Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Desarrollo de un Data Lake para la gestión de datos de estadísticos de la competición NBA *(National Basketball Association)*

Alumno: Héctor Sáenz Niño

Tutores: José Belarmino Pulido Junquera
Pedro César Álvarez Esteban

Agradecimientos

A mis tutores, Belarmino y Pedro César, cuya docencia logró despertar en mí un gran interés por la informática en mis inicios como estudiante. Quisiera agradecer, también, sus innumerables aportaciones y sugerencias, sin las cuales el desarrollo del presente trabajo no habría sido posible.

Resumen

El término “Big Data” se utiliza para describir los grandes volúmenes de datos que se generan actualmente. Sin embargo, la dificultad para tratar de estos conjuntos de datos mediante tecnologías tradicionales, ha fomentado el desarrollo de nuevas herramientas y arquitecturas de sistemas altamente escalables.

Este trabajo se centra en el desarrollo de un repositorio capaz de recolectar, almacenar y enriquecer datos de baloncesto. Para la recolección de los datos, se ha desarrollado un sistema capaz de obtener las estadísticas de páginas *web* de manera automática. Uno de los objetivos del trabajo era analizar la opción de usar un Data Lake para almacenar los datos, tanto adquiridos como generados. Por ese motivo los datos se transfieren a un Data Lake, donde son almacenados y transformados. Por último, se ha desarrollado una interfaz para la visualización de los datos enriquecidos.

Abstract

The term “Big Data” is used to describe the large volume of data currently generated. However, the difficulty of handling these datasets by traditional means, has motivated the develompent of new highly scalable tools and system architectures.

This project focuses on the development of a central repository capable of collecting, storing and enriching basketball related data. In order to collect the data, a system capable of automatically scrape websites has been developed. One of the objectives of this proyect was to consider the usage of a Data Lake to store the data, both scraped and generated. For this reason, the data is then delivered to a Data Lake, responsible for the storage and enrichment of the dataset. In addition, an interface to visualize the enriched data has been implemented.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

Gran parte de los avances tecnológicos de los últimos años han tenido que ver con la obtención y el tratamiento de grandes volúmenes de datos: el Big Data. Es por ello que los datos se han convertido en el nuevo petróleo, y las nuevas herramientas y metodologías, la maquinaria para sacarle partido. El análisis de estas ingentes cantidades de datos permite a las empresas reducir sus costes, mejorar el alcance de su publicidad e incluso predecir la demanda de diferentes productos [4]. Por poner un ejemplo, en el ámbito médico, posibilita el análisis de historial y la predicción de enfermedades, además de ser utilizado en la investigación [40].

La industria del deporte profesional no es una excepción. En la NBA, los ingresos anuales promedio por equipo ascienden a 267 millones de dólares [8]. En la liga estadounidense de fútbol americano, la NFL, estas cifras aumentan hasta los 452 millones anuales [33], y esta situación se repite en los diferentes deportes de masas. No es de extrañar, por tanto, que hoy día un gran número de clubes deportivos cuenten con departamentos para el análisis de datos.

Pese a la novedad del Big Data, la recolección y análisis de estadísticas para aplicaciones deportivas no es nada nuevo. En béisbol, hasta los años 80 se recolectaban medidores básicos de rendimiento, como número de *home runs*. En 1977, sin embargo, se produjo una revolución gracias a Bill James por la publicación de *Bill James Baseball Abstracts*, artículos en los cuáles se discutía la eficacia de los parámetros hasta entonces utilizados para medir la eficiencia de los jugadores, y posteriormente proponer nuevos estadísticos. En el mundo del baloncesto encontramos a Dean Oliver, quien comenzó a cuestionar el uso de las estadísticas individuales para comenzar a crear estadísticas centradas en los equipos. Tras esta primera revolución sobre los medidores de eficacia, el surgimiento del Big Data descrito en el párrafo anterior supuso un nuevo avance para la toma de decisiones deportivas.

Por otro lado, la popularización de Internet ha facilitado la distribución de este tipo de datos y métodos estadísticos. En el caso de la NBA, la propia organización comparte públicamente datos sobre los diferentes partidos, aunque se encuentran otras fuentes de estadísticas como ESPN[1] o Basketball Reference[2]. Adicionalmente, cadenas deportivas como ESPN cuen-

[1] `https://www.espn.com/nba/`
[2] `https://www.basketball-reference.com/`

tan con equipos que han desarrollado nuevas métricas, fruto de sus investigaciones [17]. De hecho, la necesidad de estos datos y herramientas para su análisis por parte de equipos y casas de apuestas, ha ocasionado el surgimiento de empresas como Stathead[3] y Sportradar[4] ofrecen soluciones de pago para el tratamiento y acceso a estos datos.

Además, el baloncesto y la NBA en concreto, resulta un escenario especialmente atractivo para el uso de técnicas de aprendizaje automático. En esta liga hay un gran número de partidos por temporada (alrededor de 1200), y el número de datos recogido por partido es alto. Además, según el autor Michael J. Mauboussing en su libro *The success equation* [27], la NBA resultó ser la competición deportiva de equipo analizada que menos dependía de la suerte. En sus estimaciones basadas en un estudio de la varianza, dedujo que el 15 % de los resultados de un equipo se deben a la suerte en la NBA, frente al 53 % de la NHL (*Natiohal Hockey League*, liga de hockey de Estados Unidos y Canadá) o el 31 % en la *Premier League* inglesa de fútbol. Estas conclusiones sugieren que el baloncesto es uno de los deportes de equipo más predecibles, y por tanto, más interesantes en un contexto de aprendizaje automático para la predicción de resultados.

1.1. Descripción

La NBA (*National Basketball Association*) es la liga de baloncesto más reconocida a nivel mundial. Su fundación se produce en agosto de 1949, tras unirse las dos ligas existentes en Estados Unidos hasta la fecha: la BAA (*Basketball Association of America*) y la NBL (*National Basketball League*).

La NBA se estrenó en la temporada 1949-50, y desde ese momento se han registrado las denominadas *boxscore*: tablas en las que se registran estadísticas individuales de un partido, como la que se puede ver en la Figura 1.1. Sin embargo, estas tablas han cambiado sustancialmente a lo largo de la historia de la competición.

En un principio se comenzó registrando los tiros libres e intentos de tiro libre, canastas y puntos totales. Posteriormente, se fueron registrando el número de faltas personales, rebotes e intentos de canasta. En la década de los 70 se produjo un importante avance en lo que respecta a los datos estadísticos: en la temporada de 1973-1974, se comienza a diferenciar rebotes ofensivos y defensivos y en la 1977-1978 se introducen los robos. El cambio más relevantes, sin embargo, se produce en la temporada 1979-80, con la introducción de la línea de triples. A partir de estas estadísticas "básicas", se fueron calculando nuevos indicadores, para evaluar, de una manera más objetiva, el rendimiento de cada jugador y de cada equipo.

El número de equipos participantes ha ido aumentando desde unos 10 en sus primeras décadas hasta 30 desde la temporada 2004-2005. La NBA consta de una temporada regular y una fase de *playoffs* con eliminatorias al mejor de 7 partidos (formato que ha ido cambiando a lo largo de la historia de la competición). La temporada regular consta de un total de 82 partidos por equipo, lo que supone un total de 1230 partidos para 30 equipos. El número de partidos de los *playoffs* depende del número de partidos, que oscila entre 28 y 49 partidos.

[3] `https://stathead.com/`
[4] `https://www.sportradar.com/`

Boston Celtics (1-1) Share & more ▾ Glossary

	Basic Box Score Stats																	
Starters	**MP**	**FG**	**FGA**	**FG%**	**3P**	**3PA**	**3P%**	**FT**	**FTA**	**FT%**	**ORB**	**DRB**	**TRB**	**AST**	**STL**	**BLK**	**TOV**	**PF**	**PTS**	**+/-**
Jaylen Brown	38:04	11	20	.550	2	7	.286	1	1	1.000	0	9	9	4	0	2	1	2	25	+2
Gordon Hayward	36:54	5	12	.417	3	4	.750	2	2	1.000	3	5	8	2	0	0	3	2	15	+9
Kemba Walker	36:13	8	22	.364	2	8	.250	4	4	1.000	3	3	6	2	2	2	2	3	22	+7
Jayson Tatum	35:10	8	22	.364	4	7	.571	5	5	1.000	3	6	9	4	3	0	0	0	25	+9
Daniel Theis	14:59	0	7	.000	0	1	.000	1	2	.500	3	3	6	1	0	2	0	3	1	+3
Reserves	**MP**	**FG**	**FGA**	**FG%**	**3P**	**3PA**	**3P%**	**FT**	**FTA**	**FT%**	**ORB**	**DRB**	**TRB**	**AST**	**STL**	**BLK**	**TOV**	**PF**	**PTS**	**+/-**
Marcus Smart	28:06	4	11	.364	2	7	.286	0	0		0	1	1	3	1	0	0	1	10	-2
Grant Williams	21:53	2	9	.222	0	3	.000	0	0		6	1	7	4	1	0	1	3	4	+5
Robert Williams	14:39	3	3	1.000	0	0		0	0		2	4	6	1	2	1	1	4	6	-3
Brad Wanamaker	9:38	1	3	.333	0	1	.000	2	2	1.000	1	2	3	0	1	1	1	2	4	0
Semi Ojeleye	4:20	0	0		0	0		0	0		0	1	1	0	0	0	0	0	0	0
Javonte Green	0:04	0	0		0	0		0	0		0	0	0	0	0	0	0	0	0	0
Vincent Poirier	Did Not Play																			
Carsen Edwards	Did Not Play																			
Team Totals	**240**	**42**	**109**	**.385**	**13**	**38**	**.342**	**15**	**16**	**.938**	**21**	**35**	**56**	**21**	**10**	**8**	**9**	**20**	**112**	

Figura 1.1: *Boxscore* del encuentro entre Toronto Raptors y Boston Celtics de octubre de 2019 en Basketball Reference

1.2. Contexto

El presente trabajo se enmarca dentro de un proyecto conjunto que consta de tres trabajos fin de grado, con el objetivo de predecir resultados deportivos de la NBA. Las diferentes partes de este proyecto serán:

- **Análisis de los índices de eficiencia**: se estudiarían los principales índices de eficiencia de las diferentes ligas y cómo se relacionan con la influencia que tienen en los resultados. Dado su alto contenido en estadística, el análisis de estos índices será realizado por el alumno de estadística Javier Martínez García en su correspondiente trabajo fin de grado.
- **Aprendizaje automático *(Machine Learning)***: se aplicarán modelos de aprendizaje automático para la predicción de resultados de encuentros. Realizado por Álvaro Berrío Galindo.
- **Creación de un espacio de almacenamiento, tratamiento y recuperación de datos que de servicio al resto de Trabajos de Fin de Grado (Data Lake)**: Sería especialmente interesante para los alumnos de informática, más aún teniendo en cuenta que la gestión de datos en entornos Big Data no es una materia de grado en los actuales estudios de Graduado en Informática.

Este trabajo se centrará en el último apartado: el diseño y desarrollo de un Data Lake, además de la extracción de los datos.

1.3. Objetivos

El principal cometido de este trabajo será el de encontrar e implementar tecnologías apropiadas para el almacenamiento y procesamiento de los datos de baloncesto, para el

futuro desarrollo de modelos predictivos y el análisis estadístico de diferentes indicadores ampliamente utilizados. Para ello, se explorarán tecnologías de Big Data y se perseguirá la planificación e implementación de un Data Lake.

La solución aportada por el sistema que se propondrá deberá ser capaz de gestionar de manera integral todo el flujo de datos, desde su extracción hasta su utilización por aplicaciones externas, además de proporcionar una solución de almacenamiento escalable que sirva para futuros trabajos y que permita almacenar datos de competiciones similares fácilmente.

Por último, se harán todos los esfuerzos posibles por hacer esta aplicación extensible a nuevos datos o nuevos orígenes de datos. De este modo, se intentarán consultar diferentes fuentes y hacer su extracción flexible para poder adaptarla a posibles cambios.

Capítulo 2

Planificación

En este capítulo se describe la planificación de este proyecto. Las tareas para el desarrollo de este proyecto son numerosas, debido a los diferentes campos que abarca su ejecución completa. En primer lugar, se estudiarán diferentes fuentes de datos para encontrar la más adecuada. Posteriormente se realizará un sistema capaz de extraer esta información, probablemente mediante *web scraping*. Por último se deberá estudiar las diferentes herramientas que son necesarias para la creación de un Data Lake. Dado que no se ha estudiado ninguna de estas herramientas de Big Data durante el grado universitario, la mayor parte de las tareas recaerán en el estudio y aprendizaje de las mismas.

2.1. Identificación de tareas

1. **Exploración de los datos y sus fuentes**. Este apartado será prioritario, ya que los datos extraídos serán necesarios para el análisis en los trabajos pertenecientes al proyecto común explicado en el Apartado 1.2.

   a) Búsqueda de fuentes de datos. Consulta de las diferentes páginas que ofrecen servicios y la viabilidad de extracción.

   b) Modelización de los datos disponibles.

   c) Determinación de los cálculos y procesos de transformación que se desean realizar.

2. **Estudio de las tecnologías y arquitecturas de *Big Data*** para la construcción de un sistema capaz de manejar los datos de la NBA.

   a) Estudio de las arquitecturas y prácticas habituales.

   b) Estudio de las tecnologías más apropiadas para implementación de la arquitectura elegida.

3. **Desarrollo del sistema de captura y obtención de datos**. Lo que requerirá:

   a) Codificación del sistema de captura de datos.

   b) Despliegue de la aplicación.

4. **Implementación de las tecnologías de *Big Data* para la constitución del *Data Lake***. Se desarrollará el sistema descrito en el capítulo primero.
   a) Tras haber estudiado las diferentes opciones y la naturaleza de los datos específicos para este proyecto, se elegirá la arquitectura más apropiada.
   b) Configuración del entorno.
   c) Desarrollo del sistema de almacenamiento: ingesta y transformación de los datos.
5. **Diseño y desarrollo de una interfaz** para la visualización de los datos procesados. Se estudiará el uso de herramientas de visualización de datos disponibles y de inteligencia de negocio *(business intelligence, BI)*.
   a) Estudio sobre aplicaciones de BI frente a la implementación de una interfaz propia.
   b) Codificación de la interfaz.
6. **Documentación**

2.1.1. Estimación horaria

Para la realización de este proyecto se supone una carga de trabajo aproximada de 300 horas, que deberán ser planificadas para su correcto aprovechamiento. En la Tabla 2.1 se recogen las estimaciones en horas de trabajo para cada actividad. MC significará "Mejor Caso", CP, "Caso Promedio" y PC, "Pero Caso". Por otro lado, "Pred." recoge las actividades predecesoras (esto es, que se deben terminar para continuar con la especificada).

Hito	Actividad	MC	CP	PC	Pred.
1	Exploración datos				
1.a	Búsqueda de fuentes de datos.	6	10	14	
1.b	Modelado de los datos.	16	24	32	1.a
1.c	Definición de las transformaciones.	4	8	16	
2	Estudio sobre Big Data				
2.a	Arquitecturas y prácticas.	50	80	120	
2.b	Tecnologías apropiadas.	35	60	80	
3	Sistema de captura				
3.a	Codificación del sistema.	20	40	70	1.b
3.b	Despliegue del mismo.	5	8	16	3.a
4	Implementación Data Lake				
4.a	Configuración del entorno.	30	50	80	2.b
4.b	Desarrollo de los procesos.	10	16	24	3.a
5	Desarrollo interfaz				
5.a	Estudio herramientas BI.	2	4	6	1.b
5.b	Codificación.	20	35	50	3.a
6	Documentación				
6.a	Memoria.	24	60	71	1,2,3,4,5
	TOTAL ESTIMADO	224	395	579	

Tabla 2.1: Tareas y estimación horaria inicial.

Capítulo 3

Arquitecturas de Big Data

En los últimos años de progreso tecnológico se ha visto un aumento sin precedentes de la cantidad de datos generados por el ser humano y sus dispositivos. Este aumento ha sido causado por el auge de Internet, las redes sociales y la creciente adaptación de dispositivos electrónicos que constantemente generan datos biométricos, de localización y acerca del uso de los dispositivos. De esta manera, el aumento de usuarios en Internet pasó de 413 millones en 2000 a cerca 3.400 en 2016 [28]. Esta gran cantidad de usuarios y sensores puede llegar a generar cantidades de información que requieren un enfoque distinto a las aplicaciones tradicionales. Facebook, la conocida red social que cuenta con más de 2.300 millones de usuarios, genera 4 petabytes de datos nuevos cada día [20].

Esta nueva época centrada en la recolección y explotación de estas cantidades de datos presenta muchas oportunidades, pero también nuevos retos.

En este capítulo se discutirán diferentes arquitecturas de Big Data, para poder así elegir la más apropiada para este proyecto. Estas arquitecturas propondrán soluciones para la ingesta, procesamiento, visualización y el posterior análisis de los datos.

3.1. Introducción

Cabe plantearse qué es el Big Data, y cuáles son sus características. Si nos referimos al nombre, queda claro que se puede considerar Big Data sólo por tratar con terabytes de datos. Sin embargo, los problemas actuales tienen más dimensiones, que aumentan su complejidad de manera considerable. Estas diferentes características del Big Data son denominadas por algunos autores como las "uves del Big Data" [21, 22, 39]. Cabe destacar que hay autores que hablan de 3 "uves" y otros de hasta 7, pero las siguientes 4 se pueden considerar las más relevantes:

- Volumen: esta será la principal definición de los "grandes datos". Volúmenes cada vez más grandes, del orden de los terabytes en adelante.
- Velocidad: la velocidad de procesamiento y extracción de la información puede ser crítica para aportar valor a nuestras decisiones. En ámbitos como las apuestas, la banca o el mercado bursátil, contar con la información más reciente puede significar una

ventaja crucial, lo cual puede ser difícil al contar con tantos datos. Según la velocidad o la inmediatez con la que se acceda o procesen los datos los sistemas Big Data se clasifican en:

  - Batch (en lotes): sistemas que no necesitan ser procesados de manera rápida y pueden esperar incluso días, como por ejemplo, un sistema de nóminas. Al tener una latencia más grande, son más fáciles de diseñar.
  - Near real-time: en el caso del "casi en tiempo real", no es crítico y el usuario puede ver información con una latencia de unos segundos. Un caso de "near real-time" serían los sistemas de *logs* de servidores.
  - Real-time: se caracteriza por tener un flujo de datos continuo, y que necesitan ser procesados con unas restricciones de tiempo muy pequeñas. Las transacciones con un cajero automático son un buen ejemplo de este escenario.

- Variedad: no siempre se puede disponer de datos perfectamente estructurados y preparados para su explotación. Se estima que el 80 % de los datos son no estructurados [41]. Se puede diferenciar entre [21]:

  - Estructurado: datos definidos por un modelo de datos, como los presentes en las bases de datos relacionales. Dentro de estos se encuentran:
    - Datos en crudo: sin tratamiento ni comprobaciones.
    - Datos limpios: se han desechado o restaurado los datos incorrectos.
  - Semi-estructurado: pese a carecer de la estructura formal de los datos estructurados, presenta una estructura aparente y cuenta con elementos que permiten diferenciar elementos semánticos y definir jerarquías. Se puede considerar este tipo de datos como auto-descriptivos. Formatos como XML o JSON.
  - No estructurado: el resto de datos, como vídeos, imágenes, etc.

- Veracidad: hay que contar con encontrarnos datos que no son fiables, o son incompletos. Esto hace mucho más complicado el tratamiento y el almacenamiento de los mismos, y requerirá de una atención especial.

3.1.1. Retos

La gestión Big Data en casos reales se centra en grandes corporaciones, que son los principales organismos que necesitan lidiar con datos tan diversos y abundantes. El objetivo final es el de conseguir una vista única; un portal que integre todos los datos, denominado *Single Customer View* (SCV).

La necesidad de un sistema de integración de la información en las corporaciones surge de la evolución natural de las mismas, que en su crecimiento, desarrollan numerosos sistemas aislados. Posteriormente, se buscará integrar estos sistemas mediante un repositorio central (ver Figura 3.1), que ofrezca una vista única de los datos del sistema (SCV). Este repositorio central necesitará hacer frente a los siguientes retos:

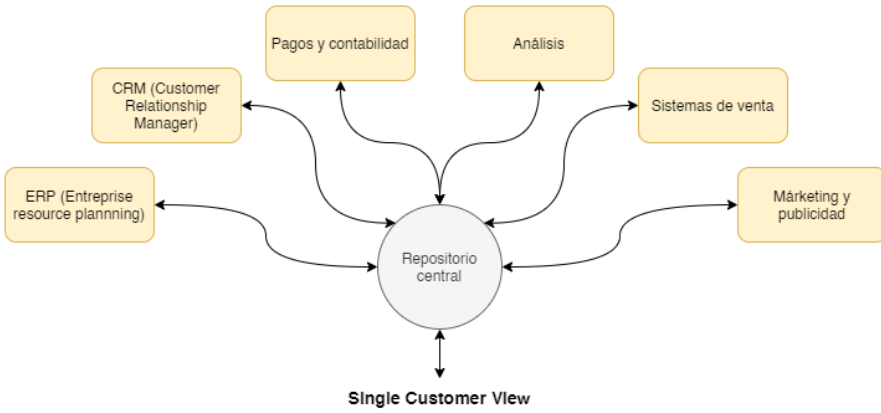


Figura 3.1: Repositorio central integrando los datos de sistemas habituales de una empresa. *(Fuentes: elaboración propia a partir de [21],* `https://www.ricston.com/single-customer-view/`*).*

Procesamiento de los datos El fin último para la recolección de estos datos es el de su procesamiento para la generación de información relevante en el negocio. La capacidad de procesar la totalidad de los datos corporativos permite obtener una visión de conjunto de la corporación y una mejor toma de decisiones.

Crecimiento de los datos El mero funcionamiento de los sistemas anteriores y su progresivo desarrollo y extensión generan grandes cantidades de datos. A esto, hay que añadir la incorporación de nuevos sistemas a la organización debido a su crecimiento y mejoras tecnológicas. Como ejemplo para la introducción de nuevos sistemas se puede pensar en la introducción de un nuevo sistema de gestión de personal. Por otro lado, como ejemplo de crecimiento de datos por mejora tecnológica, encontramos el registro de nuevas interacciones de usuario con la página web de una tienda *online*.

Flexibilidad ante el cambio Este sistema central deberá ser capaz de incorporar los cambios que se produzcan debido a la modificación de los requisitos de negocio en cualquiera de los sistemas que se integran.

Accesibilidad y catalogación El repositorio central busca que los datos de una organización sean fácilmente accesibles. Es decir, que los datos generados por un departamento o rama de la organización puedan ser consultados por otros directivos o departamentos. Sin embargo, los datos de una corporación son tan complejos que resulta imposible recordarlos, y por ello será necesaria la catalogación de los mismos (generación de meta-datos y documentación); los datos no sirven de nada si los miembros de la organización no pueden acceder a ellos.

Seguridad de los datos La información corporativa contiene datos sensibles cuyo acceso debe ser restringido. Es más, la gestión de estos datos se ve afectada por leyes y normativas

de protección de datos. No obstante, se deberá garantizar la seguridad comprometiendo lo menos posible la accesibilidad.

Integridad de los datos Los datos de una corporación son usados para el análisis de los mismos. Podemos diferenciar dos tipos de análisis: formal e informal [19]. El primero supone un análisis con consecuencias legales o muy graves, como una auditoría. Los informales no tienen consecuencias tan graves, y por tanto, no se requiere que los datos sean altamente fiables. Esto ocasionará que haya datos sobre los que se deberán aplicar unas altas exigencias de calidad e integridad.

Para la implementación de este repositorio central se ha utilizado, históricamente, el Data Warehouse con procedimientos ETL. De manera más reciente se ha propuesto el uso de los denominados Data Lakes, que proponen un cambio de paradigma.

