad y capacidad informática avanzada en comparación con otros sistemas operativos de equipamiento móvil ya que facilita la comunicación entre el hardware y el software con la interfaz de usuario [TON13]. Es importante tener en cuenta que Android no solo es utilizado en smartphones, sino que también puede ser instalado en otros tipos de dispositivos, como por ejemplo hornos, aires acondicionados, refrigeradores, lavadoras, etc. de manera que las características de estos dispositivos pueden ser actualizada y extendida en base a las necesidades del usuario o proveedor.

Entre sus principales características se puede destacar que está basado en el Sistema Operativo Linux (software libre), y que cuenta con la posibilidad de agregar aplicaciones o programas para extender funcionalidades en los dispositivos [KHO18][YAD15]. Estas aplicaciones en su mayoría no vienen instaladas en el teléfono, por lo que es indispensable disponer de conexión a Internet (Wi-Fi o 4/5G) para sacar el máximo partido al equipamiento.

Las aplicaciones de Android se desarrollan (nativamente) con los lenguajes de programación Java o Kotlin. Para su ejecución se requiere de la máquina virtual de Android, es decir, DVM (Dalvik Virtual Machine) [SHA17][HYE12]. Esta DVM se utiliza para ejecutar las aplicaciones y producir un archivo. dex, es decir, archivo ejecutable Dalvik.

Android consta de cuatro capas básicas [NET15][WHA11] que se muestran en la figura 2.12 a continuación. A saber:

1. Capa de aplicación.
2. Capa de Framework de aplicación.
3. Capa de Tiempo de ejecución y bibliotecas de Android.
4. Capa de Kernel de Linux.

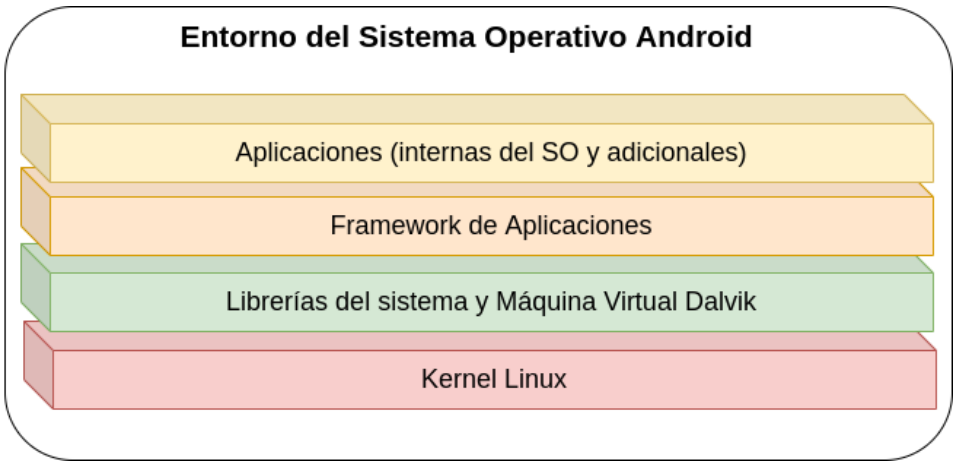

Figura 2.12 - Capas básicas que componen la arquitectura del Sistema Operativo Android. Fuente: Propia.

1. **Capa de aplicación:**

Es la capa superior de la arquitectura de Android, y se utiliza para la instalación de aplicaciones, por ejemplo, teléfono, correo, calendario, etc. La mayoría de las aplicaciones son nativas, como mapas de Google, cámara, navegador, SMS, contactos y calendarios [NET15]. El marco y entorno de aplicación del usuario final se utiliza para operar estas aplicaciones.

2. **Framework de aplicación:**

El framework ofrece diferentes paquetes que las aplicaciones pueden utilizar como un servicio. Esta capa incluye clases y funciones necesarias para el despliegue de aplicaciones Android. Los desarrolladores reutilizan y amplían los componentes que ya están presentes en la interfaz de programación de las mismas [WHA11]. Las principales funciones se detallan a continuación:

- Administrador de actividades: controla y gestiona adecuadamente todas las actividades y se ocupa de su ciclo de vida.
- Administrador de recursos: proporciona acceso de manera segura a los recursos del sistema.
- Administrador de notificaciones: permite que todas las aplicaciones, si lo requieren, muestran alertas personalizadas en la barra de estado.
- Administrador de ubicación: cuando el usuario ingresa o sale de una ubicación geográfica en particular, notifica y registra los movimientos y alertas.
- Administrador de paquetes: los datos sobre los paquetes instalados en el dispositivo se recuperan con el uso del administrador de paquetes.
- Administrador de telefonía: maneja la configuración de conexión de red y toda la información sobre los servicios en el dispositivo.
- Administrador de ventanas: Permite interactuar y crear vistas y diseños para las aplicaciones.
- Proveedor de contenido: Permite que las aplicaciones accedan a los datos de otras aplicaciones y al sistema de archivos del equipo.

3. **Tiempo de ejecución y bibliotecas de Android:**

Esta capa contiene un componente central llamado máquina virtual Dalvik (DVM) [SHA17][HYE12], y cada proceso se ejecuta en una instancia separada. Es decir, es una instancia independiente en la DVM. Debido a la propiedad básica de DVK, los usuarios pueden ejecutar múltiples aplicaciones al mismo tiempo [WHA11].

Además de la DVM, Android posee sus propias librerías y bibliotecas. Estas, están escritas en C/C++ y no se pueden acceder directamente. Con la ayuda de la segunda capa (framework) las aplicaciones y servicios pueden acceder a ellas. Existen numerosas librerías ya desarrolladas que permiten, por ejemplo, acceder a navegadores web, reproducir material de audio y video, etc. [WHA11].

4. **Kernel de Linux:**

Esta capa es el núcleo de la arquitectura de Android. Proporciona un sistema operativo estable y probado para plataformas móviles y ofrece servicios como administración de memoria, administración de energía, seguridad, etc. [WHA11].

Esta capa ofrece numerosas funciones de seguridad claves para el funcionamiento y evolución constante del sistema Android [KHO18][KAR17], que incluyen:

- Un modelo de permisos definido por el usuario.
- Aislamiento de procesos.
- Procedimiento disponible para IPC (inter-process communication) seguro.
- Capacidad para eliminar partes innecesarias e inseguras del kernel.

Para cada usuario del sistema, Android separa y protege los recursos correspondientes.

- No permite el acceso de lectura a los datos de los demás usuarios.
- No permite el uso de espacio de memoria de otro usuario.
- No permite el uso de recursos de CPU de los demás.
- Evita que un usuario utilice los dispositivos de los demás (GPS, telefonía, Bluetooth, etc.) [NET15].

2.8.4 - Arquitectura de procesadores ARM y masividad de dispositivos

La arquitectura de procesadores más utilizada en los dispositivos móviles corresponde a la empresa ARM (por sus siglas en inglés “Advanced RISC Machine”). Sus chips se basan en la arquitectura RISC [IEE91] (por sus siglas en inglés “Reduced Instruction Set Computing”), la cual se compone por un conjunto de instrucciones reducidas. Una arquitectura RISC típica consta de un único archivo de registro uniforme, arquitectura de carga y almacenamiento y dispone de un modo de direccionamiento simple e instrucciones de longitud fija. Debido a estas características, consigue un alto rendimiento respetando un bajo consumo de energía [CRI19][BHA19][PAN11] y un tamaño reducido de chip [BLE13].

Aunque los procesadores y GPU de los dispositivos móviles son relativamente más lentos que los servidores tradicionales, la “brecha” de rendimiento se está cerrando rápidamente. De esta manera, varios artículos de investigación [GED20][BAR19][LAV19][HIR18][SCH18][MAC17] y los servicios de Benchmarking como PassMark y CPU/GPU Bench [FRU20] han comprobado las capacidades de cálculo y características actuales de estos dispositivos. Al mismo tiempo, han adquirido una gama cada vez mayor de funciones, como pantallas táctiles y receptores GPS, además de las cámaras y reproductores de música estándar, lo que los convierte en dispositivos completamente versátiles y útiles [PRI14] [CER12].

Por otra parte, además del aumento de la capacidad de hardware y los nuevos sensores incluidos, los dispositivos móviles cuentan con otro aspecto interesante que diversas instituciones han investigado: la gran masividad y penetración en el mercado mundial. Según los informes presentados en BankMyCell con los aportes de las compañías Wikipedia, Medium, ComputerWorld, entre otros, en 2021 se estima que existirán, aproximadamente, 4.8 billones de dispositivos móviles (3.5 billones de celulares inteligentes) [BAN21]. Este trabajo también informa que en el lapso de 2016 a 2020, los usuarios de los teléfonos inteligentes aumentaron en un 40% y se estima que para 2023, la cantidad de dispositivos móviles habrá incrementado a 7.33 billones [BAN21]. Esto puede significar una potencia computacional sin precedentes, especialmente con la amplia difusión de procesadores multinúcleo en teléfonos móviles. Este mercado, aún en crecimiento, ha creado una competencia extraordinaria para las industrias y ha dado lugar a una gran tasa de innovación. Otros estudios agregan que [ODE20][NGA19][ODD18] la tasa promedio a la que los respectivos propietarios reemplazan los teléfonos inteligentes es de, aproximadamente, entre 18 y 32 meses, cómo se puede ver en la figura 2.13.

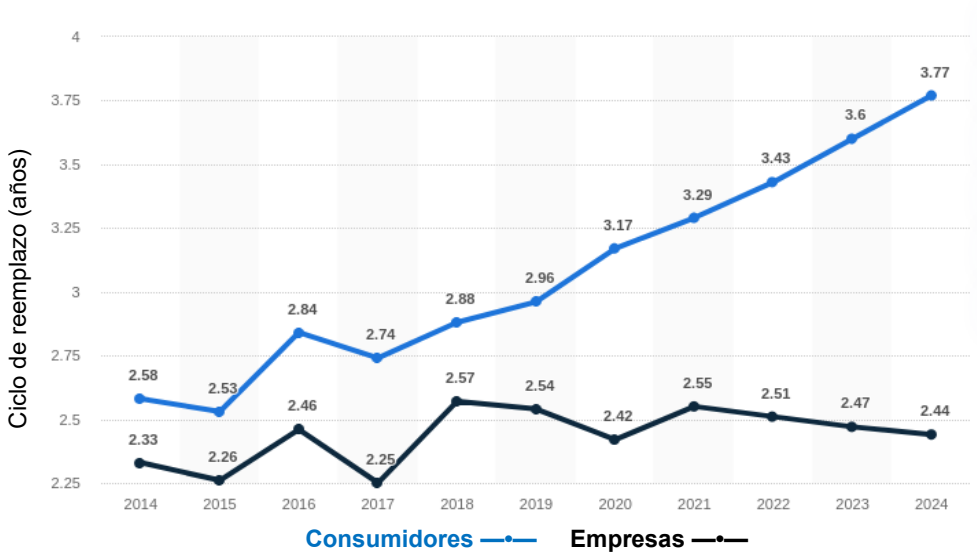

Figura 2.13 - Tendencia estimada de reemplazo de los celulares por parte de los usuarios (Estados Unidos) para el período 2014-2024. Fuente: Propia. Versión adaptada de [ODE20].

3. - Diseño e Implementación de la plataforma HPC

A diferencia de muchas de las arquitecturas HPC tradicionales, en las que la información y los recursos se encuentran disponibles en grandes servidores de datos y procesamiento, por ejemplo, en arquitecturas Cliente/Servidor; en este prototipo de software, se utiliza un modelo distribuido donde el trabajo se realiza de manera colaborativa entre los integrantes involucrados, siendo estos:

- Equipos en la Nube o locales, para la administración y gestión de la plataforma.
- Recursos de los usuarios finales reutilizados (móviles y equipos tradicionales), para el procesamiento de las tareas HPC

Gracias al trabajo en conjunto y a la reutilización de los nodos extremos, el sistema dispone de una capacidad masiva de procesamiento, distribuida y paralela, que permite reducir la carga en los nodos de administración centrales (Nube/locales) impactando directamente en los costos a la hora de realizar las tareas. Además, esta arquitectura permite definir, por las características que se detallan a lo largo de este capítulo, mecanismos de escalabilidad, fiabilidad y disponibilidad.

Este cambio de rol en los usuarios, cuyos dispositivos se convierten en nodos de procesamiento, supone una ventaja en relación a la mayoría de las herramientas analizadas; ya que descentraliza el procesamiento de las tareas, el acceso a los recursos, y la seguridad de la red. De esta manera las tareas de procesamiento no dependen de un reducido grupo de grandes equipos “centrales”, garantizando que, si uno o varios nodos dejan de funcionar inesperadamente, el sistema continuará desempeñándose correctamente.

3.1 - Objetivo

El objetivo de este capítulo es describir cómo se diseñó, desarrolló e implementó la plataforma HPC. Para ello, se presenta a continuación la arquitectura, componentes, aspectos tecnológicos y de seguridad que fueron incluidos para la construcción del sistema. También se describe de qué manera se configuran e interrelacionan los diversos subsistemas y componentes de la plataforma para:

- Garantizar los servicios de escalabilidad, flexibilidad y tolerancia a fallos.
- Respetar la premisa de ofrecer un bajo costo de funcionamiento y administración.

De esta manera, se sientan las bases para que la infraestructura pueda ser utilizada y/o adaptada a las necesidades de cualquier necesidad de procesamiento HPC.

3.2 - Descripción general

En esta sección se presenta la solución HPC desarrollada. La plataforma aprovecha las capacidades de hardware disponible y ocioso, para construir una plataforma que dé soporte a las necesidades HPC de los usuarios. Se construyó como una arquitectura distribuida, flexible, escalable y tolerante a fallos basado en una arquitectura híbrida que utiliza cómputo en la nube, local y móvil, y que se ajusta automáticamente en tamaño y capacidad, dependiendo de la carga y necesidades de trabajo.

La plataforma, descrita brevemente en la figura 3.1, permite cumplir con los siguientes objetivos:

- Gestión de tareas HPC:
    - Recepción, almacenamiento y gestión de las tareas HPC.
    - Fragmentación, distribución y posterior unificación de trabajos a realizar.
    - Ejecución de tareas de procesamiento HPC en nodos trabajadores.
- Gestión de dispositivos trabajadores (*workers*).
- Gestión de recursos de plataforma elástica y flexible.
    - Gestión de servicios de acceso (frontend).
    - Gestión de servicios de plataforma (backend).
    - Gestión de servicios de almacenamiento, colas y estadísticas.
    - Gestión de servicios de monitoreo.
    - Gestión de parámetros, escalado y flexibilidad.

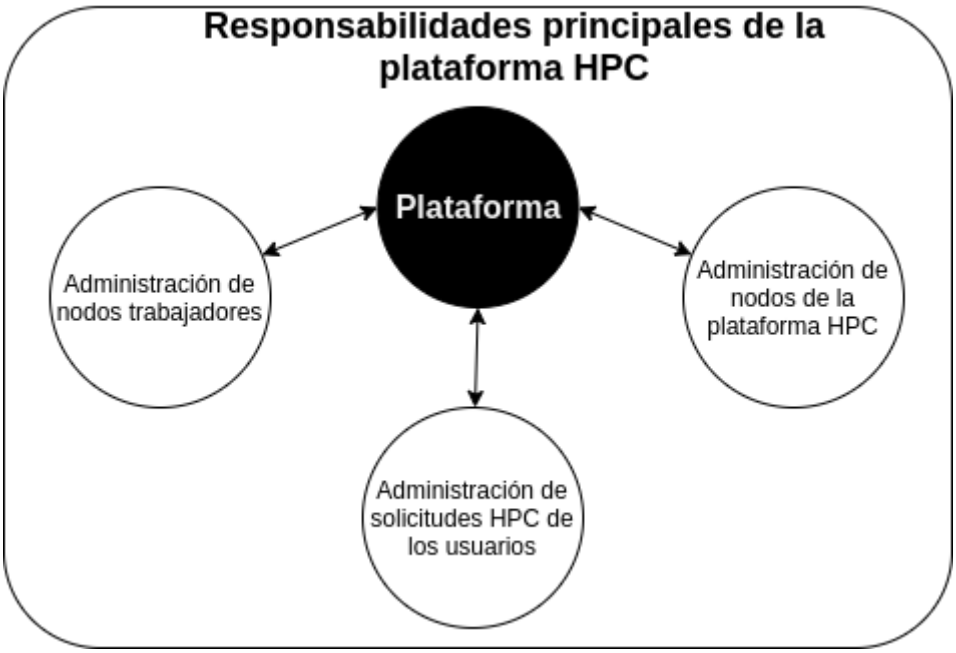

Figura 3.1 - Responsabilidades principales de la plataforma desarrollada. Fuente: Propia.

3.3 - Arquitectura de la plataforma HPC

A continuación, se detalla cómo está compuesta la arquitectura, tanto a nivel de diseño y descripción de componentes, así como también la implementación de esas responsabilidades en la plataforma.

3.3.1 - Diseño y características de la plataforma

La arquitectura HPC planteada se basa en un modelo híbrido compuesto por (a) usuarios interesados, (b) recursos de administración de la plataforma, y (c) recursos de procesamiento, como se muestra en la figura 3.2.

Figura 3.2 - Componentes básicos de la plataforma. Fuente: Propia.

Los usuarios interesados (a) son aquellas personas o servicios que requieren realizar una tarea de carácter HPC y que no cuentan con la capacidad o arquitectura suficiente para poder ejecutarlo en sus equipos o infraestructura. Por lo tanto, deciden acceder a los servicios ofrecidos por la plataforma.

Los recursos de administración de la plataforma (b) se utilizan para:

- Administrar las solicitudes HPC de los usuarios y convertirlas en trabajos acordes a las capacidades del producto.
- Administrar, gestionar, clasificar y controlar las tareas de procesamiento HPC.
- Dividir las tareas, almacenar los trabajos y unificar las actividades completadas.
- Almacenar los fragmentos y trabajos a realizar, así como la información estadística relacionada.
- Administrar de las solicitudes e intercambios con los dispositivos de procesamiento
- Unificar los trabajos procesados por los nodos de procesamiento.

Las responsabilidades antes descritas, fueron desarrolladas, implementadas y desplegadas sobre una arquitectura que permita ofrecer los servicios de forma confiable, redundante y escalable, buscando mantener costos de administración y funcionamiento reducidos.

Finalmente, los recursos de procesamiento (c), se utilizan para ejecutar de manera distribuida y naturalmente paralela los fragmentos de trabajos que las entidades comprendidas en el apartado (b) se encargan de entregarles. Para ello, cada dispositivo interesado dispondrá de una aplicación que puede abrir en su equipo cuando desee. De esa manera, automáticamente, se conectará al sistema HPC para realizar las actividades. En caso de querer dejar de participar, el usuario cerrará la aplicación y el sistema dejará de utilizar los recursos del dispositivo.

En la figura 3.3, se muestra la interrelación básica entre estos componentes.

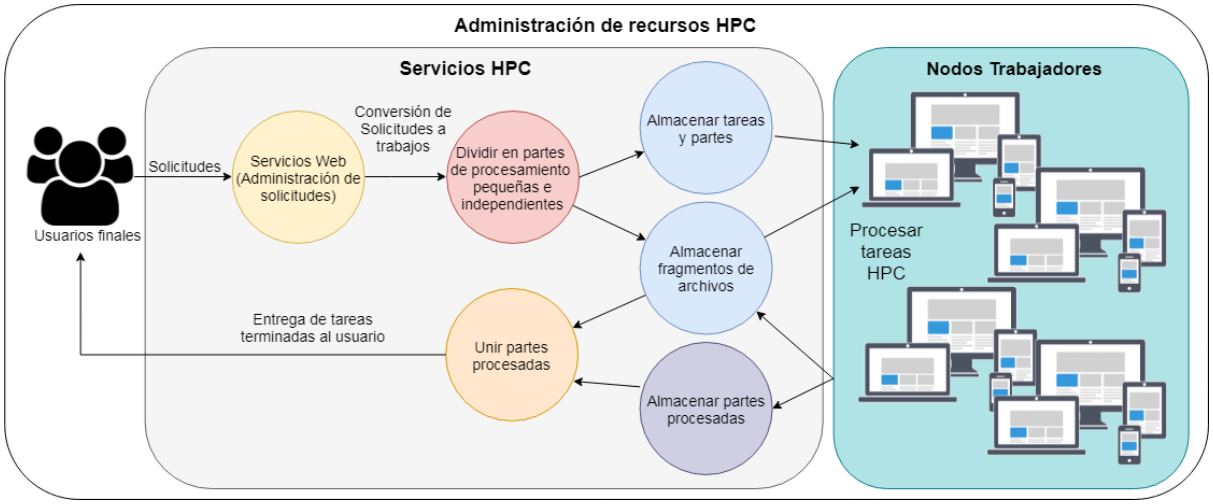

Figura 3.3 - Interrelación básica entre componentes de la plataforma HPC. Fuente: Propia.

A continuación, se detalla cada una de las funciones y procesos alcanzados por cada servicio.

Componentes básicos de la plataforma:

(a) Usuarios interesados:

Aquellos usuarios que requieran realizar tareas HPC podrán utilizar la plataforma a través de cualquier navegador web actual. Una vez dentro del sitio, podrán acceder a los tipos de tareas que estén disponibles, como servicios ofrecidos por la plataforma. El usuario deberá:

- Seleccionar la tarea de su interés.
- Subir los datos fuente de trabajo para ser procesados.
- Configurar los parámetros y ajustes correspondientes al servicio elegido.
- Confirmar la realización de la tarea.

A partir de ese momento el interesado podrá visualizar el progreso de su actividad en la plataforma; y será notificado cuando la misma esté finalizada.

En base a las características del sistema, este proceso podrá repetirse indefinidamente por el o los usuarios interesados, de manera transparente.

(b) Recursos de administración de la plataforma:

Cómo se indicó anteriormente, los recursos de administración de la plataforma conforman el “core” de la aplicación; ya que es el encargado de manejar los servicios, procesos y mecanismos para completar las tareas HPC de inicio a fin. A continuación, se describen los componentes esenciales para poder brindar una solución que cumpla con las necesidades y metas de este proyecto.

- Punto de acceso al sistema: La estructura diseñada, evita disponer de puntos únicos de acceso [DIO20] [AND20] [SAR15]. La plataforma se encarga siempre de mantener cada servicio replicado, como se puede ver en la figura 3.4, en al menos 3 nodos para mantener el sistema disponible. Además de las réplicas, el sistema ofrece servicios de balanceo de carga y tráfico en clúster para poder garantizar un punto de acceso redundante (IP o nombre de DNS dependiendo del entorno y las configuraciones); y al mismo tiempo administrar las solicitudes de manera ordenada derivando las necesidades a los servicios escalables y redundantes de la plataforma.

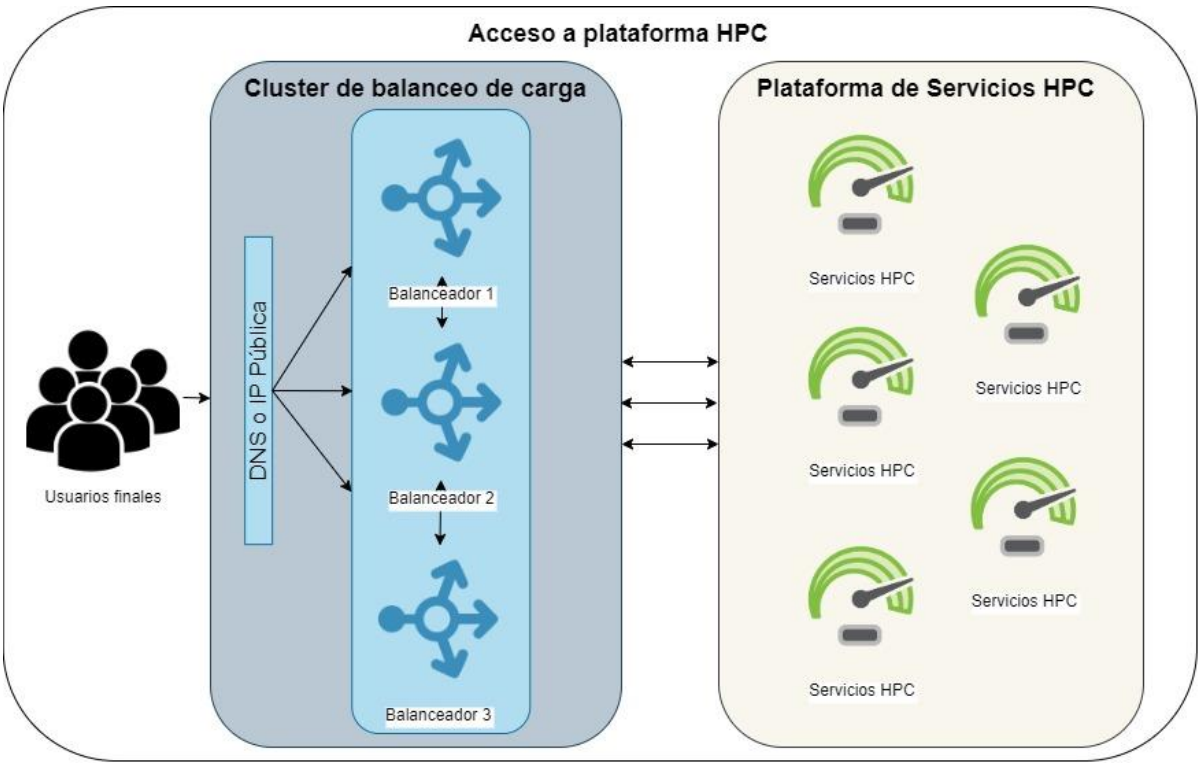

Figura 3.4 - Servicios de acceso y funcionamiento redundante ofrecido por la plataforma. Fuente: Propia.

- Servicios escalables y redundantes: Detrás de la capa de balanceo, se encuentra un conjunto de servicios web, sistema de archivos, colas y base de datos replicados (3 o más instancias), los cuales son los encargados de mantener el flujo de trabajo de la plataforma HPC. Los componentes y su interacción se describen en la figura 3.5.

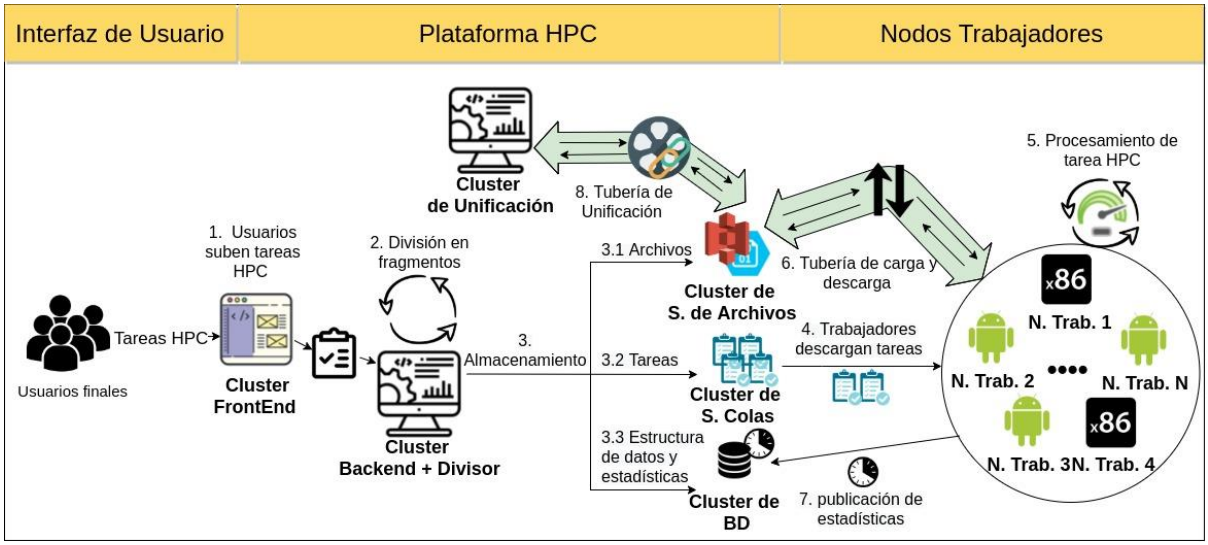

Figura 3.5 - Detalle de componentes, flujos e interacción de la Plataforma HPC. Fuente: Propia.

A continuación, se detallan los módulos y funciones de cada uno:

- Servicios frontend: Estos servicios interconectan a los usuarios con la plataforma y sus servicios. Es decir, los usuarios entrarán al punto de acceso del sistema (nombre DNS o IP) y serán redirigidos al componente frontend. Una vez que ejecuten una acción, este servicio se comunicará con la red del backend para dar comienzo a la resolución de la tarea HPC. Las responsabilidades esenciales son las siguientes:
  - Mostrar un panel para la selección del servicio HPC a ejecutar.
  - Mostrar un área para subir los archivos fuente, para las tareas de procesamiento.
  - Mostrar un área estructurada de tipo clave-valor para la asignación de parámetros y configuraciones de ejecución de la tarea HPC.
  - Mostrar un botón para confirmar la ejecución de la tarea HPC.
  - Mostrar un panel actualizable para ver el progreso de la tarea en cuestión.
  - Mostrar un área para visualizar el resultado de la ejecución de la tarea.
- Servicios Backend: Estos servicios son los encargados de manejar todo el proceso de recepción, división, fragmentación, gestión del procesamiento y unificación de tareas. Con el objeto de cumplir con las metas del sistema; y para trabajar de manera prolija y ordenada, esta funcionalidad se divide en

varios servicios más pequeños que dan respuesta a las necesidades de la plataforma. A continuación, se detallan los componentes implicados y sus características:

1) Backend Web + Split: A partir de las peticiones de los usuarios finales, un nodo de administración atiende su solicitud y se encarga de derivar la tarea al fragmentador de trabajos. También es el responsable del almacenamiento de los archivos dentro de la plataforma y, cuando la tarea está completa, es el punto de acceso para el usuario final. Sus actividades más importantes son:

- Recibir la tarea definida por el usuario a través del Frontend.
- Guardar en el sistema de archivos el/los recurso/s enviados por el usuario.
- Generar y registrar la estructura general de trabajo que debe ser procesado.
- Originar la división de tareas, según las políticas definidas para cada caso de procesamiento distribuido; y registrar:
    - Fragmentos de archivo.
    - Parámetros y configuraciones.
    - Estructuras del trabajo.
- Monitorear las partes procesadas y notificar al Joiner cuando una tarea está completa.
- Monitorear cuando el trabajo fue terminado por el Joiner, para finalizarlo y notificar al Frontend.
- Guardar en el sistema de datos el archivo final, y eliminar los archivos ya utilizados (original, fragmentos sin procesar y procesados).

2) Backend Joiner: Una vez que todos los trabajos de una tarea en particular son resueltos por los nodos trabajadores, el servicio Joiner es el encargado de tomar la totalidad de los fragmentos y unificarlos. Más tarde, el usuario, accederá a esta información completando el ciclo de trabajo HPC. Las funciones principales son las siguientes:

- Buscar continuamente tareas para unificar.

        - Combinar los fragmentos procesados y almacenar el archivo resultante.
        - Notificar al Backend Web que se ha completado la tarea.

    - Servicio de Sistema de Archivos: Este servicio es utilizado para guardar los archivos que envían los usuarios dentro de la plataforma en cuestión. Además, se almacenan los fragmentos a procesar, los procesados y el resultado final. Presenta una configuración en clúster y redundancia de datos, entre los distintos nodos que lo componen, con objeto de garantizar que la información esté almacenada de manera confiable.

      Este servicio es ofrecido de manera de poder brindar un endpoint (url) que pueda ser consumido, tanto dentro de la plataforma (para los servicios que la componen) como fuera de la misma (por los nodos trabajadores). A su vez, el Backend Web es responsable de la limpieza de archivos obsoletos, evitando el consumo innecesario de espacio y afectar a los costos.

    - Servicios de colas asincrónicas: Es utilizado para registrar las tareas a realizar de manera segura, asincrónica y persistente. El uso de un middleware de colas permite desligar a la plataforma de realizar los procesos HPC de manera sincrónica, ante la solicitud de los usuarios. Además, se configura de manera replicada (en clúster), lo que permite garantizar el funcionamiento y recuperación ante errores en los mensajes o la infraestructura. En particular el motor de colas registra:
        - Servicio HPC a ejecutar o unificar.
        - Parámetros y configuraciones a ser tomadas por el nodo trabajador.
        - Dirección (endpoint) del recurso almacenado.

    - Servicios de base de datos: El SGBD se incluye en este proyecto con objeto de dar soporte al registro de información estadística de las tareas. En particular se registran:
        - Cantidad de partes totales a procesar y procesadas (en un determinado momento) de una misma tarea.
        - Información estadística de procesamiento general (tiempos de ejecución).

        - Información estadística de procesamiento de fragmentos (nodo de procesamiento, tiempos de ejecución por fragmento de tarea, etc.).

- Auto administración de servicios: El diseño de la plataforma propone que sus componentes tienen la capacidad de administrar su propia estructura flexible. Esto significa que la red de administración conoce el estado de cada nodo, y de la red en general, de manera de decidir si es necesario aumentar o reducir la cantidad de nodos y servicios que actualmente están atendiendo a los clientes, tareas y procesos. Cómo se describió en el apartado de servicios de escalado y redundancia, se requiere un mínimo de 3 réplicas por servicio para garantizar disponibilidad. Sin embargo, eso no es suficiente para definir una plataforma que pueda crecer y flexibilizarse ante una demanda concurrente y creciente de tareas (SLOs - objetivos de nivel de servicio [DIN19][SAU05]); y reducir, hasta el punto mínimo (cuando la plataforma no disponga de carga de procesamiento), qué es la otra premisa del proyecto.

  Para afrontar este requerimiento el producto de software tiene en cuenta, para todos sus elementos, una definición de parámetros, métricas y actividades de monitoreo contínuo que permiten tomar acciones para escalar (hacia arriba y abajo) dependiendo de la carga del sistema. Para ello, la plataforma considera el uso de:

  - (a) porcentaje de uso del procesador y memoria.
  - (b) Tiempo de respuesta promedio.
  - (c) Cantidad de peticiones encoladas.

  En caso de disponer de valores de (a) superiores al 75 % de manera sostenida, en promedio, se iniciará un proceso de escalado y creación de un nuevo nodo para afrontar la carga del sistema. Por el contrario, si se reduce por debajo del 40%, en promedio, por un período de cinco minutos y existen más de tres réplicas del servicio, se procederá a dar de baja un nodo de este servicio. Estos valores fueron tomados en cuenta en base a las recomendaciones y experimentaciones de diversos trabajos y artículos para el orquestador Kubernetes [RED21] [GOO21] [NGU20] [HAN20] [TUC19].

(c) Recursos de procesamiento:

Propone que los dispositivos en desuso puedan aportar su poder de cálculo en los momentos que se encuentran ociosos. Si bien la plataforma está orientada, principalmente, a integrar los dispositivos móviles basados en Android, también dispone de una versión para equipos

de escritorio, lo que busca aumentar aún más la cantidad de dispositivos participantes y la capacidad de procesamiento de esta arquitectura HPC.

Restricciones de ejecución:

Para que un dispositivo móvil sea considerado disponible deben darse un conjunto de condiciones:

- Debe contar con la aplicación móvil abierta y en ejecución.
- Debe estar conectado a la red eléctrica y disponer de su carga de batería completa (100%).
- Estar conectado a una red WiFi.

Estos tres puntos buscan garantizar que, por un lado:

- No se utilice si el usuario lo está utilizando.
- No le consuma la batería al usuario y disminuya la vida útil del equipo.
- No produzca un gasto al usuario.

Y por otro, que, al cumplir con estas restricciones, se promueve la inclusión de equipos que están en reposo y ociosos. De esta manera, se evita que los nodos de procesamiento estén entrando y saliendo constantemente a la red. Esto favorece a disminuir las tareas de gestión y reasignación de trabajos.

Para el caso de los dispositivos tradicionales de escritorio (x86) las condiciones son más flexibles:

- Disponer la aplicación abierta y en ejecución.
- Si el usuario desea, configurar los límites de uso del equipo (% de CPU y memoria) y los horarios de ejecución permitidos.

Modo de funcionamiento:

Una vez abierta y configurada la aplicación de los nodos trabajadores, se encarga de descargar, procesar y subir los trabajos de procesamiento a la plataforma HPC. Independientemente de la capacidad y tipo de dispositivo, el nodo trabajador realiza continua e iterativamente los siguientes pasos:

- Conexión y sincronización con el sistema de colas.
- Continuamente, hasta que el usuario cierre la aplicación:
    - Descarga de tareas y parámetros del sistema de colas.
    - Descarga del recurso a procesar desde el endpoint (url) del sistema de archivos.
    - Procesamiento de la tarea HPC en base a los parámetros definidos.
    - Subida del recurso procesado al endpoint (url) destino del sistema de archivos.
    - Subida y actualización de la tarea al sistema de colas.
    - Subida y actualización de la información estadística en el sistema de base datos.

En caso de que el dispositivo sufra alguna falla, como puede ser:

- Desconexión de la red o de alguna de las condiciones previamente establecidas.
- Problemas al descargar o subir:
    - Los mensajes de la cola.
    - La información estadística.
    - Los archivos de datos.
- Problemas al realizar el procesamiento de la tarea HPC.

Se descarta la tarea del nodo trabajador y automáticamente los servicios del sistema de colas se encargan de re-encolar el trabajo, en el apartado correspondiente, para que otro nodo pueda tomar el requerimiento HPC.

3.3.2 - Desarrollo e implementación de la plataforma

Sobre la base definida en la etapa de diseño, la arquitectura desarrollada y desplegada se basó en los siguientes cuatro apartados esenciales:

a) Microservicios **Dockerizados** para la administración y gestión de la plataforma.

b) Servicios de colas (tareas) y base de datos (estadísticas) Dockerizadas.

c) Orquestación, escalado, replicación y automatización de contenedores a través de Kubernetes.

d) Trabajadores móviles Android ARM y x86 Dockerizados.

En la figura 3.6, se presenta un esquema de los componentes y su interrelación.

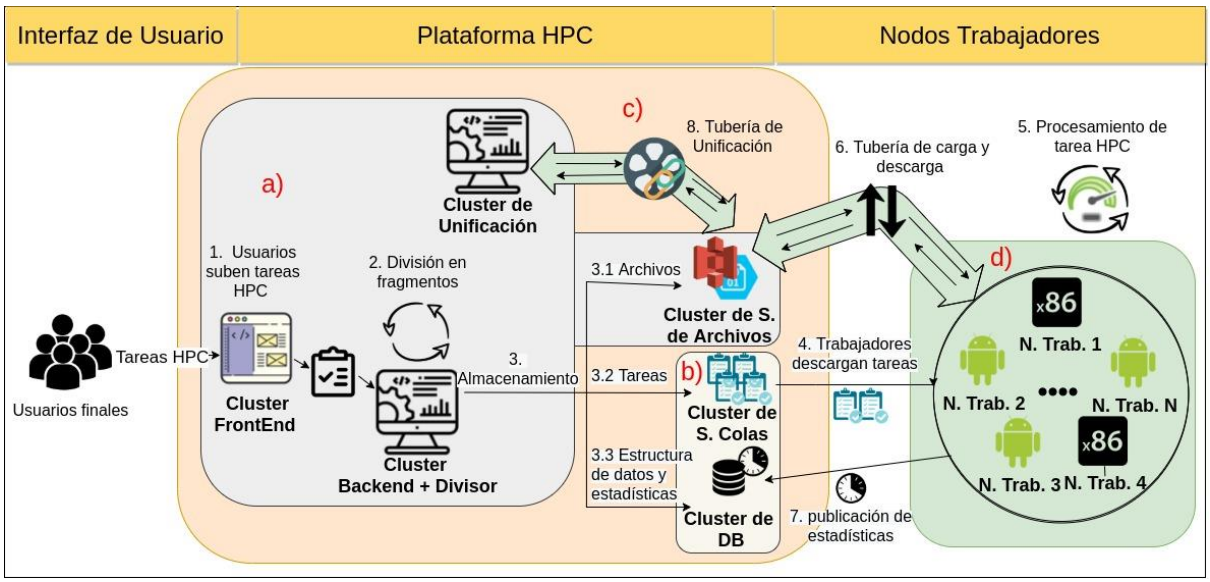

Figura 3.6. Arquitectura y componentes que forman parte de la arquitectura desarrollada. Fuente: Propia.

A continuación, se detalla cada una de las partes, sus responsabilidades, relación y alcance.

a) Microservicios Dockerizados para la administración y gestión de la plataforma

Para el desarrollo y despliegue del sistema se adoptaron herramientas, tecnologías y arquitecturas que permitieran aprovechar al máximo los recursos informáticos disponibles (locales, Nube y móviles) para realizar las tareas HPC.

Las funciones y responsabilidades de la plataforma HPC se dividieron siguiendo una arquitectura basada en microservicios [VIG18]. De esta manera fue posible:

- Descomponer las responsabilidades del sistema HPC en unidades de servicio desplegables independientes [VIG18].
- Definir un contexto limitado, basado en un dominio acotado, específico para cada servicio, produciendo funciones pequeñas, autónomas, resistentes, confiables y fáciles de intercambiar, replicar, y resguardar [VIG18].

- Integrar ciclos de Iteración rápida, flexibilidad a la hora de incluir nuevos cambios, actualizaciones y/o dependencias [ERI17].
- Integrar procesos de automatización, pruebas e implementación continuas [NEW14].
- Integrar y comunicar los servicios a través de interfaces REST API y mensajes sobre el protocolo HTTP [GIE15].

Cómo se muestra en la figura 3.6, las responsabilidades de la plataforma se fragmentaron en los siguientes microservicios HTTP: (a) FrontEnd MsA; (b) Backend Web+Split MsA; (c) Backend Joiner+Unifier MsA y (d) File Server MsA. A continuación, se detallan las funciones y características de cada uno.

(a) FrontEnd MsA:

Cómo se describió en el apartado de diseño, el servicio de frontend permite a los clientes subir los archivos fuente, seleccionar y definir parámetros para la tarea HPC; así como también ver el progreso y el resultado de los trabajos realizados.

El desarrollo de este microservicio se realizó a través del framework Sails.JS v1.0 (Express + ejs) con todos los paquetes npm necesarios integrados en la carpeta correspondiente para poder contar con el despliegue de este microservicio.

(b) Backend Web+Split MsA

El servicio Backend Web + Split se desarrolló utilizando el framework Spring Boot (2.3.4. RELEASE) para generar una API REST, de modo que los clientes, servicios y nodos puedan interactuar a través de mensajes JSON vía HTTP [ROD16][CAN14]. Por lo tanto, el Backend, define un servicio @RestController el cual permite intercambiar mensajes HTTP GET y POST (RequestMethod.GET, RequestMethod.POST).

Con esta funcionalidad, se convierten las solicitudes HTTP de los clientes en tareas de procesamiento distribuido. Una vez recibido el/los archivo/s fuente/s y los parámetros; se fragmentan los recursos fuentes en secciones más pequeñas e independientes entre sí, basado en los parámetros definidos por parte del sistema y del usuario. Esto permite generar tareas HPC independientes y más reducidas. Por cada tarea, el Servidor Web genera un Hilo de administración independiente (Split Thread), derivando la responsabilidad a dicha instancia. Este hilo es una clase Java que llama a un proceso (pipe) que se ejecuta en la consola. La función ejecutada (runBash), se ejemplifica a continuación:

```
....
public String runBash (String command){
        String totalLines = "";
        try {
            //[PASO 0] - Se define el runner (/bin/bash linux ; /bin/sh android) y se concatena el
comando a ser ejecutado
            String[] cmdArray = {"/bin/bash", "-c", command};
            //[PASO 1] - Obtener un Processo Runtime
            Process runner = Runtime.getRuntime().exec(cmdArray);
            //[PASO 2] - Obtener los canales de entrada/salida del proceso
            BufferedReader stdOut = new BufferedReader(new
InputStreamReader(runner.getInputStream()));
            BufferedReader stdErr = new BufferedReader(new
InputStreamReader(runner.getErrorStream()));
            String line = "";
            //[PASO 3] - Leer y esperar hasta que el proceso haya finalizado
            while ((line = stdOut.readLine()) != null) {
                totalLines=line;
            }
        }
....
```

Este segmento de código permite entonces, recibir un comando a ejecutar y generar un Proceso (tubería) en segundo plano manejada por Java y que permite comunicarse, escribir y leer en la consola del sistema operativo.

Una vez que los recursos fuentes son divididos, los segmentos se guardan en un almacenamiento HTTP de bloques en la Nube de bajo costo, cómo Azure Blob Storage o Amazon S3 [ZOU18] [BOC14], o en un sistema de archivos local, dependiendo el ambiente de ejecución, tecnologías e infraestructura disponible. Para la carga y descarga de los archivos almacenados en los sistemas de archivos (endpoints) se utiliza la herramienta Curl [STE18], que se encuentra integrada al servicio.

En el caso de la nube, para cada proveedor, se deben definir las configuraciones y credenciales correspondientes para lograr acceder a los recursos de manera segura y protegida, adaptando mínimamente el código para que pueda utilizar los servicios de almacenamiento [SAA19]. A modo de ejemplo, se presentan las configuraciones utilizadas para los casos de Amazon S3 y Azure Blob Storage (Containers)

Conexión y almacenamiento en Amazon S3:

En el caso de Amazon es necesario disponer del Bucket S3 creado y con los permisos necesarios para lograr leer y escribir. Para su creación puede utilizarse la interfaz Web o la herramienta de consola AWS CLI [AWS20], cómo se muestra en el siguiente contexto:

```
$ aws s3api create-bucket --bucket my-bucket --region us-east-1
```

Una vez creado, se debe aplicar una política de seguridad [AWS20b] que permita las acciones de lectura/escritura deseadas. A continuación, se muestra la configuración utilizada en este proyecto para el Bucket “s3-dp-in”:

```
{
   "Version": "2008-10-17",
   "Statement": [{
     "Sid": "AllowAnonMSG",
     "Effect": "Allow",
     "Principal": { "AWS": "*" },
     "Action":["s3:PutObject","s3:PutObjectAcl"],
     "Resource": ["arn:aws:s3:::s3-dp-in/*" ]
   }]
 }
```

Esta política permite que se publiquen y lean los contenidos del destino. Además de las políticas de acceso al Bucket (Policy), es necesario contar con las credenciales de acceso, las cuales deben ser incluidas de manera segura y cifrada, para poder conformar las peticiones CURL y poder subir/bajar los archivos a los destinos remotos. A continuación, se muestra un fragmento de código donde se presentan los pasos definidos para este proveedor:

```
if (this.context.startsWith("aws")) {
                                    //[PASO 0] - Nombre del Bucket S3
                                    String outS3Bucket      = "s3-dp-out";
                                    //[PASO 1] - Nombre en AWS del Bucket S3
                                    String outAws           = outS3Bucket + ".s3.us-east-
1.amazonaws.com";
                                    //[PASO 2] - Defino las credenciales (Securizadas)
                                    String s3AccessKey   = "KeySecured";
                                    String s3SecretKey   = "SecretSecured";
                                    //[PASO 3] - Defino la estructura HTTPS del Bucket teniendo
en cuenta el nombre del archivo
                                    String target       =
msgRearmed.getName()+"_"+msgRearmed.getPart()+".ts";

                                    //[PASO 4] - Defino las configuraciones para utilizar CURL y
vincularlo al Bucket
                                    SimpleDateFormat sdf = new
SimpleDateFormat("YYYYMMdd'T'HHmmss'Z'");
                                    sdf.setTimeZone(TimeZone.getTimeZone("GMT"));
                                    String dateFormatted = sdf.format(new Date());
                                    String relativePath  = "/" + outS3Bucket + "/" + target;
                                    String contType    = "application/octet-stream";
                                    String stringToSign  = "PUT\n\n" + contType + "\n" +
dateFormatted + "\n" + relativePath;
                                    String signature     = hmacSha1(stringToSign, s3SecretKey);
                                    //[PASO 5] - Armo el CURL en base a los parámetros,
```

```
credenciales y binarios
                                curl = "curl -X PUT --data-binary @- -H \"Host: " + outAws +
"\" -H \"Date: " + dateFormatted + "\" -H \"Content-Type: " + contType + "\" -H
\"Authorization: AWS " + s3AccessKey + ":" + signature + "\" \"https://" + outAws + "/" +
target + "\"\n";
                                //[PASO 6] - Llamo al ejecutador de herramientas pasando
los parámetros del curl
                                this.runBash(curl);

}
```

En este ejemplo, se chequea que el proveedor seleccionado sea AWS. Una vez verificado, se define cual es el Bucket S3 destino, se construye la url HTTPS del mismo según las buenas prácticas de AWS en las que se define las credenciales de acceso (Access y Secret Keys) y se genera un mensaje codificado [AWS20b]. Con estos datos, se prepara la petición CURL vinculada con el archivo a subir la cual, finalmente, será enviada al ejecutor de comandos de consola (runBash).

Conexión y almacenamiento con Azure Blob Storage (Container):

Al igual que el caso anterior, en el proveedor Azure [SHI20] también es necesario disponer de una cuenta de almacenamiento Blob Storage y haber creado un Container (contenedor) [SID19]. Más tarde, al igual que en Amazon, se deben definir los permisos de lectura/escritura. En este caso, se creó un Container con acceso público utilizando el Azure Cli de Azure:

```
$ az storage container create -n dp-ms --public-access blob
```

Una vez creado, se requiere también disponer las credenciales de acceso, las cuales deberán ser incluidas de manera segura y cifrada, para poder conformar las peticiones CURL que permitan subir/bajar los archivos con este proveedor. A continuación, se define un fragmento de código donde se muestran los pasos definidos para operar con Azure Blob Storage a través de un token SAS (Shared Access Signature) [MYE20][MOI19].

```
....
if (this.context.startsWith("az")) {
                        //[PÂSO 0] - Si es AZ, definir el nombre del Azure Storage Account
                        String AzStorageAccount ="dp-ms";
                        //[PÂSO 1] - Escribir la SecretKey
                        String keyStr = "SecretKey";
                        //[PÂSO 2] - Generar el SAS Token (Función explicada más abajo)
                        String sasToken = this.GetFileSAS(AzStorageAccount, keyStr);
                        //[PÂSO 3] - Definición de los Buckets
                        String AzSplittedContainer = "s3-bcra-split";
                        String AzCompressedContainer = "s3-bcra-compressed";
                        //[PÂSO 4] - Construcción del CURL con los buckets y los nombres
```

```
de los archivos
                        source = "https://" + AzStorageAccount + ".blob.core.windows.net/"
+ AzSplittedContainer + "/" +
msgRearmed.getOriginalName().substring(5,msgRearmed.getOriginalName().length());
                        String target =
msgRearmed.getName()+"_"+msgRearmed.getPart()+".ts";
                        curl = "curl -X PUT --data-binary @- -H \"x-ms-date: $(date -u)\" -H
\"x-ms-blob-type: BlockBlob\"
\"https://"+AzStorageAccount+".blob.core.windows.net/"+AzCompressedContainer+"/"+tar
get+sasToken+"\"";
                        //[PASO 5] - Ĺlamada a la función
                        this.runBash (curl);
}
....
....
public String GetFileSAS(String accountName , String key ){
                //[STEP 0] - Tomo los parámetros definidos por el usuario y construyo la
conexión al storage de Azure
                String storageConnectionString =
"DefaultEndpointsProtocol=https;AccountName="+accountName+";AccountKey="+key+";
EndpointSuffix=core.windows.net";

                String sasToken="";
                try {
                        //[STEP 1] - Defno una serie conexiones y pasos para obtener los
permisos (temporales) para escrbir en Azure
                        CloudStorageAccount storageAccount =
CloudStorageAccount.parse(storageConnectionString);
                        SharedAccessAccountPolicy sharedAccessAccountPolicy = new
SharedAccessAccountPolicy();
                        sharedAccessAccountPolicy.setPermissionsFromString("racwdlup");
                        //[STEP 1] - Defino un tiempo de vida en Milisegundos
                        sharedAccessAccountPolicy.setSharedAccessExpiryTime(new
Date(new Date().getTime()+ 8640000));
                        sharedAccessAccountPolicy.setResourceTypeFromString("sco");
                        sharedAccessAccountPolicy.setServiceFromString("bfqt");
                        //[STEP 2] - Genero el token
                        sasToken = "?" +
storageAccount.generateSharedAccessSignature(sharedAccessAccountPolicy);
                }catch (Exception e){
                        e.getMessage();
                }
                //[STEP 3] - Devuelvo el Token
                return sasToken;
        }
....
```

En este ejemplo, se chequea que el proveedor seleccionado sea Azure. Una vez comprobado, se define cual es el Container destino, se construye la url HTTPS del mismo, se definen las credenciales de acceso (Secret Keys) y se genera el Shared Access Signature (SAS) con un tiempo de vida limitado para brindar los permisos de acceso hacia el contenedor blob de Azure (GetFileSAS) y se genera un mensaje codificado según las recomendaciones del

proveedor. Con la estructura generada, se prepara la petición CURL que finalmente será enviada al ejecutor de comandos de consola.

Conexión y almacenamiento con File Server local (Microservicio propio):

En el caso que no se disponga de un proveedor de almacenamiento de terceros, se cuenta con la opción del microservicio desarrollado, el cual puede ser utilizado en entornos de Kubernetes Locales.

Cómo este microservicio se desarrolló de manera simple, no incluye el manejo de credenciales de seguridad, tiempos de vida y cifrado de conexiones. A continuación, se muestra un breve ejemplo para mostrar cómo se realiza una petición contra este servicio.

```
String myCurl = "curl -X PUT -F \"file=@/tmp/" + file + "\" -H
\"Content-Type: multipart/form-data\" \"http://" + this.microserviceUrl
+ ":8080/uploadFile?folder=s3-bcra-split\"";
this.runBash(saveCurl);
```

Donde los únicos pasos necesarios son contar con el microservicio levantado (función cubierta por Kubernetes), con el archivo a subir/bajar (file), la dirección del microservicio (microserviceUrl) y la carpeta (folder) involucrada.

Por su parte, las tareas a realizar y sus parámetros de ejecución se guardan en el sistema de colas persistente. Cómo se detalla en el apartado correspondiente, su uso permite que las actividades se procesen de manera asincrónica por parte de los nodos trabajadores. Finalmente, la estructura de la tarea y su información estadística se registra en la base de datos (cantidad de partes, partes completas y tiempos de ejecución). Para esto, el Web Server se conecta al sistema de colas, mediante la Factoría de Conexión (Connection Factory) y al SGBD mediante el JDBC correspondiente.

(c) Backend Joiner+Unifier MsA

Los servicios del Joiner y Unifier MsA se encuentran desarrollados en Java con JDK 14 y no presentan API REST ya que no ofrecen servicios al usuario final y su responsabilidad se limita a consumir servicios de terceros y procesar trabajos.

El microservicio Joiner consulta continuamente la base de datos para validar cuando un trabajo de compresión ha finalizado, es decir que la cantidad de partes procesadas es igual a la cantidad de partes totales del trabajo. Si encuentra un caso, genera un registro de tarea en el sistema de colas para indicar que se encuentra disponible una tarea de unificación que debe ser ejecutada. Es ahí donde los microservicios Unifier levantan los trabajos a realizar y aplican el proceso de unificación. Una vez completado, se guarda el archivo final en el sistema

de archivos correspondiente, registra la información estadística de la tarea y elimina el registro de unificación del sistema de colas.

(d) File Server MsA

Para el caso que no se utilice un sistema de archivos HTTP externo, como el caso de Amazon S3, Azure blob Storage o Google Cloud Storage, se desarrolló un servidor de archivos NFS vía API REST HTTP [NET20] [MER19]. Su responsabilidad es recibir las peticiones curl (carga y descarga) y persistir los archivos de forma segura y redundante.

Para ello, se integró el framework de almacenamiento Rook.io (versión 1.4) [GAR20][RUB18], que proporciona una plataforma y soporte para definir una solución de almacenamiento que se integra de forma nativa en un entorno de Kubernetes. Se construyó un microservicio Web basado en Java Spring Boot el cual se presenta al usuario a través de una API REST. De la gama de servicios de sistemas de archivos distribuidos por Rook, se configuró NFS mediante la definición de recursos personalizados (NFS Server CRD).

Cabe destacar que cada uno de estos microservicios fueron Dockerizados (FrontEnd MsA; Backend Web+Split MsA; Backend Joiner+Unifier MsA; File Server MsA) con objeto de lograr las siguientes metas:

- Incluir todos los recursos necesarios para que los servicios sean portables, ligeros y puedan ejecutarse de manera encapsulada dentro de una sola imagen, completamente independiente del servidor anfitrión [RAD17].
- Construir servicios que optimicen los recursos de hardware de la infraestructura subyacente, sin utilizar una capa de abstracción adicional entre el kernel y la aplicación y evitando virtualizar los dispositivos (más eficiente en un orden de magnitud respecto de las máquinas virtuales).
- Aislar cada microservicio del entorno de ejecución del host (procesos, jerarquía de archivos y stack de red).
- Contar con servicios que se pueden iniciar y detener en una fracción de segundo, gracias a la escasa o nula sobrecarga sobre el sistema operativo host [TUR19].
- Hacer posible y transparente tanto la simulación como el despliegue de los servicios en producción.

En el caso de la plataforma, la dockerización de los microservicios se realizó aplicando los siguientes pasos:

1. **Parametrizar la ejecución de los microservicios.**

Cada microservicio desarrollado, debe ser configurado para poder leer los parámetros de funcionamiento en tiempo de ejecución; poder adaptarse al ambiente donde se desarrolla y saber cómo interconectarse con los otros microservicios. Esto puede darse recibiendo los mismos por consola, por un servicio de descubrimiento (por ejemplo, Consul [MSV19]) o leyendo dicha información de un recurso externo.

En este caso, se optó por leer la configuración de un servicio externo. Para ello, al iniciarse cada microservicio, se conecta a un repositorio en Github (que se actualiza cuando es necesario) y lee los parámetros de ejecución. Por ejemplo:

```
String urlGit =
"https://raw.githubusercontent.com/dpetrocelli/LinuxRepo/master/configFi
le";

this.downloadConfigurationFile(urlGit, context);
```

2. **Exportar el microservicio cómo un archivo ejecutable.**

Cada servicio Java se exporta con todas sus correspondientes propiedades, librerías y dependencias (definidos en el archivo pom.xml del proyecto) en un único archivo ejecutable de salida en formato ".jar", utilizando Maven. Por ejemplo, el servicio Backend Web + Split se exporta bajo el nombre dev-be-split.jar

- Archivo pom.xml (definiciones del ".jar" + librerías + configuraciones)

```
...
<modelVersion>4.0.0</modelVersion>
<groupId>thesis</groupId>
<artifactId>v1</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>REST Web Server + Split </name>
<description> Spring Boot APP - THESIS 2020</description>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.0.5.RELEASE</version>
<relativePath/>
<!--  lookup parent from repository  -->
....
```

```
....
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
<dependency>
<groupId>com.rabbitmq</groupId>
<artifactId>amqp-client</artifactId>
<version>5.8.0</version>
</dependency>

....
</dependencies>
....
```

- Comando para exportar a archivo empaquetado “.jar” (Maven)

```
mvn package -Dmaven.test.skip=true
```

3. Definir el archivo Dockerfile para la construcción de la imagen Docker

Una vez que se dispone del archivo ejecutable, es necesario llevarlo a un formato dockerizado para que sea compatible con cualquier arquitectura que utilice este motor de contenedores. Entonces, se requiere definir un Dockerfile que permita sentar la base para realizar la imagen del servicio

```
FROM adoptopenjdk/openjdk14
COPY dev-be-split.jar /dev-be-split.jar
EXPOSE 8081
CMD ["java", "-jar", "/dev-be-split.jar"]
```

Donde:

- FROM: Es la imagen base que se utiliza para la construcción del contenedor. En este caso, una versión de Java JDK que permitirá correr el microservicio.
- COPY: Comando encargado de copiar el archivo “.jar” creado anteriormente dentro de la “imagen base”.

- EXPOSE: Comando utilizado para definir qué puerto de escucha tendrá el contenedor una vez instanciado y puesto en ejecución.
- CMD: Comando utilizado para definir qué servicio correr una vez que se instancie el contenedor.

4. **Crear la imagen Docker y publicarla en Registro (Docker Hub)**

Una vez exportado el archivo empaquetado ".jar" del microservicio, y contando con el Dockerfile, se procede a construir la imagen Dockerizada del servicio. Para ello se utiliza el siguiente comando:

```
docker build -t dpetrocelli/dev-be-split:latest .
```

Seguidamente, se publica la última versión generada identificada por el tag: latest, al registro de contenidos Docker Hub. En este caso, a través del siguiente comando:

```
docker login
docker push dpetrocelli/dev-be-split:latest
```

Para realizar estos cuatro pasos de forma automatizada, se define un único archivo bash (deploy-automation.sh) que contiene todos los pasos antes descritos.

5. **Automatizar el empaquetado, construcción y publicación**

```
mvn package -Dmaven.test.skip=true
cp ../target/dev-be-split.jar dev-be-split.jar
docker build -t dpetrocelli/dev-be-split:latest .
docker login
docker push dpetrocelli/dev-be-split:latest
```

b) Servicios de colas (tareas) y base de datos (estadísticas) Dockerizadas

Se utilizan las imágenes oficiales de Bitnami RabbitMQ y MariaDB Galera para Kubernetes (Helm Charts) [SPI19], con licencia Apache License v2.0, las que permiten crear un sistema de colas y bases de datos auto escalables y redundantes, donde los trabajos y las estadísticas se publican.

El middleware de colas RabbitMQ se utiliza para almacenar, publicar y distribuir los trabajos (jobs) de forma segura y asincrónica a los nodos de procesamiento. La alta disponibilidad está garantizada por la definición de políticas de HA en RabbitMQ [ROS14], la integración de múltiples contenedores RabbitMQ en Kubernetes, el uso de colas de tipo durable y mensajes persistentes [DON20][LIN19]. Una política define una expresión regular y un conjunto de

argumentos que hace que a una o más colas que coincidan con dicho patrón, se les apliquen las propiedades definidas. En este trabajo se definieron las políticas que se presentan en la figura 3.7.

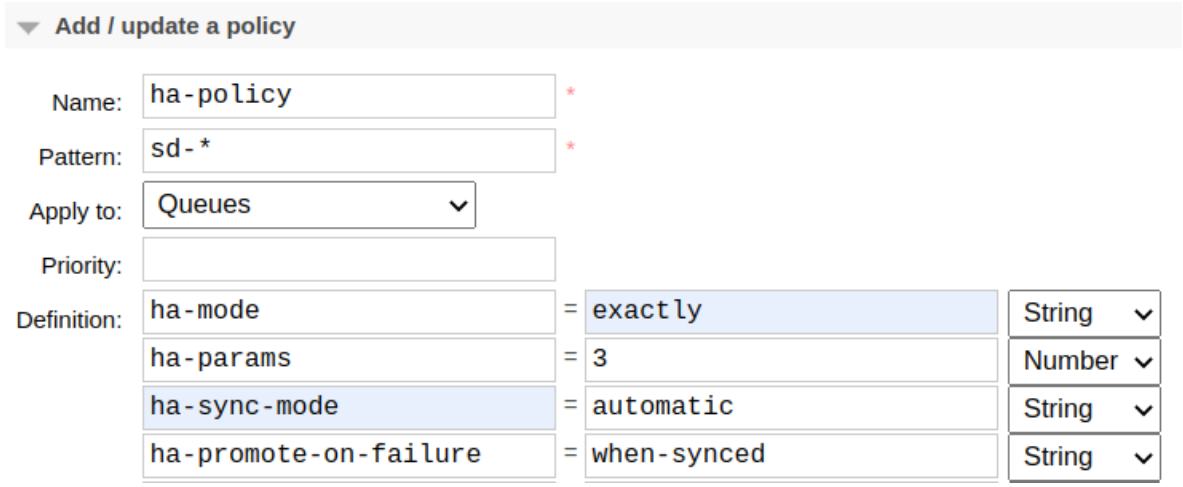

Figura 3.7 - Definición de políticas HA en RabbitMQ. Fuente: Propia.

Esta política (policy) garantiza que el contenido de cada cola estará replicado en tres (3) nodos del Clúster; y que la sincronización de dicho contenido estará manejada automáticamente por el sistema de colas. En caso de que un nodo se caiga o falle, Rabbit automáticamente replicará el contenido hacia un nuevo nodo del clúster, y lo pondrá disponible una vez que los datos se hayan sincronizado. De esta manera, se asegura la disponibilidad del servicio y la coherencia de los datos.

Para el despliegue de RabbitMQ se utilizan múltiples contenedores en Kubernetes, siguiendo las recomendaciones del proveedor (values-production.yaml), donde se establecen tres réplicas del servicio (replicaCount: 3), permitiendo aplicar las reglas de HA antes definidas.

La administración de las tareas, seguridad, resguardo y coherencia están garantizadas gracias a la implementación de un modelo de confirmación (ACK) manual donde, si se produce un error en un servidor, cliente o expira una marca de tiempo, RabbitMQ devuelve automáticamente la tarea incompleta a la cola, y allí esperará nuevamente a ser solicitada.

El motor de bases de datos MariaDB por su parte, se utiliza para registrar información de estado y estadística de las tareas (parámetros, fragmentos, trabajos completados y almacenamiento de punto de acceso al recurso). La plataforma hace uso de estos valores para verificar el estado de las operaciones y unificar las partes cuando se ha finalizado la misma. De hecho, la interfaz de usuario utiliza esta información periódicamente para mostrar el progreso de las actividades.

Además, se almacena información sobre las tareas ejecutadas (trabajo, nodo trabajador encargado y tiempo de ejecución). Estas estadísticas se utilizan cuando se realiza la experimentación y evaluación sobre la plataforma, permitiendo registrar el rendimiento y eficiencia de los nodos trabajadores al ejecutar diferentes escenarios de carga, y validar el comportamiento del sistema en general.

La alta disponibilidad en el motor de base de datos se configura a través de la topología multimaestro activo-activo, ofrecida por el plugin Galera para MariaDB [LAR19][VAS18] en Kubernetes, siguiendo también las recomendaciones del proveedor (values-production.yaml). Dicha definición genera tres réplicas del servicio (replicaCount: 3) y configura los mecanismos de copia de contenido (wsrep_on=ON, wsrep_replicate_myisam=ON) lo que garantiza escalabilidad y confiabilidad en las transacciones al mismo tiempo que se aseguran mínimas latencias.

c) Orquestación, escalado, replicación y automatización de contenedores a través de Kubernetes

Cómo se describió, (a) los microservicios de la plataforma, (b) las herramientas de soporte de la arquitectura HPC y (d) la versión del worker x86 (que se describe más adelante), se estandarizaron y empaquetaron utilizando Docker para correr, controlar, automatizar, desplegar e implementar los servicios de manera transparente a la infraestructura subyacente. Sin embargo, como el sistema debe estar preparado para escalar y ser flexible, es decir que puede existir una cantidad muy variable de cada uno de sus servicios, gestionar y monitorear dicha infraestructura manualmente no resulta una solución viable.

