UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Sistema Adaptativo para la Gestión de Rankings Masivos en Tiempo Real

David Gómez del Pulgar Caba

Mayo, 2016

Sistema Adaptativo para la Gestión de Rankings Masivos en Tiempo Real

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de Información

TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

Sistema Adaptativo para la Gestión de Rankings Masivos en Tiempo Real

Autor: David Gómez del Pulgar Caba

Director: David Vallejo Fernández

Mayo, 2016

David Gómez del Pulgar Caba

Ciudad Real – Spain

E-mail: David.GomezlPulgar@alu.uclm.es
Teléfono: 650 288 294

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

En este Trabajo de Fin de Grado se va a centrar en desarrollar un sistema inteligente que optimice la velocidad de actualización de los ranking, sin importar el número de peticiones para su realización, y el uso de los recursos, permitiendo la gestión de rankings en juegos masivos multijugador.

Esta desarrollado con Python y las herramientas que nos ofrece Google App Engine, plataforma que pertente a Google y que nos permite desarrollar aplicaciones web escalables. La aplicación implementada va a estar alojada en los servidores de Google, permitiendo el despliegue de esta.

El sistema creado pueden instanciarlo e integrarlo otros desarrolladores en sus proyectos, por ese motivo se ha facilitado su uso.

Abstract

English version of the previous page.

Agradecimientos

Ahora toca el momento de agradecer a todas las personas que han hecho posible que llegue donde estoy hoy, y han hecho que mis años como estudiante universitario no sean agradables. Se lo quiero agradecer:

A esos padres que han estado ahí en las dificultades

A mi hermano Oscar, por esos momentos en los que compartíamos ideas sobre la informática cada viernes que nos veíamos, por esas veces que ha estado cuando me encontraba solo en Ciudad Real y por esos momentos en los que cada uno teníamos que hacer nuestras tareas y acabábamos cantando.

A Alex, Juan Miguel y David, buenos compañeros de piso y clase, por esos grandes momentos que amenizaban estos cuatro años de carrera como los grandes vicios con golpes de mesa incluido, las charlas de los recorridos al gimnasio, las experiencias en los recorridos en coche, el flameado de Alex y los días "trampa".

A Miguel Ángel y Rubén, por hacer entretenidas las horas en las que teníamos que hacer trabajos.

A Antonio, por ese miedo que me metía cuando iba a empezar una nueva asignatura y por la ayuda que me ha ofrecido a lo largo de la carrera.

A David Vallejo, por ayudarme a realizar este proyecto y por hacer más fáciles asignaturas, que pensaba que iban a ser imposibles de aprobar, gracias a sus explicaciones.

A José Jesús Castro, por hacer las clases entretenidas haciendo que los alumnos participáramos mucho y preocupándose porque lleváramos los contenidos de las clases aprendidos a casa, sin ninguna duda de por medio.

A los amigos de Villacañas, que hacían que me despejará de los estudios cada fin de semana.

David

A alguien querido y/o respetado

Índice general

Resumen V

Abstract VII

Agradecimientos IX

Índice general XIII

Índice de cuadros XVII

Índice de figuras XIX

Índice de listados XXI

Listado de acrónimos XXIII

1. Introducción 3

1.1. Videojuego multijugador masivo en línea 4

1.2. Importancia sobre la actualización rápida de datos 8

1.3. Propuesta de este Trabajo Fin de Grado (TFG) 9

1.4. Estructura del documento 11

2. Objetivos 13

2.1. Objetivo general 13

2.2. Objetivos específicos 13

2.2.1. Soporte de operaciones básicas relativas a rankings 14

2.2.2. Integración de técnicas avanzadas 14

2.2.3. Desarrollo de un sistema escalable 14

2.2.4. Desarrollo de un sistema adaptativo 14

2.2.5. Desarrollo de un sistema con arquitectura modular 14

2.2.6. Uso de tecnologías y estándares abiertos 14

3. Estado del arte **15**

3.1. Computación en la nube 15

3.1.1. Ventajas y desventajas 17

3.1.2. Arquitectura 19

3.1.3. Virtualización 25

3.1.4. Tipos 27

3.1.5. Actores 30

3.1.6. Proveedores de servicios en la nube 32

3.2. MMOs y rankings masivos 34

3.2.1. Applibot 34

3.2.2. Google Code Jam Ranking Library 38

3.2.3. Árboles 38

3.3. Herramientas de desarrollo web 41

3.3.1. Google App Engine (GAE) 41

3.3.2. Amazon Web Services (AWS) 57

3.3.3. Windows Azure 65

4. Método de Trabajo **71**

4.1. Metodologías de desarrollo 71

4.2. Metodología utilizada 72

4.2.1. Motivo 73

4.3. Medios hardware y software 73

4.3.1. Medios hardware 73

4.3.2. Medios software 74

5. Arquitectura del sistema **75**

5.1. Visión general de la arquitectura 75

5.2. Autenticación del usuario 77

5.2.1. Métodos de identificación 77

5.3. Soporte para la creación de usuarios y gestión de la sesión 79

5.3.1. Creación de usuarios 79

5.3.2. Procedimiento para la gestión de las sesiones 80

5.4. Soporte para la gestión de rankings 81

5.4.1. Procedimiento para la gestión de rankings 81

5.5. Establecimiento de puntuación 82

5.5.1. Uso de pull queues 84

5.5.2. Fragmentación del ranking 86
5.5.3. Uso de buckets 86
5.5.4. Cálculo del número máximo de tareas arrendadas 89
5.6. Obtención del rango 93
5.7. Organización y componentes del proyecto 95
5.7.1. Estructura general del proyecto 95
5.7.2. Estructura de la base de datos 97

6. Resultados **99**
6.1. Aspecto y funcionalidad del sistema 99
6.2. Distribución de trabajo 99
6.3. Integración de código 99

7. Conclusiones **103**

A. Ejemplo de anexo **107**

Referencias **109**

Índice de cuadros

3.1. Diferencias entre el Cloud Datastore y una base de datos relacional con los conceptos usados. 48

5.1. Estructura del directorio de la parte del servidor 96

Índice de figuras

1.1. Ingreso promedio por persona en los videojuegos MMO y F2P (Fuente: SuperData Research) 4

1.2. Mejores juegos de ordenador de recaudación por los ingresos en 2015 (Fuente: PCGamesN) 5

1.3. Mejores juegos de ordenador de recaudación por los ingresos en 2016 (Fuente: SuperData Research) 5

1.4. Mejores juegos MMO basados en subscripción (Fuente: SuperData Research) 6

1.5. Recaudación de videojuegos MMO P2P y F2P (Fuente: SuperData Research) 7

1.6. Videojuegos MMO que gozan de éxito 7

1.7. Número máximo de usuarios que se encuentran conectados en el mismo instante de tiempo en videojuegos MMO en cada año (Fuente: MMOData) . . . 9

1.8. Idea inicial de la arquitectura del sistema 10

3.1. Computación en la nube (Fuente: https://es.wikipedia.org/wiki/Computación_en_la_nube) 16

3.2. Arquitectura de computación en la nube (Fuente: http://timesofcloud.com) . 19

3.3. Usuarios correspondientes a cada capa de la arquitectura (Fuente: itsteziutlan) 24

3.4. Responsabilidades de cada capa de la arquitectura de la computación en la nube (Fuente: https://kkanakas.com/category/cloud/) 24

3.5. Virtualización(Fuente: http://www.diceitwise.com/how-does-cloud-computing-work-and-technology-behind-it/) 27

3.6. Tipos de nube(Fuente: http://www.keywordsuggests.com) 30

3.7. Modelo conceptual de los actores que realizan las tareas de *cloud computing*(Fuente: National Institute of Standards and Technology) 31

3.8. Proveedores de servicios en la nube 33

3.9. Logo de Applibot 35

3.10. Distribución de los *buckets*(Fuente: Google) 37

3.11. Ejemplo de un ranking con la estructura que usa Google Code Jam Ranking Library 38

3.12. Elementos de un árbol 40

3.13. Árbol general 41

3.14. Árbol binario 41
3.15. Funcionamiento de GAE(Fuente: http://www.manejandodatos.es/2014/12/como-funciona-el-front-end-back-end-en-google-app-engine/) 42
3.16. Arquitectura de aplicaciones web con GAE (Fuente: Google) 43
3.17. Aplicaciones que hacen uso de GAE(Fuente: Google) 43
3.18. Herramientas de desarrollo que son compatibles con GAE(Fuente: Google) . 44
3.19. Arquitectura para aplicaciones moviles con *Google Cloud Endpoints* (Fuente: https://www.3pillarglobal.com/insights/building-backend-applications-with-google-app-engine-google-cloud-endpoints-and-android-studio) 46
3.20. Ejemplo de índice 49
3.21. Éxito de memcache 50
3.22. Fallo de memcache 50
3.23. Funcionamiento de las *task queues* y los *cron jobs* (Fuente: https://www.slideshare.net) 53
3.24. Ejemplo de interfaz de AppStats Console 57
3.25. Servicios ofrecidos por la infraestructura de AWS (Fuente: http://blog.clearpathsg.com) 59
3.26. Arquitectura AWS con balanceo de carga (Fuente: http://creately.com) . . . 61
3.27. Arquitectura AWS (Fuente: http://creately.com) 66
3.28. Componentes de *Microsoft Azure* (Fuente: Microsoft) 67
3.29. Servicios ofrecidos por *Microsoft Azure* (Fuente: https://msftdude.files.wordpress.com) 67

5.1. Arquitectura del sistema 76
5.2. Diagrama de secuencia del funcionamiento de OAuth2 78
5.3. Diagrama de clases correspondiente a la gestión del ranking 83
5.4. Diagrama de secuencia del manejador que se encarga de establecer las puntuaciones 83
5.5. Diagrama de sequencia de la *pull queue* que se encarga de establecer la puntuación de los usuarios 85
5.6. Diagrama de clases correspondiente a la gestión del bucket 87
5.7. Diagrama de sequencia de la *push queue* que se encarga de actualizar la información de los *buckets* 88
5.8. Diagrama de clases correspondiente a la gestión del tiempo 91
5.9. Diagrama de clases correspondiente a la gestión de la configuración del sistema 91
5.10. Diagrama de secuencia de la *push queue* que se encarga de calcular el nuevo número máximo de tareas arrendadas por las *pull queues* 94
5.11. Estructura del directorio de la parte del servidor del sistema 97
5.12. Tipos de entidades definidas en el *Datastore* 98

Índice de listados

5.1. Pseudocódigo de la pull queue que se encarga de establecer las puntuaciones 84
5.2. Pseudocódigo de la push queue que se encarga de actualizar los buckets . . 87
5.3. Pseudocódigo de la función que se encarga de actualizar la información de cada bucket 87
5.4. Pseudocódigo de la *push queue* que se encarga del análisis y el cálculo del nuevo número máximo de tareas que puede arrendar una *pull queue* 92
5.5. Pseudocódigo de la función que se encarga de cálcular del nuevo número máximo de tareas que puede arrendar una *pull queue* 92
5.6. Pseudocódigo de la función que se encarga calcular el rango de un usuario . 93

Listado de acrónimos

AMI	Amazon Machine Image
API	Application Programming Interface
AWS	Amazon Web Services
CPU	Central Processing Unit
CS:GO	Counter Strike: Global Offensive
DNS	Domain Name System
EBS	Elastic Block Storage
EC2	Elastic Compute Cloud
ECS	EC2 Container Service
F2P	Free to Play
GAE	Google App Engine
GQL	Google Query Language
IAAS	Infraestructure as a Service
IP	Internet Protocol
ML	Machine Learning
MMO	Massively Multiplayer Online
MSF	Microsoft Solution Framework
P2P	Pay to Play
PAAS	Platform as a Service
S3	Simple Storage Service
SAAS	Software as a Service
SDK	Software Development Kit
SES	Simple Email Service
SNS	Simple Notification Service
SO	Sistema operativo
SQL	Structured Query Language

SQS	Simple Queue Service
SSL	Secure Sockets Layer
SWF	Simple Workflow Service
RDS	Relational Database Service
RUP	Rational Unified Procces
TFG	Trabajo Fin de Grado
TI	Tecnologías de la información
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VPC	Virtual Private Cloud
WoW	World of Warcraft
XP	Extreme Programming

-

Capítulo 1

Introducción

Con el paso de los años, los videojuegos han ido teniendo cada vez más importancia en la sociedad, comenzando por videojuegos que tenían poca calidad gráfica y escasa argumentación, hasta llegar a los de ahora gracias a los avances de las tecnologías, en los que cada vez hay más personas que invierten varias horas jugando con ellos, sin importar la edad ni el sexo. Los videojuegos enseñan cosas que de otras formas supondría más esfuerzo aprenderlas y permiten ejercitar la mente. Estos se han convertido en el único medio cultural que es consumido por igual por toda la sociedad, por lo que goza de gran importancia en la industria cultural.

Con el avance de la tecnología, han implementado nuevas formas de jugar en los videojuegos, en las que se incluye jugar con varios jugadores en una misma partida sin tener que estar usando el mismo dispositivo todos los jugadores que se encuentren en ella, evitando de esta forma que el usuario tenga que jugar sólo o tenga que depender de otra persona que conozca para jugar en el mismo dispositivo. Con la nueva forma de jugar, mediante una conexión a Internet, una persona puede jugar con muchas personas que se encuentren en cualquier punto geográfico sin apenas conocerlas.

Los videojuegos online, más concretamente los videojuegos multijugador masivo en línea, conllevan un problema. Este consiste en que es necesario que los datos de los usuarios se actualicen demasiado rápido, sin tener en cuenta el número de usuarios que lleven a cabo esta operación en un momento determinado. Por este motivo, se tienen que implementar mecanismos que permitan llevar a cabo el proceso de actualización lo más rápido posible.

Uno de estos videojuegos que se ha tenido que enfrentar a este problema es el famoso *World of Warcraft (*WoW*)*, que utiliza un sistema de clasificación de jugadores donde la posición de cada viene determinada por unos puntos, obteniendolos al ganar batallas y cuantas más batallas de forma seguida se gane, más puntos se suman a la puntuación actual, llamada *índice*. Este éxitoso juego tuvo que hacer frente al problema y solucionarlo para seguir teniendo gran éxito durante mucho tiempo y beneficios, porque a ningún jugador le gustaría perder puntos conseguidos o tener la sensación de que no te los han añadido por la lenta actualización de la puntuación debido al exceso de usuarios que están jugando en ese momento.

Videojuego multijugador masivo en línea

Con la evolución de Internet y la informática, se favoreció la creación de videojuegos para el ordenador, lo que hizo que aprovecharan las posibilidades que ofrecía la red. De esta ventaja más tarde se aprovecharían las demás plataformas. De esta forma este tipo de videojuegos consisten en que **un gran número de jugadores juegan una misma partida conectados entre sí mediante la conexión de Internet**. Los jugadores interactúan para competir entre ellos o cooperar para conseguir unos objetivos. Este tipo de videojuegos se denomina **videojuegos multijugador masivo en línea(MMO)**. Muchos de estos videojuegos requieren que los usuarios inviertan una gran cantidad de tiempo, lo que lleva a que gran cantidad de usuarios estén jugando a la vez en cualquier instante de tiempo.

AVERAGE REVENUE PER USER FOR MAJOR FREE-TO-PLAY MMO TITLES—WORLDWIDE

Rank	Title	Publisher	Average revenue per user
1	**World of Tanks**	Wargaming.net	$4.51
2	**Team Fortress 2**	Valve Corporation	$4.36
3	**Guild Wars 2**	ArenaNet	$3.88
4	**War Thunder**	Gaijin Entertainment	$3.26
5	**Planetside 2**	Sony Online Entertainment	$2.86
6	**Combat Arms**	Nexon	$2.81
7	**Crossfire**	SmileGate	$1.58
8	**DOTA 2**	Valve Corporation	$1.54
9	**Heroes of Newerth**	S2 Games	$1.48
10	**League of Legends**	Riot Games	$1.32

Shown: Average dollar amount spent by a player in the last twelve months on top free-to-play online games, ending in March, 2014. Numbers calculated based on monthly transactions collected from publishers, payment service provides and other industry sources, and dividing the estimated total dollar earnings by the monthly active user base. Please note that this list does not represent a worldwide top ten of free-to-play titles. Copyright © 2014 SuperData Research. All numbers subject to change. For more information on our methodology, please visit www.superdataresearch.com

Figura 1.1: Ingreso promedio por persona en los videojuegos MMO y F2P (Fuente: SuperData Research)

Con el éxito obtenido, en la actualidad existen muchos juegos de este tipo. El éxito de estos videojuegos se debe a que la mayoría son **Free to Play (F2P)**, como *League of Legends*, que son videojuegos a los que se puede jugar de manera gratuita, pero ofrecen al jugador algunas ventajas que se obtienen al ser pagadas, y también porque son juegos sociales, permitiendo que varias personas se comuniquen. Los videojuegos **F2P** obtienen grandes beneficios porque

Top Grossing PC Games by Revenue, 2015 (mil $)

1	*League of Legends*	Tencent/Riot Games	$1,628
2	*CrossFire*	SmileGate	$1,110
3	*Dungeon Fighter Online*	Neople	$1,052
4	*World of Warcraft*	Activision	$814
5	*World of Tanks*	Wargaming.net	$446
6	*Lineage I*	NCSOFT	$339
7	*Maplestory*	Nexon	$253
8	*DOTA 2*	Valve Corporation	$238
9	*Counter-Strike: Global Offensive*	Valve Corporation	$221
10	*Grand Theft Auto V*	Take-Two Interactive	$205

Figura 1.2: Mejores juegos de ordenador de recaudación por los ingresos en 2015 (Fuente: PCGamesN)

juegan muchas personas al ser gratuito y se puede llegar a ser el mejor sin realizar ninguna compra dentro de este, aunque siempre existe el caso de que acabas gastando dinero por alguna cosa que tenga un costo que puede que no sea bastante caro o que permite obtener una mejora y merezca la pena comprarlo o que simplemente guste a la persona.

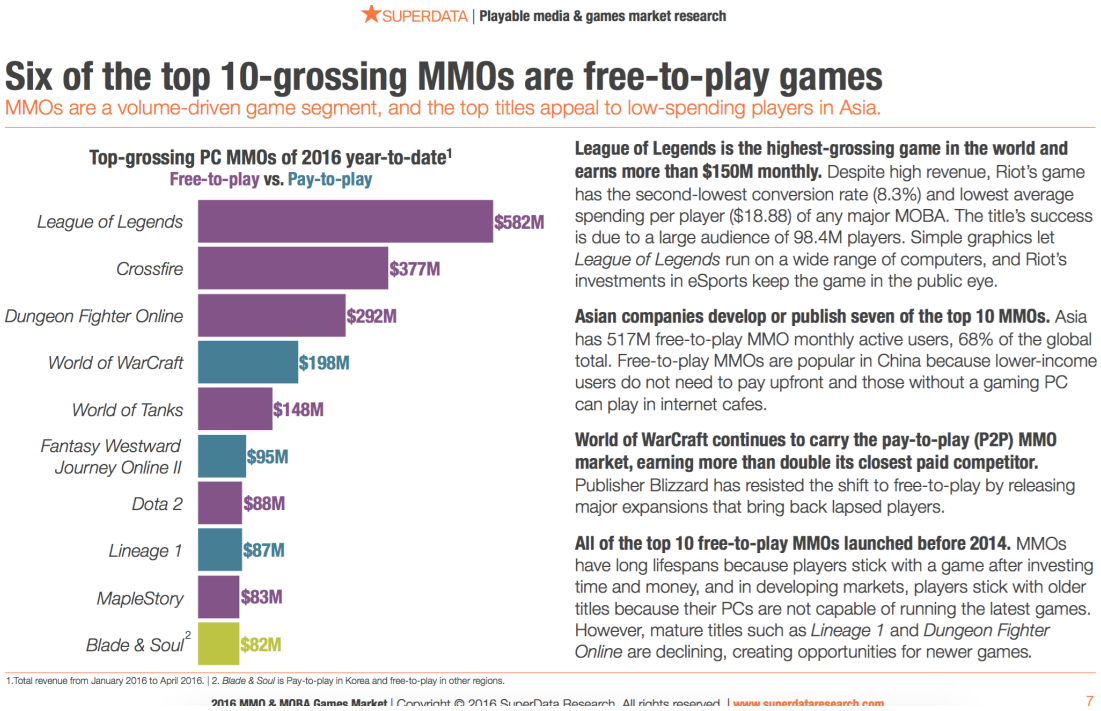

Figura 1.3: Mejores juegos de ordenador de recaudación por los ingresos en 2016 (Fuente: SuperData Research)

El mayor ingreso promedio por persona en los videojuegos **MMO** que son **F2P** en 2014 es de 4,51 dólares en el videojuego ***World of Tanks***, que podría no significar tanto dinero, pero

hay que pensar que existen muchos jugadores jugando a ese videojuego(ver Figura 1.1).

En 2015, uno de los videojuegos de ordenador que recaudo más dinero fue ***League of Legends*** con 1.628 millones de dólares, videojuegos que hay que recalcar que es **MMO** y **F2P**(ver Figura 1.2).

En 2016, más concretamente desde Enero hasta Abril, ***League of Legends*** sigue siendo el mejor en relación a la recaudación, porque en ese tiempo ha recaudado 582 millones de dólares y los videojuegos **F2P** son los que más beneficio han obtenido en relación a los **Pay to Play**(**P2P**)(ver Figura 1.3).

Los videojuegos que no son **F2P** también logran beneficios porque se paga una subscripción mensual para poder jugar a ellos y a estos se les llama **P2P**. En 2013, **WoW** fue el videojuego que más dinero recaudó con las subscripciones llegando a ganar 1.041 millones de dólares(ver Figura 1.4).

TOP SUBSCRIPTION-BASED MMO TITLES, 2013—WORLDWIDE

RANK	TITLE	PUBLISHER	WORLDWIDE REVENUES (MIL $)	MARKET SHARE 2013
1	World of Warcraft (East & West)	Activision/Blizzard	$1,041	36%
2	Lineage 1	NCsoft	$253	9%
3	TERA: Online	NHN Corporation	$236	8%
4	Star Wars: The Old Republic	Electronic Arts	$165	6%
5	Lord of the Rings Online	Turbine, Inc.	$104	4%
6	EVE Online	CCP Games	$93	3%
7	Aion	NCsoft	$88	3%
8	Blade and Soul	NCsoft	$65	2%
9	Lineage 2	NCsoft	$45	2%
10	RIFT	Trion	$36	1%
	Worldwide market for subscription-based MMOs, 2013		$2,882	

Source: SuperData Research, Inc. Worldwide market, revenue distribution and title-level earnings based on the monthly spending of 36.9 million digital gamers, worldwide, collected from developers, publisher and payment service providers. Pay-to-play MMO here defined as Massively multiplayer online games that earn revenue from subscriptions, expansion packs, and microtransactions based virtual items and services (e.g. experience boosts, items, mounts and server transfer fees. For more information about our awesome methodology, please visit: www.superdataresearch.com

Figura 1.4: Mejores juegos MMO basados en subscripción (Fuente: SuperData Research)

EN 2015, han recaudado más dinero los videojuegos **F2P**, con 16.9 billones de dólares, que los **P2P**, con 3.1 billones de dólares, y se estima que los próximos tres años sigan igual(ver Figura 1.5).

Como se muestra en las imágenes, este tipo de videojuegos recaudan mucho dinero, lo que haría que un problema como el que se trata en este trabajo podría provocar la pérdida de importantes beneficios, sobre todo de grandes empresas como *Blizzard Entertainment*, creadores de WoW o *Riot Games*, creadores de *League of Legends*.

Estos videojuegos suelen hacer uso de **un ranking**, que es una lista que establece una relación entre el conjunto de elementos que la constituyen, es decir, que hay una característica común en todos los elementos de la lista que hace que cada elemento se sitúe en una determinada posición en ella, para tener almacenado el rango o el nivel del jugador y pueda emparejar a personas que tengan un rango o nivel similar en partidas.

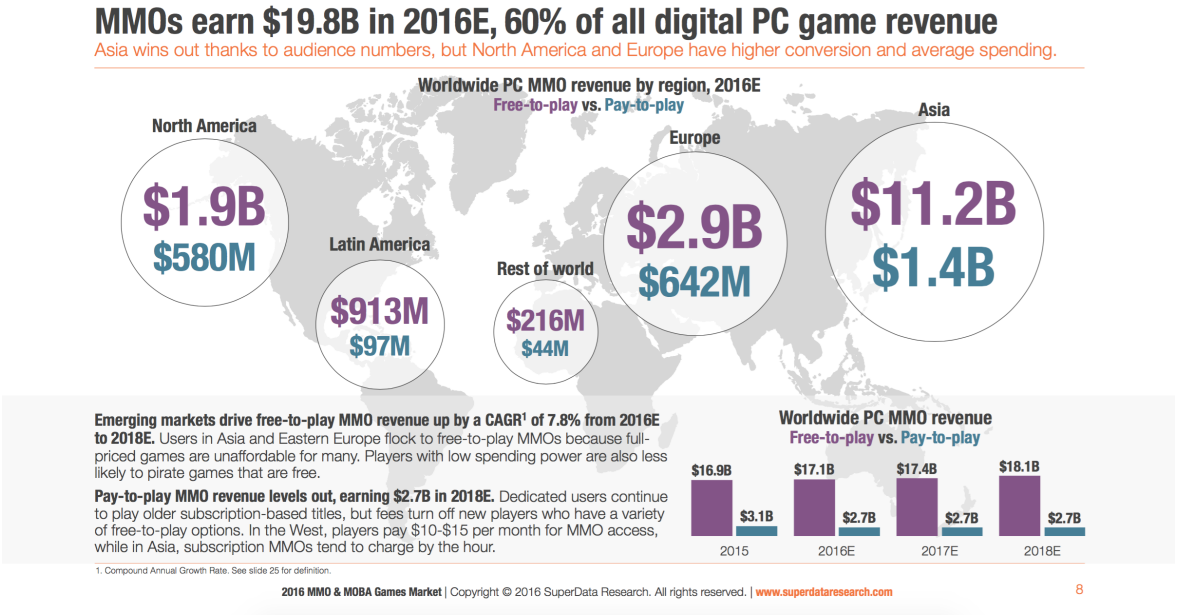

Figura 1.5: Recaudación de videojuegos MMO P2P y F2P (Fuente: SuperData Research)

Uno de los videojuegos más populares de este tipo desde sus inicios en 2004 y que sigue cosechando éxito es **WoW**, el videojuego de rol en el que los jugadores manejan un personaje dentro de un mundo explorando el entorno, llevando a cabo misiones, interactuando con personajes que pueden ser otros jugadores y combatiendo contra varios monstruos y otros jugadores. Como se ha visto, es un videojuego que tiene tantas cosas para hacer que hay que dedicarle mucho tiempo y nunca pasa de moda. Otros videojuegos que han tenido éxito son ***EVE online*** y ***Guild Wars 2***.

Figura 1.6: Videojuegos MMO que gozan de éxito

El éxito de los videojuegos MMO ha llevado a la creación de un nuevo deporte, denominado **e-sports**, que son competiciones donde varios jugadores compiten en equipo en un videojuego. Esto comenzó con el éxito de retransmisiones de los videojuegos ***League of Legends***, ***Counter Strike: Global Offensive (*CS:GO*)***, ***Hearthstone***, **WoW** y ***Dota 2***. Los distintos competidores de este deporte se sientan delante de un ordenador para llevar a cabo la partida. Este tipo de competiciones están siendo muy virales, lo que ha hecho que actualmente en algunos locales se televisen estas, al igual que se hace con cualquier deporte. Además, con el paso del tiempo, se añaden más cadenas de televisión que reproducen o hablan sobre este tipo de acontecimiento, algo que inició con la plataforma de *streaming* de video en vivo llamado *twich*. Incluso se ha pensado en añadirlo como deporte olímpico. Algunos clubes deportivos ya se han unido a este sector como el caso del equipo de futbol español *Valencia CF*. Esto conlleva gran movimiento de dinero, incluso más que la industria del cine y la música y va mejorando con el paso del tiempo.

Importancia sobre la actualización rápida de datos

Existen situaciones o contextos en los que la velocidad de actualización de unos determinados datos es importante, como es en el caso de **los videojuegos multijugador masivo en línea**. En este caso, si la actualización no se lleva a tiempo lo más rápido posible, puede que ocurran varios problemas que tenemos que evitar. Uno de estos problemas puede ser que tarde en mostrar los datos actualizados a los usuarios, aunque este problema puede que no sea tan importante como el de actualizar los datos que habían sido actualizados anteriormente pero no se hayan aplicado todavía los cambios, por lo tanto, si esos datos que se actualizan en ese momento necesitan la información anterior, como es el caso de una puntuación, en el que se le añaden puntos a la antigua puntuación, podría provocar serios problemas.

En instantes de tiempo en los que hay pocos usuarios usando un videojuego específico, estos problemas no suelen ocurrir, pero lo normal es que haya muchos usuarios(ver Figura 1.7), y esto puede generar gran lentitud en la aplicación de la actualización. Por lo tanto, hay que evitar que con el aumento del número de actualizaciones en un momento se ralentice demasiado el proceso para evitar problemas.

Applibot[1] se tuvo que enfrentar a un problema muy común y muy similar al que se da en este trabajo, **el problema de clasificación de los jugadores**, cuyos requisitos eran que muchos usuarios jugaban a su juego, que con algunas acciones que realizaba el usuario cambiaba su puntuación y que se tenía que mostrar la última puntuación del jugador en una página web [FK16].

La obtención de un rango puede llegar a ser sencilla, pero no llegar a ser **escalable y rápida**. Por ejemplo, se puede hacer una consulta que obtenga los usuarios que tengan una puntuación mayor que la del que lo solicita, y así obtener el rango mediante el número de

[1] https://www.applibot.co.jp

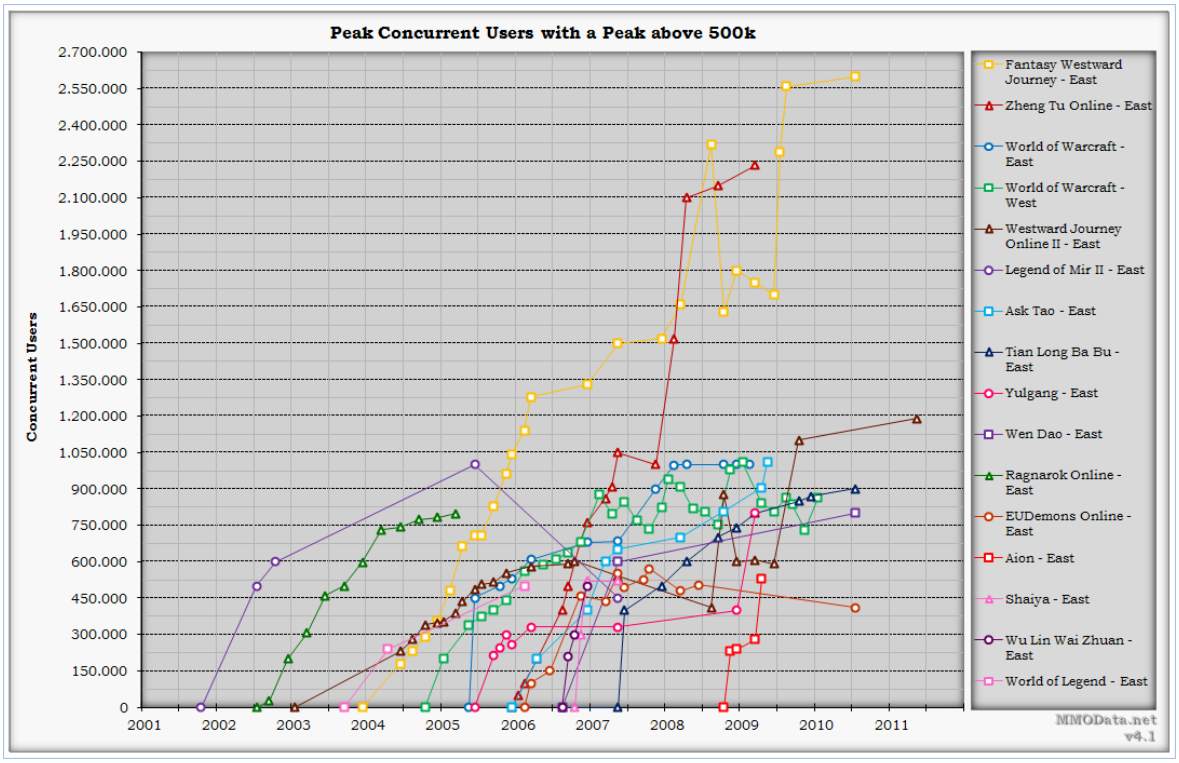

Figura 1.7: Número máximo de usuarios que se encuentran conectados en el mismo instante de tiempo en videojuegos MMO en cada año (Fuente: MMOData)

usuarios obtenidos, pero esto sería muy lento si se tiene que hacer cada vez que un usuario quiere ver su rango y sobre todo, si hay muchos usuarios que tienen mayor puntuación que el usuario que solicita el rango. Este proceso sería lento, costoso y progresivamente iba a peor con el aumento de usuarios. Este caso de estudio se verá más tarde en el capítulo 3 más profundamente.

Este problema es muy común en la actualidad, ya que la mayoría de los videojuegos permite comunicar a todos los jugadores y hacen uso de un ranking, para hacer más competitiva la partida y hacer que los jugadores tengan más ganas de jugar.

Propuesta de este Trabajo Fin de Grado (TFG)

Una vez expuesto el problema que abarca este proyecto, se va a presentar la idea que va a servir como base para definir la solución. La propuesta consiste en desarrollar **un sistema adaptativo** que optimice el uso de los recursos, para hacer que en cada momento haya un número adecuado de recursos y cada recurso tenga la carga adecuada, y que funcione de manera correcta. De esta manera, el proceso de actualización se llevaría de manera rápida sin depender del número de usuarios con el mínimo costo posible.

Con esta propuesta, el sistema sería escalable en varios sentidos: el sistema respondería perfectamente a las peticiones realizadas por los clientes en todo momento, haciendo que la

gestión de los datos se haga a una velocidad adecuada en cualquier instante de tiempo sin depender del número de usuarios que quieran hacer la misma acción en la base de datos, y en el futuro se podrá integrar en otros sistemas y mejorar con facilidad, debido a la implementación realizada siguiendo buenas prácticas. También se optimizaría el uso de los recursos, gracias al uso de un algoritmo que analizaría unos datos para obtener información útil para llevar a cabo la optimización, disminuyendo de esta forma los costos correspondientes y repartiendo de forma adecuada la carga entre los distintos recursos disponibles.

Una idea más general sobre el funcionamiento y la arquitectura del sistema es como la que se muestra en la figura 1.8. Mediante el uso de un dispositivo, el usuario solicita la actualización de su puntuación de manera indirecta con la realización de alguna acción. Esta solicitud le llega al servidor, que va a seleccionar la operación, de todas las disponibles, que el usuario quiere realizar y el núcleo de procesamiento donde se va a realizar. El núcleo de procesamiento se encarga de aplicar la nueva puntuación del usuario en el ranking, donde se encuentran almacenadas las puntuaciones de todos los usuarios. Este ranking se encuentra almacenado en la base de datos del servidor. Por último, el servidor proporciona una respuesta al usuario en la que indica cuál es el resultado sobre la realización de la actualización y su nueva puntuación, que es mostrada en el dispositivo que el usuario esté usando.