3.2. Data Warehouse

Los Data Warehouses son sistemas de datos relacionales que almacenan datos a nivel corporativo y están diseñados para la consulta y análisis de datos. Se diferencian de las bases de datos comunes (OLTP, *Online Transaction Processing*, es decir, para Procesamiento de Transacciones en Línea) en dos puntos clave. En primer lugar, los Data Warehouses trabajan con volúmenes de datos mucho mayores, ya que tratan todos los datos corporativos. En segundo lugar, los datos de un Data Warehouse contienen, principalmente, datos históricos, frente a las OLTP. Dada la complejidad que supone desarrollar e implementar un Data Warehouse a nivel corporativo, existe el concepto de Data Mart, que consiste en un sistema de almacenamiento relacional, pero a un nivel más pequeño.

3.3. Data Lake

Un Data Lake es un sistema para el almacenamiento de grandes cantidades de datos en su formato original, sin aplicar agregaciones ni transformaciones. Adicionalmente, aporta herramientas para su posterior procesamiento y administración. Esto se debe al auge de tecnologías de Big Data, que permiten analizar datos que antes eran imposibles de relacionar. Por esta razón, en un Data Lake se pretende almacenar todos los datos de una organización; tengan o no un uso conocido. Guardando todos los datos posibles abrimos la posibilidad al análisis futuro de los mismos, pese a que en el momento no se disponga de la tecnología apropiada.

Los datos en las corporaciones se encuentran de manera dispersa en numerosas aplicaciones y servicios, a menudo aislados. Se pueden resumir, según [21], en:

- **Data Warehouses (DW), Data Marts y *Business Intelligence* (BI) convencionales**: se trata de datos limpios extraídos mediante ETL *(Extract Load Transform)*. Sus mayores problemas son:
  - Cambiar el modelo para añadir un campo o una relación requiere a menudo un esfuerzo importante.
  - Los datos más antiguos se envían a menudo a almacenamiento permanente, que dificulta notablemente su acceso.

- **Big Data *puddles* (charcos)**: se trata de sistemas de *Big Data*, pero que comúnmente se han creado en un departamento en concreto con unas aplicaciones específicas. Al no haber una cooperación interdepartamental, se limita mucho el valor que se puede obtener. En [13], se denomina a estos sistemas "charcos de datos" (*Big Data puddles*). Supone más costes y menos beneficios que si estuviera unificado.

- **Sistemas aislados**: aplicaciones de diferente naturaleza que no comparten información. Por ejemplo, los sistemas de nóminas y los de ventas estarán aislados. En ciertos casos, es posible que las diferentes divisiones tengan sus sistemas de venta aislado (por ejemplo, entre franquicias de diferentes países).

La problemática, sin embargo, no acaba cuando se consigue integrar todos los datos en una misma aplicación. Tratar tal cantidad de datos es extremadamente difícil por su volumen. Es por ello que la implementación de un Data Lake con las tecnologías adecuadas aportará numerosos beneficios, como:

- Realizar análisis usando la totalidad de los datos de la empresa, siendo estos datos de clientes, procesos o empleados. Utilizar todos los factores posibles aporta una mayor visión sobre la corporación.

- Completar los análisis en unos tiempos mucho más reducidos, tanto por eficiencia computacional como por facilidad para recoger y obtener los datos.

- Mayor rendimiento para la optimización de procesos, al tener información más reciente y precisa.

El concepto de Data Lake hace referencia a las capacidades y características anteriormente mencionadas, sobre la que destaca la ingesta de datos en crudo. Sin embargo, no tiene que ver con la arquitectura elegida para su implementación ni con las tecnologías utilizadas en la misma. No obstante, el desarrollo de un sistema de estas características resulta complicado si no se aplica un conjunto de buenas prácticas. A continuación se presentarán las dos arquitecturas de Big Data más extendidas.

3.3.1. Arquitectura Lambda

La arquitectura Lambda es un patrón para el diseño de sistemas de procesamiento de datos. Por ello, no depende de tecnologías concretas, sino que se basa en ciertos principios que harán posible cumplir los requerimientos y evitar los errores más comunes [21]. El término Lambda fue acuñado por Nathan Matz en su libro *Big Data: Principles and Best Practices of Scalable Realtime Data Systems* [26].

3.3.1.1. Principios

Capacidad de recuperación ante fallos Al tratar con tantos datos es cierto que se van a producir fallos, ya sean humanos, de *hardware* o de *software*. Como se verá más adelante, se requerirán muchas máquinas, por lo que serán comunes los errores de disco y red, y además, se deberá contar con posibles *bugs* en el código.

Inmutabilidad de los datos Esto permite la re-computación, y hace viable recuperarse de los errores descritos anteriormente. Los datos deberán guardarse en el formato más "crudo" posible, de manera que sea más robusta frente a la corrupción de datos y la pérdida de datos. Se permitirá insertar datos, pero no actualizarlos o borrarlos, como norma general. Esto refuerza el principio de la tolerancia de fallos. Asimismo, se posibilitará la inferencia de nuevas métricas en el futuro.

Re-computación La re-computación significa tener la capacidad de volver a calcular todas las métricas de nuevo. Esto es posible gracias al principio anterior, que al mantener los datos originales se puede calcular nuevas métricas y vistas.

3.3.1.2. Componentes

La arquitectura Lambda cuenta con tres capas: *batch layer* (o capa por lotes), *speed layer* (o capa de velocidad) y *serving layer* (o capa de servicio) [21]. Una visualización esquemática de esta arquitectura se puede observar en la Figura 3.2.

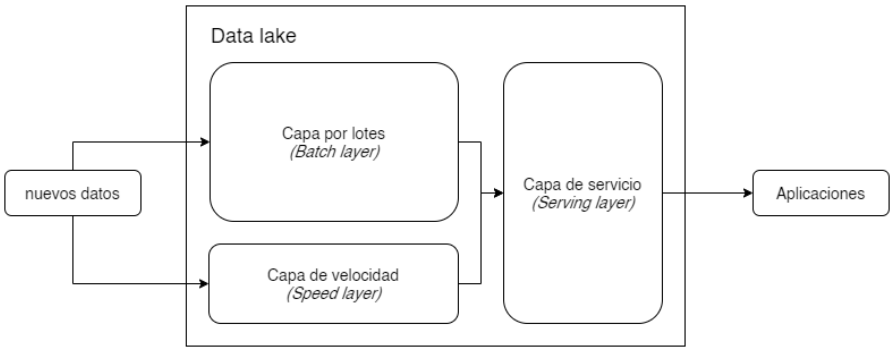


Figura 3.2: Arquitectura Lambda *(Fuentes: elaboración propia a partir de [26]).*

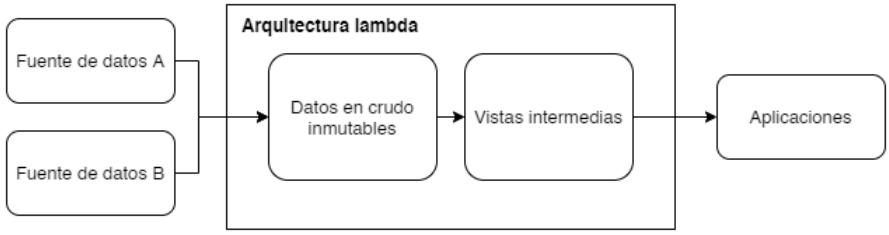


Figura 3.3: Flujo de datos en una arquitectura Lambda *(Fuentes: elaboración propia a partir de [26]).*

Batch layer En esta capa la información se almacena de la manera más pura y con el menor tratamiento posible. Al no existir esta transformación durante el almacenamiento, es

posible usar estos datos para casos de uso con perspectivas muy diferentes. En este punto se encuentran los datos maestros, que serán usados como referencia y ni se modificarán, ni borrarán, ni actualizarán. La actualización de los datos se implementará mediante la adición de *timestamps* o huellas temporales a los registros, de manera que cuando se pida la información más reciente se obtenga el que tenga *timestamps* más recientes.

Sin embargo, las consultas al archivo de datos en crudo pueden resultar muy lentas. Esta lentitud puede estar causada por el formato de los datos en crudo o por la necesidad de consultar más datos. Para agilizar las consultas se introducen vistas. Estas vistas se computan y guardan de manera periódica en un formato más apropiado para el caso de uso concreto. Cuando una vista se vuelve a recomputar, la anterior se descarta, aunque sin descartar los datos en crudo, naturalmente. Esto aporta una gran tolerancia a los fallos, ya que por ejemplo, un error humano al calcular una vista se soluciona volviéndola a calcular, ya que ningún dato se ha descartado (siempre se dispone del "archivo" de datos en crudo). Dependiendo del caso de uso, podrán ser necesarias numerosas vistas intermedias, ya que cuando el volumen de datos es muy grande, una aproximación ingenua al problema puede causar que estos cálculos duren muchas horas o incluso días.

Las vistas intermedias se pueden diferenciar entre ellas, como se hace en [16]. Según la información y lo refinados que se encuentren los datos en cada vista, se podrá diferenciar entre diferentes tipos (ver Figura 3.4):

- **Bronce *(bronze)***: tablas de ingestión en crudo. Estas tablas no han sufrido transformaciones: es el archivo de datos en crudo.
- **Plata *(silver)***: tablas intermedias que han sido filtradas y/o se han aumentado, añadiendo campos o combinándolas con otras tablas. Vistas necesarias para la generación de vistas *gold*, ya sea por rendimiento o por simplicidad de cálculos.
- **Oro *(gold)***: datos aptos para ser usados en las aplicaciones de negocio. Datos finales.

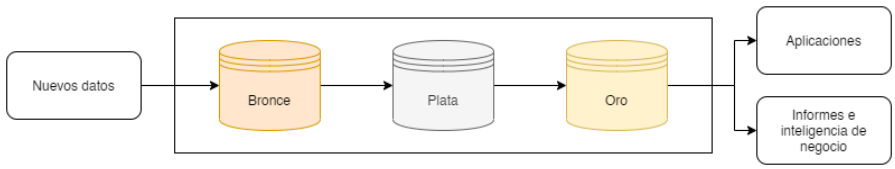


Figura 3.4: Vistas intermedias de un Data Lake en la arquitectura Lambda *(Fuentes: elaboración propia a partir de [21]).*

Esta capa de persistencia deberá ser capaz de procesar un gran número de lecturas aleatorias, aunque no será necesario que soporte escrituras aleatorias, ya que los datos se cargan en lotes grandes.

Capa de velocidad *(speed layer)* La capa por lotes *(batch layer)* maneja los datos históricos, y realiza cálculos de manera periódica. Sin embargo, estos intervalos suelen ser

grandes y no permiten tratar con datos generados de manera constante, que se pueden necesitar en las aplicaciones de negocio o análisis. Por tanto, la responsabilidad de la capa de velocidad será la de combinar los datos de las vistas de la capa por lotes con los datos en tiempo real que todavía no han sido procesados por la primera. Aunque se aumente la frecuencia de recomputación de las vistas de la capa por lotes, estos cálculos siguen siendo relativamente lentos al tratar con muchos datos, lo que hace que los procesos de negocio no puedan esperar tanto tiempo. Por ello es necesaria esta capa de velocidad. Una vez que las vistas de la *batch layer* se han puesto al día, los datos ya recomputados presentes en la *speed layer*, son descartados.

Para conseguir estas características, esta capa se adhiere a dos principios:

- **Computación incremental**: la vista en tiempo real se calcula usando una vista en tiempo real ya existente y combinándola con los nuevos datos.
- **Consistencia eventual**: el cálculo de ciertas vistas en tiempo real es complejo y lleva tiempo. Por ello, el principio de consistencia eventual propondrá obtener estos cálculos de manera aproximada en un principio, y que con el tiempo y la finalización de los cálculos esta información se corrija. En caso de que los datos de la capa de velocidad se vean corrompidos o dañados, se recomputarán gracias a la *batch layer*. Esto es importante porque se espera que sean sistemas que garanticen la disponibilidad por encima de la consistencia.

A diferencia de la capa por lotes, en esta capa sí que será necesario disponer de lecturas y escrituras aleatorias rápidas, pero el volumen de datos será mucho menor, ya que se trabajará con periodos de horas o días.

Capa de servicio *(serving layer)* El cometido de esta capa es el hacer disponibles y combinar las vistas de la capa por lotes con la capa de velocidad, de manera que las aplicaciones cliente no tengan que preocuparse de trabajar con dos fuentes.

Esta capa, tiene una importante responsabilidad orquestando las otras dos, ya que es necesario comprobar el estado de la capa por lotes para poder descartar una parte de la capa de velocidad al haber terminado un proceso *batch* (ver esquema de la Figura 3.5).

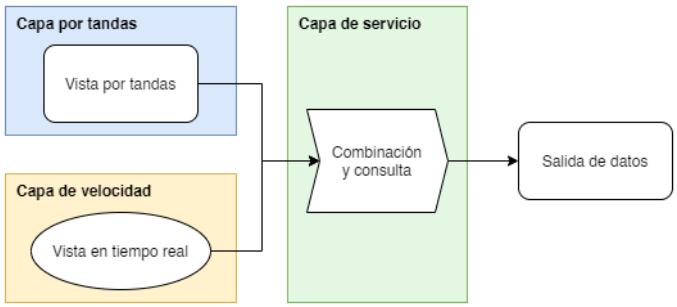


Figura 3.5: Capa de servicio junto al resto de capas *(Fuentes: elaboración propia a partir de [13, 26])*.

3.3.2. Arquitectura Kappa

La arquitectura Kappa se trata de una simplificación de la arquitectura Lambda. Esta arquitectura elimina la *batch layer* o capa por lotes, y sólo se mantiene la capa de velocidad, que se denomina capa a tiempo real (ver Figura 3.6). De esta manera, no se necesita programar los procesos para la capa por lotes y la capa de velocidad dos veces, si no que sólo es necesaria la segunda, y los costes de desarrollo son menores. Permite recomputaciones volviendo a ejecutar las tareas sobre los *streams* de entrada, aunque no es posible almacenar de manera histórica todos estos datos, que acaban siendo descartados. De esta manera, este tipo de arquitectura es más simple, pero es menos versátil que la arquitectura Lambda.

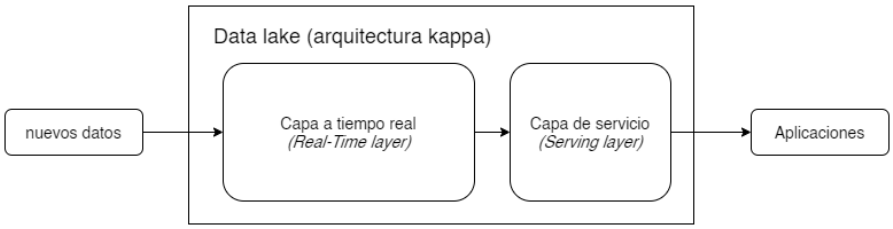


Figura 3.6: Flujo de trabajo en la arquitectura Kappa *(Fuentes: elaboración propia a partir de [13, 26])*.

3.4. Diferencias entre Data Lakes y Data Warehouses

Los Data Lakes y los Data Warehouses proponen planteamientos diferentes para el problema de unificación de datos y procesamiento a nivel corporativo. En la Tabla 3.1 se resumen las diferencias más importantes entre los dos sistemas.

	Data Lake	**Data Warehouse**
Tipo de dato	Todo tipo de datos y formatos.	Datos históricos que responden a un esquema relacional.
Propósito	Almacenamiento barato y eficaz para *Big Data*.	Generación de reportes para tomar decisiones de negocio.
Usuarios	Ingenieros de datos	Analistas de datos y ejecutivos.
Tareas	Almacenamiento y procesamiento de *Big Data*.	Almacenamiento de datos y realización de consultas de manera eficiente.
Volumen	Todos los datos posibles, tengan o no relevancia a día de hoy.	Sólo los datos relevantes para el análisis.
Esquema	Estructura en lectura *schema-on-read*.	Estructura en escritura *schema-on-write*.

Tabla 3.1: Diferencias entre Data Warehouse y Data Lake *(Fuentes: [21, 47])*.

3.4.1. Almacenamiento

Los Data Warehouses guardan la información necesaria para los procesos de negocio y para la obtención de métricas e indicadores clave de rendimiento (KPIs o *Key Performance Indicators*) necesarios. Estos datos tienen una fuerte estructura y tienen un propósito. Por otro lado, en un Data Lake, se guarda información de todo tipo y sin un propósito claro.

Al guardar información de diferente naturaleza y volumen, encontraremos diferencias entre las tecnologías usadas. De esta manera, los *data warehouse* utilizan bases de datos relacionales para almacenar la información. Por otro lado, en los Data Lake se opta por sistemas de ficheros distribuidos, que permiten una mayor escalabilidad y un coste de operación menor.

3.4.2. Seguridad

La definición del esquema completo de los Data Warehouses hace posible la implementación de permisos de seguridad en datos sensibles. Además, estos sistemas cuentan con una madurez de varias décadas. En cambio, la naturaleza abierta de los datos en un Data Lake dificulta la seguridad de los datos sensibles.

3.4.3. *Schema-on-read* vs. *schema-on-write*

En los DW se tiene un esquema o estructura en escritura *(schema-on-write)*. Esto quiere decir, que los datos cargados en el DW deben cumplir una estructura concreta. Así, se garantiza la estructura de todos los datos y sólo encontramos datos estructurados en su interior. En un Data Lake se aplica la estructura en lectura *(schema-on-read)*. Esto es, no se pondrán restricciones ni se aplicarán transformaciones a los datos al escribirlos, si no que se leerán en función de la aplicación a la que vayan destinados. Las principales desventajas del *(schema-on-write)* comparado con el *(schema-on-read)* son [34]:

- **Esquema único.**

  En *schema-on-write*, se requiere un esquema único para todas las aplicaciones, por la naturaleza de los esquemas relacionales. De esta manera, se sacrifica un rendimiento óptimo en cada aplicación, lo que dificulta su uso con volúmenes de información muy grandes. Con un *schema-on-read*, se creará un esquema óptimo para cada uso, sin afectar a los demás.

- **Dificultad de modificación del esquema.**

  La complejidad del esquema descrita en el punto anterior complica enormemente la modificación del mismo. Incluso el cambio de una sola columna puede tener implicaciones importantes.

- **Aumento del tiempo de desarrollo.**

  La necesidad de desarrollar el esquema de manera completa alarga el tiempo de desarrollo y no permite avanzar una tarea de análisis. Con un *schema-on-read*, se puede avanzar con tareas que requieran modelos sencillos de manera casi inmediata, que puede suponer una gran ventaja.

- **Necesidad de transformación de los datos.**

  Un *schema-on-write* requerirá preparar los datos para adaptarlos a su esquema. Pese a que esto garantice la coherencia y validez de los datos, también implica un coste de desarrollo mayor y habitualmente requerirá quedarnos con una parte de la información, en vez de con toda. Por otro lado, en un *schema-on-read*, se guarda el dato en su totalidad, permitiendo un uso futuro que todavía no se puede prever. Y más importante: sin ningún esfuerzo adicional.

- **Procedimientos ETL *(Extract Transform Load)* en vez de ELT *(Extract Load Transform)***. En un DW las transformaciones de los datos se realizarán antes de su ingesta. Por el contrario, en un Data Lake, se introducirán en el sistema en "crudo" y dentro del mismo se transformarán de acuerdo a las necesidades. Ver la Figura 3.7. Por ello, el Data Lake contará con una mayor capacidad para el diseño de nuevas consultas, mientras que un Data Warehouse está limitado a las consultas para las cuáles fue diseñado.

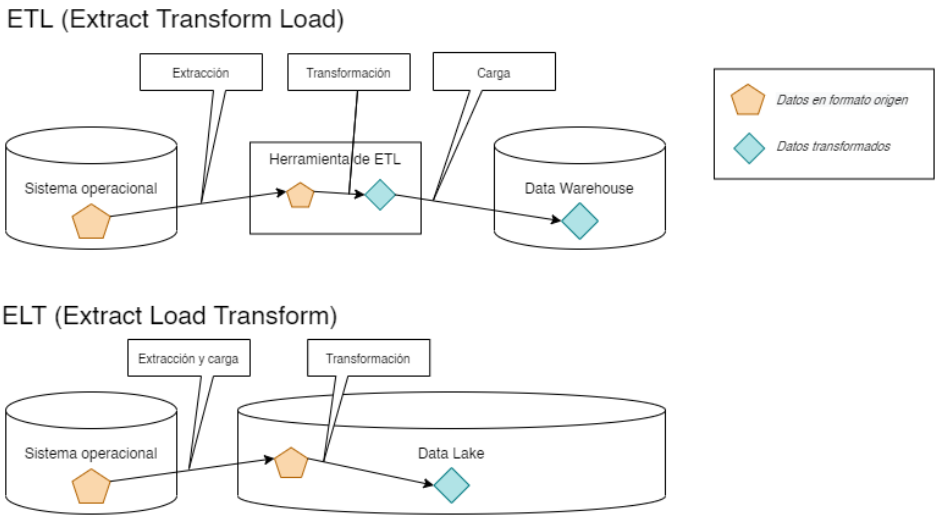


Figura 3.7: Procesos ETL y ELT *(Fuente: elaboración propia a partir de [26]).*

Los *data lakes* tienen una estructura muy flexible y poco definida. Esto facilita el acceso a la información. Sin embargo, también hace más difícil gobernar los datos más sensibles y es más difícil garantizar la seguridad de los mismos. Por el contrario, en un *data warehouse*, la fuerte estructura facilita la implementación de permisos para acceder a la información más sensible.

3.5. Arquitectura propuesta

Para este proyecto se ha decidido utilizar un Data Lake con una arquitectura Lambda simplificada, sin la capa de velocidad.

El primer motivo para la la elección de esta arquitectura es la falta de datos a tiempo real, pero su posible implementación en el futuro. La flexibilidad que aporta una arquitectura Lambda permite añadir una capa de velocidad en el futuro sin afectar a la capa por lotes. En segundo lugar, los datos con los que se trabaja se componen de archivos CSV y JSON. Los archivos CSV podrían ser almacenados fácilmente en un Data Warehouse gracias a su estructura tabular. No obstante, los archivos JSON que actualmente son utilizados no disponen de una estructura tabular y no se extrae, en estos momentos, todos los datos ahí almacenados. Además, es posible que en proyectos futuros se incorporen datos con otros formatos.

La decisión del uso de un Data Lake frente a una solución de Warehousing viene dada por la variedad de los datos que se prevé que puedan ser incorporados al sistema. Estos constarán de datos de diferentes ligas y nuevos datos como información acerca de los tiros realizados o la incorporación de datos de sistemas como SportVU[1], que aumentarían la complejidad y volumen de los datos enormemente.

[1] `https://www.statsperform.com/team-performance/basketball/` SportVU es un sistema de cámaras que recoge información acerca de la posición de los jugadores a tiempo real.

Capítulo 4

Tecnologías y ecosistemas

En el Capítulo 3 se proponen arquitecturas, pero sin concretar ninguna tecnología concreta ni entorno. En este, se describirán diferentes entornos y tecnologías concretas para implementar cada uno de los componentes de la arquitectura propuesta.

4.1. Hadoop

Hadoop es un ecosistema de tecnologías para el almacenamiento y procesamiento de datos en sistemas distribuidos [49]. Fue creada por Doug Cutting, creador de Apache Lucene (un motor de búsqueda de código abierto). Hadoop origina de Apache Nutch, un buscador de páginas web que formaba parte del proyecto Apache Lucene.