En este punto entonces, es necesario contar con un orquestador de servicios contenedorizados que nos permita:

- Administrar de manera global los distintos servicios que componen la arquitectura distribuida.
- Contar con una coordinación para hacer su despliegue.
- Supervisar el estado de los servicios en ejecución.
- Manejar el escalado automático (horizontal y vertical).

Por tal motivo, se investigaron las plataformas de Orquestación de contenedores y se decidió integrar, configurar y desplegar Kubernetes (K8s) [BUR19]. Esta decisión fue tomada en base a que es el motor de orquestación más difundido, tanto a nivel empresarial como académico,

y también el que ofrece mayores ventajas, características y posibilidades de configuraciones para correr, alojar, orquestar, controlar y monitorear la arquitectura Dockerizada del proyecto [PAN19][JAW19][TRU18][TRU18b]. Específicamente, la propuesta se beneficia de K8s, en los siguientes apartados:

- Alta disponibilidad: Kubernetes es una plataforma de código abierto y líder de mercado que permite construir una infraestructura de alta disponibilidad para alojar y correr contenedores.

- Portabilidad: Permite utilizar y portar su infraestructura, en cualquier lugar y ambiente, y así orquestar despliegues in situ, en Nubes públicas y entornos híbridos.

- Gestión de la configuración: Integra y maneja servicios de seguridad, redes y almacenamiento que se asocian a los contenedores. Dentro de las características más relevantes se encuentran las siguientes:

    - Despliegue declarativo: Kubernetes permite definir y administrar los servicios de forma declarativa, lo que garantiza que las aplicaciones implementadas siempre se ejecuten del modo que se describieron.

    - Escalabilidad: Este orquestador está preparado para trabajar a escala (horizontal y vertical) en tiempo real, ya que fue diseñado sobre los mismos principios desde el inicio que permitieron a sus creadores (Google) a ejecutar miles de millones de servicios cada día.

    - Planificación y optimización de costos: Decide sobre qué nodo ejecutará cada contenedor, de acuerdo a los recursos que requiera y a otras restricciones. Mezcla cargas de trabajo críticas y de mejor esfuerzo (best-effort) para potenciar la utilización y el ahorro de recursos. También se encarga de realizar comprobaciones de estado y autoregeneración de sus aplicaciones con ubicación, reinicio, replicación y escalamiento automáticos.

    - Descubrimiento de servicios y balanceo de carga: Kubernetes asigna a los contenedores sus propias direcciones IP y un nombre DNS único, para un conjunto de contenedores, y puede balancear la carga sobre ellos.

    - Transparencia: También permite que los sistemas distribuidos sean percibidos como un todo; y no como una colección de componentes cooperantes. Las ubicaciones de recursos involucrados en las operaciones, trabajos

concurrentes, replicación de datos, descubrimiento de recursos de múltiples sitios, fallas, recuperación del sistema, etc. se ocultan a los usuarios finales.

- Automatización y reversiones: Se encarga de integrar automatizaciones para las implementaciones (despliegues y actualizaciones) y reversiones. Si luego de aplicar un cambio en una aplicación surge una falla, Kubernetes permite revertir a la versión anterior, de manera simple y transparente.

A continuación, se describe cómo se implementa e integra Kubernetes en este proyecto, y cómo se definen y aplican las características antes mencionadas con objeto de construir una plataforma de alta disponibilidad, escalable y flexible para resolver las responsabilidades requeridas.

Alta disponibilidad y portabilidad:

Con objeto de validar la capacidad de la plataforma para ejecutarse y adaptarse a diversos entornos de trabajo (in situ, Nubes públicas o híbridas, y ambientes de bajos recursos), además del desarrollo de los microservicios que integran el sistema, los componentes se despliegan sobre diversos entornos Kubernetes. Se detallan a continuación:

- Despliegue en la Nube: Para desplegar el sistema en Nubes públicas, se seleccionaron los dos proveedores más grandes y representativos del mercado, en la actualidad: Amazon Web Services y Azure Cloud Computing [MUH20] [PIE20] [AJL18] [KOT18] [DOR14]. Con estos proveedores, la plataforma se implementó a través de los servicios de Kubernetes como Servicio (KaaS). Microsoft ofrece KaaS bajo el nombre de Azure Kubernetes Service (AKS) [BUC19] y Amazon ofrece Elastic Kubernetes Service (EKS) [IFR19].

- Despliegue On Premise x86: Para el caso de ambientes locales y Nube privada, se utilizó hardware x86 disponible (servidores de trabajo Blade HP ProLiant BL460c G8), montados sobre el virtualizador Vmware ESXi 6.5 y Vmware Vsphere VCenter y máquinas virtuales subyacentes basadas en CentOS 8. En esta arquitectura, se construyó el clúster de Kubernetes (K8s) con las herramientas nativas de Kubernetes (Docker, Kubeadm, Kubelet y Kubectl). Con ellas, se automatiza la instalación y la configuración de componentes de Kubernetes (servidor API, Controller Manager, DNS, entre otros). Para la creación automatizada y escalado de máquinas virtuales (Infraestructura como Código) se utiliza el producto Terraform de HashiCorp [WRI19] [MIK19] [BRI19] [RED18].

- Despliegue On Premise ARM: Para el caso de dispositivos de bajo costo de tipo IoT, se utilizó hardware ARM disponible (Raspberry Pi 3b+) corriendo sobre su sistema operativo de base (Raspberry OS). En dicha infraestructura se construyó el clúster de Kubernetes utilizando la distribución Rancher K3s [RAN20]. K3s es una versión certificada y de alta disponibilidad, diseñada para desplegarse en equipamiento que dispone recursos limitados o dentro de dispositivos de IoT. Está empaquetado como un único reducido binario de menos de 40 MB que reduce las dependencias y los pasos necesarios para instalar, ejecutar y actualizar automáticamente un clúster de producción de Kubernetes.

Tanto para el caso x86 cómo ARM, fue necesario definir: servicios de descubrimiento (DHCP) para interconectar los equipos; un servicio de nombres (DNS) para identificar los nodos; y configurar los permisos y reglas de acceso (Firewall) de cada estación de trabajo, para que se interconecten sin problemas.

Los detalles de los archivos y configuraciones utilizadas para todos los despliegues se presentan en el Anexo I - Despliegue de la plataforma en Kubernetes.

Al requerir de arquitecturas que garanticen un alto nivel de servicio y disponibilidad (HA), se utilizaron y definieron configuraciones de clústeres de Kubernetes multimaestro. Para ello, todas las distribuciones de Kubernetes necesitan que los servicios de base de datos de estados (etcd), administración (scheduler), controlador (controller-manager) y API REST (apiserver) estén replicados. La documentación del proveedor propone dos tipos de configuraciones para cumplir con dicho objetivo, a saber:

- Configuración de nodos compartidos: Se requieren al menos 3 nodos. En cada uno de ellos, corren los 4 servicios antes descritos, como se presenta en la figura 3.8.

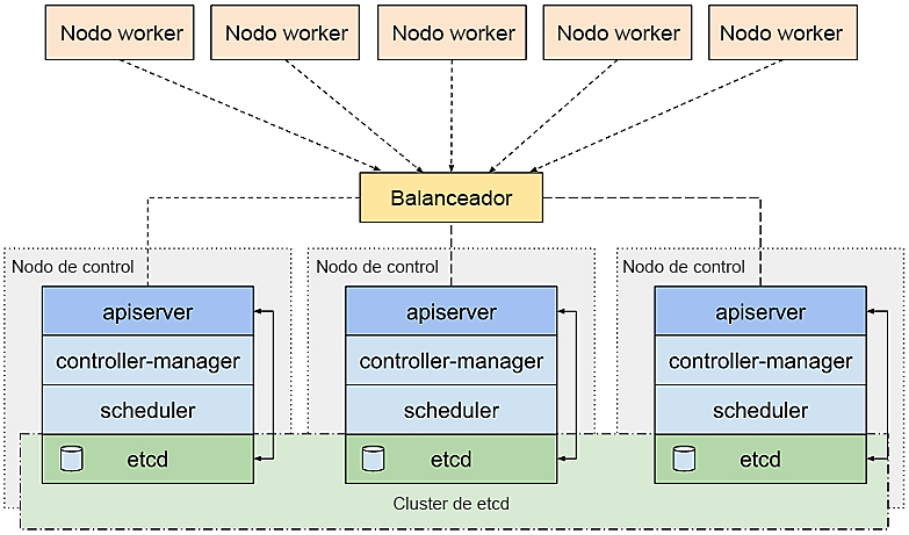

Figura 3.8 - Clúster de Kubernetes configurado para brindar alta disponibilidad (versión etcd integrada). Fuente: Propia. Versión adaptada de: https://d33wubrfki0l68.cloudfront.net/d1411cded83856552f37911e

- Configuración de nodos independientes: En esta arquitectura, al menos 3 nodos maestros contienen los servicios de apiserver, controller-manager y scheduler; y al menos otros 3 están completamente dedicados para construir un clúster de base de datos (etcd), como se muestra en la figura 3.9.

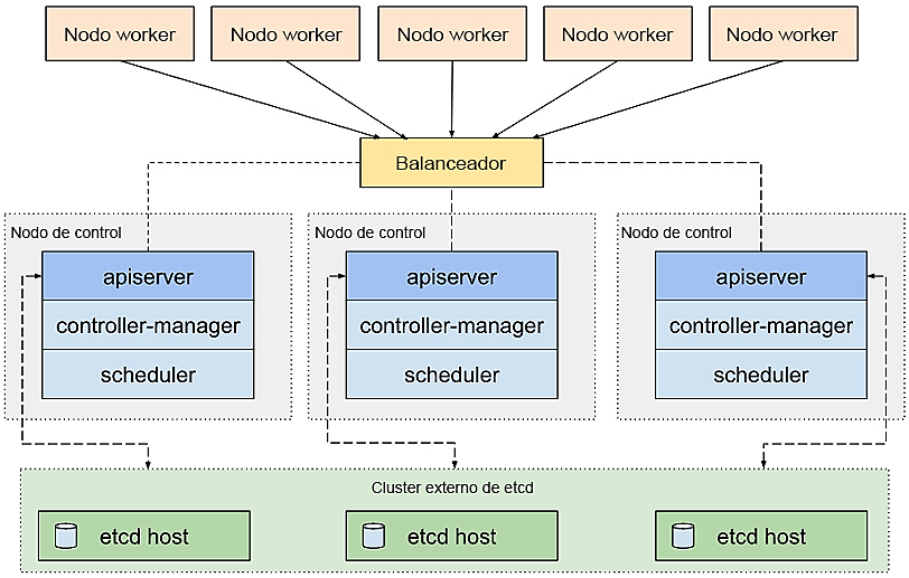

Figura 3.9 - Clúster de Kubernetes configurado para brindar alta disponibilidad (versión etcd en equipos independientes). Fuente: Propia. Versión adaptada de: https://d33wubrfki0l68.cloudfront.net/ad49fffce42d5a35ae0d0cc1186b97209d86b99c/5a6ae/images/kubeadm/kubeadm-ha-topology-external-etcd.svg

En el caso de la Nube, al consumir Kubernetes como servicio, la administración de la alta disponibilidad es gestionada por los proveedores; y es transparente para el usuario final.

Para el caso de los clústeres locales (sobre servidores x86 y sobre dispositivos ARM), se optó por crear la infraestructura HA compartida utilizando los 3 nodos maestros con todos los servicios integrados. Los detalles de los pasos utilizados en el clúster x86 con Kubeadm y en el clúster ARM con Rancher K3s se encuentran detallados en el Anexo I - Despliegue de la plataforma en Kubernetes.

Gestión de la configuración, automatización y reversiones

El paradigma clave de Kubernetes es su modelo declarativo. En este, el usuario proporciona el estado deseado y Kubernetes intenta ajustarse a dicha definición y se encarga de mantenerlo. Todas las acciones, necesidades y configuraciones se representan de forma declarativa a través de archivos YAML o JSON. Habitualmente se definen:

- Qué aplicaciones u otras cargas de trabajo se quieren ejecutar.
- Qué imágenes de contenedores usan y número de réplicas.
- Qué arquitectura de red y qué recursos de almacenamiento se necesitan.

Una vez construidos los archivos, el usuario los aplica a través de la API de Kubernetes donde el módulo controlador (control-plane) realizará las acciones necesarias, para que el estado actual del clúster coincida con el deseado. Para ello, Kubernetes realiza diferentes tareas de forma automática, como ser: parar, reiniciar o arrancar contenedores; o escalar la cantidad de réplicas de un servicio, cómo se muestra en la figura 3.10.

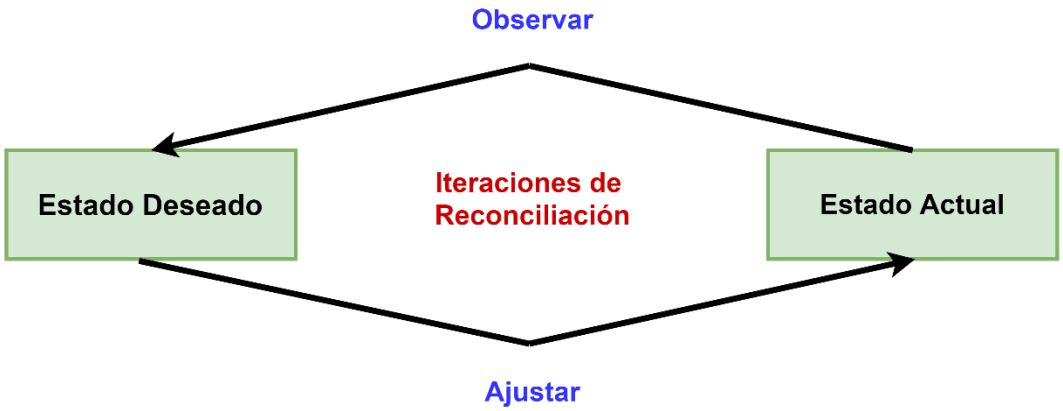

Figura 3.10 - Funciones de los operadores en Kubernetes para mantener el estado de los servicios definidos. Fuente: Propia. Versión adaptada de: https://blog.container-solutions.com/hs-fs/hubfs/kubernetes_operators_diagram1.png?width=750&name=kubernetes_operators_diagram1.png

Se decidió definir los servicios del clúster utilizando el formato YAML, debido a que por ser menos detallado, es más simple de editar y leer (indentado) que la estructura JSON.

Para realizar una gestión ordenada de estos archivos, sus despliegues y modificaciones, definimos el siguiente proceso de trabajo:

- Administrar el versionado (cambios) en los archivos YAML (VCS GitHub).
- Validar que los archivos estén correctos sintácticamente y semánticamente (kubeval - VCS GitHub).
- Ejecutar/Desplegar las configuraciones en K8s (VCS GitHub - kubectl apply).
- Validar que la configuración aplicada responda a las necesidades.
- Disponer de un proceso de reversión en caso de falla (VCS GitHub - kubectl apply).

Cabe destacar que para realizar el versionado sobre los archivos en el VCS Github, se utiliza un esquema de etiquetas para las versiones que van a desplegarse en el ambiente productivo. Esto implica que puede haber N actualizaciones en el repositorio, pero los procesos automáticos de despliegue solo tendrán en cuenta, tanto para la actualización y/o despliegue de un servicio, como para la reversión en caso de falla, las versiones publicadas con la etiqueta “producción”, como se muestra en la figura 3.11.

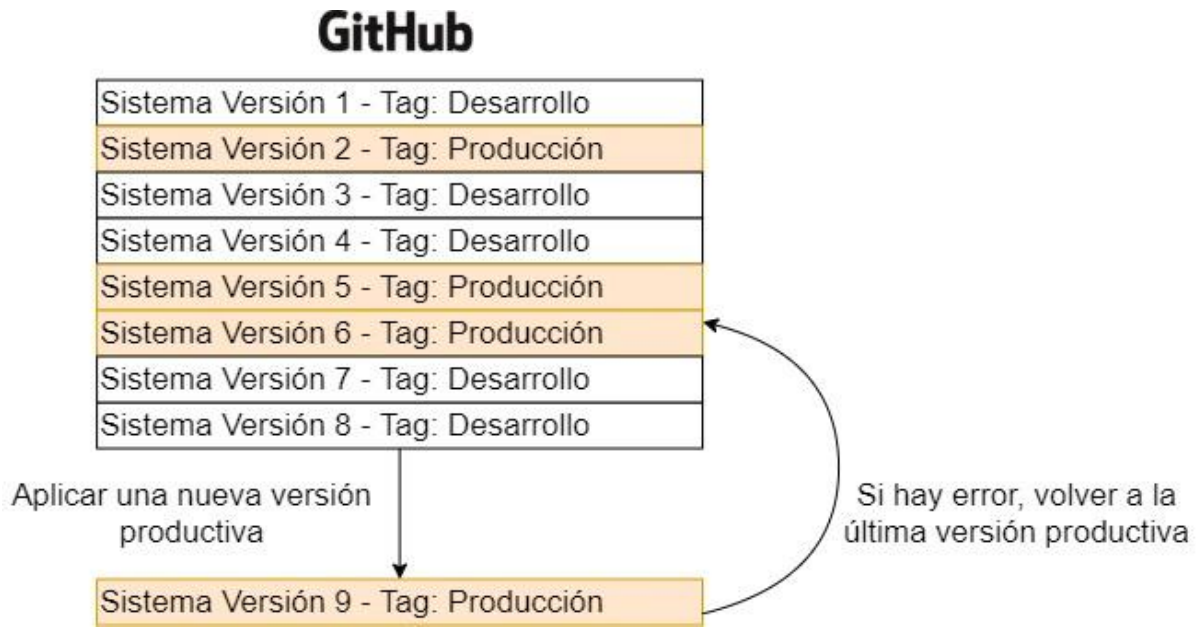

Figura 3.11 - Manejo de versiones, etiquetas e implementaciones de código en la plataforma. Fuente: Propia.

La gestión de la configuración del proyecto cubre dos grandes apartados. Por un lado, la definición, despliegue y aplicación de actualizaciones (modificaciones) sobre:

- Los microservicios Dockerizados desarrollados.
- Los servicios de base de datos y middleware de colas.
- Los servicios de almacenamiento.

Por otro lado, la definición de todas las herramientas necesarias para que dichos servicios y recursos cuenten con:

- Acceso seguro y confiable.
- Posibilidad de intercomunicación tanto dentro como fuera del clúster de Kubernetes.
- Alta disponibilidad, tolerancia a fallos y balanceo de carga.

Para ello, se definen varios archivos YAML que permiten definir los siguientes objetos y componentes en el Clúster:

a) Aislamiento y control de Recursos: Namespaces, ResourceQuota, Container requests - Container limits

b) Redes y Balanceo de carga: Ingress Controller y Services.

c) Despliegue de aplicaciones con HA: deployments, pods y ReplicaSet.

d) Escalabilidad: Horizontal-Pod-Autoescaler (HPA) y Clúster Autoscaler.

a) Aislamiento de Recursos:

En Kubernetes es posible crear múltiples clústeres virtuales respaldados por el mismo clúster físico, a través de la definición de diversos espacios de nombres (namespaces). Esto permite aislar, dividir y proteger los recursos del clúster [SHA20]; y definir reglas y políticas de acceso entre múltiples servicios [SHA20], a través de la definición de cuotas de recursos (Resource quota).

En este proyecto, a la hora de desplegar la arquitectura desarrollada, se definieron dos espacios de nombres, como se muestra en la figura 3.12.

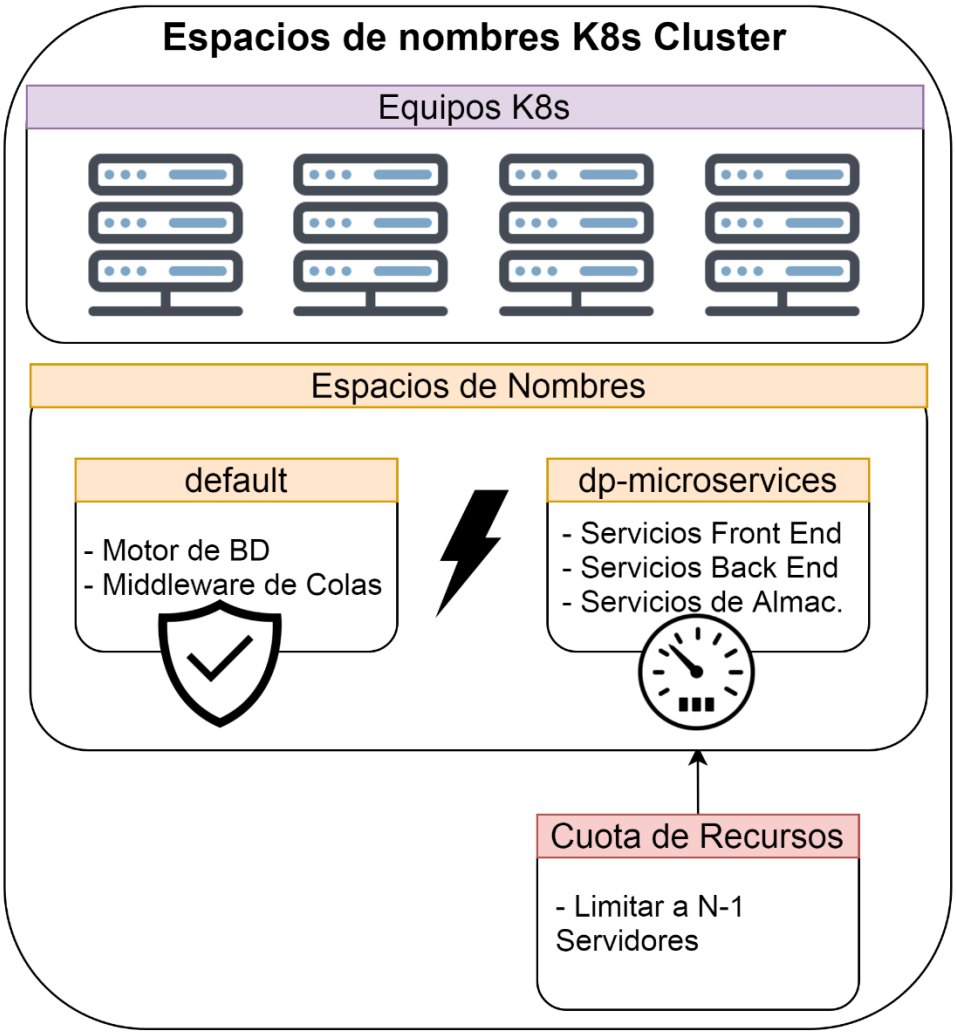

Figura 3.12 - Espacio de nombres definidos para el despliegue de la plataforma HPC en Kubernetes. Fuente: Propia.

El primero, es el que viene configurado por defecto con el clúster (default), en el cual se desplegaron los servicios del sistema de colas y del motor de bases de datos, sin establecer límites de recursos ni otras restricciones. El segundo, que se definió con el nombre de “dp-microservices”, en donde se crean e inician todos los microservicios definidos para el funcionamiento y administración de la plataforma (frontend, backend, almacenamiento). El espacio dp-microservices fue limitado a través de la definición de una cuota (dp-microservices-quota.yaml) que controla que se utilice una capacidad de N-1 servidores físicos del clúster. Esa información se obtiene a través de un pequeño código desarrollado (check-limit-resources.sh) que se encarga de consultar la cantidad y capacidad de los nodos del clúster periódicamente.

Los límites en el espacio “dp-microservices” permiten garantizar que los microservicios desplegados no consuman la totalidad de las capacidades del clúster. De esta manera, se

garantiza el funcionamiento tanto del motor de base de datos como del middleware de colas que se encuentran aislados en el espacio “default”.

Además del aislamiento a nivel de espacio de nombres, Kubernetes proporciona diferentes niveles de calidad de servicio a los contenedores en ejecución (pods), según los recursos que soliciten, y los límites que se establezcan para ellos. De esta manera, los pods que necesitan mantenerse activos pueden hacerlo, y consistentemente solicitar recursos garantizados; mientras que los contenedores con requisitos menos exigentes pueden usar recursos con menor prioridad o sin garantía. Para gestionar correctamente estos recursos, se debe definir la cantidad mínima de cpu y memoria que necesita para funcionar (resource requests), así como también los valores máximos (limit requests) que pueden alcanzar de cada uno de los pods. De esta manera, Kubernetes tiene la capacidad de asignar una categoría de nivel de servicio (QoS classes) a cada pod; y se garantiza que el orquestador podrá gestionar ordenadamente los despliegues, asignación de prioridades y desalojos de los servicios (pods), en la medida que sus recursos disponibles comienzan a verse afectados.

En la arquitectura desarrollada, cada microservicio dockerizado tiene definido tanto las reservas cómo los límites de recursos, de manera que Kubernetes lo asignará en la Clase QoS “Ajustable” (Burstable). Para definir estos límites correctamente se realizaron varias pruebas de configuración, de manera que cada función se cree y ejecute correctamente (cuota reserva) y trabaje intensivamente cuando así se requiera (cuota límite). En la figura 3.13 se muestra el comportamiento de los pods basados en una arquitectura burstable.

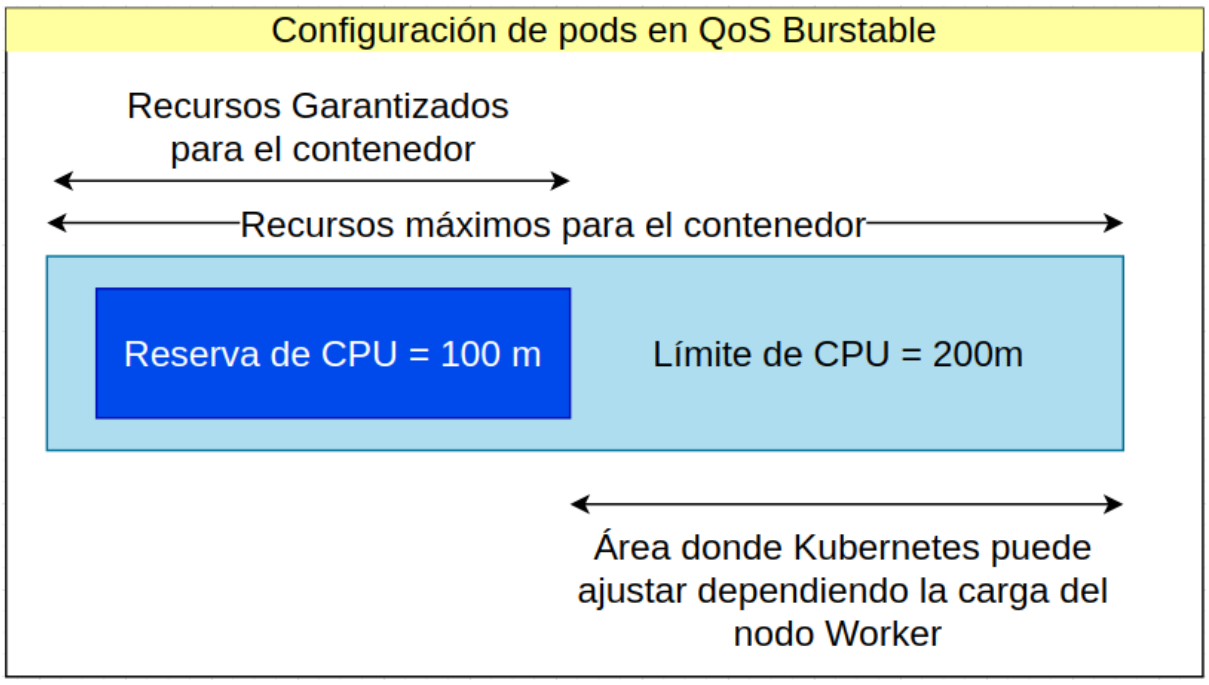

Figura 3.13 - Comportamiento de una aplicación basada en una Clase QoS Burstable (ampliable). Fuente: Propia.

En particular, estos recursos se definieron de la siguiente manera:

- Servicios Front-end, Servicios Back-end, Servicios de BD y Colas: Reserva: 0,2 cpu y 200 Mb; Límites: 2 cpu y 2 Gb de memoria ram.

Se define como tope máximo un valor de 2 cpu y 2 GB de memoria volátil para cada pod, debido a que es la capacidad mínima que puede llegar a tener un nodo trabajador (worker) en Kubernetes; tanto en Nube como Local. El objetivo es que un pod nunca supere la capacidad de un nodo Worker de Kubernetes y produzca incompatibilidades o inestabilidad en la plataforma de K8s.

b) Redes y Balanceo de carga: Ingress Controller y Services

En lo que respecta a la resolución de nombres y descubrimientos, Kubernetes trae configurado el componente CoreDNS (o KubeDNS para versiones antiguas), que permite que los contenedores (pods) puedan resolver diferentes nombres de recursos (servicios, despliegues, pods) a direcciones IP. CoreDNS se comunica con el servidor de API y comprueba los servicios y pods creados, para gestionar los diferentes registros de sus zonas de DNS.

Por lo tanto, una vez que se despliega un servicio (deployment) y que se inician los contenedores asociados (pods), el orquestador se encarga automáticamente de configurar dicha interconexión básica. No obstante, para permitir el acceso desde fuera de un clúster u otorgar balanceo de carga [DUA20] cuando se dispone de pods replicados, es necesario establecer algunas reglas y servicios adicionales.

El primer paso para lograr ese objetivo es definir un servicio (service) en Kubernetes, que es un recurso creado con el fin de mantener un único y constante punto de acceso a un grupo de pods (grupo de contenedores), luego de haber realizado su despliegue (deployment). Esta es la herramienta que proporciona Kubernetes para conectar diferentes elementos de un clúster internamente (entre ellos) y externamente (desde Internet, por ejemplo). Para el caso interno, automáticamente con el despliegue del servicio, se asigna una dirección IP de clúster que nunca cambia. Para exponerlos a Internet existen varias maneras que se detallan a continuación:

- Exponiendo puertos en las máquinas del clúster (NodePort).
- Exponiendo un Balanceador de Carga automático (Load Balancer), que sólo está disponible de forma automática para los proveedores en la Nube.
- Definiendo un Ingress Controller [VMW20] [VAY19], disponible en ambas arquitecturas.

Debido a las limitaciones del acceso vía NodePort y a que el servicio de Load Balancer automático sólo está disponible en los servidores en la Nube, se optó por configurar y utilizar Ingress Controller. Un Ingress Controller es básicamente un proxy, el cual permite, (utilizando el dominio definido en la solicitud que está llegando a dicho servicio) redirigir esa request a distintos despliegues (servicios) dentro de nuestro clúster; independientemente de la arquitectura subyacente. Este objeto Ingress debe estar asociado con uno o más elementos (cada uno de ellos está vinculado a un conjunto de pods). Al mismo tiempo, el servicio Ingress se ocupa de balancear la carga de manera equilibrada hacia los contenedores de cada servicio, cómo se muestra en la figura 3.14.

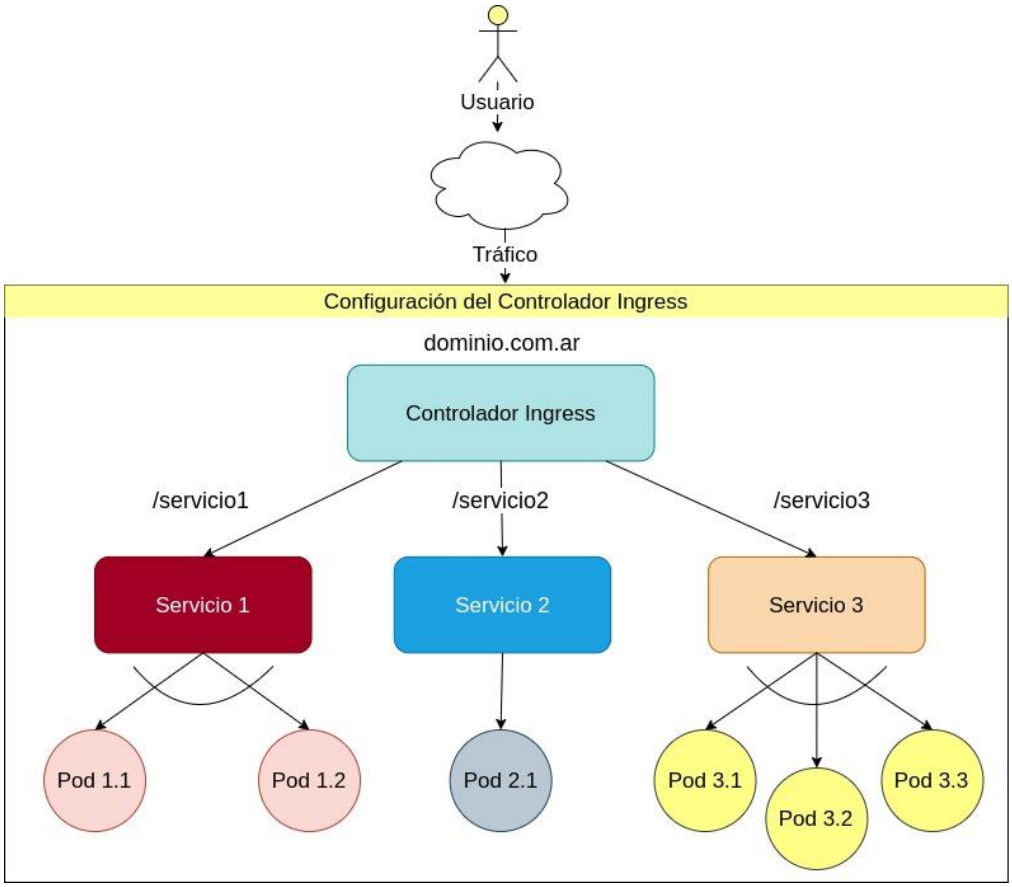

Figura 3.14 - Publicación de servicios a Internet utilizando Ingress Controller (Controlador Ingress). Fuente: Propia.

En la arquitectura desarrollada se utilizó un modelo en N capas, donde el usuario final sólo puede acceder a un sector reducido, y el resto de los componentes están ocultos, lo que permite proteger al sistema de accesos a datos, acciones y configuraciones no deseadas.

Los únicos puntos publicados al mundo exterior son:

- Los microservicios Front-end.
- Una parte de los microservicios Back-End Web (System Status API).

El primero permite a los usuarios interactuar con la plataforma para acceder a las funciones HPC, visualizar las tareas en ejecución y los resultados. El segundo, posibilita a través de un punto de acceso especial del API del Back-End Web, verificar el estado general de los servicios de la plataforma.

El resto de los servicios no son presentados al usuario final y disponen de una interconexión interna con sus vecinos. Esta interacción se presenta a continuación en la Figura 3.15.

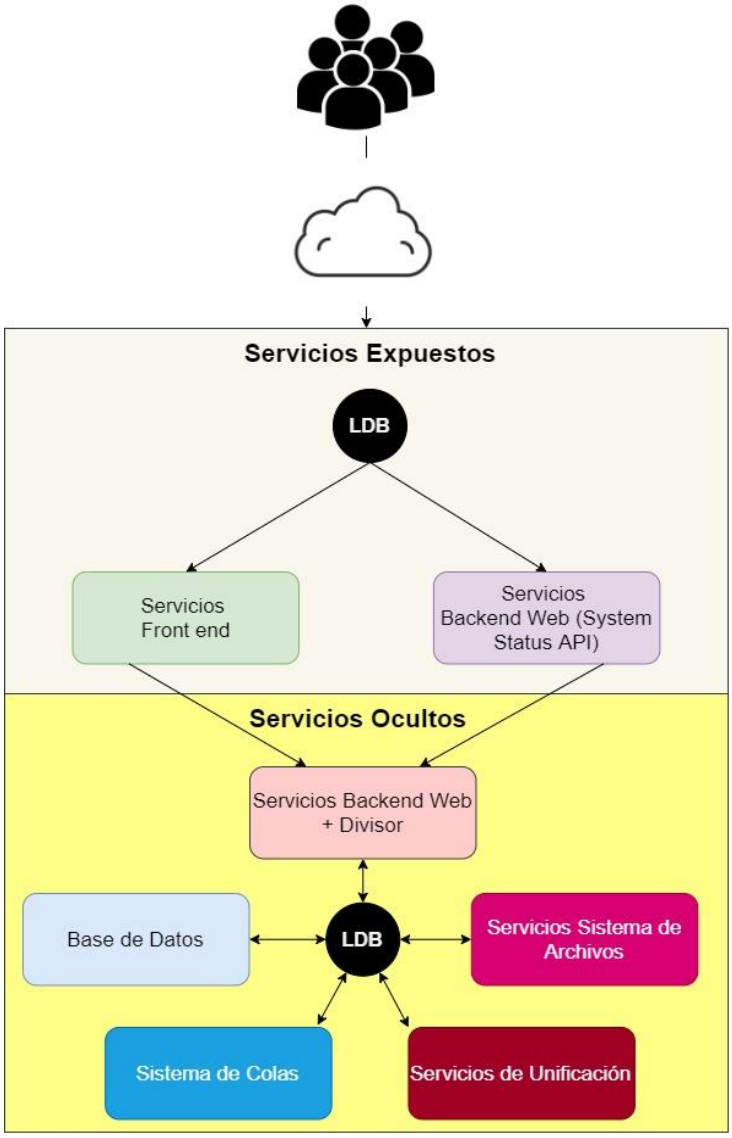

Figura 3.15 - Esquema de N capas en la plataforma desarrollada (Servicios expuestos e internos). Fuente: Propia.

Para cumplir con esta tarea se requiere entonces:

- Definir un servicio Ingress controller.
- Definir todos los servicios asociados a cada una de las responsabilidades.

i) Definición de Ingress Controller:

Como proxy reverso se decidió utilizar, de entre todas las posibilidades de integración que ofrece Kubernetes, Traefik 2 [SHA21] en su versión 2.3.6 (última estable). Se elige este proveedor Ingress debido a que:

- Es un Controlador Ingress de código-abierto.
- Está en el top 3 de implementaciones y reconocimiento (Github stars).
- Es simple de implementar, no requiere de una estructura compleja.
- Dispone de una interfaz Web dinámica.
- Tiene mecanismos de Auto descubrimiento.
- Ofrece compatibilidad con LetsEncrypt para generar certificados automáticos TLS (HTTPS) de los sitios y servicios.

Para utilizar el servicio de Traefik, fue necesario definir 4 archivos de configuración. En primer lugar se define un CRD en Kubernetes (001-crd.yaml) [DOB20]. Esto permite que el orquestador entienda las definiciones propias de Traefik y que se autorice la creación de reglas, permisos y servicios de balanceo. En segundo lugar, se determina un servicio (002-services.yaml), el cual permite hacer una asignación y manejo de puertos (NodePorts) de los hosts Maestros, para atender las solicitudes desde el exterior. En nuestra configuración se definieron los puertos 80/TCP para servicios HTTP, y 443/TCP para HTTPS. Más tarde se define un despliegue (003-deployments.yaml) el cual define la configuración de los servicios seguros (443/TCP) y no seguros (80/TCP), la cantidad de pods para ofrecer alta disponibilidad (réplicas), y finalmente, se despliegan las propiedades del Controlador Ingress (004-ingress.yaml); donde se establece la url de acceso para cada uno de los servicios, siendo “/tls” para el caso HTTPS y “/notls” para el caso HTTP. En la figura 3.16 se presenta un resumen de las propiedades y configuraciones definidas.

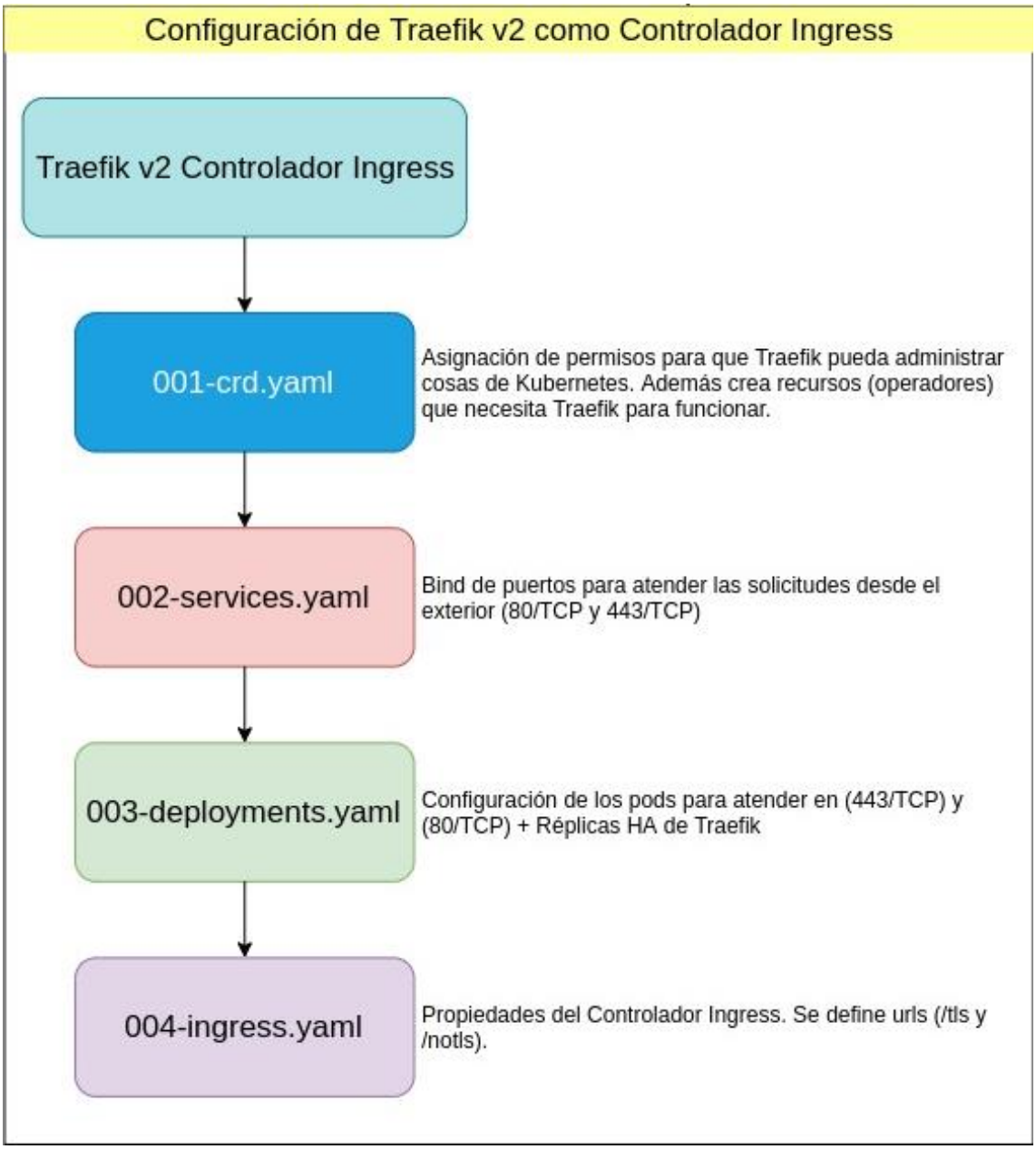

Figura 3.16 - Configuración de Traefik 2 como Ingress Controller (Controlador Ingress). Fuente: Propia.

ii) Definición de Servicios:

Se definió un servicio por cada una de las funciones desarrolladas (microservicename-service.yaml). Para los casos de Front-End y Backend-Web, tendrá la vinculación con el Ingress Controller + Clúster IP y para el resto solo la definición del Clúster IP.

Una vez iniciados los servicios en el clúster de Kubernetes, la información de las direcciones obtenidas se publicará en el repositorio de GitHub para que los microservicios puedan tomarlas para su comunicación y funcionamiento, a través de un pequeño script (obtain-services-ips.sh) cómo se muestra en la Figura 3.17.

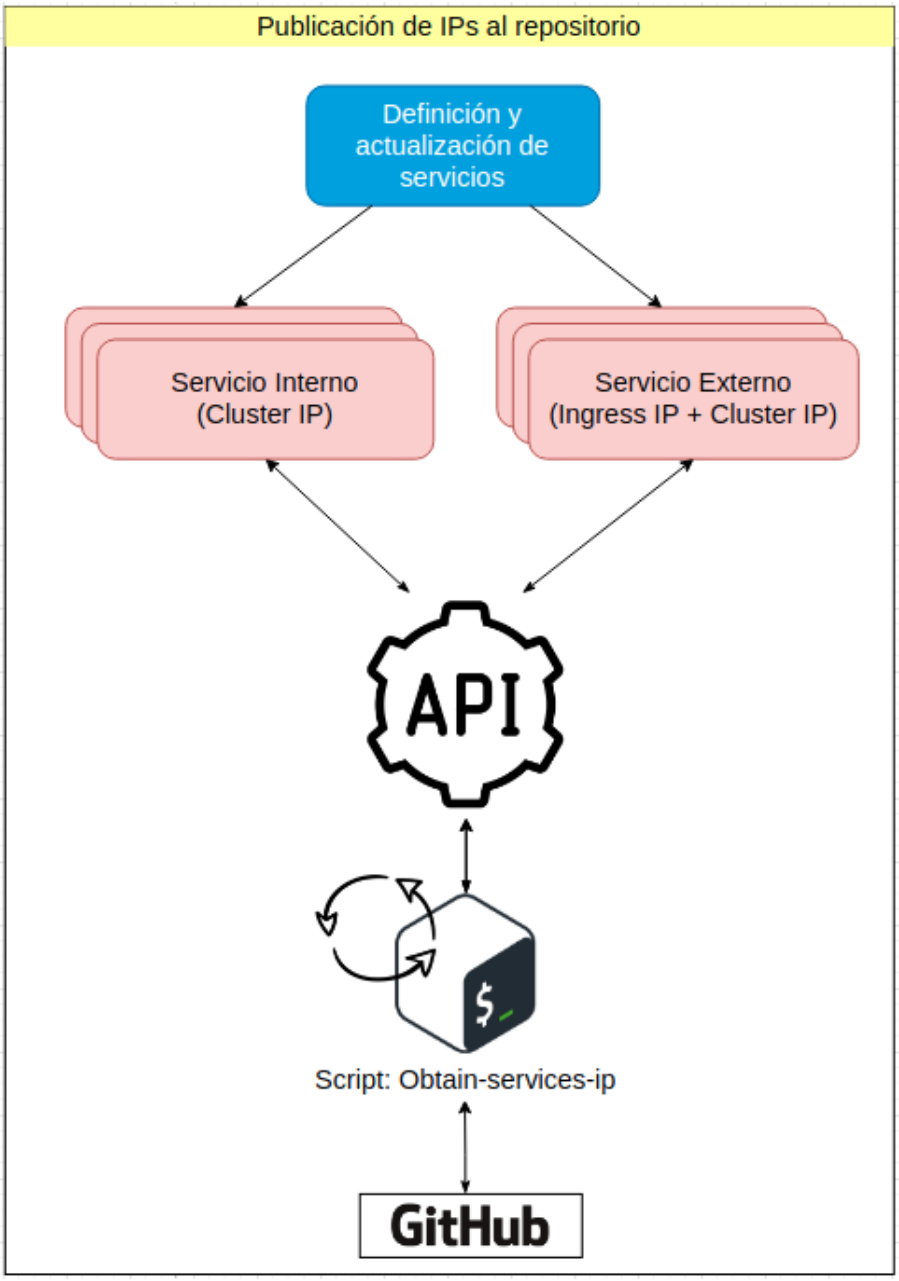

Figura 3.17 - Publicación de puntos de acceso para las aplicaciones en el repositorio GitHub. Fuente: Propia.

c) Despliegue de aplicaciones con HA: deployments, pods y réplicas

En este apartado, se explica cómo se construyeron los despliegues (deployments) para levantar los microservicios Dockerizados con alta disponibilidad [WUQ19][ABD19]. Un Deployment es la unidad de más alto nivel que se puede gestionar en Kubernetes. Permite, a través de un archivo YAML y su aplicación, definir diferentes acciones sobre una aplicación:

- Creación de la aplicación.
- Control de réplicas.

- Escabilidad de pods.
- Actualizaciones continuas.
- Despliegues automáticos.
- Rollback a versiones anteriores.

En este caso, para cada servicio se definió un archivo YAML (microservicename-deployment.yaml) que permite poner en ejecución, actualizar y/o revertir el microservicio correspondiente. En ese archivo, se define entonces:

- Microservicio a levantar.
- Cantidad de réplicas iniciales (réplica: 3).
- Límites de uso en cpu y memoria (reserva y máximo).

Una vez desplegada la aplicación, se generan la cantidad de pods (contenedores) y se le asigna a cada uno los límites de consumo definidos en el archivo YAML, cómo se puede ver en la Figura 3.18. De esta manera, se garantiza que Kubernetes cree un servicio con Alta Disponibilidad, evitando únicos puntos de fallo [VAY19].

En el caso que un pod tenga algún problema, Kubernetes automáticamente lo reiniciará o generará uno nuevo, para mantener la regla establecida de réplicas, siendo 3 en este caso.

Además, se vincula con la capa de servicios definida, donde es posible, brindar acceso a las aplicaciones ya sea interna o externamente; y balancear la carga entre los pods disponibles, dependiendo de los requisitos de cada microservicio en ejecución.

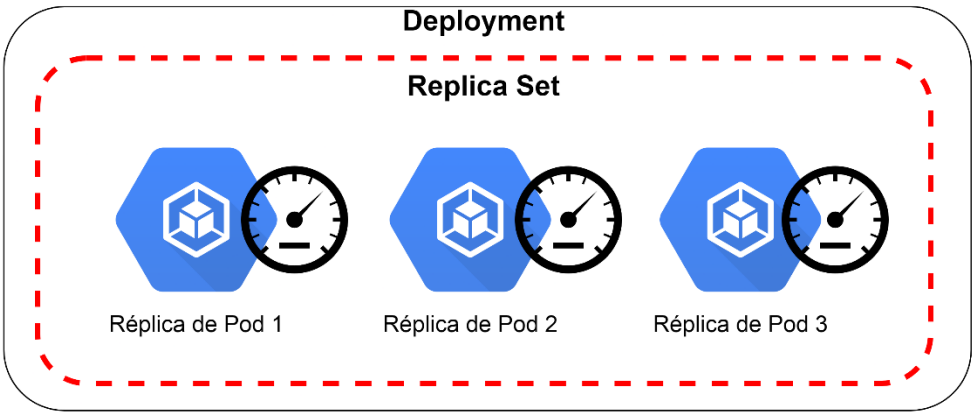

Figura 3.18 - Réplica y límites de recursos para los pods generados en un despliegue (deployment). Fuente: Propia.

En caso de que se detecte una modificación en la versión del archivo YAML en el repositorio y que esta se encuentre etiquetada con el nomenclador “producción”, automáticamente se aplicará la actualización sobre el despliegue (kubectl apply). De la misma manera, si se necesitara revertir a la última versión válida, también identificada con la etiqueta “producción”, se descarga el archivo YAML de dicha versión y se aplica utilizando el mismo comando (kubectl apply).

d) Escalabilidad: Horizontal-Pod-Autoescaler (HPA) y Clúster Autoscaler

Además del ReplicaSet que se genera a la hora de definir un despliegue (3 réplicas), Kubernetes brinda la posibilidad de definir políticas más avanzadas para el manejo de carga y escalabilidad de los servicios. Por un lado, el Horizontal-Pod-Autoescaler (HPA) [NGU20b] [VER19] permite variar el número de pods desplegados mediante un controlador de replicación (replication controller), que se asocia a nuestro despliegue (deployment), en función de diferentes métricas [HEZ20] [DEW19]. Estas pueden provenir de:

- El API de resource metrics, que brinda, sin necesidad de configurar detalles adicionales, el consumo de memoria y CPU de los pods.
- El API de custom metrics, que ofrece un bucle de control que comprueba de forma periódica el valor de alguna métrica manual y personalizada configurada en el HPA.
- Múltiples metrics, que permite combinar ambas arquitecturas de métricas (nativas y personalizadas)

En este proyecto se utilizó la primera opción, a través de la cual es posible determinar cuándo es necesario escalar (crecer, decrecer) horizontalmente los pods del servicio. Para lo cual se definió por cada despliegue de microservicio un HPA con un valor mínimo y máximo de pods alcanzables, y un rango de acción (threshold). La cantidad mínima por servicio es de 3 réplicas, lo que se corresponde con la cantidad establecida en el despliegue, y que nos permite garantizar un servicio de alta disponibilidad. El máximo es de 30 pods por servicio. Este último valor se definió debido a que da un margen grande de escalabilidad respecto de los valores utilizados en todas las pruebas realizadas. En el laboratorio nunca fueron necesarios más de 15 pods por tipo, con objeto de tener mecanismos de escalabilidad para un ambiente productivo concurrente.

El evento de ajuste se dispara cuando alguna de las dos métricas (cpu/memoria), en promedio de todos los pods de un servicio, supera el 75 % de uso por más de 30 segundos (targetAverageUtilization: 75). Estos valores elegidos, provienen de tomar como referencia

las recomendaciones generadas por la plataforma de Kubernetes [RED21] [GOO21] [NGU20] [HAN20] [TUC19]

Por otro lado, para el caso de desplegar la arquitectura en la Nube, también está disponible la característica de "Clúster Autoscaler" (limitado sólo a algunos proveedores) [TAM20][WAN20]. Esta función de escalado automático permite aumentar o disminuir el tamaño de un clúster de Kubernetes, para obtener la cantidad correcta de nodos trabajadores necesarios para atender las cargas de trabajo, cómo se muestra en la Figura 3.19. Para eso, Kubernetes chequea cada 10 segundos las siguientes métricas básicas:

- Utilización de nodos del clúster.
- Presencia de despliegues y pods pendientes en el clúster.

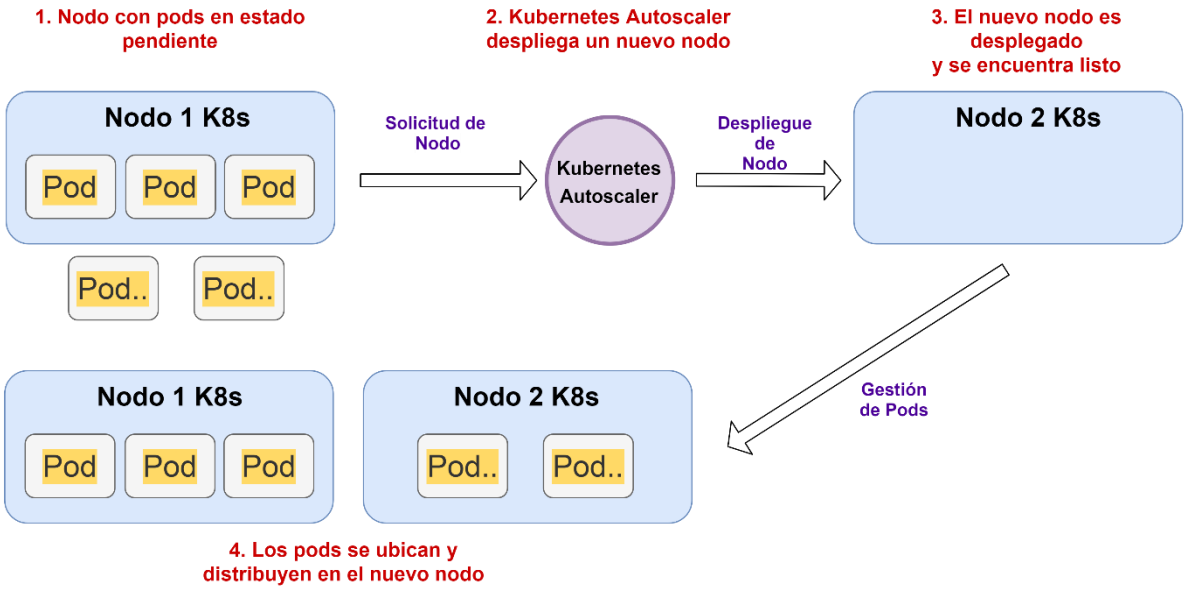

Figura 3.19 - Ejemplo de Clúster Autoscaling. Fuente: Propia. Versión adaptada de: https://blog.scaleway.com/content/images/2020/10/image-1.png

La configuración del autoescalador es dependiente del proveedor de servicio en la Nube, pero en todos los casos es necesario definir los siguientes parámetros:

- Tipo de instancia de máquina virtual.
- Cantidad mínima y máxima de nodos.
- Parámetros de reducción del clúster.
- Parámetros de incremento del clúster.
- Precio máximo que se desea pagar por una VM.

En esta plataforma, los parámetros fueron configurados de la siguiente manera:

- Tipo de instancia de máquina virtual: Instancias Spot de 2 cpus + 2gb Ram.
    - Precio máximo que se desea pagar por una VM: < $ 0.01 por hora.
- Cantidad mínima y máxima de nodos: min= 3; max= 20.
- Parámetros de reducción del clúster:
    - --scale-down-unneeded-time=5 min.
    - --scale-down-utilization-threshold=50 min.
- Parámetros de incremento del clúster:
    - --scale-up-utilization-threshold=70.

Todas las configuraciones fueron desarrolladas teniendo en cuenta las recomendaciones de los proveedores de los servicios y de la plataforma Kubernetes [GOO21][STA20][ZHO20] [ROD18b]

Es importante recordar que este proceso debe ser óptimo, debido a que al estar corriendo en la Nube afecta directamente al costo incurrido por la plataforma (Cost Optimization). Por lo tanto, como los nodos y agentes de Kubernetes pueden entrar y salir del clúster, se decide utilizar VMs de tipo “Spot Instances” [DAN20][SAM19][EKW18][VEEN16], que proveen grandes descuentos (entre un 60 y 90%) en comparación con los precios de las VM de pago por uso [PEV20][PET19b], cómo se muestra en la figura 3.20. Este es un tipo de máquina virtual, a través de la cual los proveedores de la Nube subastan a un precio reducido su capacidad ociosa, reservándose el derecho de terminar dicha instancia si ya no tiene suficiente capacidad para prestar el servicio.

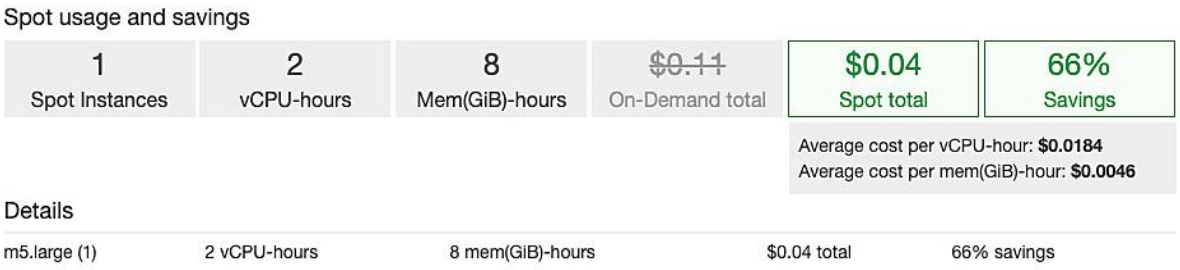

Figura 3.20 - Escala de ahorro obtenido para un caso de una instancia de AWS utilizando Spot Instances. Fuente: https://www.miguelg.com/2020/04/usar-spot-instances-amazon-jenkins.html

Mecanismos de Transparencia

La plataforma desarrollada se basa en una arquitectura distribuida [BRE18]. En este esquema diseñado, la solución se presenta ante el usuario como un solo sistema, ocultando la complejidad del protocolo distribuido. Los sistemas distribuidos deben cumplir, en lo posible, características de transparencia [BRE18]. Dentro de las más conocidas se encuentran las siguientes: transparencia de acceso, localización, concurrencia, replicación, fallos, migración y escalabilidad, cómo se presenta en la figura 3.21.

Figura 3.21 - Aspectos de transparencia para un Sistema Distribuido. Fuente: Propia. Versión adaptada de: https://www.codifica.me/wp-content/uploads/transparencia.gif

En esta plataforma, el usuario accede a los recursos del sistema de la misma manera en que accedería a componentes de un sitio web tradicional, es decir, a través de la interfaz gráfica presentada (GUI). Cuando un usuario realiza una solicitud, no advierte que la misma se distribuye por distintos nodos, con diferentes roles, así como tampoco conoce el lugar donde residen los datos de la aplicación. Una vez que es procesada la misma, el usuario sólo percibe

un conjunto respuesta, del cual desconoce por completo la complejidad del proceso HPC ejecutado por detrás.

Por lo tanto, se puede decir que la plataforma cumple con las condiciones de transparencia gracias a las siguientes características:

- Desarrollo del protocolo, funciones y servicios del sistema basado en microservicios Dockerizados.
- Implementación de procesos y técnicas DevOps para el despliegue y monitoreo de la infraestructura.
- Administración de la plataforma a través del orquestador Kubernetes.

Gracias a la arquitectura definida, es posible especificar políticas de comportamiento y alta disponibilidad, esquemas de replicación y escalabilidad para los servicios ofrecidos (gestor de base de datos, servidores HTTP, sistemas de colas, etc.).

A continuación, se describe en detalle cómo aplica a cada uno de los apartados de transparencia.

Transparencia de Acceso: El acceso a los recursos remotos se realiza mediante peticiones HTTP con los métodos conocidos (GET, POST, PUT), los cuales se comunican con la API REST presentada por la plataforma. La API del frontend atenderá los requests de los usuarios y se le presentará los resultados obtenidos de manera transparente. Está construida, como todas las del sistema, sobre una arquitectura en cluster, por lo que el usuario no podrá determinar a través de cual microservicio fue atendido.

Transparencia de Localización: El usuario realiza las peticiones HTTP contra un nombre de DNS o IP pública, provistos por el servicio de la Nube, o una dirección de clúster en el caso de ejecutarse en un laboratorio local. En ninguna de las situaciones, el usuario podrá saber específicamente dónde y cómo se ejecutan las aplicaciones, gracias a las capas de abstracción y al balanceador de carga que proporciona el clúster de Kubernetes.

Transparencia de Concurrencia: Los usuarios pueden solicitar paralelamente tareas HPC que el orquestador administra internamente y de manera transparente.

Gracias al uso de Kubernetes y al haber definido réplicas y mecanismos de escalado horizontal (HPA) para los microservicios de la plataforma (FrontEnd, Backend, Split y Joiner) y los sistemas Base de datos y middleware de colas, se garantiza que se cuenta con la

capacidad para responder ante la demanda creciente, concurrente y paralela de los usuarios, ocultando la complejidad de este proceso.

Transparencia de Escalado, Replicación, Tolerancia a Fallos y Migración: Las políticas y servicios definidos para el clúster de Kuberentes, permiten administrar la cantidad de sus servicios activos. Esto implica que cuando se agrega un recurso de administración a la red, el mismo se adosa a los servicios ya definidos, y se incluye en los mecanismos de balanceo de carga para explotar sus capacidades. De la misma manera, cuando un nodo de administración se retira, se realiza el proceso inverso, sin afectar el funcionamiento de la plataforma.

En caso de que la plataforma se encuentre en un entorno elástico (Nube o virtualizadores compatibles) puede también encargarse de flexibilizar y escalar la arquitectura (creando o eliminando nodos del clúster K8s) en base a la carga de trabajo.

Si un nodo falla, se detiene o presenta errores, Kubernetes se encargará de ubicar un nuevo servicio en algún otro nodo del clúster, de manera transparente para al usuario. En el caso particular de los servicios de Base de Datos y Middleware de Colas, no solo creará una nueva tarea, sino que lo enlazará al clúster de alta disponibilidad del servicio correspondiente.

En caso de que la falla se produzca durante la ejecución o transferencia de una tarea en un nodo de procesamiento, el sistema tendrá capacidad de recuperarse de manera transparente gracias a los servicios del middleware de colas, entregando dicho trabajo a un nuevo nodo de procesamiento disponible

d) Trabajadores móviles Android ARM y x86 Dockerizados.

Los nodos trabajadores se desarrollaron utilizando el lenguaje Java. Para la versión móvil con el SDK nativo (compatibilidad Android 7+, API Level 24) y Java JDK 14 dockerizando el producto para la versión desktop x86.

Características Generales:

Al iniciarse, el nodo trabajador, al igual que el resto de los microservicios obtendrá los parámetros de configuración desde el repositorio de Github, mediante los cuales podrá establecer las conexiones con el Sistema de Gestor de Base de Datos, el middleware de Colas y el microservicio Backend Web, para cumplir con sus objetivos.

Una vez configurado, los nodos trabajadores descargan, sincronizada e iterativamente, tareas del sistema de colas para ser procesadas. Para ello, establece una comunicación de tipo

Consumer definiendo la confirmación de los mensajes de manera no automática (ACK manual). Por esto, si el nodo de procesamiento experimenta algún error (falla en tarea HPC, desconexión, cierre inesperado, etc.) el mensaje se encola nuevamente y se evita la pérdida de tareas.

Una vez obtenido un trabajo, se inicia la marca de tiempo, se define el tipo de trabajo a realizar, y se descarga el recurso fuente en el nodo trabajador. Una vez con el archivo y los parámetros relacionados con la tarea, se inicia la actividad HPC.

Finalizado este proceso, se detiene la marca de tiempo y se actualiza la información estadística (tiempos) en la base de datos. A la vez, se sube el archivo resultante y se confirma la tarea realizada en el sistema de colas.

Características del nodo trabajador x86 Dockerizado

El Nodo trabajador x86 está desarrollado, como el resto de los servicios de la plataforma, con el lenguaje de programación Java (JDK 14).

Para construir este servicio de manera portable, se estandariza utilizando los mismos pasos definidos en el apartado Dockerización, mencionado anteriormente. Es decir:

1. Parametrizar la ejecución del nodo trabajador.
2. Exportar el nodo trabajador cómo un archivo ejecutable (Proyecto maven).
3. Definir el archivo Dockerfile para la construcción de la imagen Docker (adoptopenjdk/openjdk14).
4. Crear la imagen Docker y publicarla en Docker Hub (dpetrocelli/dev-worker:latest).
5. Automatizar el empaquetado, construcción y publicación (deploy-automation.sh).

Una vez disponible la imagen en el repositorio global de Docker Hub (dpetrocelli/dev-worker:latest), puede ejecutarse en cualquier equipo que disponga del motor de Docker instalado. En particular, está preparado para correr tanto en: a) un host Docker (individual) ; b) un clúster Kubernetes (deployment).

a. Ejecución en un host Docker

Para ejecutar el agente trabajador en un host Docker, existen dos maneras: a.1) de forma automática, y a.2) de forma parametrizada.

a.1) Ejecución automática:

El nodo trabajador viene listo para correr de manera automática, sin definir ningún tipo de configuración. A continuación, se muestra que con tan solo una línea se puede poner a funcionar:

```
$ docker run --name worker dpetrocelli/dev-workerx86:latest
Unable to find image 'dpetrocelli/dev-workerx86:latest' locally
latest: Pulling from dpetrocelli/dev-workerx86
d7c3167c320d: Pulling fs layer
131f805ec7fd: Download complete
322ed380e680: Download complete
6ac240b13098: Waiting

Digest:
sha256:af0dd63297bca4c5f77418e48a74f0dc782d1573f8beab429c39807374af6501

Status: Downloaded newer image for dpetrocelli/dev-workerx86:latest

…..

 Obtaining Configuration file

 MARIADB CONNECTED

 RABBIT CONNECTED

….

RabbitMQ Consume Looper has started

...
```

Sin embargo, no se recomienda ejecutarlo de esta manera ya que, a la hora de realizar la tarea HPC, puede consumir todos los recursos del equipamiento y afectar a la experiencia del usuario.

a.2) Ejecución parametrizada:

Para evitar los problemas antes mencionados se definió un script extra (limited-runner.sh) que permite configurar los parámetros de corrida del producto. Básicamente, se definieron dos categorías que se deben ajustar según las necesidades del usuario:

- Hora de inicio y hora de fin de ejecución, lo que busca no afectar la capacidad del equipamiento mientras se encuentra en los horarios productivos de trabajo del usuario.
- Límites de cpu y memoria, con objeto de evitar consumir más recursos de los esperados por parte de la plataforma.