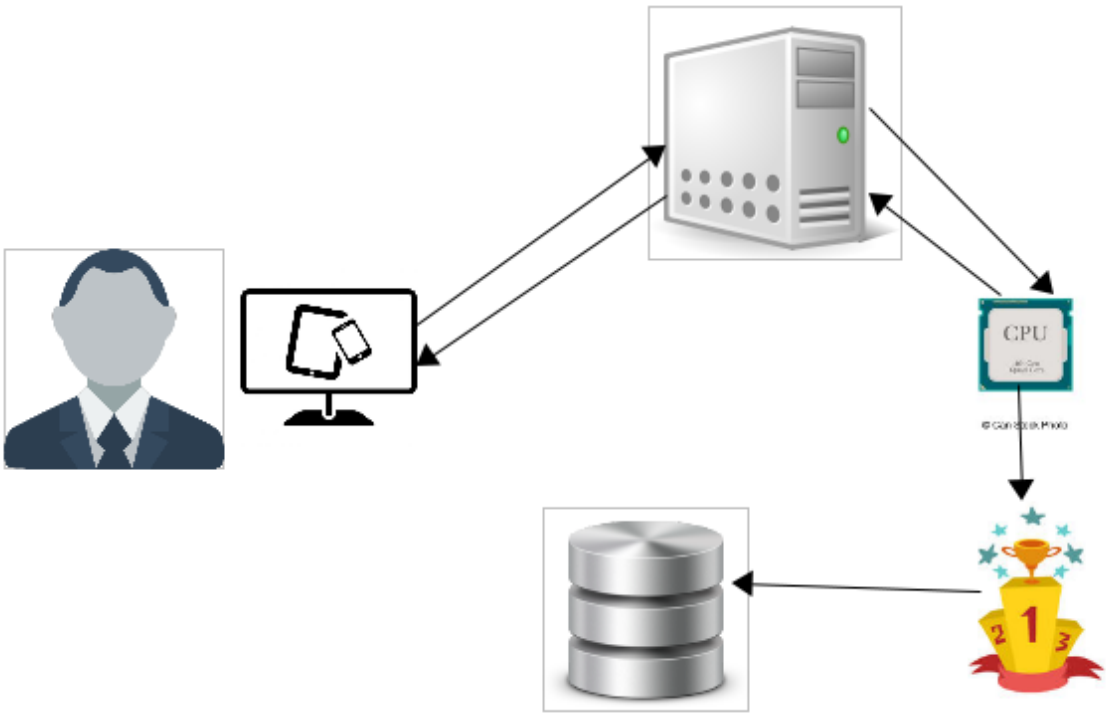

Figura 1.8: Idea inicial de la arquitectura del sistema

Para desarrollar este sistema, se va a utilizar ***Google App Engine*** para lograr la escalabilidad y disponibilidad de este. Se va a desarrollar una aplicación cliente en *Python*, con la

librería *Requests*, para realizar las pruebas iniciales y una aplicación cliente en *Android*, con *Google Cloud Endpoints*, para las pruebas finales. Las herramientas utilizadas se verán con detalle en el capítulo 3.

Estructura del documento

A continuación, se va a mostrar cómo va a estar formado el documento con una breve descripción y el orden de cada apartado:

Capítulo 2: Objetivos

Se van a listar, con un breve descripción, las metas que se deben alcanzar con la realización de este trabajo y que van a indicar cuando el resultado es satisfactorio.

Capítulo 3: Estado del arte

En este capítulo se va a exponer el conocimiento obtenido a través de varias fuentes, que ha servido para el desarrollo de este proyecto. Más concretamente, recopila toda la información importante relacionada con la computación en la nube, los videojuegos MMO y rankings masivos y las herramientas de desarrollo web utilizadas.

Capítulo 4: Método de Trabajo

Se va a hablar sobre los tipos de metodologías que existen, y después se va a hacer hincapié en la metodología que se ha seguido para el desarrollo de este trabajo y se van a exponer los motivos por los que se ha elegido esa. Por último, se van a mostrar las herramientas software y hardware que han permitido realizar el trabajo.

Capítulo 5: Arquitectura del sistema

Se va a mostrar lo que se ha ido realizando a los largo del desarrollo del proyecto para llegar a la solución requerida.

Capítulo 6: Resultados

Aquí se muestra los resultados obtenidos al aplicar las soluciones.

Capítulo 7: Conclusiones

Para finalizar se expondrá lo más importante del proyecto y todas las posibles mejoras que se pueden llegar a aplicar.

Capítulo 2

Objetivos

Una vez expuestas las ideas principales en la introducción, en este capítulo se va a definir lo que se quiere conseguir con el desarrollo del trabajo. Primero se hará hincapié sobre el objetivo principal de este trabajo, la meta final que se debe de alcanzar, y después sobre un conjunto de objetivos que derivan del objetivo principal. Estos objetivos van a mostrar cuando el desarrollo del proyecto ha llegado a su fin una vez cumplidos todos. De esta manera se definen los pasos que hay que seguir para llegar a la meta deseada.

Objetivo general

Como objetivo principal, se propone el diseño, desarrollo y despliegue de un sistema adaptativo que va a permitir la gestión de *rankings* en juegos masivos multijugador. Este sistema realizará la gestión de los *rankings* de la manera más rápida, sin importar el número de usuarios disponible en el mismo momento. Se quiere hacer más sencillo el uso del sistema creado para que otros desarrolladores puedan instanciarlo e integrarlo en sus proyectos.

Para que este sistema sea reutilizable para otros proyectos, se ha dividido en módulos para hacer más entendible el código y se puedan añadir más módulos o se puedan usar de manera sencilla. Además, hay algunos valores estáticos que están separados del código en un archivo de configuración para que el usuario pueda modificarlos a su gusto, sin tener que meterse dentro del código y buscarlos para llevar a cabo su modificación. También se aplican nombres entendibles para saber a simple vista lo que es cada cosa.

Para el desarrollo de este proyecto se ha utilizado como base las ideas que ha implementado la empresa *Applibot* para resolver el mismo problema de actualización que se plantea en este trabajo en uno de sus videojuegos [FK16] y la librería proporcionada por *Google* para la gestión de rankings *Google Code Jam Ranking Library* [Goo09].

El despliegue de esta aplicación se llevará a cabo mediante *Google App Engine*.

Objetivos específicos

A continuación, se van a exponer los objetivos específicos que se van a plantear y que van a formar parte del objetivo general.

Soporte de operaciones básicas relativas a rankings

Permitirá que un gran número de usuarios puedan realizar operaciones básicas relativas a la creación, actualización y consultas de *rankings* en videojuegos masivos multijugador en cualquier momento y sin ocasionar ningún problema.

Integración de técnicas avanzadas

En este proyecto se van a integrar técnicas avanzadas para poder lograr una tasa de actualizaciones estable y sostenida en el tiempo.

Desarrollo de un sistema escalable

El sistema a implementar va a ser escalable respecto a las necesidades de cómputo de cada momento, teniendo en cuenta el número de jugadores y la frecuencia de actualización del ranking. Este sistema va a estar preparado para que haya cualquier número de personas usándolo en todo momento, utilizando siempre un número adecuado de núcleos de procesamiento para evitar costos innecesarios y permitir que haga las acciones que los usuarios requieran.

Desarrollo de un sistema adaptativo

Mediante un sub-módulo de *Inteligencia Artificial* que va a contener el sistema, se va a estimar la mejor manera posible el despliegue de nuevos nodos de procesamiento. Ese sub-módulo se va a encajar de analizar el tiempo que se tarda en llevar a cabo una acción aplicada por varias personas a la vez, en este caso el establecimiento de la puntuación del jugador, para saber cuántos núcleos de procesamiento son necesarios y cuanta información pueden llegar a procesar para evitar demasiada lentitud en la realización de las tareas.

Desarrollo de un sistema con arquitectura modular

El sistema va a estar estructurado de manera modular para facilitar la integración de nuevos módulos a nivel de software. De esta forma, se va a saber qué hace cada módulo sin necesidad de tener que mirar el código que contiene.

Uso de tecnologías y estándares abiertos

Para la implementación de este sistema se usan tecnologías y estándares que nos van a garantizar que el sistema sea portable, por lo que se puede utilizar en cualquier plataforma. Este sistema está desarrollado en *Python* [Gon], que al ser un lenguaje interpretado es más portable.

Capítulo 3

Estado del arte

En este capítulo se va a exponer la información que se ha recopilado necesaria para el desarrollo del proyecto y que va a formar la base del conocimiento. Más concretamente, se va a hablar de *Applibot*, que tuvo el mismo problema que se habla en este trabajo, y de cómo planteo y solucionó el problema y sobre las herramientas, como *Google App Engine* y la librería *Google Code Jam Ranking*, y conceptos, como la estructura de datos en árbol y la computación en la nube, que tienen mayor importancia en cuanto al desarrollo de este sistema.

Computación en la nube

La computación en la nube, o como normalmente se nombra, **cloud computing**, se basa en los servicios que son ofrecidos mediante la red, los cuales se puede acceder desde un navegador web. Cuando se usan estos tipos de servicios, **la información usada y almacenada es procesada y ejecutada por un servidor que se encuentra en la nube**. El término nube hace referencia a la infraestructura desde el cual las empresas y los usuarios pueden **acceder a los servicios desde cualquier lugar y bajo demanda**, es decir, que hace referencia de manera metafórica a *Internet*. Este tiene un papel importante, debido a que representa el medio o la plataforma mediante la cual se ofrecen y hacen accesibles los servicios.

La visión que se propone en este tipo de computación consiste en que **los servicios se encuentren disponibles cuando se demanden**, en vez de anticiparse a la provisión de los servicios. De esta forma, los usuarios solamente tienen que pagar a los proveedores cuando acceden a ellos. En este modelo, los usuarios no acceden a los servicios en base a dónde se alojan, sino a sus requisitos. La computación en la nube **permite alquilar infraestructura, entornos de ejecución y servicios basados en pagos por uso** [BVS09].

Estos tipos de servicios ofrecidos pueden ser el almacenamiento, el uso de aplicaciones alojadas en *Internet*, el correo electrónico, etc(ver Figura 3.1).

La computación en la nube está formada por **una serie de elementos que permiten su funcionamiento**. Cada elemento tiene un propósito y desempeña un papel específico [VVE13].

- **Clientes.** Son los dispositivos mediante los cuales el usuario interactúa para solicitar el servicio.

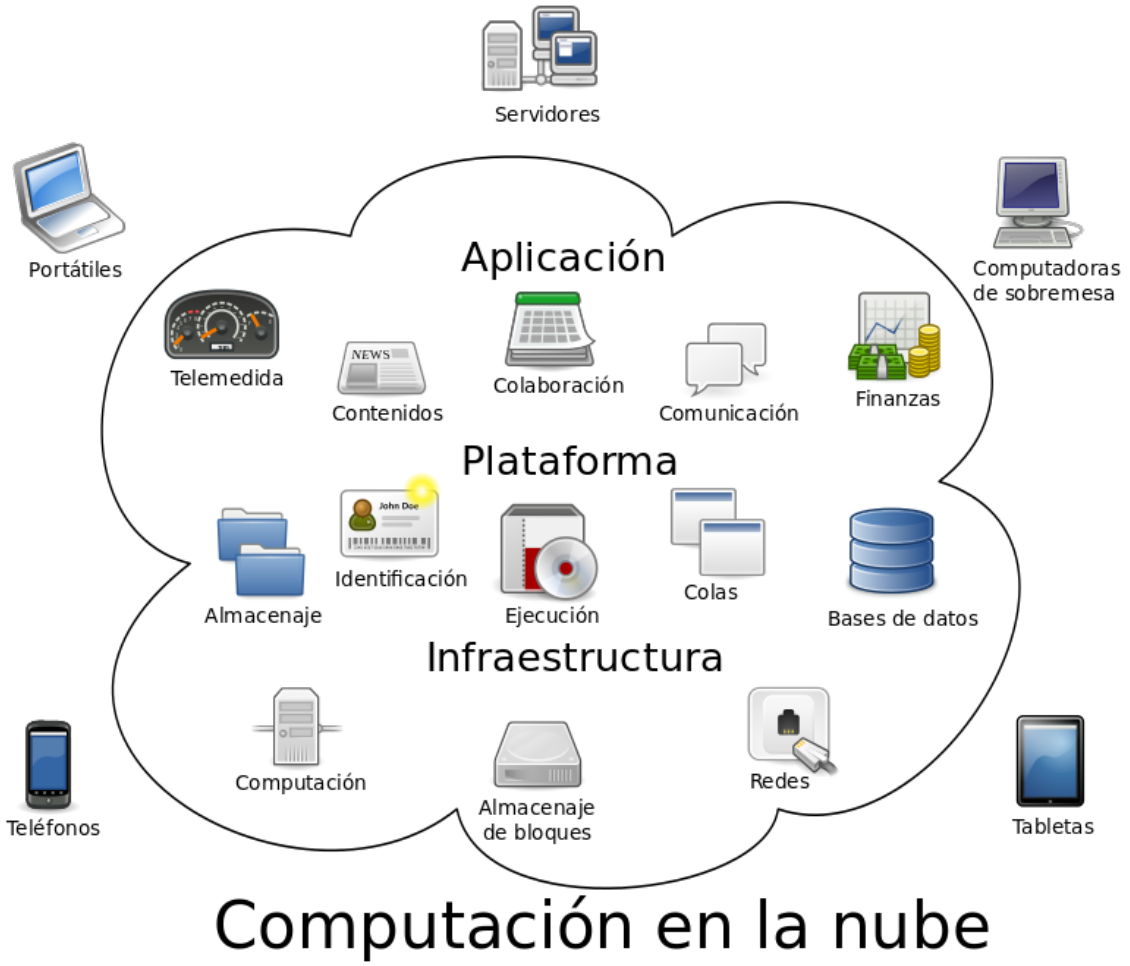

Figura 3.1: Computación en la nube (Fuente: https://es.wikipedia.org/wiki/Computación_-en_la_nube)

- **Centro de datos.** Es el conjunto de servidores donde se encuentra alojada la aplicación a la que el cliente se subscribe. Se suele aplicar cada vez más la virtualización de los servidores, lo que significa que se puede instalar software que permita instalar varias instancias de servidores virtuales.
- **Servidores distribuidos.** Son los servidores que no se encuentran alojados en el mismo sitio, sino que están en lugares geográficamente distintos. Aunque la sensación del cliente es como si estuvieran juntos.

Esta innovadora tecnología tiene varias características que merecen ser nombradas. Una de las características más importantes es que no se necesita un computador que sea muy potente, **sólo es necesario tener un aparato que proporcione conexión a *Internet***, debido a que los servidores donde se encuentran los programas que se van a usar son los que se encargan de llevar a cabo las tareas complicadas, que anteriormente se tenían que realizar en nuestro

computador de manera local, y los ficheros pueden ser almacenados en la nube. Otra de sus características importantes es que **los servicios ofrecidos pueden pagarse dependiendo de alguna medida de consumo**, por lo que el usuario únicamente paga por lo que consume según la tarifa que tenga establecida.

Otras de las características de la computación en la nube son las que se mencionan a continuación [Avi11]:

- **Auto reparable.** Si ocurre un fallo, la última copia de seguridad que hay de la aplicación automáticamente pasa a ser la copia primaria y se genera uno nuevo a partir de ésta.
- **Escalable.** Todo el conjunto del sistema es predecible y eficiente.
- **Virtualización.** Las aplicaciones se pueden ejecutar en cualquier máquina, permitiendo de esta manera que el usuario pueda utilizar la plataforma que desee en su terminal.
- **Alto nivel de seguridad.** Permite que diferentes usuarios compartan la infraestructura sin que se tenga que preocupar de ello y sin poner en un compromiso su seguridad y privacidad.
- **Disponibilidad de la información.** La información se encuentra disponible en todo momento debido a que esta almacenada en Internet, permitiendo de esta forma que podamos acceder a ella, con una autorización previa para que no pueda acceder cualquier persona a la información privada de otra, desde cualquier dispositivo conectado a la nube.

Ventajas y desventajas

El trabajo con datos en la nube puede beneficiar a cualquiera si se usa de manera correcta y en la medida de lo posible. Algunas de las ventajas de la computación en la nube son las siguientes [Avi11]:

- **Costo eficiente.** Es el método más eficiente con mejor costo para usar, mantener y actualizar. A las compañías le cuesta mucho dinero el software que se encuentra en el escritorio tradicional, debido a que todos los derechos de licencia para varios usuarios pueden llegar a ser muy caros. En cambio, la nube se encuentra disponible a precios más baratos y les permite ahorrar además de en licencias, en la administración del servicio y en los equipos necesarios, lo que permite reducir a la empresa los gastos relacionados con las Tecnologías de la información (TI).
- **Almacenamiento casi ilimitado.** La capacidad que nos ofrece la nube para el almacenamiento es prácticamente ilimitada, por lo que no hay que preocuparse por quedarse sin espacio o aumentar la disponibilidad de espacio de almacenamiento actual.
- **Copia de seguridad y recuperación.** Debido a que todos los datos son almacenados

en la nube, la copia de seguridad de estos y su recuperación es más sencilla que si se almacenan en un dispositivo físico.

- **Integración automática de software.** Esto quiere decir que no se necesita medidas adicionales para personalizar e integrar las aplicaciones de un usuario según sus preferencias. El usuario puede seleccionar los servicios y aplicaciones que se adapte mejor a su empresa.
- **Despliegue rápido.** Cloud Computing proporciona una rápida implementación, permitiendo que todo el sistema sea completamente funcional en poco tiempo.
- **No necesita hardware.** Todo se encuentra alojado en la nube, por lo que ya no es necesario un centro de almacenamiento físico.
- **Flexibilidad para el crecimiento.** La nube es fácilmente escalable, lo que provoca que las empresas puedan añadir o eliminar recursos dependiendo de sus necesidades.
- **Portabilidad de información.** Lo que empezó a ser para empresas, con el paso del tiempo ha acabado siendo también para usuarios particulares, debido a que ahora existen muchos dispositivos que hacen uso de los servicios de la nube.
- **Uso eficiente de la energía.** Los servidores que se encuentran en los centros de datos tradicionales consumen mucha más energía de la que se necesita en realidad. Sin embargo, la energía que se consume en las nubes es únicamente la necesaria, disminuyendo de esta forma el desperdicio de esta.

Al igual que esta tecnología tiene sus ventajas, también tiene sus desventajas, que se tienen que tener en cuenta antes de usarlas [Avi11].

- **Problemas técnicos.** Es cierto que se pueden acceder a los datos almacenados en la nube en todo momento y desde cualquier lugar, pero hay momentos en los que este sistema puede tener algún problema que evite que se pueda acceder a los datos. Esta tecnología es propensa a interrupciones y otras cuestiones técnicas. Hasta los mejores proveedores de servicios en la nube tiene estos problemas, aunque se mantenga un alto nivel de mantenimiento. Además, se necesita una buena conexión a Internet para acceder al servidor en todo momento, lo que hace que el usuario este sujeto a la cobertura de red.
- **Seguridad en la nube.** Aunque existan varias medidas de seguridad para evitar que una persona ajena a esos datos pueda acceder a ellos, esos datos confidenciales se van a entregar a un proveedor de servicios en la nube de terceros. Esto podría poner en riesgo a una empresa o a un usuario cualquiera. Por ese motivo, hay que estar seguro de elegir al proveedor de servicio más fiable, que mantendrá toda la información segura.
- **Propenso al ataque.** Esta tecnología es propensa a amenazas y ataques de *hackers*, lo que haría a cualquier empresa vulnerable.

- **Velocidad de respuesta.** En algunos casos la velocidad de respuesta no es la adecuada, sobre todo en sistemas que usan un gran volumen de datos o críticos.
- Si se produce un fallo de proveedor, surgen dificultades para poder reestablecer el servicio, migrarlo a otro proveedor o establecerlo en local.
- **Escalabilidad a largo plazo.** Según vaya aumentando el número de usuarios que comiencen a compartir la infraestructura de la nube, irá aumentando la sobrecarga de los servidores de los proveedores, lo que puede provocar degradaciones en el servicio si la empresa no tiene un esquema de crecimiento óptimo.

Arquitectura

Cloud Computing se divide en capas, que van desde dispositivos hardware hasta sistemas software. Los recursos que tiene la nube se aprovechan para **proporcionar la potencia computacional necesaria para poder ofrecer los servicios**. La infraestructura de la nube puede ser heterogénea debido a la variedad de recursos que la constituyen. Los sistemas de base de datos y otros servicios de almacenamiento pueden ser parte de la infraestructura también.

La infraestructura física está gestionada por **el middleware central**, que tiene como objetivo proporcionar un entorno de ejecución apropiado para las aplicaciones y el mejor uso de los recursos. En la parte más baja, las tecnologías de virtualización son usadas para personalizar el entorno de ejecución, el aislamiento de aplicaciones, la calidad del servicio y el entorno de seguridad. Este middleware soporta capacidades como la negociación del servicio, el control de admisión, la gestión y supervisión de la ejecución, la contabilidad y la facturación.

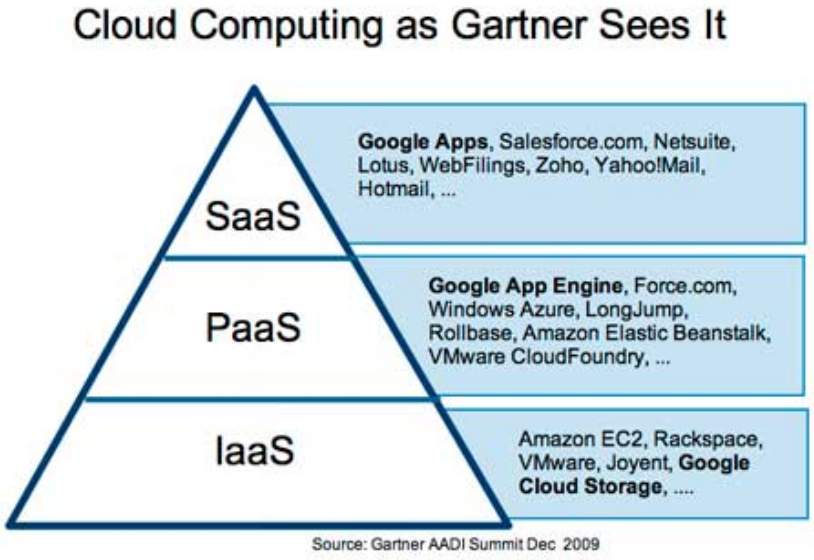

Figura 3.2: Arquitectura de computación en la nube (Fuente: http://timesofcloud.com)

La arquitectura se divide en las siguientes capas [VVE13](ver Figura 3.2):

- **Sofware como Servicio (SAAS).** Está en la capa más alta y **es donde una aplicación se encuentra alojada como un servicio** en el que acceden los clientes mediante Internet. Cuando el software se encuentra alojado, el cliente no tiene por qué mantenerlo ni darle soporte. Pero se encuentra fuera de las manos del cliente cuando el servicio alojado decide cambiarse. La idea de esta capa consiste en que se pueda usar el software como si fuera una caja y no es necesario hacer muchos cambios ni se requiere la integración a otros sistemas. El proveedor del servicio es el que se encarga de aplicar todos los parches y actualizaciones y de mantener la infraestructura en funcionamiento.

  Los costos de acceso al software pueden ser continuos, lo que significa que en vez de pagar por ello una única vez, cuanto más se utilice, mayor será el costo. Pero no se tiene que pagar tanto previamente y sólo hay que pagar por el uso de la aplicación.

  Para los proveedores, esta capa ofrece una mayor protección de su propiedad intelectual y la creación de un flujo continuo de ingresos.

  Se prestan muchos tipos de software al modelo de esta capa. El software que se encarga de llevar a cabo una tarea sencilla sin demasiada necesidad de interactuar con otros sistemas son los candidatos ideales para esta capa. Los clientes que no se van a encargar en el desarrollo del software, pero necesitan usar aplicaciones de alta potencia también pueden beneficiarse de esta capa. Algunas de las aplicaciones incluyen gestión de recursos de clientes, videoconferencia, gestión de los servicios TI, contabilidad, analista de la red y gestión de contenidos de la web.

  Las aplicaciones que se encuentran en esta capa se desarrollaron específicamente para utilizar herramientas web. También permite que varios clientes puedan usar una aplicación. Esta capa proporciona acceso mediante el uso de la red a software comercial que se encuentre disponible. Los clientes pueden acceder a sus aplicaciones desde cualquier lugar que tenga acceso a *Intenet* debido a que el software se gestiona en una ubicación central.

  A menudo se suele usar junto con otro software, usándose como un componente de otra aplicación. A esto se le conoce como *plugin* o *mashup*.

  Entre los beneficios de esta capa, se encuentran:

  - **Es más barato que comprar la aplicación de manera directa.** El proveedor de servicios ofrece aplicaciones que cuestan menos dinero y más confiables que las organizaciones por sí solas.
  - **Familiaridad con la Web.** La mayor parte de los usuarios tienen acceso a un ordenador y saben cómo usarlo en la Web, por lo que la curva de aprendizaje para la utilización de las aplicaciones externas puede ser menor.

- **Es menor el personal necesario.** La capacidad de sacar provecho de estas aplicaciones reduce la necesidad de mucho personal de las TI.
- **Personalización.** Las aplicaciones que se encuentran en esta capa son más sencillas de personalizar y pueden proporcionar una organización exacta de los que se quiere, al contrario de las aplicaciones antiguas que eran difíciles de personalizar y era necesario ajustarlas con el código.
- **Mejor comercialización.** Con la ayuda de esta capa, las aplicaciones que proporcionan los proveedores pueden ser usadas por mucha gente. De esta forma, se evita que un proveedor desarrolle una aplicación para un mercado muy estrecho que le puede crear problemas para comercializarse.
- **Confiabilidad de la Web.**
- **Seguridad.** Es usado ampliamente la capa de socket seguros(SSL), la cual proporciona confianza. Esto permite que los clientes puedan acceder a sus aplicaciones de manera segura sin tener que aplicar configuraciones complejas de *backend*.
- **Mayor ancho de banda.** El ancho de banda ha ido aumentando con el paso del tiempo y la calidad de las mejoras del servicio está ayudando al flujo de datos, lo que permite que las organizaciones puedan confiar en que pueden acceder a sus aplicaciones con buenas velocidades y baja latencia.

Esta capa **se enfrenta a obstáculos para su implementación y su uso**. El primer obstáculo es que una organización que tiene una necesidad computacional muy específica podría no encontrar la aplicación disponible en la capa. En casos como estos, la organización puede que necesite comprar un software e instalarlo en sus máquinas locales. Las empresas que tengan necesidades únicas pueden encontrar algunos de los componentes en una SAAS.

También existe un elemento de bloqueo con los proveedores. Esto significa que el cliente puede pagar a un proveedor para poder utilizar una aplicación, pero esta **no puede ser trasladada a un nuevo proveedor**. Se podrá hacer eso, pero habría que pagar que pagar al antiguo proveedor una tarifa para poder realizar el traslado.

También se enfrenta a retos de disponibilidad de aplicaciones de código abierto y hardware más barato. Las empresas pueden poner sus aplicaciones de código abierto en el hardware de mejor funciona y cuesta menos de lo que solía.

- **Plataforma como servicio (PAAS).** Es la capa que **se encarga de proporcionar todos los recursos necesarios para poder construir aplicaciones y servicios** de forma completa desde Internet, sin necesidad de descargar e instalar software.

  Los servicios que ofrece esta capa incluyen diseño, desarrollo, pruebas, despliegue y alojamiento de aplicaciones, colaboración de equipo, integración de servicio web,

integración de base de datos, seguridad, escalabilidad, almacenamiento, gestión de estado y control de versiones.

Una caída de esta capa es **la falta de interoperabilidad y portabilidad entre los proveedores**. Esto significa que, si se desarrolla una aplicación con un proveedor y se decide trasladar a otro, puede que eso no se pueda realizar o se tiene que pagar un precio alto para poder hacerlo. Además, si el proveedor deja de operar, las aplicaciones y los datos se perderán.

PAAS ofrece algún soporte para ayudar a la creación de interfaces de usuario, que normalmente están basadas en *HTML* o *JavaScript*

Esta capa está diseñada para que pueda ser usada por muchas personas a la vez y proporciona facilidades automáticas para administrar de simultaneidad, escalabilidad, conmutación por error y seguridad.

PAAS soporta interfaces de desarrollo web como *SOAP* y *REST*, las cuales **permiten construir varios servicios web**, algunas veces llamados *mashups*.

PAAS se encuentra en uno de los tipos de sistemas que se muestran a continuación:

- **Instalaciones de desarrollo adicionales.** Permiten que las aplicaciones PAAS puedan ser personalizadas. Normalmente, los desarrolladores y usuarios de PAAS están obligados a suscribirse a una aplicación PAAS complementaria.
- **Entornos independientes.** Estos entornos no incluyen dependencias de licencias, técnicas o financieras de aplicaciones PAAS específicas. Este entorno se usa para desarrollos generales.
- **Entornos de entrega de aplicaciones.** Estos entornos soportan servicios que se encuentran en el nivel de alojamiento.

En esta capa existen **dos obstáculos principales para los desarrolladores**. Algunos desarrolladores tienen miedo a estar encerrados en un solo proveedor debido a que este usa servicios o lenguajes de desarrollo de su propiedad. El proveedor permite que la solicitud se lleve a un proveedor distinto, pero esto conlleva unos altos costes.

- **Infraestructura como Servicio (IAAS).** Es la capa que se encarga de proporcionar el hardware a los clientes para que puedan poner lo que quieran en él. El proveedor de esta capa alquila los recursos, evitando de esta forma que los clientes tengan que comprar estos.

  Los recursos que se pueden alquilar son los siguientes:

  - Espacio del servidor.
  - Memoria.
  - Ciclos de la unidad central de procesamiento(CPU).

- Espacio de almacenamiento.

Además, **la infraestructura puede ser escalada dinámicamente** dependiendo de las necesidades de los recursos de la aplicación. Varios clientes se pueden encontrar en el mismo equipo en el mismo momento. Los proveedores cobran por los recursos que se consumen.

Esta capa está formada por varias partes:

- **Acuerdos a nivel de servicio.** Es un acuerdo entre el proveedor del servicio y el cliente, lo que garantiza un nivel de rendimiento del sistema.
- **Hardware de un ordenador.** Estos son los componentes que serán alquilados.
- **Red.**
- **Conectividad a Internet.** Esto permite que los clientes puedan acceder al hardware correspondiente a sus organizaciones.
- **Entorno de virtualización de plataformas.** Permite ejecutar las máquinas virtuales que los clientes deseen.
- **Facturación de las utilidades computacionales.** Está configurado para cobrar a los clientes por el número de recursos que utilizan.

Proporciona una infraestructura que **es personalizable bajo demanda**. Las opciones que se encuentran disponibles en esta capa son variadas, desde servidores individuales hasta infraestructuras completas, donde se incluyen los dispositivos de red, equilibradores de carga y servidores de bases de datos y Web.

La tecnología principal que es usada para entregar e implementar las soluciones que proporciona esta capa es **la virtualización del hardware**, que consiste en que una o más máquinas virtuales que están configuradas e interconectadas forman un sistema distribuido en el que se instala e implementan las aplicaciones. Las máquinas virtuales forman los componentes atómicos que se implementan y tienen un precio según las características específicas del hardware virtual.

Esta capa goza de todas las ventajas de la virtualización del hardware, como es la partición de la carga de trabajo, el aislamiento de aplicaciones, el *sandboxing* y el ajuste de hardware. Las ventajas que proporciona esta capa para el proveedor de servicios son que permite **una mejor explotación de la infraestructura de las TI y otorga un entorno más seguro para la ejecución de aplicaciones de terceros**. En cuanto al cliente, **reduce los costos de administración, mantenimiento y los que se asignan para la compra de hardware**.

Los usuarios aprovechar la personalización que proporciona la virtualización para desplegar su arquitectura en la nube.

En la figura 3.3 se puede ver el tipo de usuarios que usa cada capa de la arquitectura y en la figura 3.4 se muestra las responsabilidades que tiene cada capa.

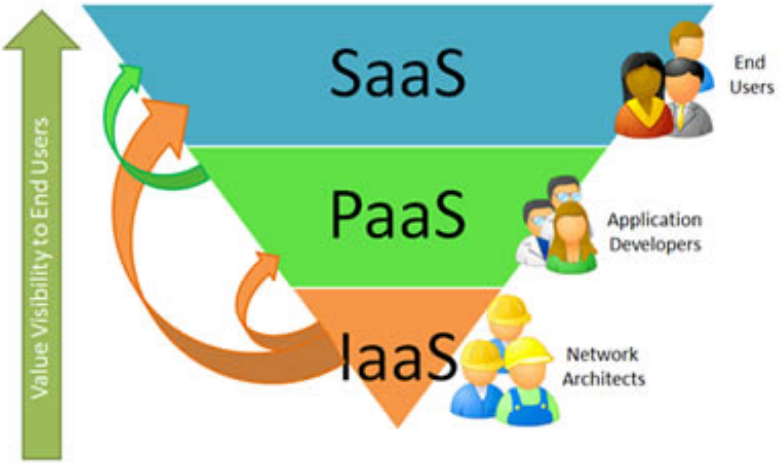

Figura 3.3: Usuarios correspondientes a cada capa de la arquitectura (Fuente: itsteziutlan)

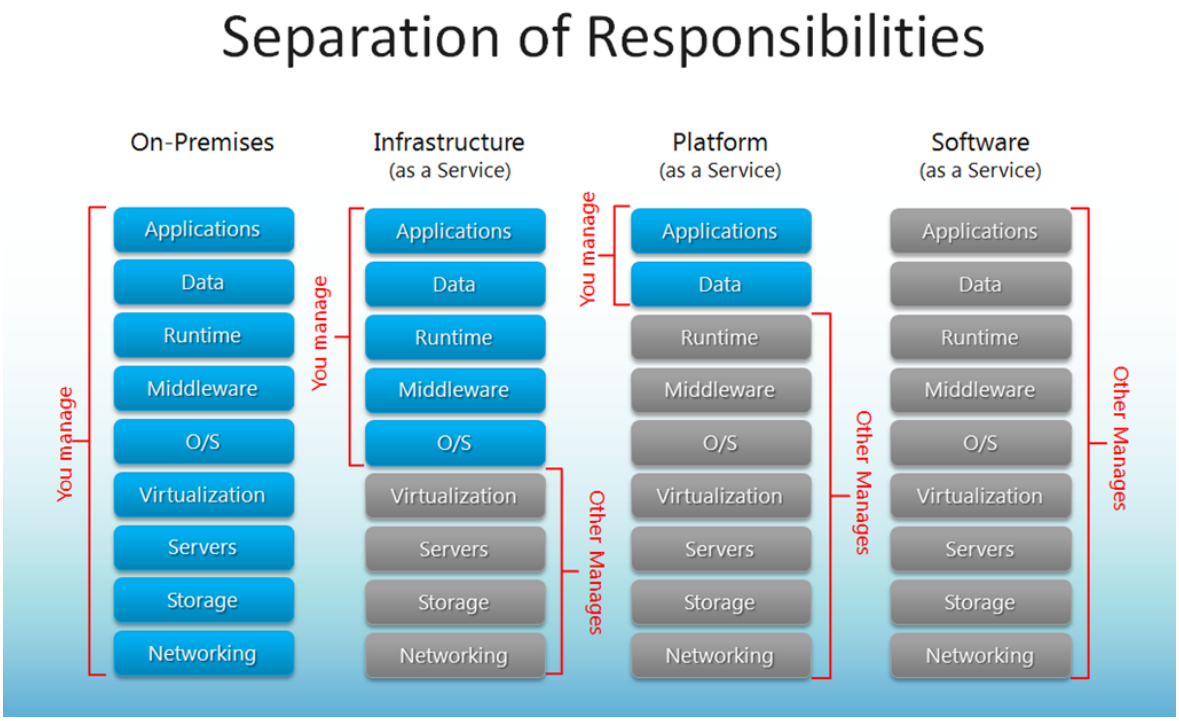

Figura 3.4: Responsabilidades de cada capa de la arquitectura de la computación en la nube (Fuente: https://kkanakas.com/category/cloud/)

Virtualización

Es uno de los componentes fundamentales de la computación en la nube [BVS09]. Permite **la creación de un entorno abstracto de ejecución seguro, personalizable y aislado para aplicaciones que se encuentren en ejecución**, sin que afecte a las aplicaciones de otros usuarios(ver Figura 3.5). La base de esta tecnología es la capacidad que tiene un programa de un computador de emular un entorno de ejecución diferente del que aloja esos programas. La virtualización **permite construir sistemas elásticamente escalables que pueden proveer capacidad adicional con mínimos costes**. No sólo proporciona el entorno para la ejecución de aplicaciones, sino también para el almacenamiento, memoria y redes. El entorno virtualizado tiene tres elementos principales: **la capa de virtualización**, que es la responsable de recrear el mismo o un entorno distinto donde el invitado operará; **el invitado**, que representa el elemento que interactúa con la capa de virtualización; y **el *host***, que representa el entorno original donde el invitado es administrado. El entorno virtual es creado mediante un programa software.