El desarrollo de este proyecto fue extremadamente complicado, ya que un motor de búsqueda web requiere de muchos componentes. Sin embargo, su objetivo era claro y prosiguieron con el desarrollo. Tras haber desarrollado un *web scraper* (software usado para leer el contenido de páginas web), se hizo evidente que la arquitectura no escalaría lo suficiente como para manejar la información de miles de millones de páginas web. En 2003, desde Google, se publicó un artículo (*The Google File System [11]*) sobre la arquitectura usada en el sistema de archivos que utilizaba la compañía para hacer frente a los ficheros de gran tamaño generados por los *web crawlers*.

Figura 4.1: Logo de Apache Hadoop ©.

En 2004, los creadores de Apache Nutch crearon el NDFS (Nutch Distributed File System): una implementación de código abierto del GFS. Ese mismo año Google publicó un nuevo artículo que introdujo MapReduce [6]. “MapReduce es un modelo de programación

y su asociada implementación para el procesamiento y generación de grandes conjuntos de datos" [6]. El uso de un modelo funcional (operaciones *map* y *reduce*) permite paralelizar operaciones de gran magnitud de manera sencilla usando la recomputación como mecanismo principal para la tolerancia de fallos. A comienzos de 2005, los desarrolladores de Nutch ya habían desarrollado una implementación de MapReduce para NDFS. Al tener más casos de uso que la implementación de motores de búsqueda, se creó un nuevo proyecto en 2006 para llevarlo a cabo: Hadoop (ver logo en la Figura 4.1).

De esta manera, el proyecto Hadoop engloba todo un ecosistema de tecnologías que trabajan de manera conjunta resolviendo cada problema específico y que se mencionarán en este artículo. Cabe destacar que es el ecosistema más maduro y extendido, e incluso las aplicaciones comerciales están basadas en él.

4.1.1. Hadoop Distributed File System (HDFS)

El gran volumen de datos que requiere manejar exige una amplia escalabilidad. Para ello, HDFS escalará horizontalmente en vez de verticalmente. Esto supone, usar más nodos en vez de mejorar las prestaciones de la máquina, como se puede ver en la Figura 4.2. De esta manera, se consigue un sistema que elimina los puntos de fallo únicos y al mismo tiempo aumenta la disponibilidad del mismo.

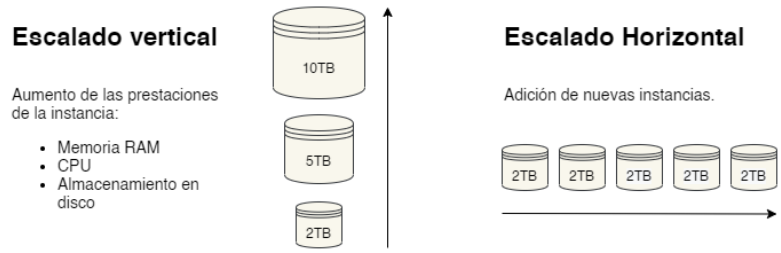


Figura 4.2: Escalado vertical y horizontal.

HDFS está programado en Java, y está optimizado para trabajar con archivos de gran tamaño. Para ello, HDFS divide los archivos en bloques de entre 64 MB y 256 MB, y estos bloques son repartidos y replicados por los diferentes nodos del sistema de archivos.

A nivel de nodos podremos diferenciar entre dos tipos: nodos maestros (NameNode) y nodos esclavos (DataNode). El NameNode contendrá la información de los bloques y los archivos, y los guardará en memoria RAM (lo que agiliza el procesamiento). Los DataNodes almacenarán la información de cada bloque. Estos bloques se repartirán de manera uniforme y replicada en diferentes DataNodes del sistema, para prevenir la pérdida de datos ante fallos del sistema y para agilizar la lectura, ya que al tener un archivo repartido en numerosos ordenadores, permite leerlos con mayor velocidad. Cabe destacar en este apartado la importancia de la robustez, ya que si tenemos un sistema con 100 discos duros y se cuenta con una vida media de 3 años por disco, es de esperar un fallo de disco cada 10 días (35 fallos de disco al año). Afortunadamente, HDFS permite recomponer un nodo tras haber fallado, y todo esto

sin que el usuario de los datos se inmute.

En la Figura 4.3 se puede apreciar la distribución habitual de los bloques en el sistema de archivos. El rectángulo naranja representa el NameNode, y los amarillos el destino de los bloques. `d1` y `d2` representan dos centros de datos independienes, `r1, r2` y `r3` diferentes racks de ordenadores y `n1`, `n2`, `n3` y `n4` los diferentes nodos. Las "d" en los arcos de líneas discontinuas representan una distancia que indica la cercanía de los nodos al nodo `n1` (el NameNode). De esta manera, se procura distribuir los datos entre diferentes ordenadores, racks de discos e incluso centros de datos, para garantizar la seguridad de la información almacenada [35].

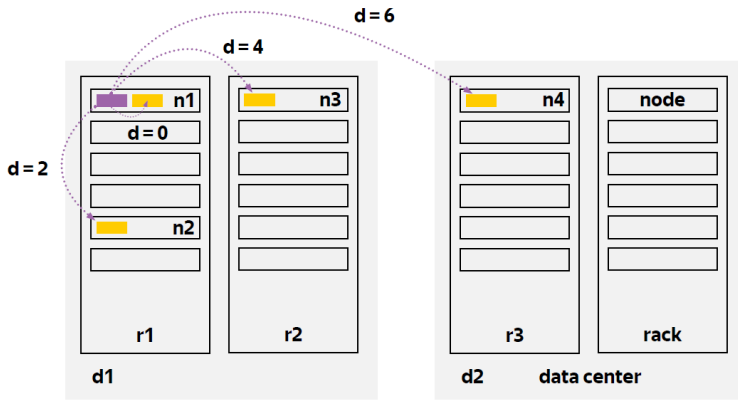


Figura 4.3: Distribución de bloques en HDFS *(Fuente: [35])*.

4.1.2. YARN: Yet Another Resource Negotiator

Apache YARN es un administrador de recursos para un *cluster* de Hadoop. Fue introducido para mejorar la implementación de MapReduce, y por ello es también llamado MapReduce Versión 2. YARN aporta una API para trabajar con los recursos de un *cluster* de ordenadores de Hadoop. Sin embargo, no es común usar directamente la API de YARN, si no que se programa sobre otras APIs de mayor nivel de abstracción como MapReduce o Spark [30]. En la Figura 4.4 se puede ver cómo funciona YARN junto a otras aplicaciones del ecosistema.

4.1.3. Spark

Apache Spark es un motor unificado y un conjunto de librerías para le procesamiento paralelo de datos en un *cluster* de ordenadores. Spark soporta los lenguajes Python, Java, Scala y R, lo que lo hace accesible a un gran número de programadores y científicos de datos. Spark cuenta también con librerías de diferente índole, como SQL, aprendizaje automático y *streaming*. Es, por tanto, adecuado para proyectos tanto de pequeña como gran escala [3].

La principal diferencia entre procesos MapReduce y Spark

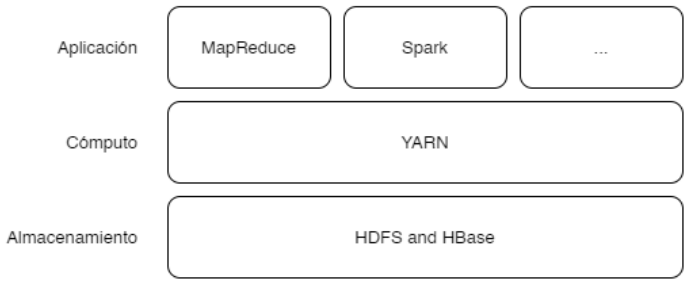


Figura 4.4: Aplicaciones [30].

Figura 4.5: Logo de Spark ©.

Como herramienta de cómputo, podría ser comparada con MapReduce. Sin embargo, ambas tecnologías cuentan con diferencias que las hacen adecuadas para diferentes casos de uso. Por un lado, tenemos que Spark realiza los cálculos en memoria, en vez de en disco como MapReduce. Esto lo hace más rápido, pero al mismo tiempo exige más recursos. Por tanto, para trabajos que necesiten ser procesados en tiempo real, Spark será la herramienta adecuada. Si por el contrario la latencia no es un problema, el uso de MapReduce será más indicado. En el contexto de un DataLake, Spark tendrá su lugar en la capa de velocidad, ya que pese a requerir más recursos, también trabaja con una cantidad de datos muy inferiores. Para la capa por lotes, en dónde son aceptables latencias mucho mayores, será más apropiado utilizar procesos MapReduce.

4.1.4. Hive

Hadoop, hace necesario utilizar el modelo MapReduce para aprovechar sus capacidades de cómputo a gran escala. Sin embargo, es común que los usuarios de los datos no sean sólo programadores o desarrolladores de *software*. En ocasiones, incluso para gente experimentada con Java, no resulta trivial realizar operaciones como por ejemplo, la media aritmética [2]. Por otro lado, SQL es un lenguaje ampliamente conocido y utilizado para la obtención de datos y su manejo es intuitivo y resulta fácil de aprender para gente con diferentes perfiles y permite realizar fácilmente operaciones de consulta y manipulación de dátos básicas. El cometido de Hive es el de proporcionar una interfaz de SQL para procesos MapReduce.

Hive traduce la mayor parte de estas consultas son traducidas a trabajos MapReduce. Es importante resaltar que pese a que se use un dialecto de SQL, Hive no se comporta ni debe ser usado para aplicaciones en las que se desea una funcionalidad a una base de datos tradicional por diferentes razones:

Figura 4.6: Logo de Apache Hive ©.

- **Velocidad**: los trabajos MapReduce están orientados a procesamiento por lotes *(batch processing)*, y cuentan con una latencia (de posiblemente más de 10 segundos) de lanzamiento del proceso. Esto es mucho más lento que para una base de datos convencional, y además no depende del tamaño de la consulta.
- **Imposibilidad de modificación**: Hadoop y HDFS tienen limitaciones por diseño en lo referente a la modificación y borrado de registros; algo esencial para una base de datos al uso.
- **Falta de transacciones**: Hive no implementa transacciones, a diferencia de la mayoría de bases de datos.

En caso de necesitar una funcionalidad similar a una base de datos a gran escala, sería conveniente optar por una base de datos NoSQL (ver en la sección 4.3).

4.1.5. Apache Impala

Impala es un motor de consultas que corre sobre el motor de Apache Hadoop [38]. De la misma manera que Hive, utiliza un dialecto de SQL para realizar las consultas, y a diferencia de Hive, la latencia de las consultas es mucho menor y se puede considerar interactiva. De esta manera, Apache Impala puede ser usado en la capa de servicio. Sin embargo, sufre de las mismas restricciones de Hive en lo que se refiere a la modificación y borrado de filas. Tampoco soporta transacciones, lo que lo hace diferente a una base de datos. Sin embargo, para el análisis de datos no suele ser necesario más que la consulta de datos. Impala nos permite tener la latencia de lectura de una base de datos con el beneficio de unos menores recursos que aporta el ecosistema de Hadoop. Además, Apache Impala puede realizar consultas sobre HBase; no está limitado a HDFS.

Figura 4.7: Logo de Apache Impala ©.

4.1.6. Apache Pig

Pig está diseñado para ejecutar flujos de datos en paralelo sobre Hadoop. Hace uso tanto del HDFS como de MapReduce [10].

Pig hace uso de un lenguaje de "flujo" propio, denominado Pig Latin. Esto permite a los usuarios definir cómo se debe leer, procesar y almacenar los datos hacia uno o más destinos en paralelo. Hablando de manera más precisa, un programa en Pig Latin define un grafo dirigido acíclico, donde los arcos describen flujo de datos y los nodos son operadores que procesan los datos.

Pig aporta más flexibilidad que Hive, ya que permiten trabajar con esquemas desconocidos, incompletos o inconsistentes, al poder manejar datos anidados. Es por ello una herramienta muy útil para explorar los datos que todavía no han sido limpiados ni cargados.

Figura 4.8: Logo de Apache Pig ©.

4.1.7. Apache Flume

Apache Flume es un servicio distribuido y tolerante a fallos para recoger, transformar y transportar grandes cantidades de registros *log*. Su arquitectura es sencilla y flexible, y se basa en flujos de datos.

Figura 4.9: Logo de Apache Flume ©.

4.1.8. Apache Sqoop

Sqoop es una herramienta diseñada para la transferencia masiva de datos entre Apache Hadoop y otros tipos de almacenamiento estructurados como las bases de datos [48]. Esta herramienta facilita por tanto la ingesta de datos procedentes de bases de datos relacionales, que pueden proceder de aplicaciones ya existentes que usen estos sistemas. También permite exportar los datos desde Hadoop a las bases de datos.

Figura 4.10: Logo de Apache Sqoop ©.

4.1.9. Apache Oozie

Oozie es un servicio diseñado para orquestar y dirigir flujos de trabajo en entornos Hadoop. Permite lanzar y monitorizar trabajos tanto de la terminal de comandos como de Hive, trabajos MapReduce, trabajos de Spark, etc. Si se combina con Hue, permite realizar estos trabajos mediante una interfaz gráfica, aunque si no se dispone de esta herramienta los flujos de trabajo siempre pueden ser realizados mediante ficheros XML.

Figura 4.11: Logo de Apache Oozie ©.

4.1.10. Hue

Hue es un cliente web de código abierto para Oozie, Hive, Impala y el sistema de ficheros de Hadoop (HDFS). Permite hacer peticiones de manera interactiva a Hive e Impala, y monitorizar trabajos de MapReduce. Además, facilita la exploración de los ficheros de HDFS y dispone de una interfaz gráfica para la realización de flujos de trabajo de Oozie.

Figura 4.12: Logo de Hue ©.

4.2. Plataformas de *streaming*

Un *stream* de datos es un flujo de datos de cualquier naturaleza usando cualquier medio, a tiempo real. Esto incluirá a dos de las "v" definidas en el Capítulo 3: velocidad y variedad. Se pueden diferenciar dos tipos de *streams*: acotados y no acotados. Los acotados tienen un principio y fin acotados. El tratamiento de estos datos se suele denominar tratamiento por lotes o *batch processing*. Por el contrario, los no acotados no tienen un final definido, y el procesamiento comienza desde el principio de los datos.

Una herramienta de *streaming* tiene tres funcionalidades clave:

- Publicar y suscribirse a *streamings* de diferentes registros.
- Almacenar los flujos de datos de una manera tolerante a fallos.
- Procesar los datos según ocurren.

Pese a parecer que estas plataformas no tienen nada que ver con un *data lake*, son clave para la ingesta de los datos y un primer pre-procesamiento. Un primer pre-procesamiento puede parecer ir en contra de los principios de un *data lake*, sin embargo, estas herramientas permiten transformar los datos de manera que sea más apropiado su almacenamiento en el Data Lake. Además, ante un fallo del sistema de ingesta del Data Lake, la capacidad de almacenamiento de la plataforma de *streaming* puede permitir recuperar los datos al guardar los datos que no se pudieron registrar durante el fallo del sistema. Esto también protege contra fallos en la red o fallos de codificación en la ingesta. Además, en las corporaciones, una plataforma de *streaming* supone el "sistema nervioso" de la misma, de manera que todos o prácticamente todos los datos producidos por la organización se transmiten mediante ese sistema.

4.2.1. Apache Kafka

Kafka es una herramienta de procesamiento de *streams* de datos de código abierto. Fue desarrollado inicialmente por LinkedIn en 2011 [24], y pasó a ser un proyecto de Apache Software Foundation en 2012. Usa un patrón *publish/subscribe* con colas. Apache Kafka se puede usar para introducir los datos en el sistema de manera fiable: Apache Kafka es un sistema distribuido que introduce redundancia para asegurar la recepción de los mensajes por los "suscriptores" de cada cola [7].

Figura 4.13: Logo de Apache Kafka ©.

4.2.2. Apache NiFi

Apache NiFi se define en su página web como "Un sistema para procesar y distribuir datos fácil de usar, potente y fiable". Cuenta con una interfaz web muy intuitiva y que permite realizar numerosas operaciones sobre los datos, además de una monitorización del flujo de datos y los *buffers* y componentes del mismo.

Figura 4.14: Logo de Apache NiFi ©.

4.3. Almacenamiento NoSQL

En casos que se necesite un sistema de almacenamiento con un alto número de escrituras y lecturas pero que el volumen de datos sea del orden de terabytes, un sistema de bases de datos relacional clásico no es apto. Por este motivo se crearon las bases de datos no relacionales o

"no SQL". Estos sistemas de bases de datos tienen las siguientes ventajas frente a un sistema de almacenamiento de datos relacional [46]:

- Utilizan menos recursos que los sistemas relacionales.
- Capacidad de manejar mayor cantidad de datos, que es posible ya que utiliza una estructura distribuida.
- Escala horizontalmente, al contrario que las bases de datos relacionales. Esto se debe a la estructura distribuida de NoSQL frente a la de maestro-esclavo de la mayoría de sistemas relacionales o SQL.
- Utilizan un lenguaje de consultas específico, y en general diferente para cada sistema.
- No se utilizan tablas ni estructuras fijas para almacenar los datos.
- Las operaciones *JOIN* no suelen estar permitidas o restringidas (no permitiendo realizar *JOIN* por un atributo que no sea la clave). Esto se debe a que cuando no se realiza esta operación por clave es muy costosa.

Las clasificaciones más comunes de este tipo de bases de datos son las orientadas a documentos y las orientadas a columnas.

4.3.1. Orientadas a documentos

Las orientadas a documentos serán aquellas que almacenan datos semiestructurados, y pueden llegar a ofrecer funcionalidades similares a las de una base de datos tradicional, como el de consultar y filtrar de una manera similar a una SQL. Los ejemplos más importantes de este tipo de herramienta son:

- **MongoDB**: esta base de datos cuenta además con un conector para Hadoop, de manera que puede ser integrada fácilmente.
- **CouchDB**.
- **Amazon DynamoDB**.

4.3.2. Orientadas a columnas

Las bases de datos orientadas a columnas se especializan en la agregación de grandes cantidades de datos. Se diferencian de las bases de datos relacionales en la manera de almacenar los datos, principalmente, como se puede ver en la Figura 4.15.

Ejemplos de este tipo de base de datos son:

- **HBase**: forma parte del ecosistema de Hadoop.
- **Cassandra**.
- **BigTable**: desarrollada por Google.

Figura 4.15: Ejemplo de datos almacenados en una base de datos relacional y en una columnar.

4.4. Plataformas

Las herramientas vistas a lo largo de este capítulo trabajan en conjunto para realizar todas las tareas que un sistema de *Big Data* debe realizar. Sin embargo, la puesta en marcha de las mismas es laboriosa y puede llevar mucho esfuerzo. Para solucionar este problema, han surgido plataformas que facilitan la puesta en marcha y coordinación de todos estos servicios.

4.4.1. Plataformas instalables

En primer lugar hablaremos de plataformas instalables. Estas plataformas suelen distribuir imágenes de máquinas virtuales o contendedores de Docker que permiten instalarlas en una máquina propia. Aportan una configuración inicial para la mayoría de los servicios y simplemente requieren un cambio de configuración (para la mayoría de casos de uso). Por tanto, son indicadas tanto para aprendizaje de entornos de Big Data como para proyectos de tamaño restringido.

4.4.1.1. Cloudera

Cloudera fue fundada en 2008 y fue una de las primeras empresas en distribuir *software* del entorno Hadoop. La empresa trabaja con sistemas que hacen uso de herramientas de *software* libre como Apache NiFi, YARN MapReduce, Hive, Sqoop, etc. Sin embargo, las complementa con una herramienta propietaria de gestión de recursos y que facilita enormemente la gestión y la configuración del *cluster*: Cloudera Manager. Esta herramienta consta de una interfaz web que permite cambiar los parámetros de configuración de cada una de las herramientas anteriormente mencionadas en un sólo lugar, lo que facilita enormemente el desarrollo.

4.4.1.2. HortonWorks

Hortonworks, antes de unirse a Cloudera en 2019, disponía de plataformas similares a las descritas en el apartado anterior.

4.4.1.3. MapR

MapR es una compañía fundada en 2009 que ofreció distribuciones de Hadoop y su ecosistema hasta 2019 a causa de una falta de capital. Esta compañía ha contribuido al desarrollo de Apache Hive, Pig y HBase.

4.4.2. Plataformas *cloud on-demand*

Uno de los beneficios de plataformas *cloud* escalables *on-demand* como las ofrecidas por Amazon, Google o Microsoft, han sido adoptadas por un gran número de organizaciones. Los beneficios de este tipo de servicios es numeroso. En primer lugar, permite ampliar los recursos en caso de necesidad, sin tener que adquirir nuevas máquinas. Además, estos recursos adicionales pueden ser contratados por un corto periodo de tiempo, lo que reduce los costes. Si tenemos en cuenta que muchas de las herramientas anteriores escalan de manera prácticamente lineal, será posible utilizar recursos para un único trabajo durante un corto periodo de tiempo o de pocos recursos durante mucho tiempo. El costo económico para la corporación será similar, pero el ahorro de tiempo puede ser esencial. Por último, estos proveedores garantizan una gran disponibilidad, y tienen una capacidad de escalado virtualmente ilimitada (siempre se podrá disponer de computación o espacio en disco).

4.4.2.1. Amazon Web Services

Amazon Web Services (AWS) es una empresa subsidiaria de Amazon surgida en 2006. Sus servicios se basan en un modelo de "pago por uso", aunque también permite alquilar por máquinas virtuales y ordenadores dedicados, como la gran mayoría de proveedores.

La base de los servicios de Amazon es EC2 *(Elastic Cloud Computing)* y S3 *(Simple Storage Service)*. El primero permite contratar y configurar recursos de cómputo y memoria RAM de manera variable, mientras que el segundo proporciona un almacenamiento barato, seguro y fácilmente ampliable. Alrededor de los mismos, se han desarrollado otros servicios, de los cuales se describirán los más conocidos orientados a *Big Data*.

- **Amazon EMR *(Elastic MapReduce)***: este servicio configura procesos de EC2 y permite realizar operaciones de MapReduce sobre su sistema de almacenamiento S3. Permite utilizar herramientas como Spark, Hive, ...
- **Amazon MSK *(Managed Streaming for Apache Kafka)***: servicio totalmente gestionado de Apache Kafka en sus servidores. Dispone de una consola propietaria para configurar, monitorizar y desplegar *clusters* de Apache Kafka.
- **DynamoDB**: base de datos NoSQL propietaria. Diseñada para grandes volúmenes de datos, desde Amazon Web Services afirman que es capaz de manejar 1 billón de peticiones por día y picos de 20 millones de peticiones por segundo.

4.4.2.2. Google Cloud

Google Cloud ofrece una cantidad de servicios con un sistema de pago y escalado similar a los de AWS. De manera similar, cuenta con numerosos servicios orientados a gestionar y tratar grandes volúmenes de datos. Las tecnologías y servicios más relevantes son:

- **Cloud Bigtable**: base de datos no relacional gestionada por Google. Cuenta con capacidad de gestionar millones de consultas por segundo [12].
- **Cloud Dataflow**: alternativa propietaria para Apache Kafka o Amazon MSK.
- **Cloud Dataproc**: permite la ejecución de trabajos de Hadoop o Spark sobre servicios de almacenamiento de objetos (OSS) como S3.

4.4.2.3. Microsoft Azure

- **Cosmos DB**: propuesta de Microsoft para una base NoSQL.
- **Azure Data Lake (HDInsight)**: plataforma de Hadoop administrada que proporciona clústeres para Spark, Hive, Map Reduce, HBase, Storm y Kafka. Los datos son almacenados en HDFS.

Capítulo 5

Datos

En este capítulo se estudiarán diferentes fuentes de datos, los datos de los que se dispone y las transformaciones realizadas sobre los mismos para su limpieza y enriquecimiento.