A continuación, se presenta un pequeño fragmento del código que permite visualizar cómo se definen los parámetros y cómo impactan a la hora de correr el nodo trabajador:

```
initTime="10:00"
endTime="23:00"
cpus="2.0"
cpusetCpus="0,1"
memory="4000m"
containerName="worker"
while true; do
      currenttime=$(date +%H:%M)
      status=$(docker ps -a --filter "name=$containerName" --filter
"status=running"| grep Up)
      length=${#status}
      if [[ $currenttime>=$endTime ]] || [[ $currenttime<$initTime ]]; then
          if [[ $length>0 ]]; then
              docker container stop $containerName
              echo "Worker Process has been stopped"
          fi
      fi
      if [[ $currenttime>=$initTime ]] || [[ $currenttime<$endTime ]]; then
          if [[ $length<1 ]]; then
              docker container start $containerName --cpus=$cpus --cpuset-
cpus=$cpusetCpus --memory=$memory
              echo "Worker Process has been started"
          fi
      fi
      sleep 100
done
```

En este ejemplo, chequeando el horario actual del sistema y revisando el estado del proceso, se limita la ejecución del programa desde las 22hs. hasta las 6hs. del día siguiente. Fuera de ese horario, se detiene la actividad del mismo. Además, para la ejecución se limita a asignar 2 cpus fijos (cores 0 y 1) y 4GB de memoria RAM para el procesamiento de las tareas HPC.

b. Ejecución en un Clúster de Kubernetes

Además de permitir la ejecución del agente trabajador en un host aislado de algún usuario que disponga del motor Docker instalado, también se brinda la posibilidad de utilizar esta imagen dentro de una arquitectura de Kubernetes.

En este proyecto, esto brindó la posibilidad de contar no solo con la plataforma de administración de los servicios, sino también con el procesamiento dentro del clúster de Kubernetes. De esta manera, es posible aplicar las mismas reglas de disponibilidad, escalabilidad y flexibilidad que se presentaron a la hora de administrar los servicios de la plataforma. En consecuencia, se definieron los siguientes archivos:

- Un archivo YAML para el deployment (worker-x86-deployment.yaml), el cual permite construir una cantidad de contenedores iniciales (3 réplicas), así como también las reservas y límites de consumo para cada pod (cpu: min=0.2, max=2; memoria: min=100m, max=2000m).

- Un archivo YAML para la definición de reglas de flexibilización basado en el escalado horizontal de nodos (HPA). En particular, se definió un mínimo de 3 recursos en línea (compatible con el despliegue), y un máximo de 30 réplicas. La asignación de nodos según la cota máxima estará condicionado a la capacidad de la infraestructura subyacente y su posibilidad de escalar.

Características del nodo trabajador móvil

El agente trabajador móvil se desarrolló utilizando el SDK nativo de Java para Android, a través de Android Studio. Con este framework, se genera una aplicación empaquetada en formato .apk y se define una compatibilidad (API Level 24) para funcionar con cualquier dispositivo que tenga Android 7 (Nougat) o superior. Dicha versión fue presentada en 2016 y según la información oficial del Sistema Operativo, esto permitiría correr la aplicación en más de un 73% de los dispositivos alrededor del mundo, como se muestra en la figura 3.22. Cabe aclarar que en la etapa de evaluación de la plataforma y de los agentes trabajadores, se presenta una tabla de compatibilidad sobre los dispositivos móviles evaluados, que corrobora los datos aquí presentados.

Plataforma (Versión de Android)	Nivel de API	Distribución Cumulativa
4.0 - Ice Cream Sandwitch	15	
4.1 - Jelly Bean	16	99.8 %
4.2 - Jelly Bean	17	99.2 %
4.3 - Jelly Bean	18	98.4 %
4.4 - KitKat	19	98.1 %
5.0 - Lollipop	21	94.1 %
5.1 - Lollipo	22	92.3 %
6.0 - Marshmallow	23	84.9 %
7.0 - Nougat	24	73.7 %
7.1 - Nougat	25	66.2 %
8.0 - Oreo	26	60.8 %
8.1 - Oreo	27	53.5 %
9.0 - Pie	28	39.5 %
10 - Android 10	29	8.2 %

Figura 3.22 - Compatibilidad de dispositivos Android según el nivel de API. Fuente: Propia. Versión adaptada de: https://www.xda-developers.com/files/2020/04/Android-Distribution-Numbers-Android-Studio-1024x700.jpg

A la hora de la construcción y ejecución de la aplicación, fue necesario incluir una serie de configuraciones que se mencionan a continuación:

1. **Habilitación de permisos entre el .apk y el Sistema Operativo**.

Fue necesario brindar permisos entre la aplicación y el sistema operativo. En particular, habilitar la comunicación con el sistema de archivos de Android y navegación hacia Internet (sólo vía Wi-Fi para evitar el consumo de los datos móviles del usuario), lo que permite cargar y descargar los recursos necesarios. Para ello fue imprescindible:

- Definir la configuración de permisos: En la raíz del proyecto de Android Studio, se configuran los permisos (AndroidManifest.xml) como se muestra a continuación:

```
<uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
        android:name="android.permission.INTERNET" />
    <uses-permission
        android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- Habilitar permisos: Para que los permisos sobre el sistema de archivos se hagan efectivos, luego de instalar la aplicación .apk, el usuario debe habilitarlos, como se muestra a continuación, en las figuras 3.23 y 3.24.

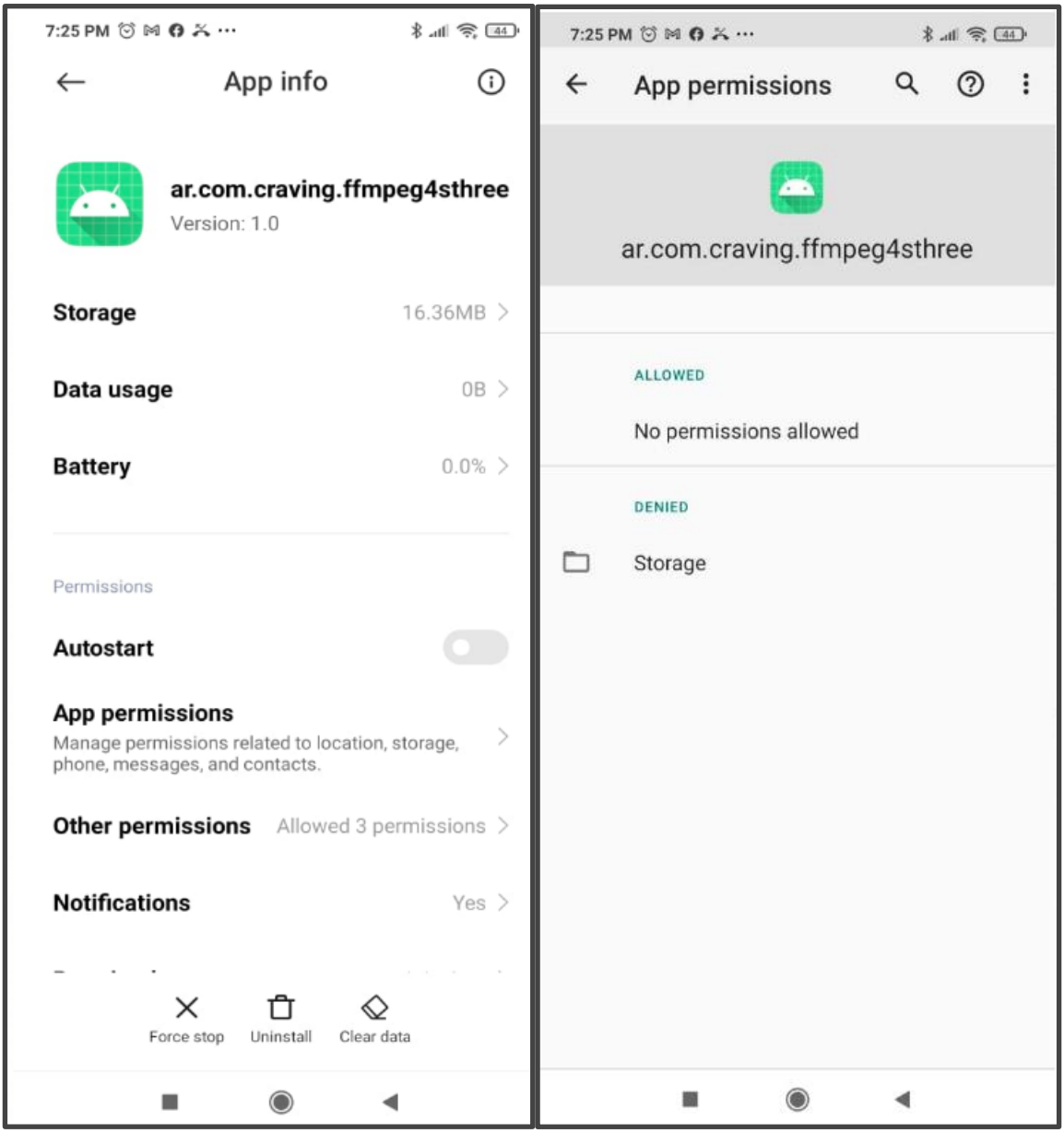

Figura 3.23 - Imagen inicial de aplicación instalada sin permisos de almacenamiento. Fuente: Propia.

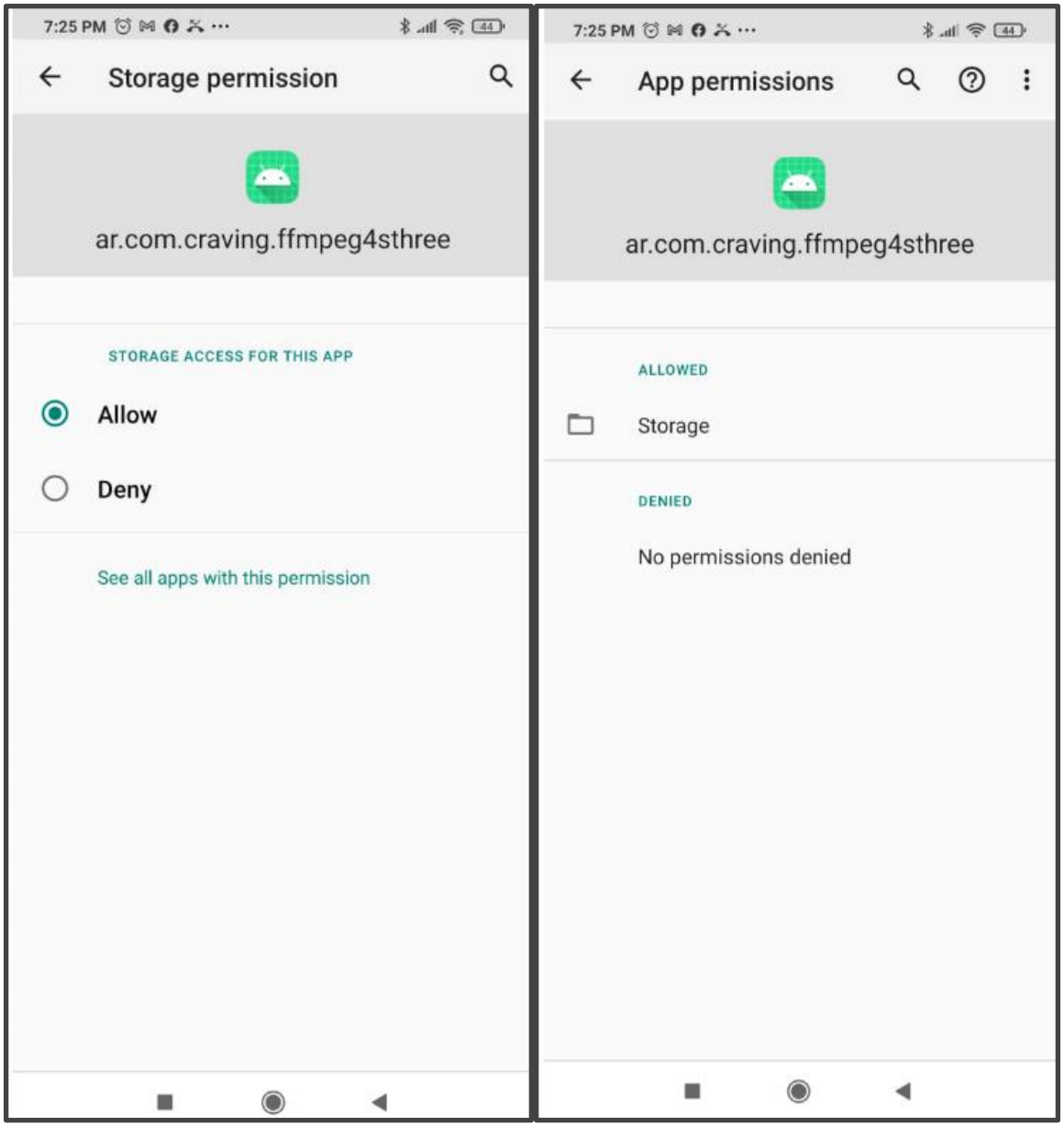

Figura 3.24 - Pasos de habilitación de permisos que debe seguir el usuario para comenzar a utilizar la aplicación móvil HPC. Fuente: Propia.

- Lanzar aplicación: Una vez otorgados los permisos se puede lanzar la aplicación, la cual verificará que los permisos se hayan otorgado correctamente, cómo se muestra en el siguiente apartado:

```
...
if (!isGranted(Manifest.permission.READ_EXTERNAL_STORAGE) ||
!isGranted(Manifest.permission.WRITE_EXTERNAL_STORAGE)
!isGranted(Manifest.permission.WIFI))

!isGranted(Manifest.permission.INTERNET))
```

```
!isGranted(Manifest.permission.ACCESS_NETWORK_STATE))

!isGranted(Manifest.permission.ACCESS_WIFI_STATE)) {
                out.append("\n" + "[ERROR] Sin permisos");
                return;
}
out.append("\n" + "Permisos listos");
...
```

La interfaz de la aplicación es simple y resumida. Básicamente, lo único que debe hacer el usuario es presionar un botón para comenzar a recibir trabajos en segundo plano y procesar las tareas HPC. En la figura 3.25, se puede visualizar la estructura de la interfaz con su botón de inicio, notificación de eventos y verificación de permisos habilitados.

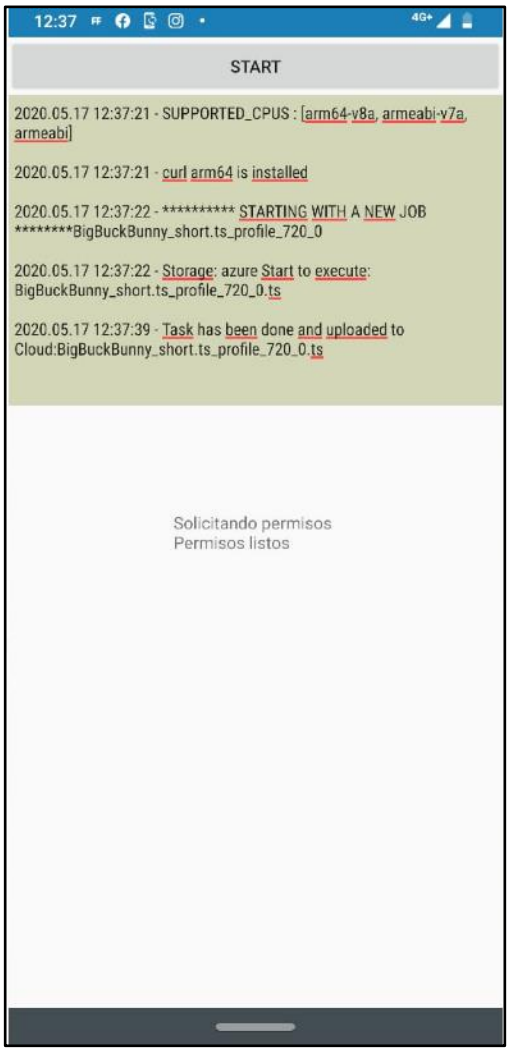

Figura 3.25 - Pantalla de inicio de la aplicación móvil HPC. Fuente: Propia.

2. Compatibilidad de la aplicación con arquitecturas de 32 y 64 bits

La aplicación fue desarrollada para que sea compatible tanto con dispositivos modernos, con chips de 64 bits, así como también con equipos más antiguos y de menores prestaciones, con cpus de 32 bits.

Para lograr dicho objetivo, por un lado, se utilizó una línea base de código que es compatible con el set de instrucciones de los dispositivos de 32 bits (armeabi-v7a, armeabi) y los de 64 bits (arm64-v8a), como se muestra en la figura 3.26.

ABI	CONJUNTO DE INSTRUCCIONES
armeabi	• ARMV5TE y posterior • Thumb-1
armeabi-v7a	• armeabi • Thumb-2 • VFPv3-D16 • Otro, opcional
arm64-v8a	• AArch-64

Figura 3.26 - Set de instrucciones básicas para cada tipo de procesador ABI ARM. Fuente: https://www.xatakandroid.com/aplicaciones-android/arm-arm64-x86-dpi-como-saber-que-apk-descargarte-para-tu-movil

Por otro lado, para incluir librerías y recursos externos adicionales a la aplicación fuente, donde a diferencia de una arquitectura tradicional se pueden instalar paquetes por medio de instaladores, repositorios o comandos por terminal; en Android, se deben seguir los siguientes pasos:

1. Obtener y descargar los binarios compilados para ambas arquitecturas (32/64 bits) y guardarlos en la carpeta “raw” del proyecto Android Studio (/raw/binaryname_32; /raw/binaryname_64).

2. Configurar la aplicación para que detecte el tipo de arquitectura. Para lograr eso, en el código se incluyen las siguientes líneas:

```
String processorInfo = Arrays.toString(Build.SUPPORTED_ABIS);
logger.logToUI(TAG, "SUPPORTED_CPUS : " + processorInfo);
```

Por ejemplo, en la imagen 3.25, en el apartado de “SUPPORTED_CPUS”, muestra que el dispositivo era compatible con 32 y 64 bits ([arm64-v8a, armeabi-v7a, armeabi]).

3. Instalar y asignar permisos de ejecución al binario correspondiente. Para ello se debe enviar el nombre y ubicación del archivo a instalar a la función “installBinary” construida. El método asignará los permisos de ejecución correspondientes y guardará el binario en un área específica del sistema de archivos de la aplicación, para que pueda funcionar y no afecte al Sistema Operativo en general. A continuación, se muestra el proceso de instalación para la herramienta CURL.

La aplicación desarrollada requiere de la herramienta CURL, al igual que la versión x86 y el resto de los microservicios construidos, para poder cargar y descargar archivos desde/hacia los sistemas de archivos. En las versiones de Android (7+) para cpus de 32 bits CURL viene integrada. Sin embargo, en las versiones para chips de 64 bits, esta herramienta no está instalada. Por lo que fue necesario incluirlo mediante el proceso antes descrito. A continuación, se ejemplifica cómo se realiza dicho proceso y se logra igualar la aplicación, tanto para 32 como 64 bits.

```
...

// [STEP 0] - Obtener información de Arquitectura ABI

String processorInfo = Arrays.toString(Build.SUPPORTED_ABIS);
logger.logToUI(TAG, "SUPPORTED_CPUS : " + processorInfo);

…

// [STEP 1] - Chequeo de arquitectura

if (processorInfo.contains("arm64")){

// [STEP 2] - Si es ARM64_v8a, instalar CURL
curlBin = installBinary(MainActivity.this, R.raw.curl, "curl", true);
}

…

// [STEP 3] - Método de instalación
 private String installBinary(Context ctx, int resId, String filename,
boolean upgrade) {
        try {
            File f = new File(ctx.getDir("bin", 0), filename);
            if (f.exists()) {
                f.delete();
            }
            copyRawFile(ctx, resId, f, "0777");
            return f.getCanonicalPath();
```

```
        } catch (Exception e) {
            out.append("\n" + "[ERROR] " + e.getMessage());
            return null;
        }
    }