Las características de los entornos virtualizados son las siguientes:

- **Seguridad incrementada.** La capacidad de poder controlar la ejecución de un invitado de manera transparente abre nuevas posibilidades para proporcionar un entorno seguro y controlado. La máquina virtual representa un entorno emulado donde se ejecuta el invitado. Todas las operaciones que realiza el invitado se hacen contra la máquina virtual, que las traduce y las aplica al *host*. Este nivel de indirección permite que el administrador de la máquina pueda controlar y filtrar la actividad del invitado, evitando de esta manera que se realicen operaciones dañinas. Los recursos expuestos por el *host* pueden ser ocultos o protegidos del invitado. La información sensible que contiene el *host* puede ocultarse sin tener que instalar políticas complejas de seguridad.
- **Ejecución gestionada.** El uso compartido, la agregación, la emulación y el aislamiento son las características más importantes de la virtualización del entorno.
  - **Uso compartido.** La virtualización permite la creación de entorno de computación separados en el mismo *host*, haciendo que se puedan explotar las capacidades de un invitado de gran alcance. Esta característica se usa para reducir el número de servidores activos y limitar el consumo de energía.
  - **Agregación.** Un conjunto de *hosts* separados se puede representar como un único *host* virtual.
  - **Agregación.** Los programas invitados son ejecutados en un entorno que está controlado por la capa de virtualización, que es un programa, permitiendo de esta forma controlar y ajustar el entorno que está expuesto a los invitados. Está característica es muy útil para los propósitos de prueba, donde un invitado determinado tiene que ser validado en distintas plataformas o arquitecturas y la gran variedad de opciones no es accesible fácilmente durante el desarrollo.

- **Aislamiento.** Permite proporcionar a los invitados un entorno completamente separado en el que se ejecutan. El invitado realiza su actividad interactuando con una capa de abstracción, que va a proporcionar acceso a los recursos subyacentes. Esta característica tiene varias ventajas, como permitir que varios invitados se ejecuten en el mismo *host* sin que interfieran unos con otros o proporcionar una separación entre el invitado y el *host*.

Además de las características anteriores, otra de ellas es **la optimización del rendimiento**. Se facilita el control del rendimiento del invitado ajustando con precisión las propiedades de los recursos expuestos mediante el entorno virtual. Esta capacidad proporciona un medio para implementar de manera eficaz una infraestructura de calidad de servicio que cumpla de manera más sencilla el acuerdo de nivel de servicio establecido por el invitado. Otra ventaja de la ejecución gestionada es que permite **capturar sencillamente el estado del programa invitado, mantenerlo y reanudar su ejecución**.

- **Portabilidad.** Esta se aplica de distintas maneras dependiendo del tipo de virtualización. Cuando la virtualización es del hardware, el invitado se empaqueta en una imagen virtual que se puede mover y ejecutar en diferentes máquinas con seguridad. Cuando la virtualización es a nivel de programación, el código que representa los componentes de la aplicación puede ejecutarse sin tener que volver a compilar en ninguna implementación de la máquina virtual correspondiente. Esta característica permite tener un sistema propio siempre y que esté listo para usarse cuando el administrador de la máquina virtual está disponible.

El uso de la virtualización en la computación en la nube tiene algunas ventajas. Las ventajas más importantes son **la ejecución administrada y el aislamiento**, porque permiten construir entorno de computación seguros y controlables. Un entorno de ejecución virtual puede ser configurado como un **entorno limitado**, evitando de esta forma que cualquier operación perjudicial entre en el *host* virtual. También se simplifica la asignación de recursos y la partición entre los distintos invitados a través del control del *host* virtual mediante un programa. Otra de las ventajas en la portabilidad. Las instancias de la máquina virtual están representadas por uno o más archivos que pueden ser transportados de manera sencilla. Suelen ser autónomos debido a que no tienen dependencias. La portabilidad y la autocontención ayuda a disminuir los costes de mantenimiento, debido a que el número de *hosts* será menor que el número de instancias de máquinas virtuales. Mediante el uso de la virtualización se puede conseguir un uso más eficiente de los recursos. Varios sistemas pueden existir juntos de manera segura y compartir recursos del *host* subyacente.

La virtualización también tiene sus desventajas y la más evidente es **la disminución del rendimiento de los sistemas de invitados** como resultado de la intermediación realizada por la capa de virtualización, que puede hacer que el invitado experimente un aumento de

la latencia. Además, el uso subóptimo del *host* debido a la capa de abstracción introducida mediante el software de administración de virtualización puede conducir a un uso muy ineficiente del *host* o una mala experiencia del usuario. Las desventajas más peligrosas son **las implicaciones para la seguridad**, que se deben a la capacidad de emular un entorno de ejecución distinto.

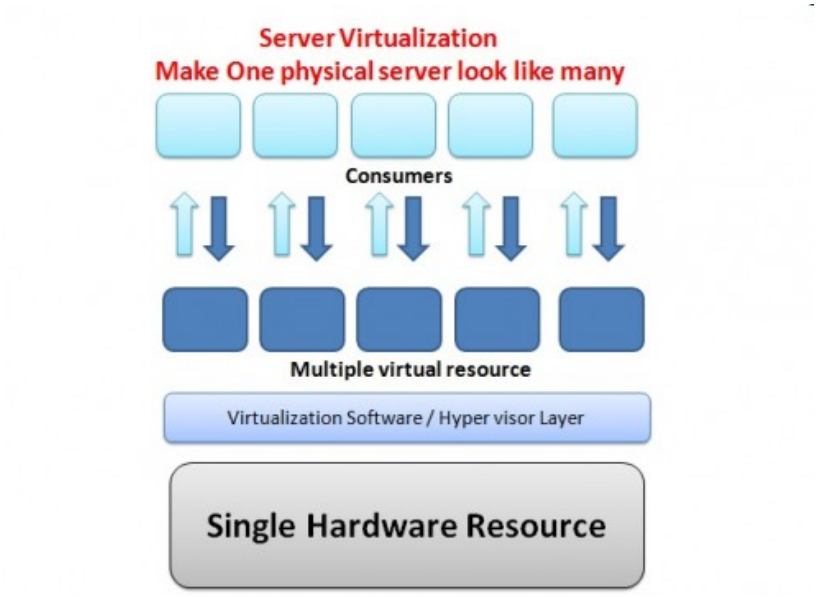

Figura 3.5: Virtualización(Fuente: http://www.diceitwise.com/how-does-cloud-computing-work-and-technology-behind-it/)

Tipos

Según las necesidades que tengan las empresas, el modelo de servicio que se ofrece y como se desplieguen en las mismas, se usan distintos tipos de nubes [BVS09](ver Figura 3.6).

- **Nubes públicas.** En este tipo de nubes **los servicios ofrecidos se ponen a disposición de cualquier persona**, desde cualquier sitio y en cualquier momento mediante el acceso a Internet. **Son manejadas por terceras partes y los trabajos de distintos clientes pueden estar mezclados en los servidores**. Desde el punto de vista estructural, es **un sistema distribuido formado por uno o más centro de datos que están conectados entre sí**, en los que se encuentra los servicios ofrecidos por la nube. Cualquier usuario puede iniciar sesión de manera sencilla con el proveedor de la nube, introducir sus credenciales y su configuración de facturación y usar los servicios de los que dispone. Ofrecen soluciones que disminuyen al máximo los costos de la infraestructura de las TI y es una opción viable para poder manejar las cargas máximas en la infraestructura local. Es una opción interesante para las pequeñas empresas, ya que permiten iniciar los negocios sin realizar grandes inversiones iniciales basándose en la infraestructura pública para sus necesidades de las TI. Lo que hizo que este tipo de nubes fueran

atractivas fue **la capacidad de crecer o reducir de acuerdo con las necesidades de negocio relacionadas**. En la actualidad, estas nubes se usan tanto para reemplazar la infraestructura de las TI de la empresa completamente como para extenderla cuando sea necesario.

La característica fundamental de estas nubes es su **multiplicidad**, debido a que sirve a varios usuarios, no a un único cliente. Cualquier cliente necesita un entorno de computación virtual que está separado y aislado de otros usuarios. Una parte de la infraestructura software se encarga de monitorear los recursos de las nubes, facturar de acuerdo al contrato hecho con el usuario y mantener un historial completo del uso de la nube para cada cliente.

Este tipo de nube puede ofrecer cualquier tipo de servicio: **infraestructura, plataforma o aplicaciones**. Lo que diferencia este tipo de nube de las demás es la manera en la que se consumen: están disponibles para todo el mundo y están diseñadas para soportar gran cantidad de usuarios. Las caracteriza **su capacidad de escalar dependiendo de la demanda que tengan y de sostener las cargas máximas**.

La seguridad y la carga operacional de los datos recae sobre el proveedor del hardware y software, haciendo que el riesgo por la adopción de nueva tecnología sea bastante bajo.

- **Nubes privadas.** Es un tipo de nubes similar a las públicas, pero su modelo de provisión de recursos se encuentra **limitado dentro de los límites de una organización**. No suele ofrecer servicios a terceros, solo a los miembros de la organización. Son sistemas distribuidos virtuales que dependen de una infraestructura privada y proporcionan a los usuarios internos un aprovisamiento dinámico de recursos informáticos. Estas nubes tienen la ventaja de **mantener operaciones empresariales básicas basándose en la infraestructura de las TI existentes y reducir la carga de mantenimiento** una vez que se ha configurado la nube. En estas nubes, las preocupaciones sobre la seguridad son menos críticas, debido a que la información confidencial no fluye de la infraestructura privada. Además, los recursos de las TI pueden ser mejor usados porque estas nubes proporcionan a una distinta clase de usuarios.

  Con este tipo de nubes el usuario puede probar las aplicaciones y los sistemas a un precio más bajo que en las nubes públicas antes de desplegarlas en su infraestructura virtual. Desde el punto de vista arquitectónico, las nubes privadas pueden implementar un hardware más heterogéneo. Estas suelen basarse en la infraestructura de las TI existente que se encuentra previamente instalada en las instalaciones privadas.

  Estas nubes proporcionan soluciones internas para la computación en la nube, pero tiene una capacidad más limitada para escalar elásticamente a las demandas. La desventaja que tiene este tipo de nubes es que **supone una gran inversión inicial**.

- **Nubes híbridas.** Para aprovechar lo mejor de las nubes públicas y privadas, surgió este tipo de nube. Estas nubes permiten a las empresas explotar las infraestructuras de las TI existentes, mantener la información privada en el interior de las instalaciones y añadir o reducir el aprovisionamiento de recursos externos y liberarlos una vez que ya no se necesiten. Las preocupaciones de seguridad se limitan solamente a la parte pública de la nube que se puede usar para hacer operaciones que tengan restricciones menos estrictas pero que siguen formando parte de la carga de trabajo del sistema. Este tipo de nubes son **un sistema distribuido heterogéneo formado por una nube privada que contiene servicios o recursos adicionales de una o más nubes públicas**. Este tipo de nubes consisten en la combinación de las aplicaciones locales con las de la nube pública.

  Las nubes híbridas abordan la escalabilidad haciendo uso de los recursos externos para superar la capacidad de demanda. Estos recursos o servicios se alquilan temporalmente según el tiempo necesario y luego se liberan. Esto se le conoce como **cloudbursting**. El aprovisamiento dinámico introduce un algoritmo y políticas de programación más difíciles, cuyo objetivo es optimizar el presupuesto gastado en recursos públicos.

  Estas nubes tienen la ventaja de **hacer que la inversión inicial sea más moderada y contar con SaaS, PaaS o IaaS cuando se requiera**.

- **Nubes comunitarias.** Son **sistemas distribuidos creados mediante la integración de los servicios de distintas nubes** para satisfacer las necesidades específicas de una industria, comunidad o sector empresarial. Los usuarios de este tipo de nubes se encuentran en una comunidad bien identificada y comparten las mismas preocupaciones o necesidades. Pueden ser organizaciones gubernamentales, industrias o simples usuarios, pero todos ellos se centran en los mismos problemas cuando interactúan con la nube.

  Desde el punto de vista arquitectónico, estas nubes están implementadas en **varios dominios administrativos**. Lo que quiere decir que distintas organizaciones contribuyen con sus recursos para construir la arquitectura de la nube.

  Este tipo de nubes se forma sacando provecho de los recursos subutilizados de las máquinas del usuario y proporcionando una infraestructura en la que cada usuario puede ser al mismo tiempo un consumidor, un productor o un coordinador de los servicios que ofrece la nube.

  Los beneficios que proporcionan estas nubes son los siguientes:

  - **Sinceridad.** Al eliminar la dependencia con los proveedores de la nube, las nubes son sistemas abiertos en los que puede producirse una competencia leal entre distintas soluciones.

- **Comunidad.** Se basa en un colectivo que proporciona recursos y servicios. La infraestructura en más escalable porque el sistema crece simplemente expandiendo su base de usuarios.
- **Conveniencia y control.** En el interior de la nube no existe conflicto entre la conveniencia y control porque la nube es compartida y propiedad de la comunidad, en la cual las decisiones son tomadas mediante un proceso democrático colectivo.
- **Sostenibilidad ambiental.** Aprovecha todos los recursos subutilizados.

Este tipo de nube es suministrada para **el uso exclusivo de una específica comunidad de consumidores de organizaciones que tienen preocupaciones compartidas**. Sirven para que varias organizaciones que comparten negocios, servicios y objetivos, puedan también compartir sus recursos de computación y tecnológicos. Puede ser poseído, gestionado y operado por una o más de las organizaciones que se encuentran en la comunidad, un tercero o una combinación de estos. Este tipo de nube tiene menor número de usuarios que una nube pública y ofrece mayores niveles de privacidad y seguridad.

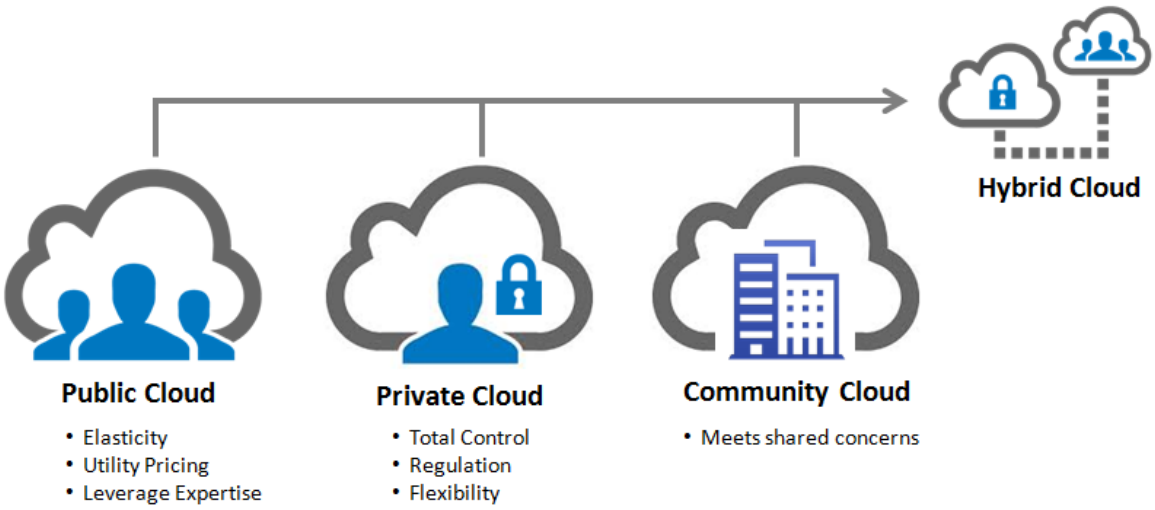

Figura 3.6: Tipos de nube(Fuente: http://www.keywordsuggests.com)

Actores

Existen cinco actores principales que pueden ser una entidad, una organización o una persona que se encarga de formar parte de una transacción o proceso y realizar tareas en la computación en la nube [LTM+11](ver Figura 3.7).

- **Consumidor de la nube.** Es el actor principal para el servicio de computación en nube y puede ser una persona o una organización que mantiene una relación de negocios y usa los servicios que proporciona el proveedor de la nube. Este actor mira el conjunto

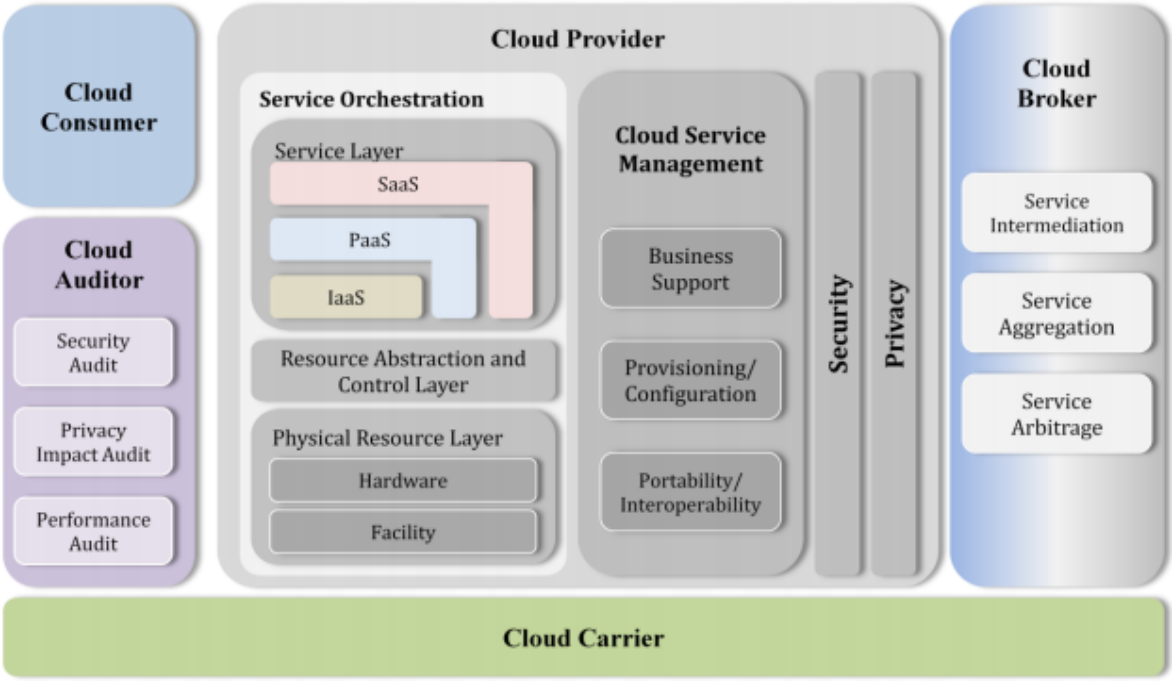

Figura 3.7: Modelo conceptual de los actores que realizan las tareas de *cloud computing*(Fuente: National Institute of Standards and Technology)

de servicios del proveedor de la nube, solicita el servicio que sea más apropiado para su negocio, establece los contratos de servicios con el proveedor de la nube y usa el servicio que ha contratado. El consumidor puede tener que pagar por el servicio que se le proporciona.

- **Proveedor de la nube.** Este tipo de actor puede ser una persona o una organización que se encarga de poner los servicios a disposición de las partes que están interesadas en ellos. El proveedor obtiene y gestiona la infraestructura informática necesaria para proporcionar los servicios, ejecuta el software que hace que los servicios estén disponibles en la nube para ser usados y realiza las modificaciones necesarias para proporcionar los servicios a los consumidores a través de la red.

  Se pueden distinguir cinco áreas para las actividades que realiza el proveedor en la nube: Despliegue y orquestación de los servicios, administración de los distintos servicios que se encuentran en la nube, seguridad y privacidad.

- **Auditor de la nube.** Es el encargado de realizar un examen a los servicios en la nube con el fin de expresar su opinión. La auditoría consiste en la revisión de evidencias objetivas para verificar si se cumple con los estándares establecidos.

- **Intermediario de la nube.** Es una entidad que se encarga de administrar el uso, el desempeño y la entrega de servicios que se encuentran en la nube y lleva a cabo una negociación de las relaciones entre los proveedores y los consumidores. Esto hace que

los consumidores no tengan que contactar directamente con el proveedor de la nube.

El intermediario proporciona varios servicios:

- **Intermediación del servicio.** Destaca un servicio determinado proporcionándole una mejora en alguna función específica y proporcionando servicios que tengan un valor agregado para los consumidores.
- **Agregación del servicio.** Combina e integra varios servicios en uno o más servicios nuevos. Este proporciona la integración de los datos y garantiza que el movimiento de los datos entre los consumidores y múltiples proveedores sea seguro.
- **Arbitraje de servicios.** Es similar a la acción anterior pero el intermediario tiene la flexibilidad para elegir los servicios desde distintas agencias, lo que hace que los servicios que se agregan no son fijos.

- **Operador de la nube.** Es un intermediario que se encarga de garantizar la conectividad y transporte de los servicios entre consumidores y proveedores.

Proveedores de servicios en la nube

Existen varios proveedores que proporcionan servicios en la nube. Las cosas que estos ofrecen varían dependiendo del proveedor y sus modelos de precios. Entre todos los proveedores disponibles, los más conocidos y los que comenzaron con a usar esta tecnología son los siguientes [VVE13]:

- **Amazon**[1]**.** Es uno de los primeros proveedores que ofrece servicios en la nube y son muy sofisticados. Entre los servicios que ofrece, se encuentran:
  - **Elastic Compute Cloud.** Ofrece máquinas virtuales y ciclos de CPU extras a las organizaciones.
  - **Simple Storage Service.** Permite almacenar fichero que ocupen hasta 5 GB en el servicio de almacenamiento virtual de Amazon.
  - **Simple Queue Service.** Permite que las máquinas del cliente puedan comunicarse mediante el uso de esta Interfaz de Programación de Aplicaciones(API) de paso de mensajes.

  Estos servicios pueden llegar a ser difíciles de ejecutarse porque se tienen que realizar mediante la línea de comandos.

  Las máquinas virtuales que usa Amazon son versiones de distribuciones de *Linux*, por lo que el cliente puede implementar las aplicaciones en su máquina y luego subirlas a la nube.

[1] https://aws.amazon.com/es/

- **Google.** Google ofrece documentos en línea y hojas de cálculo y anima a los desarrolladores a crear funciones para esos y otros programas que estén en línea mediante *Google App Engine*[2], herramienta que se va a ver más a fondo en la sección 3.3.1. *Google* disminuyó las aplicaciones web a un conjunto básico de características y creo un *framework* para poder entregarlas. Google también proporciona funciones para la depuración.
- **Microsoft.** La solución de computación en la nube de Microsoft se denomina *Microsoft Azure*[3], anteriormente denominado *Windows Azure*, del cual se va a hablar en la sección 3.3.3. Este en un sistema operativo que permite que las organizaciones puedan ejecutar aplicaciones de *Windows* y almacenar archivos y datos haciendo uso de los centros de datos de *Microsoft*. También proporciona su plataforma de servicios *Azure*, cuyos servicios permiten que los desarrolladores establezcan identidades de usuarios, administren flujos de trabajo, sincronicen datos y lleven a cabo otras funciones según vayan construyendo programas en la plataforma de computación en línea de *Microsoft*. Entre los servicios que proporciona la plataforma de servicios de *Azure* se encuentran:
  - **Microsoft Azure.** Proporciona alojamiento y administración de servicios y almacenamiento, computación y redes de bajo nivel de escalamiento.
  - **Microsoft SQL services.** Proporciona servicios y reportes de base de datos.
  - **Microsoft .NET services.** Proporciona implementaciones basadas en servicio de conceptos del *framework* de *.NET* como flujo de trabajo.
  - **Live services.** Permite compartir, almacenar y sincronizar documentos, foto y archivos en distintos dispositivos.
  - **Microsoft Office services.** Permite que los usuarios puedan leer y modificar documentos en línea.

Figura 3.8: Proveedores de servicios en la nube

[2]https://cloud.google.com/appengine/
[3]https://azure.microsoft.com

MMOs y rankings masivos

Applibot

Con el paso de los años, los videojuegos han ido teniendo mayor importancia en la sociedad. Cada vez más gente dedica algo de tiempo a estos. Con el éxito que tuvieron los videojuegos, se decidió crear un tipo de videojuegos que consiste en jugar con muchas personas en una misma partida mediante la conexión a Internet, permitiendo que los jugadores pudieran cooperar o competir entre ellos. Estos videojuego se denominan MMO y se habla sobre ellos con mayor profundidad en la sección 1.1. Estos hacen uso de un ranking, donde la posición que ocupa cada jugador en él depende de la puntuación que tenga. Esa puntuación es actualizada con algunas acciones realizadas por el jugador, lo que provoca una gran ralentización al haber tantos jugadores en la misma partida. Además existen videojuegos en los que en todo momento hay una gran cantidad de jugadores usándolo, como es el caso de uno de los videojuegos de Applibot[4], por que se ha tenido que enfrentar a este problema, como se menciona en la sección 1.2.

Applibot es uno de los principales desarrolladores de aplicaciones sociales en Japón. Fue fundada el 7 de Julio de 2010 y se encargan de **la planificación, producción y operación de una amplia gama de servicios** para *smartphones* y otros dispositivos de próxima generación. La singularidad de esta empresa es su gran experiencia y conocimiento en la construcción de los servicios de juegos sociales mediante **Google App Engine**, la plataforma como servicio que ofrece *Google*. Aprovechando el potencial de esta plataforma, *Applibot* ha tenido mucho éxito al capturar oportunidades de negocio en el mercado de los juegos de tipo social en Japón y en Estados Unidos.

Legend of the Cryptids, uno de sus grandes títulos, alcanzó el puesto número uno en la categoría de juegos de *Apple AppStore* de Norteamérica en octubre de 2012. Este es un juego multijugador masivo en línea de cartas coleccionables. Este videojuego fue lanzado en 2012 para móviles con sistema operativo *iOS* o *Android*. El juego se desarrolla en un mundo ficticio llamado *Neotellus*, un planeta en órbita por tres lunas en una galaxia distante. El planeta está formado por humanos y unas criaturas conocidas como *Cryptids*. El juego gira entre el conflicto entre los seres humanos y los *cryptid*.

La serie *Legend* registró 4,7 millones de descargas. Otro título, *Gang Road*, alcanzó el puesto número 1 en el ranking de ventas totales de *AppStore* de Japón en diciembre de 2012.

Como se ha indicado anteriormente, esta empresa se llegó a enfrentar a un problema relacionado con el que se va a tratar en este trabajo, **la clasificación de un jugador**, y la solución que va a aplicar, va a servir de ayuda para la realización de este proyecto. Debido a que les resultó complejo encontrar una solución óptima, decidieron pedir ayuda al *Técnico Gerente*

[4]https://www.applibot.co.jp

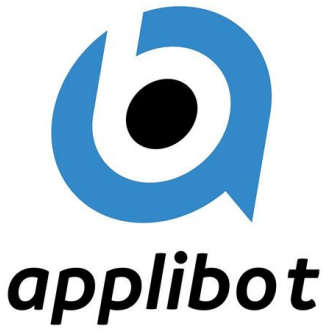

Figura 3.9: Logo de Applibot

de Cuentas(TAM) que *Google* les asignó para obtener una solución [FK16].

Se plantearon minimizar la complejidad de la solución. La solución simple que consiste en **escanear todos los usuarios que tienen una puntuación mayor para obtener el rango de un jugador** tiene una complejidad de tiempo de O(n), lo que significa que el tiempo que se necesita para ejecutar la consulta aumenta de forma proporcional al número de jugadores, por lo tanto este algoritmo no es escalable.

Entonces decidieron usar una implementación de código abierto de un algoritmo que encontraron que estaba basado en el árbol de clasificación para el *Datastore*: **la librería Google Code Jam Ranking**. Decidieron usar esta librería porque pensaron que los algoritmos de árboles pueden actuar con un complejidad de tiempo O(log n) para encontrar un elemento.

Sin embargo, cuando realizaron las pruebas de carga, encontraron una limitación crítica con esa librería. **Su escalabilidad en cuanto al rendimiento de la actualización era muy baja**. Esto ocurre por el costo de mantener la consistencia del árbol. La librería solo usa un grupo de entidades para mantener el árbol entero para asegurar la consistencia de los recuentos en los elementos del árbol.

Pero un grupo de entidades en el *Datastore* tiene una limitación de rendimiento. El *Datastore* sólo soporta una transacción por segundo en un grupo de entidades. La librería usada es muy coherente y transaccional, y bastante rápida, pero no funciona tan bien para una gran cantidad de actualizaciones simultáneas.

Entonces, se decidió **agregar lotes de actualizaciones en un sola operación**, en vez de ejecutar cada actualización como una transacción separada, para obtener un mayor rendimiento en la actualización de un grupo de entidades. Pero debido a que una transacción contiene un gran número de actualizaciones, **cada transacción tardaría más tiempo**, aumentando la posibilidad de que entrara en conflicto con transacciones simultáneas.

Por ello se aconsejó **el uso del patrón de diseño de agregación de tareas**, que consiste en la utilización de un solo hilo para llevar a cabo un lote de actualizaciones. Esta solución

no provoca fallos de transacción debido a las actualizaciones simultáneas porque sólo hay un hilo y sólo una transacción abierta en el grupo de entidades. Sin embargo, está el inconveniente de que sólo se usa un hilo para agregar todas las actualizaciones e **impone un límite a la rapidez en la que se aplican los cambios en el *Datastore***.

La solución planteada distingue tres componentes:

- **Frontend.** Acepta las peticiones de actualización de puntuación de los usuarios y las añade a una *pull queue*.
- **Pull queue.** Recibe y retiene las solicitudes de actualización de puntuación del *frontend*.
- **Backend.** Ejecuta un bucle infinito con un único hilo que obtiene las solicitudes de actualización de la cola y las ejecuta con la librería *Google Code Jam Ranking*, que va a tratas el conjunto de solicitudes como un lote en una sola transacción.

Esta solución funcionaba bien, pero se encontró el problema de que **el rendimiento de la *pull queue* fluctúa de vez en cuando**. Esto ocurre porque la *pull queue* depende de ***Bigtable*** como de su capa de persistencia. Cuando una tableta *Bigtablet* crece demasiado grande, se divide en diversas tabletas. Mientras se ha dividido la tableta, las tareas no se entregan, lo que provoca la fluctuación de rendimiento cuando la cola está recibiendo tareas a un ritmo rápido.

Entonces decidieron emplear **Queue Sharding**, que consiste en distribuir la carga en varias colas en vez de en una. Cada cola puede almacenar en un servidor una tableta *Bigtable* diferente para disminuir el efecto de una fracción de la tableta y mantener de esta manera una tasa de procesamiento de tarea alta.

Se utilizaron diez *pull queues* y el *backend* de estas, lo que realizaban en su bucle infinito era arrendar hasta 1000 tareas, conteniendo cada una el nombre del jugador y su nueva puntuación. Estos pares se van a almacenar en un diccionario para pasarlo como parámetro al método de la librería *Google Code Jam Ranking* que se va a encargar de aplicar los cambios de puntuación en el *Datastore*. Si no ocurre ningún error, se eliminan las tareas arrendadas. Si ocurre algún error o un cierre del bucle o de la instancia de *backend*, las tareas se mantienen en las colas de tareas, para que puedan procesarse cuando la instancia se reinicie.

Esta solución hace que se minimice de forma eficaz las fluctuaciones de rendimiento de las colas y puede sostener 300 actualizaciones por segundo durante varias horas, es decir, que es lo **suficientemente rápido**. También hace que el sistema sea escalable para que lo puedan manejar muchas personas y persistente y consistente, por lo que cumple con todos los requisitos originales de *Applibot*.

Sin embargo, todavía el rendimiento tiene una limitación, alrededor de 300 actualizaciones por segundo. Para aumentar la actual velocidad de actualización, **fragmentaron el árbol de**

clasificación en tres partes, haciendo que el *backend* de cada *pull queue* se de actualizar las puntuaciones que se encuentran en el mismo fragmento del árbol.

En general, no se necesita la coordinación de los fragmentos del árbol. Esto hace que se incremente tres veces el rendimiento de la actualización. Sin embargo, cuando se solicite el rango de un usuario, **se tiene que consultar todos los fragmentos para obtener el rango de una determinada puntuación, y luego sumar el rango obtenidos en todos los fragmentos** para obtener el rango real, lo que provoca que se tarde más tiempo en realizar esa operación. El tiempo de consulta de esta acción se podría reducir sustancialmente manteniendo las entidades en *Memcache*.

Para mejorar el tiempo que tarda en ejecutarse la solicitud encarga de obtener el rango del jugador, aplicaron un solución que consistía en **dividir el rango total de puntuaciones en buckets**. Cada *bucket* contiene un sub-rango de las puntuaciones y el número de jugadores que tienen una puntuación que se encuentra en ese rango(ver Figura 3.10). Mediante estos datos, se puede encontrar con una aproximación cercana el rango de cualquier puntuación. Los *buckets* son muy parecidos al nodo de nivel superior del árbol, pero en vez de descender a unos nodos más detallados, el algoritmo interpola dentro de un *bucket*.