5.1. Fuentes

En este apartado, se estudiarán diferentes fuentes que proveerán de datos a nuestro sistema. Pese a la gran cantidad y variedad de datos, en el resto de trabajos del proyecto descrito en el Apartado 1.2, se estudiarán los estadísticos referentes a los *boxscore*. Se deberá incorporar al menos una fuente capaz de aportar esos datos. En primer lugar, se estudió el uso de APIs que permitiesen el acceso a los datos de manera sencilla para una máquina:

- **iSports API**: empresa cuyo producto son APIs que ofrecen acceso tanto a tiempo real como posterior a los partidos (a un precio más reducido). Sin embargo, el paquete más barato tenía un coste de 99 $ al mes.
- **SportsDataIO**: compañía similar a la anterior, que ofrece una API con datos a tiempo real, pero sin posibilidad de planes gratuitos.
- **ProBasketball API**: API que ofrece las estadísticas buscadas por un precio de 20 $ al mes. Sin embargo, esta no ofrece datos a tiempo real, si no actualizados diariamente.
- **BallDontLie**: API gratuita que aporta diferentes datos sobre los partidos, pero la información de los *boxscores* es parcial.

Al no encontrar una API que ofreciese los datos requeridos, se procedió al estudio de páginas web de las que extraer los datos mediante *web scraping*.

- **NBA**: página oficial de la NBA. Se muestran las estadísticas *boxscore* de cada partido. Sin embargo, la información de temporadas anteriores es limitada y la navegación dentro de la página engorrosa.
- **ESPN**: página de la cadena deportiva ESPN. Ofrece los *boxscores* de cada partido y ciertas estadísticas adicionales. A pesar de ello, la compleja navegación por la página y la información limitada de otras temporadas lo hace indeseable para el uso de esta técnica.

- **Basketball Reference**: esta página cuenta con los registros completos de la NBA, desde su fundación. Además, esta web tiene una navegación relativamente sencilla y la estructura de su código HTML es consistente. Además, cuenta con un enorme número de estadísticas y datos complementarios.

Finalmente se ha decidido consultar la fuente Basketball Reference. No se consultarán más fuentes que esta, ya que nos aporta todos los datos necesarios. Cabe mencionar que dado que se usa la técnica de *web scraping* para la consulta de datos, no se podrá disponer de datos a tiempo real; realizar *web scraping* de manera ética implica no sobrecargar el sistema, por lo que no es viable actualizar los datos con una frecuencia de segundos ni minutos.

5.1.1. Basketball Reference

Este apartado está dedicado a explicar los datos disponibles en Basketball Reference. La cantidad de datos y estadísticas calculadas en esta *web* es muy amplia. En este apartado se describe las diferentes puntos que ofrece esta web, y que quedan resumidos en el modelo de datos de la Figura 5.2.

Figura 5.1: Logo de Basketball Reference.

5.1.1.1. Datos por jugador

Esta página dispone de una página por cada jugador, en la que se encuentran, predominantemente tablas resumen de sus estadísticas por temporada y equipo. Además, se recoge información básica como peso, posición, fecha de nacimiento, debut en la liga, etc. La información encontrada en este apartado sobre las estadísticas se dividirá en varias páginas, que se listarán a continuación[1] (los indicados con el símbolo $*$ no son descargados en el trabajo actual):

- **Lista de jugadores**: lista con los jugadores y ciertas estadísticas básicas. Se desarrolla en el Apartado 5.2.3.
- **Página principal del jugador**: página con tablas resumen de estadísticas anuales.
  - **Totales**: Recoge, agrupados por temporada y equipo, las estadísticas básicas (definición de estadísticas básicas en el Apéndice A) totales. Se puede ver un ejemplo en la Figura 5.3.
  - **Por partido**: Tabla que recoge las estadísticas de la tabla "Totales", pero divididas por el número de partidos jugados.
  - **Por 36 minutos**: Las estadísticas "por 36 minutos" representan los datos para un jugador si hubiese jugado 36 minutos. Con estas estadísticas se pretende equiparar a los jugadores independientemente del tiempo que hayan pasado en el terreno de juego.

[1] Los apartados listados con el símbolo $*$ no serán descargados por el *web scraper*, pero son listados aquí por marcar posibles pautas para un trabajo futuro, ya sea calculando o recogiendo nuevos datos.

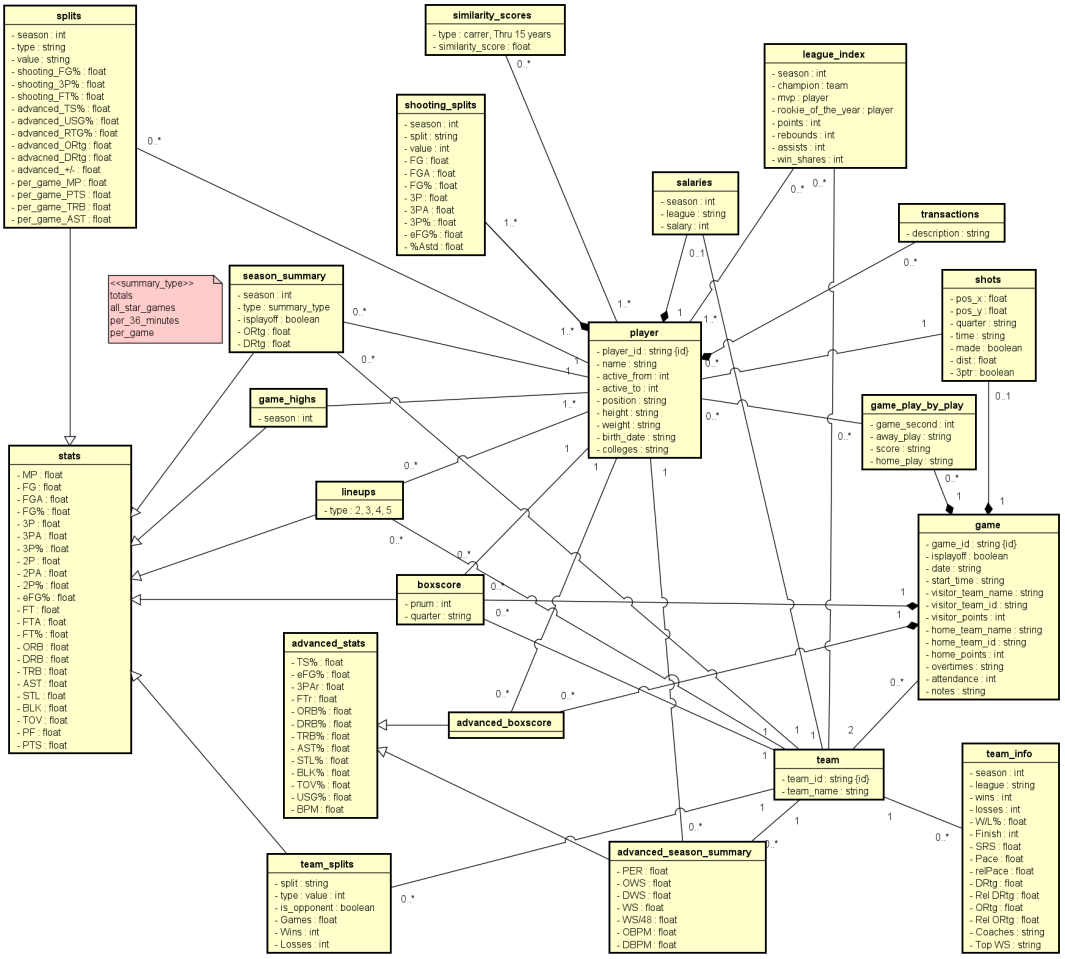


Figura 5.2: Modelo de datos de Basketball Reference.

Totals Share & more ▾ Glossary

Season	Age	Tm	Lg	Pos	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
1984-85	21	CHI	NBA	SG	**82**	82	3144	837	1625	.515	9	52	.173	828	1573	.526	.518	630	746	.845	167	367	534	481	196	69	291	285	**2313**
1985-86	22	CHI	NBA	SG	18	7	451	150	328	.457	3	18	.167	147	310	.474	.462	105	125	.840	23	41	64	53	37	21	45	46	408
1986-87	23	CHI	NBA	SG	**82**	82	**3281**	**1098**	**2279**	.482	12	66	.182	**1086**	**2213**	.491	.484	**833**	**972**	.857	166	264	430	377	236	125	272	237	**3041**
1987-88	24	CHI	NBA	SG	82	82	**3311**	**1069**	**1998**	.535	7	53	.132	**1062**	**1945**	.546	.537	**723**	860	.841	139	310	449	485	**259**	131	252	270	**2868**
1988-89	25	CHI	NBA	SG	81	81	**3255**	**966**	1795	.538	27	98	.276	**939**	1697	.553	.546	674	793	.850	149	503	652	650	234	65	290	247	**2633**
1989-90	26	CHI	NBA	SG	**82**	82	3197	**1034**	**1964**	.526	92	245	.376	**942**	**1719**	.548	.550	593	699	.848	143	422	565	519	**227**	54	247	241	**2753**
1990-91	27	CHI	NBA	SG	**82**	82	3034	**990**	**1837**	.539	29	93	.312	**961**	**1744**	.551	.547	571	671	.851	118	374	492	453	223	83	202	229	**2580**
1991-92	28	CHI	NBA	SG	80	80	3102	**943**	**1818**	.519	27	100	.270	**916**	**1718**	.533	.526	491	590	.832	91	420	511	489	182	75	200	201	**2404**
1992-93	29	CHI	NBA	SG	78	78	3067	**992**	**2003**	.495	81	230	.352	**911**	**1773**	.514	.515	476	569	.837	135	387	522	428	**221**	61	207	188	**2541**
1994-95	31	CHI	NBA	SG	17	17	668	166	404	.411	16	32	.500	150	372	.403	.431	109	136	.801	25	92	117	90	30	13	35	47	457
1995-96	32	CHI	NBA	SG	82	82	3090	**916**	**1850**	.495	111	260	.427	**805**	**1590**	.506	.525	548	657	.834	148	395	543	352	180	42	197	195	**2491**
1996-97	33	CHI	NBA	SG	82	82	3106	**920**	**1892**	.486	111	297	.374	809	**1595**	.507	.516	480	576	.833	113	369	482	352	140	44	166	156	**2431**
1997-98	34	CHI	NBA	SG	**82**	82	3181	**881**	**1893**	.465	30	126	.238	**851**	**1767**	.482	.473	565	721	.784	130	345	475	283	141	45	185	151	**2357**
2001-02	38	WAS	NBA	SF	60	53	2093	551	1324	.416	10	53	.189	541	1271	.426	.420	263	333	.790	50	289	339	310	85	26	162	119	1375
2002-03	39	WAS	NBA	SF	82	67	3031	679	1527	.445	16	55	.291	663	1472	.450	.450	266	324	.821	71	426	497	311	123	39	173	171	1640
Career			**NBA**		**1072**	**1039**	**41011**	**12192**	**24537**	**.497**	**581**	**1778**	**.327**	**11611**	**22759**	**.510**	**.509**	**7327**	**8772**	**.835**	**1668**	**5004**	**6672**	**5633**	**2514**	**893**	**2924**	**2783**	**32292**
13 seasons		**CHI**	**NBA**		**930**	**919**	**35887**	**10962**	**21686**	**.505**	**555**	**1670**	**.332**	**10407**	**20016**	**.520**	**.518**	**6798**	**8115**	**.838**	**1547**	**4289**	**5836**	**5012**	**2306**	**828**	**2589**	**2493**	**29277**
2 seasons		**WAS**	**NBA**		**142**	**120**	**5124**	**1230**	**2851**	**.431**	**26**	**108**	**.241**	**1204**	**2743**	**.439**	**.436**	**529**	**657**	**.805**	**121**	**715**	**836**	**621**	**208**	**65**	**335**	**290**	**3015**

Figura 5.3: Tabla de estadísticas totales de Michael Jordan en Basketball Reference.

- **Por 100 posesiones**: Estadísticas básicas por temporada y equipo recogidas por 100 posesiones.
- **Avanzadas**: Estadísticas avanzadas (ver estadísticas avanzadas en el Apéndice A) recopiladas por jugador.
- **Tiros Ajustados**: En esta tabla se compara los diferentes estadísticos de tiro (FG, 2P, 3P, eFG, FT, TS, FTr y 3PAr) con la media del resto de jugadores de la liga.
- **Tiros**: Tabla que agrupa los porcentajes de intentos de tiro a diferentes distancias, agrupadas de tres en tres pies. Se incluye también la frecuencia de mates y tiros de 3 desde la esquina. Esta tabla sólo está disponible desde la temporada 1996-1997.
- **Jugada por Jugada**: Disponible a partir de la temporada 1996-1997, dispone de información adicional sobre las faltas cometidas, faltas infringidas sobre él o el porcentaje de temporada empeñado en cada posición.
- **Récords por partido**: Récords por temporada y equipo de estadística básica. En esta tabla se indica, por ejemplo, el partido donde un jugador marcó más triples esa temporada y el número de triples.
- **Playoffs**: Para jugadores que hayan participado en los *playoffs*, se incluyen las siguientes tablas, explicadas más arriba, pero para datos exclusivamente de los mismos: Totales, Por Partido, Por 36 Minutos, Por 100 Posesiones, Avanzadas, Tiros, Jugada por Jugada y Récords por partido.
- **Partidos All-Star**: El *"All-Star"* es un evento que se realiza anualmente y enfrenta a 24 de los mejores jugadores de la liga, independientemente del equipo. Esta tabla refleja los totales de estadísticas básicas para este encuentro.
- **Puntuaciones de similitud**: Se utiliza el indicador "Puntuación de Similitud" para recoger a los jugadores con un rendimiento más parecido. Un valor próximo indica rendimiento similar, pero no un estilo de juego similar.
- **Universidad**: Se muestran las estadísticas básicas de las temporadas jugadas en ligas universitarias.
- **Salarios**: Se recogen los salarios recibidos por temporada.

- **Registro de partidos*** (*Game Logs*): tabla con las estadísticas básicas de todos los partidos de una temporada. Hay tantas páginas de *game logs* como temporadas haya jugado.
- ***Splits****: estadísticas básicas y avanzadas de un jugador por *splits*. Los *splits* consisten en agrupaciones de estadísticas según ciertos factores. De esta manera, se pueden ver los *splits* de un jugador según el día de la semana o el equipo oponente.
- **Tiros***: página con tablas detalladas sobre los tiros del jugador.
- **Alineaciones***: estadísticas agrupadas según las alineaciones con las que jugó este jugador.

5.1.1.2. Datos por partido

Por cada partido se muestra diferente información en Basketball Reference. A continuación se listarán los apartados (los indicados con el símbolo $*$ no son descargados en el trabajo actual):

- **Boxscores**: desarrollado en el Apartado 5.2.1.
- **Jugada por Jugada***: en este apartado se expone, en lenguaje natural, lo sucedido en cada momento. En este apartado se puede ver, por ejemplo, que un jugador cometió una falta personal en un momento concreto.
- **Gráfico de tiro***: mapa con la posición de los tiros realizados en cada momento. Se puede ver un ejemplo en la Figura 5.4. Se planteó extraer estos datos para este proyecto, pero por restricciones de tiempo y falta de interés por estos datos en los trabajos usuarios de este sistema, se excluyó su colección y tratamiento.

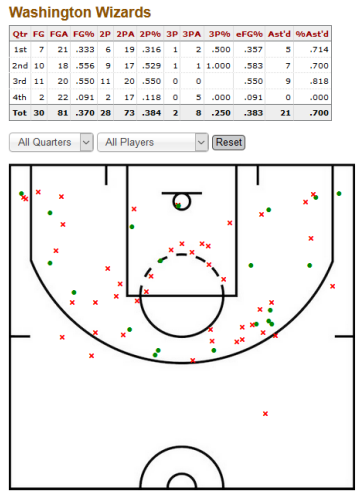


Figura 5.4: Gráfico de tiro para un partido en Basketball Reference.

5.2. Datos en crudo *(Raw layer)*

En esta sección se describirán los datos en la capa en crudo o *raw layer*. Estos datos se caracterizan por no incorporar ninguna transformación: son los datos que se han extraído directamente de las fuentes.

5.2.1. Boxscores

Los datos de partida son los denominados "boxscores básicos" de cada partido. En estas tablas se recoge la información referente a nivel de jugador y periodo del encuentro. Podemos ver las "boxscores" en la Figura 5.5, dónde se muestran las estadísticas totales del encuentro para los jugadores de *Los Angeles Lakers*.

Los Angeles Lakers (0-1) Share & more ▾ Glossary

	Basic Box Score Stats																		
Starters	**MP**	**FG**	**FGA**	**FG%**	**3P**	**3PA**	**3P%**	**FT**	**FTA**	**FT%**	**ORB**	**DRB**	**TRB**	**AST**	**STL**	**BLK**	**TOV**	**PF**	**PTS**	**+/-**
Anthony Davis	37:22	8	21	.381	0	2	.000	9	14	.643	3	6	9	5	1	2	3	3	25	+3
LeBron James	36:00	7	19	.368	1	5	.200	3	4	.750	1	9	10	8	1	1	5	3	18	-8
Danny Green	32:20	10	14	.714	7	9	.778	1	1	1.000	1	6	7	0	2	1	0	3	28	+7
Avery Bradley	24:02	3	7	.429	2	5	.400	0	0		0	3	3	0	0	0	2	3	8	-1
JaVale McGee	17:20	2	3	.667	0	0		0	0		1	1	2	0	0	2	1	0	4	0
Reserves	**MP**	**FG**	**FGA**	**FG%**	**3P**	**3PA**	**3P%**	**FT**	**FTA**	**FT%**	**ORB**	**DRB**	**TRB**	**AST**	**STL**	**BLK**	**TOV**	**PF**	**PTS**	**+/-**
Kentavious Caldwell-Pope	27:23	0	3	.000	0	2	.000	0	0		0	3	3	3	0	0	1	5	0	-8
Dwight Howard	19:02	1	3	.333	0	0		1	1	1.000	3	3	6	1	0	1	0	4	3	-1
Quinn Cook	16:56	2	7	.286	0	3	.000	0	0		0	1	1	2	0	0	1	0	4	-9
Troy Daniels	16:14	2	6	.333	1	5	.200	1	1	1.000	0	0	0	1	0	0	1	1	6	-13
Jared Dudley	13:21	2	2	1.000	2	2	1.000	0	0		0	0	0	0	0	0	0	2	6	-20
Alex Caruso	Did Not Play																			
Zach Norvell	Did Not Play																			
Kostas Antetokounmpo	Did Not Play																			
Team Totals	**240**	**37**	**85**	**.435**	**13**	**33**	**.394**	**15**	**21**	**.714**	**9**	**32**	**41**	**20**	**4**	**7**	**14**	**24**	**102**	

Figura 5.5: Estadísticas "básicas" para un partido.

Esta información es extraída mediante *Scrapy*, que genera tablas con las columnas indicadas en la Tabla 5.1, en formato CSV. Cabe destacar que aunque se extraigan las denominadas "*boxscores* avanzadas" (estadísticas calculadas a partir de las básicas), posteriormente se decidió calcularlas a partir de las "básicas". De esta manera se consigue, en primer lugar, más independencia de la fuente (ya que en otras páginas como NBA.com no se facilitaban las estadísticas avanzadas), lo que aporta flexibilidad en caso de que se tenga que cambiar de fuente. Además, sólo se dispone de estas estadísticas avanzadas por encuentro, pero no a nivel de período (cuarto, mitad y tiempo extra), a diferencia de las *boxscores* básicas. Los ficheros CSV extraídos por partido con la información de las *boxscores* básicas tienen las columnas que se indican en la Tabla 5.1.

5.2.2. Calendario de partidos (*schedule*)

La información obtenida en el apartado anterior es incompleta, ya que no indica si el partido se trata de un partido de la temporada regular o de los *playoffs*. Por tanto, resulta necesario obtener tal información. Por ello, se extraen los datos del calendario de partidos, en

Campo	Ejemplo	Descripción
`game_id`	201910220LAC	Identificador del partido.
`team_id`	LAL	Identificador del equipo al que pertenece el jugador.
`box_type`	game, q1, h2, ot1, ...	Periodo al que corresponde la estadística.
`date`	2019-10-22	Fecha del encuentro. Formato ‘‘`YYYY-MM-DD`’’.
`ishome`	true	Indica si el partido fue en casa o no.
`pnum`	1, 2, 3, ...	Indica la posición en la convocatoria. Menor o igual a 5 implicará titularidad.
`player`	LeBron James	Nombre del jugador.
`player_href`	/players/j/jamesle01.html	Link al jugador en Basketball Reference.
`player_name`	James,LeBron	Nombre alternativo usado en Basketball Reference.
`player_id`	jamesle01	Identificador del jugador.
`mp`	36:00	Minutos jugados.
`sp`	2160	Segundos jugados.
`fg`	7	Canastas anotadas (tanto de 2 como de 3 puntos).
`fga`	19	Tiros a canasta.
`fg_pct`	0.36	Porcentaje de acierto de canastas.
`fg3`	1	Triples marcados.
`fg3a`	5	Intentos de triples.
`fg3_pct`	0.20	Porcentaje de triples.
`ft`	3	Tiros libres marcados.
`fta`	4	Tiros libres intentados.
`ft_pct`	0.75	Porcentaje de tiros libres.
`orb`	1	Número de rebotes ofensivos.
`drb`	9	Número de rebotes defensivos.
`trb`	10	Número de rebotes defensivos.
`ast`	8	Número de asistencias.
`stl`	1	Robos.
`blk`	1	Tapones.
`tov`	5	Pérdidas de balón.
`pf`	3	Faltas personales.
`pts`	18	Puntos marcados.
`plus_minus`	−8	*Plus minus*.
`reason`		Comentario.
`year`	2019	Año en el que se disputó el partido.

Tabla 5.1: Columnas de *boxscores_basic* obtenidos mediante *web scrapping*.

el cuál se incluyen todos los partidos de la temporada regular y los *playoffs*, como podemos ver en la Figura 5.6. Convenientemente, también se obtiene el nombre completo del equipo en esta tabla. Los archivos generados por el *web scraper* son de formato CSV y su estructura de columnas se especifica en la Tabla 5.2.

October Schedule Share & more ▾

Date	Start (ET)	Visitor/Neutral	PTS	Home/Neutral	PTS			Attend.	Notes
Tue, Oct 22, 2019	8:00p	New Orleans Pelicans	122	Toronto Raptors	130	Box Score	OT	20,787	
Tue, Oct 22, 2019	10:30p	Los Angeles Lakers	102	Los Angeles Clippers	112	Box Score		19,068	
Wed, Oct 23, 2019	7:00p	Chicago Bulls	125	Charlotte Hornets	126	Box Score		15,424	
Wed, Oct 23, 2019	7:00p	Detroit Pistons	119	Indiana Pacers	110	Box Score		17,923	
Wed, Oct 23, 2019	7:00p	Cleveland Cavaliers	85	Orlando Magic	94	Box Score		18,846	
Wed, Oct 23, 2019	7:30p	Minnesota Timberwolves	127	Brooklyn Nets	126	Box Score	OT	17,732	
Wed, Oct 23, 2019	7:30p	Memphis Grizzlies	101	Miami Heat	120	Box Score		19,600	
Wed, Oct 23, 2019	7:30p	Boston Celtics	93	Philadelphia 76ers	107	Box Score		20,422	
Wed, Oct 23, 2019	8:30p	Washington Wizards	100	Dallas Mavericks	108	Box Score		19,816	
Wed, Oct 23, 2019	8:30p	New York Knicks	111	San Antonio Spurs	120	Box Score		18,354	
Wed, Oct 23, 2019	9:00p	Oklahoma City Thunder	95	Utah Jazz	100	Box Score		18,306	
Wed, Oct 23, 2019	10:00p	Sacramento Kings	95	Phoenix Suns	124	Box Score		18,055	
Wed, Oct 23, 2019	10:00p	Denver Nuggets	108	Portland Trail Blazers	100	Box Score		19,991	

Figura 5.6: Calendario de encuentros.

También se obtiene el calendario exclusivo de *playoffs*.