...
```

3. Publicación de información estadística

Por último, es importante destacar que Android no dispone de una versión nativa que permita conectarse directamente a una base de datos MySQL/MariaDB vía JDBC. En consecuencia, fue necesario agregar al servicio Backend Web un método extra que funcione de pasamanos entre la base de datos y los nodos móviles.

Cuando el nodo de procesamiento concluye con la tarea HPC, confirma dicha actividad en el sistema de colas y almacena en el sistema de archivos el resultado, formula un objeto JSON, con toda la información estadística recopilada. Con este recurso, genera una petición HTTP GET (método: /statistics) y lo envía al Servidor Web, cómo se puede ver en la imagen 3.27.

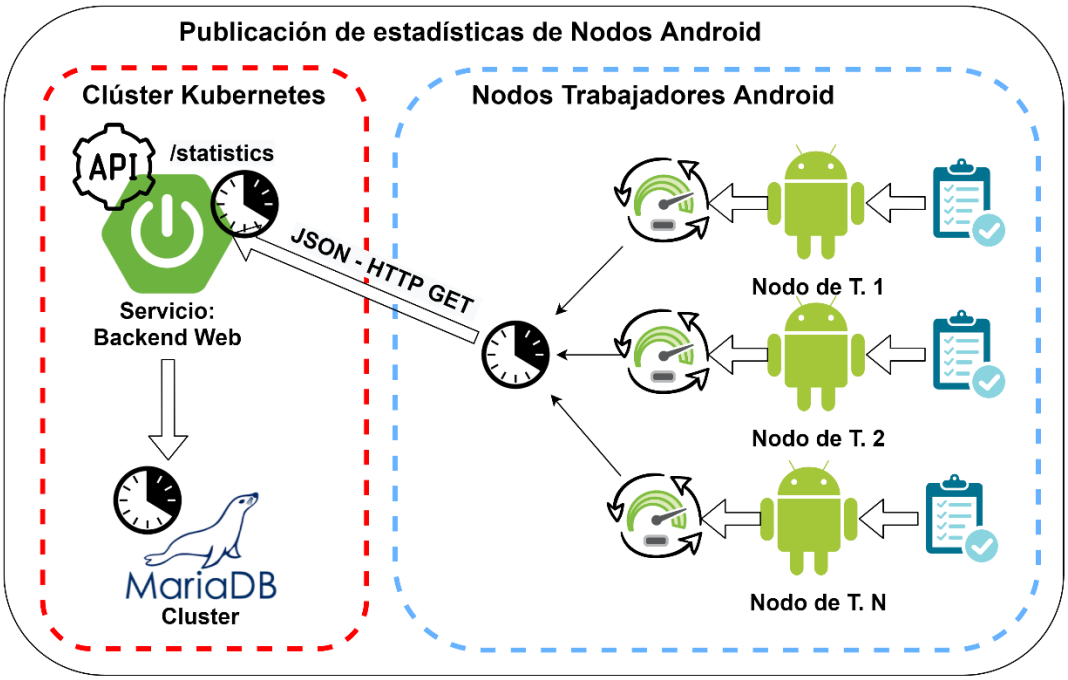

Figura 3.27 - Servicio web para manejar las peticiones de información estadística de los móviles. Fuente: Propia.

3.4 - Conclusiones del capítulo

Aprovechar de manera eficiente plataformas de computación masivamente paralelas y proporcionar niveles predecibles de servicio es un desafío excepcional para la investigación de sistemas distribuidos y paralelos. Estas plataformas, tienen en común, que agregan una gran cantidad de elementos de procesamiento para ofrecer un enorme potencial informático. Sin embargo, los problemas que dificultan la materialización total de este potencial son numerosos. Se relacionan con la minimización de los gastos generales computacionales de las aplicaciones paralelas y distribuidas, proporcionando un rendimiento predecible, eficiencia energética, usabilidad (por ejemplo, soporte de lenguaje de programación para aplicaciones paralelas de datos) o mantenibilidad (por ejemplo, capacidad para identificar y reparar problemas de manera eficiente). Por tal motivo, en este trabajo, se presenta una plataforma novedosa, modularizada, distribuida y escalable para administrar cargas de tareas HPC.

En esta arquitectura, las actividades se realizan de manera colaborativa entre los recursos de administración y los nodos de procesamiento voluntarios, realizando las tareas HPC aprovechando la capacidad ociosa de los nodos trabajadores extremos x86 y móviles.

La arquitectura garantiza, gracias al uso del orquestador Kubernetes, un uso eficiente de los recursos administrados. Al mismo tiempo, a la hora de completar las tareas involucradas, permite optimizar y flexibilizar la capacidad requerida y los costos asociados. De esta manera, se sientan las bases para que la infraestructura pueda ser utilizada y/o adaptada a las necesidades de cualquier requerimiento de procesamiento HPC.

4. Caso de aplicación: Compresión de video

El crecimiento de la audiencia en Internet y la preferencia inherente de los usuarios hacia el consumo de contenido multimedia, a través de esta plataforma, lleva a pensar que este fenómeno ocasionará que la distribución de audio y video domine el tráfico generado a través de Internet. La clave del éxito de la transmisión de video por Internet es ofrecer alta calidad, proporcionando inicios inmediatos y una reproducción sin interrupciones ni distorsiones.

Esta tecnología se está convirtiendo en un beneficio clave para cualquier negocio digital. Sin embargo, la entrega de contenido audiovisual impone grandes restricciones, a la infraestructura subyacente y a los recursos informáticos, para satisfacer las necesidades y expectativas de los usuarios finales. Este servicio, requiere codificar cada video en varios formatos para cumplir con las especificidades de los dispositivos. En la práctica, no es posible almacenar videos, en todos los formatos, del lado del servidor. Para aliviar esta carga, el streaming adaptativo a través de los protocolos HLS y MPEG-DASH [IEE18], ha ganado impulso para convertirse en el estándar de facto a la hora de ofrecer una transmisión de video. Con este servicio, los videos se dividen en una secuencia de segmentos, disponibles en una cantidad de velocidades de bits diferentes (es decir, en distintas calidades). De modo que los clientes puedan descargar automáticamente, en función de sus características y de las condiciones de su red, el siguiente fragmento adecuado para reproducir [DEV15].

Este proceso de transcodificación de video es engorroso, ya que requiere una gran cantidad de recursos del lado del servidor y, por lo tanto, una infraestructura que debe escalarse correctamente. El hecho de que los fragmentos del video a transcodificar sean independientes entre sí, permite configurar varios patrones, para distribuirlos en un conjunto de servidores.

Se han realizado muchos trabajos de investigación para proporcionar, en la medida de lo posible, los algoritmos más adecuados que permitan ofrecer diferentes estrategias de optimización, tareas de control y distribución [GOT19] [HAN19] [ARO17] [DEV15] [TUM15] [LEU09] [OZC05] en los procesos de transcodificación [YOO19][SAM19b][LIX18][LIU18][LIX18b][PEN16], para brindar un servicio de streaming de video de calidad. Se busca analizar y optimizar la gestión y consumo de CPU, memoria y de ancho de banda de la red, el tiempo de transcodificación requerido, el tamaño de los fragmentos de videos y la calidad de experiencia de los usuarios; o una combinación de todas estas propiedades.

En este contexto, independientemente del códec de recodificación elegido, es claro que la transcodificación de video en general conlleva un costo significativo en tiempo y

procesamiento. Por lo tanto, la mayoría de las soluciones existentes (ya sean académicas o de industrias como Netflix [MAD19] [ADH15] [MAR13] o Amazon Prime), aprovechan las infraestructuras distribuidas (cluster, grid, Cloud, P2P) para realizar el proceso de transcodificación de manera distribuida y paralela.

4.1 - Objetivos

El objetivo de este apartado fue integrar la transcodificación de video bajo demanda (VoD), como servicio HPC, para ejecutarse sobre la plataforma desarrollada. Esto implicó:

- Estudiar y analizar las características, casos de uso, complejidades y desafíos implicados en la recodificación de videos.
- Examinar y evaluar las herramientas de código abierto, para realizar transcodificación de videos sobre hardware x86 y ARM.
- Definir mecanismos y políticas adecuadas para integrar este servicio HPC en el ecosistema de la plataforma desarrollada.

Gracias a la incorporación del servicio de transcodificación de video como tarea HPC, se sentaron las bases para poder evaluar la performance y consumo energético de los nodos trabajadores y validar también la robustez y escalabilidad de la plataforma, resultados que se presentan en el próximo capítulo.

4.2 – El Servicio de Streaming

Cuando hablamos de vídeo a través de IP, es natural distinguir entre la entrega de vídeo en dos grandes tipos: a través de un circuito abierto o de una red cerrada. Por una red abierta se entiende a Internet, la cual es accesible de forma gratuita, donde todo el mundo puede conectarse y proporcionar sus propios videos o ver el contenido de otros y que no puede ser controlada por los participantes. La entrega de vídeo a través de esta red se realiza normalmente sin la participación del proveedor de servicios Internet (ISP), y también es conocida mediante el nombre de servicio OTT [ELA19][LAT17]

Por otro lado, cuando se habla de la entrega de contenido de vídeo a través de una red cerrada hablamos de IPTV [MAI09][YAR05], normalmente gestionada por los ISP, CDN y grandes empresas de telecomunicaciones. En la figura 4.1 se muestra un resumen de las diferencias entre IPTV y OTT. A fines prácticos, se puede decir que la diferencia técnica más importante entre la entrega de video OTT e IPTV es la capacidad de la IPTV para ofrecer garantías de servicio a través de sus redes privadas y administradas. Por otra parte, los

proveedores de servicios OTT son completamente dependientes de las características propias de Internet. La interacción con el reproductor de vídeo también suele ser algo diferente, ya que los usuarios de IPTV acostumbran a utilizar Electronic Program Guide (EPG[2]). Un servicio de video OTT se proporciona, normalmente, al usuario final a través de una interfaz web con un plug-in que permite la transmisión del vídeo.

	OTT	IPTV
Tipo de Red	Internet Pública	Red controlada
Usuarios	Generalmente utilizada en PC	Generalmente utilizada en TV
Características de la Red	Basada en mejor esfuerzo (QoS complejo)	QoS garantizada
Tipo de servicio	Flujos (streams)	Canales Broadcast
Publicacion y consumo de contenido	Sitios Web + Plugin en el reproductor	EPG
Costo del Servicio	Gratis, basada en avisos, pagas	Pagas
Acceso al servicio	Acceso abierto	Acceso restringido

Figura 4.1 – Comparación de características básicas entre OTT e IPTV. Fuente: Propia.

Actualmente, para que el servicio de streaming (OTT) pueda ponerse en marcha, existen diversos elementos y entidades que se interrelacionan para que el sistema funcione. El flujo de trabajo debe contemplar aspectos de:

- Protocolos de streaming.
- Codificación (objeto de estudio del trabajo).
- Distribución.
- Reproducción.
- Estadísticas (aspecto no obligatorio).

4.2.1 - Protocolos de Streaming

La transmisión de video permite a las personas acceder a contenidos multimedia a través de Internet [IEE18][GON17][WUD01]. Este propósito se logra a través de la transmisión de datos desde un servidor a uno o más clientes. El cliente suele comenzar la reproducción del contenido un par de segundos después de que comience a recibir el contenido desde el servidor. Difiere de una descarga de archivos normal donde todo el archivo tiene que ser descargado antes de iniciar la reproducción.

[2] EPG, Electronic Program Guide es una aplicación nativa del reproductor de video (TV, STB, etc) el cual organiza de manera rápida, sencilla y transparente al usuario final todos los canales que ofrece un distribuidor IPTV

La entrega de contenido multimedia en Internet, hoy en día, se consume a través de tres métodos generales y diferentes:

- Transmisión tradicional
- Descarga progresiva
- Streaming adaptativo.

Streaming Tradicional

El protocolo RTSP[3] permite a los usuarios interactuar con el contenido multimedia [KUR07], y es considerado como un buen ejemplo de un protocolo de flujo tradicional. Este tipo de protocolos mantienen el estado de las conexiones, lo que significa que el servidor es el encargado de disponer la información actualizada en relación con el estado del cliente, hasta que el mismo se desconecte. Esta relación puede visualizarse en la figura 4.2. Cuando se establece una conexión entre un cliente y un servidor, éste último empieza a enviar en tiempo real un flujo constante de paquetes, sobre el protocolo RTP[4], que contienen los datos de audio y vídeo. El protocolo RTP, generalmente se ejecuta sobre UDP[5]: un protocolo sin mecanismos inherentes de controles [BEG11].

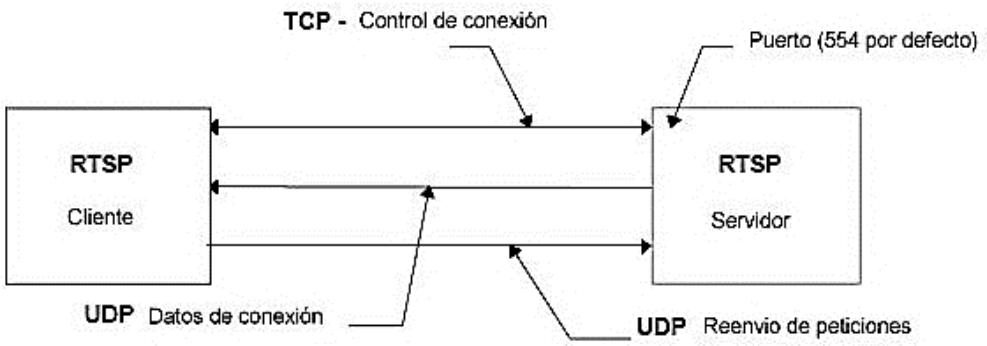

Figura 4.2 – Funcionamiento del protocolo RTSP. Fuente: https://upload.wikimedia.org/wikipedia/commons/b/b6/Udp_rtsp.jpg

Mediante mecanismos de monitoreo, el cliente envía periódicamente al servidor información de estado, utilizando el protocolo de control de RTP y/o RTCP[6] [BEG11] (control de ancho de banda, pérdida de paquetes, retraso ocurrido, buffers disponibles) dejando a éste último la capacidad de enviar los paquetes de datos a una velocidad, decidida por él (debido a que

[3] RTSP, Real Time Streaming Protocol.

[4] RTP, Real Time Protocol

[5] UDP, User Datagram Protocol

[6] RTCP, Real Time Control Protocol

habitualmente el flujo de video se encuentra codificado en diversas tasas de bits), y proporcionando una reproducción del lado del cliente de forma suave y sin pausas.

Hacia Streaming basado en HTTP

Utilizando el streaming tradicional es posible lograr la reproducción multimedia sin problemas. Sin embargo, la utilización de este tipo de protocolos conlleva un conjunto de inconvenientes. Uno de ellos, es que el uso de UDP tiene una alta probabilidad de ser bloqueado al atravesar cortafuegos y proxies en la red [ZAB18][BEG11]. Además, si un protocolo de flujo tradicional como RTSP se usa para el streaming, se requiere tener servidores especializados de aplicación de este protocolo, lo que significa más costos para los proveedores de contenido.

Por el contrario, si se utiliza HTTP como base para la distribución de medios, se tienen algunas ventajas. En primer lugar, dado que se corre sobre el protocolo TCP en la capa de transporte y los puertos 80/443, se evita cualquier tipo de problema de bloqueo de servicios [ZAM09]. En segundo lugar, como la mayoría de los nodos de red soportan HTTP no hay necesidad de implementar servidores especializados, reduciendo así también el costo implicado. Por último, se puede decir que HTTP es un protocolo sin estado, lo que conduce la lógica de trabajo a los usuarios finales, en lugar de mantenerla en los servidores. Estas características dan lugar a un sistema mucho más escalable, en comparación cuando se utiliza un protocolo tradicional [ZAM09][WAN04][OOD03].

La desventaja del uso de TCP tiene que ver con la demora que existe cuando se establece la conexión entre dos extremos, así como también, los problemas pertinentes asociados con la retransmisión de paquetes, cuando estos sufren retraso o pérdidas.

Descarga Progresiva

YouTube[7], Vimeo[8], MySpace[9] fueron los grandes actores que utilizaron en sus inicios la descarga progresiva en sus servicios ofrecidos [ZAM09]. La técnica era bastante simple; no era más que una descarga normal de archivo desde un servidor Web mediante una petición HTTP. Este mecanismo permitía al cliente reproducir el contenido mientras que la descarga de archivos se encontraba en curso [PER14]. De ahí el nombre de la descarga progresiva. Esto permitía al usuario pausar la transmisión, a la espera de que termine la descarga,

[7]Youtube: https://www.youtube.com/

[8]Vimeo: https://vimeo.com/

[9]MySpace: https://myspace.com/

logrando así una reproducción pseudo-fluida cuando el usuario decidía continuar con la reproducción, como se muestra en la figura 4.3.

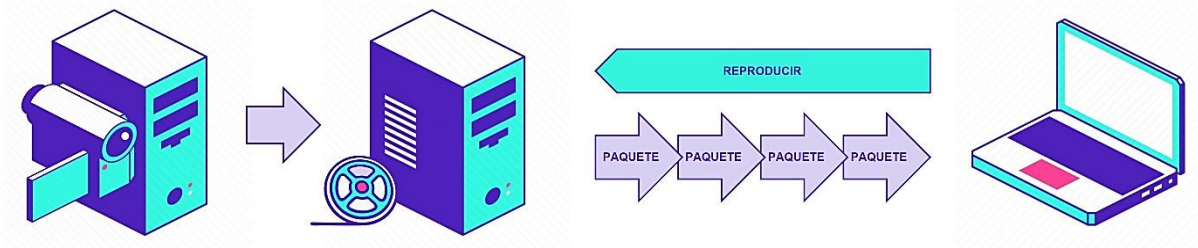

Figura 4.3 – Funcionamiento básico sobre un protocolo de descarga progresiva. Fuente: Propia.

A pesar de su popularidad, la descarga progresiva no es la técnica más adecuada. Principalmente, el problema tenía que ver con la falta de flexibilidad, debido a que antes de empezar la descarga el usuario debe seleccionar una tasa de bits para realizar la misma. Esta pauta no podía ser cambiada durante la reproducción (en caso de hacerlo, se cancelaba la descarga en curso y se reiniciaba con la nueva tasa seleccionada), y era la misma durante toda la sesión, inclusive si el ancho de banda disponible por el usuario final no era suficiente [BEG11]. Esto, como es de esperarse, puede causar una falta de información en el buffer ("underflow") que conduce finalmente a una interrupción en la reproducción de vídeo [PER14].

Streaming de Video Adaptativo (ABR) basado en HTTP

El servicio de streaming de video adaptativo es un híbrido entre el modelo de descarga progresiva y el streaming tradicional, pero basado completamente en el protocolo HTTP [PER14] [BEG11].

Para este tipo de transmisión hay varias soluciones propietarias, entre ellas se encuentran Smooth Streaming de Microsoft[10], HTTP Live Streaming de Apple[11], Adobe HTTP Dynamic Streaming[12] y MPEG-DASH[13] (como estándar de ITU). Ellos son diferentes en muchos aspectos, pero tienen en común el uso de HTTP y que el contenido requerido puede ser descrito como una larga serie de pequeñas fracciones que se descargan en forma progresiva, en lugar de una única gran descarga progresiva [SEU15][ZAM09]. En una implementación de una solución de streaming adaptativo, el video fuente se divide en segmentos diminutos, y se codifica en el formato y velocidad de entrega deseada. La longitud del segmento varía de una

[10] HSS – Smooth Streaming: http://www.iis.net/downloads/microsoft/smooth-streaming

[11] HLS – Apple Streaming: https://developer.apple.com/streaming/

[12] HDS – Adobe Streaming: http://www.adobe.com/products/hds-dynamic-streaming.html

[13] MPEG-DASH – ITU Streaming: http://dashif.org/mpeg-dash/

implementación a otra, pero los mismos son típicamente un par de segundos de duración (2 a 10 segundos). A nivel códec de vídeo, esto significa que cada segmento se corta a lo largo del vídeo sin dependencias con segmentos pasados y futuros, de manera que los segmentos pueden ser decodificados del lado del cliente de forma independiente.

Para que la transmisión entre cliente y servidor se realice, el primero envía solicitudes GET HTTP al servidor para recuperar las secciones de vídeo deseadas. Esto hace que el cliente actúe como parte esencial del sistema. El servidor solo se encarga de mantener publicados cuáles son los servicios ofrecidos y disponer los recursos disponibles y accesibles.

Más importante aún, y haciendo de esta solución de streaming más interesante, son sus propiedades adaptativas. Esta estructura entra en juego cuando el vídeo está codificado en múltiples calidades (“bitrates”). Esto permite al cliente alternar, de acuerdo a una lógica de la velocidad de adaptación, entre las diferentes tasas de bits cada vez que se solicita un nuevo segmento de vídeo. El cliente se esfuerza por alcanzar la mejor calidad de experiencia (QoE) mediante la visualización de la más alta calidad posible, proporcionando una rápida puesta en marcha, buscando reducir los saltos abruptos de calidad, imágenes congeladas y micro cortes. La lógica de la adaptación de velocidad y la decisión se basa en el seguimiento y estimación de parámetros relacionados tales como [BEG11]:

- Los recursos de red disponibles (es decir, el ancho de banda disponible).
- Las capacidades del dispositivo (es decir, la resolución de pantalla, CPU disponible, etc.).
- Las condiciones de transmisión actual (es decir, tamaño del búfer de reproducción).

Antes de poder iniciar una sesión de video streaming adaptativo, el cliente tiene que recuperar la información sobre el contenido de vídeo. La información pertinente como bitrates disponibles, duración del segmento, son de importancia para que el cliente pueda enviar solicitudes GET al servidor consultando por el segmento de vídeo que desea. Esta información se almacena en el servidor en un archivo de manifiesto (“Manifest”), cómo se resume en la figura 4.4.

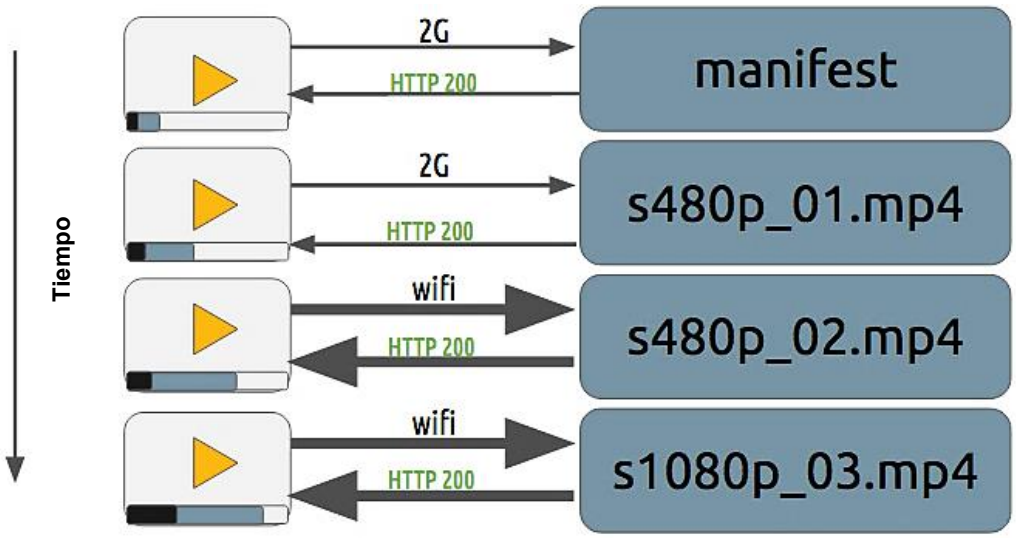

Figura 4.4 - Funcionamiento de una arquitectura de streaming ABR. Fuente: Propia.

4.2.2 - Codificación

Para comprender cómo se lleva a cabo el proceso de transcodificación de un video, es necesario, previamente, entender cómo se forma el video digital. A continuación, se explica brevemente este proceso, así como las necesidades requeridas para lograr una transcodificación de videos de manera efectiva y eficiente [IGA04] [SCH95].

Digitalización

Cuando una cámara de video captura luz, debe transformar esos datos físicos (analógicos) en datos digitales (binarios en 0 y 1). La calidad de reproducción de un sistema digital de video bien diseñado es independiente del medio y dependiente únicamente de la calidad del proceso de conversión.

Para la conversión, se deben realizar dos procesos denominados Muestreo y Cuantificación [POY96]. La luz en el mundo físico puede asumir una cantidad infinita e incontable de valores. El proceso de muestreo reduce la señal continua de luz en una cantidad finita y discreta de muestras, o Pixeles. La cuantificación asigna o “mapea” cada una de estas muestras a un conjunto finito de valores de pixel. Cabe recordar que este proceso convierte una información analógica (y por lo tanto continua) en información digital. Lo que conlleva a comprender un punto esencial de este proceso: la pérdida de información. El proceso de conversión analógico/digital se presenta, de forma esquemática, en la figura 4.5.

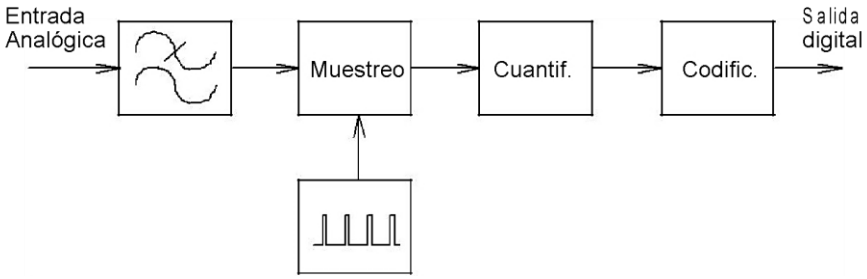

Figura 4.5 – Esquema conceptual de un conversor analógico/digital. Fuente: Propia.

En resumen, la digitalización de la señal analógica de vídeo se basa en los mismos principios de la modulación por codificación de pulsos (PCM)[14] mediante la cual, una señal analógica limitada en banda se convierte en una secuencia de señales binarias con un código específico y cuya teoría está tratada ampliamente en la literatura [CAR86] [OPP83] [SCH80]. Esta actividad puede visualizarse en la figura 4.6.

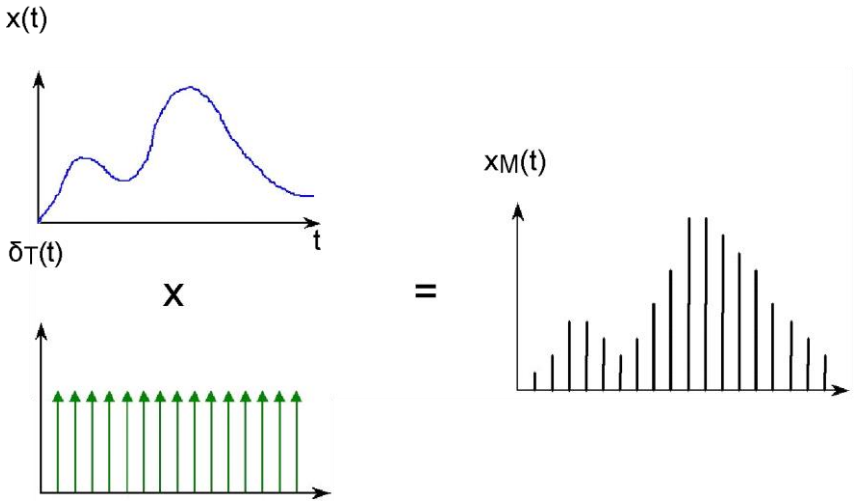

Figura 4.6 – Muestreo y cuantificación de una señal analógica para ser digitalizada. Fuente: Propia.

Proceso de Cuantificación

El proceso de cuantificación implica convertir una señal muestreada, cómo la presentada en la figura 4.6, en una secuencia de pulsos de la misma amplitud. De esa manera, es posible lograr una codificación binaria. Para ello es necesario dividir la amplitud de la señal en un número de niveles discretos. Generalmente esta división se realiza en 256 partes, por lo que cada nivel puede representarse mediante una secuencia de 8 bits (256 colores = 2^8 bits).

Para efectuar esta conversión, la señal muestreada se aplica a través de una cadena de divisores de voltaje, a una serie de comparadores, cuyo número es igual al de niveles de

[14] Se designa también como Modulación por Impulsos Codificados (MIC)

cuantificación, 256 en este caso, como se ilustra en la figura 4.7. Debido a la acción de los divisores de voltaje, tanto para la señal como para el voltaje de referencia, los valores serán coincidentes a la entrada de uno solo de los comparadores de la cadena, el cual producirá una salida “1”, en tanto que todos los restantes tendrán salida “0”. Es decir, en cada punto de muestreo, solamente uno de los comparadores entregará una señal diferente a los demás, que corresponderá al nivel de cuantificación de la señal de entrada.

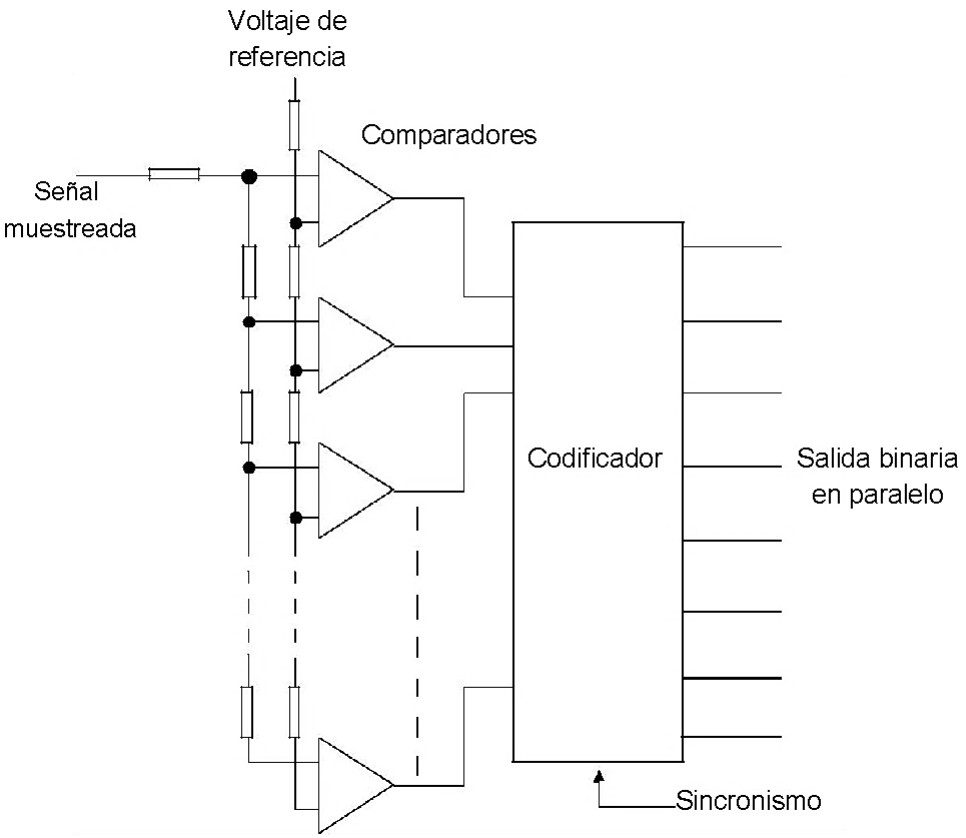

Figura 4.7 – Proceso de cuantificación y codificación. Fuente: Propia.

Como resultado de la conversión analógica-digital, la representación binaria resultante no será exacta, ya que en el proceso de cuantificación sólo se identifican niveles discretos y las amplitudes de las muestras no corresponden con exactitud a los valores de amplitud asignados a los niveles de cuantificación. Así, a cada muestra se le asignará el nivel más cercano, introduciendo con ello un error en el proceso de cuantificación, al que se designa como ruido de cuantificación, que puede ser más o menos apreciable en la reproducción de la señal. Este proceso se presenta en la figura 4.8.

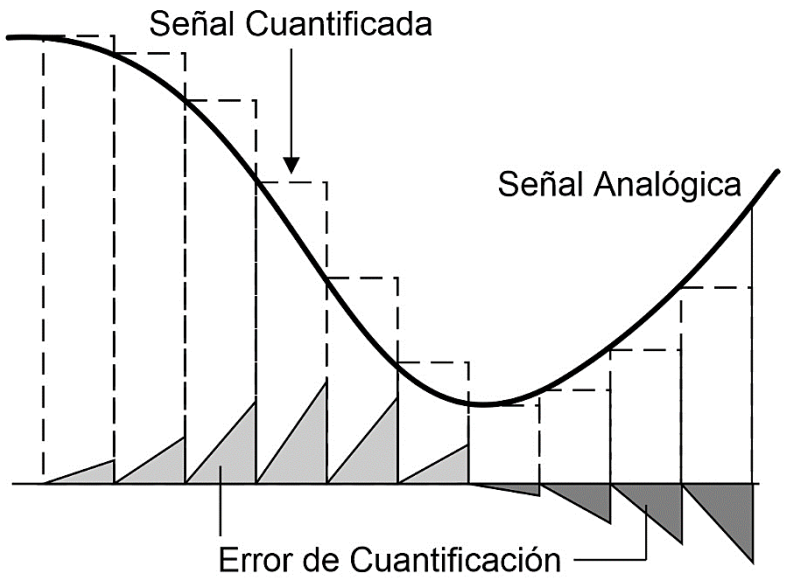

Figura 4.8 – Ejemplo de error de cuantificación producido por el proceso de digitalización de una señal analógica. Fuente: Propia. Versión adaptada de https://aiu.edu/publications/student/spanish/Comunnicacion%20de%20Systemas_clip_image003_0000.gif

Este error, depende de la cantidad de bits que serán asignados para el proceso de digitalización. Como puede preverse, a mayor cantidad de bits asignados, existe una mayor cantidad de 0 y 1 posibles para representar la señal original, y por consiguiente el error obtenido entre la señal analógica y la digital será menor. Un ejemplo de la pérdida de calidad se presenta en la figura 4.9. Tradicionalmente se utilizan 8 bits, es decir en 256 niveles, pero dependiendo de la necesidad de reducir esta degradación acumulativa se ha considerado la cuantificación de la señal con 10 bits, es decir en 1024 niveles, reduciendo así el error de cuantificación a la cuarta parte que se tiene con la asignación de bits anterior.

Imagen analógica

Imagen digital, digitalizada con 4 bits para el rojo (16 grados de intensidad)

Figura 4.9 – Comparación entre una imagen analógica y una imagen digital. Fuente: Propia

Codificación

Obtener contenido con calidad profesional en la producción de contenido multimedia digital es el principal objetivo de las emisoras de servicios de videostreaming. Aunque sería una solución “rápida y fácil” tomar el contenido en el formato “puro” proveniente de los dispositivos fuente, es imposible manejar el gran volumen de datos que generan, ya que su peso y su tasa de bits no hace posible que el mismo sea enviado a través de Internet. Sin comprimir, un único fotograma de vídeo de definición estándar utiliza casi 1 MB de almacenamiento. Con una velocidad de 60 fotogramas por segundo (estándar), el vídeo sin comprimir se reproduce a una velocidad de datos de casi 60 MB por segundo, por lo que 35 segundos de secuencia de video consumen cerca de 2 GB de almacenamiento. A partir de ello, es que cobra vital importancia la compresión [VET03].

El proceso de compresión de video requiere de un profundo conocimiento del tipo servicio que se busca ofrecer, las características de los clientes que lo consumen y el costo económico asociado a este servicio. Es importante destacar que, si esta etapa de preparación del material no es la correcta, todo que se realice posteriormente no será de alcance profesional, y la calidad de servicio y experiencia recibida por el usuario final no será satisfactoria. Por lo tanto, se debe encontrar un equilibrio entre los parámetros de calidad, tamaño del archivo y velocidad de bits, reduciendo así la información del archivo de forma selectiva [VLC20] [AZN14] [SIM14] [SON12].

Para preparar un recurso multimedia que pueda ser reproducido en un tipo de dispositivo específico, con un ancho de banda determinado, se elige un compresor/descompresor (también denominado codificador/descodificador, o códec) y un contenedor compatible con ese códec [LIM94] [PUR93] [CHE93] [GAL92] [JAI81].

Un códec (abreviatura de codificador-decodificador) es un algoritmo especial que reduce el número de bytes que ocupa un archivo de video. Los archivos codificados con un códec específico requieren el mismo códec para ser decodificados y reproducidos. No hay un único códec adecuado para todas las situaciones. Por ejemplo, el mejor códec para comprimir dibujos animados no suele ser el más adecuado para comprimir vídeos de acción. Un ejemplo simple de las actividades implicadas en este proceso se presenta en la figura 4.10.

Figura 4.10 – Lógica del proceso de encoding y decoding. Fuente: http://www.maran.com/dictionary/c/codec/image.gif

Dado que los códec son los que se utilizan dentro de los contenedores, algunos son utilizados en conjunto porque comparten una misma especificación, pero pueden utilizarse por separado o implementarse de diferente manera, si el caso lo requiriere. Existen muchos códecs diferentes y con distintas configuraciones que se pueden aplicar para obtener, dependiendo del uso y la aplicación destino, calidades resultantes diferenciadas. [APP20][LAN14][DOR13][VOS13][BAN12].

Al comprimir un material, puede optimizarse para una reproducción de mejor calidad en una computadora tradicional, en la Web o en un dispositivo móvil. Dependiendo del codificador que se utilice, se puede reducir el tamaño de los archivos comprimidos mediante la eliminación de artefactos que interfieren en la compresión [ROB04].

Las técnicas de compresión de vídeo recurren a los procedimientos generales de reducción de tamaño de datos, aprovechando además la redundancia espacial de una imagen (áreas uniformes o similares), la correlación entre puntos cercanos y la menor sensibilidad del ojo a los detalles finos de las imágenes fijas (JPEG) y para imágenes animadas (MPEG). También se saca provecho también de la redundancia temporal entre imágenes sucesivas [JOS13].

En todo material, ya sea de audio o vídeo, hay dos tipos de información almacenada en la señal [EDP18]:

- Componentes que son nuevos o impredecibles.
- Componentes que pueden anticiparse o deducirse.

Los componentes nuevos son llamados entrópicos y corresponden a la verdadera información en la señal. Los restantes son llamados redundancia ya que no son esenciales. La redundancia puede ser espacial, tal como un área plana de una imagen, donde los píxeles

cercanos tienen todos los mismos valores; o temporal, donde se explota la similitud de imágenes sucesivas.

Las técnicas de compresión se basan en una serie de algoritmos que permiten reducir la cantidad de información sin que el resultado se vea afectado (o al menos que se note lo menos posible).

Primero se decodifica la imagen en sus componentes originales, RGB (Red, Green, Blue), YUV (Luminancia, Chroma) o cualquier método de almacenamiento de vídeo digital. Tras esto se le aplican los algoritmos que permiten la compresión.

Algunos de los problemas que ocasionan estas técnicas son la pérdida apreciable de información del color, contraste/brillo erróneo (overloading) o degradación de la señal que se da por errores en los códecs teniendo como resultado cuadros corruptos. Otro de los típicos problemas es el efecto Gibbs, que se da en recuadros con alto contraste o cambio brusco de color o textura. Se da por el error relativo en la transformación trigonométrica de la información de color/luminancia. Pero el fallo más característico y apreciable es el "Blockiness", haciendo que los cuadros o regiones en que se ha dividido la imagen aparezcan perfectamente visibles, debido a la gran diferencia de calidad con respecto a la imagen original. En ese caso, se debe al bajo bitrate asignado para esa secuencia.

Compensación de movimiento y tipos de cuadros (Frames)

La compensación de movimiento es una técnica utilizada en la codificación de vídeo, cuyo principal objetivo es eliminar la redundancia temporal, existente entre las imágenes que componen una secuencia [WAT13], con el fin de aumentar la compresión. Para ello se requiere:

- Un algoritmo que examina la sucesión de fotogramas.
- Detección, análisis y estimación de movimiento entre fotogramas.
- Selección de zona de codificación (sólo partes en movimiento).
- Predicción, es decir, compensación de movimiento basándose en las imágenes anteriores.

A continuación, se explican los tipos de cuadros (frames) [WAT13] y su comportamiento sobre la base de un ejemplo. Si se considera como caso de análisis una película a 30 cuadros por segundo, y se cortan los primeros 4 fotogramas, un esquema posible es el presentado en la figura 4.11.

Figura 4.11 - Fotogramas iniciales (4) de un video. Fuente: Propia.

I Frame (intra, keyframe)

Un fotograma I [ZAT10] (referencia, fotograma clave, intra) es autónomo, es decir que no depende de nada para su renderización. Un I frame se parece a una foto, que es estática. El primer fotograma de un video suele ser un fotograma I y es un frame intra codificado, siendo una imagen completamente construida (se codifica independientemente). También son conocidos como Keyframes (parámetros de codificación). A continuación, se presenta un ejemplo a través de la figura 4.12.

Figura 4.12 - Tipo de cuadro I. Fuente: Propia.

P Frame (predicted)

Un fotograma P aprovecha el hecho de que, casi siempre, la imagen actual se puede renderizar utilizando el fotograma anterior [EDP18]. Es decir, la imagen solo registra los cambios respecto de la imagen previa. Su referencia puede ser a cuadros I y P anteriores. El tamaño promedio de un frame P es de, aproximadamente un ⅓ del I frame. Por ejemplo, en el segundo cuadro, el único cambio fue la bola que avanzó. De esta manera, se puede reconstruir el cuadro 1, solo usando la diferencia y haciendo referencia al cuadro anterior, cómo se puede observar en la figura 4.13.

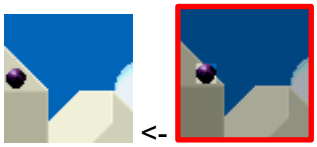

Figura 4.13 - Tipo de cuadro P. Fuente: Propia.

B Frame (bi-predictive)

Este tipo de *frame* utiliza cuadros anteriores y posteriores para alto nivel de compresión [MAK18], cómo se muestra en la figura 4.14. Los perfiles de compresión de alto nivel utilizan algoritmos de codificación más complejos y producen una mejor calidad de vídeo [PAT15], a

una velocidad de transferencia menor. Al mismo tiempo, esto implica la necesidad de un hardware más potente para lograr tanto la codificación como la decodificación. El tamaño final es de, aproximadamente una sexta parte de un cuadro de tipo I.

Figura 4.14 - Tipo de cuadro B. Fuente: Propia.

GOP (Group of Pictures)

el GOP es un conjunto de cuadros (frames) consecutivos que pueden ser decodificados sin referencia a otro frame [HUS10] (por lo general 8, 12 o 15 frames). Habitualmente se comienza con un I frame. La estructura GOP suele estar referenciada por dos números, por ejemplo, M=3, N=12. El primero de ellos nos dice la distancia que hay entre dos imágenes tipo I o P [HUS10]. El segundo menciona la distancia que hay entre dos imágenes enteras, es decir, entre dos imágenes tipo I. Este parámetro es conocido como la longitud del GOP.

Su estructura típica es I-B-B-P-B-B-P. El I frame se utiliza para predecir la primera trama P, y tanto los cuadros I y los cuadros P se utilizan para predecir la primera y la segunda trama B. El segundo fotograma P se predice usando la primera P, y se unen para predecir los fotogramas B tercero y cuarto. Siempre un cuadro I (completo) arranca la secuencia GOP [EDP18]. Cuantas más imágenes tipo I haya en un stream de vídeo más fácil será su edición, pero en contraposición ocupará más espacio. Para ahorrar ancho de banda y espacio en el disco, los vídeos preparados para difusión en Internet solo tienen una imagen tipo I por GOP.

Tipos de Códecs:

Dentro de los distintos sistemas que utilizan los códecs para comprimir el vídeo, podemos distinguir básicamente dos tipos [CAS13][ROD10]:

- **Códecs sin pérdida**, es decir, los que conservan los datos originales y aseguran que las imágenes sean las mismas después de la compresión y posterior descompresión. En estos sistemas, se intenta que el codificador extraiga la redundancia de la señal y envíe solo la entropía al decodificador. Sin embargo, las técnicas de compresión sin perdida no son, en general, muy efectivas con el vídeo digital, ya que éste tiene pocas áreas de color continuo y está formado por numerosas variaciones de color.
- **Códecs con pérdida:** Estos intentan eliminar información de las imágenes de forma que sea lo más inapreciable posible para el espectador. Se intenta eliminar la

información irrelevante, o no tan crítica, para el observador antes de analizar los componentes importantes en la señal. Solo la entropía (lo que se considera la verdadera información) es almacenada o transmitida, y el decodificador calcula la redundancia con la señal recibida. Esa información eliminada no puede ser recuperada. La cantidad de información perdida depende del grado de compresión y es proporcional a la disminución de calidad.

A su vez, dentro de los algoritmos de compresión con pérdida podemos distinguir dos tipos básicos:

- **Algoritmos de compresión especial:** comprime cada imagen (fotograma) independiente del vídeo.
- **Algoritmos de compresión temporal:** aprovechan la ventaja que existe cuando las imágenes sucesivas son similares.

La compresión de video digital hace uso de la redundancia de datos en dos dominios: el dominio espacial, conocida como intra-codificación (Intracoding) y el dominio temporal, conocida como inter-codificación (Intercoding) [LIB12][OHM12][ZHA00][GER93], cómo se muestra en la figura 4.15.

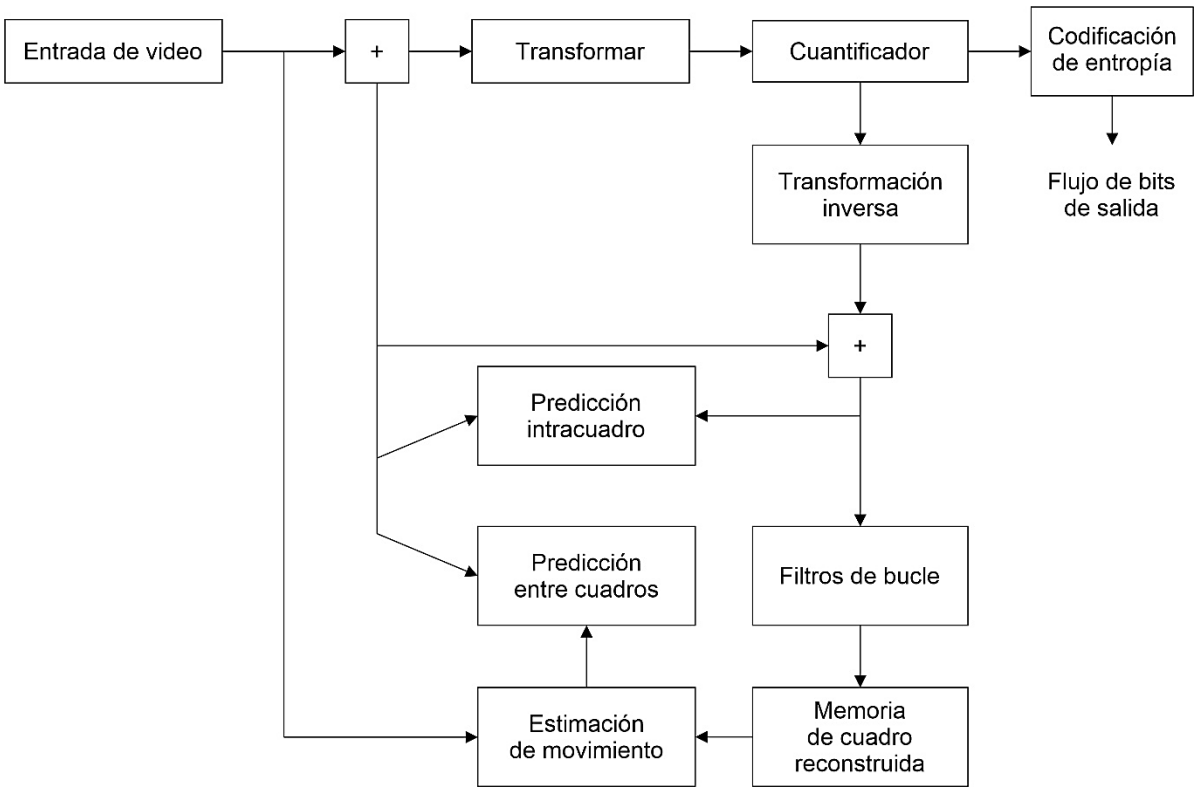

Figura 4.15 - Estructura de un Encoder de video. Fuente: Propia. Versión adaptada de https://github.com/leandromoreira/digital_video_introduction/raw/master/i/thor_codec_block_diagram.png

Intra-Codificación, o Codificación dentro del cuadro (Intra-Frame)

En este caso, cada imagen es tratada como instantánea, y codificada independientemente del contexto. Al contener la información completa, es el tipo de imagen elegida como primera en una secuencia de Inter-Codificación. El proceso de Intra-Codificación involucra el uso de transformaciones tales como la Transformada Discreta del Coseno (DCT por sus siglas en inglés) [AHM74], así como también, la cuantificación de los coeficientes resultantes de la transformación.

A medida que la codificación espacial trata cada imagen independientemente, esta puede emplear ciertas técnicas de compresión desarrolladas para las imágenes fijas.

Si se analiza cada fotograma de un video, se puede visualizar que también existen muchas áreas que están correlacionadas, cómo se puede ver en la figura 4.16.

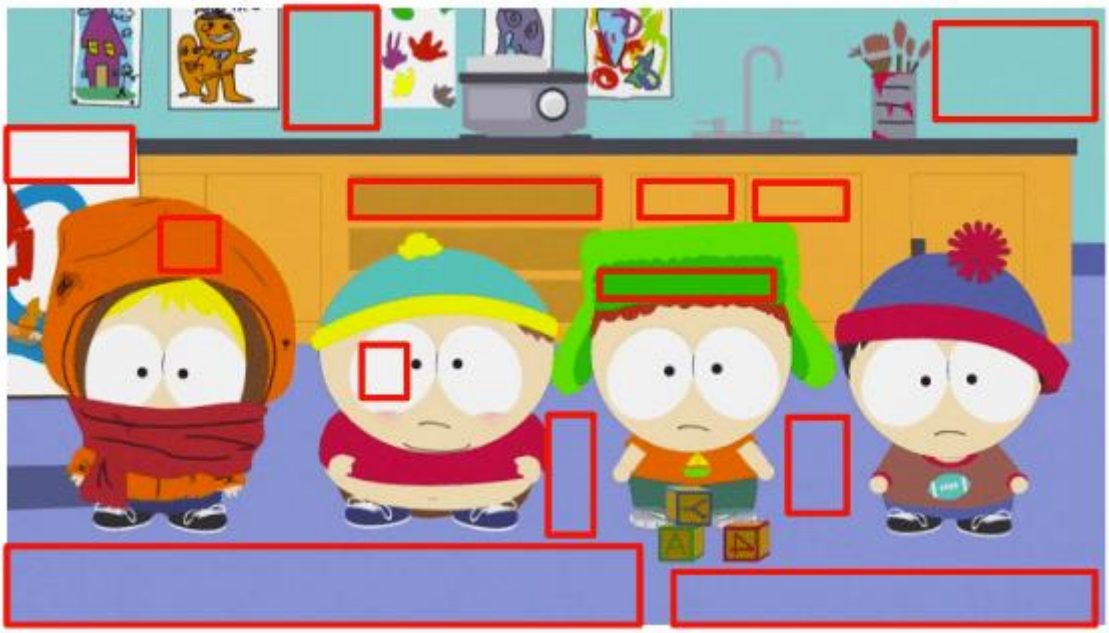

Figura 4.16 - Elementos similares dentro de una misma imagen. Fuente: Propia.

La compresión espacial se vale de las similitudes entre píxeles adyacentes en zonas de la imagen lisas, y de las frecuencias espaciales dominantes en zonas de color muy variado. El proceso de cuantización es la parte del algoritmo que causa pérdidas. La cuantización asigna un número de bits específico a cada coeficiente de frecuencias, y entonces comprime los datos asignando unos cuantos bits a los coeficientes de alta frecuencia. sin que lo note el observador.

Inter-Codificación, o Codificación entre cuadros (Inter-Frame)

Cuando hay dos imágenes consecutivas de un video, se asume que en ellas existe una correlación [EBR02]. En lugar de codificar una imagen completamente nueva, la Inter-Codificación hace uso de esta redundancia temporal, formando una predicción de la siguiente imagen, basada en la que ya ha sido codificada [NAI18].

MPEG es el principal ejemplo de este tipo de compresores, y se caracteriza porque puede mantener una alta calidad en la imagen, a pesar de utilizar ratios de compresión mayores que la compresión intra. Cada cierta cantidad de fotogramas, el compresor retiene la información de la secuencia, lo que servirá de referencia a la hora de saber qué datos debe desechar en las secuencias posteriores, hasta encontrar un nuevo fotograma de referencia [PAT15]

Si se define un cuadro inicial (con toda la información almacenado en dicho cuadro) como se presenta en la figura 4.17 y considerando que los frames subsiguientes son los que se muestran en la figura 4.18, sucede que, al aplicar una compresión de video con las características previamente definidas, los cuadros Inter-Frame resultantes serían los presentados en la figura 4.19.

Figura 4.17 - Cuadro inicial de la composición del video. Fuente: https://www.animemusicvideos.org/guides/avtech/images/guide_inter_start.jpg

Figura 4.18 - Cuadros subsecuentes a la imagen de la figura 4.17. Fuente: https://www.animemusicvideos.org/guides/avtech/images/guide_inter_3a.jpg

Figura 4.19: Compresión Inter-Frame. Fuente: http://www.animemusicvideos.org/guides/avtech/video3.htm

Es importante conocer los diferentes tipos de códecs que existen y las principales características de cada uno de ellos para elegir el más adecuado a las necesidades de compresión actuales. Siempre habrá que tener en cuenta la compatibilidad entre las distintas plataformas y sus ratios de compresión.

Es preciso saber dónde va a ser decodificado el archivo, y de qué tipos de recursos dispone el potencial destinatario de la película digitalizada, ya que muchos de los compresores permiten indicar la tasa de transferencia de datos para optimizar la compresión, en función del medio desde el cual se reproducirá la película.

MPEG cómo estándar de Compresión

MPEG (Moving Picture Experts Group), es el nombre de una familia de estándares utilizados para codificar información audiovisual (por ejemplo, películas, videos, música) en un formato digital comprimido. Su evolución se muestra en la figura 4.20.

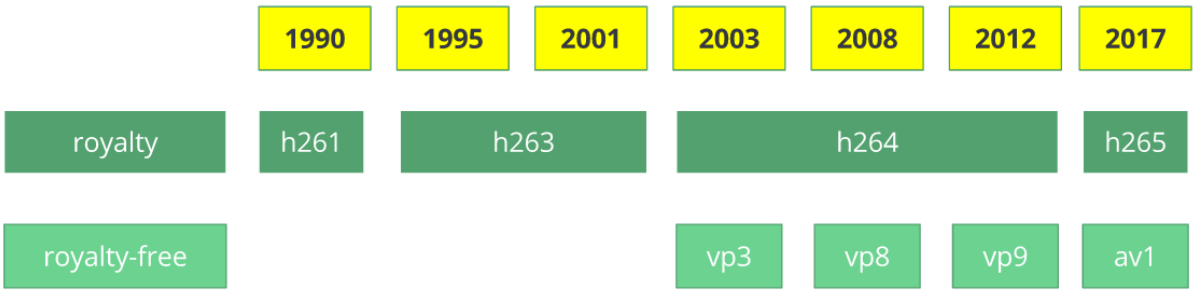

Figura 4.20 - Evolución de los formatos MPEG. Fuente:
https://github.com/leandromoreira/digital_video_introduction/raw/master/i/codec_history_timeline.png

H.264 AVC

H.264 o MPEG-4 parte 10 es una norma que define un códec de vídeo de alta compresión, desarrollada conjuntamente por el ITU-T Video Coding Experts Group (VCEG) y el ISO/IEC Moving Picture Experts Group (MPEG) [CHE06][SUL05]. La intención del proyecto

H.264/AVC fue la de crear un estándar capaz de proporcionar una buena calidad de imagen con tasas binarias notablemente inferiores a los estándares previos (MPEG-2, H.263 o MPEG-4 parte 2), además de no incrementar la complejidad de su diseño [RIC11].

A continuación, se destacan las principales características de este estándar en relación con los de codificación de video anteriores.

- En términos de compensación de movimiento, H.264 admite tamaños de bloque más flexibles (4 x 4 - 16 x 16), y una compensación de movimiento precisa de un cuarto de muestra.
- Se diferencia de sus predecesores por el uso de una gran cantidad de tramas decodificadas previamente, para predecir los valores de un cuadro entrante [RIC11].
- Las operaciones de transformación y cuantificación se mejoran. Permite reducir el procesamiento de los cálculos, mientras se mantiene la precisión de la transformación y se eliminan los datos sin importancia, para lograr una mayor eficiencia de compresión [RIC11].
- Otra característica importante es un filtro de desbloqueo, que puede mejorar la calidad visual, al reducir los artefactos de bloqueo.
- H.264 AVC incluye dos métodos de codificación de entropía, llamados CAVLC (codificación de longitud variable adaptativa al contexto) y CABAC (codificación aritmética binaria adaptativa al contexto), como una actualización de VLC (códigos de longitud variable) para mapear niveles de coeficientes transformados, y mejorar la eficiencia de la codificación.

H.265 HEVC

Por su parte, H.265 o MPEG-H Parte2, llamado de forma común como High Efficiency Video Coding (HEVC), es el sucesor de H.264/MPEG-4 AVC (Advanced Video Coding), desarrollado conjuntamente por la ISO/IEC Moving Picture Experts Group (MPEG). [SHA13].

En términos de eficiencia de compresión de video, H.265 puede lograr aproximadamente un 50% de ahorro en la tasa de bits, en relación con los estándares anteriores, especialmente para video de alta resolución [MAL19][UHR14].

Además de los estándares anteriores, H.265 también se basa en el enfoque híbrido de los procesos de codificación de predicción inter/intra, con una estructura redefinida de bloques. El macrobloque se renombra en una denominada Unidad de árbol de codificación (CTU), cuyo tamaño puede ser mayor que el macrobloque tradicional. La CTU consta de un bloque de árbol de codificación de luma (CTB) y los CTB de croma, donde el tamaño es de 16, 32 o 64

muestras cuadradas [FLY16]. Los CTB se dividen en bloques de codificación con un tamaño mínimo de 4 x 4 muestras. Los bloques de tamaño variable brindan más flexibilidad para adaptarse al contenido de la escena de video: los más pequeños se utilizan en áreas con mayor detalle, mientras que los más grandes se pueden usar en áreas con menos detalles (por ejemplo, cielo) [KAW15]. Las principales diferencias y características entre estos dos códecs se presenta a continuación en la figura 4.21.

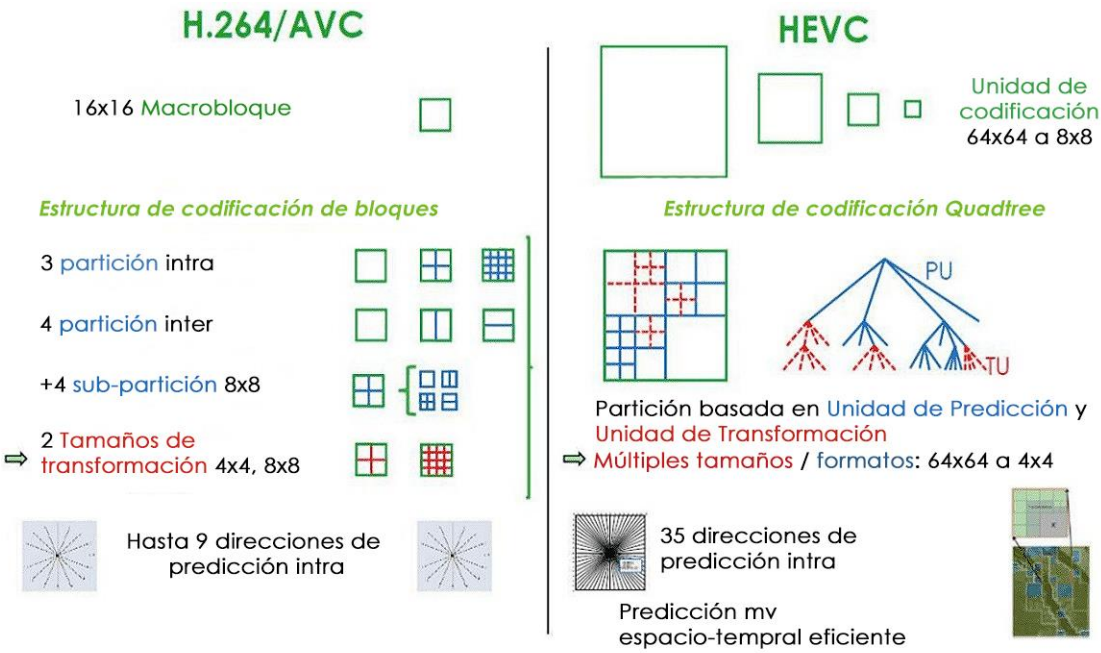

Figura 4.21 - Estructuras de los códec H.264 y HEVC. Fuente: Propia. Versión adaptada de https://github.com/leandromoreira/digital_video_introduction/raw/master/i/avc_vs_hevc.png

VP9

Debido a la necesidad de contar con códecs de vídeo de código abierto, Google desarrolló VP9 (sucesor de VP8 [UHR16]) como parte del proyecto WebM, al que cualquiera puede contribuir libremente. Este códec es una alternativa y competencia a los códecs de MPEG H.264 y H.265 [UHL18][KUF16][KUF15].

Entre las principales características del códec VP9 [MUK13] se encuentran los llamados Superbloques (SB) con un tamaño máximo de 64 x 64 bloques, que se pueden dividir en partes de 4 x 4 en 13 tamaños de punto final diferentes. El códec admite 10 modos de predicción intra y 4 modos de predicción inter para calcular los vectores de movimiento [MUK15]. Cada marco puede tener hasta tres búferes de parámetro seleccionados. Se pueden usar marcos de referencia alternativos, que nunca se muestran, para mejorar la eficiencia de la codificación. El codificador puede elegir la precisión entre un cuarto de

muestra o un octavo de muestra. Similar a H.264 y H.265, el filtro de bucle se utiliza para eliminar artefactos de bloqueo [MUK15].

Configuraciones asociadas para recodificar videos

Una vez que se comprende la estructura de un video, cómo se compone, cuáles son sus características básicas y que se toma un códec estándar y compatible con los recursos de reproducción objetivo para realizar la recodificación del material audiovisual, es necesario entender cómo se debe configurar y parametrizar el transcoding utilizando códec. Para ello, es necesario entender qué son y cómo afectan al proceso de codificación los siguientes conceptos:

Formatos de vídeo (Contenedor)

Los videos digitales se pueden guardar en archivos de distintos formatos. Cada uno se corresponde con una extensión específica del archivo que lo contiene. Cuando se habla de vídeo nos estamos refiriendo al formato contenedor multimedia que en su interior alberga cadenas de datos relativas al audio, vídeo y, en algunos casos. hasta subtítulos o informaciones complementarias. El contenedor define el modo en el que se guardan los datos de un archivo de vídeo con el fin de que puedan ser interpretados por el dispositivo en cuestión. En la figura 4.22 se presenta un resumen de las características contenidas en un formato contenedor.

Figura 4.22. Datos almacenados en un contenedor de video. Fuente: http://i.emezeta.com/weblog/formatos-de-video/formato-contenedor.png

Por ejemplo, un fichero en formato MP4 es un contenedor que en su interior almacena una parte de información dedicada al vídeo, que podría estar comprimido con el códec H.264; y

otra parte dedicada al audio, que puede ser codificado con MP3, AAC o AC3. Juntos se combinan en un solo archivo contenedor con extensión .mp4.

Modelos de color

El modelo RGB (del inglés red, green, blue: "rojo, verde, azul") de un color hace referencia a la composición del color en términos de la intensidad de los colores primarios luz con que se forma un espacio de un video. Este modelo se basa en la propiedad de síntesis aditiva, a través del cual es posible representar un color mediante la mezcla por adición de esos tres colores primarios. Para indicar con qué proporción se mezcla cada color, se asigna un valor, por ejemplo, el 0 expresa que no interviene en la mezcla y, a medida que ese valor aumenta, aporta más intensidad. La intensidad de cada una de las componentes se mide según una escala que va del 0 al 255. Por lo tanto, el rojo se obtiene con (255,0,0), el verde con (0,255,0) y el azul con (0,0,255), produciendo, en cada caso un color resultante monocromático. En la figura 4.23 se presenta un ejemplo de la composición del color RGB. La ausencia de color —lo que nosotros conocemos como color negro— se obtiene cuando las tres componentes son 0, (0,0,0). La combinación de dos colores a nivel 255 con un tercero en nivel 0 da lugar a tres colores intermedios: el amarillo (255,255,0), el cyan (0,255,255) y el magenta (255,0,255). Obviamente, el blanco se forma con los tres colores primarios a su máximo nivel (255,255,255).

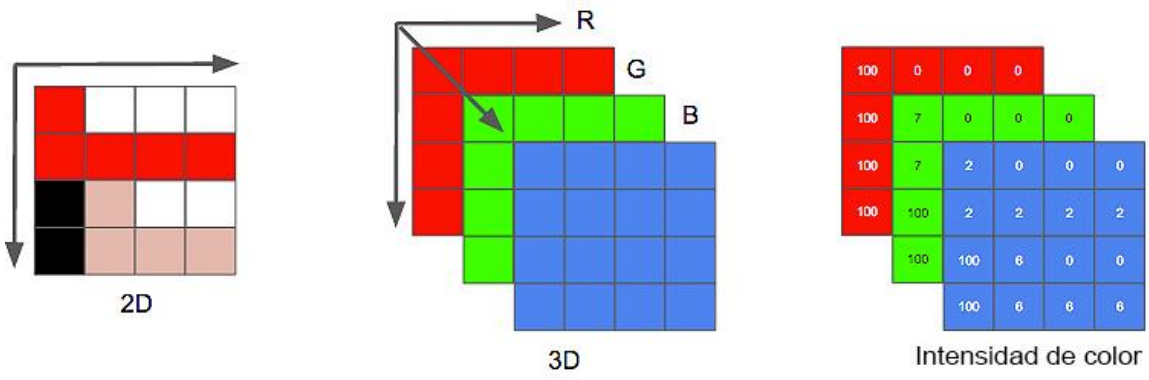

Figura 4.23 - composición de color RGB. Fuente: Propia. Versión adaptada de https://github.com/leandromoreira/digital_video_introduction

Cuadros

Un video es representado como una secuencia ordenada de cuadros. Cada cuadro consiste en un conjunto de franjas horizontales de la pantalla. Existen dos tipos de cuadros, el progresivo y el entrelazado.

En el video progresivo, cada imagen es representada completamente por un cuadro. Los fotogramas de vídeo no se separan en campos. Un monitor de búsqueda progresiva muestra un fotograma de vídeo mediante el dibujo de todas las líneas horizontales, de arriba a abajo, en un pase. Por ello, los dos campos que conforman un fotograma de vídeo se muestran de forma simultánea.

Por otro lado, en el Entrelazado, cada imagen es representada por dos campos separados. Cada uno es un conjunto de líneas alternadas, como se muestra en la figura 4.24. El campo superior (o Campo 1) contiene todas las líneas con numeración impar y el campo inferior (o Campo 2) incluye todas las líneas con numeración par. En este tipo de formato, solo un campo es actualizado en cada ciclo de muestreo, por lo que es posible mostrar video al doble de frecuencia temporal con respecto al video progresivo, logrando una sensación de mayor fluidez.

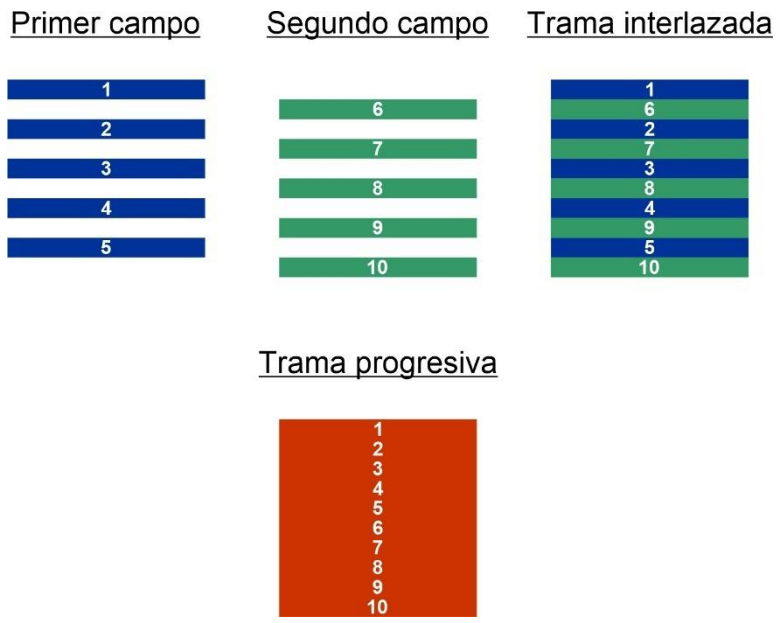

Figura 4.24 – Video Entrelazado y Progresivo. Fuente: Propia.

La mayor parte del vídeo de difusión es entrelazado, aunque los nuevos estándares de televisión de alta definición tienen variantes entrelazadas y no entrelazadas.

Velocidad de cuadros

El número de fotogramas que aparece cada segundo se denomina velocidad de cuadros y se mide en fotogramas por segundo (fps). Cuanto más elevada sea esa velocidad, más fotogramas por segundo se utilizan para presentar la secuencia de imágenes, lo que produce un movimiento más suave. No obstante, el equilibrio necesario para obtener una mayor calidad reside en que las velocidades de fotogramas más elevadas requieren una mayor

cantidad de datos para mostrar el vídeo, con lo que se utiliza más ancho de banda y aumenta el tamaño del archivo.

Debido a que el aspecto del vídeo es mucho mejor con velocidades de fotogramas nativas (velocidad con la que el vídeo se filmó en un principio), se recomienda dejar una velocidad alta si los canales de distribución y las plataformas de reproducción lo permiten. Para NTSC de movimiento total (estándar definido por la Comisión Nacional de Sistemas de Televisión, National Television System Committee, en EE. UU), se debe utilizar 29,97 fps; para el formato PAL (Phase Alternating Line) (Línea con alternancias de fase, el estándar de televisión dominante en Europa), se utilizan 25 fps. Si disminuye la velocidad de fotogramas (lo cual puede reducir considerablemente los datos de vídeo que se deben codificar), se eliminan fotogramas con una velocidad lineal para obtener la nueva velocidad en fps. No obstante, si se reduce la velocidad de los cuadros, los mejores resultados se obtienen de una división equitativa. Por ejemplo, si el origen tiene una velocidad de 24fps, es recomendable reducir la velocidad de los fotogramas a 12 fps, 8 fps, 6 fps, 4 fps, 3 fps o 2 fps. Si la velocidad de los fotogramas de origen es de 30 fps, en la mayoría de los casos puede ajustarla a 15 fps, 10 fps, 6 fps, etc.

Fotogramas clave

Los fotogramas clave (keyframes), cómo ya se describió previamente, son imágenes de vídeo completas que se insertan en intervalos constantes en un clip de vídeo, y que introducen un cambio en la animación. Por ejemplo, si un vídeo muestra a una persona entrando por una puerta, los fotogramas clave contendrán la imagen completa de la persona y la puerta en el fondo, mientras que los de intervalo incluirán la información que describa el movimiento de la persona conforme camina delante de la puerta.

El valor del intervalo del fotograma clave, transmite al codificador la frecuencia con la que se debe volver a evaluar la imagen de vídeo y grabar un fotograma completo, o fotograma clave, en un archivo.

En general, el valor predeterminado del intervalo de keyframe proporciona un nivel de control razonable cuando se realizan búsquedas en un clip de vídeo. Si se selecciona un valor personalizado, se debe tener en cuenta que cuanto menor sea el intervalo entre fotogramas clave, mayor será el tamaño del archivo. Si la secuencia cuenta con muchos cambios de escena o animaciones y movimientos rápidos, la calidad global de la imagen puede beneficiarse de un intervalo menor entre fotogramas clave.

Los keyframes sirven como puntos de referencia en la reproducción de vídeo. Si se realiza una captura de larga duración (toda una película, por ejemplo) y no se usan cuadros clave, y se quiere empezar a ver la captura desde un punto concreto, el equipo tendrá que leer el archivo de vídeo desde el principio, cuadro a cuadro, hasta llegar al punto seleccionado, empleando un tiempo considerable. Muchos códecs de captura permiten elegir cada cuantos fotogramas se quiere un “keyframe”. Normalmente se suele usar un cuadro clave cada 15-50 fotogramas.

Velocidad de bits

La velocidad de bits (también denominada velocidad de datos) afecta a la calidad de un clip de vídeo y a la audiencia que puede descargar el archivo, considerando sus limitaciones de ancho de banda.

Para publicar vídeos a través de Internet, se deben crear archivos a velocidades de bits bajas. En ese caso podrá alcanzarse a todos los usuarios disponibles en la web. El bitrate variable consigue mayor calidad de imagen porque recoge más calidad en escenas muy cargadas o con mucho movimiento y ahorra en aquellas más estáticas.

Independientemente del códec utilizado, se pueden utilizar dos métodos distintos de codificación: velocidad de bits constante (CBR) y velocidad máxima de bits variables (VBR máxima restringida). Cada uno de estos métodos es óptimo para escenarios de reproducción o transmisión por secuencias específicas.

La codificación CBR está diseñada para trabajar de forma óptima en diversos escenarios de transmisión por secuencias. Se puede restringir la velocidad de bits para garantizar una reproducción coherente en una amplia variedad de sistemas. La velocidad de bits es bastante constante y próxima a la velocidad de bits deseada en el transcurso de la secuencia. La desventaja de la codificación CBR es que la calidad del contenido codificado no es constante. Puesto que algunas partes del contenido son más difíciles de comprimir que otras, algunas secciones de una secuencia CBR son de menor calidad que otras.

La codificación VBR está diseñada para trabajar de forma óptima en escenarios con un ancho de banda alto, y resulta especialmente adecuada para codificar contenido con datos simples y complejos. El codificador asigna menos bits a las partes simples y deja suficientes bits disponibles para producir una calidad adecuada para las partes más complejas. Cuando se usa en un video de complejidad variable, la codificación VBR genera una salida mucho mejor que la codificación CBR, incluso si ambos métodos producen archivos de tamaño idéntico.

Proporciones de aspecto (tamaño del fotograma)

Al igual que sucede con la velocidad de fotogramas, la proporción de aspecto (o el tamaño del fotograma) es importante para obtener vídeo de alta calidad. A una determinada velocidad de bits, al aumentar el tamaño del fotograma disminuye la calidad de vídeo. Cuando se selecciona el tamaño de fotograma para el material multimedia, es importante tener en cuenta la proporción de aspecto del clip de vídeo de origen y sus preferencias, para crear una presentación de vídeo correcta. La proporción de aspectos más comunes son 16:9, 4:3 y 2:1.

Generalmente el vídeo se debe codificar utilizando la misma proporción de aspecto con la que se capturó. En la modificación de proporción de aspecto de un clip de vídeo pueden aparecer barras negras (o máscaras) en los laterales o en las partes superior e inferior de la imagen.

Dimensiones

Es el tamaño del vídeo (ancho x alto) expresado en píxeles, cuando se visualiza al 100%. Los reproductores pueden mostrar un video a pantalla completa o con una ampliación del 200%, 300%, etc. En estos casos el video pierde calidad de imagen. Esta pérdida depende del formato de archivo.

Por ejemplo, si el vídeo usa una proporción de aspecto de 4:3, la ecuación aparecería del siguiente modo:

$$\text{alto} = \text{ancho} \times \frac{4}{3}$$

Por ejemplo, si va se va a codificar un vídeo cuyo tamaño de fotograma es de 640 x 480 píxeles y se desea obtener una proporción de aspecto de 4:3, en primer lugar, se debe determinar el ancho con la que se desea construir el fotograma de vídeo en píxeles:

$$640 = 480 \times \frac{4}{3}$$

Cómo consecuencia, el resultado es una altura de imagen de vídeo de 640 píxeles.

4.3 - Compresión de vídeo utilizando FFmpeg

Como se explicó en los apartados anteriores, la transmisión de un servicio de streaming adaptativo basa su funcionamiento en la codificación de un mismo flujo de video en distintos bitrates o calidades. El objetivo a través de la codificación múltiple es, entonces:

- Alcanzar distintos dispositivos y plataformas:
    - Smartphones y Tablets: Android, IOS, Blackberry, Windows Phone, etc.
    - MicroPCs, PCs, Notebooks: Windows, Linux, OsX, etc.
    - Smart TVs: Android TV, Apple, Roku, etc.
- Adaptar la transmisión de acuerdo a las características disponibles de cada uno de los participantes:
    - Velocidad de conexión.
    - Tamaño de pantalla.
    - Capacidad de procesamiento (CPU, memoria).

Las principales resoluciones de pantalla actuales se presentan a continuación en la figura 4.25.

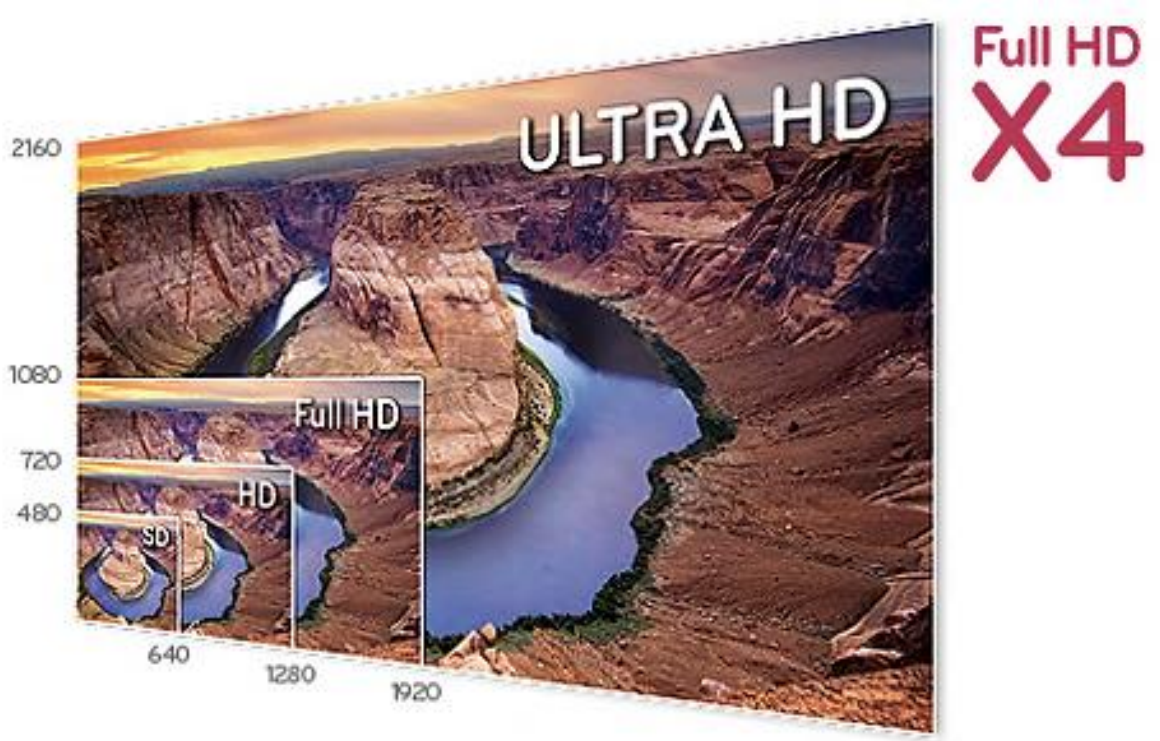

Figura 4.25 – Resoluciones estándar para la transmisión de video sobre DVB. Fuente: http://cdn.64bitz.com/wp-content/uploads/2014/10/4k-vs-full-hd-vs-hd-ready-650x365.jpg

Por lo tanto, las plataformas de streaming deben implementar mecanismos de recodificación eficientes que permitan llegar a los dispositivos de los usuarios, con la mayor calidad de imagen y sonido. Con este fin, se requieren fuentes de contenido adecuadas para transmitir el flujo de video, a través de Internet, en calidades muy diversas. Esto implica producir fuentes

de 8K y 4K para equipos y conexiones de última tecnología (8K cuenta con el equivalente a 8 veces la cantidad de píxeles activos de una señal Full HD); y perfiles mucho más reducidos: de 240p o 360p para conexiones lentas y/o inestables, corriendo sobre dispositivos más limitados, cómo se muestra en la figura 4.26.

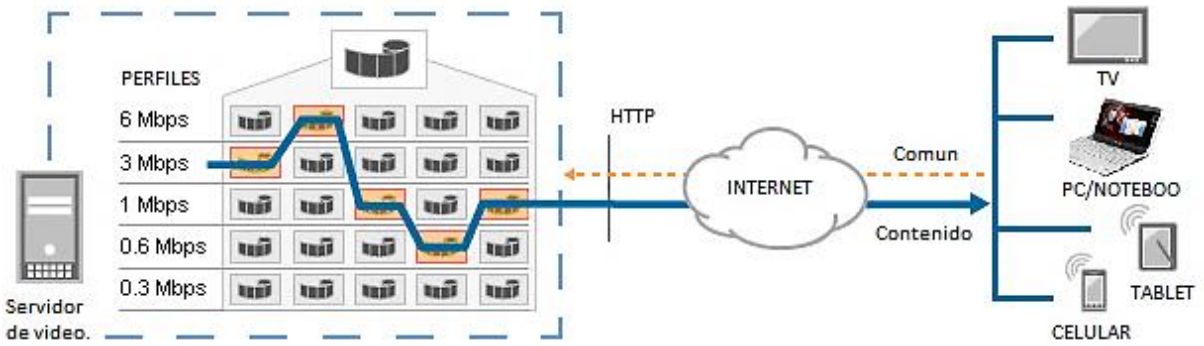

Figura 4.26 – Ejemplo de codificación en distintos perfiles y funcionamiento sobre protocolo de streaming adaptativo sobre internet. Fuente: Propia.

Se decidió incluir el servicio de transcoding de video como actividad HPC, para ser ejecutada sobre la plataforma desarrollada, debido a:

- La complejidad de cómputo intensivo requerido por el transcoding de video.
- Posibilidad para trabajar con la transcodificación de video de manera distribuida y paralela gracias a la fragmentación en *chunks* independientes.

Para lograr la integración de este servicio con el sistema implementado, se requiere la inclusión y configuración de la herramienta de código abierto FFmpeg, para realizar las tareas de transcoding de video, atendiendo a los siguientes apartados:

- Selección, definición y configuración de los perfiles de transcodificación, que serán utilizados con objeto de brindar un contenido adecuado para ser transmitido a través del streaming de video adaptativo.
- Definición del proceso de fragmentación de las fuentes de video, utilizando formatos y longitudes (duración de un fragmento de video) estándares para trabajar distribuida y paralelamente con dichos segmentos de manera independiente. Posteriormente, aplicar un mecanismo para unificar los fragmentos de video, una vez que han sido codificadas todas sus partes.
- Configuración de la herramienta para recibir un fragmento de video y aplicar la transcodificación, basada en los parámetros de compresión definidos.

4.3.1 - FFMpeg como herramienta de recodificación de video

Cómo herramienta de transcodificación de ćodigo abierto se decidió utilizar FFmpeg. El proyecto FFmpeg fue iniciado por el programador francés Fabrice Bellard. Su propósito era desarrollar un sistema de reproducción multimedia, brindar un conjunto de herramientas para la captura de audio y video y conversión de formatos de código abierto. La letra FF representa Fast Forward; y MPEG porque cumple con ese estándar de codificación de video. FFmpeg admite más de 40 tipos de codificación, y 90 tipos de decodificación.

La herramienta lee desde archivos de entrada (que pueden ser archivos normales, tuberías, flujos de red, dispositivos de captura, etc.), y escribe en un número arbitrario de archivos de salida, que se especifican mediante formato de URL, cómo se muestra en la figura 4.27.

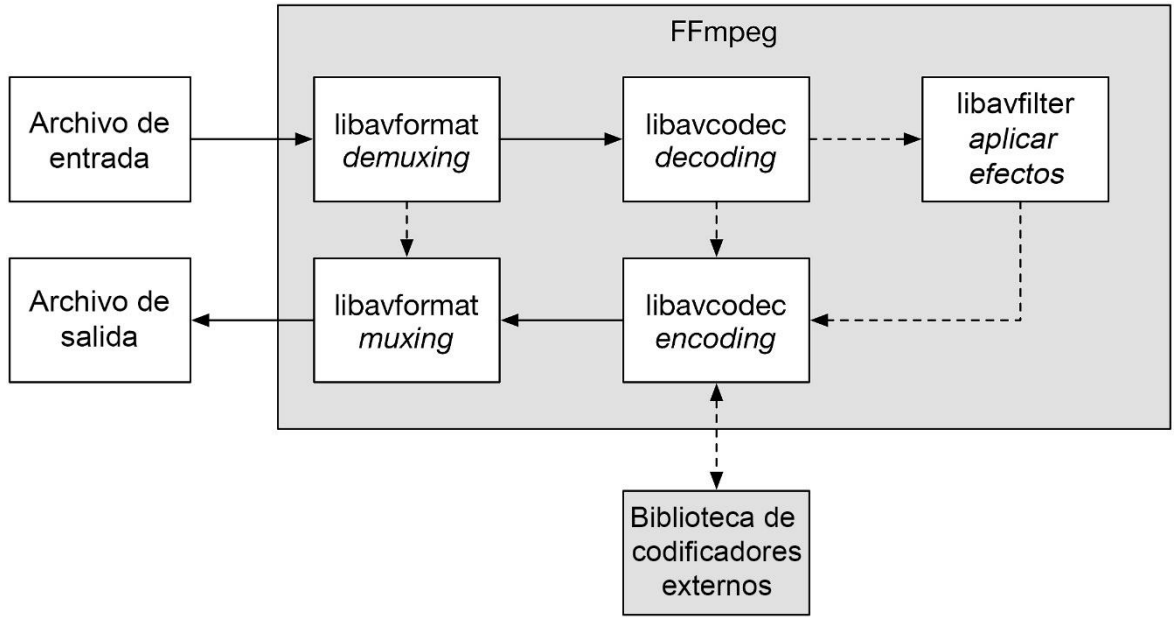

Figura 4.27 - Arquitectura básica de FFmpeg. Fuente: Propia. Versión adaptada de http://slhck.info/ffmpeg-encoding-course/#/9

Cada URL de entrada o salida puede, en principio, contener cualquier número de transmisiones de diferentes tipos (video, audio, subtítulos, extras). El número y/o tipos de flujos permitidos pueden estar limitados por el formato del contenedor.

En resumen, FFmpeg es una herramienta tan poderosa y versátil que se decidió utilizar en este proyecto como la base central y fundamental en el proceso de transcodificación [ZEN16][LEI13]. Permite, a través del envío de parámetros de configuración, definir las acciones y propiedades para la recodificación.

Si bien la herramienta permite realizar una innumerable cantidad de operaciones sobre las fuentes de video, en este proyecto se utilizó para [ZEN16]:

- Convertir un video de un formato a otro.
- Cambiar la compresión de un video para hacerlo más pequeño.
- Cortar y unificar segmentos de un video.
- Cambiar el tamaño del video (ancho * alto).
- Cambiar la velocidad, time lapse o cámara lenta.

En particular, se utilizó FFmpeg para transcodificar desde archivos fuentes a varios perfiles de salida, basados en la librería estándar de H.264 (libx264).

4.3.2 - Perfiles de codificación H.264 en FFmpeg

Para que la transcodificación de archivos multimedia orientada a un servicio de streaming adaptativo vía Internet funcione correctamente, requiere que FFmpeg sea configurado y utilizado adecuadamente de modo de entregar archivos eficientes y de calidad. Para ello, se construyeron múltiples perfiles de compresión.

En este apartado se describe en detalle el conjunto de resoluciones, velocidades de bits y configuraciones utilizadas para la codificación de video H.264 de alta calidad, así como también la justificación de cada una de esas elecciones. Cómo se describió anteriormente, la codificación de video es un juego de compensaciones y estas configuraciones representan un equilibrio entre el peso deseado y la calidad obtenida.

Para formar los perfiles de compresión en FFmpeg utilizando H.264 AVC (lib x.264) se debe conocer y configurar los siguientes apartados [RIC11]:

- **Perfiles:** Los perfiles de codificación se encargan de determinar la complejidad utilizada para codificar y decodificar un archivo de video [RIC11]. Cada uno de los cuales va destinado a una clase concreta de aplicaciones. En ellos se define qué conjunto de características puede usar el codificador y limita la complejidad de implementación del decodificador. En H.264 existen tres perfiles: línea base (baseline), principal (main) y extendido (high).
    - El perfil base (baseline) es utilizado cuando el video es liviano, como en el caso de las videoconferencias, o reproducción en teléfonos celulares con limitaciones en poder de cómputo. Provee la compresión menos eficiente de

las tres opciones, y el consumo más bajo de CPU en la decodificación. En este perfil no existen frames de tipo B.

- El perfil principal (main) es más capaz que el anterior, lo que generalmente se traduce en mayor eficiencia, si bien requiere una mayor demanda de poder de procesamiento (pero menos que el perfil High). Se usa en videos de calidad media para las aplicaciones de video en la web o móvil.

- El perfil extendido (high) es aplicable también a servicios de multimedia en Internet, pero orientado a aquellos que desean garantizar aspectos de QoS y QoE. También se utiliza para guardar copias de video en alta calidad que se almacenan en disco no volátil.

La codificación en una gama tan amplia de resoluciones permite, cómo se explicó, que el reproductor del dispositivo sea capaz de detectar la capacidad del dispositivo de lectura, el tamaño de pantalla y la velocidad de conexión a Internet del espectador, y de esa manera, elegir el archivo de video más apropiado en función de estas variables.

- **Niveles:** Los perfiles (baseline, main y high) abordan el problema de la complejidad del código y potencia de procesamiento [RIC11]. Además, H.264 tiene 11 grados de capacidad para limitar los requisitos de rendimiento, ancho de banda y memoria. Cada nivel define la frecuencia de bits y la frecuencia de codificación en macrobloque. A mayor resolución, mayor nivel requerido.

  Estos valores definen, como se muestra en la figura 4.28, los siguientes apartados:

  - Máximo de macrobloques.
  - Máximo de Tamaño de Frame.
  - Máximo video bitrate.
  - Resoluciones soportadas.

Nivel 1:	**Nivel 2:**	**Nivel 3:**	**Nivel 4:**	**Nivel 5:**
Max Video: 768 kbit/s	Max Video: 4 Mbit/s	Max Video: 20 Mbit/s	Max Video: 50 Mbit/s	Max Video: 240 Mbit/s
Max Resol: 352×288	Max Resol: 720×576	Max Resol: 1280×1024	Max Resol: 2048×1088	Max Resol: 4096×2304

Figura 4.28 - Máximas configuraciones soportadas por cada uno de los niveles en H.264. Fuente: Propia.

- **Tamaño de Pantalla:** Tamaño basado en los estándares de la televisión digital, en general siguiendo el siguiente patrón: 144p, 240p, 360p, 480p, 720p [HD], 1080p [FHD], 1440p [2K], 2160p [4K] y 4320p [8K].

- **Bitrate de video:** La velocidad de datos para un archivo de vídeo se basa en la tasa de bits. Así que una especificación de velocidad de datos de contenido de vídeo que funciona a 1 megabyte por segundo se administra como una tasa de bits de 8 megabits por segundo (Mbps). La tasa de bits para un vídeo Blu-ray de alta definición está típicamente en el rango de 20-50 Mbps. En su definición estándar, un DVD está compuesto, por lo general, a una tasa de entre 6-10 Mbps. Por su parte, la web de vídeo de alta calidad a menudo funciona a alrededor de 1 a 15 Mbps, y el video para móviles de acotadas características normalmente se da en los kilobits (kbps) [UHL18]. Es importante entender cómo el control de la tasa de bits corresponde a una calidad de vídeo y al tamaño del archivo. En general, una mayor tasa de bits acomodará más alta calidad de imagen en la salida de vídeo, al mismo tiempo que aumentará el tamaño del archivo.

- **FPS (Frames per second):** Se refiere a la cantidad de cuadros (imágenes) que se integran en un segundo de video. Los estándares se basan en 15, 24, 25, 30, 48, 50 y 60 fps. A mayor cantidad de imágenes, mayor suavidad de la misma a costas de un tamaño de archivo y necesidad de codificaciones/decodificaciones superiores.

- **Preset:** La definición del “preset” determina cuán rápido y “profundo” se realiza el proceso de encoding, impactando directamente en el resultado de la compresión. Esto quiere decir que, si se elige el perfil *ultrafast*, el proceso de transcoding correrá rápido, pero el tamaño del archivo será mayor a que si lo corriera, por ejemplo, con el perfil *medium* (la calidad percibida visual será la misma) [LEI13]. Los presets válidos para x.264 en FFMpeg son: ultrafast, superfast, veryfast, faster, fast, medium, slow, slower, veryslow y placebo.

- **Bframes:** Según cual sea el perfil de H.264, el codificador puede utilizar diferentes tipos de fotogramas (imágenes estáticas): fotogramas I, fotogramas P y fotogramas B. A modo de resumen, si el perfil lo permite, un mayor nivel de B frames implica más alta compresión, y la reducción del tamaño del archivo, con un costo computacional superior para comprimir y descomprimir el mismo.

- **Referencias entre Bframes:** Hay posibilidad de definir el esquema de relación y ubicación entre los fotogramas B (*B-frames*) dentro de una estructura GOP. Puede

no haber referencia entre ellos o realizar una predicción determinada. A mayor relación establecida, mejor será la predicción y la compresión de la estructura GOP.

- **Códec, bitrate y canales de Audio:** De la misma manera que se selecciona un códec de video para comprimir el material visual, es necesario determinar un códec para comprimir el audio. Para los casos de calidades más reducidas, habitualmente el audio se codifica en formato AAC-LC. Se recomienda mantener una configuración estéreo, sobre una base de 128 kbps lo que produce, a pesar de la compresión, una excelente calidad, prácticamente indistinguible de la original, y sin ningún "estallido" audible. Por su parte, para los perfiles más potentes, se recomienda definir una versión de 6 canales (sonido 5.1 o superior) y codificado en AC3. Para aprovechar el códec al máximo y otorgar una máxima calidad, se recomienda configurar sobre una base de 512 kbps. El archivo resultante (perfil *high* para video y AC3 para audio), permite contemplarlo como un material audiovisual “maestro a largo plazo”. Esto significa que se generará un recurso que presenta una mínima pérdida de calidad, a partir del cual, si se requiere, se puede volver a recodificar en diversas calidades.

- **Muestreo de Audio:** Es la frecuencia de muestreo con la que el audio es sensado. Por defecto es 44100 Hz. Sin embargo, en algunos casos, se puede elevar a 48000 Hz con el objetivo de obtener una calidad superior.

Sobre la base de conocimiento establecida y de relevar casos de aplicaciones reales, se definen los siguientes perfiles de codificación:

- 3 perfiles de compresión basados en h.264 *baseline*, para dispositivos antiguos y conexiones a Internet con velocidades acotadas.

- 3 perfiles de compresión sustentados en h.264 *main*, para dispositivos actuales hasta calidad HD.

- 3 perfiles de compresión en h.264 *highline*, para dispositivos actuales y de calidad hasta 8K.

Para los mismos se establecieron los parámetros de configuración que se muestran en la tabla 4.1.

Tabla 4.1 - Composición de cada perfil de compresión. Fuente: propia

	Tam. de Pantalla	Cód. de Video	Bit. de Video	Niv. de Comp.	FPS	Preset	Bframes
Perfiles High							
	4096x2160	libx264	15600	L@5.1	60	very slow	6
	2560x1440	libx264	7800	L@5.1	48	slower	6
	1920x1080	libx264	3900	L@4.1	30	slow	6
Perfiles Main							
	1280x720	libx264	2000	L@4.1	25	medium	3
	1024x576	libx264	1400	L@4.0	25	medium	3
	852x480	libx264	900	L@3.1	25	fast	3
Perfiles Base							
	720x406	libx264	900	L@3.0	24	fast	0
	640x360	libx264	700	L@3.0	24	faster	0
	320x200	libx264	500	L@3.0	24	ultra fast	0

4.3.3 - Proceso de Fragmentación y Unificación de archivos

Una vez que un video es recibido, se debe convertir automáticamente al formato Transport Stream (.ts), que es el recomendado para la transmisión de video en Internet [ETS17]; y se ha probado que tiene un impacto positivo en la latencia, la compatibilidad de reproducción y la experiencia de visualización [GLO19]. Además, elimina los errores comunes que tienen otros formatos al ser difundidos a través de este medio. Por ejemplo, los archivos MP4 requieren que cada parte disponga de una cabecera de información extra al comienzo del archivo (“moov atom”) [HOA16][MAX10]. Sin esta información, el streaming basado en MP4 no funcionará y el reproductor no podrá leerlo.

Un ejemplo del proceso de conversión a formato Transport Stream se presenta a continuación:

```
$ ffmpeg -y -i file -c copy fileconverted.ts
```

Donde,

-y	Sobreescribe el archivo resultante (sin preguntar)
-i	El archivo (local o remoto)
-c copy	Determina que se mantiene sin cambios los códec de audio y video de los recursos

Esta conversión a Transport Stream (.ts) permite, posteriormente, fragmentar el video en partes más pequeñas e independientes entre sí, lo que posibilita la recodificación, transmisión y reproducción del vídeo de manera distribuida y paralela.
A la hora de elegir una duración de segmento, para fragmentar los archivos de video, se debe

asociar al tamaño que va a manejar el servidor de streaming para servir los mismos a través de un protocolo HTTP encargado para dicha función, por ejemplo, HLS o DASH.

En base a los estudios realizados, MPEG-DASH generalmente recomienda dividir los videos en segmentos más pequeños para su entrega que HLS. La longitud predeterminada del segmento para HLS es de 10 segundos, mientras que los segmentos MPEG-DASH suelen tener una duración de entre 2 y 4 segundos [LED15]. Esto significa que MPEG-DASH, con su configuración base, permite un cambio más rápido entre niveles de calidad, para ajustes más rápidos a las condiciones de la red.

En todos los casos, la decisión de tomar un tamaño de fragmento es particular de cada proveedor de servicios. Sin embargo, se debe verificar minuciosamente cómo funcionan estas opciones de configuración en las plataformas de destino antes de la implementación.

En este estudio, siguiendo las recomendaciones de los distintos protocolos actuales de videostreaming, se definieron los siguientes tamaños de fragmentos:

- 10” para dar soporte directo a HLS.
- 3” para dar soporte transparente a MPEG-DASH.

Estos valores se encuentran, cómo vimos anteriormente, dentro del tamaño recomendado para el servicio de streaming de video.

A continuación, se muestra una línea de ejemplo de FFMpeg para realizar esta tarea:

```
$ ffmpeg -y -i -codec copy -map 0 -f segment -segment_time 10 -segment_format mpegts file_part_%d.ts
```

Dónde,

-y	Sobreescribe el archivo resultante (sin preguntar)
-i	El archivo (local o remoto)
-c copy	Determina que se mantiene el codec de audio y video de los recursos sin cambios
-map 0	Se utiliza para elegir qué flujos de la entrada (s) deben incluirse en la salida (s). 0 implica el primer canal
-f segment	El archivo (local o remoto)
-segment_time 10	Determina la duración del fragmento (en segundos)
-segment_format mpegts	Determina el formato de salida de los fragmentos
file_part_%d.ts	Dependiendo de la duración del video, genera N partes con el formato file_numpart.ts

En lo que respecta al proceso de unificación (que se explicará en el apartado 4.4), una vez que todas las partes están procesadas, se integran a través de un comando único de ffmpeg entregando también un archivo Transport Stream.

Un ejemplo del comando de unificación se presenta a continuación:

```
$ ffmpeg -f concat -safe 0 -protocol_whitelist file,https,tls,http,tcp -i urlfile -c copy -f mpegts result.ts
```

Donde,

-f	concat informa que se concatena la fuente
-safe 0	indica que no verifique el nombre del archivo (expresiones regulares inseguras)
-protoloc_whitelist	informa los protocolos de entrada/salida permitidos
-i	el archivo con las direcciones físicas de los archivos (locales, remotas)
-c copy	para determinar que se mantiene el codec de audio y video de los recursos
-f mpegts	Define el formato de salida (mpegts en este caso para streaming)

4.3.4 - Proceso de Transcodificación

Sobre la base de los perfiles de codificación definidos, y los fragmentos de videos, se cuenta con un conjunto de equipos que serán los encargados de tomar esos segmentos independientes y procesarlos de manera paralela. Para ello, la herramienta FFmpeg toma el archivo fuente y aplica, en base a los parámetros definidos, una transcodificación.

A continuación, se muestra una línea de comando de FFmpeg para realizar una compresión relacionada con el primer perfil definido (4K):

```
$ ffmpeg -y -i source.mp4 -s 4096x2160 -aspect 16:9 -c:v libx264 -g 50 -b:v 15600k -profile:v high -level 5.1 -r 60 -preset veryslow -threads 0 -c:a ac3 -b:a 512k -ar 48000 -ac 6 result.mp4
```

Donde,

-y	Sobreescribe el archivo resultante (sin preguntar).
-i	Recurso fuente.
-r	Frames per sec. (cuadros por segundo). En este caso 60.
-s	Resolución (Tamaño de Pantalla). En este caso 4096x2160.
-aspect	Aspecto 4:3 o 16:9 (primer valor: Ancho; segundo valor: alto). En este caso 16:9.

-c:v	Librería de encoding de video. En este caso x264.
-b:v	Video Bitrate. Es un promedio. Donde hay poca información se encodea a baja tasa de bits y se reserva para donde hay mucho contenido. En este caso se define una tasa promedio de 15 mbps.
-profile:v	Perfiles (baseline, main, high). En este caso high.
-level	Nivel de compresión. En este caso L@5.1.
-bf	Máximo consecutivo de frames b (no en baseline).
-b_strategy	0 (sin ubicación dinámica); 1 (ubicación rápida, rápido pero poco eficiente); 2 (ubicación lenta pero eficiente). En este caso 1.
-refs	Número de frames referenciados, la recomendación es no mayor a 6. En este caso 3.
-preset	Son un conjunto de opciones que proveen la velocidad de encodeo a un determinado nivel de compresión (ultrafast a very slow). En este caso very slow.
-threads	Asignación de cores al proceso. Si se elige 0, es el formato automático, donde FFMPEG decide los threads. Caso contrario, debe asignarse manualmente. En este caso, se definió 0 para una configuración automática
-c:a	Librería de encoding para audio. En este caso AC3.
-b:a	Bitrate de audio. En este caso 512 kbps.
-ar	Muestreo de audio. En este caso 48Khz.
-ac	Canales de Audio. En este caso 6.

4.4 - Integración de FFmpeg en microservicios de la plataforma

En el apartado anterior (4.3), se describió un proceso básico de cómo se realizan estas tareas de transcodificación de un video, definición de perfiles de compresión, conversión a Transport Stream, fragmentación, procesamiento y unificación, utilizando la herramienta FFMpeg.

El objetivo aquí es detallar cómo estos procesos se integraron a las tareas de la plataforma desarrollada, y las optimizaciones que se realizaron para explotar sus características y ofrecer un servicio que las aproveche correctamente.

Cómo se explicó en el capítulo 3, la plataforma HPC desarrollada está compuesta por los siguientes componentes:

a) Microservicios Dockerizados para la administración y gestión de la plataforma.

b) Servicios de colas (tareas) y base de datos (estadísticas) Dockerizadas.

c) Orquestación, escalado, replicación y automatización de contenedores, a través de Kubernetes.

d) Trabajadores móviles Android ARM y x86 Dockerizados.

A continuación, en la figura 4.29, se presenta un resumen de los ajustes realizados, para que el sistema responda a la actividad HPC de transcoding de video:

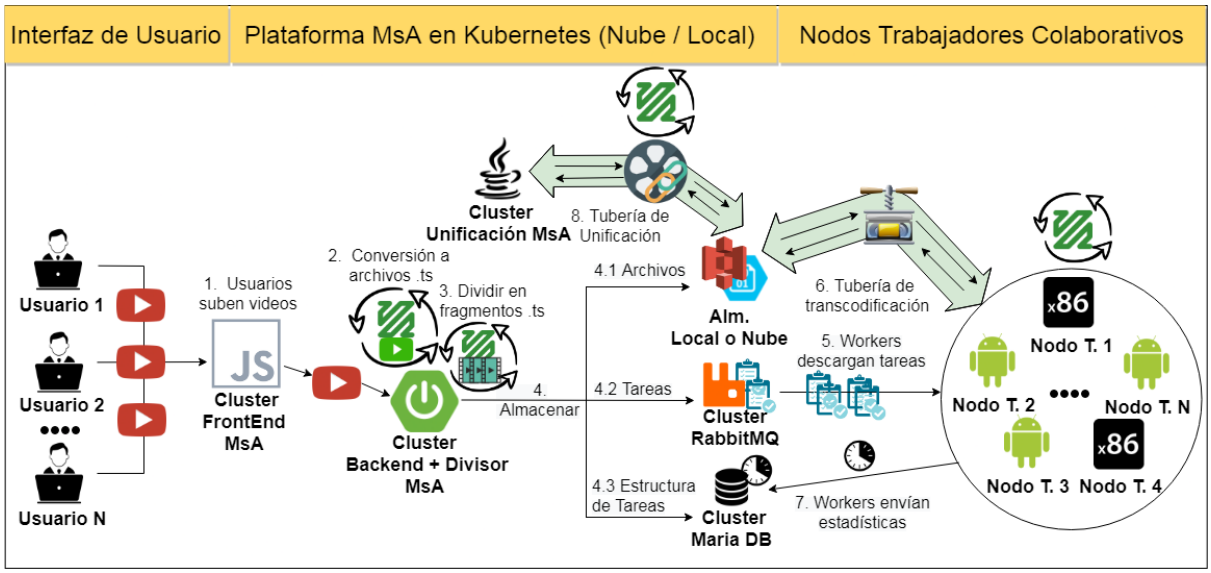

Figura 4.29 - Plataforma HPC ajustada para la resolución de la transcodificación de video de manera distribuida y paralela. Fuente: Propia.

En lo que resta de este apartado, se describen los cambios realizados sobre cada uno de los componentes.

a) Microservicios Dockerizados para la administración y gestión de la plataforma

1) Frontend MsA:

El servicio de Frontend permite a los clientes subir los archivos fuente, seleccionar y definir parámetros para la tarea de compresión de video; ver el progreso y resultado del proceso de transcodificación. Se presenta a continuación, en la Figura 4.30, una captura de pantalla del formato y las opciones disponibles.

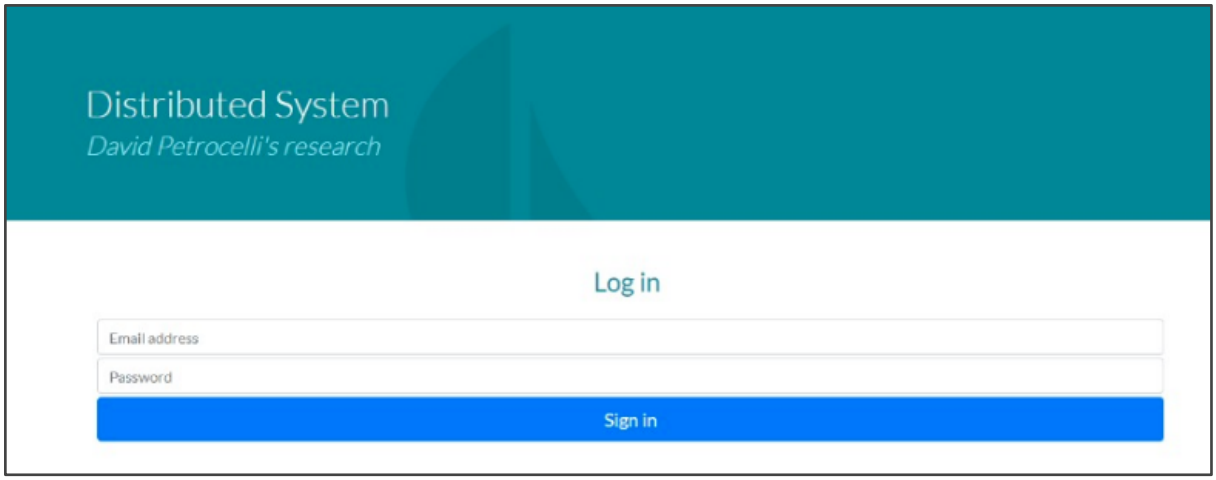

Figura 4.30 - Captura de pantalla de la GUI del sistema. Fuente: propia

Al mismo tiempo, con objeto de automatizar y simplificar los procesos de prueba y análisis de la plataforma, se configuró la herramienta CURL para realizar las tareas desde la consola, generando una petición GET con los videos fuente y los parámetros de codificación. A continuación, se presenta un ejemplo de la configuración de esta herramienta:

```
$ curl -H 'Content-Type: multipart/form-data' -F 'file=@sample.mp4'
'http://url:8081/upload?user=david&encodingParams=240-480-720-hd-2k'
```

Donde,

-H 'Content-Type: multipart/form-data'	Define el header necesario para que sea posible subir un archivo en múltiples partes.
-F 'file=@sample.mp4'	Define el archivo que se va a ir subiendo por partes.
http://url../upload?user=david &encodingParams=240...'	Define la url del servidor BackendWeb. Además, define, por un lado, el usuario que está enviando el archivo a comprimir; y por otro, los identificadores de los perfiles de transcoding.

2) Backend Web+Split MsA

El Backend Web recibe el archivo de video y los parámetros definidos por el usuario. En el ejemplo presentado, es el video sample.mp4, y los parámetros de encoding asociados (perfiles: 240, 480, 720, hd, 2k). Esto significa que el video debe ser transcodificado en 5 perfiles de compresión. Una vez recibido, la clase Java encargada, llamará a un proceso (pipe) a ejecutarse a través de la consola, llamando a la función *bash* (runBash).

En esta adaptación de la plataforma para compresión de video, la función se utiliza para fragmentar dichos archivos, con una duración de 10 segundos por fragmento (cuyo parámetro puede ser configurado dependiendo de las necesidades). El comando utilizado se presenta a continuación:

```
String command = "ffmpeg -y -i sample.mp4 -codec copy -map 0 -f segment -
segment_time 10 -segment_format mpegts sample_part_%d.ts"
this.runBash(command);
```

Una vez que los recursos fuentes son divididos, cada segmento se guarda en un almacenamiento HTTP de bloques en la Nube de bajo costo, cómo Azure Blob Storage o Amazon S3 [ZOU18]; o, cómo en el resto de los casos, en un sistema de archivos local, dependiendo el ambiente de ejecución, tecnologías e infraestructura disponible. Al mismo tiempo se registran las tareas en el sistema de colas y la estructura de datos y estadísticas en el SGBD.

3) Backend Joiner+Unifier MsA

Una vez que el Backend Joiner detecta que la cantidad de partes completadas es igual a la cantidad de partes totales de un trabajo, genera una tarea de unificación que se registra en el sistema de colas. Posteriormente, alguno de los servicios Unifier toma dicha actividad y se encarga de realizar la unificación sobre el material audiovisual correspondiente.

Cabe destacar que en este punto es donde se aplica una de las mejoras específicas para este servicio.

Para poder realizar múltiples trabajos en paralelo, se incluyeron tuberías Linux [WAM14]. Una tubería Linux tiene la siguiente estructura:

```
$ task_1 | task_2 ..|... task_N
```

Donde, mientras se procesa *task_1*, el flujo de salida se va enviando al siguiente bloque de instrucciones, y así sucesivamente con los respectivos segmentos de la tubería. De esta manera, las tareas se completan de manera paralela, reduciendo los tiempos de ejecución y optimizando la capacidad de hardware para la realización de la misma.

En el caso particular de la unificación de tareas, como primer paso se obtienen todas las direcciones de las partes procesadas y se genera un archivo con esta información. Luego, utilizando tuberías de Linux, se realiza el proceso de unificación de manera concurrente. Es decir que mientras se van descargando los fragmentos de video con CURL, FFmpeg va uniendolos con los parámetros definidos, y subiéndolos al almacenamiento de destino (también a través de un flujo de datos continuos a través de CURL). Se presenta a continuación un fragmento de código donde se realizan estas actividades:

```
...
// [PASO 1] - Armar la llamada curl del archivo destino
String curl = "curl -X PUT -F \"file=@-\" -H \"Content-Type:
multipart/form-data\" -H \"Host: " + context + "\" \"http://" + context
+ ":8080/uploadFile?folder=s3-bcra-finished&fileName=" + finalName +
"\"\n";
// [PASO 2] - Obtener los fragmentos procesados a partir de la lista de
urls
String concatenate  =  "ffmpeg -f concat -safe 0 -protocol_whitelist
file,https,tls,http,tcp -i urls"+nThread;
// [PASO 3] - Invocar a la tubería Linux
concatenate             +=    "-c copy -f mpegts pipe:1 | ";
// [PASO 4] - Indicar que la salida del pipe sea envíado al CURL
concatenate         +=  curl;
// [PASO 5] - Llamar al ejecutor de comandos
runBash(concatenate);
...
```

Cómo puede visualizarse en el segmento de código presentado, se utiliza una petición CURL para configurar la ubicación del archivo a concatenar (en este caso de un archivo alojado en el microservicio Servidor de Archivos). Más tarde, se toma el archivo que se había generado previamente con la información de todas las ubicaciones de los fragmentos procesados, los cuales, gracias a la tubería definida, FFMpeg irá descargando paulatinamente, unificando y subiendo al almacenamiento destino.

4) **File Server MsA**

No se realizaron adaptaciones al Sistema de archivos sobre el presentado en la plataforma general.

b) Servicios de colas (tareas) y base de datos (estadísticas) Dockerizadas

No se realizaron adaptaciones al middleware de colas ni al SGBD.

c) Orquestación, escalado, replicación y automatización de contenedores a través de Kubernetes

No se realizaron adaptaciones al orquestador de contenedores.

d) Trabajadores móviles Android ARM y x86 Dockerizados.

A la aplicación trabajadora HPC, tanto móvil ARM como tradicional x86, se le adiciona una clase que será la encargada de realizar la compresión de video.

Una vez obtenida la tarea a realizar, y habiendo detectado que es del tipo “VideoTranscodingTask”, se inicia el proceso de compresión paralelo. Para ello, se detecta el sistema de archivos que se está utilizando en el momento (AWS, Azure o Local) y se construye la estructura básica necesaria para enviar la petición CURL con los resultados, de manera similar a la que se presentó en el proceso de unificación.

Por otro lado, se obtiene la dirección del archivo a descargar (endpoint) y también los parámetros de compresión asociados. Con estas tres partes (fragmento fuente, parámetros de compresión y destino) se construyen dos tuberías de Linux paralelas. Mientras se está descargando el video con una petición curl HTTP GET, FFmpeg toma el flujo de datos y va transcodificando, a la vez que escribe el flujo de salida en el almacenamiento destino con una petición curl HTTP PUT. A continuación, se muestra un fragmento del código utilizado:

```
...