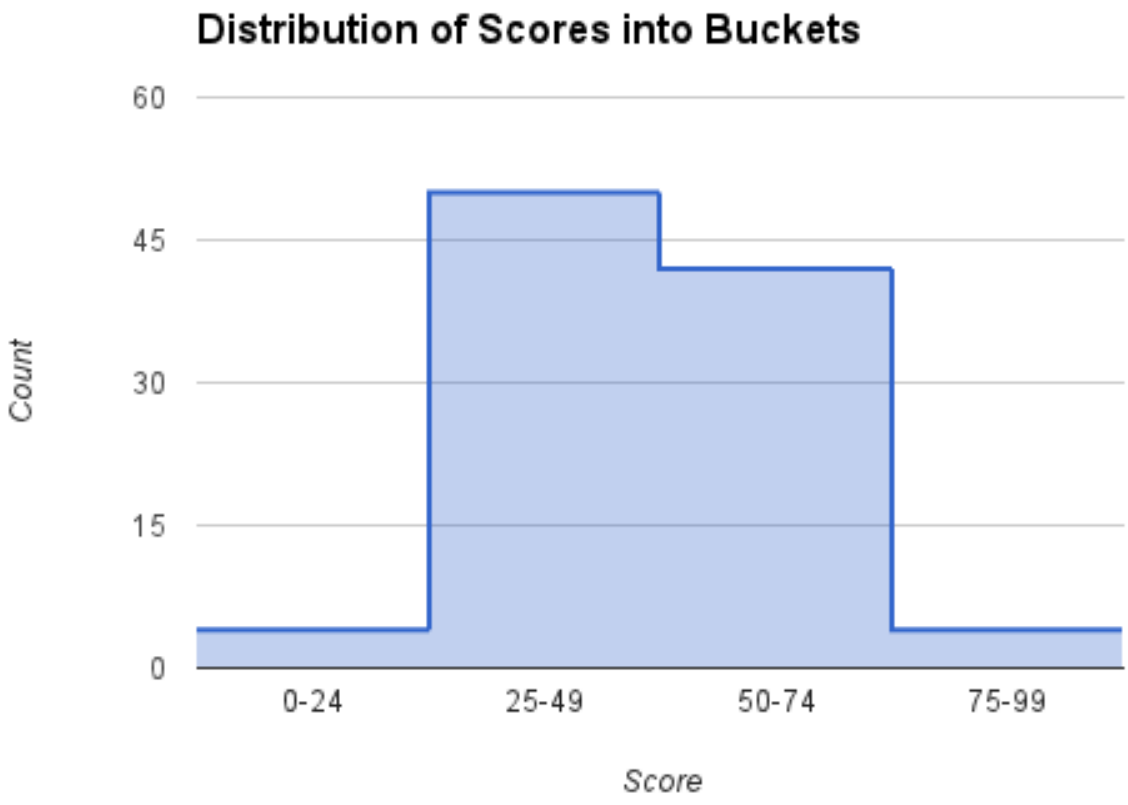

Figura 3.10: Distribución de los *buckets*(Fuente: Google)

Cuando se solicita el rango de un jugador, se encuentra el *bucket* apropiado mediante la puntuación del jugador. El *bucket* tiene incluido el límite de rango superior y el número

de jugadores que se encuentran dentro de él. Para estimar el rango de los jugadores dentro de los *buckets*, se usa **la interpolación lineal**. Mediante esta solución, el coste en realizar la solicitud de obtener el rango de un jugador no va a del número de jugadores que se encuentren en la clasificación.

Google Code Jam Ranking Library

Es una librería implementada por *Google* que **almacena la puntuación de los usuarios en una estructura de datos de árbol *N-ario***, de tal forma que cada nodo tendrá *N* hijos. Cada nodo del árbol tendrá *N* rangos de puntuaciones con el respectivo número de jugadores que tienen una puntuación que se encuentra dentro del rango. Cada hijo del nodo tendrá otros *N* rangos donde el mínimo y el máximo valor de todos los rangos van a ser el valor mínimo y el máximo de uno de los rangos que tiene el padre. De esta manera permite encontrar de manera más fácil la puntuación de cualquier jugador. Lo que se puede hacer con esta librería y que se va a utilizar en este trabajo es, como se ha dicho anteriormente, obtener los puntos de un usuario, actualizar la puntuación de un jugador y obtener el rango de un jugador a través de su puntuación. Esta librería la usa *Applibot* en la solución de su problema y también se utiliza en este trabajo para el tratamiento de los *rankings* de manera más sencilla [Goo09].

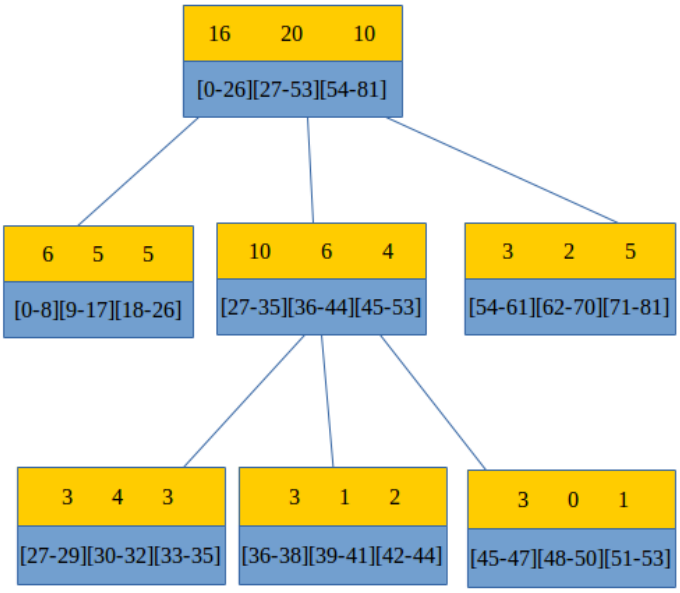

Figura 3.11: Ejemplo de un ranking con la estructura que usa Google Code Jam Ranking Library

Árboles

Son las estructuras no lineales más importantes que surgen en algoritmos informáticos. Este tipo de estructura está basado en **relaciones de ramificación entre nodos**. Formalmente hablando, un árbol es un conjunto *T* de uno o más nodos tal que [Knu95]:

- hay uno nodo designado especialmente llamado ***raíz*** del árbol, raíz(*T*); y
- los nodos restantes están particionados en $m \geq 0$ conjuntos disjuntos $T_1, ..., T_m$, y cada conjunto es un árbol. Los árboles $T_1, ..., T_m$ se denominan **subárboles** de la raíz.

Esta definición es recursiva porque se ha definido un árbol en términos de árboles. No hay ningún **problema de circularidad** involucrado, debido a que los árboles con un nodo deben constar solamente la raíz y los árboles con *n >1* nodos son definidos en términos de árboles con menos que *n* nodos.

Cada nodo de un árbol es la raíz de algún subárbol que se encuentra dentro de todo árbol. El número de subárboles de un nodo se llama el **grado** de este nodo. Un nodo de grado cero se llama **nodo terminal**, o algunas veces **hoja**. Un nodo no terminal a menudo se le llama **nodo rama**. El **nivel** de un nodo con respecto a *T* es definido recursivamente: El nivel de raíz(*T*) es cero y el nivel de otro nodo cualquiera es uno más alto que este nivel de nodo con respecto al subárbol de raíz(*T*) contenido en él.

Si el orden relativo de los subárboles $T_1, ..., T_m$ que se encuentran en la segunda parte de la definición es importante, se dice que el árbol es un **árbol ordenado**. Si no se considera dos árboles como distintos cuando sólo se distinguen en la respectiva ordenación de los subárboles de los nodos, se dice que el árbol es **orientado**, ya que sólo se considera la orientación relativa de los nodos, no su orden.

Un **bosque** es un conjunto, normalmente ordenado, de cero o más árboles disjuntos. Existe una pequeña distinción entre árboles y bosques abstractos. Si se elimina la raíz de un árbol, se obtiene un bosque y si a un bosque se le añade un nodo a cualquier bosque y conserva los árboles del bosque como subárboles del nuevo nodo, se obtiene un árbol.

Cada raíz es el **padre** de las raíces de sus subárboles y estos dos últimos nodos son **hermanos** entre ellos e **hijos** de su padre. La raíz del árbol entero no tiene padre. Se usa las palabras antecesor y descendiente para indicar una relación que puede abarcar varios niveles del árbol.

Si $n_1, n_2, ..., n_k$ es una secuencia de nodos en el árbol donde n_i es el padre de n_{i+1} para $1 \leq i \leq k$, entonces esta secuencia es denominada camino de n_1 a n_k. La longitud de este camino es *k* - 1. Si existe un camino desde el nodo *R* al nodo *M*, entonces *R* es un antecesor de *M* y *M* es un descendiente de *R*.

La profundidad de un nodo *M* que se encuentra en el árbol es la longitud de la trayectoria que va desde la raíz del árbol hasta *M*. **La altura de un árbol** es la profundidad del nodo más profundo del árbol. Todos los nodos de profundidad *d* se encuentran en el mismo nivel *d* en el árbol. La raíz es el único nodo que se encuentra en el nivel cero y su profundidad es cero.

En un árbol binario cada nodo tiene como máximo dos subárboles, y cuando sólo se mues-

tra uno, hay que distinguir entre el subárbol de la izquierda o el de la derecha. Un árbol binario es como un conjunto finito de nodos que está vacío o consiste en una raíz y los elementos de dos árboles binarios disjuntos llamados **subárboles izquierdo y derecho de la raíz**. Un árbol binario puede estar vacío, mientras que un árbol general no. Los árboles binarios tienen una forma importante: **el árbol binario completo**, donde cada nodo rama del árbol tiene exactamente dos hijos no vacíos o una hoja.

Existe un tipo de árboles cuyos nodos internos tienen exactamente N hijos y se le denomina **árbol N-ario**. Un árbol binario completo es un árbol 2-ario. Se diferencia de los árboles generales en que cada nodo tiene un número fijo de hijos.

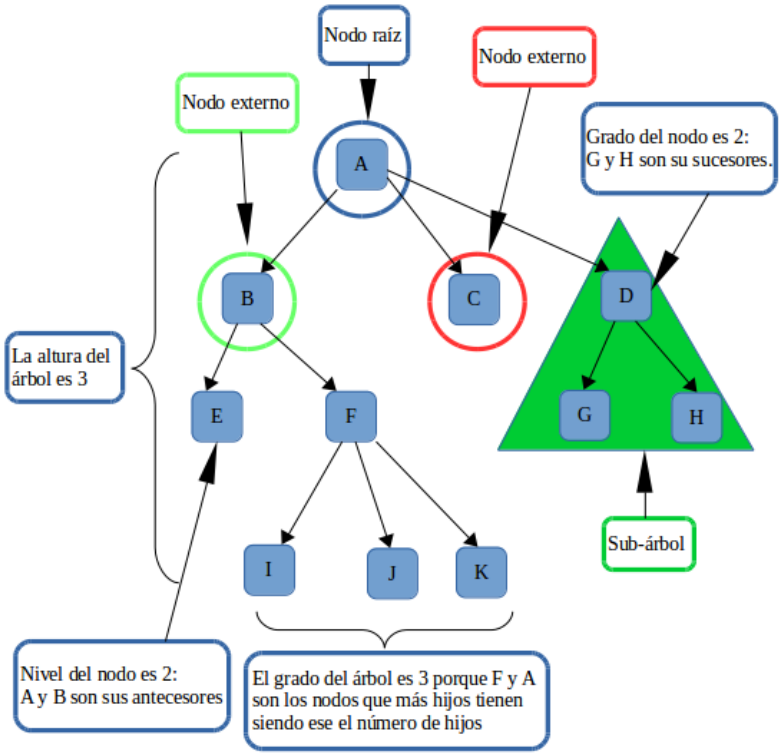

Figura 3.12: Elementos de un árbol

Tipos de recorridos

A veces, se puede procesar un árbol visitando cada uno de sus nodos, realizando cada vez una acción específica. Cualquier proceso que consiste en visitar todos los nodos siguiendo un orden, se le denomina recorrido. Cualquier recorrido que enumera todos los nodos de árbol una vez se denomina enumeración de los nodos [Sha09].

- **Pre-orden.** Consiste en visitar cualquier nodo antes de visitar a los hijos de este.
- **In-orden.** Consiste en visitar primero al hijo que se encuentra en la izquierda (incluyendo su subárbol entero), entonces se visitan los nodos y finalmente se visita el hijo

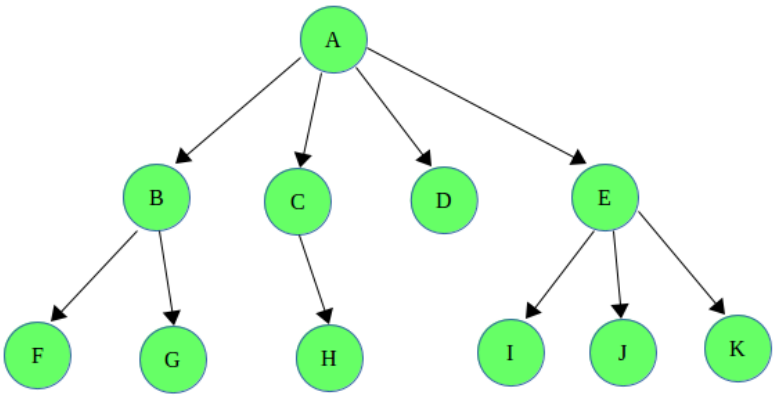

Figura 3.13: Árbol general

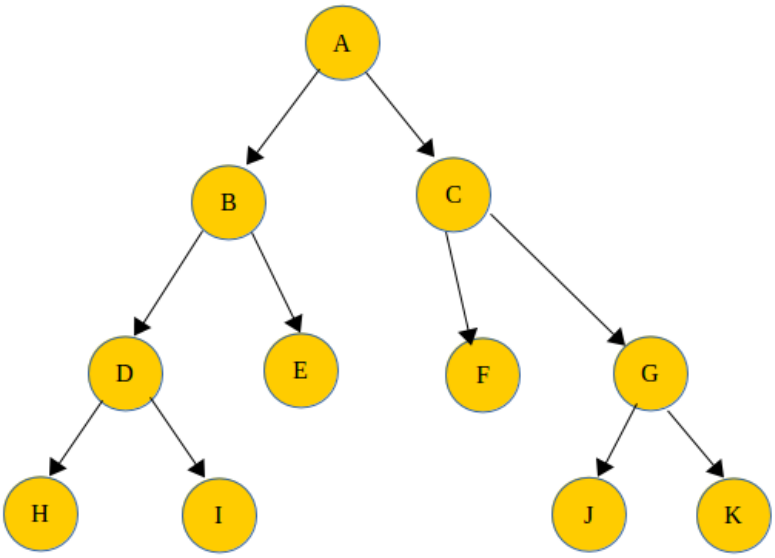

Figura 3.14: Árbol binario

de la derecha (incluyendo su subárbol).

- **Post-orden.** Consiste en visitar cada nodo después de visitar a sus hijos (y sus subárboles).

Herramientas de desarrollo web

Google App Engine (GAE)

Es uno de los servicios que constituyen la familia de **Google Cloud Platform**(un conjunto de servicios para el funcionamiento escalable de las aplicaciones, la realización de grandes cantidades de trabajo computacional, y almacenamiento, usando y analizando muchos datos) que permite desarrollar aplicaciones móviles y web que se puedan autoescalar de manera eficaz [San15]. Este servicio se lanzó el 7 de abril del 2008 como un servicio de *cloud*, pero

como un servicio de **Plataforma como Servicio**. Es un servicio de alojamiento de aplicaciones web. Tiene APIs y servicios integrados que se suelen usar en la mayor parte de las aplicaciones. Entre estos servicios que ofrece están **las bases de datos NoSQL, Memcache y una API de autenticación de usuarios**. Las aplicaciones desarrolladas con GAE usan los servicios de la plataforma si es necesario.

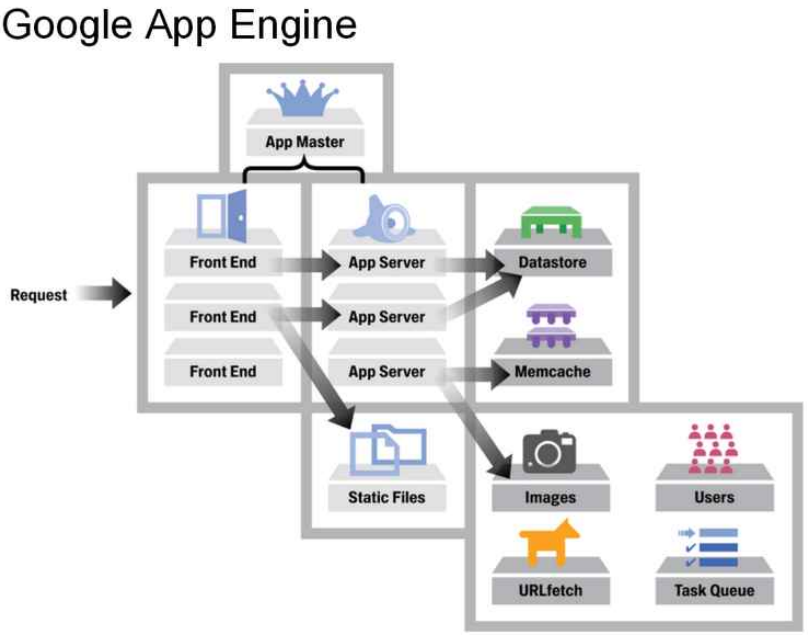

Figura 3.15: Funcionamiento de GAE(Fuente: http://www.manejandodatos.es/2014/12/como-funciona-el-front-end-back-end-en-google-app-engine/)

GAE también sirve contenido tradicional de la web, como imágenes y documentos, pero el entorno está diseñado para aplicaciones dinámicas de tiempo real. *App Engine* está diseñado **para alojar aplicaciones que tienen muchos usuarios simultáneos**.

Con la ayuda de *App Engine*, **la aplicación desarrollada escalará de manera automática dependiendo de la cantidad de tráfico que recibe**. Lo único que tiene que hacer el usuario es subir el código que ha desarrollado, y *Google* administrará la disponibilidad de la aplicación. La aplicación no necesita saber nada de los recursos que necesita. App Engine asigna más recursos a la aplicación y gestiona el uso de esos recursos. El usuario tendrá que pagar los recursos necesarios, teniendo de manera gratuita algunos de ellos, suficientes para pequeñas aplicaciones con bajo tráfico. Entre los recursos facturados se encuentran la CPU usada, almacenamiento por mes, ancho de banda de entrada y salida, y recursos específicos para los servicios que ofrece *App Engine*.

Una aplicación web de App Engine puede ser descrita por tres partes: **instancias de la aplicación, *Datastore* escalable y servicios escalables**.

Entre los clientes más importantes de *Google App Engine* se encuentran *Rovio, Khan Academy, Best Buy y Feedly* que se muestran en la figura 3.17. Estos clientes han ahorrado mucho

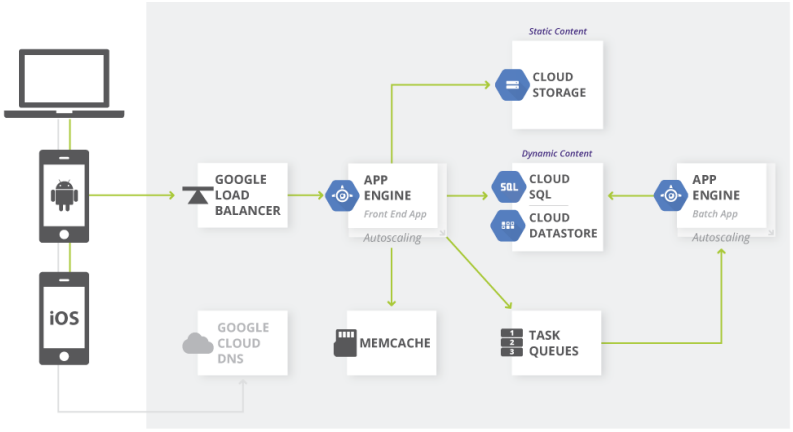

Figura 3.16: Arquitectura de aplicaciones web con GAE (Fuente: Google)

tiempo en la creación y diseño de la infraestructura usando *App Engine* y sus servicios.

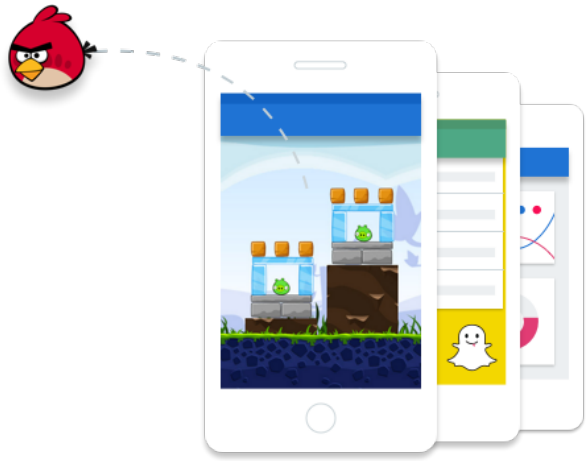

Figura 3.17: Aplicaciones que hacen uso de GAE(Fuente: Google)

Ventajas

Las ventajas que proporciona GAE son las siguientes [App]:

- **Desarrollo más rápido.** Se puede desarrollar aplicaciones móviles y web mucho más rápido con la ayuda de los servicios que contiene, como el balanceo de carga, las comprobaciones del estado y los registros de la aplicación.
- **Escalado automático.** *App Engine* contiene una opción de escalado automático que hace que las aplicaciones se escalen de manera rápida y automática dependiendo de lo que necesite el entorno, desde cero usuarios hasta un grandísimo número de estos.

- **Análisis de seguridad automático.** GAE contiene un software llamado *Security Scanner*, que analiza y detecta de forma automática los problemas de seguridad más usuales de las aplicaciones web que permite identificar las amenazas más rápido y con minúsculo índice de falsos positivos.
- **Uso de herramientas favoritas.** Con *App Engine* se puede trabajar usando las herramientas más famosas de desarrollo, como *Eclipse, Maven, Git, PyCharm, IntelliJ* y *Jenkins*(ver Figura 3.18), evitando cambiar el flujo de trabajo.

Figura 3.18: Herramientas de desarrollo que son compatibles con GAE(Fuente: Google)

Características

Las caracteristícas de esta herramienta son las siguientes [App]:

- **Autenticación de usuarios.** Los usuarios pueden iniciar sesión en las aplicaciones con sus cuentas de *Google*, y se les asignan identificadores únicos.
- **Lenguajes populares.** Se pueden desarrollar aplicaciones con lenguajes muy usados como *Java, Python, Go* o *PHP*.
- **Base de datos NoSQL.** Tiene un *Datastore* de objetos que no sigue ningún esquema, con almacenamiento escalable, una API de modelado de datos muy completa y un lenguaje de consultas muy similar a *Structured Query Language (*SQL*)*.
- **Google Cloud SQL.** Es un servicio web que está gestionado totalmente que permite crear, configurar y usar las bases de datos relacionales que se encuentran en la nube de Google.

- **Memcache.** Es un servicio de almacenamiento distribuido en memoria caché. Esto permite mejorar el rendimiento de las aplicaciones.
- **Security Scanner.** Es una herramienta que analiza y detecta automáticamente los problemas de seguridad que suelen ser más habituales en las aplicaciones web.
- **Búsqueda.** Realiza búsquedas similares como las que hace Google, pero son aplicadas a datos estructurados.
- **División del tráfico.** Las solicitudes recibidas son enviadas a diferentes versiones de la aplicación, permitiendo así mantener varias versiones de una aplicación determinada, y se realizan pruebas A/B y despliegues incrementales de funcionalidades.
- **Registros.** Acceso programático a los registros de la aplicación y a solicitudes desde la propia aplicación.
- **Colas de tareas.** Las aplicaciones pueden hacer acciones por fuera de las peticiones que envía el usuario, mediante pequeñas e individuales tareas que son ejecutadas más tarde.

Restricciones

- Las aplicaciones desarrolladas únicamente tienen permiso de lectura a los archivos que se encuentren en el sistema de archivos.
- El código solo se puede ejecutar mediante consultas *HTTP*.
- En aplicaciones *Java* está limitado el uso de clases del *JRE* estándar a las que son consideradas como seguras.
- Los hilos de ejecución no pueden ser creados por las aplicaciones.
- Los usuarios que desarrollen aplicaciones en *Python* pueden subir módulos para que puedan ser usados en la plataforma, pero no deben estar desarrollados completamente en *C* o *Pyrex*.
- Los dominios **.appspot.com* son los únicos que tienen soporte para *Secure Sockets Layer (*SSL*)*.
- Un proceso que sea iniciado por un servicio para proporcionar una respuesta a una consulta debe durar como mucho treinta segundos.
- No tiene soporte a la persistencia de las sesiones.
- No soporta la apertura de sockets.

Google Cloud Endpoints

Es un servicio y un conjunto de herramientas que permiten **la generación de APIs de manera sencilla y automática** para poder establecer una comunicación entre las aplicaciones

clientes y una web *backend*, es decir, permite acceder de manera remota a la API que se ha implementado, que va a ser el *backend*, desde cualquier dispositivo móvil que use el sistema operativo *Android* o *iOS* o cualquier navegador, que es el *frontend*(ver Figura 3.19). Pero para poder acceder a esa API, se deben de enviar **peticiones autenticadas mediante OAuth 2.0**. La parte del *backend* va a ser almacenada y gestionada por *Google App Engine*.

Facilita la llamada a métodos que se encuentran en el servidor mediante el uso de un móvil o un cliente web. Tiene librerías y herramientas para generar la funcionalidad de un servidor mediante un conjunto de métodos implementados en *Python* y *Java* y generar el código del cliente para *Android*, *iOS* y un navegador basado en *JavaScript*. También pueden generar un documento de descubrimiento que trabaje con *Google* APIs *Client Libraries* para muchos lenguajes cliente. Por último, para la autenticación se usa *OAuth*.

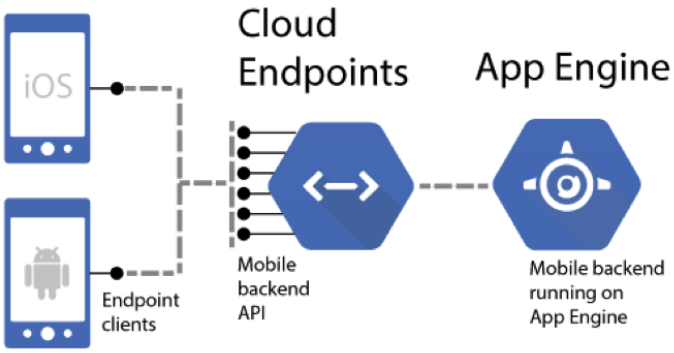

Figura 3.19: Arquitectura para aplicaciones moviles con *Google Cloud Endpoints* (Fuente: https://www.3pillarglobal.com/insights/building-backend-applications-with-google-app-engine-google-cloud-endpoints-and-android-studio)

Webapp2

Es un *framework web de Python* ligero compatible con *webapp* de *Google App Engine* que **permite crear de manera rápida aplicaciones web sencillas desarrolladas en Python 2.7**. Mantiene la simplicidad de *webapp*, pero la mejora añadiendo mejor enrutado *Uniform Resource Identifier (*URI*)* y manejo de excepciones, un objeto de respuesta con todas las funciones y un mecanismo de despacho más flexible. También ofrece un paquete llamado **webapp2_extras** con varias utilidades opcionales como **las sesiones, localización, internacionalización, enrutado de dominios y subdominios, cookies seguras y otras más**.

Es un *framework* pequeño y sencillo de utilizar. Es bueno para proyectos pequeños, aunque no tenga las características de los *frameworks* más usados.

Las aplicaciones que usan este *framework* están **formadas por uno o más manejadores de petición**, que son unidades de código mapeadas por direcciones *Uniform Resource Loca-*

*tor (*URL*)* que se ejecutan cuando un cliente o un proceso manda una petición a su dirección URL. *Webapp2* **instancia la clase y llama al método que se encuentra en la clase** que corresponde al método HTTP de la petición para generar la respuesta.

Google Cloud Datastore

Es una base de datos documental *NoSQL* construida para **el escalado automático, alto rendimiento y facilidad de desarrollo de aplicaciones** [Dat16]. Sus características son las siguientes:

- **Transacciones atómicas.** Puede ejecutar un conjunto de operaciones donde todas se realizan exitosamente o no se realiza ninguna.
- **Alta disponibilidad para leer y escribir.** Usa redundancia para minimizar el impacto de los puntos que fallan.
- **Escalabilidad masiva con alto rendimiento.** Utiliza una arquitectura distribuida para gestionar de forma automática el escalado. *Cloud Datastore* usa una combinación de índices y restricciones de consulta para que sus consultas se escalen con el tamaño de su conjunto de resultados en vez de con el tamaño de su conjunto de datos.
- **Almacenamiento y consulta de datos flexible.** Mapea de manera natural lenguajes orientados a objeto y de script y está expuesto a las aplicaciones a través de gran cantidad de clientes. También proporciona un lenguaje de consulta parecido a SQL.
- **Equilibrio de consistencia fuerte y eventual.** Garantiza que las búsquedas de entidades por clave y antepasado siempre reciben datos fuertemente consistentes y las demás consultas son eventualmente consistentes.
- **Cifrado en reposo.** Encripta automáticamente todos los datos antes de su escritura en el disco y de forma automática desencripta los datos cuando son leídos por un usuario autorizado.
- **Totalmente gestionado sin tiempo de inactividad planificado.** *Google* gestiona el servicio que ofrece *Cloud Datastore*, por lo tanto, el cliente se puede enfocar en su aplicación.
- No requiere que las entidades de la misma naturaleza tengan un conjunto de propiedades consistentes como las bases de datos relacionales que imponen un esquema.
- Los tipos de consultas que se pueden ejecutar son más restrictivas que las que se permiten en una base de datos relacional debido a que todas las consultas son atendidas por los índices que han sido constituidos anteriormente.

Cloud Datastore difiere de las bases de datos relacionales en los siguientes aspectos:

- Las entidades almacenadas en el *Cloud Datastore* del mismo tipo pueden tener propiedades distintas.

- Diferentes entidades pueden tener propiedades con el mismo nombre, pero con distintos tipos de valor.
- *Cloud Datastore* está diseñado para escalar automáticamente a grandes conjuntos de datos, haciendo que las aplicaciones puedan conservar su alto rendimiento independientemente del tráfico que reciban.
- Otra de las diferencias que tienen consiste en los conceptos que usan, que cada uno se representa de distinta manera. En la tabla 3.1 se muestran estas diferencias.

Elemento	Cloud Datastore	Base de datos relacional
Categoria del objeto	Tipo	Tabla
Objeto	Entidad	Fila
Característica de un objeto	Propiedad	Campo
Identificador del objeto	Clave	Clave primaria

Cuadro 3.1: Diferencias entre el Cloud Datastore y una base de datos relacional con los conceptos usados.

Al igual que todos los sistemas de bases de datos proporcionan un mecanismo para realizar la ejecución de las consultas, *Cloud Datastore* también, pero difiere a la técnica que usan las bases de datos tradicionales. **Cloud Datastore busca la respuesta en una lista que contiene las posibles respuestas que han sido preparadas anteriormente cuando la aplicación hace una consulta**, en vez de buscar entre los registros y realizar cálculos para obtener la respuesta.

El tipo de listas utilizadas también se denominan **índices**. **Cloud Datastore mantiene un índice para cada consulta que la aplicación puede realizar**. De esta manera, con cada consulta que se haga, se va a escanear un índice, haciendo que la aplicación obtenga los resultados de manera rápida.

Cuando la aplicación crea nuevas entidades y actualiza las ya existentes, **el *Datastore* actualiza cada índice correspondiente**. Esto genera consultas muy rápidas a expensas de cambios en las entidades. El rendimiento de la consulta de índices respaldados no está afectada por el número de entidades que se encuentran en el *Datastore*, pero si por el tamaño del conjunto de resultados.

En la figura 3.20, se muestra un índice que puede representar un *ranking*, donde se ordena las entidades por la propiedad que representa la puntuación de cada usuario, la única que contiene este tipo de entidades, yendo de manera descendiente. La clave de las entidades es el nombre del usuario.

Cuando la consulta obtiene los resultados, *Cloud Datastore* usa las claves obtenidas en el índice para encontrar las entidades correspondientes y devolver las entidades completas a la aplicación.

Clave(Nombre de usuario)	*Puntuación*
David	580
Óscar	500
María	357
Miguel	323
José	239
Ana	93

Figura 3.20: Ejemplo de índice

La API de *Cloud Datastore* dos maneras de formular las consultas: **una usando una interfaz orientada a objeto y otra basada en un lenguaje de consulta basado en texto llamado Google Query Language (GQL)**.

La segunda manera es con GQL, que esta destinado a parecerse a SQL, lenguaje de consultas que utilizan las bases de datos relacionales. Sólo soporta las características del motor de búsqueda de *Cloud Datastore*. Además, no puede realizar actualizaciones, eliminaciones e inserciones, solamente consultas. Es suficientemente expresivo y conciso para ser útil para la realización de consultas del *Datastore*.

Los servicios

La aplicación utiliza una API para poder acceder a **un sistema que se encuentra separado y que gestiona todo su propio escalamiento necesario para las instancias de la aplicación**. Tanto *Google Cloud Platform* como *App Engine* incluyen servicios autoescalables de utilidad para las aplicaciones web.

Memcache *Memcache* es **un servicio de memoria caché** que proporciona *App Engine* que guarda pares *clave-valor*. Se puede establecer un valor con una clave y de esta manera obtener el valor dando la clave asignada a ese valor.

Memcache no soporta transacciones como las que se usa en el *Datastore*, pero proporciona varias operaciones atómicas. En la memoria caché, la configuración de un único valor es atómica, es decir, que la clave obtiene el nuevo valor o retiene el antiguo. Se le puede indicar que establezca un valor solamente si no ha cambiado desde que se buscó por última vez. También puede incrementar o decrementar valores numéricos madiante una operación atómica.

Una manera de hacer que convivan *memcache* y el *Datastore* es **almacenar en caché entidades que se encuentra en el *Datastore* con sus claves**. En el momento en el que se quiere buscar una entidad con su clave, primero comprueba si se encuentra en la memoria caché y si ocurre el caso en el que se encuentra, se usa (esto se considera un **éxito de caché**). Si no está en la memoria caché (considerado como un **fallo de caché**), se busca en el *Datastore* y esta

entidad se pone en la memoria caché para cuando en el futuro se intente acceder a ella.

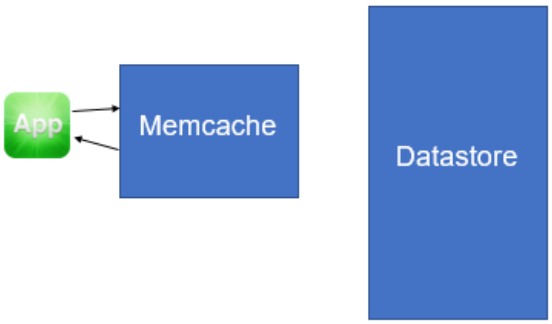

Figura 3.21: Éxito de memcache

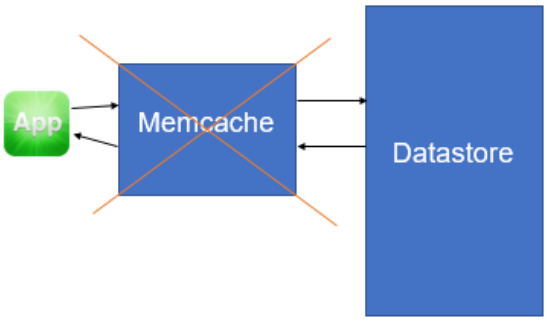

Figura 3.22: Fallo de memcache

Si una entidad cambia en el *Datastore*, se puede actualizar la memoria caché en el instante en el que esto ocurra. Esto hace que la memoria caché pueda tener dos problemas. Uno de ellos es que la actualización en memoria caché falle conservando así los datos antiguos, incluso si los datos han sido actualizados con éxito en el *Datastore*. El otro problema es que, si dos procesos actualizan el *Datastore*, y por lo tanto la memoria caché, el almacén tendrá los datos correctos pero la actualización de la memoria caché tendrá el valor de la última actualización que se haya llevado a cabo.