Playoffs Schedule Share & more ▾

Date	Start (ET)	Visitor/Neutral	PTS	Home/Neutral	PTS			Attend.	Notes
Sat, Apr 13, 2019	2:30p	Brooklyn Nets	111	Philadelphia 76ers	102	Box Score		20,437	
Sat, Apr 13, 2019	5:00p	Orlando Magic	104	Toronto Raptors	101	Box Score		19,937	
Sat, Apr 13, 2019	8:00p	Los Angeles Clippers	104	Golden State Warriors	121	Box Score		19,596	
Sat, Apr 13, 2019	10:30p	San Antonio Spurs	101	Denver Nuggets	96	Box Score		19,520	
Sun, Apr 14, 2019	1:00p	Indiana Pacers	74	Boston Celtics	84	Box Score		18,624	
Sun, Apr 14, 2019	3:30p	Oklahoma City Thunder	99	Portland Trail Blazers	104	Box Score		19,886	
Sun, Apr 14, 2019	7:00p	Detroit Pistons	86	Milwaukee Bucks	121	Box Score		17,529	
Sun, Apr 14, 2019	9:30p	Utah Jazz	90	Houston Rockets	122	Box Score		18,055	

Figura 5.7: Calendario de *playoffs*.

Las columnas de los datos generados mediante el *web scraper*, tanto para los calendarios de la temporada regular como los *playoffs*, que son idénticos, se indican en la Tabla 5.2.

5.2.3. Listado de jugadores

En la fuente de datos, Basketball Reference, se puede encontrar un listado con los jugadores que han participado en la NBA (ver Figura 5.8). Ofrece información como el peso, altura, o universidad en la que jugaron. Los campos que se obtienen al extraer esta información se pueden ver en la Tabla 5.3.

5.2.4. Datos del Jugador

Los datos del jugador consistirán en las tablas descritas dentro del Apartado 5.1.1.1, en la "Página principal del jugador". Por cada jugador, se almacenarán sus datos en un objeto JSON con la siguiente estructura:

Campo	Ejemplo	Descripción
`game_id`	201904130PHI	Identificador del partido.
`date`	2019-04-13	Fecha del encuentro. Formato "YYYY-MM-DD".
`start_time`	2:30p	Hora de inicio del partido.
`visitor_team`	Brooklyn Nets	Nombre del equipo visitante.
`visitor_team_id`	BRK	Identificador del equipo visitante.
`visitor_pts`	111	Puntos obtenidos por el equipo visitante.
`local_team`	Philadelphia 76ers	Nombre del equipo local.
`local_team_id`	PHI	Identificador del equipo local.
`local_pts`	102	Puntos obtenidos por el equipo local.
`overtimes`		Indica la existencia de tiempos extra.
`attendance`	20473	Público asistente al partido.
`game_remarks`		Anotaciones sobre el juego.

Tabla 5.2: Columnas de *schedule_games* y *schedule_playoffs* obtenidos mediante *web scrapping*.

Player	From	To	Pos	Ht	Wt	Birth Date	Colleges
Alaa Abdelnaby	1991	1995	F-C	6-10	240	June 24, 1968	Duke
Zaid Abdul-Aziz	1969	1978	C-F	6-9	235	April 7, 1946	Iowa State
Kareem Abdul-Jabbar*	1970	1989	C	7-2	225	April 16, 1947	UCLA
Mahmoud Abdul-Rauf	1991	2001	G	6-1	162	March 9, 1969	LSU
Tariq Abdul-Wahad	1998	2003	F	6-6	223	November 3, 1974	Michigan, San Jose State
Shareef Abdur-Rahim	1997	2008	F	6-9	225	December 11, 1976	California
Tom Abernethy	1977	1981	F	6-7	220	May 6, 1954	Indiana
Forest Able	1957	1957	G	6-3	180	July 27, 1932	Western Kentucky

Figura 5.8: Detalle del listado de jugadores en Basketball Reference.

Campo	Ejemplo	Descripción
`player`	Kareem Abdul-Jabbar	Nombre del jugador.
`player_href`	/players/a/abdulka01.html	Dirección web a la ficha del jugador en Basketball Reference.
`player_id`	abdulka01	Identificador del jugador en Basketball Reference.
`year_min`	1970	Temporada de debut en la NBA.
`year_max`	1989	Temporada de retirada de la competición.
`pos`	C	Posiciones habituales del jugador.
`height`	7-2	Altura del jugador en pies-pulgadas.
`height_csk`	86.0	Altura en pulgadas.
`weight`	225	Peso del jugador en libras.
`birth_date`	April 16, 1947	Fecha de nacimiento en lenguaje natural.
`birth_date_csk`	19470416	Fecha de nacimiento en formato ``YYYYMMDD''.
`colleges`	UCLA	Universidades.

Tabla 5.3: Columnas de *player_list*, obtenido mediante *web scraping*.

```
{
    "per_poss":[ ... ],
    "totals":[ ... ],
    ... ,
    "advanced":[
        {
            "player_id":"hardeja01",
            "season":"2009-10",
            "season_href":"/players/h/hardeja01/gamelog/2010/",
            "age":"20",
            "lg_id":"NBA",
            "lg_id_href":"/leagues/NBA_2010.html",
            "team_id":"OKC",
            "team_id_href":"/teams/OKC/2010.html",
            "pos":"SG",
            "g":"76",
            "per":"14.0",
            ... ,
            "ws-dum":null,
            "ows":"2.0",
            "dws":"2.5",
            "ws":"4.5",
            "ws_per_48":".124",
            "vorp":"1.1",
        },
        { ... },  ...
    ],
    ...
}
```


De esta manera, se almacenarán todas las tablas de la "Página principal del jugador" de Basketball Reference descritas anteriormente en un sólo archivo JSON.

5.3. Modelo de datos

Los datos de las fuentes necesitan ser transformados y enriquecidos para que resulten útiles. En este apartado se presenta un modelo de datos deseado, para tener un objetivo. Aunque los proyectos explicados en el Apartado 1.2 sólo requerían los datos de las *boxscores* básicas (por equipos y por jugador), se han introducido en este modelo otras estadísticas que se han considerado potencialmente útiles en futuros proyectos. Se pueden ver las tablas finales en la Figura 5.11.

5.4. Tratamiento de los datos

El primer paso del tratamiento de los datos consta de la extracción mediante *web scraping* y su ingesta al sistema de ficheros de Hadoop. Estas cuestiones se desarrollan en los Apartados 6.2.1 y 6.2.2. Una vez en el sistema de ficheros de Hadoop, estos datos serán enriquecidos

y limpiados, realizando transformaciones, y calculando nuevas estadísticas a partir de los mismos. Para ello, estos se organizan en tres capas:

- *Raw layer*: datos en crudo. Estos datos no se modifican, son los datos maestros. Almacenados mediante tablas externas de Hive. Ver Figura 5.9.
- *Silver layer*: datos enriquecidos y obtenidos a partir de la capa *raw* y de *silver*. Almacenados en tablas internas de Hive en formato ORC[2] para mayor velocidad de procesamiento. Ver Figura 5.10.
- *Gold layer*: datos buscados obtenidos de las capas anteriores. Listos para su análisis y exportación a la capa de servicio. Almacenados en formato Parquet[3]. Ver Figura 5.11.

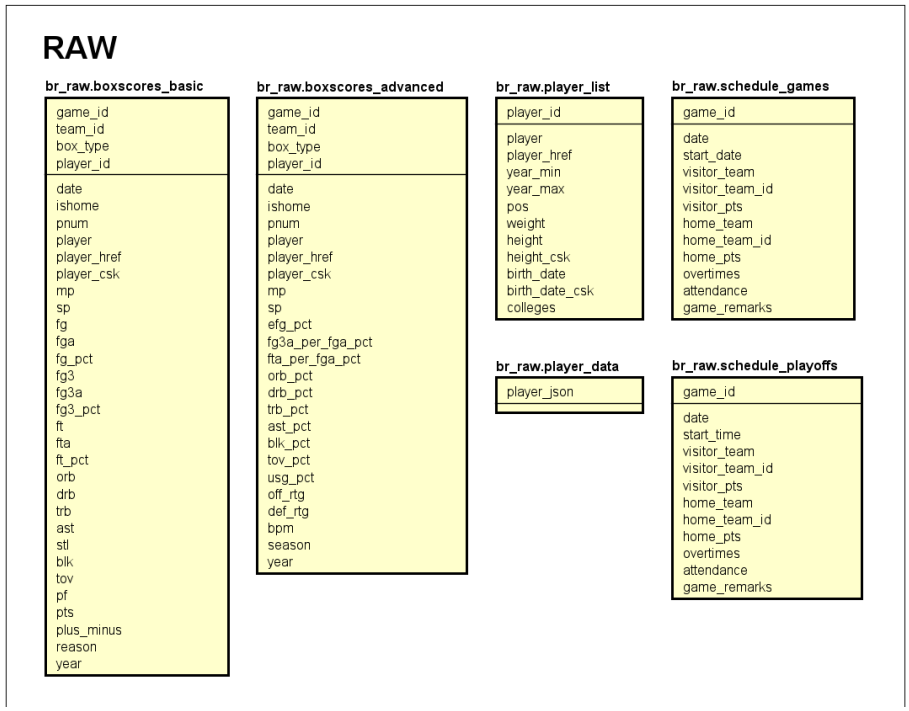


Figura 5.9: Tablas de la capa en crudo.

5.4.1. Transformaciones

Los datos de la capa en crudo *(raw layer)* han sido transformados y agrupados para llegar a los resultados deseados de la capa enriquecida *(gold layer)*. En la Figura 5.12 se presenta una visualización esquemática del flujo de datos que han seguido las sucesivas transformaciones y agrupaciones de datos. La notación que se utilizará al referirse a las diferentes tablas será: `br_<capa (raw, silver, gold)>.<nombre de la tabla>`.

[2]Formato de fichero columnar específicamente diseñado para el ecosistema Hadoop.

[3]Parquet es un formato columnar similar a ORC. Es menos eficiente al trabajar con Hive, pero a diferencia

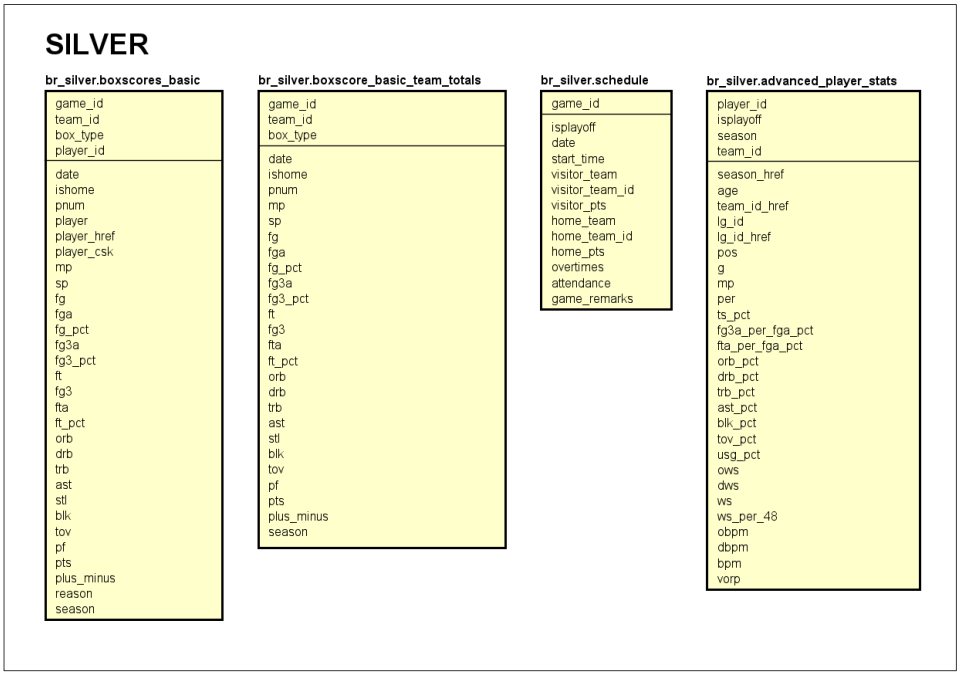


Figura 5.10: Tablas de la capa *silver*.

5.4.1.1. Capa *silver* (primera fase)

En esta sección se describirá la creación de las diferentes tablas de la capa *silver* a partir de la capa *raw*.

`br_silver.boxscores_basic` Tabla generada a partir de `br_raw.boxscores_basic`. Se introducen los siguientes cambios, implementados en el Apartado B.2.2.1 del Apéndice B:

1. Cálculo de la temporada *(season)* a partir de la fecha. Así, se toman los partidos de un año hasta el mes de septiembre inclusive como de la temporada del año actual, mientras que si la fecha es de octubre en adelante pertenecerán a la temporada del año siguiente. Por ejemplo, la fecha "03-04-2020" pertenecerá a la temporada 2019-2020 y el campo `season` será igual a `2020`, mientras que la fecha "12-10-2020" pertenecerá a la temporada 2020-2021, y el campo `season` tomará el valor `2021`.

2. Se elimina el sufijo "basic" del tipo de *boxscore (box_type)*. Esto se hace porque cuando la estadística se refiere a los valores de un juego completo, en origen toma el valor "game-basic". De esta manera se sustituye este valor por "game".

3. Se calcula si la estadística pertenece a un juego como anfitrión o como visitante a partir del identificador del partido (el identificador del partido contiene el identificador del equipo que juega en casa). Los identificadores de partido tienen la siguiente forma:

de ORC, Parquet puede ser leído por Apache Impala, que es la tecnología utilizada en la capa de servicio.

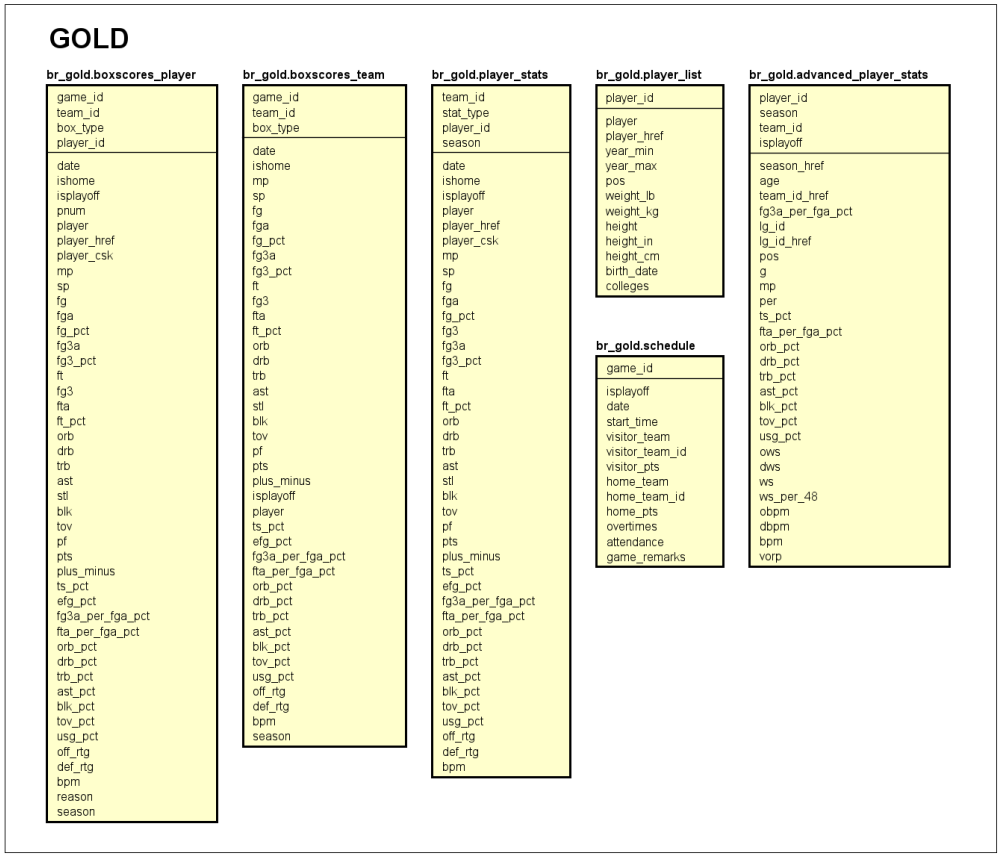


Figura 5.11: Tablas de la capa *gold*.

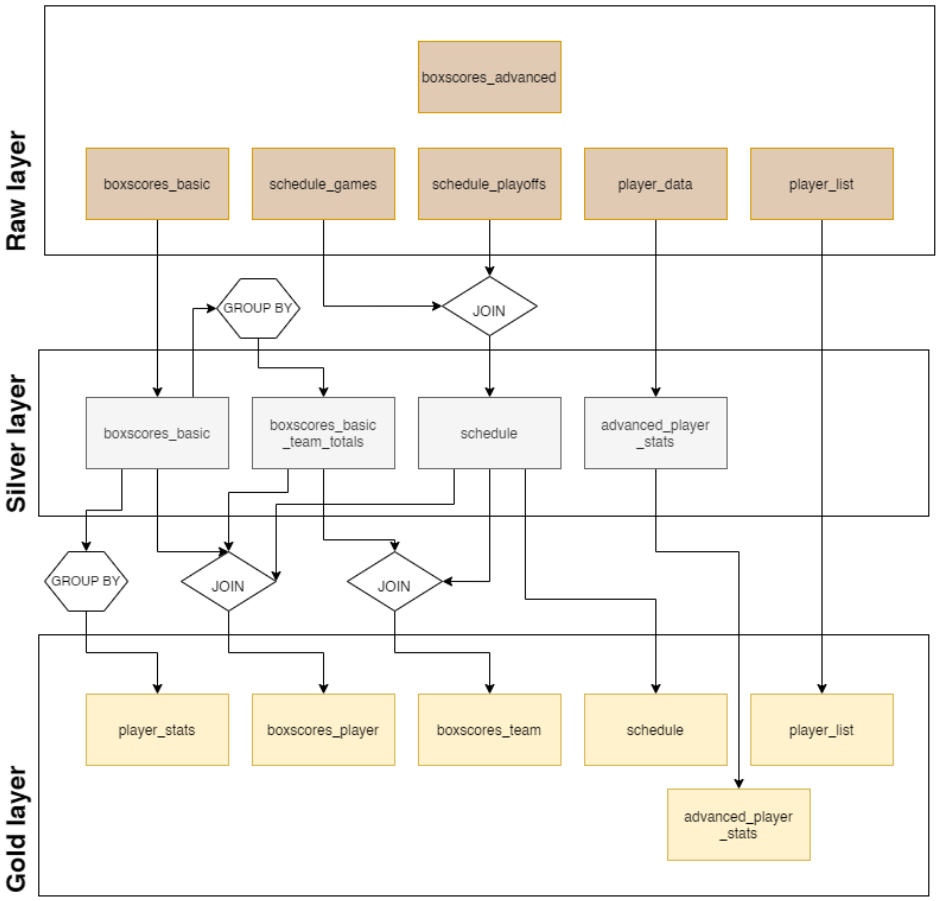


Figura 5.12: Flujo de datos entre las capas y tablas del Data Lake.

fecha en formato YYYYMMDD concatenado con un dígito y con el identificador del equipo local (por ejemplo: 201910220TOR). De esta manera, se puede determinar si el registro pertenece al equipo local o no, al ver si el campo `team_id` está contenido en el campo `game_id`.

4. Se obtiene el campo `player_id` de `player_href`. El identificador del jugador (`player_id`) se haya presente en la tabla origen (`br_raw.boxscores_basic`). Sin embargo, se ha descubierto a posteriori que no siempre está presente, a diferencia de `player_href`. De esta manera, se extrae el `player_id` haciendo uso de la función de HiveQL `regexp_extract` y se garantiza la existencia de un valor para el campo del identificador.

`br_silver.boxscores_basic_team_totals` Tabla generada a partir de `br_silver.boxscores_basic`. Esta transformación consiste en sumar las estadísticas individuales de cada jugador para hallar el total del equipo. Se puede ver con completo detalle la implementación en Hive en el Apartado B.2.2.2 del Apéndice B.

`br_silver.schedule` Tabla generada a partir de `br_raw.schedule_games` y `br_raw.schedule_playoffs`. En la primera tabla, `br_raw.schedule_games`, aparecen todos los partidos de la temporada, ya sean *playoffs* o no. En la segunda, sin embargo, sólo aparecen los de *playoffs*. De esta manera, realizando una operación JOIN, se puede determinar la variable *isplayoff*. Además, se introduce la temporada *(season)* a partir de la fecha. Ver columnas de esta tabla en las Figuras 5.9 y 5.10. El comando de Hive que realiza esta transformación se encuentra en el Apartado B.2.2.3 del Apéndice B.

`br_silver.advanced_player_stats` Tabla generada a partir de `br_raw.player_data`. Esta tabla en crudo consta de una sola columna, que es un objeto JSON perteneciente a un jugador. Para obtener las estadísticas avanzadas del mismo, y recordando la estructura que tiene este objeto JSON (ver Apartado 5.2.4), se puede extraer, desde Hive y con la ayuda de funciones específicas, los diferentes campos. Las columnas de estas tablas se muestran en las Figuras 5.9 y 5.10. El comando de Hive que realiza esta transformación se encuentra en el Apartado B.2.2.4 del Apéndice B.

5.4.1.2. Capa *gold*

`br_gold.boxscores_player` Tabla generada a partir de `br_silver.boxscores_basic`, `br_silver.boxscores_basic_team_totals` y `br_silver.boxscores_basic`. La tabla `br_gold.boxscores_player` está formada por estadísticas básicas y por estadísticas avanzadas. De las estadísticas básicas ya se dispone en `br_silver.boxscores_basic`, y para determinar si el partido es un *playoff* o pertenece a la temporada regular se utiliza `br_silver.schedules`. Sin embargo, para el cálculo de las estadísticas avanzadas hace falta calcular estadísticas de equipo (que se hayan en `br_silver.boxscores_basic_team_totals`). Estas fórmulas se han agrupado, por el orden del documento, al final del capítulo actual, en los Apartados 5.5.1 y 5.5.3. El comando de Hive que realiza esta transformación se encuentra en el Apartado B.3.2.1 del Apéndice B.

br_gold.boxscores_team Tabla generada a partir de br_silver.boxscores_basic_team_totals y br_silver.boxscores_basic. Se calcula de manera similar a br_gold.boxscores_player, pero

aplicando las fórmulas de equipo especificadas en los Apartados 5.5.2 y 5.5.3. El comando de Hive que realiza esta transformación se encuentra en el Apartado B.3.2.2 del Apéndice B.

br_gold.player_list Dado su reducido tamaño (alrededor de 5000 filas) y las escasas transformaciones que se necesita hacer en esta tabla (no se necesita realizar agrupaciones ni JOINs), resulta conveniente importar los datos directamente de la capa *raw* a la capa *gold* sin pasar por una capa de datos intermedia. De esta manera, los cambios realizados en esta tablas son:

1. Introducción del peso en el sistema métrico (kilogramos en vez de libras).
2. Introducción de la altura en el sistema métrico (centímetros en vez de pulgadas).
3. Extracción de los nombres de las universidades, ya que el *web scraper* obtuvo los datos de las universidades en HTML. Afortunadamente, la visión de *schema on read* del Data Lake posibilita acceder a estos datos sin modificar el *web scraper*.

El comando de Hive que realiza esta transformación se encuentra en el Apartado B.3.2.3 del Apéndice B.