// [PASO 1] - Armar la llamada curl del archivo destino
String curlSalida       =     "curl -X PUT -F \"file=@-\" -H \"Content-
Type: multipart/form-data\" -H \"Host: " + this.context + "\" \"http://"
+ this.context + ":8080/uploadFile?folder=s3-bcra-compressed&fileName="
+ target + "\"\n";
// [PASO 2] - Se arma el curl de descarga y el pipe con el ffmpeg de
procesamiento
String curlEntrada      = "curl \""+ source + "\" | " + basePath+"ffmpeg
-i -";
// [PASO 3] - Se configuran los parámetros de compresión (desde Mensaje)
String parameters   += curlEntrada;
parameters          += " -s "+parametersFromMsg.getVideoResolution();
parameters          += " -aspect 16:9 -c:v
"+parametersFromMsg.getVideoCodec();
parameters          += " -b:v "+parametersFromMsg.getVideoBitrate()+"k";
parameters          += " -profile:v
"+parametersFromMsg.getVideoProfile();
parameters          += " -level "+parametersFromMsg.getVideoLevel();
parameters          += " -preset "+parametersFromMsg.getVideoPreset();
parameters          += " -threads 0 ";
// [PASO 4] - Se prepara para trabajar en formato de streams (formato
mpegts) y se arma la segunda tubería
parameters          += "-f mpegts - | ";
// [PASO 5] - Se redirecciona el flujo de salida vía CURL
String runner       += parameters+curlSalida;
// [PASO 6] - Se llama al ejecutor de comandos
runBash(runner);

...
```

Como resultado, las operaciones de E/S sobre disco y el uso de memoria se reducen drásticamente, mejorando el rendimiento del sistema. Al finalizar este proceso, se confirma la

tarea realizada en el sistema de colas, y se actualiza la información estadística (tiempos) en la base de datos.

4.5 - Repositorio de código fuente

Todos las funciones aquí desarrolladas y adaptadas se encuentran disponibles y accesibles públicamente en el siguiente repositorio de Github: https://github.com/dpetrocelli/PhD-Mobile-K8s-HPC-Platform. El objetivo es que cualquier interesado pueda tomar la plataforma desarrollada y adaptarla a sus necesidades.

Por su parte, las imágenes de Docker compiladas y ya listas para utilizar se encuentran disponibles en el Registro DockerHub a través del siguiente enlace: https://hub.docker.com/u/dpetrocelli

A su vez, si bien a lo largo de este apartado se describieron los procesos implicados y las adaptaciones realizadas para servir a la transcodificación de vídeo bajo demanda, todos los aspectos de configuración y detalle del funcionamiento de la plataforma se encuentran detallados en el Anexo I - Despliegue de la plataforma en Kubernetes.

4.6 - Conclusiones del capítulo

Con la aparición de la transcodificación de vídeo bajo demanda, como una clave esencial para el desarrollo de los servicios de streaming de video adaptativo, es crucial estudiar cuales son las mejores prácticas, arquitecturas y los factores que influyen en la realización correcta de las tareas y los tiempos de ejecución asociados. Además, es necesario llegar a un equilibrio entre el rendimiento requerido y los costos adquiridos al momento de la ejecución de esta tarea. La compensación se complica aún más cuando se considera la heterogeneidad de los servicios computacionales actuales (Nube, infraestructuras híbridas y locales) que se encuentran disponibles.

En este capítulo, se definió la transcodificación de videos como tarea HPC para ser ejecutada en el entorno de la plataforma desarrollada. Para ello, se adaptaron mínimamente los componentes de la arquitectura, a fines de construir un sistema de transcodificación de video distribuido, utilizando la herramienta FFmpeg.

De esta manera, un video inicial se divide en múltiples segmentos independientes, cuya tarea es realizada por un nodo maestro. Mientras se van obteniendo los fragmentos, se almacenan en un sistema de archivos, se registra la información y estructura de los trabajos en el sistema

de colas y en el motor SGBD. Una vez realizado, los nodos extremos obtienen estos trabajos, de modo que con cada tarea se generan operaciones de transcodificación simultáneas.

Como resultado, los segmentos transcodificados se almacenan y, cuando todas las partes están completas, se concatenan para reconstruir el video en el nivel de calidad de destino.

Es relevante destacar que, para realizar esta adaptación, los microservicios incluyeron la librería FFmpeg y tuberías de Linux (pipelines), para optimizar las actividades. El uso de las tuberías ayuda, por un lado, a realizar operaciones en paralelo permitiendo explotar la capacidad de cálculo de los nodos, logrando un mejor rendimiento y mayor estabilidad en el proceso de transcodificación. Por otro, se reduce la cantidad de espacio de almacenamiento a la hora de descargar archivos completos, y también disminuir la necesidad de asignación de memoria a los procesos en ejecución FFmpeg. Esto contribuye a una mayor y más ordenada optimización de recursos, y a evitar fallos en la resolución de tareas, independientemente de la capacidad del dispositivo.

5 - Modelo de análisis y experimentación

En el capítulo anterior se describió cómo la plataforma se ajusta para poder correr tareas de compresión de video, de manera distribuida y paralela. También se mostró que la tarea HPC definida requiere una capacidad de procesamiento intensivo, flexible y escalable. En consecuencia, para asegurar que el prototipo diseñado y desarrollado (explicado en el capítulo 3) cumple con las necesidades para este tipo de actividades, se validó, comparó y evaluó su desempeño a través de diversas configuraciones y cargas de trabajo.

En este capítulo se presentan los modelos, escenarios de prueba planteados y las métricas de evaluación definidas. También se detallan los aspectos relevantes para la ejecución de las experimentaciones. Finalmente se describe el análisis de los resultados obtenidos.

5.1 - Objetivos

Los objetivos de este capítulo fueron:

- Validar la capacidad, rendimiento, consumo e integración de los dispositivos de procesamiento (workers), tanto de arquitectura x86 cómo ARM, en la plataforma desarrollada para resolver tareas HPC.
- Comprobar el funcionamiento, comportamiento, rendimiento, escalabilidad y flexibilidad de la plataforma desarrollada para resolver tareas HPC, a través de diversas cargas de trabajo.
- Validar como la arquitectura proporciona un uso eficiente de los recursos administrados; a la vez que permite optimizar y flexibilizar la capacidad requerida, y los costos asociados, a la hora de completar las tareas involucradas.

A continuación, se describe y detalla cada una de las evaluaciones realizadas.

5.2 - Descripción de la fase experimental

Para cumplir con los objetivos de esta tesis, se experimentó sobre dos grandes áreas de evaluación. Por un lado, se llevaron a cabo pruebas de rendimiento y consumo sobre dispositivos x86 y ARM móviles. Estas evaluaciones buscan validar que los dispositivos ARM cuentan con la potencia necesaria para resolver este tipo de tareas, de una manera competente y efectiva respecto de las arquitecturas tradicionales. Por otro lado, se realizaron las evaluaciones de escalabilidad y flexibilidad de la plataforma, ante diversas cargas de

trabajo, con la finalidad de comprender cómo la plataforma responde y se ajusta para atender a las solicitudes de los usuarios.

Para cada evaluación se definieron los siguientes apartados:

- Definición de escenarios y equipamiento de prueba.
- Limitaciones y restricciones.
- Selección de material audiovisual.
- Configuración de perfiles de compresión y duración de fragmentos de video.
- Definición de métricas de evaluación.
- Ejecución de pruebas y análisis de resultados.
- Conclusiones obtenidas.

5.2.1 - Evaluación de rendimiento y consumo energético de los nodos trabajadores

Desde sus inicios, las CPU x86 (Intel/AMD) y sus correspondientes GPUs (Intel/AMD/ NVidia) fueron desarrolladas para resolver problemas complejos sin tener en cuenta su consumo energético. Fue en el siglo XXI que, con el crecimiento exponencial de los centros de datos, los gigantes de los semiconductores comenzaron a interesarse y abordar más ampliamente el TDP de sus chips, optimizar sus arquitecturas para un IPC más alto y reducir su consumo de energía [MAS20][AHM19][BAR19][ZAI17]. Si analizamos el contexto actual, el uso de de energía y los medios de refrigeración son dos de los mayores costos para los centros de datos [ZAM19][KAN17][RAS12]. Es por ello que el consumo de energía y la eficiencia son claves a la hora de diseñar cualquier sistema informático en la actualidad. En este sentido, se reconoce que el logro de los benchmarks de cómputo no debe hacerse por fuerza bruta sino optimizando recursos y arquitecturas, teniendo en cuenta el alto costo económico y el costo ambiental de la energía requerida por los grandes centros de datos.

A diferencia de los procesadores x86, cómo se describió en el capítulo 2.8.4 - Arquitectura de procesadores ARM y masividad de dispositivos, los chips basados en ARM fueron concebidos teniendo como prioridad la eficiencia energética. Estos chips desde sus inicios estuvieron orientados a dispositivos móviles y microdispositivos, que en su mayor parte funcionan con baterías. Aunque la potencia bruta de los chips ARM es menor que la de sus homólogos x86, estos son mucho más eficientes en cuanto a potencia

[GIN20][BHA19][CRI19][PRA19][PET17]. Al mismo tiempo, los dispositivos basados en ARM superan ampliamente en número a las computadoras x86 tradicionales, por lo que, si bien la potencia individual puede ser menor, existe una base de instalación muy grande con largos períodos de inactividad durante la carga que, si se administran adecuadamente, podrían convertirse en centros de datos distribuidos masivos que consumen solo una fracción de la energía para la misma potencia informática que sus contrapartes tradicionales. Este principio de carga de trabajo distribuida durante el tiempo de inactividad se ha utilizado anteriormente en los equipos x86 para iniciativas colaborativas como SETI @ Home y Folding @ Home. En estos proyectos se utiliza la potencia informática de las computadoras que las personas dejaban encendidas durante los períodos inactivos para construir grandes centros de datos virtualizados. Actualmente, algunos investigadores han convertido este modelo colaborativo para que funcione en dispositivos móviles [GED20][LAV19][HIR18][SCH18][MAC17].

En base a lo que se mencionó anteriormente, en este trabajo se construyó y evaluó una plataforma colaborativa para HPC basada en dispositivos móviles que utilizan la capacidad disponible en los equipos móviles ARM. En particular se implementó la compresión de video cómo tarea HPC utilizando la biblioteca FFMpeg, que permite que el cliente solicite diferentes perfiles y configuraciones de compresión.

La plataforma se evaluó a través de una serie de métricas de rendimiento y uso de energía tanto en ARM como en chips x86. Esto permitió llevar a cabo un análisis comparativo entre las arquitecturas y demostrar que es completamente factible descargar grandes cargas de trabajo informáticas a arquitecturas ARM. Además, proporcionó datos suficientes para comparar el uso de energía para la resolución de actividades de compresión de video en chips basados en ARM en comparación con sus contrapartes x86. Para obtener estas mediciones se incluyeron y desarrollaron prototipos de medición (de hardware y software), que se presentan en el ANEXO II - Dispositivos de medición de consumo energético

Definición de escenarios y equipamiento de prueba

Para evaluar los requerimientos computacionales exigidos por las tareas de compresión de video, se utilizó una combinación de recursos basados en x86 y ARM.

Para la selección final del equipamiento y la construcción de un clúster por tipo (de alta y media gama), se tomó en cuenta la regla de seleccionar aquel equipamiento que:

- Se disponga un laboratorio con varios equipos homogéneos (al menos 5).
- De los equipos disponibles, seleccionar aquellos que sean más actuales, buscando obtener resultados actualizados.

- Presente, según las especificaciones del fabricante, el mejor rendimiento y el menor consumo.

Sobre este equipamiento, se definieron diversos escenarios de transcodificación a través de los cuales fue posible:

- Validar la integración con la plataforma
- Evaluar los tiempos de respuesta.
- Registrar el consumo energético.

Tanto para las arquitecturas x86 cómo para los equipos ARM, se definió un clúster de recursos de gama alta y un clúster de nivel intermedio, compatibles con la codificación/decodificación de vídeos x264. Las características relevantes del equipamiento utilizado se enumeran en la Tabla 5.1.

Tabla 5.1 - Arquitectura de clústeres utilizados para los escenarios de pruebas definidos. Fuente: Propia.

Cluster x86/ARM					
Nombre	**Nro Nodos**	**Procesador**	**Memoria**	**Disco Rígido**	**GPU**
x86-ClusterI7	10	Intel i7-4770-8x3,4Ghz	16 GB-DDR3	500GB-7200RPM	Intel HD Graphics 4600
x86-ClusterI5	8	Intel i5-2400-4x3,1Ghz	8 GB-DDR3	500GB-7200RPM	AMD Radeon HD 7670-480x800Mhz
ARM-ClusterS7	10	Exynos 8890-4x2.3 GHz+4x1.6 GHz	4 GB-LPDDR4	16 GB-SD IntStor	Mali-T880-12x650MHz
ARM-ClusterA5	6	Exynos 7880-8x1.9 GHz	3 GB-LPDDR4	16 GB-SD IntStor	Mali-T830-2x600Mhz

Para generar una prueba que pudiera saturar dispositivos de ambas arquitecturas, se analizaron trabajos y configuraciones previas [RAK20] [LEE15] [GAR10] y se utilizaron como referencia para esta prueba. Luego, se seleccionaron tres videos fuente con características relevantes para ambas plataformas (códecs utilizados, tasa de bits, nivel de compresión, tamaño de fotograma, relación de aspecto, bits por píxel, etc.). La descripción general de las propiedades más importantes de cada video se detalla en la Tabla 5.2.

Tabla 5.2 - Videos fuente utilizados a la hora de realizar las tareas de compresión de video. Fuente: Propia.

Videos de prueba	Duración	Tamaño	Tam. Pantalla	Códec de V.	Bitrate de V.	Perfil	Nivel	FPS	Códec de A.	Bitrate de A.	Muestreo	Canales de A.
3dmark_4k_120fps.mkv	2m 35 segs	487 MB	3840x2160	AVC x264	27545 Kbps	high	@L6	120	Vorbis	160	48000	2
bbb_sunflower-2160.mp4	10 m 34 segs	605 MB	3840x2160	AVC x264	8000 Kbps	high	@L5.1	60	aac	160	48000	6
bigbuckbunny-1500.mp4	9 m 56 segs	109 MB	1080x608	AVC x264	1080 Kbps	main	@L3.1	24	aac	128	48000	2

El códec elegido AVC h.264 [LIN19] (biblioteca x264 en FFMpeg) es ampliamente compatible con los dispositivos actuales y plataformas de transmisión de video como Youtube, Vimeo, Netflix, etc. Los perfiles de codificación de x264 que se seleccionaron para ejecutar las pruebas mencionadas en esta actividad, siguiendo las bases definidas en 4.3.2 - Perfiles de codificación H.264 en FFmpeg, se detallan a continuación en la Tabla 5.3.

Tabla 5.3. Descripción de las propiedades de los perfiles de compresión utilizados en FFMpeg. Fuente: Propia.

Códec x264	Tamaño	Códec de V.	Bitrate de V.	Nivel	Preset	FPS	Códec de A.	Bitrate de A.	Muestreo	Canales de A.
High Profile										
4K	4096x2160	libx264	15600	L@5.1	very slow	60	ac3	512	48000	6
1080 Full HD	1920x1080	libx264	3900	L@4.1	slow	30	ac3	320	48000	6
Main Profile										
720 HD	1280x720	libx264	2000	L@4.1	medium	25	aac	320	44100	2
480	852x480	libx264	900	L@3.1	fast	25	aac	256	44100	2

Este set de perfiles definidos permitió abarcar desde tareas simples y de baja compresión cómo el caso del perfil 480p, hasta actividades más complejas y de mayor costo computacional, cómo las definiciones 1080p FHD o 4K. La codificación y compresión seleccionadas son las mismas que se encuentran en las plataformas de transmisión de video, las cuales permiten al público acceder al contenido en diferentes calidades, para que coincida con las capacidades de su dispositivo y del enlace a Internet [SHI19][RAA17].

Además de los perfiles de compresión, también se definió la longitud (duración) de cada video a procesar, decidiendo dividir los videos originales en fragmentos de 1 y 3 segundos para cada nivel de compresión. Estos valores utilizados provienen de las recomendaciones de los servicios de transmisión de vídeo a través de Internet (MPEG-DASH) [SAN17][LED15].

Además de registrar el rendimiento obtenido a la hora de procesar los videos, se obtuvieron también los consumos de energía de los dispositivos utilizados. Para ello, se construyeron herramientas de medición de consumo energético personalizadas, que se detallan en el ANEXO II - Dispositivos de medición de consumo energético.

Definición de métricas a capturar

Para evaluar el rendimiento y el consumo de los distintos dispositivos y las diversas arquitecturas, se establecieron métricas y se registró la información en el motor de SGBD. El objetivo principal de esta experimentación fue construir una métrica personalizada denominada “índice de efectividad”. Éste permite determinar y probar que si bien las arquitecturas ARM ofrecen un rendimiento y performance menor respecto a las arquitecturas x86 tradicional, son capaces de resolver este tipo de tareas complejas consumiendo solo una pequeña fracción de la energía que utilizan sus contrapartes.

Para llegar a la construcción de la misma, se tuvieron en cuenta dos métricas iniciales:

- Métricas de tiempo
- Métricas de consumo energético

Métricas de Tiempo:

Esta métrica registra el tiempo de procesamiento que consume cada uno de los nodos trabajadores a la hora de realizar un trabajo de compresión de video. Esta fue utilizada en estudios previos [LAG14][KIP11], y se encarga, por un lado, de iniciar una marca de tiempo al comenzar con el trabajo, y por otro, de registrar el cese de operaciones, calculando finalmente el valor a través de la diferencia en ambas marcas de tiempo (*IndTime).* En particular, se almacenan los siguientes valores:

- Mínimo (*MinIndTime*).
- Máximo (*MaxIndTime*).
- Promedio (*AVGIndTime*).

Todas las métricas fueron registradas con la unidad de tiempo milisegundos (ms).

Métricas de consumo energético:

A la hora de evaluar el consumo energético (potencia) de un dispositivo trabajador, se registró dicha métrica a lo largo de la ejecución de cada tarea de procesamiento intensivo, siguiendo las definiciones utilizadas en estudios previos [ASK14][JAR13][WAN11]. Esta métrica fue utilizada en estudios previos. Todas las evaluaciones de consumo energético realizadas fueron registradas utilizando la unidad de potencia (Watts).

En particular, se almacenan los siguientes valores:

- Mínimo (*MinWattConsumption*)
- Máximo (*MaxWattConsumption*)
- Promedio (*AVGWattConsumption*)

Cabe destacar que:

- Para la medición de energía de una arquitectura x86 tradicional se tuvieron en cuenta todos sus componentes, excepto el monitor (en los casos de equipos de escritorio). Esto se debe a que este elemento no participa en las actividades de procesamiento.
- Para la captura de las métricas fue necesario desarrollar herramientas personalizadas. Los instrumentos utilizados se detallan en el Anexo II - Dispositivos de medición de consumo energético.

Índice de efectividad:

El índice de efectividad, cómo se detalló previamente, surge de la métrica de tiempo y de consumo energético e indica la relación entre el rendimiento y el consumo de los dispositivos involucrados.

Para ello, se toman los promedios por tarea y por arquitectura de las siguientes métricas:

- Tiempo de ejecución (*AVGIndTime*).
- Consumo energético (AVGWattConsumption).

Y se calcula el índice de efectividad a partir de la siguiente relación de fórmulas:

- ***ProcessingTime*** = *AVGIndTime (Worker ARM) / AVGIndTime (Worker x86)*

  Lo cual indica cuantas veces más lento es el dispositivo ARM respecto del equipo x86.

- ***EnergyConsumption*** = *AVGWattConsum (Worker x86) / AVGWattConsum (Worker ARM)*

  Lo que muestra cuántas veces menos energía consume el dispositivo ARM respecto del equipo x86.

Finalmente, al obtener las dos métricas previamente definidas se procede a calcular el índice de efectividad.

- ***Índice de Efectividad*** = *EnergyConsumption / ProcessingTime*

Donde,

  - si = 1, implica un equilibrio entre arquitecturas.
  - si < 1, la relación rendimiento/consumo es mejor en X86.
  - si > 1, la ganancia en relación al rendimiento/consumo es superior en la arquitectura ARM.

Ejecución de pruebas y análisis de resultados

Una vez definido el equipamiento, los parámetros y las métricas de evaluación, se ejecutaron los escenarios de pruebas definidos. El análisis de los resultados obtenidos se detalla a continuación:

Análisis de rendimiento:

Se comprobó que ambas arquitecturas completaron correctamente las tareas de compresión (4k, HD, 720p, 480p) en las configuraciones de fragmentos utilizadas (1 y 3 segundos) ya que sus capacidades de procesamiento y memoria disponible, incluido el espacio de intercambio (swap), son suficientemente grandes.

Las tareas de transcodificación de video que involucraron al perfil 4K, independientemente del archivo y de la duración del fragmento, provocan la actividad más compleja de resolver. Cómo se puede ver en las figuras 5.1, 5.2 y 5.3, existe un gran salto en el tiempo de procesamiento en comparación con los otros presets (480p, 720p, 1080HD). Esto se debe a que este perfil incluye:

- Un salto de categoría de compresión de 4 veces respecto del perfil anterior (4K = 1080HD x 4).
- Una tasa de bits que es 4 o 5 veces más alta que el resto (≈ 15 MB).
- Uso del más alto perfil disponible en H.264 (x264 high L@5.1).
- Definición de requisito de precisión más profundo (preset = very slow).

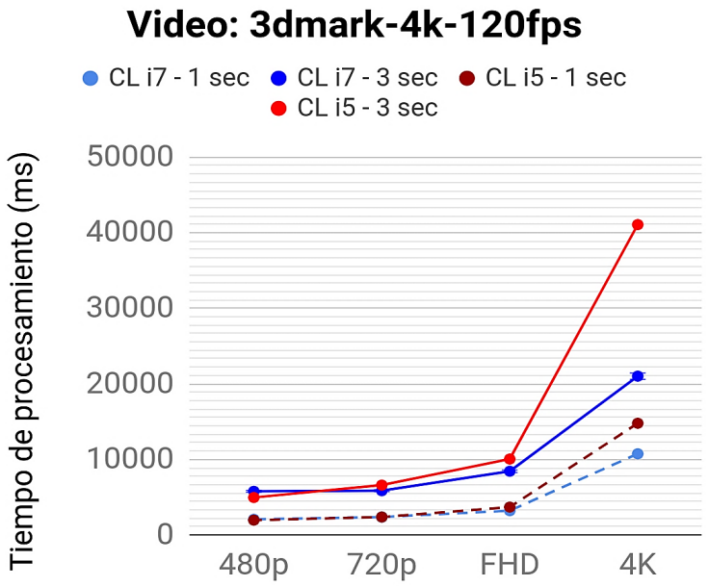

Figura 5.1 - Tiempos de compresión obtenidos para los fragmentos de 1 y 3 segundos del video 3dmark-4k-120fps en los 4 perfiles de compresión definidos, utilizando las arquitecturas de procesamiento establecidas. Fuente: Propia.

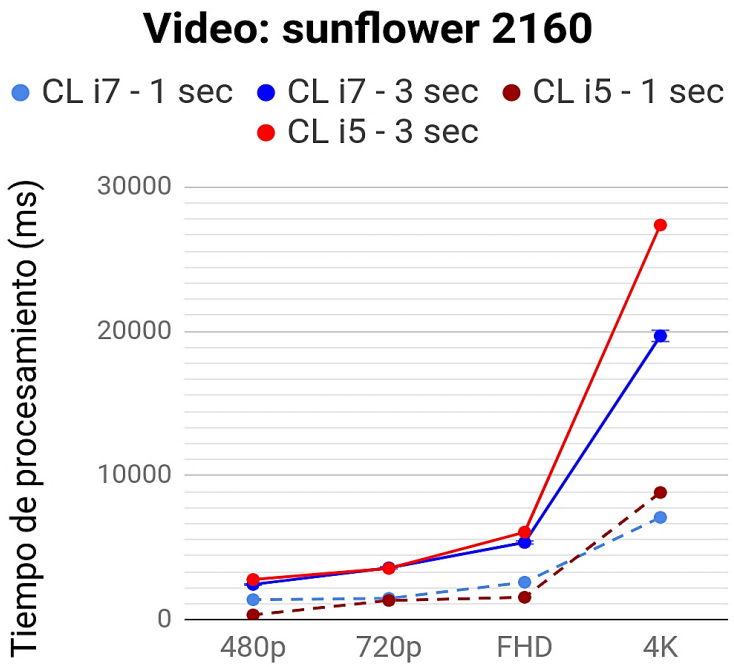

Figura 5.2 - Tiempos de compresión obtenidos para los fragmentos de 1 y 3 segundos del video sunflower-2160 en los 4 perfiles de compresión definidos, utilizando las arquitecturas de procesamiento establecidas. Fuente: Propia.

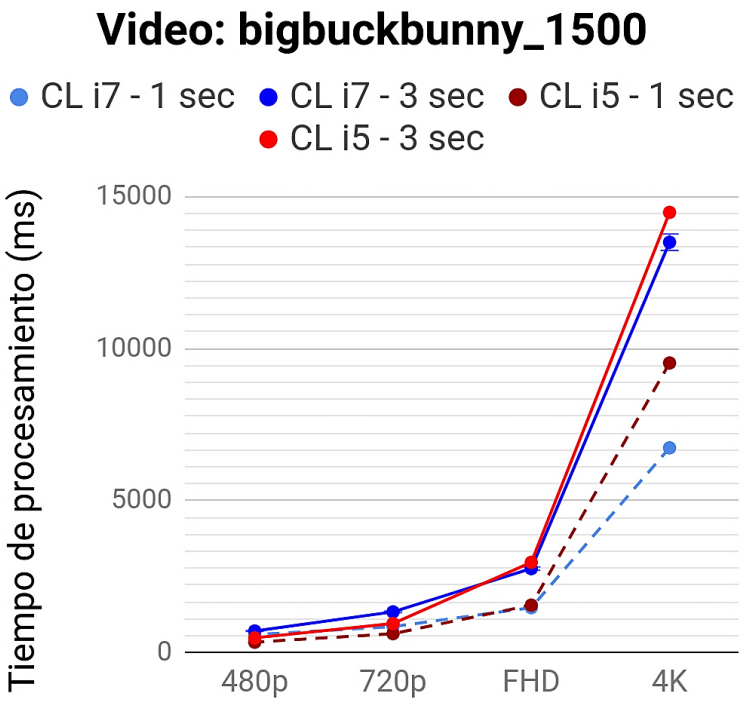

Figura 5.3 - Tiempos de compresión obtenidos para los fragmentos de 1 y 3 segundos del video bigbuckbunny_1500 en los 4 perfiles de compresión definidos, utilizando las arquitecturas de procesamiento establecidas. Fuente: Propia.

Los videos se fragmentaron en *chunks* de 1 o 3 segundos y se definieron como la unidad de análisis de procesamiento de las tareas. Cabe destacar que la duración total del video (a nivel de transcodificación de fragmentos) no es influyente, pero si lo es la complejidad del mismo (que impacta en las propiedades de cada fragmento). Según los videos fuente definidos en la tabla 5.2, el orden de complejidad de los videos es la siguiente:

- Video más complejo: 3dmark_4k.
- Video intermedio: sunflower_2160.
- Video de menor complejidad: bigbuckbunny_1500.

Lo que se condice con los tiempos obtenidos a la hora de codificar cada una de las partes de los videos (independientemente de la duración total del mismo).

Teniendo en cuenta la arquitectura más débil (x86-i5), cómo se puede ver en la figura 5.1, para el video más complejo (3dmark) el mayor tiempo de procesamiento de un fragmento de vídeo de 3 segundos con un clúster de i5 es de 0.6 minutos aproximadamente, mientras que para los videos sunflower_2160 (figura 5.2) y bigbuckbunny_1500 (figura 5.3) es de 0.45 y 0.25 minutos respectivamente. Analizando la arquitectura más potente (x86-i7), siguiendo las mismas condiciones, resulta que los datos obtenidos son 0.36 minutos para los fragmentos del video 3dmark, 0.33 para el video sunflower_2160 y 0.22 para el video bigbuckbunny_1500.

De esta manera, los resultados mostraron que la arquitectura i7 ofrece, en promedio, un rendimiento superior al 40% sobre el hardware i5 x86. Además, en base a los resultados obtenidos, se puede deducir que cuanto mayor sea la complejidad de la tarea, mejor funcionará la arquitectura i7 en comparación con la arquitectura i5 y presentará mejores resultados en cuanto a performance.

Análisis de uso de energía:

Teniendo en cuenta la ejecución de las tareas de compresión, el uso de energía de todos los componentes del equipamiento x86 (excepto el monitor), en promedio fue de:

- 115 watts para el cluster Intel i7 (37% más alto que el TDP de 84 Watts publicado de la CPU Intel) y
- 183 watts para Intel i5, (93% más alto que el TDP de 95 Watts definido por la CPU Intel). Este aumento se debe a que el i5 tiene una GPU integrada (TDP: 66W) que

ayudó en las tareas asignadas. Por esta razón, el clúster i5 consumió, en promedio, un 59% más que el i7 para todas las tareas de compresión.

Esta información puede apreciarse resumidamente en la Figura 5.4.

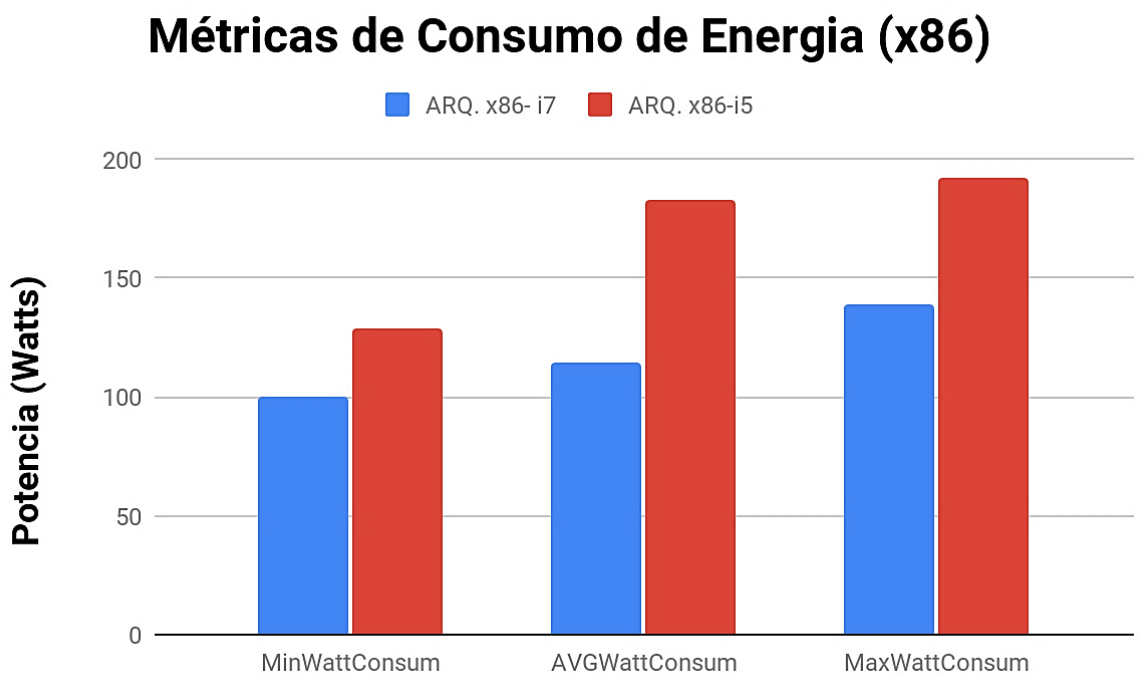

Figura 5.4 - Consumo energético (Watts) incurrido por cada arquitectura a la hora de realizar tareas de compresión de video para fragmentos de 1 y 3 segundos. Fuente: Propia.

Debido a estos resultados, tanto en lo que respecta al consumo energético como a la performance obtenida, se decidió utilizar la arquitectura i7 disponible como herramienta de comparación contra las arquitecturas ARM.

Análisis de resultados de la arquitectura ARM

Análisis de rendimiento.

El clúster de los dispositivos Samsung S7 es, en promedio, un 39% más rápido que el cluster de los equipos Samsung A5 2017 para todas las tareas de compresión, cómo se presenta en la figura 5.5. Esto coincide con el resultado esperado en función de sus especificaciones y prestaciones.

Ambas plataformas procesaron con éxito las tareas de fragmentos de video de un segundo en los cuatro perfiles (480p, 720p, 1080p, 4k), cómo se puede apreciar en la imagen de la Figura 5.5. Sin embargo, al intentar las tareas de fragmentos de video de tres segundos, las dos tareas más complejas fallaron en los cuatro perfiles (sunflower_2160 y 3dmark-4k-120fps), cómo se muestra en la Figura 5.6.

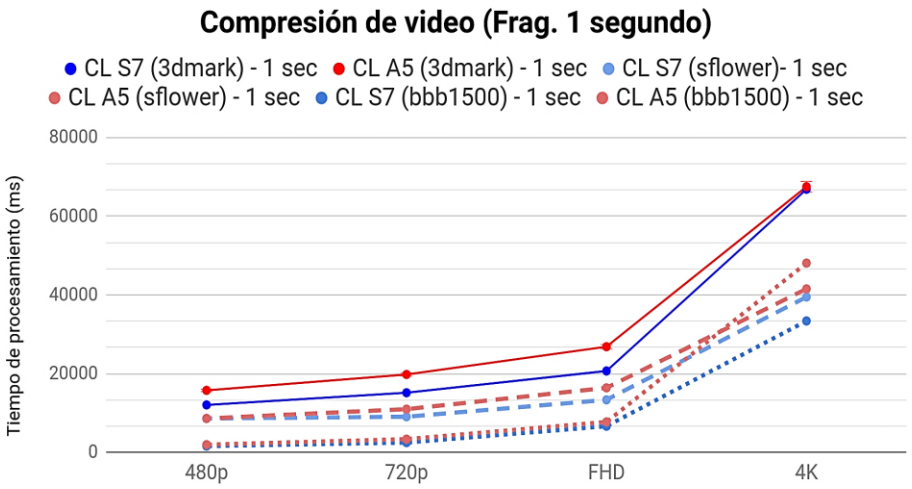

Figura 5.5 - Tiempos de compresión insumidos para las tareas de compresión de video tanto para fragmentos de 1 corriendo sobre las arquitecturas ARM-S7 cómo para ARM-i5. Fuente: Propia.

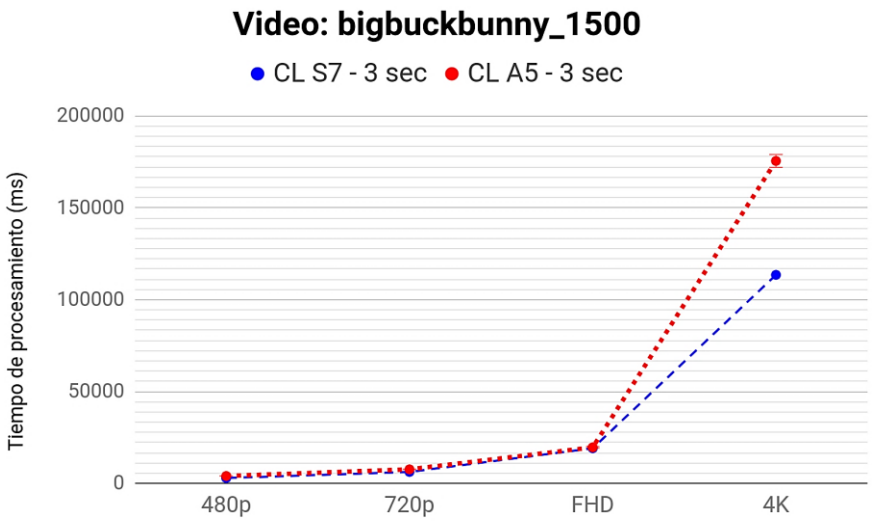

Figura 5.6 - Tiempos de compresión insumidos para las tareas de compresión sobre el video Bigbuckbunny_1500 de 3 segundos corriendo sobre las arquitecturas ARM-S7 cómo para ARM-i5. Fuente: Propia.

La causa de la falla en las tareas de fragmentos de video de tres segundos, para los videos más complejos, se debe a la incapacidad de los dispositivos móviles para asignar la cantidad necesaria de memoria que requiere la tarea, ya que los clústeres de móviles tienen mucha menos memoria RAM que sus contrapartes x86, y carecen completamente de memoria de intercambio (swap), ya que esa función no es compatible con Android.

Análisis de uso de energía.

Para la compresión de video, los resultados de la prueba muestran que el clúster Samsung S7 consume, en promedio, 3,15W, mientras que el clúster Samsung A5 2017 consume, en promedio, 2,25W. Esto significa, por un lado, que la información ofrecida por los proveedores de procesadores y dispositivos móviles en sus folletos y análisis de benchmarks resulta muy cercana a la realidad (2.5 a 5 watts). Por otro lado, se comprueba que el consumo de los dispositivos móviles es muy bajo respecto de los procesadores tradicionales x86, cómo se muestra en la Figura 5.7.

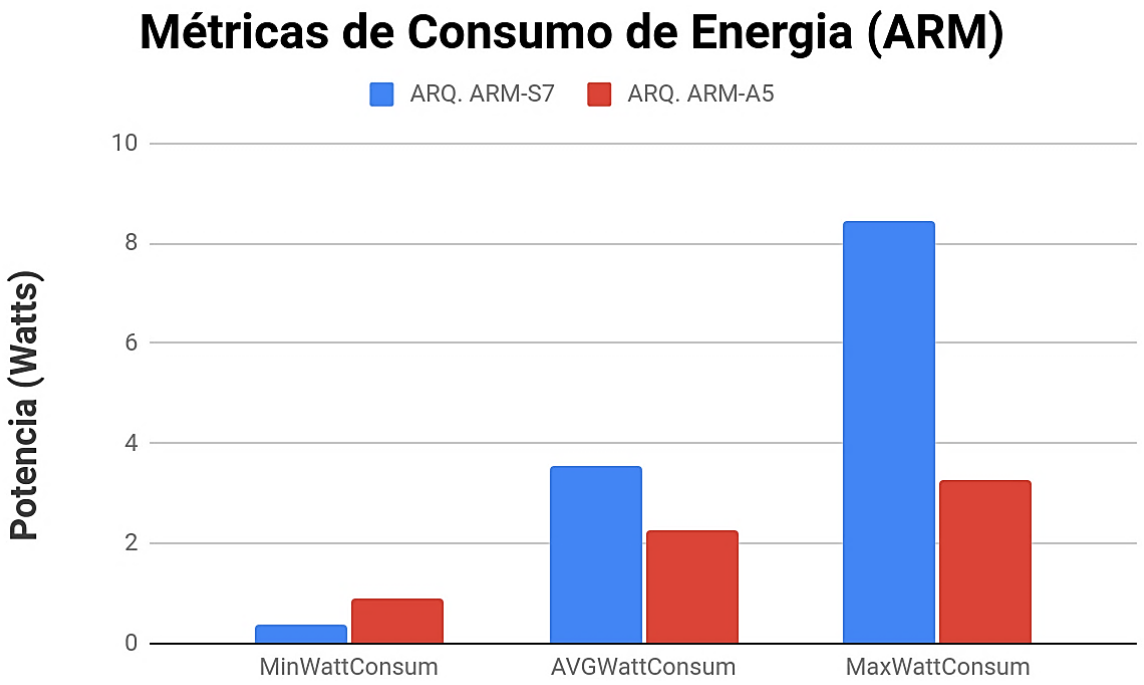

Figura 5.7 - Consumo energético incurrido por los dispositivos móviles a la hora de realizar las tareas de compresión de video. Fuente: Propia.

En particular, los dispositivos Samsung S7 (gama alta) consumen en promedio un 40% más que la plataforma de los dispositivos Samsung A5 2017 (gama media).

A la hora de analizar comparativamente los dispositivos Samsung S7 y los equipos Samsung A5 2017 resulta que:

- La capacidad de procesamiento del cluster S7 es aproximadamente un 40 % más potente.
- El consumo energético es, también, en general un 40% más.

Por lo tanto, si se emparejan las métricas de rendimiento y uso de energía, los resultados obtenidos en términos de rendimiento/potencia son similares.

Para el análisis comparativo entre arquitecturas x86 y ARM se definieron las siguientes reglas y condiciones:

- Debido a que los resultados indican que el cluster x86-i5 ofrece un menor rendimiento y un mayor consumo energético respecto del conjunto de nodos x86-i7, este último (x86-i7) se definió como punto de comparación de arquitecturas debido a que es la plataforma x86 más adecuada tanto en consumo como en potencia.
- Sobre esa base, se contrastaron las plataformas a través del índice de efectividad con los dos grupos de dispositivos móviles, es decir, x86-i7 vs Android ARM Samsung Galaxy S7 y x86-i7 vs Android ARM Samsung Galaxy A5 2017.
- Se tuvieron en cuenta sólo las tareas de procesamiento de fragmentos de video de 1 segundo, las cuales en todos los casos y en todos los dispositivos pudieron ser resueltas sin errores.

Sobre las bases definidas, los resultados indican que la arquitectura ARM, tanto en dispositivos de gama media como de alta gama, es entre 5 y 16 veces más eficaz en comparación con la arquitectura x86, variando dicho índice de ganancia dependiendo de la complejidad de la tarea y el vídeo fuente.

Por lo general, cuanto más simple sea el video fuente y la tarea a realizar, es decir que se requiere una menor potencia de cálculo, más efectivo será el equipamiento ARM. Por el contrario, cuando los fragmentos son más complejos de resolver y requieren de una mayor capacidad de cálculo, los dispositivos ARM tardan más tiempo en lograr completar la tarea, lo que impacta directamente en la relación de la métrica de ganancia obtenida. A continuación, se puede observar este efecto en la Figura 5.8.

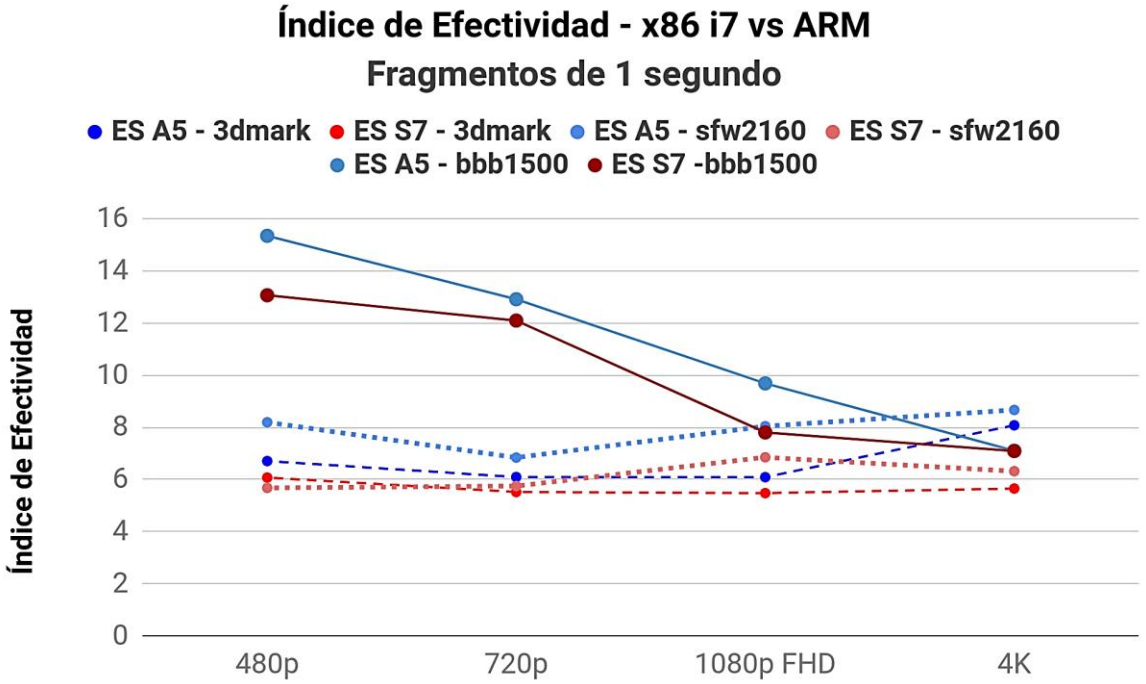

Figura 5.8 - Índice de efectividad obtenido para las tareas de compresión de video sobre fragmentos de 1 segundo. Fuente: Propia.

Conclusiones

El diseño, desarrollo y ejecución de esta experimentación permitió:

- Comprobar que los dispositivos Android de gama media y alta realizan las tareas evaluadas con una ventaja competitiva de potencia sobre las arquitecturas tradicionales. Los resultados arrojaron que el consumo de los dispositivos móviles es, dependiendo de la tarea de compresión, entre 5 y 16 veces menor respecto de los dispositivos tradicionales x86.

- Validar que los dispositivos móviles cuentan con una potencia de cálculo considerable para resolver tareas HPC. En particular los nodos móviles de rango medio, los cuales arrojaron los puntajes de eficacia más altos, no son los más rápidos, pero sin embargo son competitivos a nivel de consumo energético.

- Realizar una primera validación de la robustez de la plataforma desarrollada al soportar una variedad de escenarios de tareas de compresión de video.

Sobre estos resultados, fue posible definir las bases necesarias para mejorar las características y componentes de la arquitectura, y ejecutar nuevas experimentaciones donde sea factible:

- Solventar los problemas de saturación de recursos para poder integrar así cualquier gama de nodos móviles trabajadores, independientemente de su capacidad de cálculo.
- Probar, en detalle, las capacidades de la plataforma para responder ante cargas de trabajos concurrentes y escalables en diversas condiciones.

Estas mejoras y experimentaciones se presentan detalladamente en el siguiente apartado de pruebas (5.2.2 - Evaluación de escalabilidad y flexibilidad de la plataforma).

Por último, en trabajos futuros podría repetirse la experimentación, en diversas ocasiones y con diferente hardware, de manera de ir evaluando el comportamiento entre estas arquitecturas y definir una tendencia a lo largo del tiempo.

5.2.2 - Evaluación de escalabilidad y flexibilidad de la plataforma

Este apartado de pruebas fue diseñado una vez que se incorporaron las mejoras en el sistema y los componentes. Para ello, se tomó la arquitectura original [PET19][PET17] y se migró a una arquitectura de microservicios, lo que permitió que sus funciones fueran fragmentadas en servicios más pequeños y con responsabilidades mejor definidas [HAS20]. A su vez, se integró una arquitectura de contenedores [WAT20][VAY19], específicamente Docker [RAD17]. Finalmente, con objeto de crear clústeres de contenedores que se puedan administrar, desplegar, automatizar y escalar de manera integral, se incluyó el orquestador Kubernetes (K8s), a nivel local [DOB20][JHA19][ARU19], lo que garantiza un uso eficiente de los recursos, costos y energía [ZHO20][DEW19][TOW19].

Las pruebas sobre el sistema, a diferencia del apartado anterior donde se analizó el consumo energético y performance individual de los dispositivos, se diseñaron con objeto de verificar la capacidad de la plataforma para escalar ante diversos escenarios de carga y saturación. Al igual que el estudio previo [PET19], la tarea HPC implementada fue la transcodificación de vídeo bajo demanda, de manera distribuida y paralela.

A continuación, se presenta una descripción detallada de las actividades abordadas y los resultados obtenidos al realizar esta experimentación [PET20b][PET20a].

Definición de escenarios y equipamiento de prueba

Si bien la plataforma puede ser desplegada sobre cualquier infraestructura local o Nube que soporte el orquestador Kubernetes, la evaluación y el análisis de resultados se ejecutó sobre los siguientes casos:

- Proveedores de Nube: Se desplegó, ejecutó y validó en los dos proveedores de Nube pública más grandes y más utilizados en este momento (Amazon Web Services y Microsoft Azure), con sus servicios de almacenamiento (Amazon S3 y Azure Blob Storage). En ambas nubes se utilizó Kubernetes versión 1.17.9 para validar el funcionamiento de los nodos trabajadores móviles a través de Internet, registrando los recursos de hardware y versión de Android de los dispositivos.

- Ambientes locales: Se desplegó, ejecutó y validó en una arquitectura de kubernetes sobre una estructura empresarial de Servidores tipo Blade (9 unidades Blade HP ProLiant BL460c G8), corriendo sobre un esquema de virtualización VMware + VCenter 6.5 con objeto de evitar costos durante el proceso de prueba. Cada placa blade incluye 2 procesadores Intel Xeon E5-2640 v2 (2.0 Ghz x 16 cpus), alojando una máquina virtual de 32 cores y 32 gb de ram con SO CentOS 8, Docker 19.03.12 y Kubernetes versión 1.17.9 y con una placa de red de 1 Gbps.

En lo que respecta a la ejecución de pruebas sobre los nodos trabajadores móviles Android, se realizó sobre un conjunto amplio y heterogéneo de equipos. Sin embargo, no fue posible contar con grandes cantidades de cada uno de los modelos a la hora de ejecutar las tareas HPC. Por dicho motivo, sólo se ejecutaron las pruebas de compatibilidad, estabilidad y rendimiento sobre cada uno de los dispositivos relevados.

A la hora de realizar las pruebas de carga concurrente, estrés y escalabilidad, se optó por utilizar arquitecturas virtualizadas, limitando sus recursos en base al rango de capacidades y definiciones de los dispositivos móviles actuales. Partiendo de la disponibilidad de los 9 equipos *blade* del laboratorio local, fue posible alcanzar, teniendo en cuenta la configuración más grande definida de equipo trabajador, 32 nodos para cada una de las arquitecturas definidas. De esta manera, fue posible realizar extrapolaciones y análisis de comportamiento sobre diversas condiciones.

Aún con las limitaciones mencionadas, la experimentación sobre los casos analizados se considera significativa, y los resultados son representativos para la validación de la plataforma desarrollada.

Para cumplir con los objetivos definidos en este capítulo, se construyeron diversos escenarios y configuraciones sobre equipamientos de evaluación, para resolver tareas de procesamiento de video.

Se diseñaron y ejecutaron tres escenarios de rendimiento y carga. Los dos primeros validaron la capacidad de la plataforma para escalar al realizar múltiples tareas de transcodificación.

Para ello, se definieron diversos ambientes de prueba, donde se varió el material y los nodos trabajadores, registrando los tiempos de procesamiento.

En el primero (en adelante, E.1), se utilizó una carga fija y se varió la cantidad y tamaño de los nodos trabajadores. En el segundo (en adelante, E.2), se experimentó con material variable y se fijó la cantidad y tamaño de los nodos de procesamiento. El tercero (en adelante, E.3), se diseñó y ejecutó con el objeto de verificar la flexibilidad del sistema ante solicitudes de transcodificación concurrente, registrando el tiempo de recepción y preparación de las tareas.

Los videos utilizados para ejecutar las pruebas, con el objetivo de comprobar que era posible corregir el error de sobrecarga en los nodos trabajadores móviles que había surgido en [PET19], se dividieron en fragmentos de 10 segundos [RAK20] y se transcodificaron, al igual que el estudio anterior [PET19], utilizando el códec H.264/AVC [SHI19]. En este caso, para aumentar la cantidad y diversidad de trabajos a realizar, se definieron cinco perfiles de compresión, los cuales se detallan en la Tabla 5.4. Esto permite generar videos en diferentes calidades, adecuadas para distintos dispositivos y ancho de banda [RAA17] [SAN17].

Tabla 5.4. Perfiles utilizados para la trascodificación de videos con H.264/AVC. Fuente: Propia.

Perfiles de transcodificación	Tamaño de Pantalla	Codec de Video	Bitrate de Video	Perfil de Video	Nivel del Códec	Opciones del Codec
Perfil 2K	2560x1440	AVC x264	7800 Kbps	High	L@5.1	slower
Perfil Full-HD 1080p	1920x1080	AVC x264	3900 Kbps	High	L@4.1	slow
Perfil HD 720p	1280x720	AVC x264	2000 Kbps	Main	L@4.1	medium
Perfil 480p	852x480	AVC x264	900 Kbps	Main	L@3.1	fast
Perfil 240p	424x240	AVC x264	500 Kbps	Baseline	L@3.0	ultrafast

Para los escenarios E.1 y E.2, se definieron tres tipos de nodos trabajadores, limitando sus recursos a través de la API de Docker: low-cpu con 2 cpus y 1.5gb de memoria, mid-cpu con 4 cpus y 3gb y high-cpu con 8 cpus y 6gb, siguiendo las definiciones más tradicionales de los dispositivos móviles, corriendo sobre cinco configuraciones de clusters: 2, 4, 8, 16 y 32 nodos homogéneos, cómo se resume en la Tabla 5.5, a continuación.

Tabla 5.5 - Configuración de nodos trabajadores. Fuente: Propia.

Nodos trabajadores (Variable)					
low-cpu-dev: 2 Cores / 1.5 GBs ram	2 Workers	4 Workers	8 Workers	16 Workers	32 Workers
mid-cpu-dev: 4 Cores / 3 GBs ram	2 Workers	4 Workers	8 Workers	16 Workers	32 Workers
high-cpu-dev: 8 Cores / 6 GBs ram	2 Workers	4 Workers	8 Workers	16 Workers	32 Workers

En las pruebas E.1 y E.2 se utilizaron videos MP4 codificados en perfil High @L5.2 AVC x264, con un tamaño de pantalla de 3840x2160 (4k), con 24, 30 o 60 cuadros por segundo, obtenidos de Vimeo y Demolandia, cómo se presenta resumidamente en la tabla 5.6.

Tabla 5.6. Videos seleccionados para las cargas de trabajo (Workloads), utilizados para los escenarios de prueba. Fuente: Propia.

WK 1	High-Bitrate		WK 2	1 min 4k		WK 3	2 min 4k		WK 4	4 min 4k		WK 5	10 min 4k		WK 5	30 min 4k	
Video	Mbps	GB	Video	Mbps	GB	Video	Mbps	GB	Video	Mbps	GB	Video	Mbps	GB	Video	Mbps	GB
Norway	33	0.27	Chicago	18.6	0.13	Ghostflight	18.6	0.18	Memories	14.3	0.42	Laule	10.8	0.8	Atmos	15.8	3.37
Switzerland	19.9	0.32	Urederra	19.4	0.14	Crrousel	16.4	0.24	Humanity	15.6	0.45	Killing	11.2	0.83	Bosques	19.2	4.10
PanamaX	22.1	0.41	Portugal	19.9	0.15	Brasil	19.5	0.32	Alpine	18.3	0.54	Theresa	14.1	1	Sean	22	4.68
Landscape	125	1.68	Wonder	22	0.18	Pilot	22.2	0.33	Lake	19.9	0.56	German	14.6	1,04	longT	22.7	4.82
Europe	125	02.08	Alaska	22	0.18	Cambridge	23.2	0.34	World	22.8	0.67	Vida	21.1	1.50	Sydney	23.6	5,05

En la prueba E.1 se definió una carga de trabajo (WK) de 5 videos fuente de 1 a 2 minutos (WK 1 en Tabla 5.6) con una alta tasa de bits, donde se aplicaron los perfiles de compresión para cada video, generando así 25 tareas de compresión, cómo se detalla en la tabla 5.7 a continuación.

Tabla 5.7 - Videos utilizados para la carga de Trabajo para el escenario E.1. Fuente: Propia.

Carga de Trabajo 1 (Workload 1)						
Source Video File	Bitrate de Video	Perfil de Video	Nivel	FPS	Duración	Tamaño (GB)
UHD LG -Greece – Norway	33 Mbps	High	@L5.1	30 fps	1m 6 segs	0.27 GB
8K Switzerland cinematography	19.9 Mbps	High	@L5.2	30 fps	2m 12 segs	0.32 GB
PanamaX	22.1 Mbps	High	@L5.2	30 fps	2m 33 segs	0.41 GB
UHD LG - Landscape	125 Mbps	High	@L5.1	30 fps	1m 55 segs	1.68 GB
UHD LG - Europe	125 Mbps	High	@L5.1	30 fps	2m 20 segs	2.08 GB

En el escenario E.2 se fijaron 32 nodos trabajadores del tipo high-cpu, cómo se presenta en la tabla 5.8, a continuación; y se ejecutaron diferentes sets de videos de una duración de 1 a 30 minutos (WK 2 a WK 6 de la Tabla 5.6), que se detallan en la Tabla 5.9, también a continuación.

Tabla 5.8 - Configuración de nodos trabajadores para el escenario de carga E.2. Fuente: Propia.

Arquitectura de Nodos Trabajadores (Fija)		
Tipo de nodo (high-cpu-device)	8 Cores	6 Gb Ram
Cantidad de nodos trabajadores	32 nodes	

Tabla 5.9 - Videos utilizados en las cargas de trabajo 2 a 6 para la evaluación del escenario E.2. Fuente: Propia.

Carga de Trabajo 2 (Workload 2)						
Video fuente	**Bitrate de Video**	**Perfil de Video**	**Nivel**	**FPS**	**Duración**	**Tamaño (GB)**
Chicago in 1 Minute - Cinematic 4K	18.6 Mbps	High	@L5.2	24 fps	0m 58s	0.13 GB
Source of Urederra (4K)	19.4 Mbps	High	@L5.2	30 fps	1m 0 s	0.14 GB
4K: Portugal by Drone Flight	19.9 Mbps	High	@L5.2	24 fps	1m 0s	0.15 GB
ONE MINUTE WONDER (4k)	22 Mbps	High	@L5.2	25 fps	1m 0s	0.18 GB
Alaska Route 1 - In Another Minute	22 Mbps	High	@L5.2	30 fps	1m 0s	0.18 GB
Carga de Trabajo 3 (Workload 3)						
Video fuente	**Bitrate de Video**	**Perfil de Video**	**Nivel**	**FPS**	**Duración**	**Tamaño (GB)**
Ghostflight 4K	18.6 Mbps	High	@L5.2	24 fps	2m 7s	0.18 GB
4k carrousel	16.4 Mbps	High	@L5.2	24 fps	2m 6s	0.24 GB
4k brasil	19.5 Mbps	High	@L5.2	30 fps	2m 16s	0.32 GB
4k pilot	22.2 Mbps	High	@L5.2	24 fps	2m 3 s	0.33 GB
Cambridge [4K]	23.2 Mbps	High	@L5.2	30 fps	2m 2s	0.34 GB
Carga de Trabajo 4 (Workload 4)						
Video fuente	**Bitrate de Video**	**Perfil de Video**	**Nivel**	**FPS**	**Duración**	**Tamaño (GB)**
Max Cooper - Memories	14.3 Mbps	High	@L5.2	30 fps	4m 3 s	0.42 GB
For Humanity	15.6 Mbps	High	@L5.2	24 fps	4m 0 s	0.45 GB
Alpine - 4K	18.3 Mbps	High	@L5.2	25 fps	4m 2 s	0.54 GB
Silver Lake Sutdios 2020	19.9 Mbps	High	@L5.2	24 fps	3m 59 s	0.56 GB
Other World 1. HDR 8K macro	22.8 Mbps	High	@L5.2	30 fps	4m 4 s	0.67 GB
Carga de Trabajo 5 (Workload 5)						
Video fuente	**Bitrate de Video**	**Perfil de Video**	**Nivel**	**FPS**	**Duración**	**Tamaño (GB)**
Laule concert 360 VR 4K	10.8 Mbps	High	@L5.2	24 fps	10m 0s	0.8 GB
Killing Time	11.2 Mbps	High	@L5.1	24 fps	10m 0s	0.83 GB
Theresa & Joel 4K	14.1 Mbps	High	@L5.2	25 fps	10m 0s	1 GB
Salzburg. Brauchtum Im Herzen	14.6 Mbps	High	@L5.2	25 fps	10m 5s	1.04 GB
Mar de Vida - Full Film (4K)	21.1 Mbps	High	@L5.2	24 fps	10m 0s	1.50 GB
Carga de Trabajo 6 (Workload 6)						
Video fuente	**Bitrate de Video**	**Perfil de Video**	**Nivel**	**FPS**	**Duración**	**Tamaño (GB)**
Atmospheres - Evening Sailing	15.8 Mbps	High	@L5.2	24 fps	30m 0s	3.37 GB
Documental Bosques Relictos 4K	19.2 Mbps	High	@L5.2	30 fps	30m 23 segs	4.10 GB
Sean + Sheena // Cinematic Cut	22 Mbps	High	@L5.2	24 fps	30m 14 segs	4.68 GB
23 06 19 4k	22.7 Mbps	High	@L5.2	25 fps	30m 0s	4.82 GB
Sydney NYE Fireworks 2018	23.6 Mbps	High	@L5.2	30 fps	30m 25 segs	5.05 GB

Sobre estas configuraciones se ejecutan las tareas según la siguiente fórmula: TC = 25 x M. Dónde, 25 surge de los 5 videos fuente de cada WK (Tabla 2) multiplicado por los 5 perfiles de compresión (Tabla 5.4); y M es el multiplicador de tareas. En las pruebas ejecutadas, M varió entre 10 y 60, resultando entre 250 y 1500 tareas por cada carga de trabajo.

La prueba E.3 se llevó a cabo a través de un script bash que generó entre 250 y 1500 peticiones curl concurrentes, utilizando el video “Chicago” del set WK2 (Tabla 5.9). La cantidad de sesiones paralelas se eligió teniendo en cuenta la capacidad de la red del laboratorio de pruebas. Las ejecuciones se realizaron sobre diversas configuraciones de tamaño y cantidad de servidores web. Se definieron 3 tipos de nodos siguiendo las

configuraciones más tradicionales de las instancias de Amazon EC2 [PHA18]: a1.medium con 1 cpus y 2 gb de memoria ram; a1.large con 2 cpus y 4 gb; y a1.xlarge con 4 cpus y 8 gb, limitando sus recursos a través de la API de Docker, corriendo sobre configuraciones de 1, 2, 4 y 8 nodos web homogéneos, cómo se muestra en la tabla 5.10 a continuación.

Tabla 5.10 - Configuración de nodos servidores Web para el escenario de carga E.3. Fuente: Propia.

Nodos Servidores Web (Variable)			
a1.medium: 1 Cores / 2 GBs ram	2 Workers	4 Workers	8 Workers
a1.large: 2 Cores / 4 GBs ram	2 Workers	4 Workers	8 Workers
a1.xlarge: 4 Cores / 8 GBs ram	2 Workers	4 Workers	8 Workers

Ejecución de pruebas y análisis de resultados

Despliegue y pruebas iniciales:

Luego de desplegar la plataforma en la Nube a través de los servicios de Kubernetes de AWS y Azure, y publicar reiteradas veces las cargas definidas en WK1 (Tabla 5.7), se realizaron pruebas de compatibilidad del nodo trabajador móvil en distintos dispositivos (Tabla 5.11). Se corroboró que la aplicación funciona correctamente en equipos que tengan instalado Android 7 (lanzado en 2016) o superior (API Level 24), ya sea con arquitectura de 32 o 64 bits. También se comprobó que la inclusión de tuberías Linux en la aplicación evitó la sobrecarga de memoria, permitiendo ejecutar cada tarea sin errores, independientemente de la potencia del dispositivo. Es importante destacar este apartado, ya que:

- Los fragmentos fuente son de una gran complejidad, siempre partiendo de videos en calidad 4K.
- Su duración es de 10 segundos, superior a lo presentado en el artículo anterior [PET19] (1 y 3 segundos).

Al mismo tiempo, se probó el funcionamiento básico del sistema y sus servicios de equilibrio de carga, escalado horizontal, manejo de errores y alta disponibilidad.

Tabla 5.11 - Dispositivos móviles utilizados para corroborar el funcionamiento de la aplicación de nodo trabajador. Fuente: Propia.

Tabla Celulares Android probados					
Año	Modelo	CPU	API	Arq.	SO
2020	Xiaomi Note 10 Lite	Snapdragon 730G	29	aarch64	10
2020	Samsung Galaxy S20	Exynos990	29	aarch64	10
2019	Samsung Galaxy S10+	Exynos 9820	29	aarch64	10
2019	OnePlus 7T	Snapdragon 855+	29	aarch64	10
2019	Samsung Galaxy A20	Exynos 7884	29	aarch64	10
2019	Samsung Galaxy A30s	Exynos 7904	29	aarch64	10
2019	Xiaomi R. Note 8 Pro	Mediatek G90T	29	armv8l	10
2019	Moto G7 Play	Snapdragon 632	29	armv7l	10
2019	Samsung A70	Snapdragon 675	29	armv8l	10
2019	Moto G8 Plus	Snapdragon 665	28	aarch64	9
2019	Motorola E6	Snapdragon 435	28	armv7l	9
2019	Samsung A50	Exynos 9610	28	armv8l	9
2019	Xiaomi Redmi Note 7	Snapdragon 660	28	aarch64	9
2018	Samsung A6+	Snapdragon 450	29	armv8l	10
2018	Samsung J8	Snapdragon 450	29	armv8l	10
2018	Samsung Galaxy Note	Exynos 9810	29	aarch64	10
2018	Moto One	Snapdragon 625	29	aarch64	10
2018	Moto G6 Plus	Snapdragon 630	28	aarch64	9
2018	Moto E5	Snapdragon 425	26	armv7l	8
2017	Google pixel 2 XT	Snapdragon 835	29	aarch64	10
2017	Moto G5 Plus	Snapdragon 625	29	armv7l	10
2017	Samsung J7 pro	Exynos 7870	28	armv8l	9
2017	Samsung S8	Exynos 8895	28	aarch64	9
2017	Samsung Galaxy A5	Exynos 7880	26	aarch64	8
2017	Moto E4 Plus	Mediatek MT6737	25	armv7l	7.1
2016	Samsung Galaxy S7	Exynos 8890	26	aarch64	8.1
2016	Samsung J7 Prime	Exynos 7870	27	armv8l	8.1
2016	Samsung Gal J5 Prime	Exynos 7570	26	armv8l	8.0
2016	Moto G4 Plus	Snapdragon 617	24	armv7l	7.0
2015	Samsung S5 New Ed.	Exynos 7580	23	armv7l	6.0
2015	Moto G3	Snapdragon 410	23	armv7l	6.0
2014	Samsung Grand Prime	Snapdragon 450	21	armv7l	5.0
2013	Sony Xperia Z1	Snapdragon 800	22	armv7l	5.1

Análisis de resultados para escenario E.1:

El rendimiento obtenido estuvo relacionado con las capacidades de cada categoría de nodos y con la complejidad del perfil aplicado. En todos los casos, los fragmentos de video se transcodificaron en menos de 2 minutos, cómo puede observarse en la figura 5.9.

También puede notarse que las tareas de transcodificación de video que involucraron al perfil 2K, independientemente del video fuente utilizado, provocaron la actividad más compleja a resolver. Cómo se puede ver en la Figura 5.9, requiere al menos del doble de tiempo respecto de los otros *presets* (240p, 480p, 720p, 1080HD). Esto se debe a que este perfil incluye:

- Un salto de categoría de 2 veces (2K = 1080HD x 2).
- Una tasa de bits al menos 2 veces más alta que el resto (≈ 8 MB).
- Uso del más alto perfil disponible en H.264 (x264 high L@5.1).
- Definición del requisito de precisión más profundo (preset = slower).

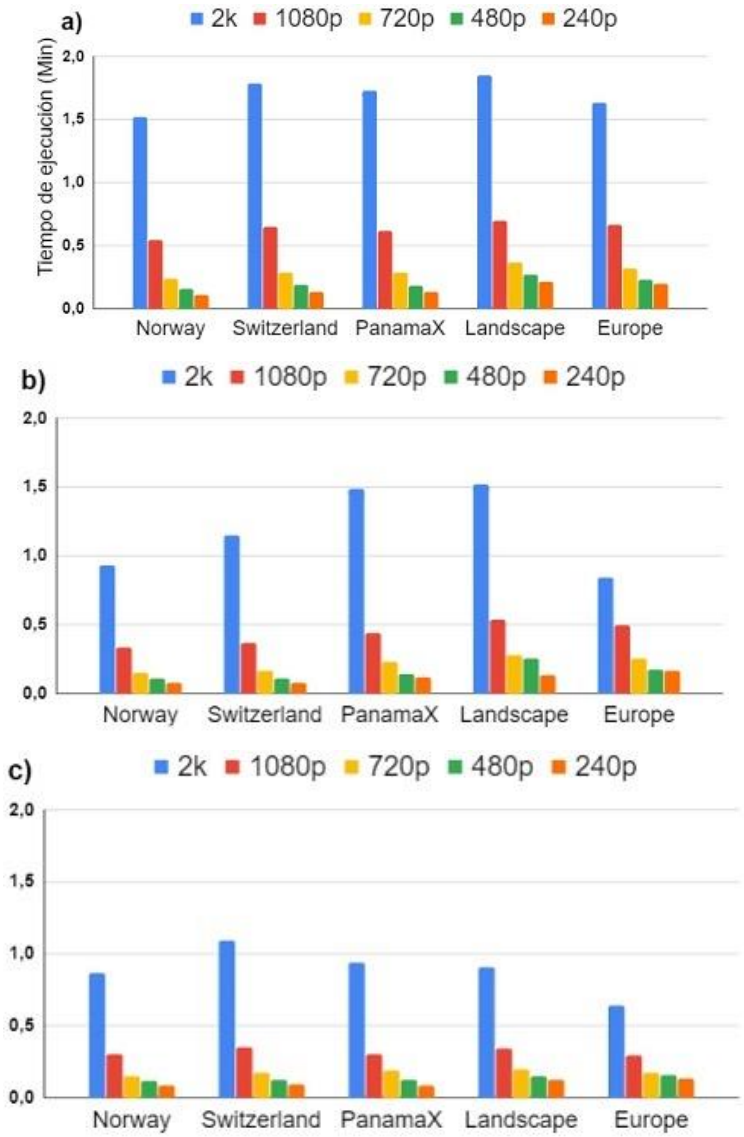

Figura 5.9. Tiempo de transcoding (minutos) para videos WK1 en nodos: a) low-cpu; b) mid-cpu; c) high-cpu. Fuente: Propia.

Considerando el perfil más complejo (2k), los resultados indican que al aumentar la capacidad de los dispositivos trabajadores de low a mid-cpu, los tiempos de ejecución se reducen en un 31%. Esto mismo se observó al incrementar de low-cpu a high-cpu, donde la reducción alcanza en promedio, un 47%, cómo se presenta en la figura 5.10.

Estos resultados permiten:

- Validar que independientemente del dispositivo las tareas pudieron ser resueltas.
- Que, si bien no es una tendencia de mejora lineal, igualmente se constata que a mayor potencia de cálculo se obtiene una mejor performance.

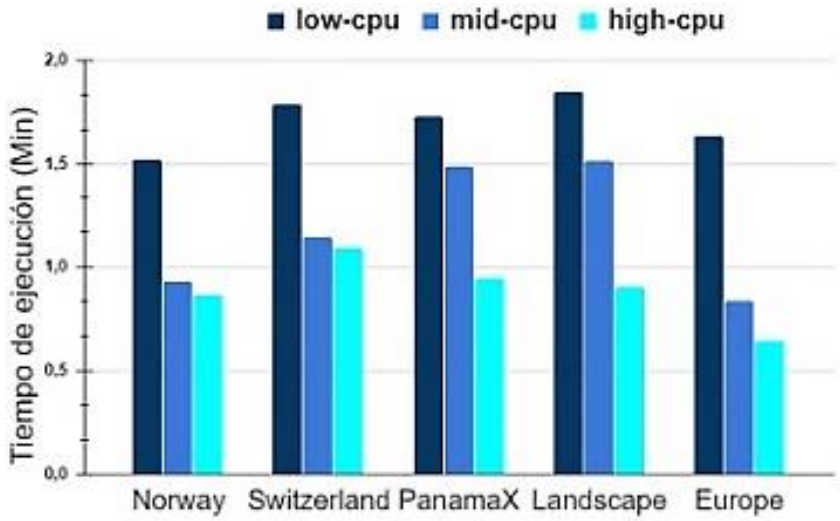

Figura 5.10. Tiempo de ejecución promedio por tipo de nodo. Fuente: Propia.

Con respecto a los tiempos totales de ejecución sobre las diversas topologías de clústeres, se advierte que, al duplicar la cantidad de nodos el tiempo insumido de compresión se reduce en promedio entre un 40% y 45%, por cada salto.

Si se compara la configuración inicial de 2 nodos con la de 32 colaboradores, el tiempo necesario se reduce de 8 a 10 veces, cómo se presenta en la figura 5.11.

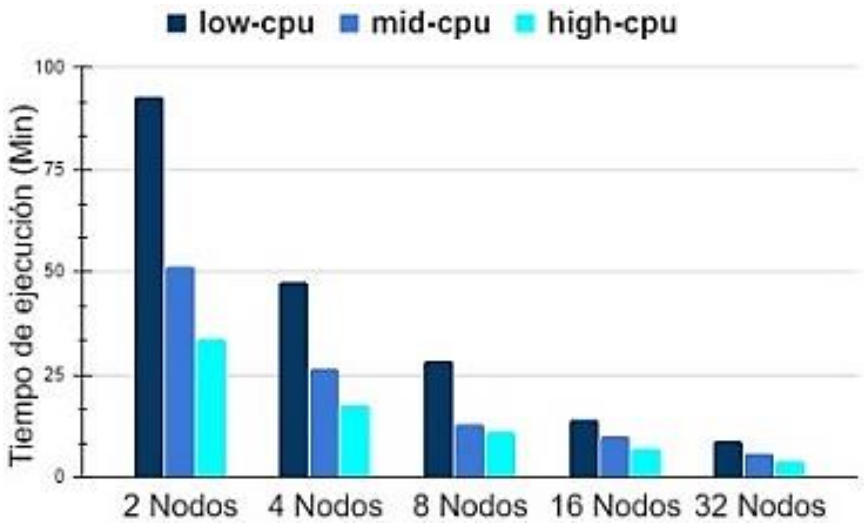

Figura 5.11. Tiempo total para diferentes cantidades de nodos. Fuente: Propia.

Esto permite corroborar los siguientes aspectos:

- Que el agregado de nodos trabajadores aporta a disponer de una mayor capacidad de cálculo paralelo y ayuda a resolver las tareas HPC independientes que la plataforma dispone en cola más rápidamente.
- Que independientemente de la cantidad de nodos trabajadores, las tareas se resuelven sin problemas, sólo viéndose afectada la performance del sistema.

Análisis de resultados para escenario E.2:

A través de los escenarios definidos, se verificó que la plataforma es flexible y escalable ante cargas de trabajo crecientes. Se probó que al mantener la cantidad de nodos de procesamiento fija y aumentar el número y la duración de los vídeos (más partes a procesar), se incrementa el tiempo total de procesamiento, cómo se puede observar en las figuras 5.12 a 5.16.

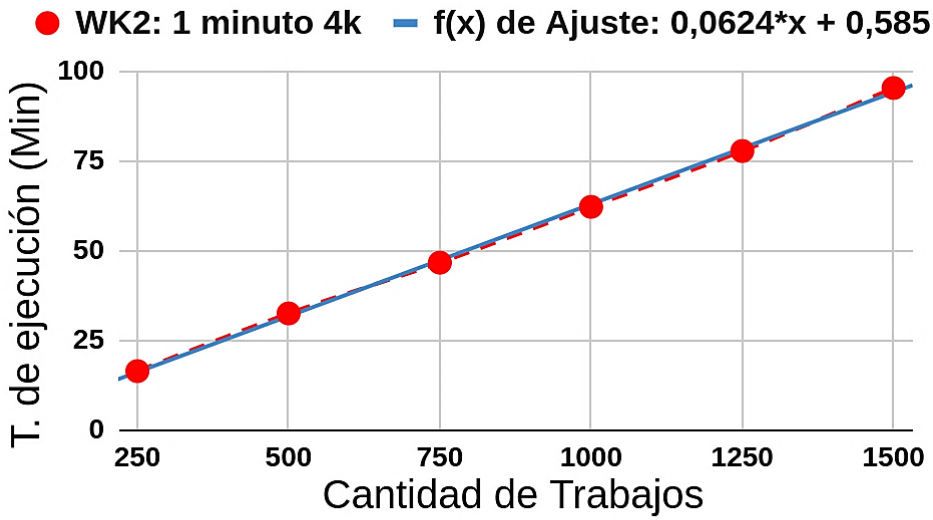

Figura 5.12 - Tiempos de ejecución (en minutos) resultante para la carga de trabajo WK2 - Fuente: Propia.

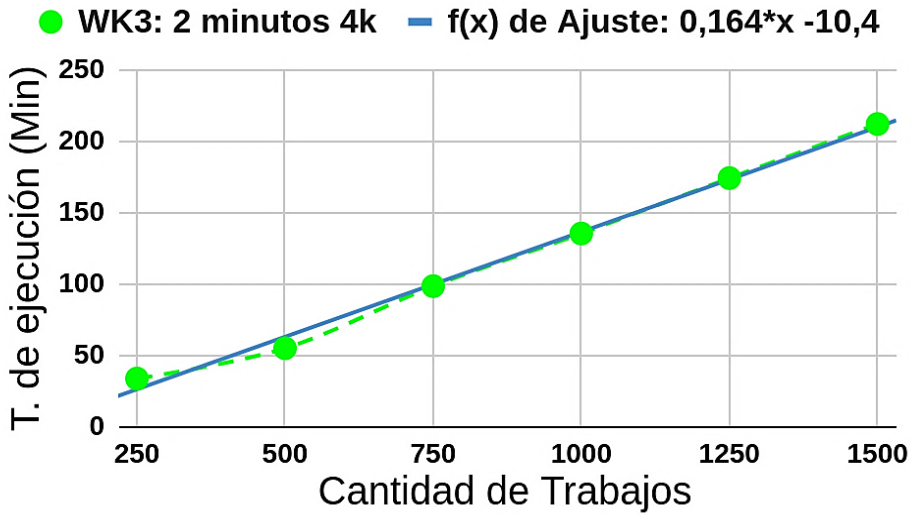

Figura 5.13 - Tiempos de ejecución (en minutos) resultante para la carga de trabajo WK3 - Fuente: Propia.

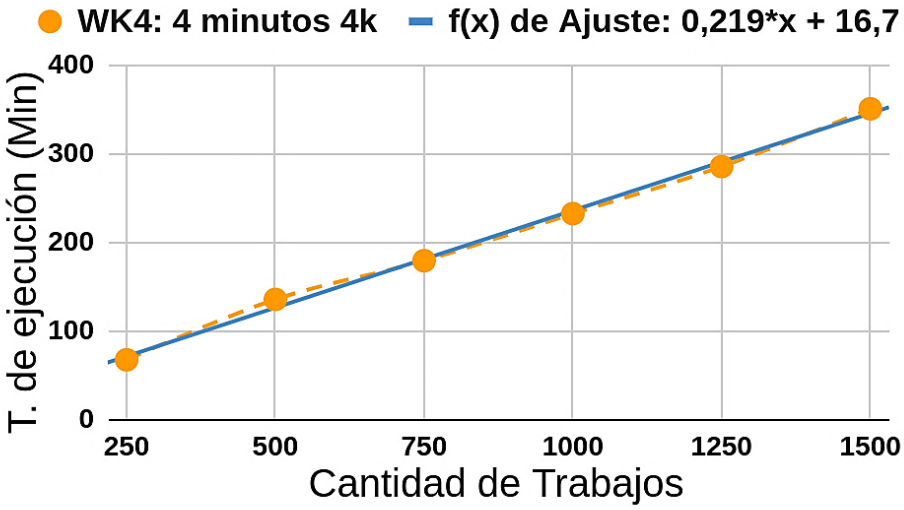

Figura 5.14 - Tiempos de ejecución (en minutos) resultante para la carga de trabajo WK4 - Fuente: Propia.

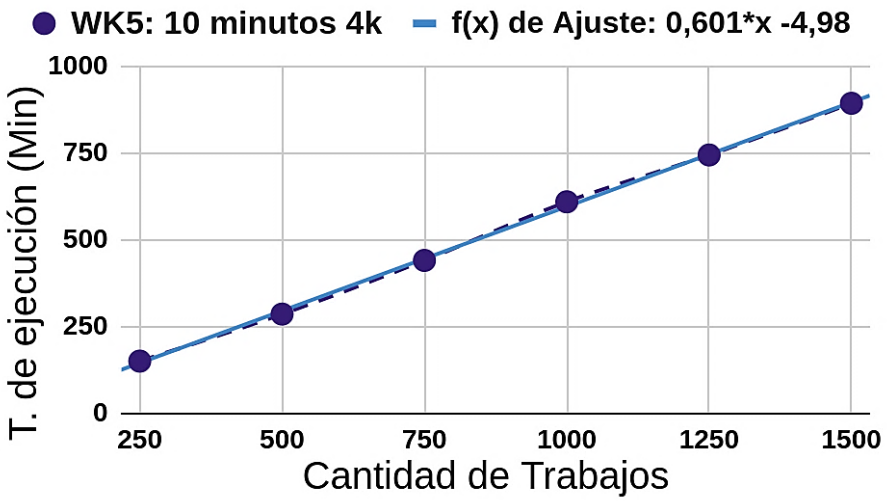

Figura 5.15 - Tiempos de ejecución (en minutos) resultante para la carga de trabajo WK5 - Fuente: Propia.

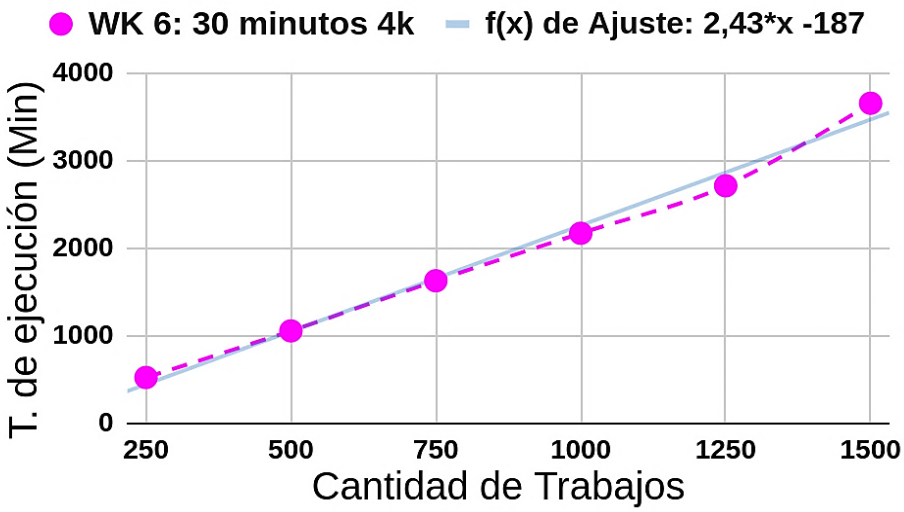

Figura 5.16 - Tiempos de ejecución (en minutos) resultante para la carga de trabajo WK6 - Fuente: Propia.

Con el objeto de caracterizar y poder extrapolar los resultados, se ajustó cada ejecución con una recta de tendencia de carácter lineal según f(x) = a*x+c, donde “a” es la pendiente y “c” la constante. Como resultado, esto permitió validar que:

- A medida que se complejizan los escenarios de prueba y por lo tanto se genera un acumulamiento de fragmentos a procesar con una capacidad fija, el valor de la pendiente “a” se vuelve mayor, impactando directamente en la performance y tiempos de respuesta, cómo se puede apreciar en la Figura 5.17.
- A pesar de que bajo estas condiciones la performance se ve afectada, se comprobó que la plataforma responde adecuadamente y es transparente a la carga de trabajo.

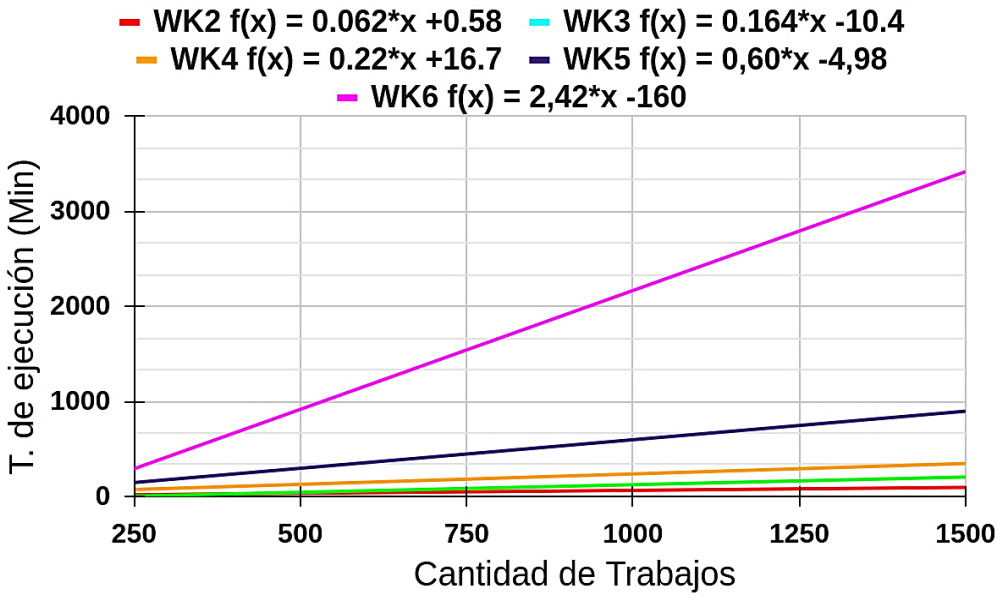

Figura 5.17 - Rectas de Ajuste para los tiempos de ejecución de las tareas de compresión WK2-6. Fuente: Propia.

En la figura anterior (Figura 5.17) no se alcanza a percibir con detalle la diferencia entre las rectas de ajuste sobre las cargas de trabajo WK2 a WK4. Por tal motivo, y con objeto de ayudar al lector a una comprensión de dicha información se presenta, a continuación, en la figura 5.18 la misma gráfica, pero representando el tiempo de ejecución (eje Y) en formato logarítmico.

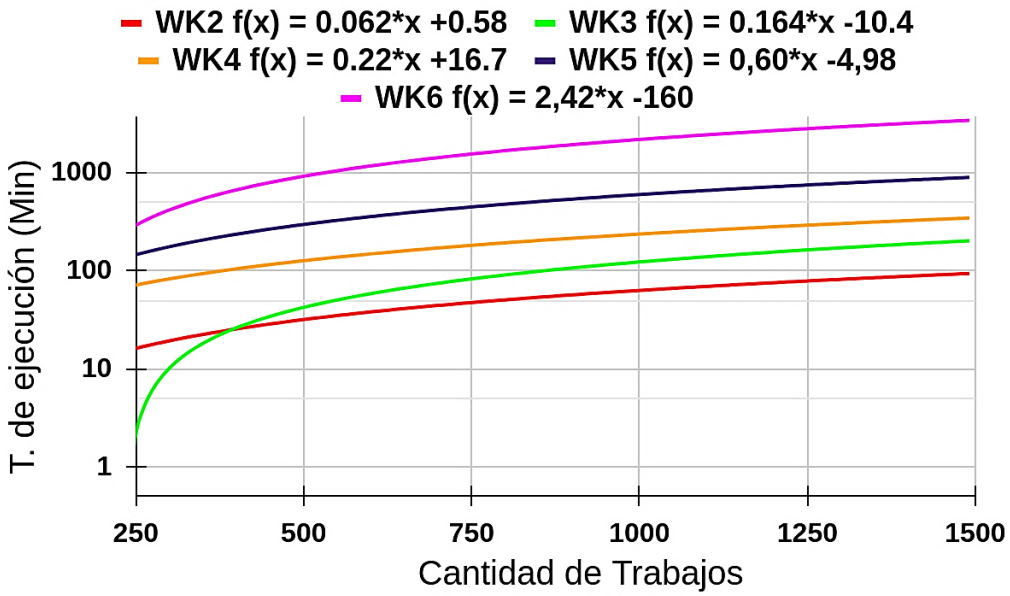

Figura 5.18 - Rectas de Ajuste para WK2-6. Escala de eje Y en formato logarítmico. Fuente: Propia.

Análisis de resultados para escenario E.3:

A través de este escenario de experimentación se validó que el sistema responde correctamente a las peticiones concurrentes de transcodificación de video establecidas. El tiempo total de recepción y preparación de los trabajos estuvo relacionado con las capacidades de la red y las características y cantidades de nodos servidores, cómo se muestra en la Figura 5.19.

Se probó que la complejidad de los procesos de lectura del buffer de comunicación, la conversión del flujo a un archivo Transport Stream (.ts), su fragmentación a partes de 10 segundos, y comunicación con sistema de cola, base de datos y sistema de archivos, todo de manera concurrente: conllevan a un uso intensivo de CPU, memoria y disco.

Si bien los nodos a1.medium atendieron a las solicitudes concurrentes sin errores, se consideran una arquitectura poco escalable, debido a los tiempos de respuesta obtenidos y a sus pendientes de crecimiento pronunciadas, al menos para las configuraciones de nodos utilizadas. En todos los casos, con esta configuración, las tareas realizadas consumieron más de 50 minutos (Figura 5.19 gráfica a). Por el contrario, los casos a1.large y a1.xlarge, con

arquitecturas de al menos 2 nodos se consideran lo suficientemente robustas ante una demanda concurrente de tareas; ya que responden siempre por debajo del umbral de los 50 minutos y tienen líneas de crecimiento más controladas (Figura 5.19, gráficas b y c)

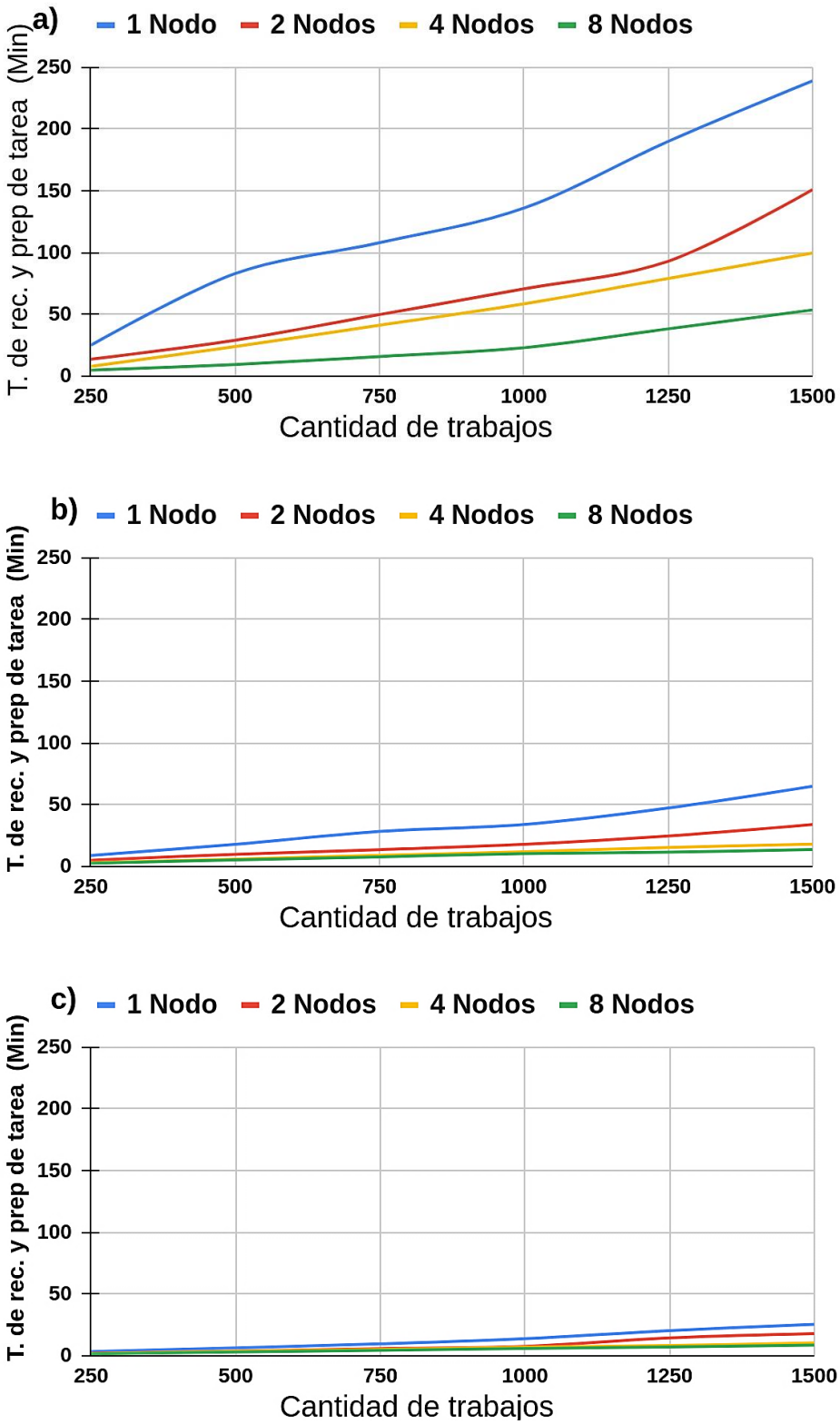

Figura 5.19 - Tiempos de recepción y preparación de tareas a través de diversas configuraciones de Nodos Web Server. a) a1.medium ; b) a1.large; c) a1.xlarge. Fuente: Propia.

En la figura anterior (Figura 5.19) la diferencia entre los tiempos insumidos para las diversas configuraciones de nodos servidores web, no se puede percibir con detalle, debido al uso de una misma escala para el eje Y. Por tal motivo, y con objeto de ayudar al lector a la comprensión de los datos, se presenta a continuación, en la figura 5.20, las mismas gráficas, pero representando el tiempo de ejecución (eje Y) en formato logarítmico.

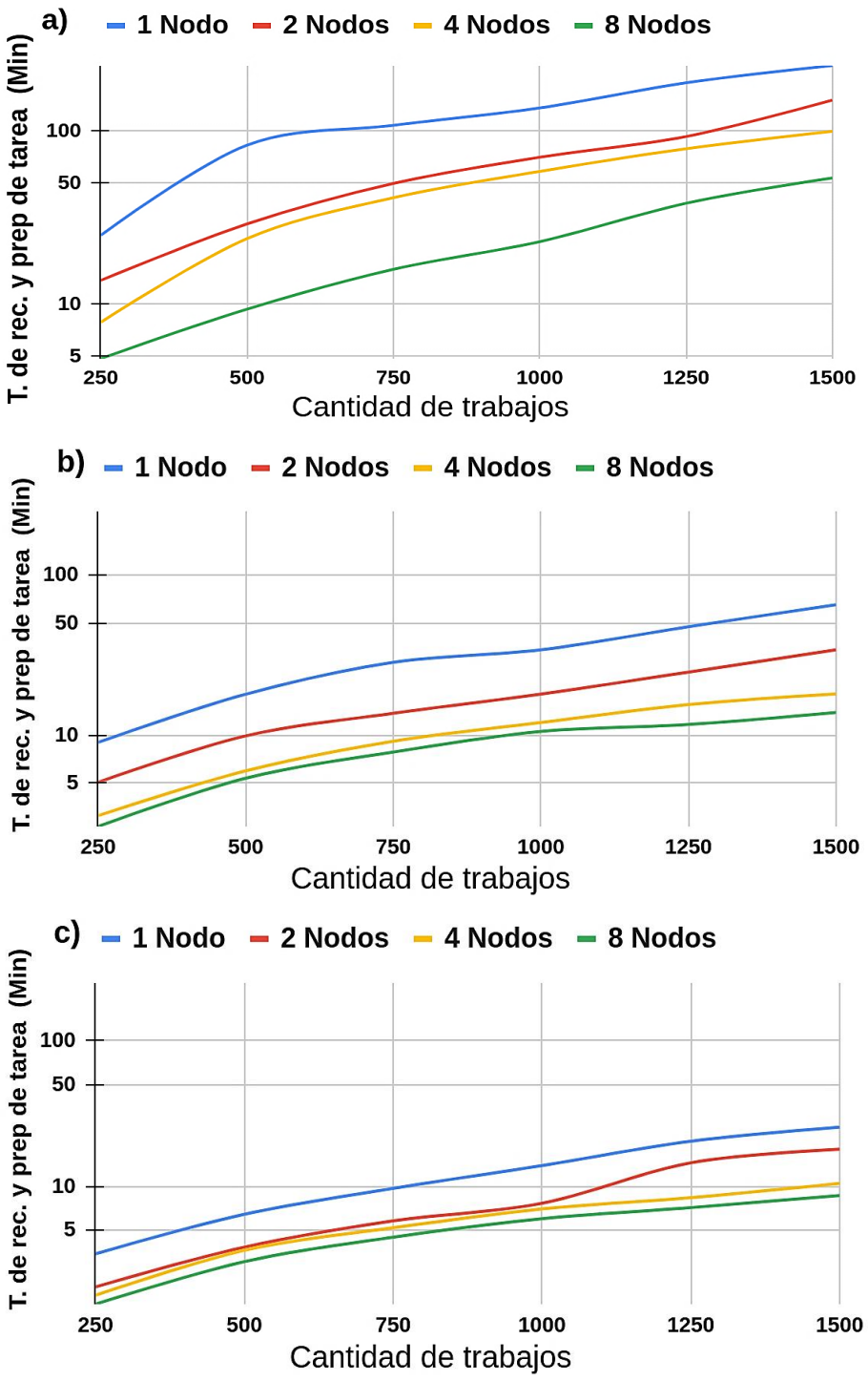

Figura 5.20 - Tiempos de recepción y preparación de tareas (Escala de eje Y en formato logarítmico) a través de diversas configuraciones de Nodos Web Server. a) a1.medium ; b) a1.large; c) a1.xlarge. Fuente: Propia.

Conclusiones

En este trabajo de experimentación se desplegó y evaluó la plataforma colaborativa, distribuida, escalable y de bajo costo; basada en microservicios, contenedores orquestados por Kubernetes y servicios en la Nube, la cual reutiliza capacidad ociosa de los dispositivos móviles y x86 para realizar tareas informáticas HPC.

A partir de los resultados obtenidos, se observó que los dispositivos móviles ARM tienen la capacidad de procesamiento suficiente para ser utilizados como una arquitectura viable a la hora de resolver este tipo de necesidades, con un costo y consumo energético mucho menor respecto a las arquitecturas tradicionales x86.

Además, a través de la experimentación, se probó sobre tareas de transcoding de video, que el sistema: a) responde de manera elástica y flexible ante diversas cargas de trabajo; y b) que resuelve las tareas de procesamiento de forma concurrente sin importar el volumen de carga y la cantidad y potencia de los nodos trabajadores.

Se evaluó también que c) la aplicación de procesamiento móvil es compatible con cualquier dispositivo que tenga Android 7 o superior instalado; y d) que, si bien la performance está relacionada con la capacidad del equipo, la inclusión de tuberías Linux reduce la cantidad de memoria necesaria, permitiendo ejecutar cada tarea sin errores, independientemente de su potencia.

6. Conclusiones

En este trabajo se desarrolló y evaluó una plataforma HPC donde las tareas de procesamiento de cómputo intensivo se ejecutan reutilizando, estratégicamente, las capacidades ociosas de los dispositivos móviles. Estos equipos, integran procesadores ARM de bajo consumo energético los cuales, en los últimos años, han incrementado su capacidad, eficiencia, estabilidad y potencia, así como también masividad y mercado.

Con el objetivo de transformar estos recursos en un centro de datos de procesamiento intensivo, la plataforma se diseñó utilizando herramientas, metodologías y prácticas DevOps, microservicios y contenedores orquestados (Kubernetes) que permiten desplegarla de manera transparente tanto en entornos de la Nube, como locales. De esta manera, se descentralizan las responsabilidades, y así se evita depender solamente de grandes y costosos equipos. Al utilizar hardware masivo, distribuido, de bajo costo y consumo energético, que además ya se encuentra disponible, la cantidad de energía, requisitos de refrigeración y espacio físico de servidores se reducen drásticamente. De esta manera, el sistema mantiene un costo reducido y garantiza características de elasticidad y flexibilidad, al tiempo que gestiona y explota adecuadamente la capacidad de procesamiento disponible.

Una vez construida y desplegada la plataforma, a los fines de validar el funcionamiento, rendimiento, escalabilidad y flexibilidad del sistema, se decidió implementar el servicio de transcoding de video bajo demanda como tarea HPC. Sobre esta base se plantearon y ejecutaron varios escenarios de evaluación y pruebas. Para todos los casos, se definieron las métricas a evaluar, las condiciones de los escenarios y el equipamiento a utilizar. Se especificó, además, un conjunto de vídeos fuentes en alta calidad (8K y 4K) con diversas duraciones, perfiles de codificación a aplicar, procesos de fragmentación (1, 3 y 10 segundos), y tareas de unificación del material audiovisual resultante.

A través de la experimentación y los resultados obtenidos, se corroboró que los dispositivos móviles presentan una potencia de cálculo relevante para resolver tareas HPC y una ventaja competitiva en lo que respecta al consumo energético. De acuerdo a nuestros resultados los equipos Android ARM consumen, en promedio, entre 30 y 40 veces menos energía que un dispositivo x86, y por lo tanto realizan, las actividades evaluadas con un índice de efectividad, dependiendo de la tarea en ejecución, de entre 5 y 16 veces menor respecto de los dispositivos tradicionales x86.

Además, se probó que la plataforma proporciona un uso eficiente de los recursos administrados ya que permite optimizar y flexibilizar la capacidad requerida dependiendo de

la carga del sistema. Los distintos escenarios analizados muestran una reducción efectiva de tiempo de ejecución cuando se agregan nodos de procesamiento, como así también la capacidad de escalar a medida que aumenta la carga de trabajo.

Por lo tanto, se concluye que la plataforma es lo suficientemente robusta para presentar este sistema como una alternativa escalable, de bajo costo y consumo energético para resolver tareas de cómputo intensivo cómo la analizada con sus variantes.

Finalmente, el sistema desarrollado se pone a disposición de la comunidad. Podrá ser extendido, adaptado y utilizado, cómo se realizó con el ejemplo de la transcodificación de vídeo, a otros ámbitos de ejecución de tareas HPC. Aquellos interesados pueden acceder a él a través del siguiente link de GitHub: https://github.com/dpetrocelli/PhD-Mobile-K8s-HPC-Platform

7. Trabajos Futuros

A partir de la realización de este trabajo surgen diferentes líneas de investigación para mejorar las características y los servicios ofrecidos por la plataforma desarrollada. A continuación, se detallan las que se consideran más relevantes.

En primer lugar, se pretende implementar un clasificador inteligente de tareas, nodos de administración y dispositivos de procesamiento, el cual tenga la capacidad de asignar los trabajos de acuerdo con el estado de la red, las capacidades del sistema y la complejidad de las tareas en ejecución. Esto se considera de carácter esencial para garantizar un uso aún más eficiente de los recursos disponibles y asegurar una mejora en los tiempos de respuesta.

En segundo lugar, otra cuestión abierta interesante es la necesidad de incorporar un modelo de incentivos basado en créditos. De esta manera, por un lado, los usuarios que colaboran con sus dispositivos durante los momentos de inactividad podrían percibir una ganancia. Por otro, las instituciones que utilicen la plataforma pueden verse beneficiadas por una masa más grande y más extensa de usuarios motivados a brindar sus equipos a tan solo una fracción del costo. Cómo resultado, se obtiene un beneficio para ambas partes y se genera una mayor disponibilidad y capacidad de procesamiento.

Tercero, resulta interesante extender las capacidades y características del servicio implementado (transcodificación de video) así como integrar nuevas tareas HPC como, por ejemplo, procesamiento de señales, simulaciones físicas, recuperación de información, análisis masivo de datos, etc. A través de estas incorporaciones y mejoras, se podría construir una mayor base de conocimiento y desarrollo, logrando estructurar, estandarizar y flexibilizar

aún más las capacidades y módulos de la plataforma. Al mismo tiempo se realizaría una mayor cantidad de pruebas y análisis del comportamiento del sistema.

En el caso particular de la transcodificación de video, es interesante ampliar el enfoque considerado e integrar no solo fuentes pregrabadas (VoD), sino también material en vivo (Live). Incluir este tipo de tarea propone nuevos y más complejos desafíos. Además, es relevante incluir nuevos formatos de codificación (H.265 y H.266) que requieran un mayor nivel de procesamiento, así como también utilizar otras combinaciones de formatos de entrada y salida de datos (.raw, .mkv, .avi, etc).

Por último, y relacionado con la evaluación continua de la plataforma, es interesante estudiar y analizar la dinámica resultante de realizar pruebas de performance y escalabilidad sobre otros escenarios. Se propone utilizar nodos de administración y procesamiento heterogéneos e incluir una frecuencia aleatoria y/o definida de errores en ellos. Este ambiente de pruebas permitiría validar cómo se comporta y escala la plataforma ante cambios constantes en tiempo de ejecución.

ANEXO I - Despliegue de la plataforma en Kubernetes

En este anexo se presenta una breve descripción de cómo está organizado el código de la plataforma construida, cómo otorgar accesos al cluster de Kubernetes y de qué manera hacer los despliegues para brindar los servicios ofrecidos a través de este orquestador de contenedores.

Para acceder al material de este trabajo, se debe acceder al siguiente link de GitHub: https://github.com/dpetrocelli/PhD-Mobile-K8s-HPC-Platform

Arquitectura básica del proyecto

En primer lugar, en la imagen AI.1, se presenta una captura que muestra las carpetas y archivos implicados en el proyecto.

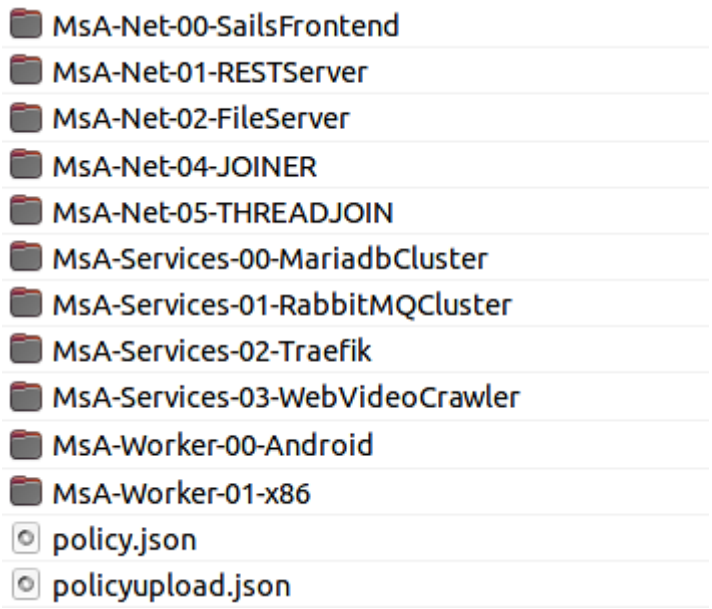

Figura AI.1 - Carpetas y archivos básicos del proyecto de investigación. Fuente: Propia.

Cómo se puede apreciar en la imagen AI.1, las funciones del proyecto se dividen en tres tipos de microservicios:

- Módulos de Red básicos ("Net"): Estos son los componentes fundacionales para que la red pueda ofrecer los servicios de la plataforma. En este apartado se encuentra desde la GUI del sistema, hasta los propios módulos de backend que dan soporte a

las funciones de la plataforma. En cada carpeta de cada módulo se encuentra el código fuente y los archivos necesarios para construir la imagen Docker.

- Módulos de Servicios de soporte (“Services”): En este paquete de servicios, se encuentran los siguientes módulos: módulos de soporte que se integran a la plataforma para que el sistema pueda registrar las tareas y actividades de procesamiento (RabbitMQ y MariaDB). Además, se cuenta con el servicio de Traefik que permite publicar los servicios desde Internet hacia el clúster de Kubernetes (Ingress Controller). Por último, el servicio que se encarga de capturar contínuamente material desde Vimeo y publicarlo hacia la plataforma para utilizar estas fuentes en las pruebas de transcoding (WebVideoCrawler).
- Módulos de Procesamiento (“Worker”): Corresponde a los dos tipos de nodos de procesamiento HPC que fueron desarrollados (x86 y móvil Android). Además, para el nodo Android se encuentra la versión compilada (.apk).

Despliegue de un Cluster de Kubernetes

Cómo se describió a lo largo de este trabajo de investigación, haber construido y empaquetado los servicios utilizando la tecnología de contenedores, específicamente Docker, permite que éstos sean desplegados en cualquier cluster de Kubernetes.

Por lo tanto, como primer paso para poder correr la plataforma es necesario crear un clúster. Para ello, dependiendo de si se desea utilizar una arquitectura local o en la Nube, existen diversas alternativas:

- **Local:** Se puede desplegar utilizando cualquiera de las siguientes herramientas y versiones de Kubernetes:
    - Kind: Es una herramienta para ejecutar clústeres de Kubernetes locales, mediante los "nodos virtuales" que corren sobre Docker.
    - K3s: Es una distribución de Kubernetes certificada y de alta disponibilidad, diseñada para cargas de trabajo de producción en ubicaciones remotas desatendidas, con recursos limitados o dentro de dispositivos de IoT. Incluye un único binario de menos de 40 MB.
    - Microk8s: Similar a K3s, es una implementación que correcompletamente en el equipo del usuario. Ejecuta todos los servicios de Kubernetes de forma

nativa (es decir, sin máquinas virtuales), mientras empaqueta todo el conjunto de bibliotecas y binarios necesarios.

    - MiniKube: Es una herramienta que permite usar Kubernetes en una máquina local. Básicamente crea una VM que ya trae todos los recursos preparados para correr independientemente de la máquina host.

- **Local (en clúster):** Se puede desplegar utilizando cualquiera de las siguientes herramientas:

    - Kubeadm: Herramienta nativa de Kubernetes, para crear clústeres en equipos locales o remotos. Utiliza la versión por defecto de Kubernetes.

    - Kubespray: Herramienta diseñada para implementar clústeres de Kubernetes en la nube o en las instalaciones locales. Para los despliegues, se basa en playbooks de Ansible.

    - Rancher: Es un software completo (Plataforma PaaS) para equipos que adoptan contenedores. Aborda los desafíos operativos y de seguridad de administrar múltiples clústeres de Kubernetes, al tiempo que brinda herramientas integradas para ejecutar cargas de trabajo en contenedores. Utiliza diferentes versiones adaptadas de Kubernetes, dependiendo del contexto (K3s para equipos de bajos recursos, RKE para entornos productivos y RKE2 para entornos de alta seguridad).

    - OpenShift: Es un producto de software (Plataforma Paas) que incluye elementos del proyecto de gestión en contenedores de Kubernetes, pero agrega funciones de seguridad y productividad que son importantes para las grandes empresas. Utiliza una versión por defecto de Kubernetes, administrada por el cliente OC de RedHat en vez de kubectl.

- **Nube:** Se puede desplegar utilizando cualquiera de las siguientes Nubes públicas, que ya ofrecen servicios de Kubernetes administrados (Amazon EKS, Azure AKS, Google Cloud GKS, IBM IKS, entre otros), o realizar despliegues automatizados con herramientas definidas por el usuario (Terraform, Ansible, o mismas Kubespray, Kops, Rancher, entre otras) que se conecten a las API de los proveedores de servicio en la Nube.

Organizar y establecer el acceso del clúster para kubectl

Una vez creado el clúster, se requiere poder contactar y administrar los servicios del mismo. Para ello, Kubernetes crea y utiliza usa un archivo YAML llamado “kubeconfig”, a fin de almacenar información de autenticación de clúster para kubectl. Por ejemplo, si se crea un clúster en el proveedor Okteto luego, cómo puede verse en la imagen AI.2, se debe descargar el archivo de acceso.

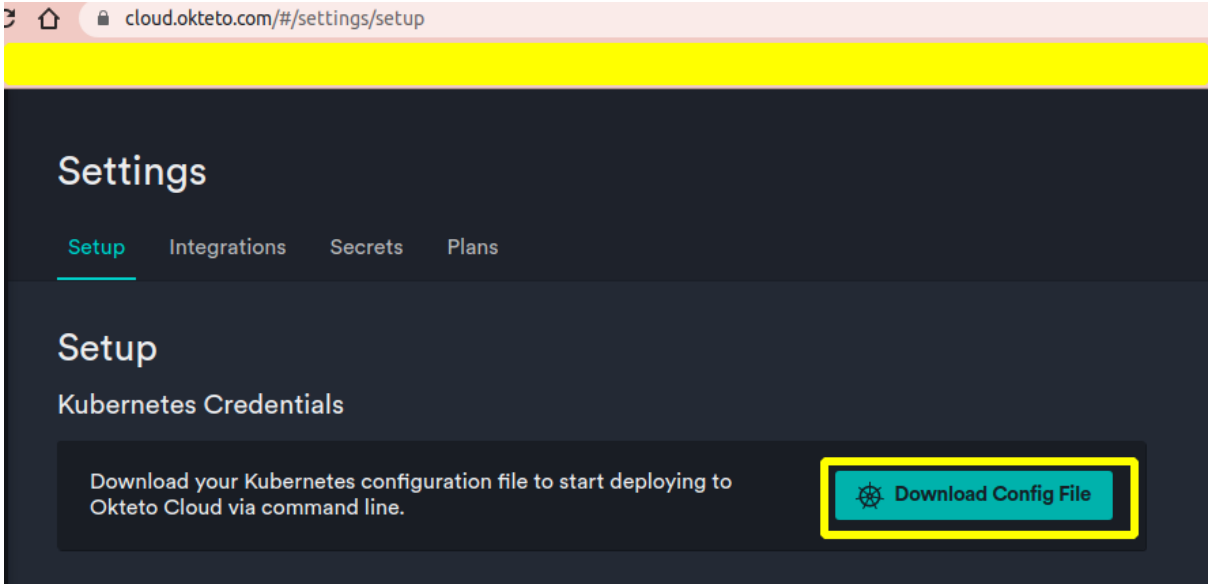

Figura AI.2 - Pasos a seguir para obtener el archivo Kubeconfig (Okteto) para conectarse y administrar un clúster de Kubernetes vía Kubectl. Fuente: Propia.

Un ejemplo de este archivo de configuración se presenta en la imagen AI.3, a continuación:

Figura AI.3 - Ejemplo de archivo Kubeconfig de un cluster de Kubernetes. Fuente: Propia.

De forma predeterminada, este archivo se debe guardar en $HOME/.kube/config para que el cliente kubectl lo lea automáticamente.

Este fichero contiene, por un lado, una lista de contextos a los que kubectl se refiere cuando ejecuta comandos y por otro, todos los parámetros de acceso. Cada contexto contiene la definición de un clúster de Kubernetes, usuarios y espacios de nombres. Si se ejecutan varios clústeres dentro de un proyecto, se debe elegir con cual se comunicará el cliente de administración kubectl. Para ello se puede configurar un clúster predeterminado para esa herramienta. Esta acción se realiza estableciendo el contexto actual en el archivo “kubeconfig” de Kubernetes.

Despliegue de reglas y estructura de servicios en el clúster

Las primeras acciones a definir en el cluster de Kubernetes son las reglas y estructura de servicios, para que se cumplan las condiciones de seguridad, rendimiento y administración correcta de recursos.

Por lo tanto, en primer lugar, se crea el espacio de nombres (Namespace) dp-microservices. Cómo se explicó en el trabajo de tesis (Capítulo 3 - Gestión de la configuración, automatización y reversiones) el objetivo es contar con dos espacios de nombres, para garantizar que ciertos servicios no invadan los recursos utilizados por módulos de mayor prioridad. Para ello, se utiliza la siguiente línea:

```
$ kubectl create namespace dp-microservices
```

Una vez creado, es necesario limitar los recursos de dicho espacio de nombres para no consumir la totalidad de recursos del sistema. Por esta razón, se ejecuta el script “check-limit-resources.sh”.

```
#!/bin/bash
while true; do

   #[STEP 1] - Obtener número de nodos
   NODES=$(kubectl --kubeconfig kubeconfig.yaml top nodes | sed -n '1!p' | tr -
s " " | cut -d' ' -f1 | wc -l)
   #[STEP 2] - Obtener los cores CPU (cores)
   CPU=$(kubectl --kubeconfig kubeconfig.yaml top nodes | sed -n '1!p' | tr -s
" " | cut -d' ' -f2 | sed 's/m//g' | paste -sd+ | bc)
   #[STEP 3] - Obtain la Memoria (bytes)
   MEM=$(kubectl --kubeconfig kubeconfig.yaml top nodes | sed -n '1!p' | tr -s
" " | cut -d' ' -f4 | sed 's/Mi//g' | paste -sd+ | bc)

   #[STEP 4] - Calcular los N -1 Nodos

   if [ $NODES > 1 ];
   then
           AVAILABLES_NODES=$(expr $NODES - 1)
   else
           AVAILABLES_NODES=$NODES
   fi

   #[STEP 5] - Calcular la cuota de CPU
   CPU_QUOTA=$(($CPU / $NODES))
```

```
    CPU_QUOTA=$(($CPU_QUOTA * $AVAILABLES_NODES))

    #[STEP 6] - Calcular la cuota de Memoria
    MEM_QUOTA=$(($MEM / $NODES))
    MEM_QUOTA=$(($MEM_QUOTA * $AVAILABLES_NODES))

    echo "$CPU_QUOTA ; $MEM_QUOTA"

    sed -i "7s/.*/       limits.cpu: $CPU_QUOTA/g" defining-dp-microservices-
quota.yaml
    sed -i "8s/.*/     limits.memory: $MEM_QUOTA/g" defining-dp-microservices-
quota.yaml
    sed -i "9s/.*/    pods: "100"/g" defining-dp-microservices-quota.yaml

    kubectl --kubeconfig kubeconfig.yaml apply -f defining-dp-microservices-
quota.yaml
    sleep 10
done
```

Este script se encarga de consultar continuamente, cada 10 segundos, la capacidad del clúster, obteniendo la cantidad de nodos, CPU (cores) y memoria (bytes). Con los resultados obtenidos, se procede a calcular los límites que deben asignarse. Para ello, con los valores AVAILABLES_NODES, CPU_QUOTA y MEM_QUOTA se restringe la capacidad a N-1 nodos, y se reemplazan estos valores en el archivo YAML de Kubernetes. Una vez modificado, se aplican los cambios en el cluster a través del comando kubectl.

Luego de creado el espacio de nombres y haber definido su cuota, se procede a configurar el servicio de Traefik como Ingress Controller. Cómo se explicó en el apartado de Redes y Balanceo de carga: Ingress Controller y Services del capítulo 3 (Capítulo 3 - Gestión de la configuración, automatización y reversiones), el servicio de Traefik utiliza 5 archivos (1 archivo de configuración básico de Traefik y 4 ficheros de definición en Kubernetes), cómo se muestra en la figura AI.4.

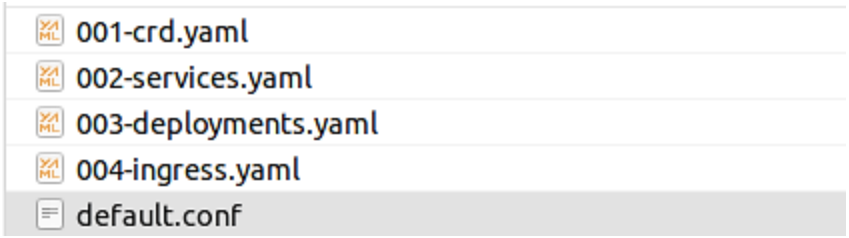

Figura AI.4 - Archivos de configuración para desplegar el servicio de Ingress Controller Traefik en Kubernetes. Fuente: Propia.

Cómo se muestra a continuación, solo deben aplicarse las cuatro definiciones a través del comando kubectl apply y el servicio se pondrá en funcionamiento:

```
$ kubectl apply -f 001-crd.yaml
$ kubectl apply -f 002-services.yaml
$ kubectl apply -f 003-deployments.yaml
```

```
$ kubectl apply -f 004-ingress.yaml
```

Despliegue de servicios de soporte

Una vez que está activo el cluster y las reglas en funcionamiento se han aplicado, se procede a desplegar los servicios de base de datos y sistemas de colas para dar soporte a las funciones del sistema desarrollado. Como ya se explicó, estos dos módulos se alojan en el espacio de nombre por defecto (default) sin limitantes de recursos. Para realizar esta tarea de manera sencilla, se utiliza el administrador de paquetes Helm. Seguidamente, se describen las dos líneas utilizadas para levantar los servicios, con sus respectivas propiedades de alta disponibilidad:

```
# [STEP 1] - Iniciar servicio de base de datos con alta disponibilidad
$ helm install mariadb-cluster --set
rootUser.password={$ROOT_PWD},db.name=distributedProcessing,db.user={$US
ER},db.password={$DB_PWD},replicaCount=3,service.type=ClusterIP
bitnami/mariadb-galera

# [STEP 2] - Iniciar servicio de RabbitMQ con alta disponibilidad
$ helm install rabbitmq-cluster --set
rabbitmq.username=admin,rabbitmq.password={$ROOT_PWD},rabbitmq.erlangCoo
kie={$COOKIE},replicas=3,service.type=ClusterIP bitnami/rabbitmq
```

Con esta definición Kubernetes automáticamente levanta tres réplicas de cada uno de los servicios y los configura en un cluster de alta disponibilidad. Sin embargo, en casos de mucha carga esto puede llegar a no ser suficiente. Por lo tanto, es necesario crear un servicio adicional llamado Horizontal Pod Autoscaler (HPA). Para ello, se utiliza la definición de un “autoscaler”, que permite reaccionar ante el porcentaje de uso de las réplicas de los servicios. Para ambos casos, se tomó cómo evento de acción el 75% de uso del CPU asignado a cada una de las réplicas. La configuración definida se muestra en el siguiente cuadro:

```
# [STEP 1] - Definir el autoscale para el motor mariadb
$ kubectl autoscale statefulset mariadb-cluster --cpu-percent=75 --min=1
--max=10
# [STEP 2] - Definir el autoscale para el middleware RabbitMQ
$ kubectl autoscale statefulset rabbitmq-cluster --cpu-percent=75 --min=1
--max=10
```

Actualizaciones y despliegue de código (funciones)

Una vez que el clúster, las reglas, los servicios de base de datos y colas estań corriendo; se pasa a implementar los microservicios desarrollados para la plataforma. En este punto, cabe destacar que para cada uno de ellos hay dos actividades relevantes a tener en cuenta.

En primer lugar, en caso de que se realice una modificación en el código fuente, es necesario que se actualice la "imagen" dockerizada que se resguarda en los registros de Docker, ya que finalmente es lo que implementa el orquestador Kubernetes. Por lo tanto, los pasos a seguir son los que definieron en el capítulo 3 (Microservicios Dockerizados para la administración y gestión de la plataforma). A modo de resumen se recuerda que se necesita:

- Parametrizar la ejecución de los microservicios.
- Exportar el microservicio cómo un archivo ejecutable.
- Definir el archivo Dockerfile para la construcción de la imagen Docker.
- Crear la imagen Docker y publicarla en Registro (Docker Hub).

Cada servicio desarrollado, dispone de una carpeta bajo el nombre de "kubefiles", cómo se muestra en la figura AI.5, la cual dispone un archivo bash (deploy-automation.sh) que se encarga de realizar estos cuatro pasos de forma automatizada.

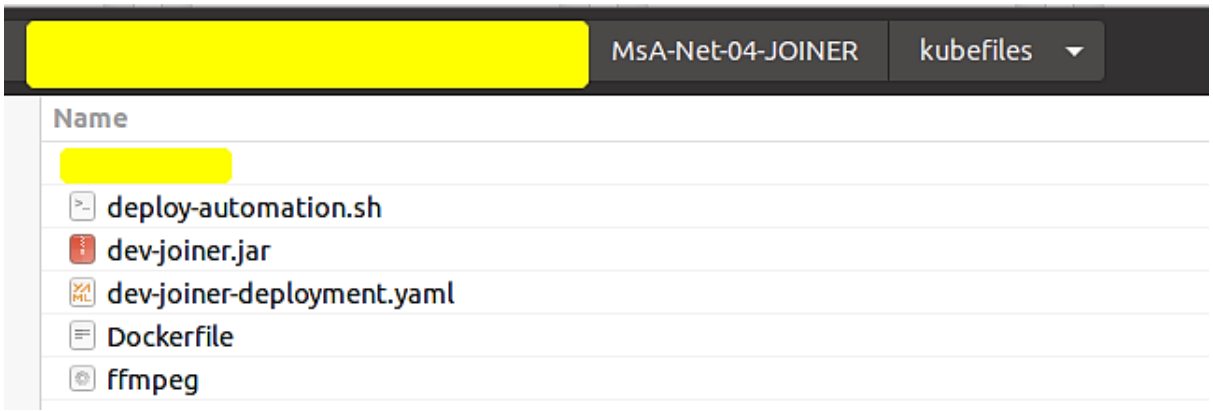

Figura AI.5 - Ejemplo de carpeta "kubefiles" dentro uno de los microservicios desarrollados. Fuente: Propia.

Debajo se muestra cómo el archivo "deploy-automation.sh" realiza todos estos pasos de manera automática:

```
cd ..
# [STEP 1] - Compilo el código
mvn package
cd kubefiles
# [STEP 2] - Muevo el paquete para poder construir la imagen
cp ../target/ex1-0.0.1-SNAPSHOT.jar dev-joiner.jar
# [STEP 3] - Construyo la imagen (basada en el Dockerfile)
docker build -t dpetrocelli/dev-joiner:latest .
# [STEP 4] - Subo la imagen al repositorio
docker login
docker push dpetrocelli/dev-joiner
```

En segundo lugar, a la hora de correr el servicio en Kubernetes, es necesario definir dos configuraciones:

- Despliegue Kubernetes.
- Autoescalador de réplicas.

El despliegue de Kubernetes define las propiedades básicas del servicio. En él se define la imagen base (previamente publicada en el registro Docker), la cantidad inicial de réplicas y los mínimos y máximos de recursos que puede tomar el contenedor al estar ejecutándose. A continuación, se muestra una parte del ejemplo de despliegue para el nodo Worker x86 (dev-workerx86-deployment.yaml):

```
apiVersion: apps/v1
kind: Deployment
...
 replicas: 4
 template:
...
   spec:
     containers:
     - name: dev-workerx86
       image: dpetrocelli/dev-workerx86
       resources:
         requests:
           cpu: 200m
           memory: 200Mi
         limits:
           cpu: 2000m
           memory: 2000Mi
...
```

Y se aplica con la misma sentencia de Kubernetes (apply), pero teniendo en cuenta que se debe ubicar dentro del espacio de nombres limitado (dp-microservices), cómo se muestra a continuación:

```
$ kubectl apply -f dev-workerx86-deployment.yaml --namespace=dp-
microservices
```

El auto escalador de réplicas (hpa), se define de la siguiente manera (autoscaler-deployment.yaml):

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
 name: autoscaler-devworkerx86
spec:
 maxReplicas: 10
 minReplicas: 1
 scaleTargetRef:
   apiVersion: apps/v1
   kind: Deployment
   name: dev-workerx86
 metrics:
 - type: Resource
   resource:
     name: cpu
     targetAverageUtilization: 75
 - type: Resource
   resource:
     name: memory
     targetAverageUtilization: 75
```

En este caso, se define un rango de réplicas con un mínimo de 1 pod y un máximo de 10, realizando los cambios cuando los recursos superan el 75% del CPU y/o memoria utilizada. Finalmente, se aplica mediante el mismo comando (kubectl apply), teniendo en cuenta que se debe ubicar dentro del espacio de nombres limitado, cómo se muestra a continuación:

```
$ kubectl apply -f autoscaler-deployment.yaml --namespace=dp-
microservices
```

ANEXO II - Dispositivos de medición de consumo energético

A la hora de realizar las mediciones de energía sobre los dispositivos x86 y Android ARM fue necesario construir herramientas (software y hardware) que permitieran capturar, registrar y documentar el consumo de los nodos de procesamiento.

Medición en arquitecturas x86

Los equipos x86 se alimentan de Corriente Alterna (AC) proveniente de un tomacorriente de 220v estándar hacia la fuente de alimentación. Para capturar la información de consumo energético se construyó un dispositivo de medición no invasivo basado en microcomponentes. Básicamente se utilizaron los siguientes elementos:

- Componente 1 (C1) - Pinza SCT 013-030.
- Componente 2 (C2) - Módulo Arduino ESP8266 nodemcu v3.
- Componente 3 (C3) - Alargue de tensión.
- Componente 4 (C4) - Multímetro True RMS ProsKit MT-1706.

La descripción, interacción y vinculación de estos componentes se detalla a continuación.
Por otro lado, toda la información capturada se publica en un servidor web (en la base de datos MariaDB) que registra dicha información para ser consumida y realizar el análisis estadístico.

Componente 1 (C1) - Pinza SCT 013-030

La pinza SCT 013-030[15] no invasiva fue utilizada para capturar la información de Corriente Alterna (entrada máxima 30A; salida máxima intensidad de 1V) consumida por los diversos dispositivos x86. En la figura AII.1 y AII.2 se presentan imágenes de esta herramienta.

[15] SCT 013-030: https://www.mcielectronics.cl/website_MCI/static/documents/Datasheet_SCT013.pdf

Figura AII.1 - Pinza SCT 013-030 (vista cerrada). Fuente: Propia.

Figura AII.2 - Pinza SCT 013-030 (vista abierta). Fuente: Propia.

Componente 2 (C2) - Módulo Arduino ESP8266 nodemcu v3

El Módulo Arduino ESP8266 nodemcu v3[16] es un chip Wi-Fi, de bajo costo con *stack* TCP/IP completo y capacidad de MCU (Micro Controller Unit), especial para manejar redes de sensores. Esta placa permite la programación con Arduino IDE, herramienta que se utilizó para desarrollar las rutinas en lenguaje C para el manejo del componente (C1). En la figura AII.3 se presenta una imagen de la placa ESP8266 en cuestión.

[16] Arduino ESP8266 nodemcu v3:
https://docs.zerynth.com/latest/official/board.zerynth.nodemcu3/docs/index.html

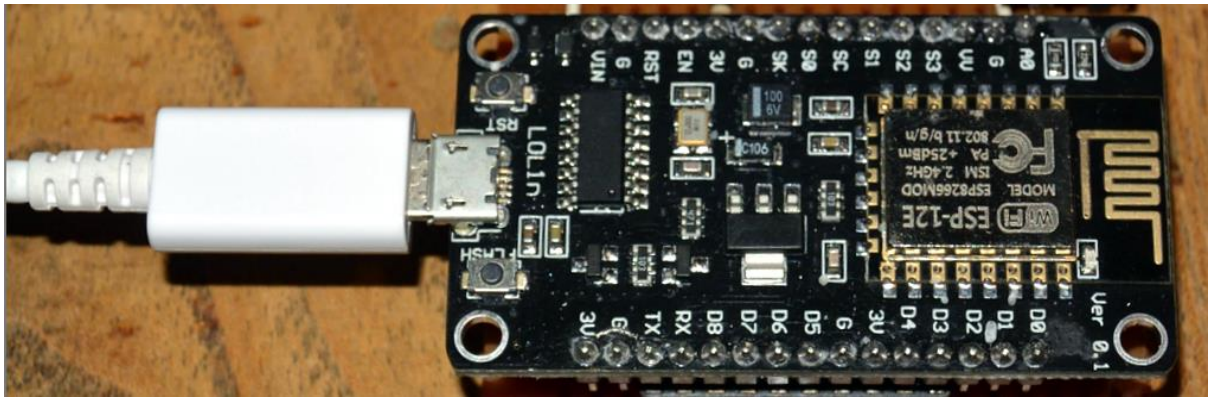

Figura AII.3 - Placa Arduino ESP8266 nodemcu v3 (vista superior). Fuente: Propia.

Componente 3 (C3) - Alargue de tensión

El alargue de tensión (Conector macho, 1x cable fase + 1x cable neutro, conector hembra) permite no invadir la conexión original de los equipos x86 hacia la fuente de alimentación, y lograr separar naturalmente los cables para la medición (fase, neutro).

Componente 4 (C4) - Multímetro True RMS ProsKit MT-1706

El multímetro True RMS ProsKit MT-1706[17] fue utilizado para validar que las métricas obtenidas por el prototipo de medición (C1-C2-C3) fueran correctas. A continuación, en las figuras AII.4 y AII.5 se presentan imágenes del producto adquirido.