Para evitar esto, lo mejor es **borrar la clave de memoria caché cuando cambia el *Datastore* y dejar que la próxima vez que se vaya a hacer una lectura a una entidad, esta se almacene en caché con el valor actual**. Para disminuir la duración en la que un valor pueda permanecer obsoleto en la memoria caché, se puede dar al valor un tiempo de caducidad cuando se establezca. Cuando el tiempo de caducidad expire, la caché desactiva la clave, provocando que una lectura posterior de como resultado un fallo de caché y se active una nueva búsqueda desde el *Datastore*.

El almacenamiento en memoria caché permite disminuir el tiempo de las solicitudes y ahorra tiempo tiempo de CPU.

Google Cloud Storage Es **un servicio de almacenamiento especialmente para valores muy grandes** proporcionado por *Google Cloud Platform*. La aplicación puede usarlo para **almacenar, gestionar y servir grandes ficheros**. *Cloud Storage* puede aceptar grandes ficheros subidos por usuarios y procesos offline. Este servicio se distingue de *Cloud Datastore* en que trabaja en limitaciones de infraestructura en tamaño de peticiones y respuestas entre usuarios, servidores de aplicaciones y servicios. El código de la aplicación puede leer valores provenientes de Cloud Storage en trozos que encajan dentro de los límites.

Google Cloud SQL Este servicio proporciona **todas las funciones de alojamiento de una base de datos *MySQL***. *Cloud SQL*, a diferencia de *Cloud Datastore* y *Cloud Storage*, no es escalable automáticamente. El desarrollador crea instancias SQL, máquinas virtuales que se encargan de ejecutar software administrado de *MySQL*. Las instancias son de gran tamaño y el propietario de la aplicación solamente paga por el almacenamiento usado y la cantidad de tiempo que una instancia esta ejecutándose. Se puede configurar cada instancia para pararla cuando está inactiva y reactivarla cuando un cliente intenta conectarla.

Servicio de búsqueda Es **un servicio de almacenamiento que se encarga de proporcionar una infraestructura de búsqueda de texto completo**. Este servicio almacena documentos con campos. La aplicación va a añadir documentos a índices. El servicio de búsqueda se usa para realizar búsquedas de texto facetado sobre los campos de los documentos en un índice, incluyendo equivalencia de cadenas de texto parciales, rango de consultas y expresiones de búsqueda booleanas. También soporta derivación y tokenización.

Servicio de búsqueda de URL Es **uno de los servicios que permite que las aplicaciones desarrolladas con App Engine puedan acceder a otros recursos web**. Este servicio crea peticiones *HTTP* a otros servidores en Internet, para obtener páginas o interactuar con servicios web. La API de búsqueda de URL soporta búsquedas de URLs en segundo plano mientras que un manejadores de la petición hace otras cosas debido a que los servidores remotos pueden ser lentos en responder, aunque la búsqueda debe comenzar y terminar dentro de la vida útil del manejador de la petición siempre. La aplicación puede establecer un límite de tiempo, haciendo que una vez superado ese tiempo, la llamada es cancelada.

Servicio de correo Es **el servicio que permite mandar correos a las aplicaciones desarrolladas con App Engine**. La aplicación puede enviar una correo electrónico en nombre suyo o en nombre del usuario que realizó la petición.

También puede recibir un correo electrónico una aplicación. Si la aplicación está configurada para eso, un mensaje que ha sido enviado a la dirección de la aplicación es enrutada al servicio de correo, que reparte el mensaje de la aplicación en forma de una petición *HTTP* a un manejador de la petición.

Las aplicaciones pueden enviar y recibir mensajes instantáneos desde y hacia los servicios que pueden soportar el protocolo *XMPP*. Una aplicación manda un mensaje de chat llamando al servicio *XMPP*. Cuando alguien recibe un mensaje a la dirección correspondiente a la aplicación, el servicio *XMPP* lo reparte a la aplicación llamando a un manejador de petición.

Servicio de canal Es **un servicio que permite la comunicación directa bidireccional en tiempo real mediante un navegador web**. Este servicio permite a los navegadores mantener una conexión de red abierta con un *host* remoto para recibir mensajes en tiempo real mucho después de que se haya terminado de cargar la página web. GAE encaja esto en su modelo de procesamiento basado en la solicitud mediante el uso de un servicio: los navegadores se conectan a canales a través de un servicio.

Cuando una aplicación envía un mensaje a un cliente durante su procesamiento, esta llama al servicio de canal con el mensaje. El manejador de este servicio se encarga de difundir el mensaje a los clientes y gestionar las conexiones abiertas. También proporciona un navegador de mensajería en tiempo real.

Google Accounts, OpenID y OAuth

GAE está integrado con **Google Accounts**, que es una cuenta del usuario del sistema utilizado por las distintas aplicaciones que *Google* proporciona. Esto se puede usar también como un sistema de autentificación de usuario de la aplicación, de tal forma que no hace falta construir ese sistema desde el principio. Y si los usuarios que utilizan las aplicaciones desarrolladas con GAE tienen una cuenta de *Google*, podrán iniciar sesión usando esa cuenta, haciendo que no sea necesario crear una cuenta solo para usar la aplicación.

Para este tipo de aplicaciones no es obligatorio el uso de *Google Accounts*. Se puede construir **un sistema de cuentas propio o usar un proveedor OpenID**. *App Engine* tiene soporte para el uso de estos proveedores.

GAE también tiene soporte incorporado para **OAuth**, un protocolo que permite conceder permiso a los usuarios para que las aplicaciones de terceros puedan tener acceso a datos de otro servicio, sin necesidad de compartir sus credenciales de la cuenta con terceros. Las características integradas de *OAuth* solo funciona cuando se usa cuentas de *Google*. No existe un soporte especial para la implementación de un cliente de *OAuth* en una aplicación de *App Engine*, pero muchas de las librerías de cliente de *OAuth* funcionan bien con *App Engine*.

Task Queues y Cron Jobs

La API de colas de tareas permite a las aplicaciones llevar a cabo un trabajo, llamado **tareas**, de manera asíncrona exterior de una petición del usuario. Si una aplicación tiene que ejecutar un trabajo en segundo plano, se añade tareas a las **colas de tareas o task queues**. Estas tareas son ejecutadas más tarde. Las colas de tareas tienen **un productor**, que es un proceso que pide que el trabajo sea realizado. El productor añade una tarea en una cola. También tiene **un consumidor**, que es un proceso que arrenda las tareas que se encuentran en la cola y que pretende realizar. Si el consumidor realiza la tarea de manera exitosa, esta es eliminada de la cola para que no pruebe a realizar otro consumidor. Si el consumidor falla a la hora de borrar tarea, la cola asume que la tarea no ha sido completada con éxito, y pasado un tiempo, el arrendamiento expira y vuelve a esta disponible para que otro consumidor la pueda realizar. Un consumidor también puede anular el arrendamiento si no puede llevar a cabo la tarea.

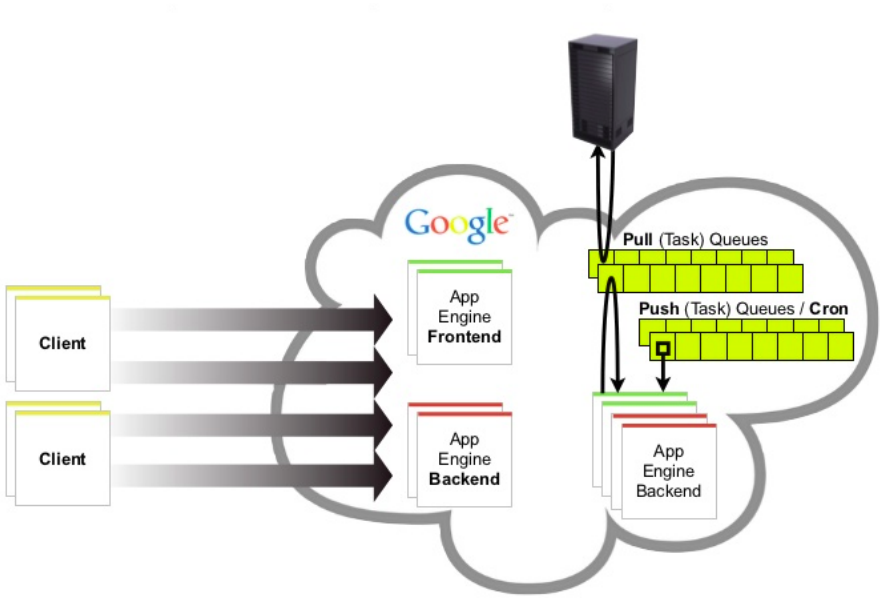

Figura 3.23: Funcionamiento de las *task queues* y los *cron jobs* (Fuente: https://www.slideshare.net)

Las colas de tareas son de dos tipos, **push queue y pull queue**. La forma en la que el servicio de la cola de tareas manda las solicitudes a los servicios que se encargan de llevar a cabo la tarea es distinto para las diferentes colas.

Las *push queues* son **colas de tareas donde estas son realizadas automáticamente por *App Engine* a una velocidad configurable**. *App Engine* realiza una tarea invocando un manejador de peticiones de la aplicación. Forma una solicitud *HTTP* basada en el contenido del registro de la tarea y emite la solicitud a una ruta URL asociada a la tarea. *App Engine* usa el código de estado *HTTP* de la respuesta para decidir si la tarea fue completada de manera

exitosa y debe ser eliminada de la cola. Las tareas que han fallado se vuelven a realizar más tarde.

Este tipo de colas va a intentar hacer otra vez una tarea hasta que se compruebe que la tarea ha sido completada. También van a reintentar tareas en la que el manejador de petición devuelve una respuesta *HTTP* con un código de estado distinto a uno de éxito. La tarea vuelve a ser puesta de nuevo en la cola con una cuenta atrás, por lo que no va a estar disponible ni se va a llevar a cabo una vez que se encuentre en la cola para tener esperanza en que la condición de error disminuirá.

El productor en este tipo de colas es el código de la aplicación ejecutado en un manejador de peticiones de *App Engine*, y el consumidor es el mecanismo de la *push queue* de *App Engine*, que llama a manejadores de peticiones para hacer el trabajo actual de la tarea.

En las *pull queues*, el usuario proporciona el consumidor lógico. Este consumidor llama a la *pull queue* para **arrendar una o varias tareas y la cola asegura que una tarea determinada es arrendada solamente por un consumidor en un instante de tiempo**. Una vez realizado el trabajo correspondiente a la tarea, el consumidor elimina la tarea de la cola, de esta forma los demás consumidores no la ven. Si el consumidor falla al borrarla, el arrendamiento caduca y la *pull queue* hace que la tarea esté disponible para los consumidores otra vez.

Una *pull queue* es útil cuando se va a personalizar el consumidor lógico. Con este tipo de cola, el consumidor puede arrendar varias tareas que estén relacionas a la vez y realizarlas juntas como si se tratase de un lote. Esto puede hacer que sea más rápido o más productivo en vez de hacer una a la vez.

El consumidor de estas colas que se ejecuta en *App Engine* usa la API de servicio de cola de tareas para arrendar y eliminar tareas. Un arrendamiento garantiza que un consumidor tiene acceso exclusivo a una tarea durante un tiempo determinado, en el cual puede realizar cualquier trabajo que corresponda con esa tarea. Una vez completada la tarea, ese consumidor elimina la tarea. Para poder arrendar tareas que se encuentren en una cola, hay que llamar a un método de la cola especificando el tiempo de duración del arrendamiento y el número máximo de tareas que se quieren arrendar. Este método devuelve los identificadores de cada tarea arrendada, que pueden ser usados para eliminar tareas o actualizar arrendamientos. Una vez que se ha realizado la tarea con éxito, se debe eliminar esta para que no la vuelva a llevar a cabo otro consumidor. Si el consumidor necesita más tiempo para completar la tarea, este puede renovar el arrendamiento sin tener que renunciar a él para que otro usuario lo haga.

Cuando la duración del arrendamiento expira, la tarea vuelve a estar disponible en la cola. Entonces un consumidor puede arrendar la tarea y realizar de la tarea de nuevo. Esto es similar al caso de reintento de una tarea en una *push queue*: si un consumidor no ha podido eliminar la tarea antes de que finalizará el tiempo de arrendamiento, se supone que la tarea

ha fallado y se debe intentar hacerla de nuevo.

GAE también tiene otro servicio para llevar a cabo la ejecución de tareas, pero en momentos específicos del día: **el servicio de tareas programadas**. A las tareas programadas también se le conocen como *cron jobs*. El servicio de tareas programadas invocan **un manejador de petición en un determinado momento del día, semana o mes**. Estas tareas son útiles para el mantenimiento regular o el envío de mensajes de notificación periódico.

Lo mismo que ocurre con las tareas que se encuentran en las *push queues*, estas tareas tienen un límite de tiempo de diez minutos para la petición. Dependiendo del tiempo que se tarde en completar la tarea, es posible que se tenga que dividir el trabajo en pequeñas partes y usar colas de tareas para que sean ejecutadas en varias instancias en paralelo.

Las tareas que fallan su ejecución no vuelven a intentarse. Si una tarea programada tiene que volverse a intentar inmediatamente, esta tarea debe poner su trabajo en una *push queue*.

Software Development Kit (SDK)

Para poder implementar aplicaciones, *Google* proporciona un kit de desarrollo(*SDK*) para cada lenguaje soportado por *Google App Engine*, que en nuestro caso va a ser *Python*. El *SDK* es una colección de herramientas y librería que nos **permite desarrollar, probar y desplegar software desde *Cloud Platform***.

El servicio de desarrollo web es la parte más útil del SDK, que hace que se ejecute la aplicación en ordenador local y simule el entorno de ejecución y servicios. Este detecta automáticamente las modificaciones que se han llevado a cabo en los ficheros de origen y vuelve a cargarlos si son necesarios, permitiendo que se desarrolle la aplicación mientras el servidor mantiene ejecutándose. Este servicio tiene incluido una consola de desarrollo que está basada en el navegador integrado para la inspección y estimulación de la aplicación. Esta consola se usa para inspeccionar y modificar los contenidos de los servicios de almacenamiento de datos simulados, gestionar las interacciones de la cola de tareas y simular eventos no relacionados con la web.

El conjunto de herramientas se utiliza para interactuar con GAE de forma directa. Se puede descargar los datos de registro con la aplicación en ejecución y gestionar los índices del *Datastore* y la configuración del servicio de la aplicación en directo.

Por último, el SDK contiene librerías que permiten probar las aplicaciones de forma automática y reunir informes sobre su rendimiento.

En el caso de este proyecto, va a permitir desplegar las aplicaciones de manera local y monitorizar los procesos que se llevan a cabo en la aplicación a través de la consola del servidor de desarrollo para poder hacer pruebas. Una vez que la aplicación esté libre de errores y haya alcanzado los objetivos de este proyecto, permitirá desplegar las aplicaciones

en los servidores de *Google* y gestionarla mediante el panel de administración de *Google Cloud Platform*.

Cloud Console

Es una interfaz basada en navegador para gestionar la aplicación que se encuentra alojada en los servidores de *Google*. Para acceder a esta consola y poder crear un proyecto, se tiene que usar solamente una cuenta de *Google*. Una vez que se ha creado el proyecto, se pueden añadir más cuentas al proyecto, teniendo cada una un rol de los disponibles: propietarios, que pueden cambiar las opciones y gestionar los permisos de otras cuentas; los editores, que puede modificar las opciones; y los espectadores, que pueden ver las opciones e información del proyecto.

La consola proporciona acceso a datos relacionados con el rendimiento en tiempo real sobre la aplicación que está siendo usada y a los datos de los registros que han sido emitidos por la aplicación. Mediante la interfaz web se puede consultar los servicios de datos de la aplicación en directo y comprobar el estado de los índices del *Datastore* y otras características.

Cuando es subido un nuevo código a la aplicación, la versión del nuevo código se le asigna **un identificador de la versión**, que se encuentra especificada en el archivo de configuración de la aplicación. La versión que es usada para la aplicación ejecutada, es la versión principal que ha sido elegida por defecto. Mediante *Cloud Console*, el propietarios de la aplicación puede seleccionar la versión por defecto. También se pueden usar versiones que no sean las principales utilizando una dirección **URL** especial que contenga el identificador de la versión, permitiendo de esta forma probar una nueva versión de la aplicación antes de que sea oficial.

También se usa la consola para administrar y configurar la cuenta de facturación de la aplicación. Cuando la aplicación necesite hacer uso de más recursos que los que se le proporciona gratis, se tiene que configurar la cuenta de facturación mediante el uso de una tarjeta de crédito y *Google Accounts*. El propietario establece un presupuesto en el que se indica la cantidad máxima de dinero que puede ser cobrado por los días. La aplicación puede consumir los recursos hasta que se agote el presupuesto y a partir de ahí solo se cobra lo que usa la aplicación.

AppStats

Es una herramienta que proporciona GAE para ayudar a comprender cómo la aplicación llama a los servicios y dónde se puede optimizar los patrones de llamada. Esta herramienta se encuentra acoplada a la lógica de la aplicación para recoger los datos de tiempo de las llamadas a servicios y se muestran estos datos en una interfaz basada en web.

Una vez que se añade *AppStats* en la aplicación, registra los datos de tiempo de todas

las peticiones, aunque se pueden añadir filtros de los datos de las peticiones que se quieren registrar. Para ver estos datos se utiliza **AppStats Console**(ver Figura 3.24), que los muestra como un cronograma de la actividad de la petición. Esta herramienta se puede usar en el servidor de desarrollo y en la aplicación que se está ejecutando.

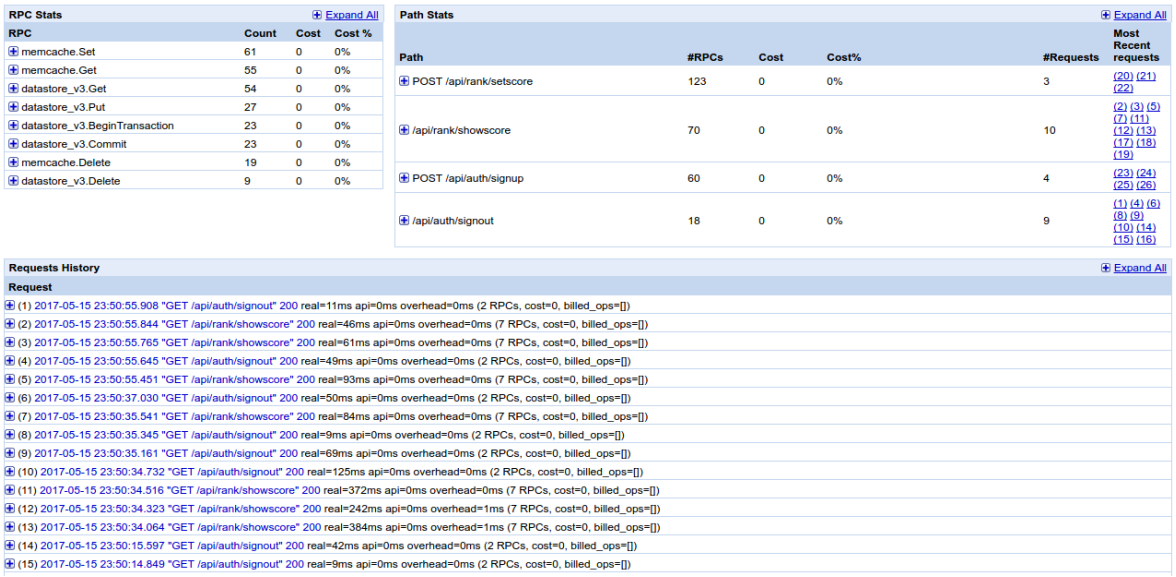

Figura 3.24: Ejemplo de interfaz de AppStats Console

AppStats está constituido por dos partes: **el registrador de eventos y AppStats Console**. El registrador de eventos se conecta a la infraestructura del servicio para que la aplicación comience el registro al principio de cada solicitud y almacene los resultados al final. Registra los tiempos de inicio y finalización del manejador de petición y de cada llamada de procedimiento remoto(*RPC*) a los servicios. También registra las trazas que se siguen en cada sitio de llamada.

El registrador guarda los datos en *memcache*, evitando de esta manera el uso del *Datastore*, y almacena dos valores: **un registro corto y otro largo**. El registro corto se utiliza para renderizar una interfaz de navegación para los registros largos mediante *AppStats Console*. Solo mantiene almacenados los últimos 1000 registros. Aunque si la aplicación hace un uso excesivo de *memcache*, los valores almacenados pueden ser desalojados de forma agresiva.

Amazon Web Services (AWS)

Es una herramienta proporcionada por Amazon que proporciona servicios y recursos bajo demanda que se encuentran alojados en la nube que sigue un modelo de precios de pago según su uso [Ama]. Se suele utilizar para los siguientes casos:

- Almacenamiento de datos.
- Alojamiento de un sitio web estático.

- Alojamiento de una aplicación web o un sitio web dinámico.
- Admisión de alumnos o programas de aprendizaje en línea.
- Procesamiento de datos empresariales y científicos.
- Observación de los picos de carga.

Ventajas

El uso de esta herramienta tiene las siguientes ventajas [AWS16]:

- **Facilidad de uso.** AWS está diseñado de tal forma que permite a los usuarios hospedar sus aplicaciones de manera rápida y segura.
- **Flexible.** Permite al usuario elegir distintas cosas a su gusto como es el sistema operativo, el lenguaje de programación, la plataforma de aplicaciones web, la base de datos y los servicios que sean necesarios. AWS proporciona acceso a un entorno virtual que permite cargar el software y los servicios necesarios para la aplicación alojada del cliente, facilitando de esta forma el proceso de migración de las aplicaciones existentes y manteniendo las opciones para poder crear nuevas soluciones.
- **Rentable.** El usuario solo tiene que pagar por el costo de la potencia de cómputo, el almacenamiento y los demás recursos usados, sin la necesidad de realizar contratos a largo plazo o compromisos iniciales.
- **De confianza.** El usuario tiene disponible una infraestructura informática global escalable, segura y de confianza.
- **Escalabilidad y alto desempeño.** Gracias a los servicios proporcionados por AWS, se puede ampliar o reducir la aplicación según la demanda.
- **Seguro.** Proporciona varias medidas que garantizan la seguridad en el uso de AWS.

Servicios

Servicios de informática y redes Proporciona servidores virtuales, configura un firewall y el acceso a *Internet*, asigna y enruta direcciones *Internet Protocol (*IP*)* y escala la infraestructura para conllevar la creciente demanda. Los principales servicios de este tipo son los siguientes:

- **Amazon Elastic Compute Cloud (EC2).** Ofrece servidores virtuales en la nube. Proporciona la capacidad de computación redimensional en el cual se puede crear y alojar sistemas software. Una *Amazon Machine Image (*AMI*)* es una plantilla que va a contener la configuración correspondiente de un software. A través de esta imagen se puede ejecutar varias instancias, que son copias de la AMI que son ejecutadas como como un servidor virtual en un host que se encuentra en el centro de datos de Amazon.

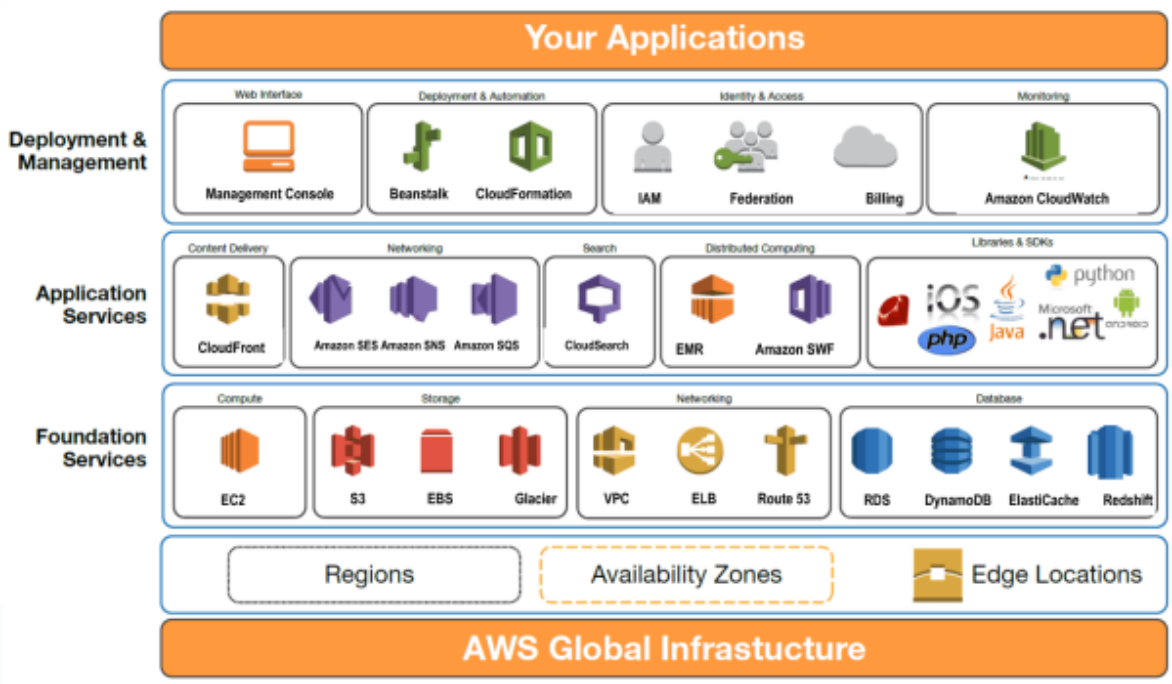

Figura 3.25: Servicios ofrecidos por la infraestructura de AWS (Fuente: http://blog.clearpathsg.com)

Cuando una instancia es lanzada, se selecciona el tipo de la instancia que va a determinar las capacidades hardware del host donde se encuentra la instancia. Las instancias van a continuar ejecutándose hasta que las detenga o las termine o hasta que se produzca algún error en ellas. Si en alguna de ellas se produce algún error, se vuelve a lanzar una nueva desde la AMI.

También proporciona volúmenes de almacén de instancias para varios tipos de instancias de EC2. Los discos que forman estos volúmenes se encuentran conectados al mismo host que la instancia, lo que permite que el acceso a estos volúmenes sea muy rápido. Un volumen de almacén de instancias solamente puede ser usado por una instancia y durante su ciclo de vida. Existen casos en los que los tipos de instancia admiten varios volúmenes, en los cuales puede distribuir los datos. Cuando la instancia finaliza su ejecución, los datos que se encuentran en el volumen son borrados. En el caso de que una aplicación use volúmenes para para almacenar datos persistentes, se replica periódicamente los datos o se almacenan también en un almacenamiento duradero.

- **Amazon Virtual Private Cloud (VPC).** Ofrece al usuario una red virtual que se encuentra aislada para los servidores virtuales. Una VPC es una red virtual que está dedicada para la cuenta de AWS. Esta se encuentra aislada de las demás redes virtuales de la nube que proporciona AWS. Ofrece al cliente la funcionalidad de red sólida y segura para sus recursos. Este tipo de red es igual a una red tradicional que es operada por el usuario desde su propio centro de datos, con las ventajas que proporciona la

infraestructura escalable de AWS.

Una subred es una subrango de direcciones IP de una VPC en el que se pueden lanzar instancias. Estas subredes van a permitir la agrupación de instancias según las necesidades operativas y de seguridad. Estas instancias deben de tener una puerta de enlace de Internet a la VPC y una tabla de enrutamiento con una ruta hacia Internet a la subred para que puedan llegar a Internet y a los servicios que ofrece AWS.

- **Elastic Load Balancing.** Reparte el tráfico entre el conjunto de servidores virtuales que proporciona. Hace uso de un equilibrador de carga, que se encarga de distribuir el tráfico entre varias instancias. Puede obtener un mayor nivel de tolerancia a fallos si se usa este equilibrador con instancias que se encuentran en varias zonas de disponibilidad. Según se vayan lanzando o finalizando las instancias, el equilibrador envía el tráfico de forma automática a las instancias que se encuentran en ejecución. Este servicio comprueba el estado en cada instancias, por lo que si una instancia no responde, el equilibrador envía el tráfico automáticamente a una nueva instancia que se encuentre en buen estado(ver Figura 3.26).
- **Auto Scaling.** Según la demanda que se realiza en cada momento, el conjunto de servidores virtuales escala de forma automática.
- **Amazon Route 53.** Envía, siguiendo una ruta, el tráfico del dominio a un recurso. Este es un servicio web *Domain Name System (*DNS*)* en la nube escalable que tenga alta disponibilidad. Este servicio se ha diseñado para proporcionar un mecanismo muy fiable y rentable para dirigir a los usuarios a los sitios web, convirtiendo los nombres de dominio en direcciones IP numéricas que son usadas por los equipos para poder llevar a cabo una comunicación entre sí. AWS establece direcciones URL permitiendo a los usuarios que puedan recordarlas sencillamente.
- **AWS Lambda.** Pone en funcionamiento en servidores virtuales proporcionados por *Amazon* EC2 en respuesta a eventos.
- **Amazon EC2 Container Service (ECS).** Ofrece contenedores *Docker* en servidores virtuales.

Servicios de almacenamiento y entrega de contenido Ofrece un conjunto de servicios que permite el almacenamiento de datos y estos son los siguientes:

- **Amazon Simple Storage Service (S3).** Este es un servicio de almacenamiento de datos escalable que gestiona una gran volumen de datos y un gran número de accesos simultáneos. Estos datos son almacenados en contenedores llamados depósitos. Se pueden controlar los permisos que tiene cada usuario para cada contenedor. Un depósito puede almacenar cualquier número de objetos de cualquier tipo. El nombre que tiene asignado cada contenedor tiene que ser único globalmente. Los objetos pueden

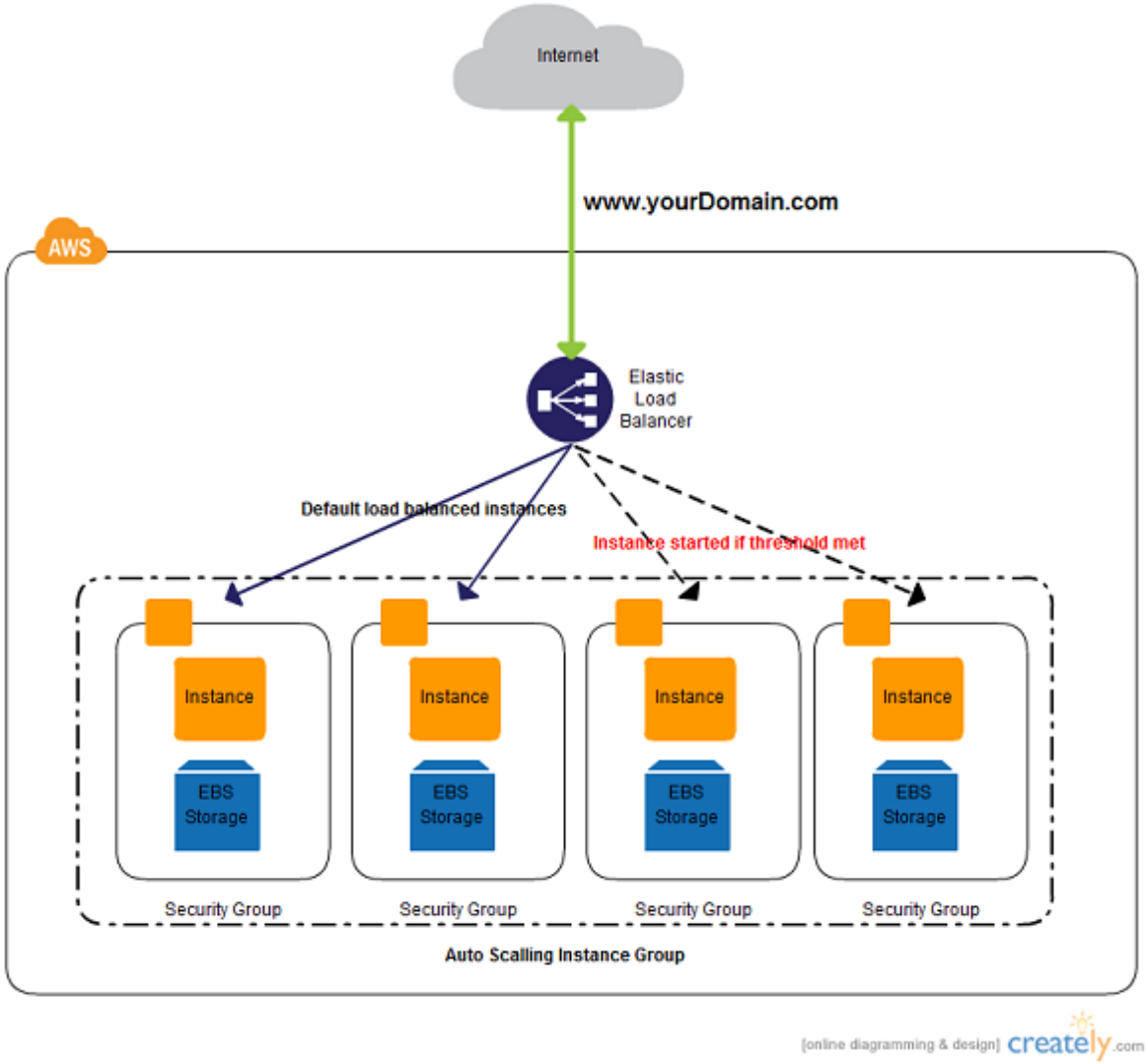

Figura 3.26: Arquitectura AWS con balanceo de carga (Fuente: http://creately.com)

ser agrupados en carpetas, que son incluidas en el nombre de la clave de un objeto. Se hace uso de una URL única para acceder a cada objeto.

- **CloudFront.** Proporciona una red que se encarga de la entrega de contenido global. Permite poner el contenido de un sitio web de un cliente a disposición de los centros de datos de todo el mundo, denominados ubicaciones de borde. Este contenido es almacenado en un servidor de origen. Cuando un usuario intenta acceder a un objeto que constituye una distribución de *CloudFront*, esta comprueba si el objeto se encuentra disponible en una memoria caché que esté cercana al usuario. Si se encuentra en memoria caché, CloudFront proporciona el contenido desde ahí, pero si no está, copia el contenido que se pide del servidor de origen a la caché.

- **Amazon Elastic Block Storage (EBS).** Proporciona volúmenes de almacenamiento que se encuentran conectado mediante la red para los servidores virtuales. Una vez

que se crea, se adjunta y se monta un volumen de este servicio en una instancia, se puede usar como una unidad física de un disco duro del equipo. Cada volumen solamente se puede adjuntar con una instancia de EC2, pero se puede separar uno de los volúmenes de una instancia y adjuntarlo a otra distinta. Varios volúmenes pueden ser adjuntados a una instancia y de esta forma se pueden distribuir los datos entre los distintos volúmenes.