`br_gold.player_stats` La tabla `br_gold.player_stats` recoge los datos de un jugador en una temporada. De esta manera, esta tabla facilita los datos que se habían extraído de los jugadores (tabla "Totales" por temporadas) pero añade las estadísticas avanzadas. Para ello se agrupan los datos por temporada, jugador y equipo y se suman los datos de cada estadística para calcular los totales. Lo más interesante de esta tabla es que facilita el cálculo de nuevas estadísticas, como por ejemplo "Totales de la primera mitad de temporada". Esto es así porque el *script* actual agrupa las estadísticas por temporada. Sin embargo, se puede agrupar de manera diferente, con lo que se conseguiría otras estadísticas diferentes. Si por el contrario se desease calcular las estadísticas medias por partido, se cambiaría la función de agregación por la media (AVG) en vez de por el sumatorio (SUM).

El comando de Hive que realiza esta transformación se encuentra en el Apartado B.3.2.4 del Apéndice B.

`br_gold.advanced_player_stats` Obtenida a partir de la tabla homónima de la capa *silver*, `br_silver.advanced_player_stats`, no se aplica en este caso ninguna transformación más que el cambio de formato de ORC a Parquet. El comando de Hive que realiza esta transformación se encuentra en el Apartado B.3.2.6 del Apéndice B.

5.5. Cálculos de las estadísticas

En un primer momento se pretendía usar los datos de los jugadores a partir de los extraídos en las páginas de cada jugador (en la Figura 5.13 se puede ver un ejemplo de estadísticas de un jugador). Sin embargo, estos datos pueden ser calculados mediante los *boxscores* de los partidos. Por tanto, se intentará, en la medida de lo posible, utilizar los *boxscores* para el cálculo de estas estadísticas, ya que además permitiría el cálculo de nuevas estadísticas e indicadores en el futuro. En este apartado se exponen las fórmulas de los indicadores que se han utilizado.

Per 36 Minutes

Season	Age	Tm	Lg	Pos	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
2009-10	20	OKC	NBA	SG	76	0	1738	4.8	12.0	.403	1.9	5.1	.375	2.9	6.8	.424	4.0	5.0	.808	1.0	4.1	5.1	2.8	1.7	0.4	2.2	4.1	15.6
2010-11	21	OKC	NBA	SG	82	5	2189	4.9	11.2	.436	1.9	5.3	.349	3.0	5.9	.514	4.8	5.6	.843	0.7	3.5	4.2	2.9	1.5	0.4	1.7	3.4	16.4
2011-12	22	OKC	NBA	SG	62	2	1946	5.7	11.6	.491	2.1	5.4	.390	3.6	6.2	.579	5.8	6.8	.846	0.6	4.1	4.7	4.2	1.1	0.3	2.5	2.8	19.3
2012-13	23	HOU	NBA	SG	78	78	2985	7.1	16.1	.438	2.2	5.9	.368	4.9	10.3	.477	8.1	9.6	.851	0.7	3.8	4.6	5.5	1.7	0.5	3.6	2.1	24.4
2013-14	24	HOU	NBA	SG	73	73	2777	7.1	15.6	.456	2.3	6.3	.366	4.8	9.4	.515	7.5	8.6	.866	0.8	3.7	4.5	5.8	1.5	0.4	3.4	2.3	24.0
2014-15	25	HOU	NBA	SG	81	81	**2981**	7.8	17.8	.440	2.5	6.7	.375	5.3	11.0	.480	8.6	10.0	.868	0.9	4.6	5.5	6.8	1.9	0.7	3.9	2.5	26.8
2015-16	26	HOU	NBA	SG	**82**	82	**3125**	8.2	18.6	.439	2.7	7.6	.359	5.5	11.1	.494	8.3	9.6	.860	0.7	5.0	5.8	7.1	1.6	0.6	4.3	2.6	27.4
2016-17	27	HOU	NBA	PG	81	81	2947	8.2	18.7	.440	3.2	9.2	.347	5.0	9.5	.530	9.1	10.8	.847	1.2	6.9	8.1	11.1	1.5	0.5	5.7	2.6	28.8
2017-18	28	HOU	NBA	SG	72	72	2551	9.2	20.4	.449	3.7	10.2	.367	5.4	10.3	.531	8.8	10.3	.858	0.6	4.9	5.5	8.9	1.8	0.7	4.4	2.4	30.9
2018-19	29	HOU	NBA	PG	78	78	2867	10.6	24.0	.442	4.7	12.9	.368	5.8	11.1	.528	9.5	10.8	.879	0.8	5.7	6.5	7.4	2.0	0.7	4.9	3.1	35.4
2019-20	30	HOU	NBA	SG	68	68	2483	9.7	22.0	.444	4.3	12.2	.355	5.4	9.7	.556	10.0	11.6	.865	1.0	5.5	6.5	7.4	1.8	0.9	4.5	3.3	33.9
Career			**NBA**		**833**	**620**	**28589**	**7.8**	**17.5**	**.443**	**2.9**	**8.1**	**.363**	**4.8**	**9.5**	**.511**	**7.9**	**9.2**	**.858**	**0.8**	**4.8**	**5.6**	**6.6**	**1.7**	**0.6**	**3.9**	**2.8**	**26.4**
8 seasons		HOU	**NBA**		**613**	**613**	**22716**	**8.4**	**19.1**	**.443**	**3.2**	**8.8**	**.362**	**5.3**	**10.3**	**.512**	**8.7**	**10.1**	**.862**	**0.8**	**5.0**	**5.9**	**7.5**	**1.7**	**0.6**	**4.3**	**2.6**	**28.8**
3 seasons		OKC	**NBA**		**220**	**7**	**5873**	**5.1**	**11.6**	**.444**	**2.0**	**5.3**	**.370**	**3.2**	**6.3**	**.506**	**4.9**	**5.8**	**.835**	**0.7**	**3.9**	**4.6**	**3.3**	**1.4**	**0.4**	**2.1**	**3.4**	**17.1**

Figura 5.13: Modelo de datos.

5.5.1. Estadísticas avanzadas individuales

Las estadísticas individuales de cada jugador se calculan según las siguientes fórmulas [1] descritas a continuación. Los subíndices *"player"*, *"team"* y *"opp[onent]"* indicarán que la estadística concreta pertenece al propio jugador, al equipo o al equipo oponente, respectivamente.

- **TS %** (True Shooting Percentage):

$$\frac{PTS_{player}}{2 \times TSA_{player}} = \frac{PTS_{player}}{2 \times (FGA_{player} + 0{,}44 * FTA_{player})}$$

- **eFG %** (Effective Field Goal Percentage):

$$\frac{FG_{player} + 0{,}5 \times 3P_{player}}{FGA_{player}}$$

- **3PAr** (Effective Field Goal Percentage):

$$\frac{3PA_{player}}{FGA_{player}}$$

- **FTr** (Free Throw Attempt Rate):

$$\frac{FTA_{player}}{FGA_{player}}$$

- **ORB %** (Offensive Rebound Percentage):

$$100 \times \frac{ORB_{player} \times MP_{team}/5}{MP_{player} \times (ORB_{team} + DRB_{opponent})}$$

- **DRB %** (Defensive Rebound Percentage):

$$100 \times \frac{DRB_{player} \times MP_{team}/5}{MP_{player} \times (DRB_{team} + ORB_{opponent})}$$

- **TRB %** (Total Rebound Percentage):

$$100 \times \frac{TRB_{player} \times MP_{team}/5}{MP_{player} \times (TRB_{team} + TRB_{opponent})}$$

- **AST %** (Assist Percentage):

$$100 \times \frac{AST_{player}}{(MP_{player}/(MP_{team}/5) \times FG_{team}) - FG_{player}}$$

- **STL %** (Steal Percentage):

$$100 \times \frac{STL_{player} \times MP_{team}/5}{MP_{player} \times Poss}$$

- **BLK %** (Block Percentage):

$$100 \times \frac{BLK_{player} \times MP_{team}/5}{MP_{player} \times (FGA_{opponent} \times 3PA_{opponent})}$$

- **TOV %** (Turnover Percentage):

$$100 \times \frac{TOV_{player}}{FGA_{player} + 0{,}44 \times FTA_{player} + TOV_{player}}$$

- **USG %** (Usage Percentage):

$$100 \times \frac{(FGA_{player} + 0{,}44 \times FTA_{player} + TOV_{player}) * MP_{team}/5}{MP_{player} \times (FGA_{team} + 0{,}44 \times FTA_{team} + TOV_{team})}$$

5.5.2. Estadísticas avanzadas de equipo

- **TS %** (True Shooting Percentage):

$$\frac{PTS_{team}}{2 \times TSA_{team}} = \frac{PTS_{team}}{2 \times (FGA_{team} + 0{,}44 * FTA_{team})}$$

- **eFG %** (Effective Field Goal Percentage):

$$\frac{FG_{team} + 0{,}5 \times 3P_{team}}{FGA_{team}}$$

- **3PAr** (Effective Field Goal Percentage):

$$\frac{3PA_{team}}{FGA_{team}}$$

- **FTr** (Free Throw Attempt Rate):

$$\frac{FTA_{team}}{FGA_{team}}$$

- **ORB %** (Offensive Rebound Percentage):

$$100 \times \frac{ORB_{team}}{ORB_{team} + DRB_{opponent}}$$

- **DRB %** (Defensive Rebound Percentage):

$$100 \times \frac{DRB_{team}}{DRB_{team} + ORB_{opponent}}$$

- **TRB %** (Total Rebound Percentage):

$$100 \times \frac{TRB_{team}}{TRB_{team} + TRB_{opponent}}$$

- **AST %** (Assist Percentage):

$$100 \times \frac{AST_{team}}{FG_{team}}$$

- **STL %** (Steal Percentage):

$$100 \times \frac{STL_{team}}{Poss}$$

- **BLK %** (Block Percentage):

$$100 \times \frac{BLK_{team}}{FGA_{opponent} \times 3PA_{opponent}}$$

- **TOV %** (Turnover Percentage):

$$100 \times \frac{TOV_{team}}{FGA_{team} + 0{,}44 \times FTA_{team} + TOV_{team}}$$

- **USG %** (Usage Percentage): siempre será igual a 100.
- **ORtg** (Offensive Ranking):

$$100 \times \frac{PTS_{team}}{Poss}$$

- **DRtg** (Deffensive Ranking):

$$100 \times \frac{PTS_{opponent}}{Poss}$$

5.5.3. Estadísticas auxiliares

- **Poss** (posesiones estimadas):

$$\begin{aligned} 0{,}5 \times ((FGA_{team} + 0{,}4 * FTA_{team} - 1{,}07 \times \frac{ORB_{team}}{ORB_{team} + DRB_{opp}} \times \\ (FGA_{team} - FG_{team}) + TOV_{team}) + \\ +(FGA_{opp} + 0{,}4 \times FTA_{opp} - 1{,}07 \times \frac{ORB_{opponent}}{ORB_{opp}DRB_{team}} \times \\ (FGA_{opp} - FG_{opp}) + TOV_{opp})) \end{aligned}$$

Capítulo 6

Desarrollo

1En este caso se optará por un modelo de repositorio, diseñado para describir cómo un conjunto de componentes pueden compartir datos [44]. Este patrón de diseño se basa en los grandes sistemas, que comúnmente disponen sus datos alrededor de una base de datos o repositorio central. Esta es la función para un *Data Lake*.

6.1. Software e infraestructura

6.1.1. Herramientas auxiliares

En esta sección se explicará brevemente el *software* auxiliar que se ha utilizado para realizar este proyecto. Estas herramientas han sido utilizadas desde un ordenador personal con Microsoft Windows 10 como sistema operativo.

Visual Studio Code Editor de texto de código abierto con un gran número de funcionalidades desarrollado por Microsoft. Cuenta con numerosas extensiones para facilitar el desarrollo en distintos lenguajes de programación. Ha sido utilizado para el desarrollo de todo el código; tanto del *web scraper* (código en Python), *scripts* de HiveQL, interfaz *web* (HTML, CSS y JavaScript).

MobaXterm Terminal de Windows que cuenta con un cliente SSH, un cliente gráfico FTP y tunelado por SSH, entre otras herramientas, para el trabajo con máquinas virtuales. Dispone también de una monitorización de uso de CPU, memoria RAM y porcentaje de disco disponible. Es una herramienta que cuenta con una versión gratuita y ha sido extremadamente útil para el acceso a la máquina virtual y evitar el uso de escritorios virtuales.

X2Go Escritorio virtual de código abierto para controlar de manera remota el servidor de producción (ver Apartado 6.1.2). Se ha usado de manera puntual, ya que se ha preferido trabajar tunelando los puertos.

PuTTY Emulador de terminal de código abierto que permite, además, tunelar los puertos. Dada la limitación de MobaXterm a 3 puertos para tunelar por SSH, se ha utilizado PuTTY en momentos en los que fue necesario el uso de más puertos.

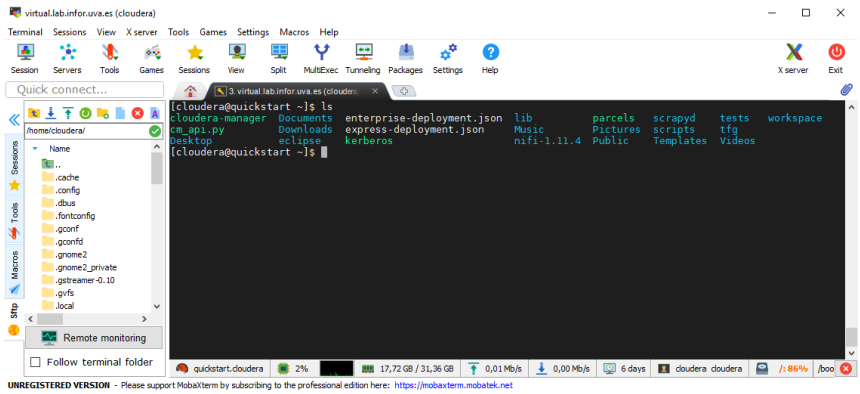


Figura 6.1: Terminal de MobaXterm conectado a la máquina de Cloudera.

Texmaker Editor de LaTeX multiplataforma y de código abierto. Se ha utilizado para la escritura la memoria. Dispone de atajos de teclado, un lector integrado de PDF y VDI y una amplia configuración.

GitLab Repositorio de la herramienta de control de versiones Git. Se ha hecho uso del servidor de Git de la Escuela de Ingeniería Informática de la Universidad de Valladolid para el control de versiones y para mantener una copia de seguridad.

Astah Herramienta de modelado para la elaboración de diagramas relacionados con la ingeniería de *software*. Se ha hecho uso de esta herramienta para la creación de diagramas para el modelado de datos. Pese a ser una herramienta propietaria, se dispone de una licencia en la presente Escuela.

Drawio Herramienta gratuita para el desarrollo de diagramas explicativos, como el de la Figura 6.2.

6.1.2. Sistema de producción

El sistema de producción se trata de una máquina con una distribución Cloudera 5.16, basada en CentOS. El resto de herramientas de producción que se han utilizado en el desarrollo del proyecto son:

1. Scrapy y Scrapyd.
2. Apache NiFi.
3. Apache Hadoop HDFS
4. Apache Oozie.
5. Apache Hive.
6. Apache Impala.

7. Flask junto a la librería Impyla.

Las herramientas arriba y su utilidad se explicarán a lo largo de este capítulo. En la Figura 6.2 se muestra un diagrama que muestra las tecnologías utilizadas y su función principal en el sistema.

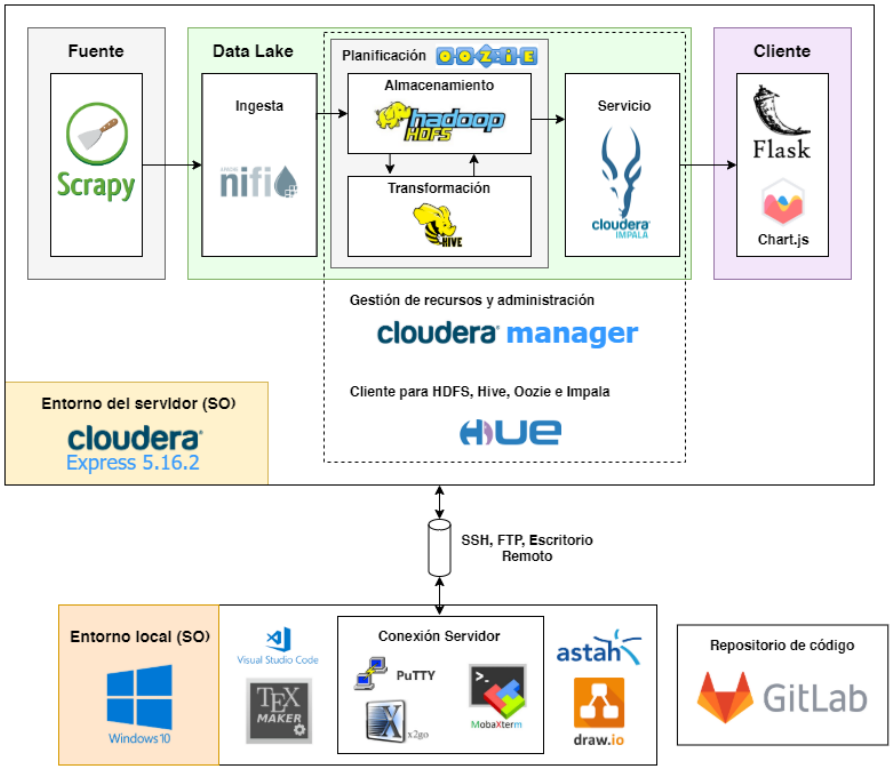


Figura 6.2: *Stack* tecnológico utilizado *(Fuente: elaboración propia)*.

6.2. Implementación

6.2.1. Web scraping

El *web scraping* consiste en obtener información de páginas web de manera automática, mediante un programa. Esta técnica se suele utilizar junto al *web crawling*, que permite indexar una página web. De esta manera, un *web crawler* accederá a diferentes URLs y registrará la estructura de la web, mientras que un *scraper* obtendrá la información de cada página.

6.2.1.1. Scrapy y Scrapyd

Para la obtención de los datos de la NBA fue necesario el uso de esta técnica, para lo que se decidió usar Scrapy. Scrapy es un *framework* de código abierto para la creación

de *web crawlers* y *web scraping*. Está basado en una arquitectura basada en eventos, que incluye flujos de trabajo para la limpieza y extracción eficiente de la información. Además, permite paralelizar tanto las consultas HTTP como el procesamiento de las mismas [25, 29]. Un esquema de la arquitectura de este *framework* se puede apreciar en Figura 6.3.

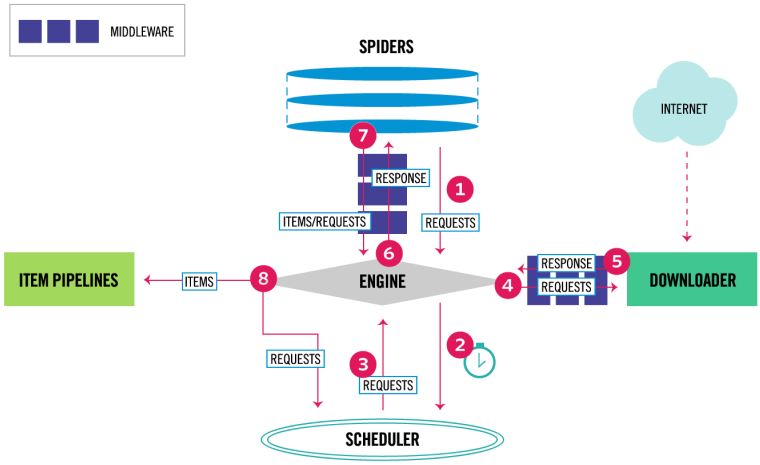


Figura 6.3: Arquitectura de Scrapy *(Fuente: https://scrapy.org)*.

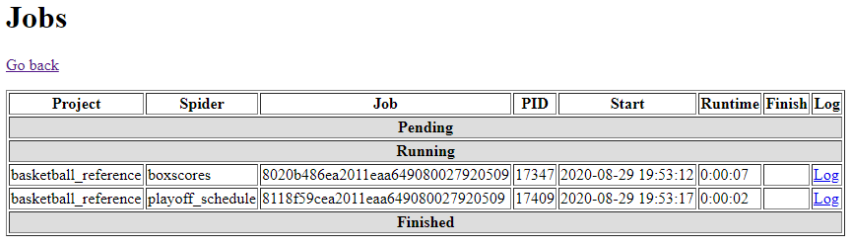
Figura 6.4: Logo de Scrapy ©.

Scrapy permite lanzar procesos mediante una interfaz de comandos CLI, pero para darle una mayor funcionalidad se ha utilizado Scrapyd, que es una aplicación para el despliegue y ejecución de "arañas". Se denomina araña a un programa capaz de extraer información de una página *web* y/o indexarla. Scrapyd aumenta la funcionalidad de Scrapy ofreciendo:

1. Interfaz web para la visualización de los diferentes procesos (tanto en ejecución como histórico), la consulta de los *logs* de los mismos, y los diferentes proyectos alojados. En las Figuras 6.5, 6.6, se puede ver parte de esta interfaz.

2. API HTTP para el lanzamiento, planificación, consulta de procesos en ejecución y *logs* de los procesos, consulta de las diferentes arañas y proyectos (en todas sus versiones).

3. Capacidad de albergar diferentes versiones de una misma araña. De esta manera, se pueden conservar diferentes versiones para una web.

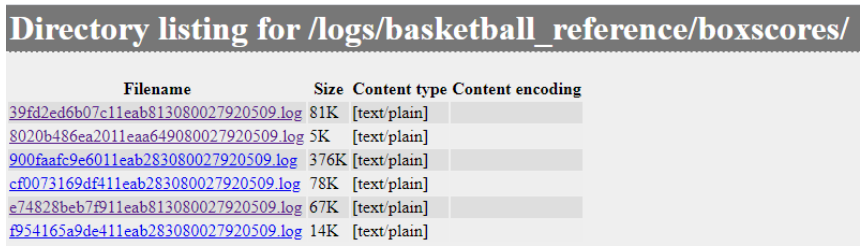


Jobs

Go back

Project	Spider	Job	PID	Start	Runtime	Finish	Log
Pending							
Running							
basketball_reference	boxscores	8020b486ea2011eaa649080027920509	17347	2020-08-29 19:53:12	0:00:07		Log
basketball_reference	playoff_schedule	8118f59cea2011eaa649080027920509	17409	2020-08-29 19:53:17	0:00:02		Log
Finished							

Figura 6.5: Interfaz web de Scrapyd para la monitorización del los trabajos.



Directory listing for /logs/basketball_reference/boxscores/

Filename	Size	Content type	Content encoding
39fd2ed6b07c11eab813080027920509.log	81K	[text/plain]	
8020b486ea2011eaa649080027920509.log	5K	[text/plain]	
900faafc9e6011eab283080027920509.log	376K	[text/plain]	
cf0073169df411eab283080027920509.log	78K	[text/plain]	
e74828beb7f911eab813080027920509.log	67K	[text/plain]	
f954165a9de411eab283080027920509.log	14K	[text/plain]	

Figura 6.6: Listado de *logs* para la araña *"boxscores"*.

6.2.1.2. Arañas

Para este proyecto se han implementado 3 arañas para extraer los datos de Basketball Reference. En el caso de este proyecto, los datos obtenidos por estos programas se descargarán en el directorio `/home/cloudera/scrapyd/data/`.