Figura AII.4 - Dispositivo ProsKit MT-1706 (vista superior, apagado). Fuente: Propia.

[17] Multímetro ProsKit MT-1706: http://www.sycelectronica.com.ar/herramientas/TESTER-MT1706.pdf

Figura AII.5 - Dispositivo ProsKit MT-1706 (vista superior encendido). Fuente: Propia.

Para las mediciones realizadas durante los procesos de cómputo intensivo se conecta el cable de fuente del equipo x86 a la toma hembra del alargue (C3), habiendo previamente conectado éste al toma corriente. Luego se toma la pinza amperimétrica (C1) y se rodea el cable fase del alargue de tensión (C3). Más tarde, se conecta la placa Arduino (C2) por el puerto USB al equipo observador, y teniendo el código compilado en la placa MCU. De esta manera, se obtienen los resultados (cada 1000 milisegundos) del consumo energético en la unidad de medida watts y se vuelcan a la base de datos del servidor REST de la plataforma. Todo el tiempo se verifica que los datos sean correctos mediante el control del multímetro (C4).

Construcción y funcionamiento del prototipo de medición

Con todas las herramientas disponibles, se integraron los componentes y se construyó un prototipo sobre una protoboard, como se presenta en la figura AII.6 a continuación.

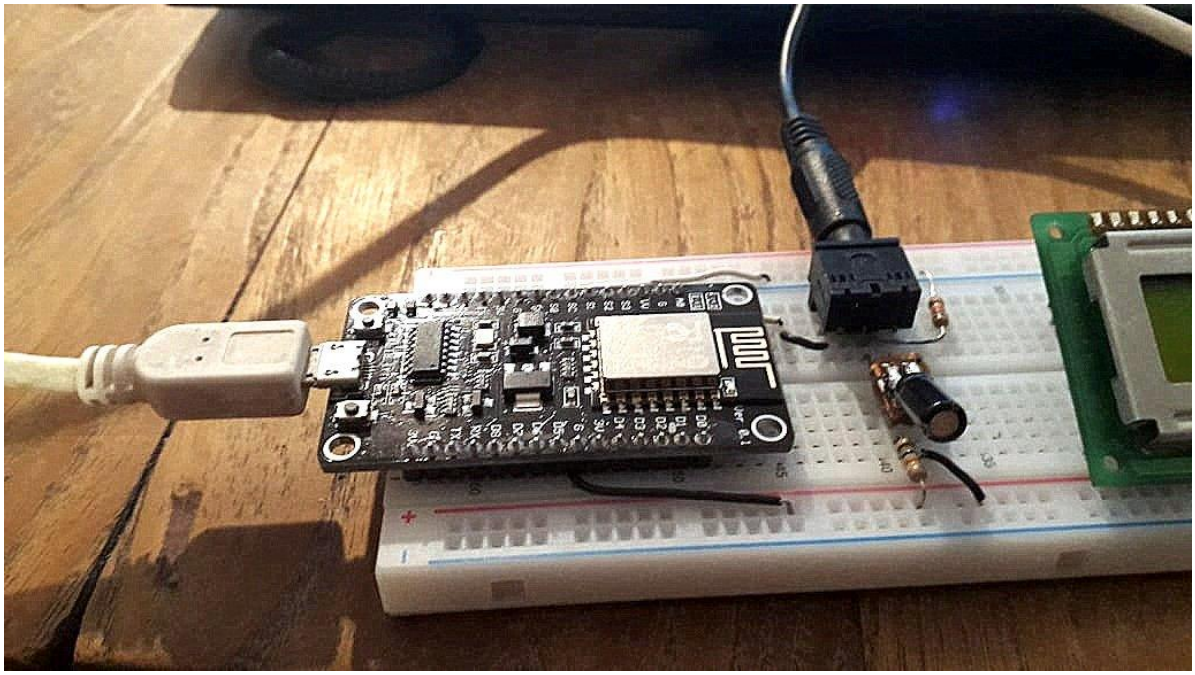

Figura AII.6 - Prototipo de medición montado sobre una protoboard temporal para validar el funcionamiento de los componentes y configuraciones. Fuente: Propia.

Luego de contar con los componentes conectados, se desarrolló y compiló el código que permite obtener la información de potencia (Watts) y su correspondiente registro, para poder ser integrado con el resto de la plataforma. Luego de haber realizado las pruebas y verificación de resultados, con objeto de evitar problemas de desconexión y ruidos innecesarios, por el movimiento de los componentes, se construyó una versión final del prototipo soldando los componentes a una placa definitiva, como se presenta en las figuras AII.7, AII.8 y AII.9.

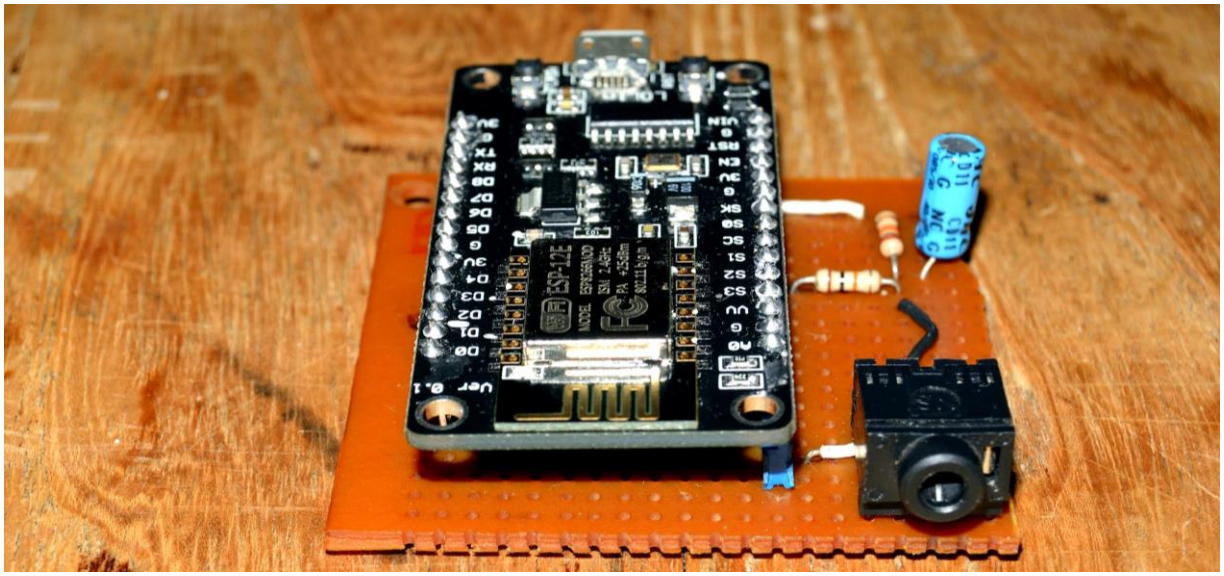

Figura AII.7 - Prototipo de medición soldado en placa definitiva (vista jack de conexión con pinza de medición). Fuente: Propia.

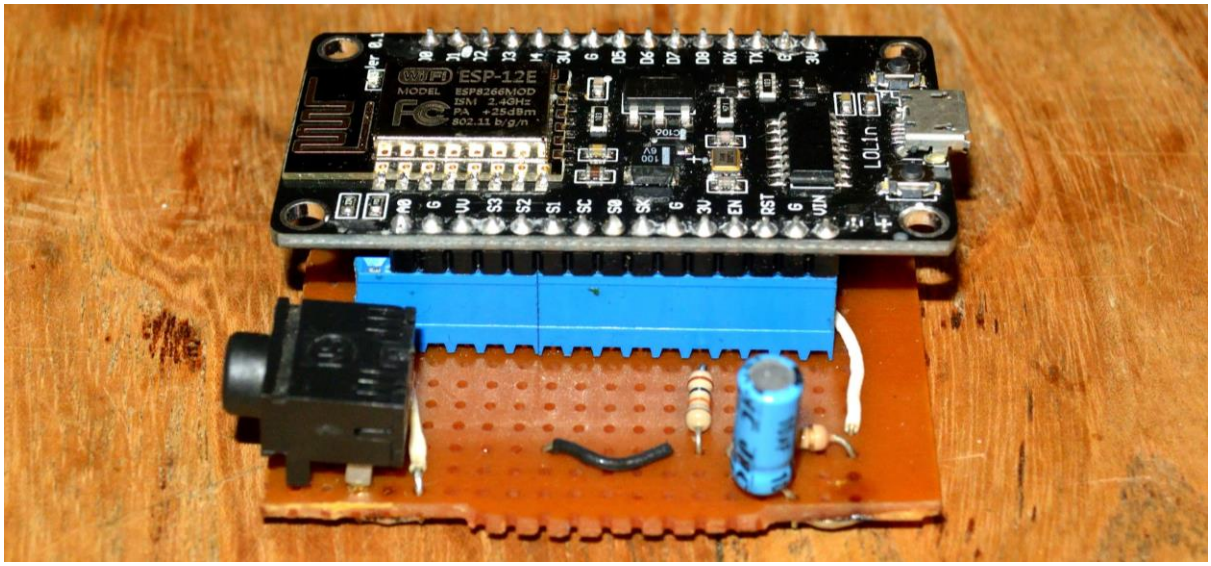

Figura AII.8 - Prototipo de medición soldado en placa definitiva (vista lateral de capacitadores de protección). Fuente: Propia.

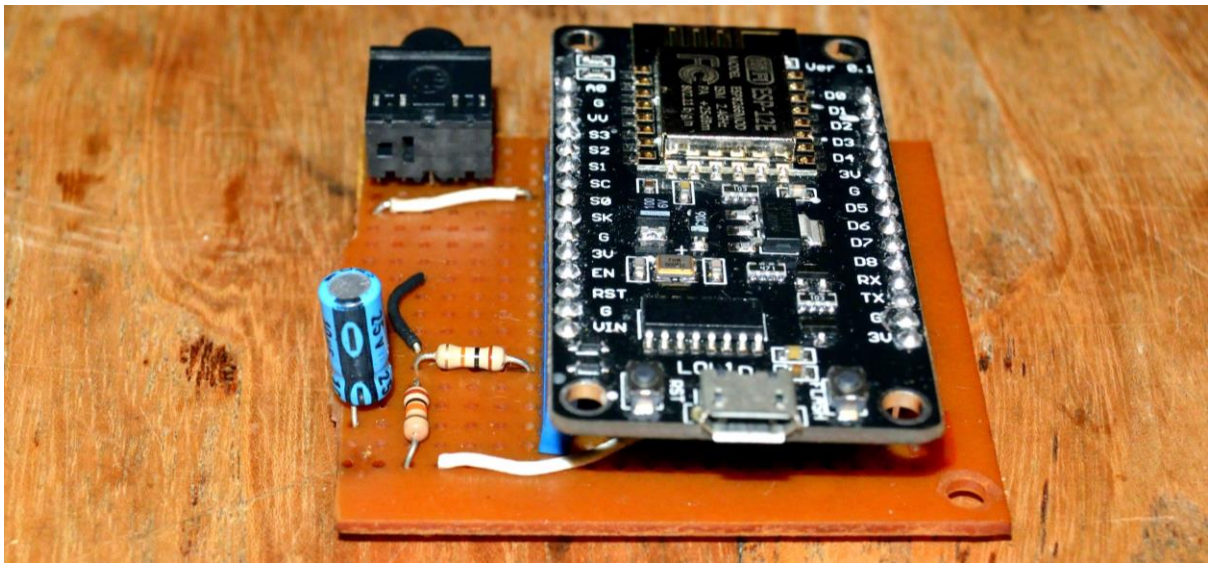

Figura AII.9 - Prototipo de medición soldado en placa definitiva (vista trasera hacia conector de energía USB). Fuente: Propia.

Con todos los componentes disponibles (módulo Arduino en placa definitiva, pinza amperimétrica, alargue y multímetro true RMS), como se muestra en la figura AII.10, se comenzaron a realizar las pruebas y mediciones.

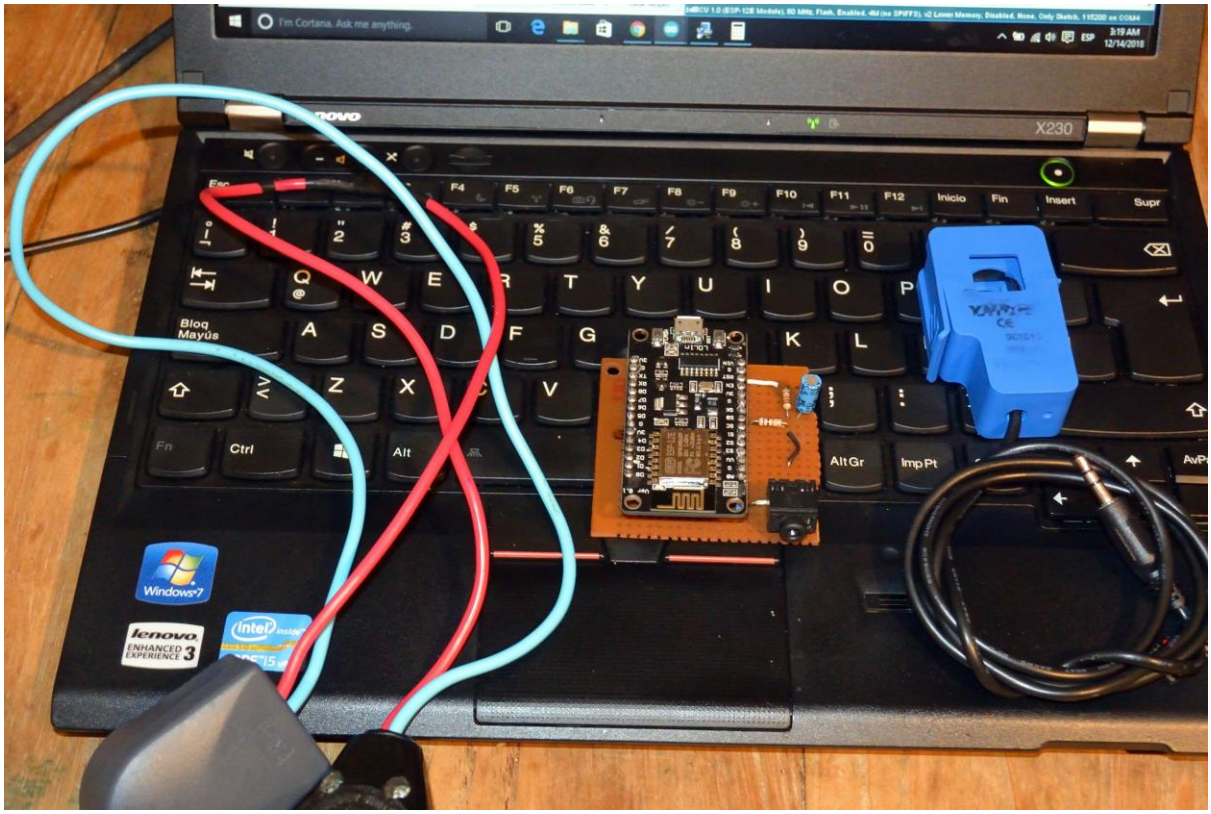

Figura AII.10 - Cable extensor, módulo Arduino en placa definitiva, pinza amperimétrica y equipo x86, con servidor Web corriendo para recibir información. Fuente: Propia.

En la figura AII.11, se visualiza como, por un lado, a través de la entrada micro USB la placa Arduino se alimenta de energía y por otro, como vía el mini jack 3.5 mm se conecta la pinza amperimétrica.

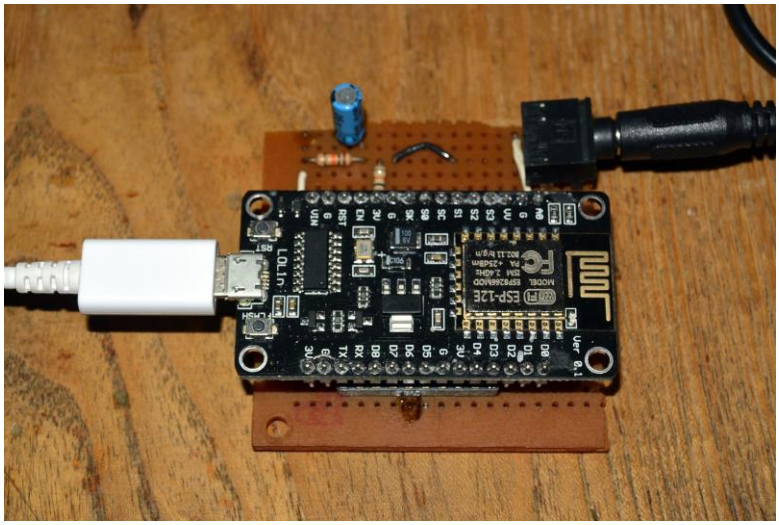

Figura AII.11 - Placa arduino conectada vía USB para alimentación y mini jack 3.5mm para recibir mediciones de la pinza amperimétrica. Fuente: Propia.

Por otra parte, en la figura AII.12, se muestra como la pinza envuelve el cable neutro de la corriente alterna (CA) que alimenta la fuente de los equipos x86, para enviar esta información

al módulo Arduino. En la misma imagen también se puede observar como las puntas del multímetro (tipo aguja) interceptan el cable de energía para obtener también la medición en tiempo real, y recibirla en dicho equipo. Esto permite comparar y verificar que los datos capturados por el prototipo construido sean fiables.

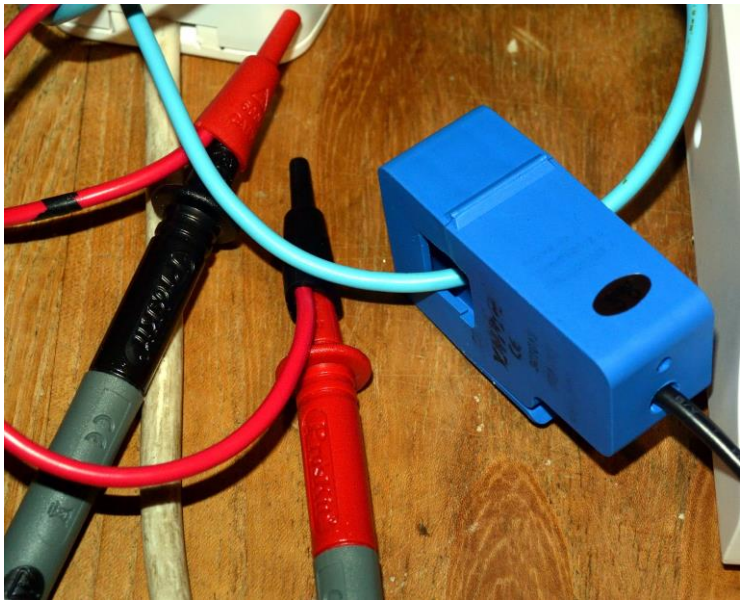

Figura AII.12 - Conexión de pinza amperimétrica y multímetro a los cables de alimentación del equipamiento x86. Fuente: Propia.

Finalmente, una vez conectados y encendidos todos los componentes, se ejecutan las tareas de procesamiento y se toman, registran y publican, al servidor REST de la plataforma, los datos de consumo energético, cómo se puede visualizar en las imágenes AII.13 y AII.14.

Al mismo tiempo, se realiza una comparación en tiempo real de la fiabilidad de las métricas obtenidas a través del multímetro True RMS.

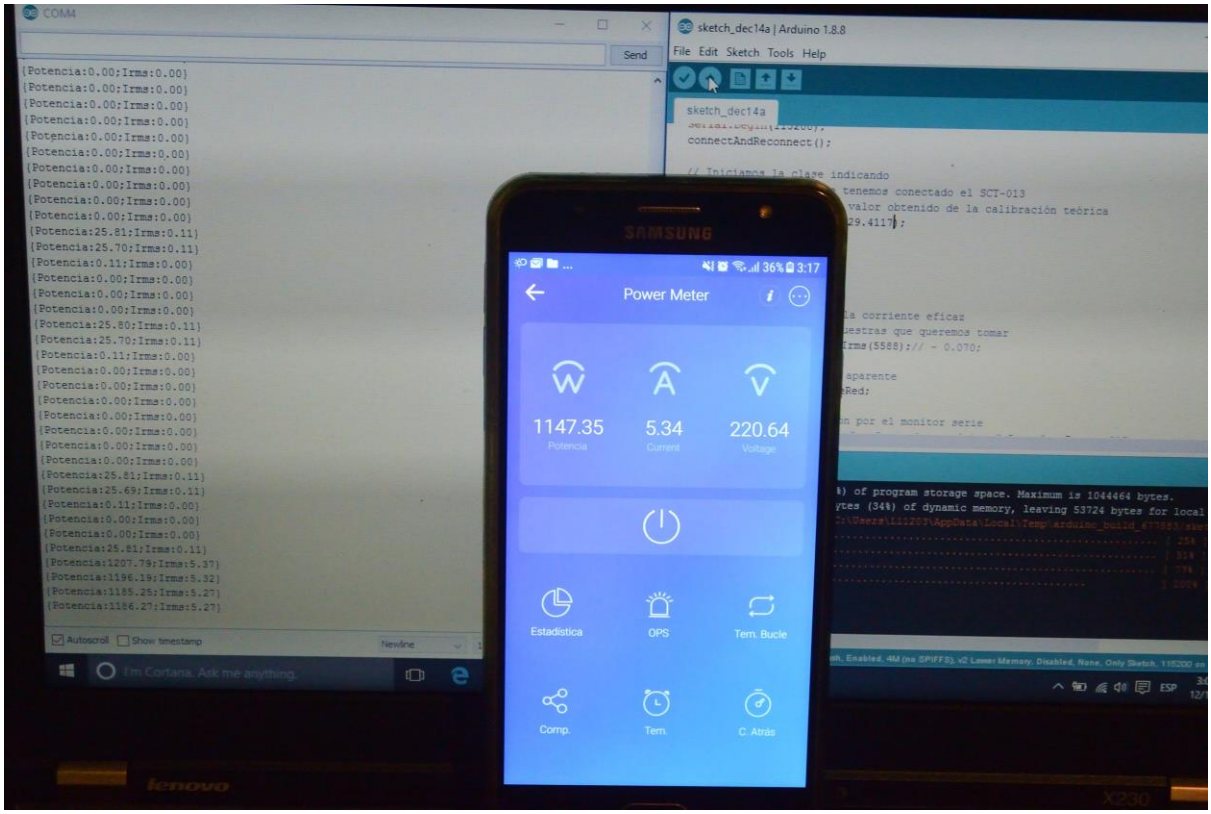

Figura AII.13 - Captura y publicación de información de consumo energético a través del prototipo de medición. Fuente: propia

```
{Potencia:25.81;Irms:0.11}
{Potencia:25.69;Irms:0.11}
{Potencia:0.11;Irms:0.00}
{Potencia:0.00;Irms:0.00}
{Potencia:0.00;Irms:0.00}
{Potencia:1207.79;Irms:5.37}
{Potencia:1196.19;Irms:5.32}
{Potencia:1185.25;Irms:5.27}
{Potencia:1186.27;Irms:5.27}
{Potencia:1186.60;Irms:5.27}
{Potencia:1180.05;Irms:5.24}
{Potencia:1179.04;Irms:5.24}
{Potencia:1182.30;Irms:5.25}
{Potencia:1179.46;Irms:5.24}
{Potencia:1188.18;Irms:5.28}
{Potencia:1179.64;Irms:5.24}
{Potencia:1175.26;Irms:5.22}
{Potencia:1183.60;Irms:5.26}
{Potencia:1174.32;Irms:5.22}
{Potencia:27.48;Irms:0.12}
{Potencia:0.42;Irms:0.00}
{Potencia:25.81;Irms:0.11}
{Potencia:25.81;Irms:0.11}
{Potencia:0.11;Irms:0.00}
{Potencia:25.81;Irms:0.11}
{Potencia:25.69;Irms:0.11}
{Potencia:0.11;Irms:0.00}
{Potencia:0.00;Irms:0.00}
```

Figura AII.14 - Captura y publicación de información de consumo energético, a través del prototipo de medición (imagen maximizada). Fuente: Propia.

Finalmente, se pudo verificar con el Tester TRUE RMS que los datos obtenidos con este dispositivo se condicen con los recabados por el multímetro.

Medición en arquitecturas ARM Android

A diferencia del caso anterior, los dispositivos móviles trabajan con Corriente Continua (DC), la cual es obtenida desde la batería del dispositivo. Debido a ello, fue necesario construir un medidor de corriente por software. Para ello, se siguieron las definiciones y métodos utilizados en [RUA19] [NUC17] [FIS15] [LID14] [TIW10] y se integró un protocolo de medición a la solución del worker Android ARM, a través de un hilo independiente que se ejecuta sólo durante el ciclo en el que se está realizando el cálculo de una tarea y no interfiere con el procesamiento de la misma. Este hilo se encarga de consultar, cada 1000 milisegundos, la información que provee el sistema operativo respecto de los sensores de la batería. Los registros se encuentran almacenados y actualizados en la carpeta del sistema operativo Linux /sys/class/power_supply/battery/. A continuación, se presenta un fragmento del código que se utiliza para recuperar esta información y luego publicar, vía un mensaje JSON, al servidor REST de la plataforma.

```
//Obtengo el archivo de información de batería actual (current)
f = new File("/sys/class/power_supply/battery/batt_chg_current");
// Chequeo que el archivo exista y esté accesible
if (f.exists())
// Retorno el valor, leyendo la última línea disponible (función propia)
return OneLineReader.getValue(f, false);
```

Glosario

3dmark: 3DMark es un programa que sirve para hacer benchmarking, o en otras palabras, llevar a los límites a un dispositivo para registrar su rendimiento máximo posible. Esto lo realiza mediante un recorrido virtual programado y medido al milímetro para forzar a los dispositivos de prueba a trabajar a máximo rendimiento.
ABR: La transmisión de velocidad de bits adaptativa (ABR) es una técnica utilizada para transmitir multimedia a través de redes informáticas. Funciona detectando el ancho de banda de un usuario y la capacidad de la CPU en tiempo real y ajustando la calidad del flujo de medios en consecuencia. El reproductor del cliente puede cambiar entre las diferentes calidades de las señales en función de los recursos disponibles.
Access y Secret Keys: Las claves de acceso se utilizan para firmar las solicitudes que envía a los servicios de Amazon (S3).
Android NDK: El NDK de Android es un conjunto de herramientas que le permite al desarrollador implementar partes de su aplicación en código nativo, usando lenguajes como C y C ++. Para ciertos tipos de aplicaciones, esto puede ayudar a reutilizar bibliotecas de código escritas en esos lenguajes.
API: API es el acrónimo de Application Programming Interface, que es un intermediario de software que permite que dos aplicaciones se comuniquen entre sí.
API REST: Una API de transferencia de estado representacional (REST), o API de RESTful, es una interfaz de programación de aplicaciones que se ajusta a los límites de la arquitectura REST. REST no es un protocolo ni un estándar, sino que se trata de un conjunto de principios de arquitectura.
ARM: Es una arquitectura de procesadores basada en el set de instrucciones RISC. Estos requieren una menor cantidad de transistores que los procesadores x86 CISC.
AWS. Servicios Web de Amazon. Servicios de plataforma de computación en la Nube que ofrece AMAZON a través de internet.
Azure: Microsoft Azure es un servicio de computación en la nube creado por Microsoft para construir, desplegar y administrar aplicaciones y servicios mediante el uso de sus centros de datos en la nube
Bare-metal: Un servidor bare-metal es un servidor informático que no utiliza virtualización
Bash: es un intérprete de órdenes que generalmente se ejecuta en una ventana de texto donde el usuario escribe órdenes en modo texto. También permite ingresar scripts de ejecución.
Benchmark: Es una técnica utilizada para medir el rendimiento de un sistema o uno de sus componentes.
B-Frame (bi-predictive): Los fotogramas B pueden utilizar fotogramas anteriores y posteriores como referencia de datos para obtener la mayor cantidad de compresión de datos.
Bitrate: Define el número de bits que se transmiten por unidad de tiempo a través de un sistema de transmisión digital o entre dos dispositivos digitales
Blockiness: bloqueado de fragmentos de video debido a errores de compresión o bajas tasas de bits.
Bucket S3: Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento.
Bucket S3 Policy: Es una política de AWS (IAM) basada en recursos para otorgar permisos de acceso a cuentas de AWS o usuarios externos.

CD: Entrega continua es un enfoque de la ingeniería del software en que los equipos de desarrollo producen software en ciclos cortos, asegurando que el software puede ser liberado en cualquier momento, de forma confiable.
CDN: Una red de distribución de contenidos es una red superpuesta de computadoras que contienen copias de datos, colocados en varios puntos de una red con el fin de maximizar el ancho de banda para el acceso a los datos de clientes por la red.
Cgroups: abreviado de grupos de control, es una característica del kernel de Linux que limita, da cuenta y aísla el uso de recursos de una colección de procesos.
Chroot: chroot en los sistemas operativos derivados de Unix, es una operación que invoca un proceso, cambiando para este y sus hijos el directorio raíz del sistema.
CI: La integración continua es una práctica de ingeniería de software que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes
CI/CD: Se refiere a las prácticas combinadas de integración continua y entrega continua.
Cloud Computing: La computación en la nube es un término general para la prestación de servicios alojados a través de Internet, y permite a las empresas consumir recursos informáticos como una utilidad (pago por uso) en lugar de tener que construir y mantener infraestructuras de computación en tu casa o tus oficinas.
Cluster: Es un conjunto de dos o más equipos que se caracterizan por mantener una serie de servicios compartidos y por estar constantemente monitorizándose entre sí. Normalmente se interconectan por una red de alta velocidad y se comportan como si fuesen un único servidor.
Clúster Autoscaler: El escalador automático de clústeres de Kubernetes es un componente que ajusta automáticamente el tamaño de un clúster para que todos los pods tengan un lugar para ejecutarse y no haya nodos innecesarios.
CM: El monitoreo continuo (CM) ayuda al área de IT a revisar los procesos y sistemas las 24 horas del día, los 7 días de la semana para ver si el desempeño, la efectividad y la eficiencia están siendo cumplidos.
Códec: Un códec es un programa o dispositivo hardware capaz de codificar o decodificar una señal o flujo de datos digitales. Códec es un acrónimo de codificador-decodificador
CBR: CBR son las siglas de Constant Bit Rate, que quiere decir que un video dispone de una tasa de bits constante.
Connection Factory: Una fábrica de conexiones es el objeto que utiliza un cliente para crear una conexión con un proveedor. Una fábrica de conexiones encapsula un conjunto de parámetros de configuración de conexiones que ha sido definido por un administrador.
Container limits: La cantidad máxima de CPU/memoria que un pod puede solicitar.
Container requests: La cantidad mínima de CPU/memoria que un pod requiere para poder funcionar.
Contenedor blob: Azure Blob Storage es la solución de almacenamiento de objetos de Microsoft para la nube. El almacenamiento de blobs está optimizado para almacenar cantidades masivas de datos no estructurados. controller-manager
Control-plane: Los componentes que forman el plano de control de Kubernetes toman decisiones globales sobre el clúster (por ejemplo, la planificación) y detectan y responden a eventos del clúster, como la creación de un nuevo pod cuando la propiedad réplicas de un controlador de replicación no se cumple.

CoreDNS: CoreDNS es un servidor DNS flexible y extensible que puede servir como el DNS del clúster de Kubernetes. Al igual que Kubernetes, el proyecto CoreDNS está alojado en CNCF.
CNCF: Esta organización es la encargada de poner orden en un universo creciente de proyectos de código abierto y de compañías que tratan de crear herramientas que apoyen a los desarrolladores de software a correr sus aplicaciones en la nube.
Curl: cURL es un proyecto de software consistente en una biblioteca y un intérprete de comandos orientado a la transferencia de archivos.
CRD: Se trata de un controlador específico de aplicaciones que amplía las funciones de la API de Kubernetes para crear, configurar y gestionar las instancias de aplicaciones complejas en nombre de un usuario de la plataforma.
CVS: Concurrent Versions Systems o CVS, es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros, que forman un proyecto y permite que distintos desarrolladores colaboren
Deployment: Una implementación de Kubernetes es un objeto que proporciona actualizaciones declarativas a las aplicaciones. Una implementación le permite describir el ciclo de vida de una aplicación, como qué imágenes usar para la aplicación, la cantidad de pods que debe haber y la forma en que deben actualizarse.
DevOps: DevOps es un acrónimo inglés de development (desarrollo) y operations (operaciones), que se refiere a una metodología de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de sistemas en las tecnologías de la información (IT)
Docker Hub: Docker Hub es la biblioteca y comunidad más grande del mundo para imágenes de contenedores. Las imágenes publicadas incluyen contenido de desarrolladores, proyectos de código abierto y proveedores de software independientes (ISV) que crean y distribuyen su código en contenedores.
Docker Registry: El Registro es una aplicación del lado del servidor altamente escalable y sin estado que almacena y permite distribuir imágenes de Docker.
EC2: Amazon Elastic Compute Cloud (EC2) permite a los usuarios de Amazon alquilar computadores virtuales en los cuales pueden ejecutar sus propias aplicaciones.
EPG: Electronic Program Guide es una de las múltiples prestaciones que ofrece la televisión digital, en la cual se encuentran organizados todos los canales que nos ofrece un sistema de televisión.
endpoint (url): Es una interfaz expuesta por un comunicante o un canal de comunicación.
Etcd: Es un almacenamiento de clave-valor distribuido y confiable para los datos más críticos de un sistema distribuido
FFMpeg: FFmpeg es una colección de software libre que puede grabar, convertir y hacer streaming de audio y vídeo.
Fps: La tasa de fotogramas, expresada como fotogramas por segundo, es la frecuencia a la cual un dispositivo muestra imágenes llamadas fotogramas o cuadros
Frames: Término inglés que significa fotograma, es decir, cada una de las imágenes instantáneas en las que se divide una película de cine que dan sensación de movimiento al ser proyectadas secuencialmente.
Maven: Maven es una herramienta de software para la gestión y construcción de proyectos Java. Tiene un modelo de configuración de construcción simple, basado en un formato XML.
MsA: La arquitectura de microservicios (en inglés, Micro Services Architecture, MSA) es una aproximación para el desarrollo de software que consiste en construir una aplicación como

un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP).
Github: Es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.
Google Cloud Storage: Es un servicio de almacenamiento de archivos en línea RESTful para almacenar y acceder a datos en la infraestructura de Google cloud Platform
GOP: En codificación de vídeo, un grupo de imágenes, o estructura GOP, especifica el orden en el que las imágenes tipo intra e inter son ordenadas.
Grid: La computación en malla (en inglés grid computing) es una tecnología que permite utilizar de forma coordinada recursos heterogéneos (entre ellos procesadores, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado.
GUI: La interfaz gráfica de usuario es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles.
HA: HA es la abreviatura “High Availability”. HA asegura un cierto grado absoluto de continuidad operacional durante un período de medición dado.
Helm: El gestor de paquetes para Kubernetes
Helm Charts: un Chart es una colección de archivos que describen un conjunto relacionado de recursos de Kubernetes y permite implementarlo de manera automática, ocultando las complejidades al administrador IT.
Hipervisor: Un hipervisor o monitor de máquina virtual es una capa de software para realizar una virtualización de hardware que permite utilizar, al mismo tiempo, diferentes sistemas operativos en una misma computadora
HLS: HTTP Live Streaming es un protocolo de comunicaciones de transmisión de velocidad de bits adaptable basado en HTTP desarrollado por Apple Inc.
Horizontal-Pod-Autoescaler (HPA): El escalador automático de pods horizontal en Kubernetes se encarga de escalar automáticamente la cantidad de pods en un controlador de replicación, implementación, conjunto de réplicas o conjunto con estado según la utilización de CPU observada (o, con soporte de métricas personalizadas, en algunas otras métricas proporcionadas por la aplicación).
Host: El término host o anfitrión se usa en informática para referirse a las computadoras u otros dispositivos conectados a una red que proveen y utilizan servicios de ella.
HTTP: El Protocolo de transferencia de hipertexto es el protocolo de comunicación que permite las transferencias de información a través de archivos en la World Wide Web
HTTP PUT: La petición HTTP PUT crea un nuevo elemento o reemplaza una representación del elemento de destino con los datos de la petición.
HTTP GET: El método HTTP GET solicita una representación del recurso especificado. Las solicitudes que usan GET solo deben usarse para recuperar datos (no deben incluir datos).
HTTP POST: El método HTTP POST envía datos al servidor. Una solicitud POST es típicamente enviada por un formulario HTML y resulta en un cambio en el servidor.
HTTPS: Es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto, es decir, es la versión segura de HTTP
IaaS: Infraestructura como servicio (IaaS) se refiere a los servicios de la Nube que proporcionan infraestructura como recursos de informática.
IaC: La infraestructura como código es el proceso de gestión y aprovisionamiento de centros de datos informáticos a través de scripts de código, en lugar de realizar dicho proceso de manera manual e interactiva.

Ingress controller: Un Ingress Controller es básicamente un proxy, que permite redirigir la solicitud externa de un usuario a distintos despliegues (servicios) dentro del cluster de Kubernetes.
Intercoding / Inter-Frame: Hace referencia a la técnica de codificación que explota la redundancia temporal que existe entre imágenes consecutivas.
Intracoding / Intra-Frame: Hace referencia a la técnica de codificación que explota la redundancia espacial que existe en una imagen mediante un análisis frecuencial de la misma.
IPC: La comunicación entre procesos es una función básica de los sistemas operativos. Los procesos pueden comunicarse entre sí a través de compartir espacios de memoria, ya sean variables compartidas o buffers, o a través de las herramientas provistas por las rutinas de
IPTV: La Televisión por Protocolo de Internet se ha convertido en la denominación más común para los sistemas de distribución por suscripción de señales de televisión de pago usando conexiones de banda ancha sobre el protocolo IP.
IoT: La internet de las cosas es un concepto que se refiere a una interconexión digital de objetos cotidianos con Internet.
Javascript: Es un lenguaje de programación ligera, interpretado por la mayoría de los navegadores y que les proporciona a las páginas web, efectos y funciones complementarias a las consideradas como estándar HTML.
JDBC: Java Database Connectivity, más conocida por sus siglas JDBC, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java
JSON: JSON es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript.
jobs: Se refiere a una tarea o trabajo que se construye para ser ejecutada por parte de un nodo.
K3s: es una distribución de Kubernetes certificada y de alta disponibilidad diseñada para cargas de trabajo de producción en ubicaciones remotas desatendidas, con recursos limitados o dentro de dispositivos de IoT
K8s: Kubernetes (K8s) es una plataforma de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones en contenedores
Kernel: Un núcleo o kernel es un software que constituye una parte fundamental del sistema operativo, y se define como la parte que se ejecuta en modo privilegiado.
Key-frame: El Key Frame, es aquel fotograma que se toma como referencia con el fin de solo almacenar esta imagen completa y a partir de ese almacenar los cambios de los siguientes fotogramas en referencia al primero.
Kubeadm: Es una herramienta que nos permite generar el despliegue de un cluster de kubernetes de manera sencilla.
kubectl: Es una herramienta de terminal para ejecutar comandos sobre despliegues clusterizados de Kubernetes
KubeDNS: Ofrece un servidor DNS para que los pods puedan resolver diferentes nombres de recursos a direcciones IP. Esta era la versión estable en versiones de Kubernetes 1.11 o más antiguas.
Kubernetes apiserver: El servidor de la API es el componente del plano de control de Kubernetes que expone la API de Kubernetes. Se trata del frontend de Kubernetes, recibe las peticiones y actualiza acordemente el estado en etcd.
Kubeval: Kubeval se utiliza para validar uno o más archivos de configuración de Kubernetes y, a menudo, se utiliza localmente como parte de un flujo de trabajo de desarrollo, así como en procesos de CI.

KWh: El kilovatio-hora, equivalente a mil vatios-hora, se usa generalmente para la facturación del consumo eléctrico domiciliario.
Linux-Vserver: Es una implementación de servidor privado virtual hecha por el agregado de capacidades de virtualización en el ámbito de Sistema Operativo y distribuida como software libre.
Load Balancer: Se refiere a la técnica usada para compartir el trabajo a realizar entre varios procesos, ordenadores, discos u otros recursos.
DVM: Dalvik es la máquina virtual que utiliza la plataforma para dispositivos móviles Android. La Máquina Virtual Dalvik permite ejecutar aplicaciones programadas en Java.
MWh: El megavatio-hora, igual a un millón de Wh, suele emplearse para medir el consumo de grandes plantas industriales o de conglomerados urbanos.
MPEG (Moving Picture Experts Group): Es un grupo de trabajo de expertos que se formó para establecer estándares para el audio y la transmisión video.
MPEG-DASH: Es una técnica de transmisión de video basada en la velocidad adaptativa de bits que permite la transmisión de contenido multimedia de alta calidad a través de Internet desde servidores web HTTP convencionales.
Namespaces: Los espacios de nombres de Kubernetes ayudan a diferentes proyectos, equipos o clientes a compartir, aisladamente, un clúster de Kubernetes.
NFS: Es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuidos en un entorno de red de computadoras de área local.
NodePort: Expone el Servicio en la IP de cada Nodo en un puerto estático (el NodePort). Un servicio ClusterIP, al que se enruta el servicio NodePort, se crea automáticamente.
Okteto: Es una plataforma gratuita (limitada) para hacer pruebas con un cluster de Kubernetes en la nube. El objetivo es reducir la configuración local necesaria para levantar un cluster de Kubernetes y eliminar los problemas de integración desarrollando de la misma manera que su aplicación se ejecuta en producción. Por último, su ventaja esencial es que está disponible en la nube y no es necesario exponer su máquina local a Internet a través de túneles remotos.
OTT: un servicio de libre transmisión o servicio OTT consiste en la transmisión de audio, vídeo y otros contenidos a través de Internet sin la implicación de los operadores tradicionales en el control o la distribución del contenido
PassMark: Es una empresa de software que crea utilidades de software para realizar pruebas comparativas en un sistema informático
PCM: La modulación por impulsos codificados es un procedimiento de modulación utilizado para transformar una señal analógica en una secuencia de bits (señal digital).
PaaS: La Plataforma como Servicio proporciona un marco que los desarrolladores pueden ampliar para desarrollar o personalizar aplicaciones basadas en la nube.
Pod: La unidad de computación más pequeña implementable en Kubernetes se llama pod
pods. Esto define un grupo de uno o más contenedores (como contenedores Docker), con almacenamiento/red compartidos, y unas especificaciones de cómo ejecutar los contenedores.
Preset: es una colección de opciones que proporcionan una cierta velocidad de codificación a una relación de compresión. Esto significa que, por ejemplo, si tiene como objetivo un determinado tamaño de archivo o una velocidad de bits constante, obtendrá una mejor calidad con un ajuste preestablecido más lento.
RTSP: RTSP es un protocolo no orientado a conexión. En lugar de esto el servidor mantiene una sesión asociada a un identificador.

Proxy: es un servidor —programa o dispositivo—, que hace de intermediario en las peticiones de recursos que realiza un cliente a otro servidor.
Spring Boot: Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java.
QoE: La calidad de experiencia se define como la aceptabilidad global de una aplicación o servicio, tal y como se percibe subjetivamente por el usuario final.
QoS: La calidad de servicio es el rendimiento promedio de una red de telefonía o de computadoras, particularmente visto a nivel de latencia o retardo de red
QoS classes: La clase Calidad de servicio (QoS) es un concepto de Kubernetes que determina la prioridad de programación y desalojo de los pods. El programador de Kubernetes utiliza la clase QoS para tomar decisiones sobre la programación de pods en nodos.
Rancher: Rancher es una pila de software completa para equipos que adoptan contenedores. Aborda los desafíos operativos y de seguridad de administrar múltiples clústeres de Kubernetes, al tiempo que brinda a los equipos de DevOps herramientas integradas para ejecutar cargas de trabajo en contenedores.
ReplicaSet: Un ReplicaSet es uno de los controladores de Kubernetes que se asegura de que tengamos un número específico de réplicas de pod en ejecución.
resource metrics
ResourceQuota: Proporciona restricciones que limitan la cantidad de objetos que se pueden crear en un espacio de nombres por tipo, así como la cantidad total de recursos informáticos que pueden consumir los recursos en ese espacio de nombres.
RGB: Es la composición del color en términos de la intensidad de los colores primarios de la luz.
RISC: es un tipo de diseño de CPU en el cual las Instrucciones son de tamaño fijo y presentadas en un reducido número de formatos. Además, sólo las instrucciones de carga y almacenamiento acceden a la memoria de datos.
Scheduler (Kubernetes): El programador de Kubernetes es parte de la plataforma de orquestación de contenedores de Kubernetes de código abierto que controla el rendimiento, la capacidad y la disponibilidad a través de políticas y conocimiento de la topología.
Traefik: Traefik es un equilibrador de carga y proxy inverso HTTP moderno que facilita la implementación de microservicios.
SAS: Una firma de acceso compartido (SAS) es una URI que otorga derechos de acceso restringidos a los recursos de Azure Storage. Al distribuir un URI de firma de acceso compartido a estos clientes, puede otorgarle acceso a un recurso durante un período de tiempo específico, con un conjunto específico de permisos.
SaaS: Es una forma de cloud computing que ofrece a los usuarios una aplicación en la nube junto con toda su infraestructura de TI y plataformas subyacentes. Todo el sistema y la complejidad del mismo se oculta al cliente.
SLA: Es un acuerdo escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio.
SLO: Los SLO se acuerdan como un medio para medir el desempeño del Proveedor de servicios y se describen como una forma de evitar disputas entre las dos partes basadas en malentendidos
SSO: Es un procedimiento de autenticación que habilita a un usuario determinado para acceder a varios sistemas con una sola instancia de identificación.
Streaming: Los medios de transmisión por secuencias son multimedia que se reciben y presentan constantemente a un usuario final mientras los entrega un proveedor.

Superbloques (SB): Un superbloque es el bloque de codificación más grande que puede procesar el códec. Un superbloque puede dividirse aún en más pequeños bloques de codificación, cada uno con sus propios modos de predicción y transformación.
SVN: Subversion está diseñado para administrar y controlar archivos y directorios y realizar un seguimiento de los cambios realizados en ellos; actúa como una “máquina del tiempo confiable” y una herramienta de gestión para los proyectos desarrollados en colaboración.
TCP: TCP (Protocolo de Control de Transmisión, por sus siglas en inglés Transmission Control Protocol) es un protocolo de red importante que permite que dos anfitriones (hosts) se conecten e intercambien flujos de datos. TCP garantiza la entrega de datos y paquetes en el mismo orden en que se enviaron. En el modelo TCP/IP se ubica en la capa 4 (Capa de transporte)
TDP: El TDP es una sigla que hace referencia a Thermal Design Power (Potencia de diseño térmico) y es una especificación medida en vatios que está en la mayoría de los procesadores del mercado. Con esta especificación el fabricante nos indica la cantidad máxima de calor que se espera que un componente produzca en un escenario de uso intenso.
Team Foundation Server (TFS): Team Foundation Server (comúnmente abreviado como TFS) es un producto de Microsoft que proporciona administración de código fuente (ya sea a través de Team Foundation Version Control o Git),
time lapse
TWh: El teravatio-hora (TWh) es utilizado habitualmente, para referirse a las energías producidas por las centrales eléctricas durante un cierto período.
UDP: Es un protocolo del nivel de transporte basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera.
UnionFS: Para combinar diferentes capas de una imagen Docker y tratarlas como una sola, se utiliza un sistema de archivos especial llamado Union File System (UnionFS) que permite combinar archivos y directorios de diferentes sistemas en un único recurso consistente.
VBR: La tasa de bits variable es un término usado en telecomunicación que se refiere a la tasa de bits utilizados en la codificación de audio o video. Este método de compresión consigue una mayor calidad de sonido o video para un tamaño de archivo determinado, en contraste con CBR.
VCS: Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas
VM: una máquina virtual es un software que simula un sistema de computación y puede ejecutar programas como si fuese una computadora real.
x86: Procesadores compatibles con el juego de instrucciones Intel 8086.
XaaS: El término XaaS fue creado para expresar la idea de “Algo como un servicio” o “todo como un servicio”. Este acrónimo se refiere a un número creciente de servicios que se suministran a través de Internet en lugar de hacerlo de forma local. XaaS viene a formar parte de la esencia de la computación en la nube.
YAML: YAML Ain't Markup Language, es un lenguaje de serialización de datos legible por humanos. Se usa comúnmente para archivos de configuración y en aplicaciones donde se almacenan o transmiten datos.
YUV: YUV es un espacio de color usado habitualmente como parte de un sistema de procesamiento de imagen en color

Swap: En informática, el espacio de intercambio (también conocido como archivo de paginación o memoria virtual) es una zona del disco (un fichero o partición) que se usa para guardar las imágenes de los procesos que no pueden mantenerse en memoria física.
Web Server: Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente.
Wh: El vatio-hora, simbolizado Wh es una unidad de energía expresada en forma de unidades de potencia por tiempo, con lo que se da a entender que la cantidad de energía de la que se habla es capaz de producir y sustentar una cierta potencia durante un determinado tiempo.

Referencias

Para una mejor organización, las referencias utilizadas en este trabajo de investigación se categorizaron por unidad temática, dividiendo el material en los siguientes apartados:

- Microservicios
- DevOps
- Virtualización, Contenedores y Orquestadores de Contenedores
- Sistemas y plataformas de procesamiento distribuido
- Arquitecturas y sistemas HPC
- Análisis de consumo de Energía y Computación Verde
- Computación en la Nube
- Computación Móvil y procesamiento basado en ARM
- Video digital, códecs, streaming y transcodificación de video

Microservicios

[ABD19] L. Abdollahi Vayghan et al. (2019). Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes. 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Sofia, Bulgaria, 2019, pp. 176-185

[DRA17] Dragoni Nicola et al. (2017). Microservices: Yesterday, today, and tomorrow. in Present and Ulterior Software Engineering, Cham: Springer International Publishing, 2017, pp. 195–216, isbn: 9783319674254.

[FET16] C. Fetzer (2016). Building Critical Applications Using Microservices. in IEEE Security & Privacy, vol. 14, no. 6, pp. 86-89, Nov.-Dec. 2016

[GAR20] Blaine Gardner y Alexandra Settle (2020). Rook Best Practices for Running Ceph on Kubernetes. Última vez visitado el 8 de enero de 2021. Recurso: https://documentation.suse.com/sbp/all/pdf/SBP-rook-ceph-kubernetes_color_en.pdf

[HAS20] Hassan S., et al. (2020). Microservice transition and its granularity problem: A systematic mapping study. Software Practice and Experience, Volume 1 – No 1, pp 1-36.

[MAU19] Mauersberger Laura (2019). Microservices: What They Are and Why Use Them. 2019. LeanIX. Última vez visitado el 8 de enero de 2021. Recurso: https://www.leanix.net/en/blog/a-brief-history-of-microservices

[MER20] P. Merson y J. Yoder (2020). Modeling Microservices with DDD. 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), Salvador, Brazil, 2020, pp. 7-8.

[MSV19] Janakiram MSV (2019). Implementing Service Discovery of Microservices with Consul. TheNewStackIO Blog. Última vez visitado el 8 de enero de 2021. Recurso: https://thenewstack.io/implementing-service-discovery-of-microservices-with-consul/

[NEW14] Newman Sam (2014). Building Microservices: Designing Fine-Grained Systems. O'Really Systems - 978-1-491-95035-7. 2014.

[RAH18b] A. Rahman (2018). Anti-Patterns in Infrastructure as Code. 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), Vasteras, 2018, pp. 434-435.

[RIC18] Richardson Chris (2018). Microservices Patterns: With examples in Java. Manning. 2018.

[RIC18b] Richardson Chris (2018). Microservice Architecture - What are microservices? Microservices.io. Última vez visitado el 8 de enero de 2021. Recurso: https://microservices.io/

[SHA21] Sharma Rahul y Askshay Matur (2021). Traefik API Gateway for Microservices. Apress, Melbourne, VIC, Australia.

[TAB19] Tabbaa Bishr (2019). Anti-Patterns of Microservices". ITNext. 2019. Última vez visitado el 8 de enero de 2021. Recurso: https://itnext.io/anti-patterns-of-microservices-6e802553bd46

[VAY19] Vayghan L. et al. (2019). Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes. 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Sofia, Bulgaria, pp. 176-185.

[VIG18] Markos Viggiato et al. (2018). Microservices in Practice: A Survey Study. VEM 2018 - 6th Workshop on Software Visualization, Evolution and Maintenance, Sep 2018, Sao Carlos, Brazil

DevOps

[AGR19] P. Agrawal y N. Rawat (2020). Devops, A New Approach To Cloud Development & Testing. 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), GHAZIABAD, India, 2019, pp. 1-4.

[ARA18] Arachchi S.A.I.B.S y Perera Indika (2018). Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. 2018 Moratuwa Engineering Research Conference (MERCon) (pp. 156-161). Moratuwa: IEEE. 2018.

[ART17] M. Artac et al (2017). DevOps: Introducing Infrastructure-as-Code. 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, 2017, pp. 497-498.

[BRI19] Brikman Yevgeniy (2019). Terraform: Up & Running: Writing Infrastructure as Code. O'Reilly Media. ISBN 9781492046905

[CAN14] Javier Cánovas y Jordi Cabot (2014). Composing JSON-based Web APIs. ICWE 2014 - 14th International Conference on Web Engineering, Jul 2014, Toulouse, France. pp.390-399.

[CAU13] Caum Carl (2013). Continuous Delivery Vs. Continuous Deployment: What's the Diff? Puppet. 2013. Última vez visitado el 8 de enero de 2021. Recurso: https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff

[CHE15] Chef Software (2015). Automation and the DevOps Workflow. Chef Software. 2015. Última vez visitado el 8 de enero de 2021. Recurso: https://pages.chef.io/rs/255-VFB-268/images/automation-and-the-devops-workflow.pdf

[CHE16] L. Chen (2016). Continuous Delivery: Overcoming Adoption Obstacles. 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), Austin, TX, 2016, pp. 84-84.

[CHE18] L. Chen (2018). Continuous Delivery at Scale: Challenges and Opportunities. 2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE), Gothenburg, Sweden, 2018, pp. 42-42.

[DAV16] Davis Jennifer y Daniels Ryn (2016). Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale. O'Reilly media - ISBN: 1491926309, 978-1491926307. 2016

[DEJ17] Desjardins Jeff (2017). Infographic: How Many Millions of Lines of Code Does It Take? VisualCapitalist. 2017. Última vez visitado el 8 de enero de 2021. Recurso: http://www.visualcapitalist.com/millions-lines-of-code/

[DES19] de Siebra Clauirton et al. (2018). From Theory to Practice: The Challenges of a DevOps Infrastructure as Code Implementation. Proceedings of the 13th International Conference on Software Technologies, ICSOFT, Porto, Portugal, July 26-28.

[DRI19] Driscoll Bart (2019). Unlock the Benefits of Cloud Platforms and Automation with IaC. Global Innovation Lead – Dell Technologies Consulting. 2019. Última vez visitado el 8 de enero de 2021. Recurso: https://infocus.dellemc.com/bart_driscoll/unlock-the-benefits-of-cloud-platforms-and-automation-with-iac/

[ERI17] Erich, F. M. A., Amrit, C., y Daneva, M. (2017). A qualitative study of DevOps usage in practice. Journal of Software: Evolution and Process, 29(6), 1885.

[FAR14] Farroha Bill and Farroha Dayla (2014). A framework for managing mission needs, compliance, and trust in the DevOps environment. in Proceedings - IEEE Military Communications Conference MILCOM, 2014, pp. 288–293.

[FIT17] Fitzgerald Brian y Stol Klaas-Jan (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123 , 176-189. 2017.

[FOW06] Fowler Martín (2006). Continuous Integration. MartinFowler.com. 2006. Última vez visitado el 8 de enero de 2021. Recurso: https://martinfowler.com/articles/continuousIntegration.html

[HEN12] Hendrix Valerie, Benjamin Doug y Yao Yushu (2012). Scientific Cluster Deployment and Recovery–Using puppet to simplify cluster management. Journal of Physics Conference Series 396 (4). 2012.

[HUM10] Humble de Jez y Farley David (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional - ISBN:978-0-321-60191-9.

[HUT12] Hüttermann Michael (2012). Devops for developers. New York: Apress. 2012.

[ITR19] IT Revolution Press (2019). The IT Revolution DevOps Guide - Selected Resources to Start Your Journey. IT Revolution Press. 2019. Última vez visitado el 8 de enero de 2021. Recurso: http://athena.ecs.csus.edu/~buckley/CSc233/DevOpsGuide.pdf

[JAR16] Jaramillo David et al. (2016). Leveraging microservices architecture by using Docker technology. In SoutheastCon 2016 (pp. 1–5).

[KAR19] Karmarkar Sagar (2019). DevOps Pipelines Part 3: Continuous Monitoring and Observability. Centric Consulting. Última vez visitado el 8 de enero de 2021. Recurso: https://centricconsulting.com/blog/devops-pipelines-part-3-continuous-monitoring-and-observability/

[KIM15] Kim Gene et al (2015). The phoenix project: A novel about it, devops, and helping your business win. Portland: IT Revolution -978-0988262508. 2015.

[KIM16] Kim Gene et al. (2016). The devops handbook. Portland: IT Revolution Press - ISBN: 1942788002,9781942788003. 2016. Última vez visitado el 8 de enero de 2021. Recurso: http://images.itrevolution.com/documents/DevOps_Handbook_Intro_Part1_Part2.pdf

[KLE19] Klein John y Reynolds Doug (2019). Infrastructure as Code: Final Report. Software Engineering Institute - White Paper. 2019. Última vez visitado el 8 de enero de 2021. Recurso: https://resources.sei.cmu.edu/asset_files/WhitePaper/2019_019_001_539335.pdf

[LOO11] Loope James (2011). Managing Infrastructure with Puppet". O'Reilly Media - ISBN: 9781449309671. 2011.

[MIK19] Krief Mikael (2019). Learning DevOps: The complete guide to accelerate collaboration with Jenkins, Kubernetes, Terraform and Azure DevOps. Packt Publishing. ISBN 978-1838642730.

[NIE17] Nielsen Pia Arentoft, Winkler Till y Nørbjerg Jacob (2017). Closing the IT development-operations gap: The DevOps knowledge sharing framework. in CEUR Workshop Proceedings, B. Johansson, Ed., vol. 1898, CEUR, 2017, isbn: 16130073.

[NIS11] NIST (2011). Final Version of NIST Cloud Computing Definition. 2011. Última vez visitado el 8 de enero de 2021. Recurso: www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published

[PLW15] Plwakatare, L. E., Kuvaja, P., y Oivo, M (2015). Dimensions of DevOps. Proceedings of the international Conference on Agile Software Development, Tallinn, 212-217. 2015.

[RAH18] A. Rahman (2018) Characteristics of Defective Infrastructure as Code Scripts in DevOps. 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), Gothenburg, 2018, pp. 476-479.

[RED18] RedHat (2018) HashiCorp Terraform and Red Hat Ansible Automation - Infrastructure as code automation. Última vez visitado el 8 de enero de 2021. Recurso: https://www.redhat.com/cms/managed-files/pa-terraform-and-ansible-overview-f14774wg-201811-en.pdf

[SHA18] J. Shah, D. Dubaria y J. Widhalm (2018). A Survey of DevOps tools for Networking. 2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York City, NY, USA, 2018, pp. 185-188

[SUD12] Pandey Sudhir (2012). Investigating Community, Reliability and Usability of CFEngine, Chef and Puppet. Network and System Administration Oslo and Akershus University College. 2012.

[WRI19] Wright Eric (2019). DevOps Automation with Terraform and VMware. O'Reilly Media, Inc. ISBN: 9781492073758

Virtualización, Contenedores y Orquestadores de Contenedores

[AND20] Andersson, Johan y Norrman, Fredrik (2020). Container orchestration - the migration path to Kubernetes. DIVA - Department of computer science and media technology (CM)

[ARU19] Arundel J. y Domingus J., (2019). Cloud Native DevOps with Kubernetes. O'Reilly Media, California, USA

[BHA17] Bhatia, Gaurav et al. (2017). THE ROAD TO DOCKER: A SURVEY. International Journal of Advanced Research in Computer Science. 8. 83-87.

[BUC19] Buchanan S., et al (2019). Introducing Azure Kubernetes Service: A Practical Guide to Container Orchestration. Apress, California, USA.

[BUR16] Brendan Burns et al (2016). Borg, Omega, and Kubernetes. Google Inc - System Evolution. Última vez visitado el 8 de enero de 2021. Recurso: https://static.googleusercontent.com/media/research.google.com/es//pubs/archive/44843.pdf

[BUR19] Burns B., et al. (2019). Kubernetes Up and Running: Dive into the Future of Infrastructure (2nd edition). O'Reilly Media, California, USA

[DEW19] Dewi L., et al. (2019). Server Scalability Using Kubernetes. 2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON), Bangkok, Thailand, pp 1-4.

[DIO20] Diouf, G. M., Elbiaze, H., y Jaafar, W. (2020). On Byzantine fault tolerance in multi-master Kubernetes clusters. Future Generation Computer Systems, 109, 407–419.

[DOB20] Dobies J. y Wood J, (2020). Kubernetes Operators - Automation the container Orchestration Platform. O'Reilly Media, California, USA

[DON20] I. Donca et al. (2020). Autoscaled RabbitMQ Kubernetes Cluster on single-board computers. 2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 2020, pp. 1-6.

[DUA20] A. Dua, S. Randive, A. Agarwal y N. Kumar (2020). Efficient Load balancing to serve Heterogeneous Requests in Clustered Systems using Kubernetes. 2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2020, pp. 1-2

[ESP20] Lennart Espe et al. (2020). Performance Evaluation of Container Runtimes. 10th International Conference on Cloud Computing and Services Science.

[FEL15] Felter Wes et al. (2015). An updated performance comparison of virtual machines and linux containers. In: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On. IEEE. 2015, pp. 171–172

[GER15] W. Gerlach et al (2015). Container Orchestration for Scientific Workflows. 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ, 2015, pp. 377-378

[GOO21] Google Cloud (2021). Prácticas recomendadas para ejecutar aplicaciones de Kubernetes con optimización de costos en GKE. Google Cloud. Última vez visitado el 8 de enero de 2021. Recurso: https://cloud.google.com/solutions/best-practices-for-running-cost-effective-kubernetes-applications-on-gke

[HAN20] Handa Michael (2020). How to Scale Using Kubernetes: From Startup to Superstar. Bluesentry. Última vez visitado el 8 de enero de 2021. Recurso: https://bluesentryit.com/how-to-scale-using-kubernetes-from-startup-to-superstar/

[HEZ20] Z. He (2020). Novel Container Cloud Elastic Scaling Strategy based on Kubernetes. 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 2020, pp. 1400-1404

[IFR19] Ifrah S., 2019. Deploy Containers on AWS: With EC2, ECS, and EKS. Apress. California. USA.

[JAW19] I. M. A. Jawarneh et a. (2019) Container Orchestration Engines: A Thorough Functional and Performance Comparison. ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 2019, pp. 1-6

[KAP20] N. Kapočius (2020). Performance Studies of Kubernetes Network Solutions. 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2020, pp. 1-6

[LAR19] Olle Larsson (2019) Running Databases in a Kubernetes cluster - An evaluation. Umea University - Master thesis.

[LAR20] Lars Larsson (2020) Impact of etcd Deployment on Kubernetes, Istio, and Application Performance. arXiv:2004.00372

[MAL19] A. M. Maliszewski et al (2019). Minimizing Communication Overheads in Container-based Clouds for HPC Applications. 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, pp. 1-6

[MAT18] Matthias Karl y Kane Sean (2018). Docker: Up & Running - 2nd Edition, Shipping Reliable Containers in Production. O'Reilly Media, Inc. 2018.

[MOU16] Mouat Adrian (2016). Using Docker. O'Reilly Media, Inc. ISBN: 9781491915752. 2016

[NEC15] Nectarios Koziris (2015). Fifty years of evolution in virtualization technologies: from the first IBM machines to modern hyperconverged infrastructures. In: Proceedings of the 19th Panhellenic Conference on Informatics. ACM. 2015, pp. 3–4.

[NET20] NetApp (2020). Persistent Volumes, Dynamic Provisioning, Cloud Storage. The NetApp Guide to Kubernetes. Última vez visitado el 8 de enero de 2021. Recurso: https://cloud.netapp.com/hubfs/Guide-Kubernetes/The-NetApp-Guide-to-Kubernetes.pdf

[NGU20] Nguyen Nam Xuan (2020) Network isolation for Kubernetes hard multi-tenancy. Master's Thesis Aalto University, School of Science Master's Programme in Security and Cloud Computing.

[NGU20b] Nguyen Thanh-Tung et al (2020). Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration.Sensors 2020, 20, 4621.

[PEN10] Peng Li (2010). Centralized and decentralized lab approaches based on different virtualization models. In: Journal of Computing Sciences in Colleges 26.2 (2010), pp. 263–269.

[PER19] A. Pereira Ferreira y R. Sinnott (2019). A Performance Evaluation of Containers Running on Managed Kubernetes Services. 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Sydney, Australia, 2019, pp. 199-208.

[PHI21] Philippe Martin (2021). Kubernetes: Preparing for the CKA and CKAD Certifications. Apress. ISBN 978-1-4842-6494-2.

[POT05] Potz Herbert, Anderson Micah y Steinbrink Bjorn. (2005). Linux-VServer Resource efficient context isolation. Free Software Magazine Issue 5, June 2005.
[POT07] Pötz Herbert y Fiuczynski Marc (2007). Linux-VServer Resource Efficient OS-Level Virtualization. Proceedings of the Linux Symposium. Volume Two. June 27th–30th, Ottawa, Ontario Canada. 2007.
[PRI04] Price Daniel y Tucker Andrew (2004). Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In: LISA. Vol. 4. 2004, pp. 241– 254.
[RAD17] Rad B., et al. (2017). An Introduction to Docker and Analysis of its Performance. IJCSNS International Journal of Computer Science and Network Security, Volume 17 - No.3.
[RAN20] Rancher (2020). K3s - Lightweight Kubernetes. Última vez visitado el 8 de enero de 2021. Recurso: https://rancher.com/docs/k3s/latest/en/
[RAU21] Alexander Raul (2021). Cloud Native with Kubernetes: Deploy, configure, and run modern cloud native applications on Kubernetes. Packt Publishing. ISBN 9781838823078
[RED21] Red Hat Documentation (2021). Automatically scaling pods. Red Hat Openshift. Última vez visitado el 8 de enero de 2021. Recurso: https://docs.openshift.com/dedicated/4/nodes/pods/nodes-pods-autoscaling.html#nodes-pods-autoscaling-creating-cpu_nodes-pods-autoscaling
[REE20] Mark Reed (2020). Kubernetes: The Ultimate Beginners Guide to Effectively Learn Kubernetes Step-by-Step. Publishing Factory, 2020. ISBN-13 : 978-1647710910
[ROB00] Robin John S y Irvine Cynthia (2004). Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. Tech. rep. DTIC Document, 2000.
[ROD18b] Rodriguez Maria A.y Buyya Rajkumar (2018). Containers Orchestration with Cost-Efficient Autoscaling in Cloud Computing Environments. ArXiv Journal, volume abs/1812.00300.
[RUB18] Rubab Zahra Sarfraz (2018). Evaluating Ceph Deployments with Rook. Cern OpenLab. Última vez visitado el 8 de enero de 2021. Recurso: https://zenodo.org/record/1967427/files/Report_Rubab%20Zahra%20Sarfraz.pdf?download=1
[SAN18] Sane, Bernard y Niang, Ibrahima. (2018). A Review of Virtualization, Hypervisor and VM Allocation Security: Threats, Vulnerabilities, and Countermeasures. 1317-1322.
[SHA08] Shannon Meier et al. (2008). IBM Systems Virtualization: Servers, Storage, and Software. In: IBM Redbooks. 2008. Última vez visitado el 8 de enero de 2021. Recurso: https://www.redbooks.ibm.com/redpapers/pdfs/redp4396.pdf
[SHA20] Shaw, Brandon (2020). Kubernetes Step-by-Step: A beginner's Guide to Build, Scale, Deploy, Manage Production-Ready Kubernetes Clusters and Application. Brandon Shaw.