Se pueden realizar una copia de seguridad de los volúmenes de *Amazon* EBS a través de la creación de instantáneas que son almacenadas en *Amazon* S3. Se puede crear un nuevo volumen mediante una instantánea y después adjuntarlo a una instancia de EC2.

- **Amazon Glacier.** Ofrece un almacenamiento que es duradero y cuyo coste es bajo. Es útil para realizar copias de seguridad y archivar datos. Para ellos se tiene que crear un almacén que va a consistir en un contenedor de archivos de almacenamiento. Un archivo de almacenamiento es un objeto que es almacenado en el almacén. Cada archivo va a tener un identificador único y una descripción que es opcional.

Servicios de seguridad e identidad Estos servicios proporcionan ayuda para proteger los datos y sistemas del usuario que se encuentren en la nube.

- **AWS Identity and Access Management.** Es un servicio que permite realizar la gestión de los usuarios y permisos de usuario en AWS. De esta forma se puede controlar el acceso de los usuarios a determinados recursos que se encuentran en AWS.
- **AWS Directory Service.** Permite al usuario conceder el acceso a AWS a usuarios y grupos de directorios. Se pueden crear nuevos directorios en la nube o conectar a directorios locales existentes.

Servicios de base de datos Proporciona servicios de bases de datos *NoSQL* y relacionales que se encuentran gestionados totalmente. También ofrece almacenamiento de caché en memoria como un servicio y una solución de *Datastore* de varios *petabytes*. Los servicios principales de este tipo que proporciona AWS son los siguientes:

- **Amazon Relational Database Service (RDS).** Ofrece base de datos relacionales que son administradas. Este tipo de base de datos se organiza en tablas que están relacionadas entre sí por valores de clave. Estas bases de datos son útiles si la aplicación usa combinaciones o transacciones complejas.

  Uno de los componentes básicos de este servicio es la instancia de base de datos, que es un entorno de base de datos que se encuentra aislado de la nube. Esta instancia puede estar formada por varias bases de datos. Cuando se pone en funcionamiento una instancia de este tipo, hay que indicar el motor de base de datos y una clase de instancia de base de datos, lo que especifica las capacidades de informática y memoria de esta

instancia. También se tiene que determinar el grupo de seguridad para esta. El *firewall* de la instancia impide que cualquiera pueda acceder a la base de datos en el caso que no se le haya concedido permiso en las reglas del grupo de seguridad.

Este servicio proporciona dos mecanismos para hacer copias de seguridad de las instancias de la base de datos y restaurarlas: las copias de seguridad automatizadas y las copias de seguridad iniciadas por el usuario.

- **Amazon Redshift.** Ofrece un *Datastore* que es rápido y totalmente administrado, con espacio de varios *petabytes*.
- **Amazon DynamoDB.** Proporciona bases de datos *NoSQL* que son administradas. Este tipo de base de datos son útiles para aplicaciones cuya función principal es indexar y consultar datos sin necesidad de realizar combinaciones ni transacciones complejas. Estas bases de datos proporcionan flexibilidad de esquema, escritura y lectura rápida, escalado ilimitado y alta disponibilidad. Las tablas de estas bases de datos al no tener esquemas, pueden ser utilizadas para almacenar documentos al estilo *JSON* o pares de clave-valor, lo que permite que sean ideales para gestionar datos estructurados o no estructurados.
- **Amazon ElastiCache.** Este servicio se encarga de proporcionar almacenamiento en memoria caché.

Servicios de análisis AWS proporciona varias herramientas de análisis que pueden escalar de manera económica y eficaz. Entre los distintos servicios disponibles se encuentran los siguientes:

- **Amazon EMR.** Para llevar a cabo el análisis usa *Hadoop*, que es una plataforma de código abierto con la que se puede realizar la administración y el procesamiento de los datos. *Hadoop* hace uso del motor *MapReduce* para repartir el procesamiento a través de un clúster. Este servicio de Amazon simplifica la instalación, configuración y gestión de *Hadoop*. *Amazon EMR* se encarga de gestionar los recursos informáticos y lanza el programa de *MapReduce* u ofrece herramientas para las consultas como *Hive* o *Pig*.
- **AWS Data Pipeline.** Este servicio hace más fácil el traslado y procesamiento de datos realizado periódicamente. El usuario tiene que crear una canalización que va a definir el origen de los datos de entrada, los recursos informáticos que se encargan de realizar el procesamiento, todas las condiciones que se tienen que cumplir para realizar para poder realizar cualquier procesamiento y la ubicación de los datos de salida.
- **Amazon Kinesis.** Se encarga de llevar a cabo el procesamiento de datos de transmisión en tiempo real a escala masiva.

- **Amazon Machine Learning (ML).** Hace posible que los desarrolladores usen de forma sencilla tecnología de aprendizaje de la máquina para conseguir predicciones para las aplicaciones desarrolladas mediante el uso de APIs sencillas. El funcionamiento de este servicio consiste en buscar patrones en sus datos existentes, crear modelos de aprendizaje de la máquina y usar estos modelos para procesar nuevos datos y producir predicciones.

Servicios de aplicaciones Es un conjunto de servicios administrados utilizados por las aplicaciones desarrolladas y pueden ser muy útiles. Entre ellos se encuentran:

- **Amazon AppStream.** Permite alojar la aplicación de *streaming* en la nube de AWS y transmitir por *streaming* la entrada y salida a los dispositivos de los usuarios.
- **Amazon CloudSearch.** Permite a la aplicación acceder a un sitio web mediante su búsqueda. Facilita la adición de de capacidades de búsqueda al contenido que se encuentra en un sitio web. Este servicio se encarga de analizar los datos y generar un índice de búsqueda, que ofrece los resultados de las solicitudes de búsqueda.
- **Amazon Elastic Transcoder.** Se encarga de transformar los medios digitales en los formatos que sólo permiten los dispositivos de los usuarios, permitiendo de esta manera la visualización de estos en varios dispositivos.
- **Amazon Simple Email Service (SES).** Permite el envío de correo electrónico desde la nube. Ofrece una manera fácil y rentable de enviar una gran cantidad de correos electrónicos. También se encarga de recopilar métricas sobre los mensajes que se han entregado, devuelto, rechazado y marcado como correo no deseado y ofrece acceso a las métricas en tiempo real.
- **Amazon Simple Notification Service (SNS).** Se encarga de que los usuarios envíen y reciban notificaciones desde la nube. Una notificación hace que un destinatario pueda saber el momento en el que ha ocurrido un evento. La información es entregada al destinatario de forma automática, por lo que no se tiene que comprobar si hay disponibles mensajes nuevos y recuperarlos. Este servicio se encarga de coordinar y administrar la entrega de los mensajes a los destinatarios. El publicador se va a encargar de producir mensajes y los envía al tema relacionado, que define un protocolo de mensajes y la lista de los destinatarios. El consumidor se subscribe a un tema y recibe los menajes que están relacionados con ese tema.
- **Amazon Simple Queue Service (SQS).** Permite que los componentes que tienen las aplicaciones alojadas almacenen datos en una cola para que puedan ser usados por otros componentes. Una cola contiene mensajes y permite coordinar el trabajo de los procesos de forma asíncrona. Estos procesos se van a encargar de leer, escribir y eliminar los mensajes que se encuentran en la cola, además de llevar a cabo el trabajo

dependiendo de la información que contiene los mensajes que son recuperados. Mediante el uso de colas, las tareas se realizan de forma independiente y se comunican entre sí cuando hay que realizar algún trabajo en vez de realizarlas de forma secuencial, esperando a que una termine para empezar la siguiente.

Este servicio garantiza que cada mensaje se entregue al menos una vez, admite la lectura y escritura de varios procesos en la misma cola y permite controlar los usuarios que pueden leer y escribir en la cola. Cada cola tiene que tener un nombre único y una URL única.

- **Amazon Simple Workflow Service (SWF).** Se encarga de la coordinación de las tareas entre los distintos componentes de la aplicación alojada. El motor de flujo de trabajo se encarga de coordinar el trabajo entre los distintos componentes que se encuentran distribuidos. El flujo de trabajo es un conjunto de actividades y la lógica que coordina esas actividades. Este servicio sirve como una ubicación central del flujo de trabajo y conserva el estado de cada flujo de trabajo.

Windows Azure

Es una plataforma en la nube abierta y flexible que proporciona *Microsoft* y que proporciona un conjunto de servicios distintos [Tul13]. *Azure* permite al usuario crear, implementar y administrar aplicaciones rápidamente mediante una red global de centros de datos que son administrados por este. El usuario puede crear aplicaciones usando cualquier lenguaje, herramienta o *framework*.

Características

Las características principales de *Windows Azure* son las siguientes [Kri10]:

- **Servicio de alojamiento.** Se pueden construir servicios que están alojados en *Windows Azure*. Los servicios hacen referencia a cualquier aplicación genérica del lado del servidor.
- **Servicio de gestión.** En *Windows Azure*, el usuario puede usar las capacidades integradas en el servidor de *Windows* y en otros productos para hacer frente a diversas tareas operacionales como el manejo de las actualizaciones de la aplicación, monitorizar la aplicación, reemplazar el hardware fallido. Permite monitorizar los servicios del usuario, tratar los fallos software y hardware y manejar el sistema operativo y la aplicación actualizada sin problemas.
- **Almacenamiento.** Proporciona almacenamiento escalable donde el usuario puede almacenar datos. Ofrece tres servicios clave: almacenamiento de objetos grandes binarios, tablas semiestructuradas y un servicio de cola. Estos servicios tienen una API *HTTP* que permite al usuario acceder a los servicios desde cualquier lenguaje y desde

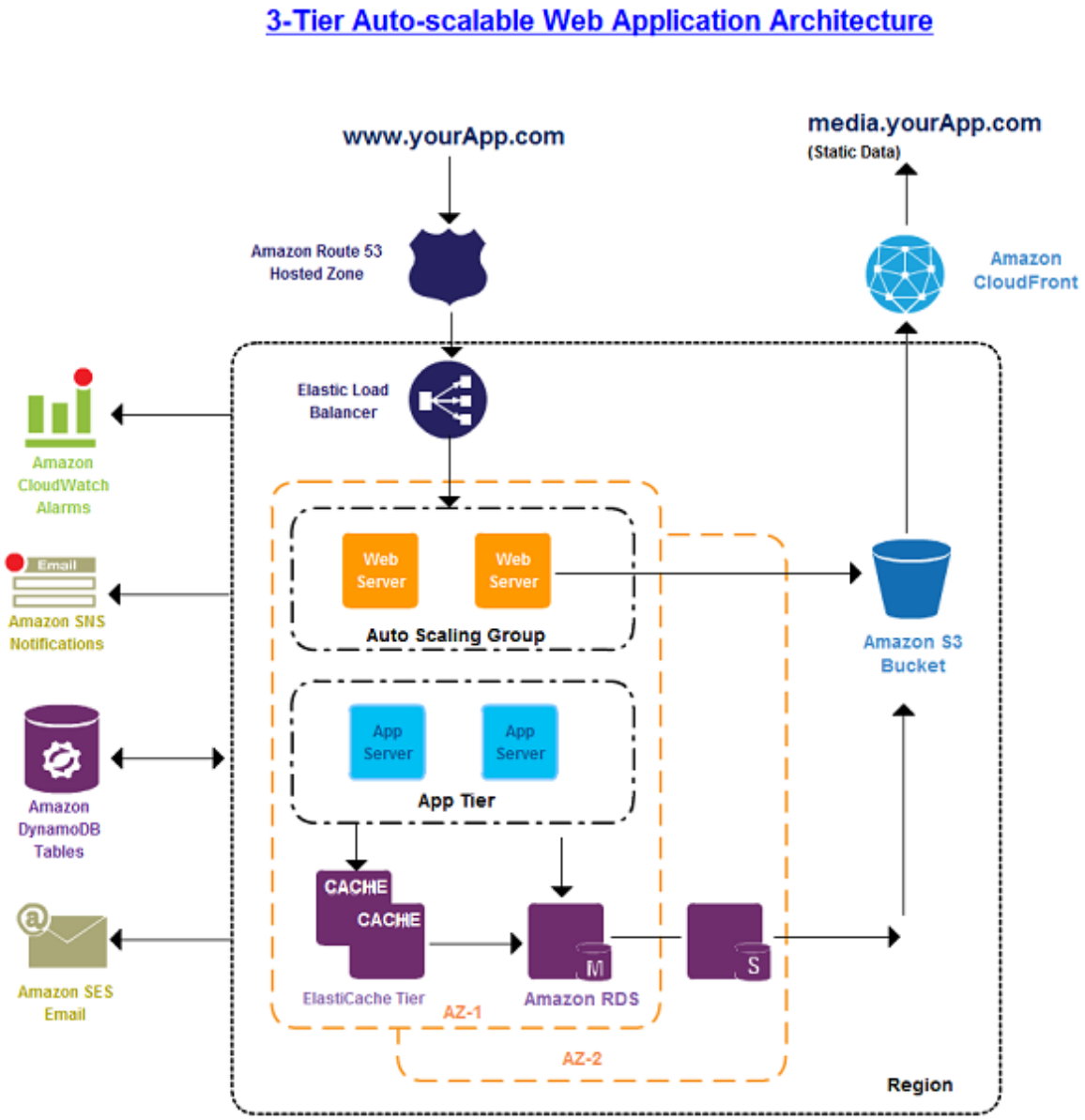

Figura 3.27: Arquitectura AWS (Fuente: http://creately.com)

fuera del centro de datos de *Microsoft*. Los datos que se encuentran almacenados con estos servicios son replicados varias veces para proteger los fallos de software y hardware. El almacenamiento se basa en un modelo de consumición donde el usuario paga solamente por los que usa.

- **Servidor de Windows.** El usuario no necesita aprender un nuevo *framework* ni el código de la aplicación va a ser distintos porque está funcionando en la nube. El usuario puede seguir implementando código que se pueda ejecutar en el servidor de *Windows*. El *framework .NET* está instalado por defecto en todas las máquinas y el código del usuario *APS.NET* trabajará.
- **Herramientas de desarrollo.** *Windows Azure* tiene un conjunto de herramientas pa-

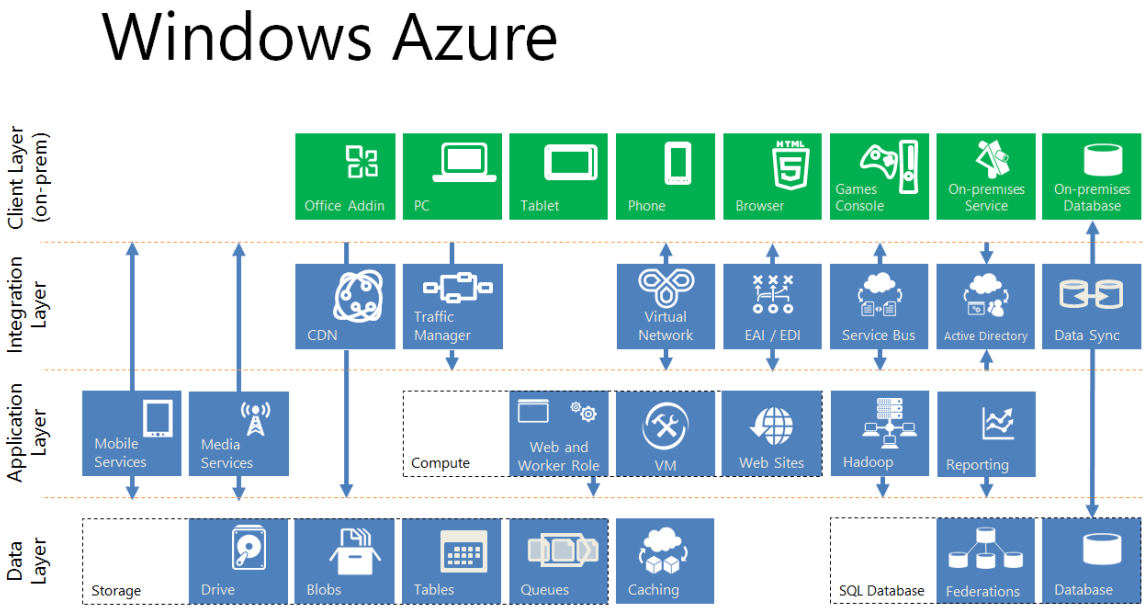

Figura 3.28: Componentes de *Microsoft Azure* (Fuente: Microsoft)

ra desarrollar fácilmente. Tiene una API que el usuario puede utilizar para registrar y reportar errores, como mecanismos para leer y actualizar los ficheros de configuración del servicio. También tiene un SDK que permite desplegar las aplicaciones en un simulador de la nube.

Servicios

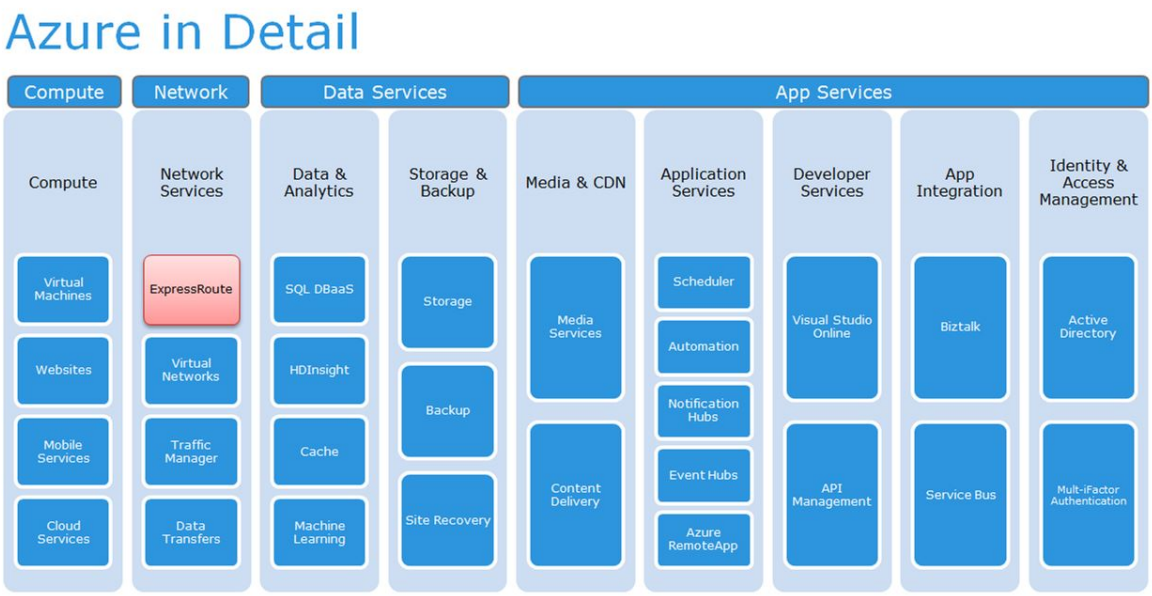

Figura 3.29: Servicios ofrecidos por *Microsoft Azure* (Fuente: https://msftdude.files.wordpress.com)

Servicios de cómputo Estos proporcionan la capacidad de procesamiento necesaria para que puedan ejecutarse las aplicaciones que están alojadas en la nube. Windows Azure ofrece los siguientes servicios de este tipo:

- **Máquinas virtuales.** Proporciona un entorno informático que permite al usuario crear, implementar y administrar máquinas virtuales que se ejecutan en los servidores de *Windows*.
- **Sitios web.** Ofrece un entorno web administrado que se usa para crear sitios web nuevos o migrar un sitio web existente a la nube.
- **Servicios en la nube.** Permite crear e implementar aplicaciones altamente disponibles y escalables con bajos costos de administración utilizando cualquier lenguaje de programación.
- **Servicios móviles.** Permite crear e implementar aplicaciones y almacenar datos para aplicaciones móviles.

Servicios de red Ofrecen distintas opciones que establece la manera en las que las aplicaciones se pueden entregar a los usuarios y los centros de datos. Los servicios ofrecidos de este tipo son los siguientes:

- **Red virtual.** Trata la nube pública de Windows Azure como si se tratase de una extensión de su centro de datos local.
- **Administrador de tráfico.** Permite enviar siguiendo una ruta el tráfico de la aplicación al usuario que está usando la aplicación en los centros de datos de Windows de tres formas.

Servicios de datos Estos proporcionan distintas formas de almacenar, administrar, proteger, analizar e informar datos empresariales.

- **Administración de datos.** Permite almacenar datos empresariales en bases de datos SQL.
- **Análisis de negocio.** Facilita el descubrimiento y el enriquecimiento de datos.
- **HDInsight.** Es el servicio que está basado en *Hadoop* de *Microsoft*.
- **Memoria Caché.** Proporciona una solución de almacenamiento en memoria caché distribuida que ayuda a acelerar las aplicaciones basadas en la nube y reducir la carga de la base de datos.
- **Backup.** Ayuda a proteger los datos que se encuentran en el servidor mediante el uso de copias de seguridad automatizadas y manuales.

- **Administrador de recuperación.** Ayuda al usuario a proteger los servicios críticos para la empresa al coordinar la replicación y recuperación de las nubes privadas en una ubicación secundaria.

Servicios de la aplicación Ofrecen al usuario maneras de mejorar el rendimiento, la seguridad, la detección y la integración de las aplicaciones que se están ejecutando en la nube.

- **Servicios de medios.** Permite crear flujos de trabajo para la creación, administración y distribución de medios usando la nube pública.
- **Mensajería.** Esta formada por dos servicios que permiten mantener las aplicaciones conectadas mediante un entorno de nube pública y privada.
- **Hubs de notificación.** Proporciona una infraestructura de notificación muy escalable y multiplataforma para aplicaciones que son ejecutadas en dispositivos móviles.
- **Servicios BizTalk.** Proporciona capacidades *Business-to-Business* y *Enterprise Application Integration* para ofrecer soluciones de integración híbridas y en la nube.
- **Directorio Activo.** Proporciona capacidades de administración de identidades y control de acceso a las aplicaciones que se encuentran en la nube.
- **Autenticación multifactor.** Proporciona una capa más de autenticación para garantizar un acceso más seguro para aplicaciones.

Capítulo 4

Método de Trabajo

En este capítulo se va a hablar sobre las metodologías de desarrollo existentes y la que se aplica en este Trabajo Fin de Grado(TFG) y el motivo que ha hecho que se emplee esa metodología.

Metodologías de desarrollo

Antes de empezar un proyecto, se debe de elegir la metodología que se va a seguir, entre todas las existentes, rigiéndose por la que pueda ser más útil para este tipo de proyecto. Una metodología de desarrollo es **el uso de unas determinadas técnicas, herramientas, modelo y métodos para llevar a cabo el desarrollo de software de manera eficaz**. Mediante el uso de estas metodologías se estructura, planifica y controla el proceso de desarrollo.

Existen diversas metodologías de desarrollo y se dividen en dos enfoques: **las metodologías tradicionales y las ágiles**.

Las metodologías tradicionales se basan en **un ciclo de vida para el desarrollo del software en cascada**, debido a que los proyectos se organizan en fases que se van ejecutando de manera secuencial y cada fase solo se puede ejecutar una vez, lo que implica que hasta que no finaliza con éxito esta, no se puede pasar a la siguiente fase. El lado bueno es que en cada fase se encargan distintos profesionales especializados en ella. El lado negativo de usar este tipo de metodología es que, si nos damos cuenta en una etapa avanzada que algo anteriormente definido está mal, volver hacia atrás para corregirlo tiene un alto costo y el usuario no puede ver cómo va avanzando el producto para ir validando hitos e ir verificando si lo que se ha construido es lo que es necesario, porque solo puede ver el producto al final, una vez finalizado. Las metodologías tradicionales se centran en la documentación, la planificación y los procesos. Entre las más conocidas de este enfoque son *Rational Unified Procces (*RUP*)* y *Microsoft Solution Framework (*MSF*)*.

Las metodologías ágiles se basan en **un ciclo de vida para el desarrollo del software iterativo e incremental**, lo que significa que se repiten las fases en cada ciclo, se añade nueva funcionalidad al producto y se comprime todo lo posible el tiempo de las iteraciones. El usuario puede ir validando si el producto cumple con los requisitos que se piden mediante entregas parciales del producto. Algunas veces se pueden solapar algunas etapas. En vez de

usar la documentación como en las metodologías tradicionales, se cambia por la interacción con el usuario y no hay separación de roles, evitando de esta forma que haya expertos solo de una fase, pudiendo ejercer su trabajo en cada etapa. El problema que existe con este tipo de metodología es que se entre en un bucle de entrega de prototipos y no se cierre el proyecto nunca y existe menos holgura para cometer fallos. Entre las más conocidas de este enfoque son *Extreme Programming (*XP*)* y *Scrum*.

Metodología utilizada

La metodología usada en este proyecto es XP. Como se ha dicho anteriormente, es una metodología ágil que se centra en mejorar las relaciones entre las personas para obtener el éxito en cuanto al desarrollo del software. Es idónea para proyectos que tienen requisitos imprecisos y muy cambiantes. Las ventajas que ofrece esta metodología son que suele haber menos errores, satisface al programador y la programación que se lleva a cabo es organizada, pero es aconsejable usarlo en proyectos a corto plazo. La programación mediante el uso de esta metodología permite obtener resultados de manera rápida.

Esta metodología depende de cuatro variables para llevarse a cabo, que son:

Coste. Este se refiere al dinero que se necesita para tener acceso a los recursos que van a permitir desarrollar el proyecto.

Tiempo. El tiempo necesario para realizar el proyecto, siendo mayor si se quiere hacer con mejor calidad y más alcance.

Calidad. Es el cumplimiento de los requisitos de la mejor forma posible.

Alcance. Es lo que abarca un proyecto, y cuanto mayor sea, más coste y tiempo necesitará.

Esta metodología está fundamentada en un conjunto de valores y principios que marcan el camino. Los valores representan los aspectos que se han considerado como imprescindibles para obtener el éxito de un proyecto. Los valores son:

Comunicación. Es de distintas formas. La comunicación del código es mejor para los desarrolladores cuanto más simple sea. El código que está autocomentado es más fiable que los comentarios porque estos pueden quedar desfasados según se vaya modificando, por lo tanto, sólo se debe comentar lo que no va a variar a través de las modificaciones aplicadas al código. Las comunicaciones de las pruebas unitarias describen el diseño de las clases y los métodos. Los programadores se comunican mediante la programación por parejas. La comunicación con el cliente es fluida debido a que se encuentra dentro del equipo de desarrollo y va a ser el que decida qué objetivos tienen prioridad y el que resuelva dudas.

Simplicidad. El diseño del código es simplificado para hacer más ágil el desarrollo y más fácil el mantenimiento. Un diseño complejo con sucesivas modificaciones reali-

zadas por distintos desarrolladores provoca en aumento de la complejidad de manera exponencial. Para que se mantenga la simplicidad mientras que va creciendo el código, se tiene que refactorizar este. Esto también se aplica a la documentación evitando el exceso de comentarios. Este valor hará que cuando el código crezca demasiado, todos los miembros del grupo comprendan de manera sencilla y rápida el código que han realizado sus compañeros.

Realimentación. Como el cliente forma parte del equipo del proyecto, se conoce en su opinión sobre el estado del proyecto en el mismo momento. Como se realizan ciclos muy cortos donde se muestran resultados, se disminuye el tener que volver a hacer las partes que incumplen con los requisitos y ayuda a que los programadores se centren en las partes más importantes.

Coraje. Permite que los desarrolladores se sientan cómodos al reconstruir su código si es necesario, lo que significa revisar el sistema existente y modificarlo si los cambios que se van a aplicar en el futuro pueden aplicarse más fácilmente. También quitando código obsoleto sin importar el esfuerzo y tiempo invertido para llegar a ese código.

Algunos añaden **la humildad** como quinto valor, que permite que varios miembros del equipo se respeten unos a otros y respeten el trabajo de los demás. Los principios están apoyados en los valores y son la realimentación veloz, las modificaciones incrementales, el trabajo de calidad y la asunción de simplicidad.

Motivo

Se usa en este TFG esta metodología porque se considera la más adecuada, debido a que existe retroalimentación con el cliente, que en este caso es el profesor, que propone ideas sobre más cosas que puede tener este proyecto y valida el cumplimiento de los requisitos. También es muy útil para este caso, porque en este proyecto se parte de una idea principal, la solución de un problema, pero a lo largo del desarrollo se añade nueva funcionalidad y mejoras relacionadas con el problema. Por ese motivo, el código está organizado y estructurado en módulos para hacerlo más legible y permitir fácilmente la modificación de código evitando errores. Además, esta metodología es aconsejable para este tipo de trabajo, debido a que se trata de un proyecto a corto plazo.

Medios hardware y software

A continuación, se muestra los componentes hardware y software necesarios para la realización de la fase de desarrollo del proyecto.

Medios hardware

En primer lugar, el dispositivo utilizado para la realización de todo el proyecto es **un ordenador portátil con unas prestaciones adecuadas**. El computador es un *HP Pavilion*

dv6 con un microprocesador *Intel Core i7* de 2,3 *GHz*, 8 *GB* de memoria *RAM*, un disco duro *SATA* con una capacidad de 1 *TB* y *NVIDIA GeForce GT 630M*. También se ha necesitado conexión a Internet para poder acceder a varios recursos que se encuentran alojados en las nubes.

Para llevar a cabo el despliegue y alojamiento de la aplicación se han usado los servidores que *Google* proporciona.

Medios software

Se han usado varias herramientas software para la realización de las distintas tareas que hay que llevar a cabo a lo largo del proyecto.

Para la implementación del código se usa como lenguaje de programación **Python** en su versión 2.7 y el editor de texto **Sublime Text** para la parte del servidor y del cliente usado para las pruebas iniciales y el lenguaje **Android** y la herramienta **Android Studio** para la parte del cliente utilizado para las pruebas finales. Para llevar a cabo con éxito esta tarea, se ha tenido que usar librerías que proporciona **Google App Engine** y uno de los *frameworks* de desarrollo web que usa, **WebApp2**. Hay que destacar que para establecer la conexión entre el cliente y servidor *Python* que se ha usado para la prueba inicial, se ha utilizado la librería de *Python* llamada **Requests**. En las pruebas finales, donde el cliente está desarrollado en Android, se usa **Google Cloud Endpoints** para establecer esa interacción entre las dos partes.

Para organizar el proyecto se ha empleado **Trello** para la parte de gestión de las tareas y el seguimiento del proyecto y para la parte de organización del código implementado se ha usado el sistema de control de versiones **Mercurial** y **Bitbucket**.

Para generar la documentación del proyecto se utiliza LaTeX y el editor de texto que trae por defecto *Linux*, **Gedit**.

El Sistema operativo (SO) utilizado para realizar todos los procesos es **Linux Ubuntu**, debido a su comodidad.

Para organizar el proyecto, se ha empleado **Trello** para realizar la gestión de las tareas y el seguimiento del proyecto.

Capítulo 5

Arquitectura del sistema

En este capítulo se va a realizar un análisis de las diferentes partes que forman el proyecto. En cada sección se va ofrecer una visión sobre todas las partes que forman el sistema, empezando por una visión más general, sin entrar en demasiados detalles técnicos, terminando por una más específica.

Primero se va a hablar sobre la arquitectura que forma el sistema, proporcionando una visión general de este y describiendo cada uno de los componentes, que va a ayudar a entender con facilidad el funcionamiento del sistema. Después se van a describir los problemas con los que se ha tenido que enfrentar algunas de las funcionalidades y las soluciones aplicadas a cada uno. Por último, para comprender mejor el proyecto, las últimas secciones incluyen material que va a ser complementario de las anteriores secciones. En la sección 5.7.1, se muestra cómo se estructura el proyecto en directorios y los componentes hardware que forman el sistema y hacen posible el funcionamiento de este. En la sección 5.7.2, se muestra como está organizado el Datastore en distintos modelos con sus respectivos atributos y las relaciones que existen entre ellos.

Visión general de la arquitectura

El sistema está dividido en dos partes: **el cliente y el servidor**. Esta arquitectura consiste en la comunicación de dos aplicaciones que se pueden encontrar en cualquier lugar geográficamente separados y los procesos de cada aplicación se realizan en distintos tipos de máquina. El programa cliente se encarga de pedir unos servicios al programa servidor y este se los proporciona. Se ha empleado este tipo de arquitectura porque es actualmente **el más utilizado e importante si se trata del envío y recibimiento de información**. También permite el guardado de los datos en una base de datos que se encuentra en la parte del servidor, lo que lo hace mucho más útil. El cliente es el que solicita las acciones y espera las respuestas que conllevan la realización de esas acciones. El servidor se encarga de llevar a cabo las acciones que el cliente pide y va a generar las respuestas. La arquitectura de ambas partes del sistema implementado está formada por diversas capas, y van desde un nivel alto de abstracción hasta el nivel más bajo.

La parte del cliente está formada por dos capas:

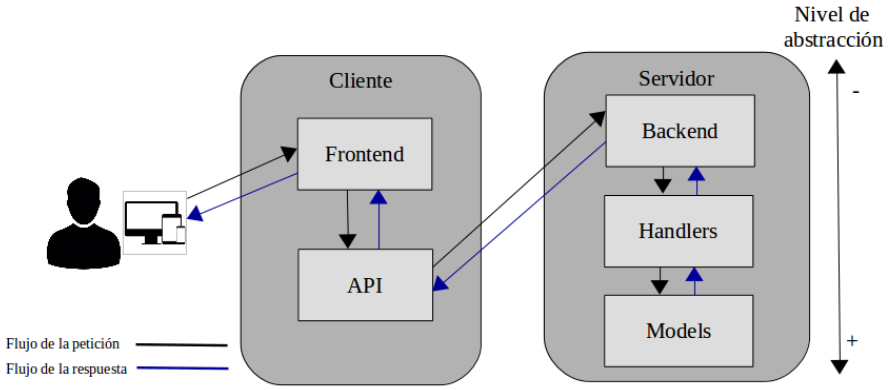

Figura 5.1: Arquitectura del sistema

- **Frontend:** Los usuarios interactúan con esta capa para llevar a cabo una serie de acciones en la aplicación. En la capa más directa con el usuario. También se encarga de mostrar al cliente el mensaje que contiene la respuesta que ha sido enviada desde el servidor a la capa inferior. Mediante ese mensaje se puede comprobar si la acción se ha aplicado con éxito o ha fallado.
- **API:** Se encarga de obtener los datos que han sido introducidos por el usuario en la capa superior y de enviar las solicitudes, que contiene los datos recogidos, al servidor para realizar una acción. Más tarde, una vez que haya llevado a cabo la acción el servidor, esta capa se encarga de recibir las respuestas provenientes del servidor.

Y la parte del servidor está constituida por tres capas:

- **Backend:** Es la capa que se encarga de administrar el sistema que se encuentra alojado en el servidor.
- **Handlers:** Se encarga de recibir las peticiones enviadas por el cliente, de aplicar las acciones adecuadas y de enviar las respuestas a los clientes.
- **Models:** Se encarga de administrar la base de datos del sistema, aplicando cambios a los datos o proporcionándolos cuando se les pide.