Partidos y *boxscores* La primera araña extrae, en primer lugar, el listado de partidos (indicado en el Capítulo 5, Figura 5.8), para acceder a la página de *boxscores* de cada partido (ver Figura 5.5). Estos datos se exportan a archivos CSV, uno por partido, y estadísticas avanzadas y básicas. Los archivos descargados se dispondrán de la siguiente manera:

```
data/
└── games/
    └── boxscores/
        ├── advanced/
        │   ├── ...
        │   ├── 2019/
        │   │   ├── 01010DEN.csv
        │   │   ├── 01010LAC.csv
        │   │   └── 01010MIL.csv
        │   └── ...
        └── basic/
            ├── ...
            └── 2019/
                └── 01010DEN.csv
```

```
    ├── 01010LAC.csv
    ├── 01010MIL.csv
    └── ...
```

La araña consta de numerosas opciones para extraer los partidos que sea necesario. Se han introducido intentando aportar la mayor flexibilidad posible a la misma:

- **seasons=2019,2020** : permite seleccionar las temporadas que se desee, separadas por comas y sin espacios.
- **from_season** y **to_season** : para seleccionar un rango de temporadas para las que descargar sus partidos.
- **date**: puede tomar los valores "today" y "season", que permiten seleccionar el día y la temporada actual, respectivamente.
- **months=0,1,3,4** : para seleccionar un número limitado de meses. Enero se corresponderá con el número 0, febrero con el 1, etc.
- **games=201910220TOR,202008300TOR** : obtendrá los datos de los partidos seleccionados. Se indicará el partido utilizando los identificadores de Basketball Reference.
- **only-schedules**: la existencia de este parámetro, indicará que sólo se desea descargar el calendario de partidos. La utilidad reside en que una temporada dispone de menos de 10 páginas de calendario, mientras que el número de partidos es mayor de 1000, con lo que el tiempo utilizado será 100 veces menor.

Playoffs En Basketball Reference, no se indica si un partido es de temporada regular o de *playoff*. Por lo tanto, ha sido necesario desarrollar esta araña para extraer los partidos que sí son *playoffs*.

```
data/
├── games/
│   └── ...
└── players/
    ├── data/
    │   ├── a/
    │   │   ├── abdelal01.json
    │   │   ├── abdulka01.json
    │   │   ├── abdulma.json
    │   │   └── ...
    │   ├── b/
    │   └── ...
    └── list/
        ├── a.csv
        ├── b.csv
        ├── c.csv
        └── ...
```

Los parámetros de esta araña se comportan de manera exactamente igual a la de la araña "`boxscores`". Sin embargo, dispone de menos:

- **`seasons`**.
- **`from_season`** y **`to_season`**.
- **`date`**.

Jugadores Para la extracción de los jugadores se consulta en primer lugar el listado de jugadores (como se puede ver en la Figura 5.8). Posteriormente se obtiene la información individual de cada jugador, que se almacena en un archivo JSON por cada jugador.

```
data/
└── schedule/
    ├── games/
    │   ├── 1950_april.csv
    │   ├── 1950_december.csv
    │   ├── 1950_february.csv
    │   └── ...
    └── playoffs/
        ├── 1950.csv
        ├── 1951.csv
        ├── 1952.csv
        └── ...
```

De manera similar a lo ofrecido en la primera araña, también se cuentan con opciones mediante parámetros:

- **`ids`**`=boldema01,greenje02` : se descargan los datos de los jugadores indicados por los identificadores de Basketball Reference.
- `l=a,b,c` : se descargan los jugadores y el listado de jugadores cuyo identificador comienza por la letra indicada (o lo que es lo mismo, la primera letra del apellido).

6.2.1.3. Despliegue

Para que las arañas anteriormente descritas se ejecuten en el servidor es necesario desplegarlas en el servidor de Scrapy. Para facilitar este proceso, se requiere disponer de un túnel SSH en el puerto 6800 desde la máquina de desarrollo hasta la máquina donde se ejecutarán estos procesos. Convenientemente, Scrapy dispone de un cliente que empaqueta el código y lo envía a través de la API HTTP. Con el comando siguiente se despliega el proyecto en el destino indicado (`cloudera_tfg`, indicando el servidor y el puerto en el fichero de configuración de Scrapy) y la versión con la que se quiere registrar este código (`2.0`). Se puede ver este proceso en la Figura 6.7.

```
scrapyd-deploy cloudera_tfg --version 2.0
```

Una vez desplegadas las arañas, se dispondrá a programar la ejecución de los procesos. Para ello se registra, mediante la utilidad `cron`, un proceso que se ejecutará cada día.

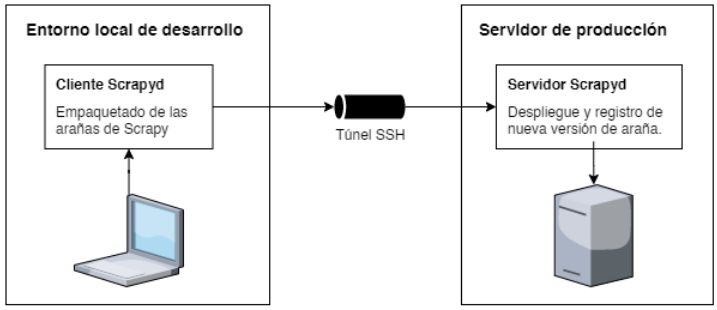


Figura 6.7: Despliegue de Scrapy a un servidor remoto desde un ordenador personal.

```
curl http://localhost:6800/schedule.json -d project=basketball_reference \
     -d spider=boxscores -d date=today

curl http://localhost:6800/schedule.json -d project=basketball_reference \
     -d spider=playoff_schedule -d date=today

curl http://localhost:6800/schedule.json -d project=basketball_reference \
     -d spider=player -d all=true
```

El *script* anterior envía trabajos para la obtención de los nuevos partidos y la actualización de los datos de los jugadores que puedan haber cambiado.

6.2.2. Ingesta

La ingesta se realiza mediante NiFi, que permite establecer un flujo de datos entre el *web scraper* y el sistema de ficheros de Hadoop. De esta manera, los datos extraídos son gestionados de manera segura y fehaciente. En la Figura 6.8, se puede ver el *workflow* actual.

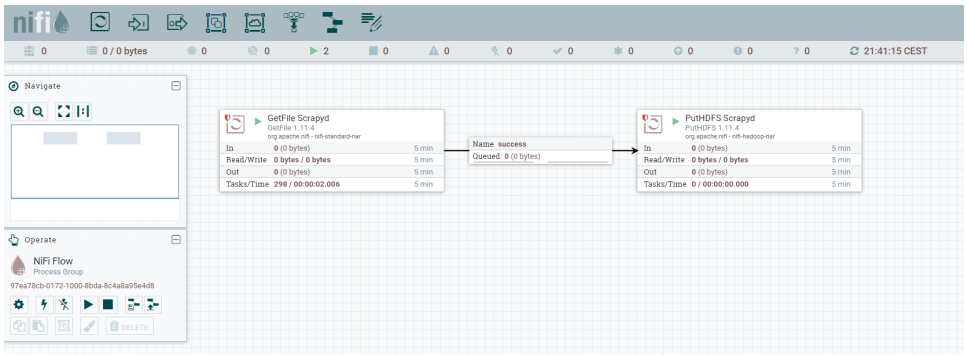


Figura 6.8: Ingesta de los archivos en crudo a HDFS.

En el flujo de trabajo empleado, los archivos se eliminan del sistema de ficheros local

(usado por Scrapy) para transportar los ficheros a HDFS.

Aunque parezca que el uso de un servicio como NiFi para un proceso tan sencillo sea desproporcionado, la naturaleza diversa de los datos y sus fuentes y la posibilidad de extender este *Data Lake* en el futuro ha sido el motivo para la instalación y configuración de esta herramienta.

6.2.3. Tratamiento de los datos

Las operaciones realizadas sobre los datos se explican en el Apartado 5.4. En este apartado se explicará la tecnología que implementa y realiza estos cálculos y transformaciones. En este apartado se enumeran los *scripts* que se utilizan para transformar los datos y crear las vistas intermedias.

Los datos se almacenan en HDFS, y se disponen en tres capas: *raw*, *silver* y *gold*. Estas capas se almacenan en tres carpetas dentro de HDFS como se ve a continuación:

```
/user/cloudera/br/
    ├── raw
    ├── silver
    └── gold
```

El manejo de estos datos se hace mediante *scripts* en Hive, que se almacenan también en HDFS. Se almacenan dentro de la carpeta `/user/cloudera/br_scripts`, de la siguiente manera:

```
/user/cloudera/br_scripts/
├── raw
│   └── create
│       ├── boxscores_advanced.sql
│       ├── boxscores_basic.sql
│       ├── player_data.sql
│       ├── player_stats.sql
│       ├── schedule_games.sql
│       └── schedule_playoffs.sql
├── silver
│   ├── create
│   │   ├── advanced_player_stats.sql
│   │   ├── boxscores_basic.sql
│   │   ├── boxscores_basic_team_totals.sql
│   │   └── schedule.sql
│   └── insert_into
│       └── [IDEM create silver]
└── gold
    └── create
        ├── advanced_player_stats.sql
        └── boxscores_player.sql
```

```
    boxscores_team.sql
    player_data.sql
    player_list.sql
    schedule.sql
  insert_into
    [IDEM create gold]
  daily_recalculation
    daily_recalculation.sql
```

Los *scripts* listados anteriormente se encuentran en el Apéndice B. Las capas *silver* y *gold* cuentan con *scripts* de creación y de población de tablas. Se ha indicado con [IDEM create silver/gold] que los nombres de los *scripts* de creación de las tablas son idénticos a los de población de las mismas. Dada la gran cantidad de ficheros SQL que se deben manejar, se utiliza Oozie para coordinar los trabajos de recomputación y creación de las tablas.

6.2.4. Coordinación de trabajos mediante Oozie

Los flujos de trabajo *(workflows)* de Oozie nos permiten orquestar y programar las tareas para re-calcular todas las tablas. Estos *workflows* se encuentran en la interfaz de Hue, en el apartado de "Documentos", como se observa en la Figura 6.9. Para ello, se han creado los siguientes 5 *workflows*:

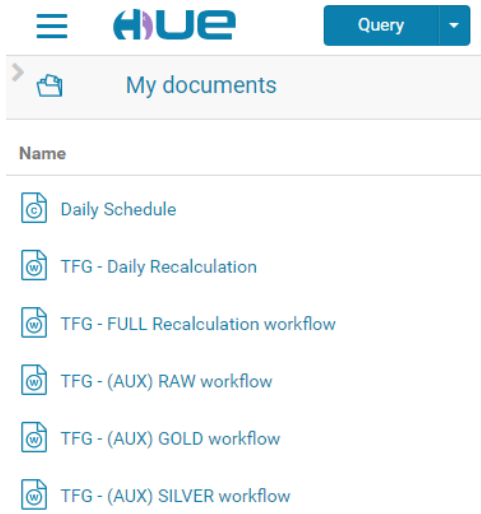


Figura 6.9: Listado de *workflows* en Hue.

- **TFG - Daily Recalculation**: trabajo que se ejecuta cada día. Ejecuta el *script* `daily_recalculation.sql`, que permite actualizar las últimas temporadas y ejecutarse de manera mucho más rápida y haciendo uso de una menor cantidad de recursos.
- **TFG - FULL Recalculation workflow**: para recalcular todas las tablas. En la Figura 6.10 se puede ver una captura de pantalla del *workflow* visualizado mediante

Hue. Este flujo de trabajo ejecuta los *sub-workflows* auxiliares (los que contienen el prefijo "(AUX)" en el nombre).

- **TFG - (AUX) RAW workflow**: ejecuta los comandos de creación de las tablas de la capa *raw*.
- **TFG - (AUX) SILVER workflow**: coordina los *scripts* de generación e inserción de tablas en la capa *silver*.
- **TFG - (AUX) GOLD workflow**: coordina los *scripts* de generación e inserción de tablas en la capa *gold*.

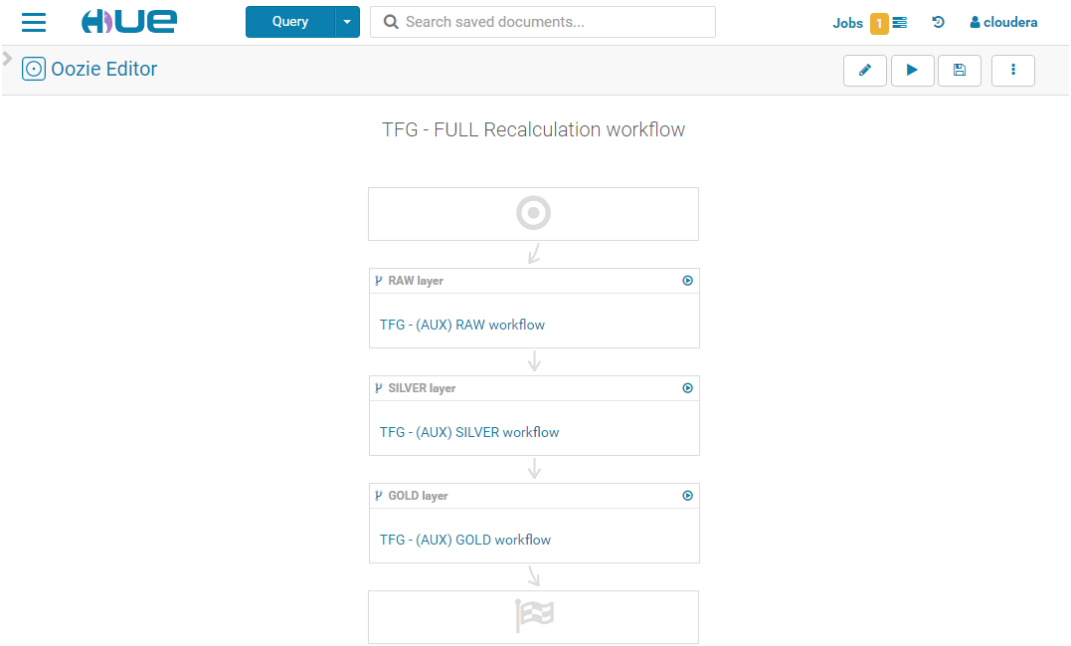


Figura 6.10: Flujo de datos de re-computación completa.

6.2.5. Interfaz

Para facilitar la visualización y acceso de los datos, se ha implementado una interfaz, que consiste en un portal *web*. Este portal ha sido implementado en Flask, un *framework* para desarrollo *web* implementado en Python. Usando la librería Impyla de Python, se accede a los datos del servidor. Para la parte cliente de la interfaz se ha hecho uso de las librerías DataTables[1] y Chart.js[2]; la primera para añadir funcionalidad a las tablas y la segunda para

[1] `https://datatables.net/`
[2] `https://www.chartjs.org/`

realizar gráficos en el navegador.

Para acceder al portal existen dos opciones. En la primera, deberemos, tras habernos conectado a la máquina, introducir la URL `localhost:8008` en un navegador. La segunda opción consiste en acceder a la URL `http://tfg-basket.xyz`. A continuación se describen las diferentes partes de la misma.

6.2.5.1. Inicio

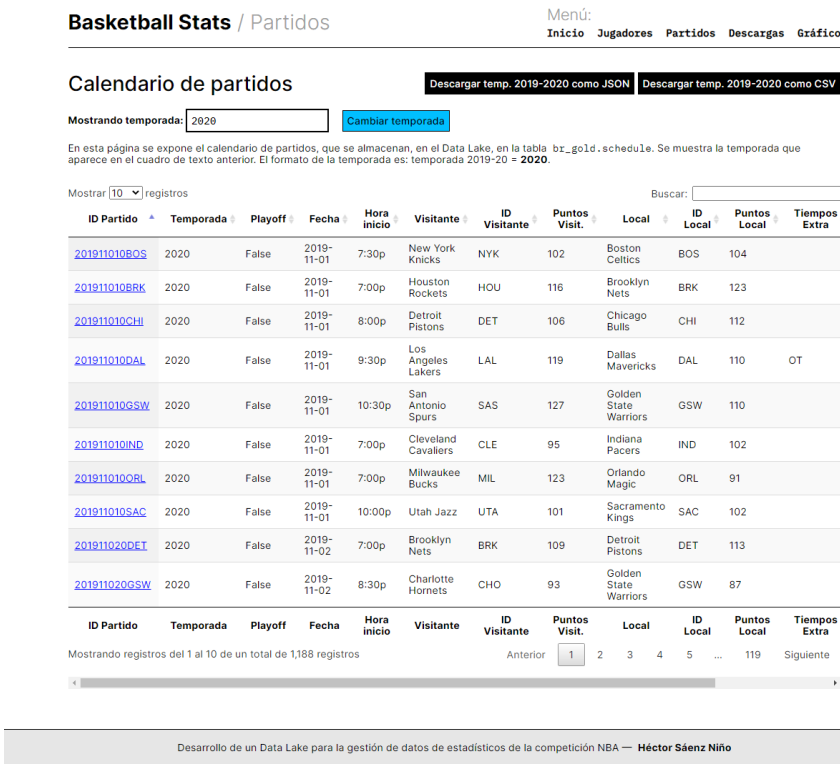
Es la página de inicio, desde donde se presentan enlaces al resto de componentes. En la parte superior derecha se muestra un menú de navegación. En el cuerpo de esta pestaña se encuentran enlaces a "Jugadores" (`/players`), "Partidos" (`/games`), "Descargas" (`/downloads`) y "Gráfico peso-altura" (`/graphs/player-size`).

Figura 6.11: Página de inicio del portal *web* para acceder a los datos.

6.2.5.2. Partidos

Haciendo *click* en el enlace "Partidos" en el menú de navegación accedemos a esta página, que se puede ver en la Figura 6.12. Se muestra una tabla con todos los partidos históricos de la NBA, almacenados en la tabla `br_gold.schedule`. En esta página se puede:

- Realizar búsquedas por cualquier columna.
- Ordenar los datos según la columna deseada (fecha, equipo, puntuación, ...).
- Acceder a la página individual de los *boxscores*, pinchando en el identificador del partido deseado.
- Descargar la lista de partidos históricos en formato CSV o JSON, haciendo *click* en los botones que rezan "Descargar como CSV / JSON".



Basketball Stats / Partidos

Menú: Inicio Jugadores Partidos Descargas Gráfico

Calendario de partidos

Descargar temp. 2019-2020 como JSON | Descargar temp. 2019-2020 como CSV

Mostrando temporada: 2020 Cambiar temporada

En esta página se expone el calendario de partidos, que se almacenan, en el Data Lake, en la tabla `br_gold.schedule`. Se muestra la temporada que aparece en el cuadro de texto anterior. El formato de la temporada es: temporada 2019-20 = **2020**.

Mostrar 10 registros — Buscar:

ID Partido	Temporada	Playoff	Fecha	Hora inicio	Visitante	ID Visitante	Puntos Visit.	Local	ID Local	Puntos Local	Tiempos Extra
201911010BOS	2020	False	2019-11-01	7:30p	New York Knicks	NYK	102	Boston Celtics	BOS	104	
201911010BRK	2020	False	2019-11-01	7:00p	Houston Rockets	HOU	116	Brooklyn Nets	BRK	123	
201911010CHI	2020	False	2019-11-01	8:00p	Detroit Pistons	DET	106	Chicago Bulls	CHI	112	
201911010DAL	2020	False	2019-11-01	9:30p	Los Angeles Lakers	LAL	119	Dallas Mavericks	DAL	110	OT
201911010GSW	2020	False	2019-11-01	10:30p	San Antonio Spurs	SAS	127	Golden State Warriors	GSW	110	
201911010IND	2020	False	2019-11-01	7:00p	Cleveland Cavaliers	CLE	95	Indiana Pacers	IND	102	
201911010ORL	2020	False	2019-11-01	7:00p	Milwaukee Bucks	MIL	123	Orlando Magic	ORL	91	
201911010SAC	2020	False	2019-11-01	10:00p	Utah Jazz	UTA	101	Sacramento Kings	SAC	102	
201911020DET	2020	False	2019-11-02	7:00p	Brooklyn Nets	BRK	109	Detroit Pistons	DET	113	
201911020GSW	2020	False	2019-11-02	8:30p	Charlotte Hornets	CHO	93	Golden State Warriors	GSW	87	

Mostrando registros del 1 al 10 de un total de 1,188 registros — Anterior 1 2 3 4 5 ... 119 Siguiente

Desarrollo de un Data Lake para la gestión de datos de estadísticos de la competición NBA — **Héctor Sáenz Niño**

Figura 6.12: Vista del calendario de partidos histórico de la NBA.

6.2.5.3. Boxscores

Se puede acceder a los *boxscores* mediante los enlaces que se muestran en el calendario de partidos (ver apartado anterior). Se muestran dos tablas, una con los *boxscores* separados por cada jugador y otro con los totales de equipo para ese partido. Estos datos se almacenan en las tablas `br_gold.boxscores_player` y `br_gold.boxscores_team`, respectivamente. En la parte superior derecha, se encuentran dos botones negros que permiten descargar los datos de ese partido en CSV y JSON, tanto para las estadísticas por jugador como para los totales de equipo. En la Figura 6.13 se puede ver las *boxscores* para un encuentro.

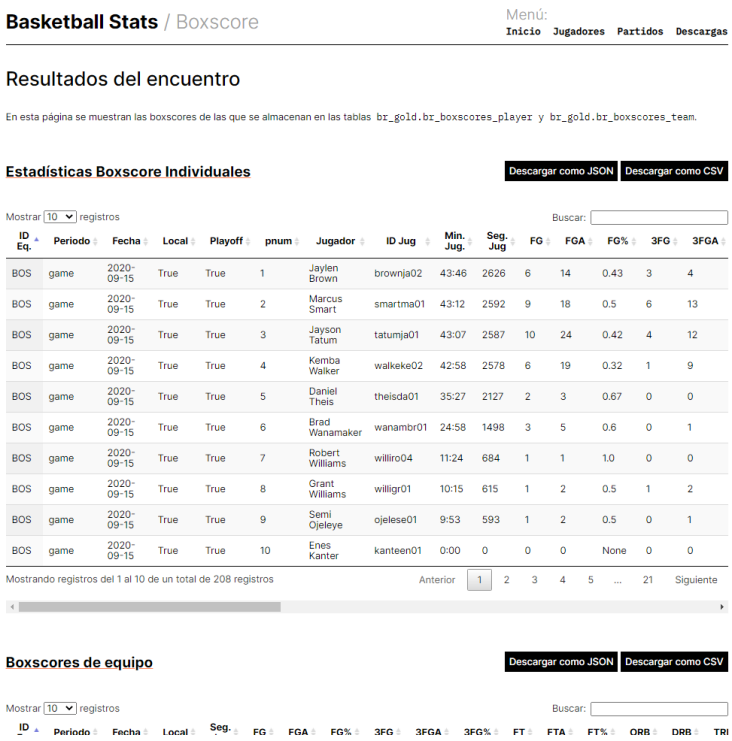


Figura 6.13: *Boxscores* para el encuentro del 9 de septiembre de 2020 entre Miami Heat y Boston Celtics.

6.2.5.4. Listado de jugadores

Se muestra una tabla con un listado de los jugadores que han participado en la competición. Se muestran los datos almacenados en la tabla `br_gold.player_list`. Es posible realizar búsquedas y ordenar los jugadores por atributos como la altura o la fecha de nacimiento. Además, se dispone de dos botones para descargar los datos de la tabla en formato CSV o JSON. Por último, se incluyen enlaces a las estadísticas avanzadas de cada jugador. La Figura 6.14 muestra el listado de jugadores en la interfaz.