[SOL07] Soltesz Stephen et al (2007). Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In: ACM SIGOPS Operating Systems Review. Vol. 41. 3. ACM. pp. 275–287. 2007.

[SPI19] Josef Spillner (2019). Quality Assessment and Improvement of Helm Charts for Kubernetes-Based Cloud Applications. arXiv:1901.00644v1

[STA20] Starmer John (2020). A Comprehensive Look at Kubernetes Cluster Autoscaling. Opsani. Última vez consultado 08/01/21. Recurso: https://opsani.com/blog/kubernetes-cluster-autoscaling-overview/

[SWS16] SWsoft (2016). OpenVZ User's Guide. SWsoft , Inc. 2016. Última vez visitado el 8 de enero de 2021. Recurso: https://docs.openvz.org/openvz_users_guide.pdf

[TAM20] Tamiru, Mulugeta et al. (2020). An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud. 12th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)At: Bangkok, Thailand

[TOW19] Townend P., et al. (2019). Improving Data Center Efficiency Through Holistic Scheduling In Kubernetes. 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), pp 156-166

[TRU18] Eddy Truyen et al. (2018). A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks. Applied Sciences. 9. 10.3390/app9050931.

[TRU18b] E. Truyen et al. (2018) Evaluation of Container Orchestration Systems for Deploying and Managing NoSQL Database Clusters. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2018, pp. 468-475

[TUC19] Tuczai Eva y Hertz Asena (2019). Managing Kubernetes performance at scale. Operational best practices. O'Reilly Media, Inc.

[VAS16] Vasconcelos Pedro, Freitas Gisele y Marques Thales (2016). KVM, OpenVZ and Linux Containers: Performance Comparison of Virtualization for Web Conferencing Systems. International Journal Multimedia and Image Processing (IJMIP), Volume 6, Issues 1/2, March/June 2016.

[VAS18] Vikram Vaswani (2018). Deploy a Production-Ready MariaDB Cluster on Kubernetes with Bitnami and Helm. Bitnami. Última vez visitado el 8 de enero de 2021. Recurso: https://engineering.bitnami.com/articles/deploy-a-production-ready-mariadb-cluster-on-kubernetes-with-bitnami-and-helm.html

[VAY19] L. Abdollahi Vayghan et al. (2019). Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes. 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Sofia, Bulgaria, 2019, pp. 176-185

[VER15] Abhishek Verma (2015). Large-scale cluster management at Google with Borg. EuroSys '15: Proceedings of the Tenth European Conference on Computer Systems. April 2015 Article No.: 18 Pages 1–17

[VER19] Verreydt, Stef et al. (2019). Leveraging Kubernetes for adaptive and cost-efficient resource management. Proceedings of the 5th International Workshop on Container Technologies and Container Clouds - WOC '19.

[VMW20] Vmware Inc (2020). Deliver Elastic Kubernetes Ingress Controller and Services - A single platform for consolidated traffic management, security, and observability. Vmware White Paper. Última vez visitado el 8 de enero de 2021. Recurso: https://cdn2.hubspot.net/hubfs/443964/Avi_Website_Resource_Center/AviNetworks_for_OpenShift-Kubernetes.pdf?t=1504706366713

[WAN20] M. Wang, D. Zhang y B. Wu (2020). A Cluster Autoscaler Based on Multiple Node Types in Kubernetes. 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chongqing, China, 2020, pp. 575-579.

[WAT20] Watada J., et al (2020). Emerging Trends, Techniques and Open Issues of Containerization: A Review. IEEE Access, Volume 7, pp. 152443-152472.

[WUQ19] Q. Wu et al. (2019). Dynamically Adjusting Scale of a Kubernetes Cluster under QoS Guarantee. 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), Tianjin, China, 2019, pp. 193-200

[XAV13] Xavier Miguel et al. (2013). Performance evaluation of container-based virtualization for high performance computing environments. In: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. IEEE. 2013, pp. 233–240.

[ZHO20] Zhong Zhiheng y Buyya Rajkumar (2020). A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources. ACM Trans. Internet Technol. 20, 2, Article 15.

Sistemas y plataformas de procesamiento distribuido

[AND04] David P. Anderson (2004). BOINC: A System for Public-Resource Computing and Storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing. GRID. IEEE, 2004, pages 4-10.

[BRE18] Brendan Burns (2018). Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. O'Reilly Inc.

[BRO07] Brogårdh Torgny (2007). Present and future robot control development - An industrial perspective. In: Annual Reviews in Control 31.1 pp. 69–79. 2007.

[CAR86] Carlson B. (1986). Communication Systems. 3rd. Edition. McGraw Hill Book Co. 1986.
[DER07] Y. Derbal (2007). Grid Architecture for High Performance Computing. 2007 Canadian Conference on Electrical and Computer Engineering, Vancouver, BC, 2007, pp. 514-517.
[GIE15] Giessler, Pascal et al. (2015). Best Practices for the Design of RESTful Web Services. Proceedings - International Conference on Software Engineering. 10. 392.
[GRO14] William Gropp et al (2014) Using MPI in Simple Programs. in Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, 2014, pp.23-68.
[GRO15] Groover Mikell (2015). Automation, Production Systems, and Computer-Integrated Manufacturing". 4rd. Upper Saddle River, NJ, USA: Prentice Hall Press. ISBN: 978-0133499612, 0133499618. 2015.
[GUO07] Guo Chang, et al (2007). A framework for native multi-tenancy application development and management. In: e-commerce Technology and the 4th IEEE International Conference on Enterprise Computing, e-commerce, and E-Services, CEC/EEE 2007. The 9th IEEE International Conference on. IEEE. 2007, pp. 551-558
[GUY90] R. G. Guy et al (1990). Name transparency in very large scale distributed file systems. IEEE Workshop on Experimental Distributed Systems, Huntsville, AL, USA, 1990, pp. 20-25.
[KUR07] Kurose J., Ross W. (2007). Computer Networking: A Top-Down Approach. Research Online Publishing - Open Journal of Web Technologies (OJWT) Volume 3, Issue 1, 2007. ISSN 1191-208X.
[LAV19] Lavoie E. y Hendren L. (2019). Personal Volunteer Computing. CF '19: Proceedings of the 16th ACM International Conference on Computing Frontiers, pp 240–246.
[LIN19] Sebastian Lindmark y Andreas Järvelä (2019). Evaluation and comparison of a RabbitMQ broker solution on Amazon Web Services and Microsoft Azure. Linköping University - Department of Computer and Information Science Master thesis.
[MAD19] S. C. Madanapalli, H. Habibi Gharakhieli y V. Sivaraman (2019). Inferring Netflix User Experience from Broadband Network Measurement. 2019 Network Traffic Measurement and Analysis Conference (TMA), Paris, France, 2019, pp. 41-48.
[MAS16] G. Massari, M. Zanella y W. Fornaciari (2016). Towards Distributed Mobile Computing. 2016 Mobile System Technologies Workshop (MST), Milan, 2016, pp. 29-35.
[OOD03] Oodan P. et al. (2003). Telecommunications Quality of Service Management: From Legacy to Emerging. Communication System Software and Middleware, 2003. Comsware 2003. First International Conference on.
[OPP83] Oppenheim A.V., Willsky A.S. y Young T. (1983). Signals and Systems. Prentice Hall International, Inc. 1983.

[PIE20] P. Pierleoni, R. Concetti, A. Belli y L. Palma (2020) Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison. in IEEE Access, vol. 8, pp. 5455-5470.
[PUT93] P. Putter y J. D. Roos (1993). Relationships: implementing transparency in distributed management systems. Proceedings of 1993 IEEE 1st International Workshop on Systems Management, Los Angeles, CA, USA, 1993, pp. 118-124.
[QUI06] David Quigley et al (2006). UnionFS: User- and Community oriented Development of a Unification Filesystem. In Proceedings of the 2006 Linux Symposium.
[ROD16] Rodríguez, C. et al. (2016). REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. Web Engineering, 21–39.
[ROS14] Rostanski, Maciej et al. (2014). Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ. 2014 Federated Conference on Computer Science and Information Systems, Warsaw, 2014, pp. 879-884.
[SAR15] Sari, Arif y Akkaya, Murat. (2015). Fault Tolerance Mechanisms in Distributed Systems. International Journal of Communications, Network and System Sciences. 8. 471-482.
[SAU05] Sauvé, J et al (2005). Optimal Choice of Service Level Objectives from a Business Perspective. In Proceedings of the International Conference on Autonomic Computing.
[WAM14] Brandon Wamboldt (2014). How Linux pipes work under the hood. Understanding Technology. Última vez visitado el 8 de enero de 2021. Recurso: https://brandonwamboldt.ca/how-linux-pipes-work-under-the-hood-1518/
[XIN97] Xingfu Wu y Wei Li (1997). Cluster-oriented software development environment and its applications. Proceedings High Performance Computing on the Information Superhighway. HPC Asia '97, Seoul, South Korea, 1997, pp. 692-695.
[XUA09] W. Xuan, S. Xue y T. Ma (2009). Architecture of Collaborative Design Based on Grid. 2009 Third International Conference on Genetic and Evolutionary Computing, Guilin, 2009, pp. 244-247.
[YEO06] Yeo Chee Shin et al. (2006). Utility computing and global grids. Technical Report, GRIDS-TR-2006-7, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, April 13, 2006.
[ZAM12] Zamboni Diego (2012). Learning CFEngine 3: Automated System Administration for Sites of Any Size. O'Reilly Media - 1449312209, 9781449312206. 2012.
[ZHA00] Zhao W., Olshefksi D. y Schulzrinne H. (2000). Internet Quality of Service: an Overview. Reporte Técnico, Columbia University, feb. 2000.

[AJL18] R. Aljamal, A. El-Mousa y F. Jubair (2018). A comparative review of high-performance computing major cloud service providers. 2018 9th International Conference on Information and Communication Systems (ICICS), Irbid, pp. 181-186

[ALB19] Albu, A.; Precup, R.E. y Teban, T.A (2019). Results and challenges of artificial neural networks used for decision-making and control in medical applications. Facta Universitatis Series Mechanical Engineering 17, 285–308.

[BAR07] Barbara Chapman et al. (2007). Using OpenMP in the Real World. in Using OpenMP: Portable Shared Memory Parallel Programming , MIT Press, 2007, pp.191-242.

[BOL12] L. Bo, Z. Zhenliu y W. Xiangfeng (2012). A Survey of HPC Development. 2012 International Conference on Computer Science and Electronics Engineering, Hangzhou, pp. 103-106

[CAN02] Cant, S. (2002). High-performance computing in computational fluid dynamics: progress and challenges. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 360(1795), 1211–1225.

[CHR20] Christie L. Alappat et al (2020). Understanding HPC Benchmark Performance on Intel Broadwell and Cascade Lake Processors. International Conference on High Performance Computing - ISC High Performance 2020: High Performance Computing pp 412-433.

[COK17] Cockrell, C. (2017). Sepsis reconsidered: Identifying novel metrics for behavioral landscape characterization with a high-performance computing implementation of an agent-based model. J. Theor. Biol. 2017, 430, 157–168.

[FAT08] M. Fatica (2008). CUDA toolkit and libraries. 2008 IEEE Hot Chips 20 Symposium (HCS), Stanford, CA, 2008, pp. 1-22.

[FIS19] Fischer Andreas, Gao David y Singh Vinai (2019). Advances in mathematical methods and high performance computing. Springer.

[GAU15] Gaudiani, Adriana (2015). Simulación y optimización como metodología para mejorar la calidad de la predicción en un entorno de simulación hidrográfica. Tesis de doctorado. Facultad de Informática - Doctorado en Ciencias Informáticas.

[JIN12] Jin, J., et al. (2012). HPC Simulations of Information Propagation Over Social Networks. Procedia Computer Science, 9, 292–301.

[KES18] Kessentini, M. et al (2018). Agent-Based Modeling and Simulation of Inventory Disruption Management in Supply Chain. In Proceedings of the International Conference on High Performance Computing Simulation (HPCS), Orleans, France, 16–20 July 2018; pp. 1008–1014.

[KUR19] Sergei Kurgalin y Sergei Borzunov (2019). A Practical Approach to High-Performance Computing. Springer

[MAT20] Matsuoka Satoshi (2020). Fugaku: Building an Arm-powered Supercomputer. ARM blueprint. Última vez visitado el 8 de enero de 2021. Recurso: https://www.arm.com/blogs/blueprint/fugaku-supercomputer

[MOL19] Molyakov, A. (2019). China Net: Military and Special Supercomputer Centers. Journal Electrical and Electronic Engineering 7(4):95-100

[NAR08] S. Narravula et al (2008). Performance of HPC Middleware over InfiniBand WAN. 2008 37th International Conference on Parallel Processing, Portland, OR, 2008, pp. 304-311.

[OZO17] Ozog, David (2017). High Performance Computational Chemistry: Bridging Quantum Mechanics, Molecular Dynamics, and Coarse-Grained Models. M.University of Oregon, ProQuest Dissertations Publishing 10252897.

[PAN07] D. K. Panda y P. Balaji (2007). Designing high-end computing systems with InfiniBand and10-Gigabit Ethernet iWARP. 2007 IEEE International Conference on Cluster Computing, Austin, TX, 2007, pp. xiv-xiv.

[PET17] Petrocelli, D., De Giusti, A. E. & Naiouf, M. Procesamiento distribuido y paralelo de bajo costo basado en cloud & movil. XXIII Congreso Argentino de Ciencias de la Computación, XVIII Workshop de Procesamiento Distribuido y Paralelo (WPDP), 216–225 (2017).

[PET19] Petrocelli, D., De Giusti, A. E. & Naiouf, M. “Hybrid Elastic ARM&Cloud HPC Collaborative Platform for generic tasks”. VII Conference Cloud Computing and Big Data (La Plata, 2019), Instituto de Investigación en Informática, ISBN: 978-3-030-27713-0, Páginas: 16-27 (2019).

[PET20a] Petrocelli, D., De Giusti, A. E. & Naiouf, M. “Collaborative, distributed and scalable platform based on mobile, cloud, micro services and containers for intensive computing tasks” Short Papers of the 8th Conference on Cloud Computing Conference, Big Data & Emerging Topics (JCC-BD&ET 2020).

[PET20b] Petrocelli, D., De Giusti, A. E. & Naiouf, M. ”Plataforma colaborativa, elástica, de bajo costo y consumo basada en recursos de la Nube, contenedores y móviles para HPC” 7ª Conferencia Ibero Americana Computação Aplicada - 2020 Lisboa, Portugal

[RIC17] Rico, Alejandro et al (2017). Arm HPC Ecosystem and the Reemergence of Vectors. Proceedings of the Computing Frontiers Conference. ACM, 2017.

[ROD18] Rodgers, A.J. et al (2018). Ground Motions for a Magnitude 7.0 Hayward Fault Earthquake with Three-Dimensional Structure and Topography. Rodgers, Geophysical Research Letters, 45(2), 739–747.

[RUH20] A. Ruhela et al (2020). Analyzing and Understanding the Impact of Interconnect Performance on HPC, Big Data, and Deep Learning Applications: A Case Study with InfiniBand EDR and HDR. 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), New Orleans, LA, USA, pp. 869-878,

[RUI18] Ruiz, Daniel, et al. (2018). The HPCG benchmark: analysis, shared memory preliminary improvements and evaluation on an Arm-based platform. (2018): 1-15.
[SHA17] Javed Ahmad Shaheen, Mian Ali Asghar y Abid Hussain (2017). Android OS with its Architecture and Android Application with Dalvik Virtual Machine Review. International Journal of Multimedia and Ubiquitous Engineering. Vol. 12, No. 7 (2017), pp. 19-30.
[SMI18] McIntosh-Smith Simon, et al. (2018). Comparative benchmarking of the first generation of HPC-optimised ARM processors on Isambard. Cray User Group (CUG) Conference. 2018.
[STE17] Thomas Sterling, Maciej Brodowicz y Matthew Anderson (2017). High Performance Computing: Modern Systems and Practices. Morgan Kaufmann. ISBN: 9780124201583.
[STE18] Stenberg Daniel (2018). Everything curl. Fandrich Dan - ISBN: 978-91-639-6501-2.
[STO18] M. Stoffel y A. Mazouz (2018). Improving Power Efficiency Through Fine-Grain Performance Monitoring in HPC Clusters. 2018 IEEE International Conference on Cluster Computing (CLUSTER), Belfast, 2018, pp. 552-561
[STR01] Strey, A. (2001). High Performance Computing. International Encyclopedia of the Social & Behavioral Sciences, 6693–6697.
[THA18] Thabet, Senan y Thabit, Thabit. (2018). Computational Fluid Dynamics: Science of the Future. International Journal of Research and Engineering - Mechanical Engineering. 5. 430-433.
[VOL17] Volokhov, V.M. et al (2017). Supercomputer simulation of nanocomposite components and transport processes in the Li-ion power sources of new types. In Russian Supercomputing Days; Communications in Computer and Information Science, Germany, 2017; pp. 299–312.
[WAN18] Wang, Y. et al (2018). An efficient parallel algorithm for the coupling of global climate models and regional climate models on a large-scale multi-core cluster. J. Supercomput. 2018, 74, 3999–4018.
[WEN00] Wenzhang Zhu, D. Lee y Cho-Li Wang (2000). High performance communication subsystem for clustering standard high-volume servers using Gigabit Ethernet. Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, Beijing, China, 2000, pp. 184-189 vol.1
[WOO11] Woodley, S. M., y Catlow, C. R. A. (2011). High-performance computing in the chemistry and physics of materials. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 467(2131), 1880–1884.
[YOK19] Yokoyama, Daniel et al. (2019). The survey on ARM processors for HPC. The Journal of Supercomputing 75(1):1-34

[YOU17] A. J. Younge et al. (2017). Enabling Diverse Software Stacks on Supercomputers Using High Performance Virtual Clusters. 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, 2017, pp. 310-321
[ZOU18] P. Zou, D. Rodriguez y R. Ge (2018). Maximizing Throughput on Power-Bounded HPC Systems. 2018 IEEE International Conference on Cluster Computing (CLUSTER), Belfast, 2018, pp. 156-157.

Análisis de consumo de Energía y Computación Verde

[AHM19] Ahmed K. et al. (2019). Electrical Energy Consumption Model of Internal Components in Data Centers. 2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe), Bucharest, Romania, pp 1-5.
[AND20] Etienne André, et al. (2020). DUF : Dynamic Uncore Frequency scaling to reduce power consumption. 2020. hal-02401796v2f.
[ASK14] Akshay, G y Mahajan, Miss (2014). Lokhande Bhagyashree and Girish Kumar Patnaik. Green Computing Metrics, Methods and Models. International journal of engineering research and technology 3.
[BAR19] Barua H. y Mondal D. (2019). Approximate Computing: A Survey of Recent Trends - Bringing Greenness to Computing and Communication. The Journal of the Institution of Engineers: Electronics and Telecommunication, pp 619-626
[BOU16] Katherine Bourzac (2016). Intel: Chips Will Have to Sacrifice Speed Gains for Energy Savings. MIT Technology Review. Última vez visitado el 8 de enero de 2021. Recurso: https://www.technologyreview.com/2016/02/05/9327/intel-chips-will-have-to-sacrifice-speed-gains-for-energy-savings/
[CUT21] Cutress Ian (2021). Intel Core i7-10700 vs Core i7-10700K Review: Is 65W Comet Lake an Option?. Última vez visitado el 8 de enero de 2021. Recurso: https://www.anandtech.com/show/16343/intel-core-i710700-vs-core-i710700k-review-is-65w-comet-lake-an-option/2
[FIS15] L. M. Fischer, L. B. d. Brisolara y J. C. Balzano De Mattos (2015). SEMA: An Approach Based on Internal Measurement to Evaluate Energy Efficiency of Android Applications. 2015 Brazilian Symposium on Computing Systems Engineering (SBESC), Foz do Iguacu, 2015, pp. 48-53, doi: 10.1109/SBESC.2015.16.
[FLA20] Flamm Kenneth (2020). Measuring Moore's Law: Evidence from Price, Cost, and Quality Indexes. University of Texas at Austin. Última vez visitado el 8 de enero de 2021. Recurso: https://www.nber.org/system/files/chapters/c13897/c13897.pdf
[FRU20] Frumusanu Andrei (2020). The Snapdragon 865 Performance Preview: Setting the Stage for Flagship Android 2020. Anandtech. Última vez visitado el 8 de enero de 2021.

Recurso: https://www.anandtech.com/show/15207/the-snapdragon-865-performance-preview-setting-the-stage-for-flagship-android-2020

[HES14] Kevin Heslin (2014). 2014 Data Center Industry Survey. Uptime Institute. Última vez visitado el 8 de enero de 2021. Recurso: https://journal.uptimeinstitute.com/2014-data-center-industry-survey/

[IEE91] IEE Colloquium (1991). IEE Colloquium on 'RISC Architectures and Applications'. (Digest No.163) IEE Colloquium on RISC Architectures and Applications, London, UK, 1991, pp. 0-10.

[JAC19] Adrian Jackson et al. (2019). Evaluating the Arm Ecosystem for High Performance Computing. PASC '19: Proceedings of the Platform for Advanced Scientific Computing Conference. Article No.: 12 Pages 1–11

[JAR13] Jarus, Mateusz et al. (2013). Performance Evaluation and Energy Efficiency of High-Density HPC Platforms Based on Intel, AMD and ARM Processors. Proc. of the Intl. Conf. on Energy Efficiency in Large Scale Distributed Systems (EE-LSDS'13)

[KIP11] Kipp, A., Jiang, T., y Fugini, M. (2011). Green Metrics for Energy-aware IT Systems. 2011 International Conference on Complex, Intelligent, and Software Intensive Systems.

[KOU20] Koutsovasilis, P. et al. (2020). Dynamic Undervolting to Improve Energy Efficiency on Multicore X86 CPUs. IEEE Transactions on Parallel and Distributed Systems, 1–1.

[LAG14] Lago, P., Qing Gu y P. Bozzelli (2014). A systematic literature review of green software metrics. Information and Software Technology 73.

[MAH11] P. Mahadevan, et al. (2011). On energy efficiency for enterprise and data center networks. IEEE Communications Magazine 49 (2011) 94–100.

[MAS20] Masanet E., et al. (2020). Recalibrating global data center energy-use estimates. Science, Volume 367, Issue 6481, pp 984-986.

[MOD15] Modassir Anis et al. (2015). Survey on Green Computing. International Journal of Advanced Technology in Engineering and Science. Volume No 03, Special Issue No. 01.

[NUC17] D. Di Nucci et a. (2017). PETrA: A Software-Based Tool for Estimating the Energy Profile of Android Applications. 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, 2017, pp. 3-6.

[NUO20] Nuoa Lei y Eric Masanet (2020). How Much Energy Do Data Centers Really Use?. McCormick School of Engineering and Applied Science. Última vez visitado el 8 de enero de 2021. Recurso: https://energyinnovation.org/2020/03/17/how-much-energy-do-data-centers-really-use/#:~:text=Data%20centers%20can%20be%20thought,%E2%80%9Cbrains%E2%80%9D%20of%20the%20internet.&text=Some%20of%20the%20world's%20largest,households%20(U.S.%20DOE%202020).

[OUZ12] Z. Ou, et al. (2012) Energy- and Cost-Efficiency Analysis of ARM-Based Clusters. 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, 2012.

[PAN11] Pang, B (2011). Energy Consumption Analysis of ARM- based System. Aalto University School of Science Degree Programme of Mobile Computing, p. 68.

[PAN16] Pandi, K. M., y Somasundaram, K. (2016). Energy Efficient in Virtual Infrastructure and Green Cloud Computing: A Review. Indian Journal of Science and Technology, 9(11).

[PRA19] Pramanik P, et al. (2019). Power Consumption Analysis, Measurement, Management, and Issues: A State-of-the-Art Review of Smartphone Battery and Energy Usage. IEEE Access, Volume 7, pp. 182113-182172.

[RAS12] Rasmussen Neil. Medición de la eficiencia eléctrica para centros de datos. White Paper 154, Revisión 2. Schneider Electric. Última vez visitado el 8 de enero de 2021. Recurso: https://download.schneider-electric.com/files?p_File_Name=NRAN-72754V_R2_LS.pdf&p_Doc_Ref=SPD_NRAN-72754V_LS

[ROB04] Roberts, I., Emerson, J. (2004). Advanced Video Compression. Última vez visitado el 8 de enero de 2021. Recurso: http://www.animemusicvideos.org/guides/avtech/video4_2.htm

[RUA19] R. Rua, M. Couto y J. Saraiva (2019). GreenSource: A Large-Scale Collection of Android Code, Tests and Energy Metrics. 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 2019, pp. 176-180.

[SAH18] Saha, Biswajit. (2018). Green Computing: Current Research Trends. INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING. 6.

[SHE18] Shehabi, A. et al. (2018). Data center growth in the United States: decoupling the demand for services from electricity use. Environmental Research Letters, 13(12), 124030.

[SON15] Z. Song, X. Zhang y C. Eriksson (2015). Data Center Energy and Cost Saving Evaluation. Energy Procedia 75:1255-1260.

[SVA11] O. Svanfeldt-Winter, S. Lafond y J. Lilius (2011). Cost and energy reduction evaluation for ARM based web servers. IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, DASC, 12-14 Dec. 2011, pp. 480-487.

[THE20] Poletti Therese (2020). How did Intel lose its Silicon Valley crown?. MarketWatch. Última vez visitado el 8 de enero de 2021. Recurso: https://www.marketwatch.com/story/how-did-intel-lose-its-silicon-valley-crown-2020-10-16

[WAG20] B. Wagner (2020). Intergenerational energy efficiency of Dell EMC PowerEdge servers. Dell, DellEMC white paper. 2020. Última vez visitado el 8 de enero de 2021. Recurso: https://www.dell.com/support/kbdoc/es-ar/000148685/intergenerational-energy-efficiency-of-dell-emc-poweredge-servers

[WAN11] Wang, L., y Khan, S. U. (2011). Review of performance metrics for green data centers: a taxonomy study. The Journal of Supercomputing, 63(3), 639–656.
[ZAI17] Zaib Sheikh Jahan, Hassan Raqeeb Ul y Khan Omar Farooq. (2017). Green computing and initiatives for environmental issues. International Journal of Computer Science and Mobile Computing. IJCSMC, Vol. 6, Issue. 7, July 2017, pg.49 – 55.
[ZAM19] Zaman S. et al. (2019). A Systems Overview of Commercial Data Centers: Initial Energy and Cost Analysis. International Journal of Information Technology and Web Engineering, Volume 14 – No. 1, pp 42-65

Computación en la Nube

[AHR10] M. Ahronovitz, D. Amrhein y P. Anderson (2010). Cloud Computing Use Cases - A white paper. 2010. pp. 20–21. Última vez visitado el 8 de enero de 2021. Recurso: http://www.cloud-council.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf
[ARM09] Armbrust Michael, et al. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Tech. rep. UCB/EECS-2009-28. EECS Department, University of California, Berkeley. 2009.
[ARM21] ARM (2021). Hyperscale Data Centers and Cloud Computing. ARM Ecosystem Approach. Última vez accedido 22 de enero de 2021. Recurso: https://www.arm.com/solutions/infrastructure/cloud-computing
[AWS17] AWS (2017). Infrastructure as Code. Amazon Web Services. (2017). Última vez visitado el 8 de enero de 2021. Recurso: https://d0.awsstatic.com/whitepapers/DevOps/infrastructure-as-code.pdf
[AWS20] Amazon Web Services (2020). AWS Command Line Interface - User Guide. Amazon Web Services, Inc. Última vez visitado el 8 de enero de 2021. Recurso: https://docs.aws.amazon.com/cli/latest/userguide/aws-cli.pdf
[AWS20b] Amazon Web Services (2020). Overview of Security Processes. AWS Whitepaper. Última vez visitado el 8 de enero de 2021. Recurso: https://docs.aws.amazon.com/whitepapers/latest/aws-overview-security-processes/aws-overview-security-processes.pdf
[BAR17] Barry Brigg, Eduardo Kassner (2017). “Enterprise Cloud Strategy - 2nd Edition”. 2017. Última vez visitado el 8 de enero de 2021. Recurso: https://azure.microsoft.com/es-es/blog/enterprise-cloud-strategy-2nd-edition/
[BOC14] E. Bocchi, M. Mellia y S. Sarni (2014). Cloud storage service benchmarking: Methodologies and experimentations. 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), Luxembourg, 2014, pp. 395-400.

[CAL15] Calabrese, Barbara y Cannataro, Mario. (2015). Cloud Computing in Healthcare and Biomedicine. Scalable Computing: Practice and Experience. 16.

[CHA11] D. W. Chadwick et al. (2011). My Private Cloud Overview: A Trust, Privacy and Security Infrastructure for the Cloud. 2011 IEEE 4th International Conference on Cloud Computing, Washington, DC, 2011, pp. 752-753.

[CHO14] N. Chopra y S. Singh (2014). Survey on scheduling in hybrid clouds. Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT), Hefei, 2014, pp. 1-6, doi: 10.1109/ICCCNT.2014.6963050.

[CLA20] Don Clark (2020). Amazon and Apple Are Powering a Shift Away From Intel's Chips. The New York Times. Última vez accedido 22 de enero de 2021. Recurso: https://www.nytimes.com/2020/12/01/technology/amazon-apple-chips-intel-arm.html

[CNC20] CNCF (2020) Cloud Native security whitepaper. Cloud Computing Native Foundation. Última vez consultado 08/01/21. Recurso: https://github.com/cncf/sig-security/blob/master/security-whitepaper/CNCF_cloud-native-security-whitepaper-Nov2020.pdf

[DAN20] Danysz, Jerry, Rosal, Victor y Gonzalez-Velez, Horacio. (2020). AWS EC2 Spot Instances For Mission Critical Services. 376-383. 34th International ECMS Conference on Modelling and Simulation

[DEY15] Dey Akon, Chinchwadkar Gajanan et al. (2015). Metadata-as-a-service. In: Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on. IEEE, pp. 6-9.

[DIN19] J. Ding et al (2019). Characterizing Service Level Objectives for Cloud Services: Realities and Myths. 2019 IEEE International Conference on Autonomic Computing (ICAC), Umea, Sweden, 2019, pp. 200-206

[DOR14] B. S. Đorđević, S. P. Jovanović y V. V. Timčenko (2020). Cloud Computing in Amazon and Microsoft Azure platforms: Performance and service comparison. 2014 22nd Telecommunications Forum Telfor (TELFOR), Belgrade, 2014, pp. 931-934

[EKW18] N. Ekwe-Ekwe y A. Barker (2018). Location, Location, Location: Exploring Amazon EC2 Spot Instance Pricing Across Geographical Regions. 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Washington, DC, 2018, pp. 370-373.

[EME16] J. Emeras, S. Varrette y P. Bouvry (2016). Amazon Elastic Compute Cloud (EC2) vs. In-House HPC Platform: A Cost Analysis. 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2016, pp. 284-293

[EZE16] Ezell, S.J. y Atkinson, R.D (2016). The Vital Importance of High-Performance Computing to US Competitiveness. Information Technology and Innovation Foundation.

Reporte. 2016. Última vez accedido 22 de enero de 2021. Recurso: http://www2.itif.org/2016-high-performance-computing.pdf

[FAT15] F. Fatemi Moghaddam et al (2015). Cloud computing challenges and opportunities: A survey. 2015 1st International Conference on Telematics and Future Generation Networks (TAFGEN), Kuala Lumpur, 2015, pp. 34-38.

[FEN12] Fengji Luo et al. (2012). Hybrid cloud computing platform: The next generation IT backbone for smart grid. 2012 IEEE Power and Energy Society General Meeting.

[GOR13] Gorelik, Eugene (2013). Cloud computing models. Master Tesis. Massachusetts Institute of Technology. 2013. Última vez accedido 22 de enero de 2021. Recurso: https://dspace.mit.edu/handle/1721.1/79811

[HUM16] M. Humphrey, R. Emerson y N. Beekwilder (2016). Unified, Multi-level Intrusion Detection in Private Cloud Infrastructures. 2016 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, 2016, pp. 11-15.

[KAN17] Kania-Richmond, A. et al. (2017). A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. J. Bodyw. Mov. Ther. 21, 274–283.

[KOT18] C. Kotas, T. Naughton y N. Imam (2018). A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing. 2018 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, 2018, pp. 1-4

[KUM14] N. Kumar et al (2013). Reduction of cost by implementing transparency in cloud computing through different approaches. 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, Ramanathapuram, 2014, pp. 1723-1725.

[MEL11] Mell, Peter y Grance, Tim (2011). The NIST de notion of cloud computing. 2011. Última vez accedido 22 de enero de 2021. Recurso: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

[MER19] Mercl Lubos y Pavlik Jakub (2019) Public Cloud Kubernetes Storage Performance Analysis. Computational Collective Intelligence (pp.649-660).

[MOI19] Moinudheen Mohammed (2019). Securing Azure Storage Account with Shared Access Signature. MSSQLtips. Última vez accedido 22 de enero de 2021. Recurso: https://www.mssqltips.com/sqlservertip/6189/securing-azure-storage-account-with-shared-access-signature/

[MUH20] A. S. Muhammed y D. Ucuz (2020). Comparison of the IoT Platform Vendors, Microsoft Azure, Amazon Web Services, and Google Cloud, from Users' Perspectives. 2020 8th International Symposium on Digital Forensics and Security (ISDFS), Beirut, Lebanon, 2020, pp. 1-4

[MYE20] Myers Tamra et al (2020). Grant limited access to Azure Storage resources using shared access signatures (SAS). Última vez accedido 22 de enero de 2021. Recurso: https://docs.microsoft.com/en-us/azure/storage/common/storage-sas-overview

[PAH17] Pahl Claus, et al. (2017). Cloud Container Technologies: a State-of-the-Art Review. in IEEE Transactions on Cloud Computing PP(99):1-1· Mayo de 2017. Última vez accedido 22 de enero de 2021. Recurso: https://bia.unibz.it/bitstream/handle/10863/5284/ContainerOrch.pdf?sequence=2

[PAN19] Y. Pan et al (2019). A Performance Comparison of Cloud-Based Container Orchestration Tools. 2019 IEEE International Conference on Big Knowledge (ICBK), Beijing, China, 2019, pp. 191-198.

[PAT12] S. Patidar, D. Rane y P. Jain (2012). A Survey Paper on Cloud Computing. 2012 Second International Conference on Advanced Computing & Communication Technologies, Rohtak, Haryana, 2012, pp. 394-398.

[PAT20] J. C. Patni, S. Banerjee y D. Tiwari (2020). Infrastructure as a Code (IaC) to Software Defined Infrastructure using Azure Resource Manager (ARM). 2020 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 2020.

[PEA19] Pearlman Joe (2019). What Is Infrastructure-as-Code (IaC), and Why It Is the Future of Cloud Automation. Cloud Native CI/CD and IaC DevOps, Dell Technologies Consulting. 2019. Última vez accedido 22 de enero de 2021. Recurso: https://infocus.dellemc.com/joe_pearlman/what-is-infrastructure-as-code-iac-and-why-it-is-the-future-of-cloud-automation/

[PET19b] Petitpierre Arthur (2019). Amazon EC2 Optimize EKS cost with Spot & A1 Instances. Amazon AWS Services. Última vez accedido 22 de enero de 2021. Recurso: https://d1.awsstatic.com/product-marketing/EKS/13.%20Arthur-Petitpierre%20-%20EKS%20cost%20optimization%20with%20EC2%20Spot%20and%20A1%20instances.pdf

[PEV20] Peven Ben (2020). Building for Cost optimization and Resilience for EKS with Spot Instances. AWS Compute Blog. Última vez accedido 22 de enero de 2021. Recurso: https://aws.amazon.com/es/blogs/compute/cost-optimization-and-resilience-eks-with-spot-instances/

[PHA18] T. Pham, S. Ristov y T. Fahringer (2018). Performance and Behavior Characterization of Amazon EC2 Spot Instances. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2018, pp. 73-81.

[POC20] Danilo Poccia (2020). Coming Soon – EC2 C6gn Instances – 100 Gbps Networking with AWS Graviton2 Processors. AWS News Blog. Última vez accedido 22 de enero de 2021. Recurso: https://aws.amazon.com/es/blogs/aws/coming-soon-ec2-c6gn-instances-100-gbps-networking-with-aws-graviton2-processors/

[PRA18] A. G. Prajapati, S. J. Sharma y V. S. Badgujar (2018). All About Cloud: A Systematic Survey. 2018 International Conference on Smart City and Emerging Technology (ICSCET), Mumbai, 2018, pp. 1-6.

[PRA18b] A. Prabhakaran y J. Lakshmi (2018). Cost-Benefit Analysis of Public Clouds for Offloading In-House HPC Jobs. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2018, pp. 57-64.

[RIG19] RightScale (2019). RightScale 2019 State of the Cloud Report from Flexera Identifies Cloud Adoption Trends. 2019. Última vez accedido 22 de enero de 2021. Recurso: https://media.flexera.com/documents/rightscale-2019-state-of-the-cloud-report-from-flexera.pdf?elqTrackId=590185f8b8bb489d8c712f699233d3ec&elqaid=4588&elqat=2

[RIM09] Rimal Bhaskar Prasad, Choi Eunmi, y Lumb Ian (2009). A taxonomy and survey of cloud computing systems. In: INC, IMS and IDC, NCM'09. Fifth International Joint Conference on. IEEE pp. 44-51.

[RUI14] P. Ruiu et al. (2014). HPC Cloud Pills: On-Demand Deployment and Execution of HPC Application in Cloud Environments. 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Guangdong, pp. 82-88.

[SAA19] I. Saeed, S. Baras y H. Hajjdiab (2019) Security and Privacy of AWS S3 and Azure Blob Storage Services. 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), Singapore, 2019, pp. 388-394.

[SAM19] A. M. Sampaio y J. G. Barbosa (2019). Enhancing Reliability of Compute Environments on Amazon EC2 Spot Instances. 2019 International Conference on High Performance Computing & Simulation (HPCS), Dublin, Ireland, 2019, pp. 708-715.

[SHI20] Shimon Ifrah (2020). Getting Started with Containers in Azure - Deploy, Manage, and Secure Containerized Applications. Apress, Melbourne, VIC, Australia

[SID19] Tanveer Ahmed Siddiqui (2019). An Introduction to Windows Azure Blob Storage. GlobalLogic. Última vez consultado 08/01/21. Recurso: https://www.globallogic.com/il/wp-content/uploads/2019/12/Introduction-to-Windows-Azure-Blob-Storage.pdf

[VEEN16] K. Veena, C. Anand y G. C. Prakash (2016). Temporal and Spatial Trend Analysis of Cloud Spot Instance Pricing in Amazon EC2. 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), Auckland, 2016, pp. 909-912.

[ZOU18] Zouheir D. y Hassan H. (2018). Cloud Storage Comparative Analysis Amazon Simple Storage vs. Microsoft Azure Blob Storage. International Journal of Machine Learning and Computing, Volume 8, pages 85-89.

Computación Móvil y procesamiento basado en ARM

[ARS12] Arslan, Mustafa et al (2012). Computing while charging: Building a distributed computing infrastructure using smartphones. CoNEXT 2012 - Proceedings of the 2012 ACM Conference on Emerging Networking Experiments and Technologies. 193-204.

[BAN17] Banchelli Gracia et al. (2017). Is Arm software ecosystem ready for HPC?. SC17: International Conference for High Performance Computing, Networking, Storage and Analysis. 2017.

[BAN21] BankMyCell (2021). HOW MANY SMARTPHONES ARE IN THE WORLD? Última vez visitado el 8 de enero de 2021. Recurso: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world

[BHA19] Bhat G., et al (2019). Power and Thermal Analysis of Commercial Mobile Platforms: Experiments and Case Studies. 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, pp 144-149

[BIC13] Bicen, Huseyin y Kocakoyun, Şenay. (2013). The Evaluation of The Most Used Mobile Devices Applications by Students. Procedia - Social and Behavioral Sciences. 89. 756–760.

[BLE13] Blem, E., Menon, J. y Sankar K (2013). A Detailed Analysis of Contemporary ARM and x86 Architectures. High Performance Computing Architecture (HPCA), 2013 IEEE 19th Int. Symp. (2013).

[BUR97] Burgess Mark, Ralston Ricky (1997). Distributed resource administration using cfengine. In: Software: practice and experience 27.9 pp. 1083–1101. 1997.

[CER12] Esther Cerdeño (2012) Phone evolution and revolution. IT Deputy Manager - Mapfre. Última vez visitado el 8 de enero de 2021. Recurso: https://app.mapfre.com/mapfrere/docs/html/revistas/trebol/n65/pdf/Articulo2-en.pdf

[CHR21] ChRoot (2012). Chroot - Linux Programmer's Manual. Chroot. 2021. Última vez visitado el 8 de enero de 2021. Recurso: http://man7.org/linux/man-pages/man2/chroot.2.html

[CRI19] Cristea V., et al. (2019). Mobile Phones and Energy Consumption. En Green IT Engineering: Social, Business and Industrial Applications (1.a ed., pp 243-271) Springer Nature, Cham, Switzerland.

[CUR18] Curiel, M. et al (2018). Parallel Processing of Images in Mobile Devices using BOINC. Open Engineering, 8(1), 87–101.

[GED20] Gedawy H., et al (2020). RAMOS: A Resource-Aware Multi-Objective System for Edge Computing. IEEE Transactions on Mobile Computing.

[GIN20] Ginny, et al, 2020. Smartphone processor architecture, operations, and functions: current state-of-the-art and future outlook: energy performance trade-off. The Journal of Supercomputing (J SUPERCOMPUT).

[GON17] Santos-González et al. (2017). Implementation and Analysis of Real-Time Streaming Protocols. Sensors (Basel, Switzerland). pp 17-24.
[HIR18] Hirsch M., et al, 2018. Augmenting computing capabilities at the edge by jointly exploiting mobile devices: A survey. Future Generation Computer Systems (FUTURE GENER COMP SY), pp 644-662.
[HYE12] Oh, Hyeong-Seok et al. (2012). Evaluation of Android Dalvik virtual machine. ACM International Conference Proceeding Series.
[KAM00] Kamp Poul-Henning and Watson Robert (2000). Jails: Confining the omnipotent root. In: Proceedings of the 2nd International SANE Conference. Vol. 43. 2000, p. 116.
[KAR17] S. Karthick y S. Binu (2017). Android security issues and solutions. 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore, 2017, pp. 686-689.
[KHO18] I. Khokhlov y L. Reznik (2018). Android system security evaluation. 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, 2018, pp. 1-2
[KMI18] Kmiec, Sebastian, et al. (2018). A Comparison of ARM Against x86 for Distributed Machine Learning Workloads. Performance Evaluation and Benchmarking for the Analytics Era (pp.164-184)
[KOO11] Koomey, Jonathan (2011). Growth In Data Center Electricity Use 2005 To 2010. A Report By Analytical Press, Completed At The Request Of The New York Times 9 (2011): 161.
[KRO08] Kroski, Ellyssa (2008). On the Move with the Mobile Web: Libraries and Mobile Technologies. Library Technology Reports, 2008, vol. 44, n. 5, pp. 1-48.
[LIH16] H. Li et al (2016). Mobile Edge Computing: Progress and Challenges. 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Oxford, 2016, pp. 83-84.
[LIU19] F. Liu et al (2019). A Survey on Edge Computing Systems and Tools. in Proceedings of the IEEE, vol. 107, no. 8, pp. 1537-1562, Aug. 2019.
[MAC17] Mach P. y Becvar Z. (2017). Mobile Edge Computing: A Survey on Architecture and Computation Offloading. IEEE Communications Surveys & Tutorials, Volume: 19 – No 3, pp 1628-1656.
[MAN20] Tobias Mann (2020). Arm Wrestles Intel, AMD for New Ground. SdxCentral. Última vez visitado el 8 de enero de 2021. Recurso: https://www.sdxcentral.com/articles/news/arm-wrestles-intel-amd-for-new-ground/2020/06/
[NET15] Neto, Edmilson et al (2015). Unveiling the Architecture and Design of Android Applications - An Exploratory Study. ICEIS 2015 - 17th International Conference on Enterprise Information Systems, Proceedings. 2. 201-211.

[NGA19] Abigail Ng (2019). Smartphone users are waiting longer before upgrading — here's why. Última vez visitado el 8 de enero de 2021. Recurso: https://www.cnbc.com/2019/05/17/smartphone-users-are-waiting-longer-before-upgrading-heres-why.html

[ODD18] Odette Penners et al. (2018). Life cycle extension of mobile phones: an exploration with focus on the end-consumer. The Central European Review of Economics and Management. Volume 2, Pages 7-37.

[ODE20] S. O'Dea (2020). Average lifespan (replacement cycle length) of smartphones in the United States from 2014 to 2024. Statista. Última vez visitado el 8 de enero de 2021. Recurso: https://www.statista.com/statistics/619788/average-smartphone-life/

[PRI14] Daniel Prince y Oliver Fitton (2014). The Future of Mobile Devices Security and Mobility. Investigation research. Security Lancaster University. Última vez visitado el 8 de enero de 2021. Recurso: https://core.ac.uk/download/pdf/42414468.pdf

[ROJ06] Olivares Rojas et al. (2006). Evaluación de Dispositivos Smartphone para su Uso como Servidores Móviles. AGECOMPAt: Cuernavaca Morelos. 968-878.

[SAR13] Sarwar, Muhammad y Soomro, Tariq. (2013). Impact of Smartphone's on Society. European Journal of Scientific Research. 98.

[SAR19] A. Sarkar et al (2019). Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems. 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2019, pp. 73-79.

[SCH18] Schaffner B., et al. (2018). Smartphones as Alternative Cloud Computing Engines: Benefits and Trade-offs. 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), pp 244-250.

[TIW10] Tiwana, B. et al. (2010). Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis 105

[TON13] A. R. Tonini et al. (2013). Analysis and Evaluation of the Android Best Practices Impact on the Efficiency of Mobile Applications. 2013 III Brazilian Symposium on Computing Systems Engineering, Niteroi, 2013, pp. 157-158.

[WHA11] Chao Wang et al. (2011). The research of Android System architecture and application programming. Proceedings of 2011 International Conference on Computer Science and Network Technology, Harbin, 2011, pp. 785-790.

[YAD15] R. Yadav y R. S. Bhadoria. Performance Analysis for Android Runtime Environment. 2015 Fifth International Conference on Communication Systems and Network Technologies, Gwalior, 2015, pp. 1076-1079.

[ZHA16] Z. Zhang y S. Li (2016) A Survey of Computational Offloading in Mobile Cloud Computing. 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Oxford, 2016, pp. 81-82.

Video digital, códecs, streaming y transcodificación de video

[ADH15] V. K. Adhikari et al. (2015). Measurement Study of Netflix, Hulu, and a Tale of Three CDNs. in IEEE/ACM Transactions on Networking, vol. 23, no. 6, pp. 1984-1997, Dec. 2015

[AHM74] Ahmed, N., Natarajan, T. R., & Rao, K. R. (1974). On image processing and a discrete cosine transform. IEEE Transactions on Computers, vol. 23, 90-93.

[APP20] Apple Company (2020). Apple ProRes. Apple Company. White Paper. Última vez visitado el 8 de enero de 2021. Recurso: https://www.apple.com/final-cut-pro/docs/Apple_ProRes_White_Paper.pdf

[ARO17] S. D'Aronco, L. Toni y P. Frossard (2017). Price-Based Controller for Utility-Aware HTTP Adaptive Streaming. in IEEE MultiMedia, vol. 24, no. 2, pp. 20-29, Apr.-June 2017, doi: 10.1109/MMUL.2017.41.

[AZN14] Aznar P. (2014) Solving Advanced Encoding Problems with FFMPEG. Code4Lib Journal, ISSUE 25. Última vez visitado el 8 de enero de 2021. Recurso: http://journal.code4lib.org/articles/9856

[BAN12] Bankoski J., Wilkins P. y Xu Y. (2012) Technical Overview of VP8, an Open Source video Codec for the Web. Google Inc. 1600 Amphitheatre Parkway, Mountain view, CA, USA.

[BAR16] Olivier Barais et al (2016). Towards microservices architecture to transcode videos in the large at low costs. TEMU 2016 - International Conference on Telecommunications and Multimedia, Jul 2016, Heraklion, Greece. pp.1 - 6.

[BAS15] Bass Len, Weber Ingo y Zhu Liming (2015). DevOps: A Software Architect's Perspective. Addison-Wesley Professional. SEI Series in Software Engineering, ISBN: 0134049845,9780134049847. 2015.

[BEG11] A. Begen, T. Akgul y M. Baugher (2011). Watching Video over the Web: Part 1: Streaming Protocols. in IEEE Internet Computing, vol. 15, no. 2, pp. 54-63, March-April 2011.

[CAS13] Castro Gil A. y Santos C. (2013). Tecnología Digital – Video Digital. Universidad Nacional de Educación a Distancia . Madrid. España

[CHE93] Chen T (1993) Video compression: Standards and applications. Visual Communication and Image Representation, nº2, pp. 103-111, June 1993.

[CHE06] Chen, Jian-Wen et al. (2006). Introduction to H.264 advanced video coding. 2006. 6 pp. Design Automation, 2006. Asia and South Pacific Conference on

[DEV15] A. Devlic et al (2015). Towards QoE-aware adaptive video streaming. 2015 IEEE 23rd International Symposium on Quality of Service (IWQoS), Portland, OR, 2015, pp. 75-76.

[DOR13] Doria T. (2013). The Great Codec Debate The Myths, Realities, and Considerations You Need to Know Before Making Your Next Videoconferencing Purchase" Cisco SMO – Collaboration Technology Group.

[EBR02] Mohammed Ebrahim Al-Mualla, C. Nishan Canagarajah y David R. Bull (2002). Chapter 2 - Video Coding: Fundamentals. In Signal Processing and its Applications, Video Coding for Mobile Communications, Academic Press, 2002, Pages 9-42, ISBN 9780120530793.

[EDP18] Edpalm, Viktor et al (2018). H.264 Video Frame Size Estimation. (Technical Document).

[ELA19] V. S. Elagin et al. (2019). Models of QOE ensuring for OTT services. 2019 Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russia, 2019, pp. 1-4

[ETS17] ETSI (2017). Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream. ETSI. Última vez consultado 08 de enero de 2021. Recurso: http://etsi.org/deliver/etsi_ts/101100_101199/101154/02.03.01_60/ts_101154v020301p.pdf

[FLE19] RightScale/Flexera (2019). State of the Cloud - As Cloud Use Grows, Organizations Focus on Cloud Costs and Governance. RightScale/Flexera. Última vez consultado 08 de enero de 2021. Recurso:http://googliers.net/static/media/uploads/download_files/2019_state_of_the_cloud_report.pdf

[FLY16] D. Flynn et al. (2016). Overview of the Range Extensions for the HEVC Standard: Tools, Profiles, and Performance. in IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, no. 1, pp. 4-19

[GAL92] Gall D. (1992). The MPEG video compression algorithm. Signal Process: Image Commun., vol. 4, nº2m pp. 129-140, 1992.

[GAR10] Adriana Garcia, Hari Kalva y Borko Furht (2010). A study of transcoding on cloud environments for video content delivery. MCMC '10 Proceedings of the 2010 ACM multimedia workshop on.

[GER93] Gersho A., Wu S. (1993). Enhanced video compression with standarized bitstream syntax. in Proc. IEEE Int Conf. Acoust. Syst, Signal Process, vol 1, 1993, pp 103-106.

[GOT19] K. Goto, Y. Hayamizu, M. Bandai y M. Yamamoto (2019). QoE Performance of Adaptive Video Streaming in Information Centric Networks. 2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Paris, France, 2019, pp. 1-2.

[HAN19] S. Han, Y. Go, H. Noh y H. Song (2019). Cooperative Server-Client HTTP Adaptive Streaming System for Live Video Streaming. 2019 International Conference on Information Networking (ICOIN), Kuala Lumpur, Malaysia, pp. 176-180

[HOA16] Xuan Tung Hoang y Tien Thanh Nguyen (2016). Reducing Startup Time in MP4 On-demand Video Streaming Services with Movie Atom Caching. VNU Journal of Science: Comp. Science & Com. Eng., Vol. 32, No. 1 (2016) 33–41

[HUS10] Huszák, Árpád y Imre, Sandor. (2010). Analysing GOP structure and packet loss effects on error propagation in MPEG-4 video streams. Communications, Control and Signal Processing (ISCCSP), 2010 4th International Symposium on

[IEE18] IEEE Inc (2018). IEEE Approved Draft Standard for Adaptive Streaming. in IEEE P1857.7/D3, July 2018 , vol., no., pp.1-70, 4 Dec. 2018.

[IGA04] Igarta, M. (2004). A Study of MPEG-2 and H.264 Video Coding. M.S.E.C.E., Purdue University, 14-50.

[JAI81] Jain A. K. (1981). Image data compression: A review. Proc. IEEE, vol 69, pp, 349-384, 1981

[JOS13] Joskowicz J. (2013). Codificación de Voz y Video. Instituto de Ingeniería Eléctrica, Facultad de Ingeniería. Universidad de la República de Montevideo, Uruguay.

[KAW15] S. J. Kaware y S. K. Jagtap (2015). A survey: Heterogeneous video transcoder for H.264/AVC to HEVC. 2015 International Conference on Pervasive Computing (ICPC), Pune, 2015, pp. 1-3.

[KUF15] J. Kufa y T. Kratochvil (2015). Comparison of H.265 and VP9 coding efficiency for full HDTV and ultra HDTV applications. 2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, 2015, pp. 168-171

[KUF16] J. Kufa, L. Polak y T. Kratochvil (2016). HEVC/H.265 vs. VP9 for Full HD and UHD video: Is there any difference in QoE?. 2016 International Symposium ELMAR, Zadar, 2016, pp. 51-55

[LAN14] Lacinak Chris (2014). A Primer on Codecs for Moving Image and Sound Archives & 10 Recommendations for Codec Selection and Management. Audiovisual Preservation Solutions. Media Archiving & Data Management Consultants.

[LAT17] P. Latkoski, M. Porjazoski y B. Popovski (2017). QoS control in OTT video distribution system. IEEE EUROCON 2017 - 17th International Conference on Smart Technologies, Ohrid, 2017, pp. 99-103, doi: 10.1109/EUROCON.2017.8011085.

[LED15] Stefan Lederer (2015). Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length. Bitmovin. Última vez consultado 08 de enero de 2021. Recurso: https://bitmovin.com/mpeg-dash-hls-segment-length/

[LEE15] Lee, B. D (2015). Empirical analysis of video partitioning methods for distributed HEVC encoding. Int. J. Multimed. Ubiquitous Eng. 10, 81–90.

[LEI13] X. Lei, X. Jiang y C. Wang (2013). Design and Implementation of a Real-Time Video Stream Analysis System Based on FFMPEG. 2013 Fourth World Congress on Software Engineering, Hong Kong, 2013, pp. 212-216.

[LEU09] J. Leu, C. Tsai y C. Yi (2009). Improving Adaptive Streaming Service across Wired/Wireless Networks. 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, Taipei, 2009, pp. 614-618.

[LIB12] Li B., Sullivan J., Xu J (2012). Compression performance of high efficiency video coding (HEVC) Working draft 4. In Proc IEEE in Conf. Circuits Syst, May 2012, pp. 886-889.

[LID14] Li, D. et al. (2014). An empirical study of the energy consumption of android applications. Proc. - 30th Int. Conf. Softw. Maint. Evol. ICSME 2014 121–130.

[LIM94] Lim G., Pang K., y Tan T (1994). A low complex software video decoder. Signal Process: Image Commun., Special Issue on Software Decoding, June 1994.

[LIN19] Linux Encoding (2010). x264 FFmpeg Options Guide. Última vez consultado 08 de enero de 2021. Recurso: https://sites.google.com/site/linuxencod-ing/x264-ffmpeg-mapping.

[LIU18] Liu, P. et al (2018). VideoCoreCluster: Energy-Efficient, Low-Cost, and Hardware-Assisted Video Transcoding System. Wireless Communications and Mobile Computing, 2018, 1–13.

[LIX18] Li, X. et al (2018). Performance Analysis and Modeling of Video Transcoding Using Heterogeneous Cloud Services. IEEE Transactions on Parallel and Distributed Systems, 1–1.

[LIX18b] Li, X. et al (2018). Cost-Efficient and Robust On-Demand Video Transcoding Using Heterogeneous Cloud Services. IEEE Transactions on Parallel and Distributed Systems, 29(3), 556–571.

[MAI09] J. Maisonneuve et al. (2009). An Overview of IPTV Standards Development. in IEEE Transactions on Broadcasting, vol. 55, no. 2, pp. 315-328, June 2009

[MAK18] V. D. Maksimovic et al. (2018). The Impact of Successive B Frames on Video Using H.264 and H.265 Compression Techniques. 2018 26th Telecommunications Forum (TELFOR), Belgrade, 2018, pp. 1-4.

[MAL19] M. Malhotra, A. V. Singh y R. Matam (2019). Comparative Performance Issues with H.264 vs H.265. 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 2019, pp. 283-288

[MAR13] J. Martin et al. (2013). Characterizing Netflix bandwidth consumption. 2013 IEEE 10th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, 2013, pp. 230-235.

[MAX10] Levkov Maxim (2010). Understanding the MPEG-4 Movie ATOM. Adobe - Streaming. Última vez consultado 08 de enero de 2021. Recurso: https://www.adobe.com/devnet/video/articles/mp4_movie_atom.html

[MUK13] D. Mukherjee et al. (2013). The latest open-source video codec VP9 - An overview and preliminary results. 2013 Picture Coding Symposium (PCS), San Jose, CA, 2013, pp. 390-393.
[MUK15] Mukherjee, Debargha et al. (2015) A Technical Overview of VP9--the Latest Open-Source Video Codec. SMPTE Motion Imaging Journal. 124. 44-54
[NAI18] Naik, Manja et al (2018). Inter Frame Coding in Advanced Video Coding Standard H.264 using Block Based Motion Compensation Technique. Second International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT-2017) At: GSSSIETW, Mysuru.
[OHM12] Ohm J. et al. (2012). Comparison of the Coding Efficiency of Video Coding Standards – Including High Efficiency Video Coding (HEVC). IEEE Transactions on circuits and systems for video technology, vol 22, Nº 22, December 2012.
[OZC05] T. Ozcelebi, M. R. Civanlar y A. M. Tekalp (2005). Minimum delay content adaptive video streaming over variable bitrate channels with a novel stream switching solution. IEEE International Conference on Image Processing 2005, Genova, 2005, pp. I-209.
[PAT15] Patnaik Yogananda y Patra Dipti. (2015). H.264/AVC/MPEG Video Coding With an Emphasis to Bidirectional Prediction frames. 2015 Annual IEEE India Conference (INDICON).
[PEN16] Peng Liu et al. (2016). Greening the Video Transcoding Service with Low-Cost Hardware Transcoders. Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC '16). June 22–24, 2016. Denver, CO, USA.
[PER14] R. Pereira y E. G. Pereira (2014). Dynamic Adaptive Streaming over HTTP and Progressive Download: Comparative Considerations. 2014 28th International Conference on Advanced Information Networking and Applications Workshops, Victoria, BC, 2014, pp. 905-909.
[POY96] Poynton C. (1996). A Technical Introduction to Digital Video. Chapter 1, Copyright John Wiley & Sons. Última vez consultado 08 de enero de 2021. Recurso: https://inst.eecs.berkeley.edu/~cs150/Documents/BasicVideoPrinciples.pdf
[PUR93] Puri A., Wong A (1993). Spatial domain resolution scalable video coding. Proc. SPIE Visual Commun and Image Processing, Boston, MA, Nov 1993.
[RAA17] Raake A., et al. (2017). A Bitstream-based, Scalable Video-Quality Model for HTTP Adaptive Streaming: ITU-T P.1203.1. Ninth International Conference on Quality of Multimedia Experience (QoMEX), Erfurt, pp. 1-6.
[RAK20] Rakesh R., et al. (2020). Bitstream-based Model Standard for 4K/UHD: ITU-T P.1204.3 - Model Details, Evaluation, Analysis and Open Source Implementation. Twelfth International Conference on Quality of Multimedia Experience (QoMEX), Athlone, Ireland, pp 1-6.

[RIC11] Iain Richardson (2011). H.264 / AVC Inter Prediction. White Paper - VCodex. Última vez visitado el 8 de enero de 2021. Recurso: http://www.staroceans.org/e-book/vcodex/H264_interpred_wp.pdf

[ROD10] Rodriguez F. et al (2010). Análisis Comparativo de codificadores de Audio sin pérdidas. Universidad Nacional de Rosario, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. 2º Congreso Internacional de Acústica UNTREF. Argentina.

[SAM19b] Sameti, S. (2019). Distributed Video Transcoding for Live and Interactive Multimedia Applications. Master's thesis. University of Calgary, Calgary, AB.

[SAN17] Sani Y., et al. (2017). Adaptive Bitrate Selection: A Survey. IEEE Communications Surveys & Tutorials, Volume 19 - No. 4, pp 2985-3014.

[SCH80] Schwartz M (1980). Information Transmission, Modulation and Noise. 3rd Ed. McGraw Hill Book Co. 1980.

[SEU15] M. Seufert et al (2015). A Survey on Quality of Experience of HTTP Adaptive Streaming. in IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 469-492, Firstquarter 2015.

[SHC95] Schafer R., Sikora T. (1995). Digital Video Coding Standards and Their Role in Video Communication. Proceedings of the IEEE, Vol. 83, Nº 6, June 1995

[SHI19] Shi Y. y Sun H., 2019. Image and video compression for multimedia engineering: fundamentals, algorithms, and standards Third Edition (Image Processing Series). CRC Press, Florida, USA.

[SIM14] Sims G. (2014). How to Encode H.265 Video Using ffmpeg on Linux. Maketecheasier. Última vez visitado el 8 de enero de 2021. Recurso: https://www.maketecheasier.com/encode-h265-video-using-ffmpeg

[SON12] Sonnati F. (2012). Video Encoding & Technologies – FFMPEG – the swiss army knife of Internet Streaming. Sonnati Blog. Última vez visitado el 8 de enero de 2021. Recurso: https://sonnati.wordpress.com/2012/10/19/ffmpeg-the-swiss-army-knife-of-internet-streaming-part-vi/

[SUL05] G. J. Sullivan y T. Wiegand (2005). Video Compression - From Concepts to the H.264/AVC Standard. in Proceedings of the IEEE, vol. 93, no. 1, pp. 18-31, Jan. 2005

[TUM15] P. Tumas y A. Serackis (2015). Peer-to-peer adaptive video streaming system. 2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), Riga, 2015, pp. 1-4

[UHL18] T. Uhl et al (2018). Comparison Study of H.264/AVC, H.265/HEVC and VP9-Coded Video Streams for the Service IPTV. 2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, 2018, pp. 1-6

[UHR14] Uhrina, Miroslav et al. (2014). Impact of H.264/AVC and H.265/HEVC compression standards on the video quality for 4K resolution. Advances in Electrical and Electronic Engineering 12(4) - AEEE.V12I4.1216.

[UHR16] M. Uhrina, J. Bienik y M. Vaculik (2016). Coding efficiency of VP8 and VP9 compression standards for high resolutions. 2016 ELEKTRO, Strbske Pleso, 2016, pp. 100-103.

[VET03] Vetro, Anthony et al (2003). Video transcoding architectures and techniques: An overview. Signal Processing Magazine, IEEE. 20. 18 - 29.

[VLC20] VLC (2020). Chapter 3. Advanced Streaming using the command line. VideoLan Networks. Última vez visitado el 8 de enero de 2021. Recurso: https://wiki.videolan.org/Documentation:Streaming_HowTo/Advanced_Streaming_Using_the_Command_Line/

[VOS13] Vos K. et al (2013). The Opus Codec. Opus Company Publication at the 135th AES Convention. 2013 October 17-20. New York, USA

[WAN04] Wang B. et al (2004). Multimedia streaming via TCP: an analytic performance study. in Proceedings of the 12th annual ACM international conference on Multimedia, MULTIMEDIA '04, New York, NY, USA, 2004, ACM, pp. 908–915.

[WAT13] Watson, Andrew. (2013). High Frame Rates and Human Vision: A View Through the Window of Visibility. SMPTE Motion Imaging Journal. 122. 18-32.

[WIK21] Wikipedia (2021). List of Codecs. Wikipedia Website. Última vez visitado el 8 de enero de 2021. Recurso: https://en.m.wikipedia.org/wiki/List_of_codecs

[WUD01] Dapeng Wu et al. (2001). Streaming video over the Internet: approaches and directions. in IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, no. 3, pp. 282-300, March 2001.

[YAR05] A. Yarali y A. Cherry (2005). Internet Protocol Television (IPTV). TENCON 2005 - 2005 IEEE Region 10 Conference, Melbourne, Qld., 2005, pp. 1-6.

[YOO19] Yoon, J., y Banerjee, S. (2019). Hardware-assisted, Low-cost Video Transcoding Solution in Wireless Networks. IEEE Transactions on Mobile Computing, 1–1.

[ZAB18] A. Zabrovskiy, C. Feldmann y C. Timmerer (2018). A Practical Evaluation of Video Codecs for Large-Scale HTTP Adaptive Streaming Services. 2018 25th IEEE International Conference on Image Processing (ICIP), Athens, 2018, pp. 998-1002

[ZAM09] Alex Zambelli (2009). IIS smooth streaming technical overview. Microsoft Corporation, 3, 2009. Última vez visitado el 8 de enero de 2021. Recurso: https://www.bogotobogo.com/VideoStreaming/Files/iis8/IIS_Smooth_Streaming_Technical_Overview.pdf

[ZAT10] Zatt, Bruno et al. (2010). GOP structure adaptive to the video content for efficient H.264/AVC encoding. Proceedings - International Conference on Image Processing, ICIP. 3053-3056.
[ZEN16] H. Zeng, Z. Zhang y L. Shi (2016). Research and Implementation of Video Codec Based on FFmpeg. 2016 International Conference on Network and Information Systems for Computers (ICNISC), Wuhan, 2016, pp. 184-188