El usuario transmite la información mediante el *frontend*. Este pasa toda la información a su capa inferior, que es la *API*, que se encarga de recoger e introducirla de forma adecuada en una petición, que se envía a servidor. El servidor recibe la petición mediante su capa superior, que es el *backend*. El *backend* pasa toda la información al *handler* adecuado para que se encargue de procesar la información. El *handler* envía al *model* la información obtenida para que la almacene en la base de datos. El *model* indica al *handler* si ha podido gestionar la información y el *handler* indica al *backend* si ha podido llevar a cabo con éxito el procesamiento. El *backend* envía una respuesta a la API del cliente, que se encarga de procesar la respuesta. Por último, la API envía al *frontend* la respuesta procesada para que se la muestre

al usuario.

Es bueno dividir en las siguientes capas el sistema porque permite **hacer una distinción de cada una y separar cada funcionalidad**. Cada capa se encarga de hacer su función, lo que permite que la realice de manera eficaz al estar especializado en realizar solamente esa función. Esta división permite ver el sistema como módulos, de tal forma que, con verlo a simple vista, se sabe lo que realiza cada parte sin tener que verlo internamente al detalle, permitiendo hacer más fácil la modificación del sistema. La separación en capas en cuanto al desarrollo del sistema hace que si se producen cambios **solo se tenga que modificar la capa necesaria**, sin tener que mirar todo el código disponible del sistema. Además, permite que se pueda trabajar sólo en una capa sin que se tenga que tener en cuenta las demás. Esto permite que la arquitectura del sistema sea escalable.

Autenticación del usuario

La información que contiene el ranking almacenado tiene que **estar vinculada a un usuario**, que ha hecho que esta haya cambiado mediante la realización de una serie de acciones. Lo que quiere decir que cada usuario va a tener un rango y una puntuación asignada.

Para que el jugador pueda acceder a esta información para verla o modificarla, se tiene que hacer uso de un mecanismo de autenticación mediante el cual pueda identificarse. De esta manera, otro usuario no va a poder modificar la información correspondiente a otro usuario.

Por este motivo, se implementa un mecanismo para la autenticación que otorgue una gran seguridad, debido a que se gestiona el almacenamiento de credenciales.

Métodos de identificación

Para la realización de este proyecto, se ha preferido darle menos importancia al mecanismo de autenticación. Por este motivo, se usado un mecanismo de autenticación propio, en el que el usuario puede introducir un nombre de usuario y una contraseña, permitiendo también de esta manera que un usuario pueda llegar a tener varias cuentas. Este mecanismo va a ser muy útil a la hora de realizar las pruebas en las que se tienen que hacer uso de una gran cantidad de clientes.

También se ha incorporado en la versión *Android* la autenticación mediante una cuenta de *Google*, para si existe algún usuario que no quiera registrar una nueva cuenta y de esta forma no tenga que crear nuevas credenciales de acceso o para que en un futuro se pueda aprovechar para hacer uso de las ventajas que proporcionan las aplicaciones de *Google*.

Autorización mediante OAuth2

OAuth2 es un protocolo de autorización mediante en el que un usuario permite **el acceso de un tercero**, que sería en este caso el sistema implementado en este proyecto, a contenidos

que se encuentran protegidos mediante credenciales que son gestionados por un servidor que está formado por recursos de confianza, sin que el usuario tenga que dar sus credenciales al tercero para que pueda tener acceso a los contenidos. Esto sería aprovechado en este trabajo para poder acceder al nombre de usuario que tiene asignado el usuario en su cuenta de *Google*.

Para poder utilizar este mecanismo hay que **registrar el dominio de este sistema en el proveedor de recursos en el que se quiera acceder**, que en este caso sería *Google*. Con esto se puede registrar un cliente, el cual puede obtener **un identificador y una clave secreta** con lo que el sistema puede ser identificado por el proveedor como un cliente válido.

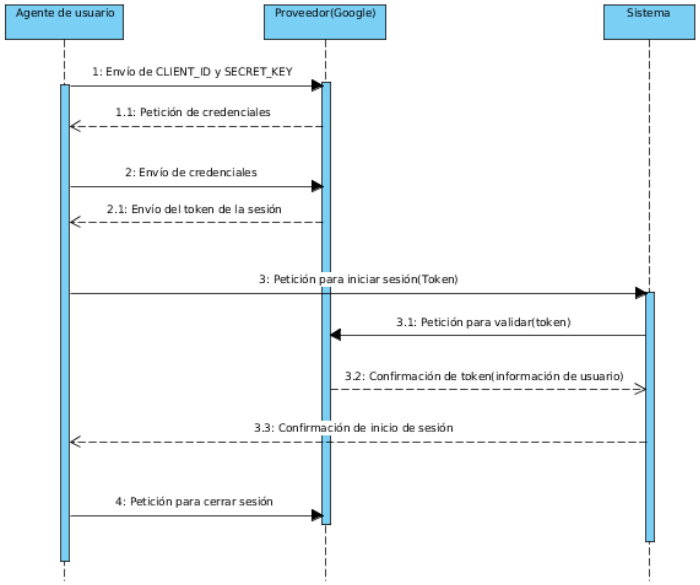

Figura 5.2: Diagrama de secuencia del funcionamiento de OAuth2

En la figura 5.2 se muestra las secuencia de acciones que realizan los tres actores que se encuentran implicados. Estos son el cliente que va a acceder al sistema, que es donde se encuentra el agente del usuario que representa la parte del *frontend* del sistema; el proveedor, que en este caso es Google; y la aplicación que se encuentra en el servidor que representa la parte del *backend*.

- El proceso inicia cuando el cliente elige iniciar sesión con una cuenta Google e inicia el procedimiento de inicio de sesión. En este momento, el agente del usuario envía el identificador y la clave secreta que tiene asignadas el sistema como cliente al proveedor. Si lo reconoce como un cliente válido, sigue el proceso.
- Mediante el agente de usuario que tiene asignado el jugador, se accede al servicio que proporciona el proveedor. Con él se puede interactuar mediante la introducción de sus credenciales. Una vez que el proveedor recibe las credenciales, comprueba que la autenticación de los parámetros que ha recibido es correcta y que el jugador muestra

que está conforme para ofrecer al sistema los permisos que se piden, y si se cumple, el proveedor envía al agente de usuario un *token*. Este va a representar la sesión que ha sido iniciada con el proveedor.

- A continuación, el agente de usuario se encarga de enviar el *token* al servidor del sistema. Este *token* recibido tiene que ser validado para asegurar que ha sido generado por el proveedor en vez de desde la parte del cliente. Para realizar esta comprobación, el servidor envía una petición que contiene la clave secreta asignada al proveedor. De esta manera se confirma la validez. Si esa clave es válida, se envían al servidor los datos correspondientes del jugador, a los que puede acceder. Por último, se puede iniciar sesión en la aplicación.
- Una vez realizado esto, el agente de usuario envía una petición al proveedor para que cierre la sesión que ha sido creada en el servicio que proporciona el proveedor, debido a que no se tiene que solicitar más información una vez que se ha completado el inicio de sesión.

Soporte para la creación de usuarios y gestión de la sesión

Para poder gestionar las puntuaciones correspondientes al ranking es necesario **gestionar a los usuarios y las sesiones de estos**, debido a que cada puntuación corresponde a un usuario que ha iniciado sesión y para cada acción que se solicite sobre ella se tiene que saber con seguridad el usuario que la aplica. Por este motivo, se tiene que añadir un sistema de autenticación, que se muestra en la sección 5.2, la creación de usuarios dentro del sistema y un sistema que proporcione una sesión a cada usuario identificado.

La solución para este problema propuesto va a consistir en utilizar **los módulos *auth* y *session* que contiene el *framework WebApp2*** usado en la parte del *backend* para permitir la asignación de una sesión a cada jugador, la definición de roles que limiten el acceso de algunas acciones y el inicio de sesión.

Para poder llevar a cabo la gestión de sesiones y el manejo de peticiones, se han añadido a este sistema **un conjunto de clases que forman la base y utilizan los módulos *session* y *auth*** que proporciona *WebApp2*.

Creación de usuarios

Para almacenar la información correspondiente al usuario, se define un modelo llamado ***User*** que va a heredar de la clase *auth.models.User*. La información proveniente de este modelo se establece cuando **el usuario se registre o inicie sesión la primera vez** en esta aplicación. Este modelo va a contener la siguiente información:

- **ID.** El identificador del usuario que va a ser generado por el sistema automáticamente.
- **Nombre de usuario.** Es el nombre con el que el usuario quiere que se le identifique

dentro del sistema. En el caso de que se use una cuenta de Google, este nombre va a ser el que tiene puesto en esa cuenta. En cambio, si se usa la autenticación propia del sistema, el usuario indica el nombre que quiere cuando se registra en la aplicación.

- **Contraseña.** Esta es la clave privada que el usuario tiene que introducir para entrar al sistema junto con su nombre de usuario. Si se usa una cuenta de Google, la contraseña no se almacena por seguridad y no es necesaria para iniciar sesión, ya que usa una cuenta ajena al sistema. Si usa una cuenta propia del sistema, esta se almacena cuando un usuario se registra.
- **Identificador del ranking.** Esto es un número que muestra en qué ranking esta almacenada la puntuación del usuario. Este se obtiene de manera automática viendo el número de puntuaciones que tiene almacenadas cada ranking y eligiendo el ranking que menos puntuaciones tenga, para así repartir la carga entre los distintos rankings. En la sección 5.5.2 se va a hablar sobre esto.
- **Rol.** Este es el papel que ejerce el usuario en este sistema. Dependiendo del rol del usuario, va a poder realizar un conjunto específico de acciones. Este valor por el momento va a ser igual para todos los usuarios, pero puede ser útil en el futuro para desarrolladores que hagan uso de este sistema.

Este sistema de autenticación puede ser **rápido** debido a que el usuario solamente tiene que introducir el nombre de usuario y la contraseña en el caso de que use una cuenta propia del sistema, los demás atributos se establecen automáticamente. En el caso de que el usuario utilice la cuenta de *Google*, no tiene que introducir ningún dato, debido a que la información que no se establece automáticamente, la proporciona directamente el proveedor.

Los casos de inicio de sesión y registro de un nuevo usuario son iguales desde el punto de vista del cliente, por lo que se podrían juntar, pero se ha preferido que haya una distinción para **facilitar la modificación del sistema por parte de otro desarrollador en un futuro**.

Procedimiento para la gestión de las sesiones

Una vez que se ha mostrado los mecanismos que se han implementado para la autenticación y las consideraciones necesarias para llevar a cabo la creación de nuevos usuarios en el sistema, se ha va a definir los recursos que son necesarios para gestionar las sesiones:

- **/auth/signin.** Este recurso permite iniciar sesión al usuario que ha sido previamente registrado en la aplicación en el caso de que haya introducido este las credenciales de manera correcta.
- **/auth/signup.** Cuando un nuevo usuario desea registrarse en el sistema, este recurso se encarga de comprobar que no existe un usuario ya registrado que tenga el mismo nombre de usuario y contraseña. Si no lo tiene, calcula de manera automática algunos

datos del usuario y lo almacena en la base de datos del sistema. Por último, inicia sesión.

- **/auth/signout.** Mediante este recurso el usuario cierra la sesión.
- **/auth/signinGoogle.** Este recurso es utilizado para los usuarios que utilizan una cuenta de Google. Se encarga de comprobar si es la primera vez que el usuario a iniciado sesión para almacenar en la base de datos los datos de este si ocurre eso.

Soporte para la gestión de rankings

Para poder gestionar de manera óptima los rankings que utiliza el sistema se usa **una librería proporcionada por *Google*** cuyo funcionamiento se expone en la sección 3.2.2. Como se ha mencionado anteriormente, el ranking es almacenado en una estructura de datos que representa **un árbol**. Este árbol va a estar representado por la definición de varios modelos del *Cloud Datasore*, concretamente tres:

- **Ranker.** Este modelo representa toda la estructura del árbol, es decir, el ranking completo. Va a contener los siguientes atributos:
  - **Factor de ramificación.** Es el número de nodos hijos que va tener cada nodo rama.
  - **Rango de puntuaciones.** Es el conjunto de puntuaciones que pueden tener los jugadores. Este atributo sirve para limitar la puntuación de los jugadores, lo que permite que los jugadores ya no puedan aumentar su puntuación si se alcanza el límite.
- **RankerNode.** Este tipo de modelo representa cada nodo rama del árbol. Está formado por el siguiente atributo:
  - **Número de hijos.** Es el número de usuarios que tienen una puntuación que se encuentra comprendida entre el rango de puntuaciones que representa el nodo.
- **RankerScore.** Este modelo representa cada nodo hoja del árbol, es decir, que en el caso del ranking sería a un jugador y contiene los siguientes atributos:
  - **Valor.** Es la puntuación del jugador que representa el nodo.

Mediante el uso de esta librería **se agilizan todas las actividades relacionadas con la gestión de rankings** debido al uso del árbol como estructura de datos, que es la adecuada y más eficaz para la búsqueda de datos.

Procedimiento para la gestión de rankings

Una vez que se ha explicado cómo está formado el mecanismo para gestionar los rankings, se define los recursos con los que está formado el sistema para llevar a cabo este tipo de gestión:

- **/rank/setScore.** Este recurso se encarga de establecer una puntuación al usuario que ha iniciado la sesión. Primero obtiene la puntuación que tiene el usuario establecida y calcula la nueva puntuación que va a tener el jugador. Comprueba si todavía no tiene ninguna puntuación establecida y si se cumple, establece como nueva puntuación los puntos que se le envían. Si ya tiene una puntuación asignada, se le suma los puntos que se le envían para obtener la nueva puntuación si la anterior puntuación no ha alcanzado el número máximo de puntos que puede tener un jugador en el sistema. Si la nueva puntuación obtenida es mayor que el máximo número de puntos, se establece como nueva puntuación el límite de puntos permitidos del sistema. Si se ha obtenido una nueva puntuación, se envía a una *pull queue* una tarea, para que establezca la puntuación del usuario, que va a contener los datos necesarios para llevarlo a cabo. Sobre esto se va a discutir en la sección 5.5.1. Si se va a aplicar la nueva puntuación, finalmente va a estar esperando hasta que se haya almacenado para asegurar que se ha llevado a cabo sin ningún error.
- **/rank/showRange.** Mediante este recurso el usuario puede ver el rango que tiene. Se encarga de calcular el rango del usuario mediante su puntuación. Este mecanismo se va a definir en la sección 5.6.
- **/rank/showScore.** Gracias a este recurso el usuario puede ver la puntuación que tiene en todo momento.
- **/rank/showRanking.** Este recurso va a obtener el ranking completo haciendo una consulta en la base de datos. En él se muestra el rango, el nombre y la puntuación de los jugadores registrados que tienen una puntuación establecida en el ranking.

Este proyecto se centra más sobre los recursos que se encargan de **establecer la puntuación de los usuarios y obtener el rango de estos**, ya que son los que pueden generar grandes problemas.

En la figura 5.3 se muestra un diagrama de clases que corresponde a las clases que se encargan de gestionar el ranking. La clase *Ranking* contiene funciones propias que han sido creadas para este proyecto y usa funciones que se encuentran en la clase *Ranker*, que es la que proporciona *Google*.

Establecimiento de puntuación

En los videojuegos donde hay muchos usuarios interactuando entre ellos y siguen un sistema de puntos donde algunas acciones aplicadas conllevan una actualización de la puntuación del usuario que las realiza, pueden provocar gran lentitud en el establecimiento de la puntuación al llevarse a cabo varios en el mismo instante de tiempo.

A continuación se muestra en la figura 5.4 el funcionamiento del manejador de petición que se encarga de establecer las puntuaciones de los usuarios. Como se puede comprobar en

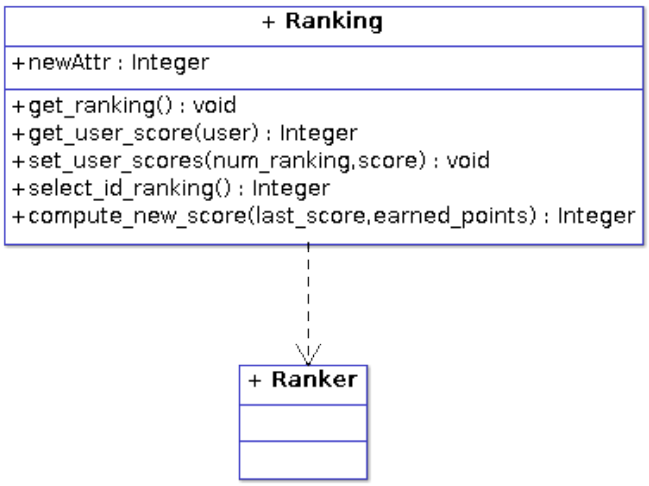

Figura 5.3: Diagrama de clases correspondiente a la gestión del ranking

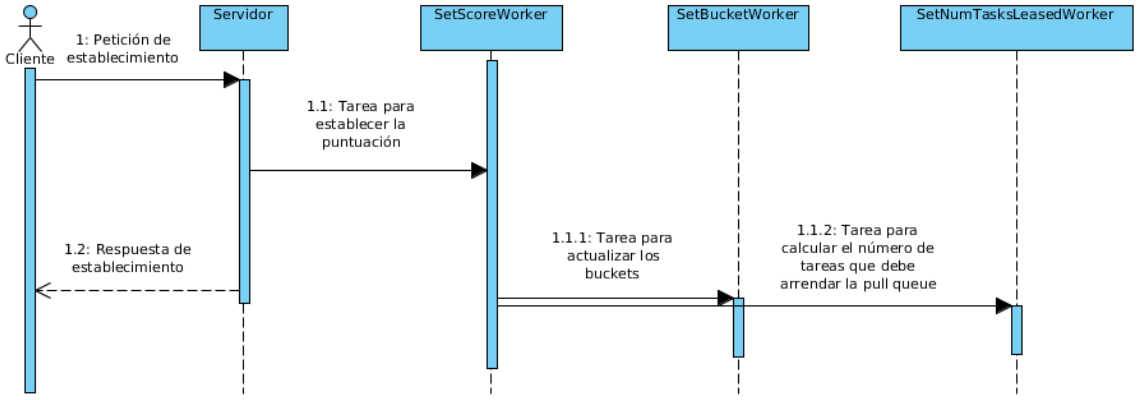

Figura 5.4: Diagrama de secuencia del manejador que se encarga de establecer las puntuaciones

la imagen, el cliente envía una petición al servidor indicando que quiere establecer una nueva puntuación. El manejador encargado de esa tarea prepara los datos necesarios para realizarla y los manda contenidos en una tarea a una *pull queue*, que se encarga de llevar a cabo el establecimiento de puntuación de varios clientes de una sola vez. Mientras, el manejador se queda esperando hasta que se haya almacenado la nueva puntuación. Cuando la *pull queue* haya llevado a cabo la tarea, envía una tarea a dos *push queues* distintas, una que se encarga de actualizar los *buckets* y otra de calcular el número máximo de tareas que debe arrendar una *pull queue* para que sea óptima, y el manejador envía la respuesta al cliente. Más adelante se hará hincapié en cada parte.

Para poder solventar este problema, en el sistema se han tenido que aplicar **una serie de mecanismos que permiten que se realice el establecimiento de la puntuación de la manera más rápida posible** en cualquier momento, sin importar el número de usuarios

que lleven a cabo la actualización en un momento específico y haciendo uso de los recursos necesarios, optimizando el uso de cada uno al máximo.

Uso de pull queues

Como se ha mencionado anteriormente, al llevar a cabo el establecimiento de puntuación por parte de muchos jugadores, se produce una gran ralentización en ese proceso. Para agilizar el proceso, se ha implementado un *worker*(explicado en la sección 3.3.1) que especifica el comportamiento de una *pull queue* que va a formar parte del sistema. Se ha decidido usar una *pull queue* para **llevar a cabo el establecimiento de puntuaciones por lotes** en vez de establecer individualmente la puntuación, ya que esto puede generar más lentitud. Pero el uso de esta *task queue*, con el paso del tiempo el rendimiento de la *task queue* fluctúa algunas veces. Por lo tanto, se van a usar diez *pull queues*, haciendo que se **reparta la carga entre todas**.

A las distintas *pull queues* van a llegar tareas que van a contener los datos necesarios para poder aplicar la nueva puntuación a los distintos usuarios. El *worker* de la *pull queue* va a recoger un número de tareas disponibles, **sin sobrepasar el límite de tareas que puede arrendar la *pull queue***, se encarga de reunir los datos de las distintas tareas y va a establecer las puntuaciones de los usuarios propietarios de la información contenida en cada tarea recibida. **El *worker* se encuentra disponible en todo momento**, permaneciendo en un bucle en el que está comprobando si hay tareas disponibles para llevarlas a cabo.

```
Mientras sea verdadero
        Obtener el numero maximo de tareas que puede recoger el worker
        Obtener tareas de un ranking especifico
        Si hay tareas
                Para cada tarea
                        Extraer informacion de la tarea
                        Almacenarla en un diccionario
                Establecer conjunto de puntuaciones
                Enviar una tarea a la queue que se encarga de calcular el numero maximo de tareas
                    que puede recoger este worker
                Enviar una tarea a la queue que se encarga de actualizar los buckets
                Eliminar tareas
```

Listado 5.1: Pseudocódigo de la pull queue que se encarga de establecer las puntuaciones

La actividad que realiza el *worker* se resume en el pseudocódigo que muestra el Listado 5.1. El funcionamiento se divide en las siguientes fases:

- Lo primero que hace es obtener el número máximo de tareas, que se ha almacenado en la configuración, que puede obtener para que el establecimiento de la puntuación sea lo más rápido posible. Este número se calcula en una de las *push queues* cuando ha llevado a cabo las tareas arrendadas y se va a explicar más a fondo en las sección 5.5.4.

Si no se ha calculado previamente, se usa un valor por defecto.

- Después obtiene las tareas que se encuentren disponibles evitando que el número de tareas obtenidas no sea mayor que el número que se ha establecido como límite y se obtienen sólo las tareas que contengan las puntuaciones pertenecientes a un *ranking*.
- Si se han obtenido tareas, se extrae la información que contiene cada tarea y se almacena junta en un diccionario, donde el índice para acceder a la información de cada usuario va a ser el nombre de este.
- Después se establecen las nuevas puntuaciones, almacenándolas en el *Datastore* y en *Memcache*, por si en algún momento cercano se necesita esa información.
- Después se manda una tarea a cada *push queue* para que realicen su función.
- Por último se eliminan las tareas que se han llevado a cabo con éxito para evitar que otra *pull queue* vuelva a realizar alguna de ellas y se vuelve a comprobar si hay tareas disponibles. Si no se ha obtenido ninguna tarea, se vuelve a comprobar si hay alguna disponible sin realizar todo el proceso que se acaba de describir.

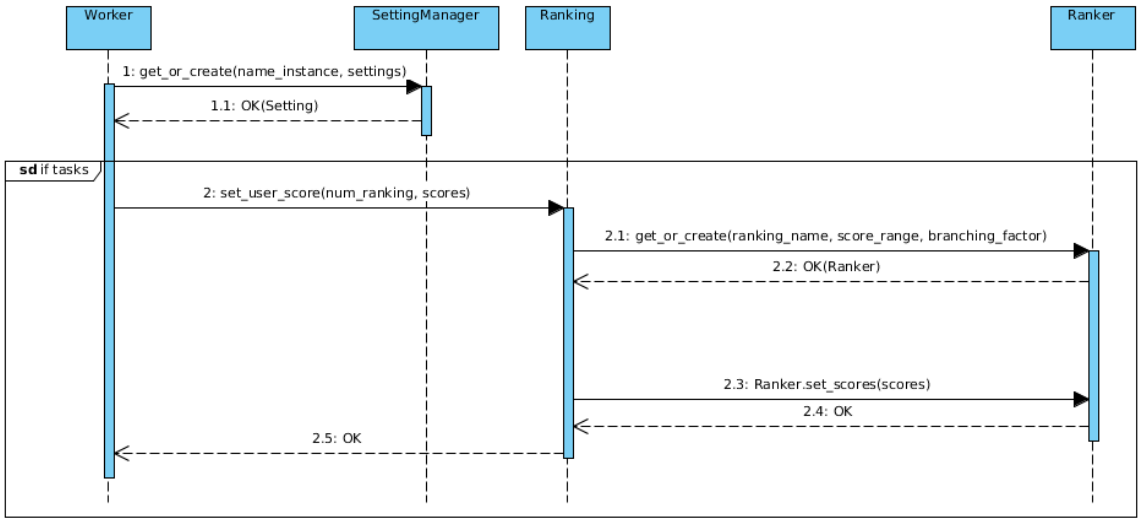

Figura 5.5: Diagrama de sequencia de la *pull queue* que se encarga de establecer la puntuación de los usuarios

En la figura 5.5 se muestra un diagrama de secuencia donde se puede observar las distintas llamadas que se realizan a las funciones de las clases correspondientes cuando una *pull queue* se encarga de establecer la puntuación de los usuarios y sus respectivos valores de retorno. Solo muestra las llamadas a métodos de otras clases en vez de a los propios para mostrar la comunicación que existe entre las clases.

Esta solución **agiliza el proceso de establecimiento de puntuación considerablemente**, manteniedose establece durante todo el tiempo.

Fragmentación del ranking

Al aplicar la anterior solución, la velocidad de establecimiento es bastante rápida, pero puede ser mejorada mediante la aplicación de otro mecanismo.

El árbol que representa el ranking se ha dividido en varias partes, más concretamente en tres, formando de esta manera tres rankings. Cada usuario tiene su puntuación almacenada en **un único árbol**. Cuando un nuevo usuario se registre, se decide y asigna el ranking donde se va a encontrar almacenada la puntuación de este, comprobando cual es el que contiene menos puntuaciones almacenadas en ese momento, aunque puede haber usuarios que tengan asignado ese ranking pero no tengan ninguna puntuación almacenada, para evitar de esta forma que uno de ellos contenga muchas puntuaciones mientras que los demás pocas. Esta información se muestra en el modelo *User*, en uno de sus atributos. Cada *pull queue* se encarga de establecer las puntuaciones de **los usuarios que pertenecen al mismo ranking**. Esto hace que los distintos rankings no tengan que coordinarse.

Con la implementación de este mecanismo, se reparte la carga, haciendo **tres veces más ágil el establecimiento de puntuación**.

Uso de buckets

La fragmentación del árbol en varios hace que **sea más lento el proceso de calcular el rango** que tiene un usuario. Esto ocurre porque para ellos se tiene que calcular el rango para cada árbol y una vez que se obtienen los distintos rangos, se suman para obtener el rango exacto. El uso de varios árboles agiliza el establecimiento de puntuaciones, pero ralentiza el cálculo de los rangos de los usuarios. Para ello se ha implementado un mecanismo que permite agilizar el proceso de la obtención de rangos. Como se ha mencionado en la sección 3.2.1, este mecanismo **no proporciona el rango con exactitud siempre**, pero va a ser cercano al valor real.

Este mecanismo consiste en el uso de ***buckets***, que van a contener la información necesaria para calcular el rango del usuario. Estos *buckets* se almacenan en el *Datastore* mediante el modelo *Bucket*. Este modelo va a estar formado por los siguientes atributos:

- **Rango de puntuaciones.** Indica el conjunto de puntuaciones que abarca el bucket.
- **Número de usuarios.** Muestra la cantidad de usuarios que tienen una puntuación que se encuentra comprendida entre el rango de puntuaciones que tiene asignado el bucket.
- **Rango mínimo.** Es el rango que tiene el usuario perteneciente al *bucket* que tiene mayor puntuación. Los usuarios pertenecientes al *bucket* no pueden tener un rango mayor que el que se indica en este atributo. Se obtiene con la suma de uno al número de jugadores que tienen una puntuación más alta que los usuarios que se encuentran en el *bucket*.

En la figura 5.6 se muestra el diagrama de clases correspondiente a las gestión de los

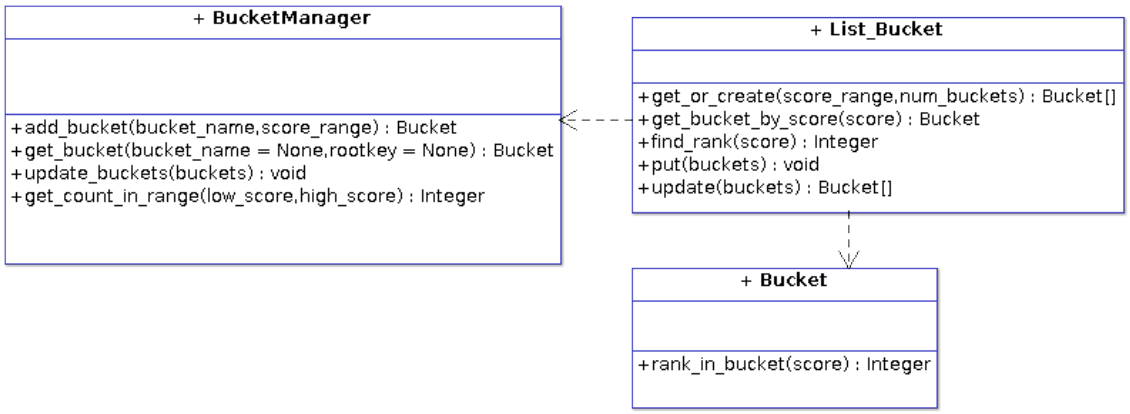

Figura 5.6: Diagrama de clases correspondiente a la gestión del bucket

buckets, donde se puede ver que la clase *List_Bucket*, que representa una lista que va a contener *buckets*, usa las funciones que se encuentran en *Bucket*, que representa un *bucket* y *BucketManager*, que representa el gestor encargado de administrar las entidades del modelo *Bucket*

La información de estos *buckets* va a ser actualizada cada vez que se establecen puntuaciones mediante el uso de una *push queue*, haciendo de este manera esta operación en segundo plano para evitar que se ralentice el establecimiento de la puntuación.

```
Obtener el conjunto de buckets
Actualizar informacion de cada bucket
Almacenar el conjunto de buckets
```

Listado 5.2: Pseudocódigo de la push queue que se encarga de actualizar los buckets

```
Establecer inicialmente el nuevo rango maximo del siguiente bucket a uno
Para cada bucket
        Obtener el nuevo numero de usuarios que tienen una puntuacion que se encuentre entre el
             rango del bucket
        Establecer el nuevo numero de usuarios que se obtiene
        Establecer el rango maximo
        Calcular el rango maximo del siguiente bucket
```

Listado 5.3: Pseudocódigo de la función que se encarga de actualizar la información de cada bucket

En el listado 5.2 se muestra el funcionamiento de la *push queue* que se encarga de actualizar los *buckets*. A continuación se explica más a fondo su funcionamiento:

- Lo primero que hace es obtener todos los *buckets*.

- Después se actualiza la información de cada uno de ellos. Para ello, por cada uno de ellos obtiene el número de usuarios que tienen una puntuación que se encuentra en el rango de puntuaciones del *bucket* en ese momento mediante una consulta en la que se obtiene el número de puntuaciones que se encuentran en el rango de puntuaciones, establece esto y el rango máximo que va a tener el usuario que tenga una puntuación que se encuentre en el *bucket*, y se calcula el rango máximo que van a tener los usuarios del siguiente *bucket* mediante la suma del valor actual del rango máximo y el número de usuarios que se encuentran en el *bucket* actual como se muestra en el listado 5.3.
- Por último, se establece el conjunto de *buckets* actualizados en el *Datastore* y en *memcache*.

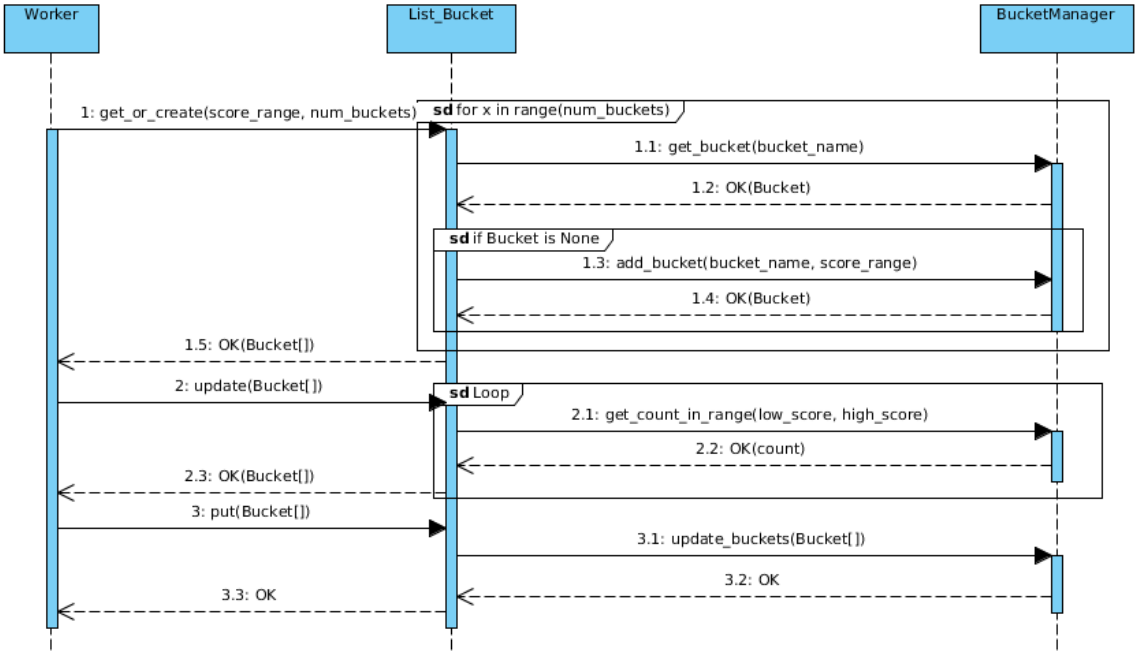

Figura 5.7: Diagrama de sequencia de la *push queue* que se encarga de actualizar la información de los *buckets*

Se puede ver en la Figura 5.7 que se muestra un diagrama de secuencia donde se puede observar las distintas llamadas que se realizan a las funciones de las clases correspondientes cuando una *push queue* se encarga de actualizar la información que contienen los *buckets* y sus respectivos valores de retorno. Solo muestra las llamadas que se realizan a métodos que corresponden a otras clases para mostrar la relación que existe entre las distintas clases.

En la sección 5.6 se expone la forma de obtener el rango mediante el uso de los *buckets*.

Mediante la implementación de este mecanismo, **se agiliza considerablemente el proceso de obtención del rango del usuario**, aunque no se obtenga todas las veces el rango exacto del usuario.

Cálculo del número máximo de tareas arrendadas

Existen momentos en los que los recursos que se usan para la realización de acciones no se aprovechan al máximo, como en este caso las diez instancias correspondientes a la *pull queue* que se están ejecutando constantemente.

Para ello, se tiene que diseñar un mecanismo que garantice **el uso adecuado de los recursos**. Debido a que GAE se encarga de forma automática sobre el escalamiento de instancias y como las *pull queues* de este sistema tienen que estar constantemente ejecutándose desde que se pone en marcha el sistema, se ha decidido modificar **el número máximo de tareas que puede llevar a cabo a la vez una *pull queue*** mediante un algoritmo.

Con este algoritmo, las cargas entre las *pull queues* se **reparten de manera adecuada para evitar ralentización**. Para el diseño de este algoritmo se ha tenido en cuenta lo que propicia esa ralentización. Esto puede ser debido a que dos condiciones: la primera, que el límite de tareas que puede obtener una *pull queue* sea un número alto, haciendo que tarde algunas tareas en aplicarse debido a que tienen que hacerse antes las que han llegado primero y puede haber muchas, y la segunda, puede ser que el número de tareas sea bajo, lo que hace que las *pull queues* disponibles se llenen enseguida y tenga que esperarse a que haya alguna disponible una vez que hayan realizado todas las tareas.

Para aplicar la solución en ambos casos, se han desarrollado dos fórmulas matemáticas para obtener la carga de tareas que deben tener las *task queues* la próxima vez según los resultados obtenidos anteriormente.

número de tareas arrendadas = número de tareas del mejor tiempo + (número de tareas del mejor tiempo - número de tareas actual) ÷ 2

(5.1)

Esta fórmula se aplica cuando el tiempo medio de todas las tareas que se han llevado a cabo con el actual número máximo de tareas que puede aceptar una *pull queue* es mayor que el del número de tareas que ha tenido el mejor tiempo.

número de tareas arrendadas = número de tareas del mejor tiempo - (número de tareas del mejor tiempo - número de tareas actual) ÷ 2

(5.2)

Esta fórmula se aplica cuando el tiempo medio de todas las tareas que se han llevado a cabo

con el actual número máximo de tareas que puede aceptar una *pull queue* es menor que el del número de tareas que ha tenido el mejor tiempo.

La idea básica de estas fórmulas consisten en aumentar o disminuir el número de tareas que puede arrendar una *pull queue* dependiendo si el tiempo mejora o empeora añadiéndole o quitándole tareas. Para que sea más entendible, se van a mostrar los cuatro casos:

- Si según va disminuyendo el número máximo de tareas arrendadas y va aumentando el tiempo, se debe de aumentar el número de tareas.
- Si según va disminuyendo el número máximo de tareas arrendadas y va disminuyendo el tiempo también, se debe de seguir disminuyendo el número de tareas.
- Si según va aumentando el número máximo de tareas arrendadas y va disminuyendo el tiempo, se debe de seguir aumentando el número de tareas.
- Si según va aumentando el número máximo de tareas arrendadas y va aumentando el tiempo también, se debe disminuir el número de tareas.

El número de tareas que se incrementa o disminuye es la mitad de la diferencia del número de tareas que ha tenido el mejor tiempo y del número de tareas actual.

El análisis y el cálculo del nuevo número máximo de tareas que puede arrendar una *pull queue* **lo realiza una *push queue* cuando se ha establecido un conjunto de puntuaciones de los usuarios**.

Se almacena en el *Datastore* el tiempo medio de duración del establecimiento de puntuación de cada usuario dependiendo del número máximo de tareas que puede realizar una *pull queue*.

Se almacena haciendo uso del modelo *Time*, que va a estar formado por los siguientes atributos:

- **Número de tareas.** Este es el número máximo de tareas que puede realizar una *pull queue* en ese momento.
- **Duracion media.** Es tiempo medio que dura el establecimiento de puntuación de cada usuario con la configuración actual, es decir, con el número máximo de tareas que se pueden arrendar. Aunque se almacene junto al número máximo de tareas, este tiempo se obtiene de la media de las tareas que se han arrendado en total, no de las que se pueden.

Hay que recalcar que el identificador del modelo *Time* es **el número máximo de tareas que arrenda la pull queue y que genera ese tiempo medio**. Esto es para evitar que se almacenen muchos tiempos distintos que tienen el mismo número de tareas y así obtener un tiempo medio más exacto con el paso del tiempo. Otros tiempos que se almacenan usan otros identificadores distintos, como es el caso del último tiempo.

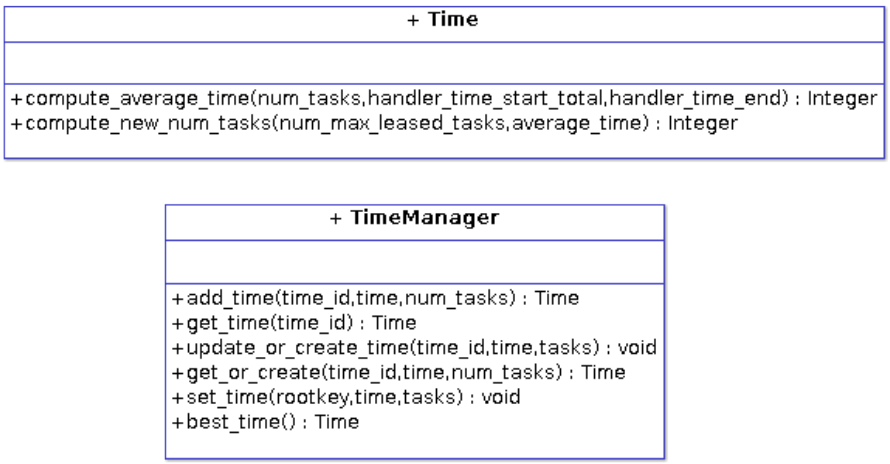

Figura 5.8: Diagrama de clases correspondiente a la gestión del tiempo

Como se puede mostrar en la figura 5.8, para la gestión del tiempo se usan dos clases, *Time*, que contiene funciones que hacen uso del tiempo obtenido en el sistema para obtener un tiempo medio o calcular el nuevo número máximo de tareas, y *TimeManager*, que se encarga de gestionar las entidades correspondientes al modelo *Time*.

También se almacena también el número máximo de tareas que puede arrendar una *pull queue* en el *Datastore* porque es el lugar idóneo para que el sistema pueda modificar este, en el modelo *Settings*. En este modelo se encuentra toda la información de configuración que va a ser modificada a lo largo de la ejecución del sistema, aunque en este caso solamente se almacena el número de tareas.

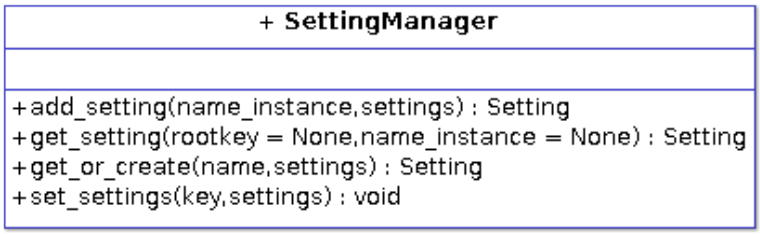

Figura 5.9: Diagrama de clases correspondiente a la gestión de la configuración del sistema

Como se puede ver en la Figura 5.9, solamente se usa una clase que se encarga de administrar las entidades del modelo *Setting*.

En el Listado 5.4 se muestra cómo funciona esta push queue, que va a ser explicado por pasos a continuación:

```
Calcular de tiempo medio de todas las puntuaciones que se han llevado a cabo en ese momento
Si el numero maximo de tareas actual arrendadas no es igual al numero maximo de tareas que tienen el
      ultimo tiempo almacenado
         Calcular el nuevo numero maximo de tareas
         Actualizar el ultimo tiempo
         Establecer el nuevo numero maximo de tareas en la configuracion
Almacenar la informacion del ultimo tiempo
```

Listado 5.4: Pseudocódigo de la *push queue* que se encarga del análisis y el cálculo del nuevo número máximo de tareas que puede arrendar una *pull queue*

```
Obtener el mejor tiempo
Si existe el mejor tiempo
        Diferencia de numero maximo de tareas arrendadas = (numero maximo de tareas que genera el
             mejor tiempo - numero maximo actual de tareas)/2
        Si el mejor tiempo es mayor al tiempo medio que genera el numero maximo actual de tareas
                Nuevo numero maximo de tareas = numero maximo de tareas que genera el mejor tiempo -
                     diferencia de numero maximo de tareas arrendadas
        Si el mejor tiempo es menor al tiempo medio que genera el numero maximo actual de tareas
                Nuevo numero maximo de tareas = numero maximo de tareas que genera el mejor tiempo +
                     diferencia de numero maximo de tareas arrendadas
Sino
        Nuevo numero maximo de tareas = numero maximo actual de tareas/2

Si nuevo numero maximo de tareas es mayor que 1000
        Nuevo numero maximo de tareas = 1000
Si nuevo numero maximo de tareas es menor que 1
        Nuevo numero maximo de tareas = 1
```

Listado 5.5: Pseudocódigo de la función que se encarga de cálcular del nuevo número máximo de tareas que puede arrendar una *pull queue*

- Primero se calcula la media de la duración de los manejadores en establecer las nuevas puntuaciones, es decir, el tiempo que tarda en llevar a cabo la acción encomendada, con el número de tareas de tareas que se han llevado a cabo, no con el número máximo de tareas que se pueden realizar. La duración que tarda en llevarse a cabo el establecimiento de puntuación se calcula obteniendo el tiempo cuando se inicia el manejador que se encarga de esta tarea y obteniendo el tiempo cuando se ha aplicado el establecimiento de la puntuación de varios clientes, y el primer tiempo se le resta al segunda para obtener la duración. Se usó primero *AppStats* para obtener la duración en la que se encuentra el manejador en ejecución, pero este no sirve cuando se inician concurrentemente varios manejadores del mismo tipo, ya que une todos los tiempos en un solo registro.
- Después se comprueba que el número máximo de tareas arrendadas actual no sea el mismo que el que se ha analizado la última vez, para evitar que se realice otra vez el mismo análisis.
- Si no es el mismo, se calcula el nuevo número máximo de tareas. Para calcularlo,

primero se obtiene el mejor tiempo. Si todavia no hay ningún tiempo almacenado, por lo que tampoco va a haber mejor tiempo, el nuevo número máximo va a ser la mitad del actual número máximo. Si hay un mejor tiempo almacenado, se calcula el número de tareas que se añade o quita al número máximo de tareas que genera el mejor tiempo dependiendo si con la configuración actual se obtiene un mejor tiempo. En este caso los que se hace es aplicar las fórmulas 5.1 y 5.2 que se han explicado anteriormente. Después se comprueba que el nuevo número máximo de tareas obtenido no sea inferior ni superior a los límites permitidos por la *pull queue*, que serían 1 y 1000. Si se sobrepasa algún límite, el valor que se establecería sería uno de los límites, dependiendo de cuál se sobrepase, como se muestra en el listado 5.5.

- Una vez que se ha obtenido el nuevo número máximo de tareas, se actualiza el último tiempo y se actualiza la configuración con el nuevo número máximo de tareas obtenido.
- Para finalizar, se almacena el tiempo medio y el número máximo de tareas del último establecimiento del conjunto de puntuaciones. Si el tiempo ya está almacenado previamente, se hace una media del tiempo almacenado y el nuevo y se actualiza el tiempo medio de este. Si no se almacena el nuevo tiempo y el número de tareas.

Como muestra la figura 5.10, se observa un diagrama de secuencia donde se puede ver las llamadas que se realizan a los métodos de las clases cuando una *push queue* se encarga de analizar el tiempo tardado en realizar el establecimiento de puntuación con cada número de tareas máximo establecido como configuración y de calcular el nuevo número de tareas.

Este algoritmo **optimiza al máximo el uso de los recursos y la velocidad de actualización**, haciendo que el sistema sea adaptativo.

Obtención del rango

Como se ha explicado en la sección 5.5.3, la fragmentación del árbol provoca una lentitud considerada a la hora de obtener el rango de un jugador. Para evitar esa lentitud, se ha tenido que hacer uso de un mecanismo que consiste en el uso de *buckets*. Cada *bucket* va a contener la información adecuada para obtener una aproximación del rango del usuario, es decir, que existen casos en los que el rango obtenido no es el real. La información de cada *bucket* se actualiza cada vez que se establecen un conjunto de puntuaciones.

```
Obtener el bucket en el que la puntuacion del usuario se encuentra dentro de su rango de
    puntuaciones
Calcular el rango del usuario dentro del bucket
Calcular el rango del usuario
```

Listado 5.6: Pseudocódigo de la función que se encarga calcular el rango de un usuario

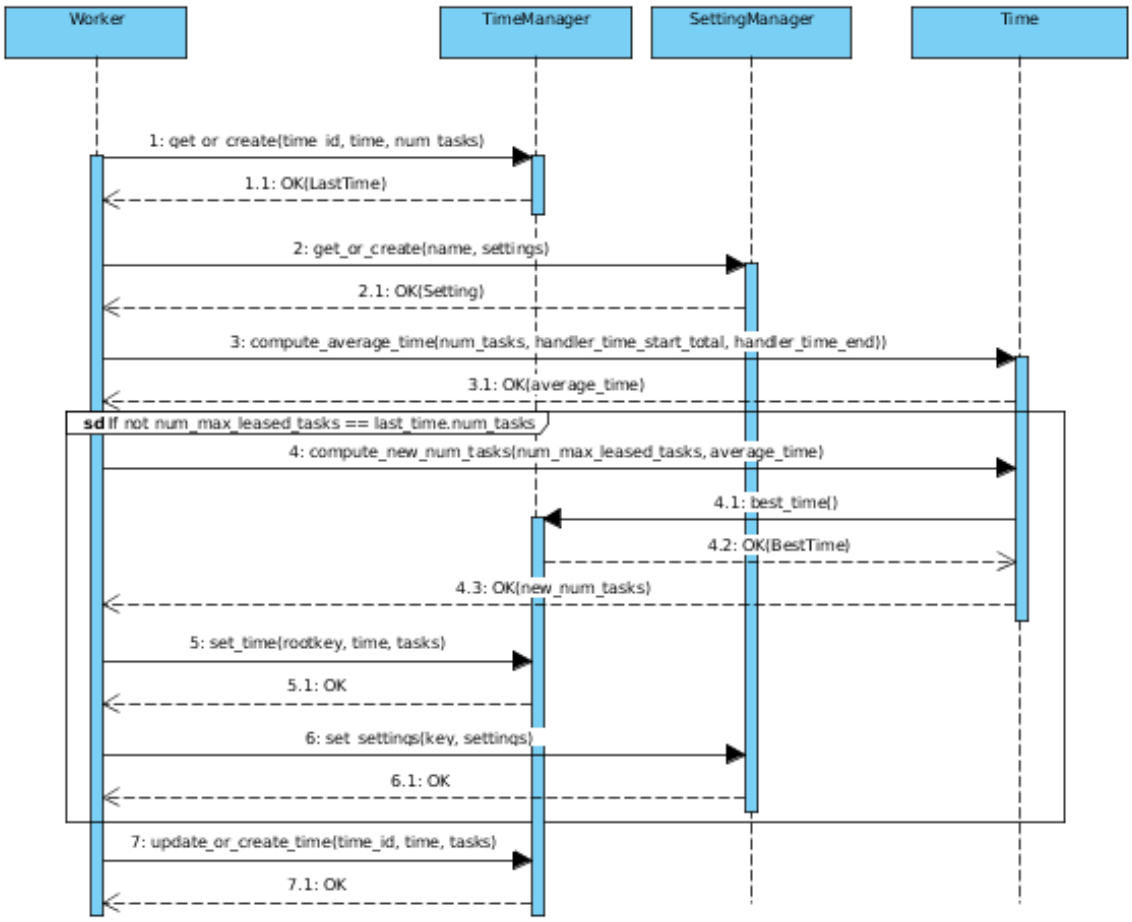

Figura 5.10: Diagrama de secuencia de la *push queue* que se encarga de calcular el nuevo número máximo de tareas arrendadas por las *pull queues*

Como se muestra en el listado 5.6, el funcionamiento de este mecanismo para obtener el rango de un usuario consiste en obtener el *bucket* en cuyo rango de puntuaciones se encuentra la puntuación del usuario que ha solicitado el rango. Una vez obtenido este *bucket*, se aplica la siguiente fórmula:

rango del usuario = rango máximo + (puntuación máxima - puntuación del usuario) * número de usuarios ÷ (puntuación máxima - puntuación mínima)

(5.3)

La fórmula 5.3 permite obtener el rango del usuario dentro del *bucket* con la información que contiene el *bucket*. El rango máximo y mínimo se refiere al límite superior e inferior del rango de puntuaciones de *bucket* y el rango máximo al rango más alto que puede tener el usuario que se encuentre en el *bucket*.

Después de obtener el correspondiente rango del usuario dentro del bucket, se aplica la

siguiente fórmula para obtener el rango total:

rango total del usuario = rango del usuario + rango máximo - 1

(5.4)

Como se puede observar, mediante este algoritmo no se tiene que ir calculando el rango buscando información en los distintos árboles. Solamente se tiene que mirar la información disponible en uno de los *buckets* y aplicarla a una fórmula para deducir un posible rango.

Puede que no sea cierto el rango que indica, pero como se ha dicho anteriormente, sería próximo al real y el algoritmo sería rápido en todo momento, sin variar su rapidez por el número de usuarios que se encuentren en el ranking.

Organización y componentes del proyecto

A continuación, se va explicar la forma en la que está estructurado el proyecto y las interacciones entre las distintas partes con la ayuda de un conjunto de diagramas y cuadros.

Estructura general del proyecto

En el cuadro 5.1 se muestra la estructura del servidor en términos de directorio, con una descripción breve de cada parte que explica la funcionalidad. Con esta estructura, se facilita el entendimiento del sistema para que sea escalable y permita en el futuro la implementación o modificación de este. Como se muestra en el cuadro, las partes lógicas del sistema se dividen en funcionalidad.

Una vez que se ha mostrado como está ordenado el sistema en directorios, se va a mostrar los componentes principales del sistema y sus conexiones mediante el diagrama de despliegue que se observa en la Figura 5.11, para comprender como se encuentra dividido el sistema en cuanto a las partes físicas necesarias.

El código que está relacionado con el *backend* se puede ejecutar en varias instancias de GAE que se encuentran alojadas en los servidores de *Google*. Los *handlers* del sistema forman una interfaz en el cual pueden exponer sus recursos y manejar las peticiones que son enviadas por el cliente llamando a la parte lógica que sea necesarias para obtener una respuesta. Para poder llevar a cabo la gestión del *Datastore* con el acceso a las entidades que se encuentra almacenadas en él, se utilizan los *managers*. Estos son clases que hacen uso del patrón ***Singleton*** y llevan a cabo la interacción con la capa de persistencia. Todas las clases correspondientes van a seguir es patrón excepto *Ranker*, ya que es una clase proporcionada por *Google*.

Estructura del proyecto			
/			
	app.yaml	Es un archivo de configuración de la aplicación. En él se suele especificar el nombre de la aplicación, versión, librerías utilizadas o el mapeo de código para el despacho de peticiones.	
	appengine_config.py	Contiene la configuración de *App Engine* para la aplicación.	
	constants.py	Está formado por valores constantes que son utilizados en la aplicación.	
	index.yaml	Es el archivo que contiene los índices formados para cada consulta que se puede realizar.	
	message_protocol.py	Conjunto etiquetas que se utilizan en los mensajes enviados entre el cliente y el servidor.	
	queue.yaml	Es un archivo de configuración de las colas utilizadas.	
	settings.py	Es un archivo que contiene un conjunto de valores de configuración utilizados dentro del código.	
	utils.py	Contiene las utilidades genéricas que son usadas por algunas partes del código.	
	backend/	Contiene todas las clases que se encargan de gestionar las entidades mediante el acceso al *Datastore* y las que contienen funciones útiles para el sistema.	
	data/	Contiene un imagen del *Datastore* que se utiliza cuando la aplicación se ejecuta de manera local.	
	handlers/	route.py	Archivo donde se índica las clases que contienen los manejadores que se encargan de llevar a cabo las peticiones recibidas.
		api/	Contiene el conjunto de manejadores que se encargar de procesar las peticiones recibidas.
		task/	Contiene el conjunto de manejadores que se encarga de llevar a cabo tareas que han sido enviadas por el servidor.
	models/	Contiene la definición para cada tipo de entidad que se encuentra almacenada en el *Datastore*.	
	tests/	Directorio que contiene un conjunto de pruebas que se realizan al sistema para comprobar su funcionamiento.	

Cuadro 5.1: Estructura del directorio de la parte del servidor

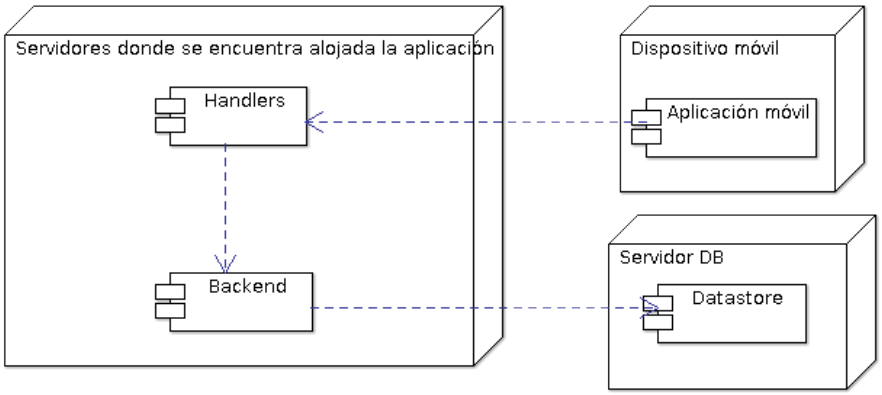

Figura 5.11: Estructura del directorio de la parte del servidor del sistema

Estructura de la base de datos

GAE proporciona diversas modalidades de almacenamiento, algunas de ellas vistas en la sección 3.3.1 donde se exponen sus servicios, como *Cloud* SQL y *Cloud Storage*, y otras como *BigQuery*, que es un *Datastore* totalmente gestionado, escalable y de bajo coste para el análisis de la empresa, y *Cloud BigTable*, que es una base de datos *NoSQL* que almacena grandes volúmenes de datos. La modalidad que se ha utilizado al ser la más apropiado para este proyecto es *Cloud Datastore*. El *Cloud Datastore* se muestra más a fondo en la sección 3.3.1. Se ha usado este por los siguientes motivos:

- **Es escalable.** Hace que el desarrollador no tenga que preocuparse por la carga que va a ser utilizada. *Cloud Datastore* escala sin que el usuario se dé cuenta y de manera automática con sus datos haciendo que las aplicaciones puedan mantener un alto rendimiento según vayan recibiendo más carga.

- **Es simple.** Se puede acceder de forma sencilla a los datos desde cualquier dispositivo.

- **El lenguaje de consulta que utiliza es sencillo.** Ofrece dos mecanismos para realizar consultas, uno que consiste en utilizar un objeto al que se le añade información mediante métodos y otro que se basa en utilizar un lenguaje similar a SQL, por lo que son fáciles de utilizar al ser familiares para el desarrollador. También proporciona un motor de búsqueda de gran alcance que permite buscar datos mediante varias propiedades y puede ordenarlos según se necesite.

Modelos definidos

Todos los tipos de entidades que son almacenados en el Datastore se definen mediante clases que heredan de la clase ***ndb.Model***. Como se puede mostrar en la figura 5.12, cada clase define un modelo o tipo de entidad con sus respectivos atributos. Las relaciones de composición y agregación en este diagrama muestran los modelos que se necesitan para calcular alguno de los atributos, sin que sea obligatoria la existencia de al menos una entidad de uno de los modelos en algunos casos. Por ejemplo, el modelo *Bucket* necesita un conjunto de entidades *RankerScore* para obtener el atributo *count*.

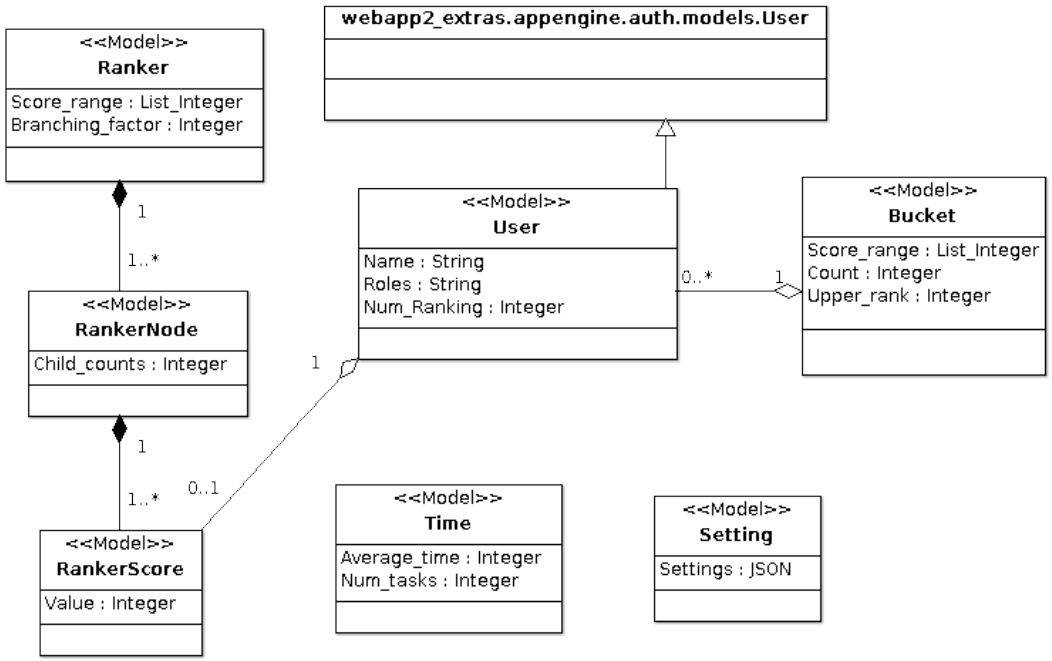

Figura 5.12: Tipos de entidades definidas en el *Datastore*

Capítulo 6

Resultados

UNa vez que se ha hablado de la problemática que se trata en este TFG, de los objetivos que se tienen que cumplir, del método de trabajo usado para el desarrollo del proyecto y las decisiones que se han tomado para solventar el problema, se va a hablar de los resultados que se han obtenido en términos de esfuerzo dedicado y del producto final al que se ha llegado. El sistema obtenido va a ser escalable, pudiendose implementar en cualquier aplicación y ser modificado con facilidad.

Aspecto y funcionalidad del sistema

Una vez que se ha desarrollado el sistema y se ha probado en local con un sólo cliente para comprobar que todas las funcionalidades funcionan de manera correcta, se ha desplegado en los servidores que proporciona *Google*. Una vez desplegado el sistema, se han realizado pruebas con una gran cantidad de clientes para comprobar si funciona correctamente. La capacidad y recursos que ofrece *Google* están limitados debido al uso de una cuenta gratuita. Se puede comprobar las limitaciones en la página de precios de *App Engine*. Una vez desplegado el sistema, el sistema se encuentra disponible a través de un dominio gratuito:

https://adaptive-management-system-pc.appspot.com/

Distribución de trabajo

A continuacion se muestra las iteracciones en las que ha sido organizado el desarrollo de este TFG. Al ser un trabajo en el que cada mecanismo añadido influye con el funcionamiento del anterior, se han solapado algunas iteracciones.

Integración de código

El desarrollo de este proyecto ha estado muy enfocado en obtener una solución que permita la gestión de rankings de forma masiva. Pero en la última fase de desarrollo se le ha dado más importancia en hacer el sistema incremental, permitiendo que se le puedan añadir nuevas funcionalidades en un futuro sin que afecte al resto del sistema, y compatible, permitiendo que se pueda implementar en otros sistemas.

1# Familiarización con las tecnologías utilizadas	
Tiempo asignado	3 semanas

Se realizó una aplicación sencilla que se encargaba de realizar las funcionalidades que contiene el sistema actual, implementando la parte del cliente y la del servidor.

El objetivo principal era tomar un primer contacto con las herramientas que se van a usar para la realización de este proyecto. En la parte del cliente se centró en el estudio del funcionamiento de la librería de Python llamada Requests, que permite la comunicación la parte del servidor, mientras que en la parte del servidor se centró en el estudio del funcionamiento de GAE, que permite el alojamiento de la aplicación en los servidores que ofrece Google, de Cloud Datastore, donde se van a almacenar los datos persistentes pertenecientes a la aplicación, y de la librería proporcionada por Google que se encarga de la gestión de los rankings. Para implementar las dos partes de la aplicación se usa el lenguaje de programación Python.

2# Implementación de varias pull queues	
Tiempo asignado	2 semanas

Se incorporó al sistema un mecanismo que consiste en hacer uso de pull queues que se van a encargar de establecer la puntuación de varios jugadores a la vez en vez de realizarlo individualmente.

Al principio, se aplicó solamente una, pero al ver que no se conseguía cumplir el objetivo por completo, se aplicaron diez. Para llevar a cabo esto se tuvo que modificar el manejador que se encarga de las peticiones encargadas del establecimiento de puntuación. Con esto se consigue agilizar el proceso de establecimiento de puntuación cuando hay varios jugadores que lo realizan.

3# Fragmentación del ranking	
Tiempo asignado	3 días

Con la iteración anterior no se conseguía la velocidad adecuada de establecimiento de puntuación con cualquier número de jugadores, aunque mejorara algo. Para lograr el objetivo se tuvo que que dividir el rango general en varias partes, o mejor dicho en varios rankings, repartiendo de esta forma los usuarios en los distintos rankings. Se tuvo que adaptar el sistema al uso de varios de rankings, modificando varios manejadores, sobre todo el que se encarga de obtener el rango del jugador, porque con varios rankings hay que calcular el número de usuarios que tienen mayor puntuación que el jugador que solicita el rango en cada ranking en vez de en uno y luego sumarlo.

4# Implementación de buckets	
Tiempo asignado	2 semanas
En esta iteración se centró en la mejora de velocidad en la obtención del rango de un jugador, debido a que lo que se realizó en la iteracción anterior provocaba una considerable lentitud. Para ello se centró en implementar un mecanismo que agiliza el cálculo del rango del usuario mediante el uso de una determinada información, aunque no es exacto el rango obtenido pero si próximo al real. Se tuvo que volver a modificar el manejador encargado de obtener el rango para que empleara este nuevo mecanismo.	

5# Implementación del algoritmo adaptativo	
Tiempo asignado	4 semanas
Una vez que se ha solucionado el problema de velocidad en cuanto al manejo de las peticiones de gran cantidad de usuarios, se ha tenido que diseñar e implementar en el sistema un algoritmo que optimice el uso de los recursos disponibles en cada momento, evitando que aumente demasiado el tiempo de respuesta de los manejadores. Se han diseñado diversos algoritmos, siendo el actual el mejor encontrado. Algunos de ellos no se han podido aplicar por limitaciones que han sido encontradas posteriormente. Para diseñar el algoritmo se han tenido que buscar las variables adecuadas que van a permitir hacer adaptativo el sistema. También, para el diseño de este, se ha tenido que tener en cuenta como gestiona el sistema GAE, lo que puede llegar a limitarlo.	

6# Integración de Cloud Endpoints en el sistema y desarrollo de aplicación Android	
Tiempo asignado	3 semanas
Una vez que se han aplicado todas las mejoras al sistema para lograr alcanzar los objetivos especificados en el proyecto, todos los manejadores han sido transformados en una API Backend para que se pueda acceder al sistema mediante dispositivos móviles o navegadores web, como en este caso, que va a ser desde un *smartphone* con SO *Andoid*. También se ha diseñado y desarrollado una aplicación *Android* que pueda acceder al sistema. También se ha añadido la autenticación mediante una cuenta de *Google*, haciendo que los usuarios puedan acceder al sistema de varias formas distintas.	

7# Despliegue del sistema	
Tiempo asignado	3 semanas
Una vez que se ha realizado todo lo anterior, se finaliza con el despliegue del sistema en los servidores de *Google*. Una vez desplegado, se realizan pruebas para comprobar su comportamiento con una gran cantidad de clientes, viendo de esta forma los resultados.	

En cuanto a la integración, se ha usado un enfoque en el que se quiere mejorar la calidad del software e incluye un conjunto de buenas prácticas que hacen que se minimice el riesgo de que el sistema tenga consecuencias que no se puedan detectar en el resto del código al realizar un cambio en una parte del código. Esto consiste en añadir cada cambio periódicamente en un entorno muy parecido al de producción, realizando una comprobación de manera automatizada de la integridad del nuevo conjunto que se ha generado mediante un *build* que se encarga de aplicar un cojunto de tests y produce los informes donde se muestran los resultados.

Capítulo 7

Conclusiones

Breve resumen de lo más destacable del trabajo con la solución propuesta. Análisis del logro del objetivo general y objetivos parciales propuestos. Concluir con posibles mejoras, ampliaciones o trabajos relacionados que quedan por hacer y que tienen interés para el tema tratado.

A modo de referencia, este capítulo tendrá una longitud aproximada de entre 2 y 10 páginas.

ANEXOS

Anexo A

Ejemplo de anexo

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Referencias

[Ama] Amazon Web Services. *Introducción a AWS*.

[App] App Engine. https://cloud.google.com/appengine/?hl=es.

[Avi11] O. Avila Mejía. Computación en la nube, 2011.

[AWS16] Visión general de los beneficios. https://aws.amazon.com/es/application-hosting/benefits/, Octubre 2016.

[BVS09] R. Buyya, C. Vecchiola, y S. T. Selvi. *Mastering Cloud Computing: Foundations and Applications Programming*. Morgan Kaufmann, 225 Wyman Street, Waltham, MA 02451, USA, 2009. ISBN 978-0-12-411454-8.

[Dat16] Google Cloud Datastore Overview. https://cloud.google.com/datastore/docs/concepts/overview, Octubre 2016.

[FK16] B. Furrow y S. Kanthak. Fast and Reliable Ranking in Datastore. https://cloud.google.com/datastore/docs/articles/fast-and-reliable-ranking-in-datastore/, Abril 2016.

[Gon] R. González Duque. *Python para todos*. Creative Commons Reconocimiento 2.5 España.

[Goo09] Google Code Jam's Ranking Library Released. http://googleappengine.blogspot.com.es/2009/01/google-code-jams-ranking-library.html, Enero 2009.

[Knu95] D. E. Knuth. *The art of computer programming*, volume 1. Addison-Wesley, Gravenstein Highway North, Sebastopol, CA, 1995. ISBN 0-201-89683-4.

[Kri10] S. Krishnan. *Programming Windows Azure: Programming the Microsoft Cloud*. O'Reilly Media, Gravenstein Highway North, Sebastopol, CA, 2010. ISBN 978-0-596-80197-7.

[LTM+11] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, y D. Leaf. NIST Cloud Computing Reference Architecture. Technical report, National Institute of Standards and Technology, 2011.

[San15] D. Sanderson. *Programming Google App Engine With Python*. O'Reilly Media, Gravenstein Highway North, Sebastopol, CA, 2015. ISBN 978-1-491-90025-3.

[Sha09] C. A. Shaffer. *A Practical Introduction to Data Structures and Algorithm Analysis*. Prentice Hall, Blacksburg, VA, 2009. ISBN 0-13-660911-2.

[Tul13] M. Tulloch. *Introducing Windows Azure: For IT Professionals*. Microsoft Press, Redmond, Washington, 2013. ISBN 978-0-7356-8288-7.

[VVE13] A. T. Velte, T. J. Velte, y R. Elsenpeter. *Cloud Computing: A Practical Approach*. McGraw-Hill, 2013. ISBN 978-0-07-162695-8.

Este documento fue editado y tipografiado con LaTeX empleando
la clase **esi-tfg** (versión 0.20170610) que se puede encontrar en:
`https://bitbucket.org/arco_group/esi-tfg`