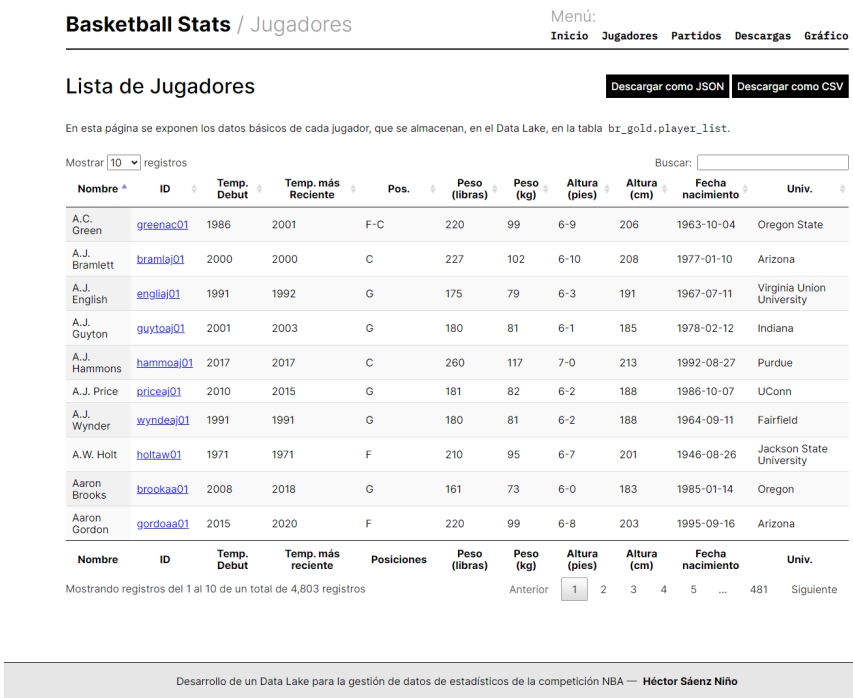


Figura 6.14: Listado de jugadores.

6.2.5.5. Estadísticas avanzadas de jugadores

Tras acceder haciendo *click* en un enlace de la tabla del apartado anterior, se muestran los datos almacenados en la tabla `br_gold.advanced_player_stats`. Se dispone de dos botones para descargar las estadísticas avanzadas de ese jugador en formato CSV o JSON. La Figura 6.15 muestra un ejemplo de estadísticas avanzadas.

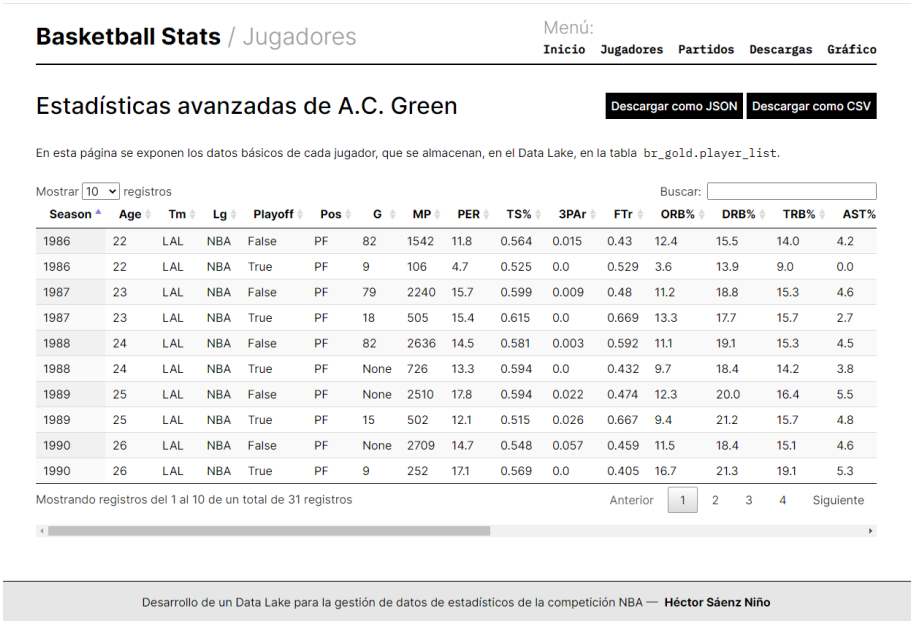


Figura 6.15: Estadísticas avanzadas del jugador A. C. Green.

6.2.5.6. Descargas

Pestaña para descargar los datos en CSV o JSON. Se puede ver en la Figura 6.16. Se pueden descargar datos de las siguientes tablas:

- `br_gold.schedule`: para obtener el calendario de partidos entre dos fechas.
- `br_gold.boxscores_player`: para obtener los datos de las *boxscores* a nivel de jugador entre dos fechas.
- `br_gold.boxscores_team`: ídem anterior, pero a nivel de equipo.
- `br_gold.player_list`: descarga del listado de jugadores completo.

6.2.5.7. Gráfico peso-altura

En esta pestaña, accesible mediante el enlace presente en la página de inicio o por la URL `/graphs/player-size`, se presenta un gráfico de dispersión de la altura y peso de los jugadores. Al pasar el cursor sobre un punto, un *tooltip* especifica el nombre, peso, altura e identificador del jugador en cuestión. En la Figura 6.17 se muestra esta gráfica.

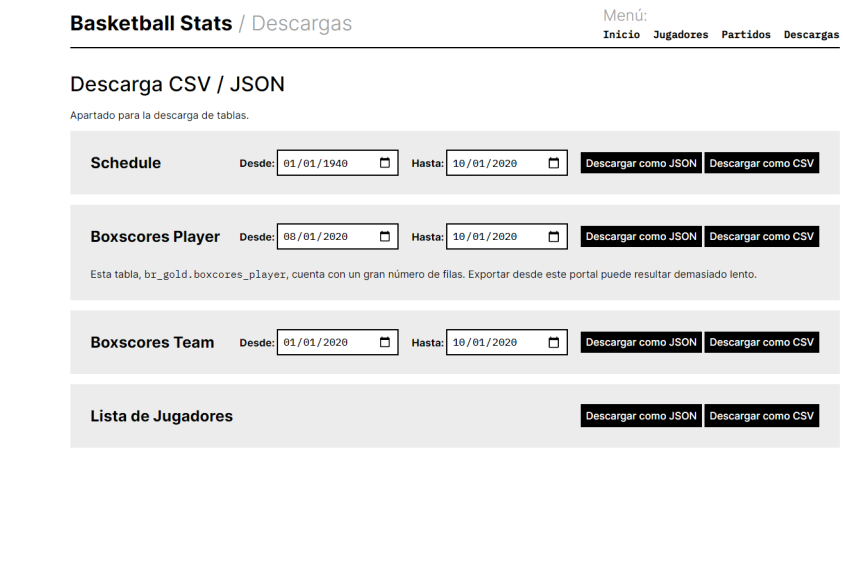


Figura 6.16: Pestaña de descargas.

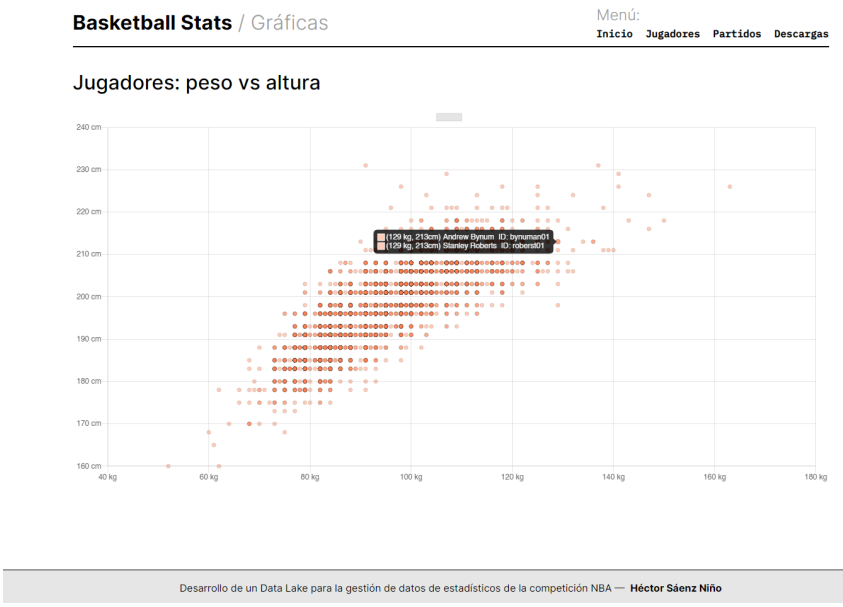


Figura 6.17: Gráfico de dispersión peso-altura.

Capítulo 7

Manual de Instalación y Configuración

En este anexo se indican los paso para instalar y poner en funcionamiento el entorno de Cloudera y las aplicaciones desarrolladas en el presente trabajo, a partir de una distribución Cloudera Express 5.16.2.

7.1. Conexión a la máquina

En primer lugar se procederá a conectarse a la máquina. Es posible conectarse por escritorio remoto. Sin embargo, dado que los servicios utilizados consisten en portales *web*, resulta más conveniente tunelar los puertos indicados en la Tabla 7.1 por SSH.

Puerto	**Aplicación**	**Descripción**
6800	Scrapyd	Servicio para la extracción de datos.
7180	Cloudera Manager	Para la administración de los servicios.
8008	Interfaz *web*	Interfaz desarrollada para la visualización de los datos.
8080	Apache NiFi	Ingesta y transporte de datos.
8088	YARN	Consulta de trabajos MapReduce en ejecución.
8888	Hue	Interfaz para ejecutar consultas y programar *worflows* de Oozie.

Tabla 7.1: Puertos de las interfaces utilizadas en el proyecto.

7.1.1. PuTTY

Como primera opción para conectarse por tunelado de puertos SSH se propone PuTTY, que es una herramienta introducida en el Capítulo 4. En este apartado se detalla la configuración de esta herramienta.

Se comienza iniciando PuTTY. Tras iniciarlo, se presenta la ventana indicada en las Figuras 7.1 y 7.2. Los números de las imágenes se corresponden con las explicaciones de la siguiente lista:

1. Se introduce el USUARIO@DIRECCIÓN de la máquina para SSH.
2. Se indica el puerto asignado a SSH.
3. Se introduce un nombre para guardar la sesión.
4. Para guardar la configuración.
5. Ampliamos el menú haciendo *click* en SSH.
6. Ídem 5.
7. Se presiona la opción *Tunnels*.
8. Introducimos los puertos que se reenviarán. Añadir los puertos de la Tabla 7.1 (6800, 7180, 8008, 8080, 8088 y 8888).
9. Se introduce el puerto, comenzando por "localhost:".
10. Se añade los puertos que se reenviarán pulsando este botón.
11. Guardamos la sesión y pinchamos en *Open*.

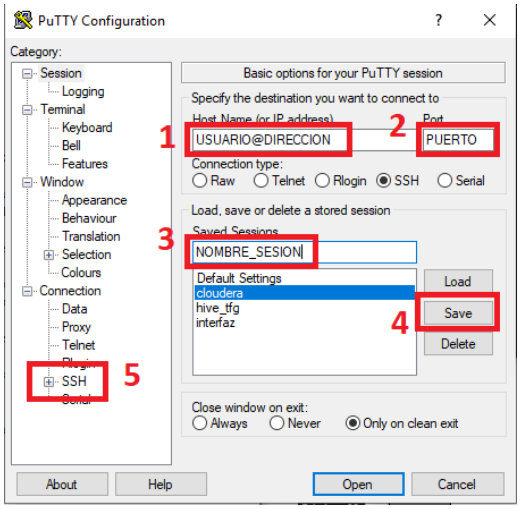


Figura 7.1: Configuración PuTTY.

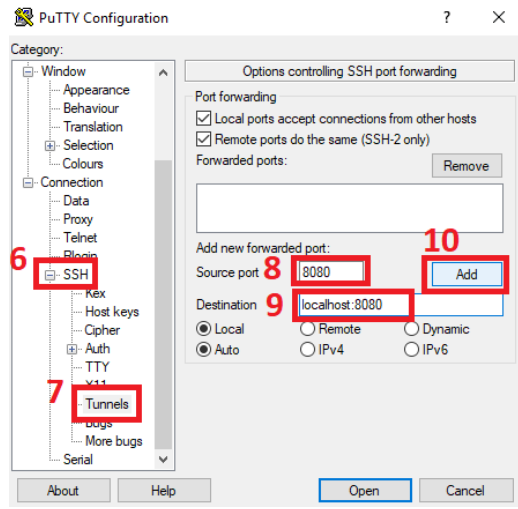


Figura 7.2: Configuración PuTTY: túneles.

7.2. Configuración Cloudera Manager

En este apartado se configurará, desde Cloudera Manager, configuración sobre Oozie, Hive, YARN e Impala. Se comienza iniciando sesión en Cloudera Manager, para lo que se introduce la URL `http://localhost:7180`. Aparece una ventana que pide usuario y contraseña (ver Figura 7.3). Los valores por defecto son "cloudera" y "cloudera".

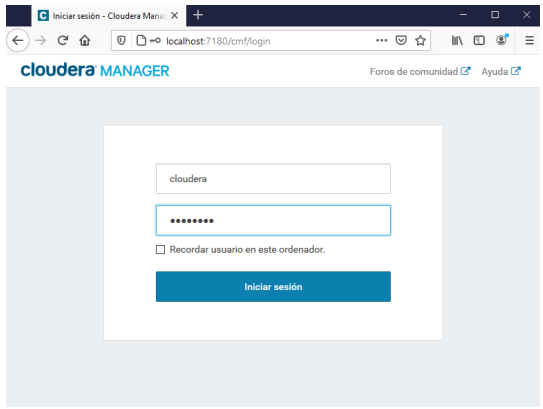


Figura 7.3: Inicio de sesión Cloudera.

Tras iniciar sesión, se presentan el menú de la Figura 7.4 numerosas tecnologías.

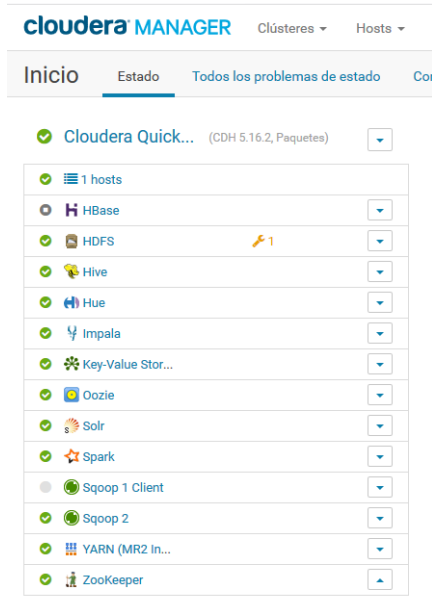


Figura 7.4: Menú con las tecnologías administradas por Cloudera Manager.

7.2.1. YARN

Se hace *click* en YARN (ver Figura 7.4). Posteriormente se accede a la configuración de YARN (Figura 7.5).

Figura 7.5: Configuración de YARN.

Se aumentarán los siguientes parámetros (como se puede ver, de manera parcial, en la Figura 7.6):

- **ApplicationMaster Memory**: aumento a 2 GiB.
- **ApplicationMaster Java Maximum Heap Size**: aumento a 1 GiB.
- **Map Task Memory**: aumento a 2 GiB.
- **Map Task CPU Virtual Cores**: aumento a 2 núcleos.
- **Reduce Task Memory**: aumento a 4 GiB.
- **Reduce Task CPU Virtual Cores**: aumento a 2 núcleos.
- **Map Task Maximum Heap Size**: aumento a 1'5 GiB.
- **Reduce Task Maximum Heap Size**: aumento 3 GiB.
- **Reduce Task CPU Virtual Cores**: aumento a 2 núcleos.
- **Client Java Heap Size in Bytes**: aumento a 756 MiB.

7.2.2. Configuración de *logs* de YARN, Hive, Impala, Oozie y Zookeeper

En este apartado se indica cómo reducir el tamaño y número de *logs* de YARN, Hive, Impala, Oozie y Zookeeper, en un esfuerzo por evitar el crecimiento excesivo de los archivos *log*. Accedemos a la configuración de los servicios anteriores.

Para cada servicio, se reduce el número de *logs* de 10 a 3 y el tamaño de los mismos de 200Mb a 20Mb. El motivo de este cambio es prevenir que la memoria se llene a causa de los ficheros *log*. Se puede ver un ejemplo en YARN para estos ajustes en las Figuras 7.7 y 7.8. Se repite el proceso para YARN, Hive, Impaña, Oozie y Zookeeper.

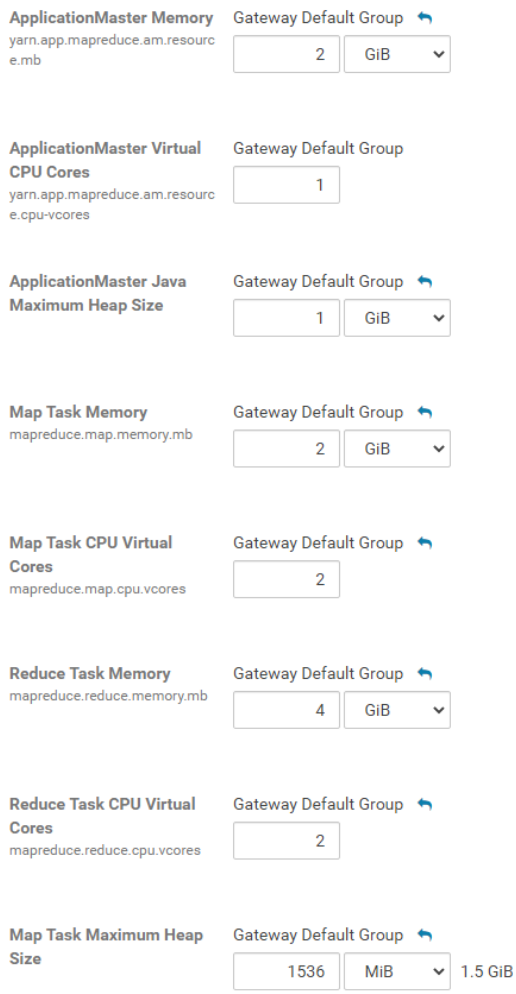


Figura 7.6: Configuración de YARN: procesos MapReduce.

Figura 7.7: Config. de *JobHistory Log*.

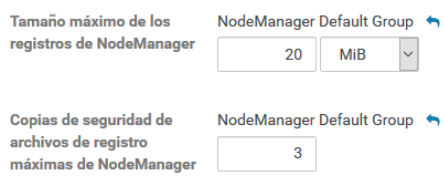


Figura 7.8: Config. de *NodeManager Log*.

7.2.3. Configuración de HDFS

Se desactivan los permisos de escritura, para permitir escribir en las tablas internas. Ver en la Figura 7.9.

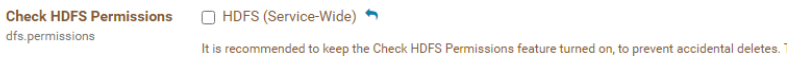


Figura 7.9: Configuración de HDFS para permitir escrituras.

7.3. Instalación de Apache NiFi

La instalación de Apache NiFi resulta la más engorrosa, ya que existen incompatibilidades con la versión de Java y es necesario instalar otra versión de Java.

En primer lugar se descarga Apache NiFi de la siguiente página: `https://archive.apache.org/dist/nifi/`. En el enlace anterior, se encuentran todas las versiones de NiFi. En el caso de este trabajo, se ha utilizado la versión 1.11.4.

```
# Directorio de descargas.
cd /home/cloudera/Downloads
# Descarga de NiFi
wget https://archive.apache.org/dist/nifi/1.11.4/nifi-1.11.4-bin.tar.gz
# Se extrae en el directorio /home/cloudera/nifi-1.11.4
tar xvzf nifi-1.11.4-bin.tar.gz -C /home/cloudera


# JAVA installation
# Java command to use when running NiFi
sudo update yum
# Install OpenJDK:
sudo yum install java-1.8.0-openjdk

# Se ejecuta el siguiente comando y se elige OpenJDK como opción por defecto.
sudo alternatives --config java
```

Posteriormente, se debe modificar la siguiente línea del fichero `/home/cloudera/nifi-1.11.4/conf/bootstrap.conf`:

```
java=/usr/lib/jvm/jre-1.8.0-openjdk.x86_64/bin/java
```

De esta manera, se especifica la versión de Java que se usará para ejecutar Apache NiFi. Para ejecutar Apache NiFi, se ejecuta el siguiente comando:

```
# Start NiFi server
sudo /home/cloudera/nifi-1.11.4/bin/nifi.sh start
```

Tras iniciar el servicio, abrimos un navegador, donde se realizan los pasos descritos a continuación (ver Figura 7.10):

1. Accedemos a la URL `http://localhost:8080/nifi/`.
2. Hacemos *click* derecho sobre el *canvas*.
3. Seleccionamos *"Upload Template"*. Subimos el archivo ``TFG_-_Data_Lake_NBA.xml'', que se encuentra en la carpeta `nifi` del Repositorio de Código (ver Capítulo 9).
4. Presionamos el botón "Play" del flujo de trabajo, para que comience a trabajar.

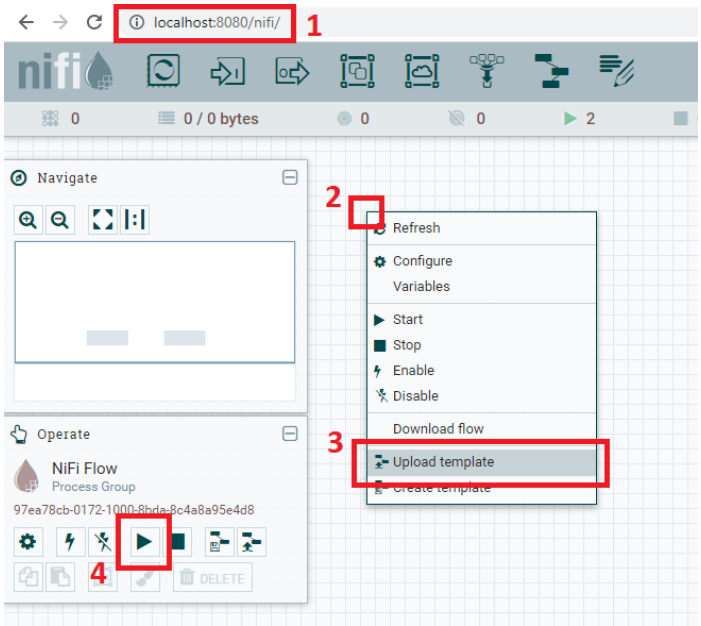


Figura 7.10: Portal de NiFi en el navegador.

7.4. Instalación Python

Para la instalación de Scrapy y Flask, se necesitará instalar Python y las siguientes dependencias:

```
sudo yum install centos-release-scl
sudo yum-config-manager --enable rhel-server-rhscl-7-rpms

sudo yum install rh-python35

# scl enable nos permite activar Python 3.5. Este comando se deberá usar
# cada vez que se quiera utilizar esta versión de Python en vez de 2.6.
scl enable rh-python35 bash

python -m ensurepip --upgrade
python -m pip install scrapy
python -m pip install scrapyd
python -m pip install flask
python -m pip install impyla
python -m pip install flask_cors
```

Ejecutados estos pasos ya se cuenta con la versión deseada de Python y utilizada por el servicio de extracción de datos y la interfaz.

7.5. Scrapy

Scrapy se ha instalado en la sección anterior. En esta sección se procederá a su configuración. En primer lugar, se crea una carpeta para alojar los datos de Scrapy:

```
mkdir /home/cloudera/scrapyd
```

Para comenzar el servidor de Scrapyd, se ejecutan los siguientes comandos:

```
source scl_source enable rh-python35

cd /home/cloudera/scrapyd
nohup scrapyd &> /dev/null &
```

Tras haber lanzado el proceso anterior, ahora se deberá empaquetar y enviar el código a Scrapyd. Para ello, se accede a la carpeta `scrapy/basketball-reference` del Repositorio de Código (ver Capítulo 9). Desde ahí, se ejecuta el siguiente comando:

```
# Ejecutado desde 'scrapy/basketball_reference'
scrapyd-deploy cloudera_tfg --version 3.0
```

Este comando puede ser ejecutado tanto desde el servidor Cloudera como desde el PC desde donde se programa. Sin embargo, el servicio Scrapyd debe estar activo y se debe tener acceso al puerto 6800 de la máquina Cloudera.

7.6. Flask

Se comienza moviendo la carpeta `flask` del Repositorio de Código (ver Capítulo 9) al directorio `/home/cloudera` de la máquina virtual. Terminado este paso, y puesto que se han instalado ya todas las dependencias de Flask en la Sección 7.4, sólo se deben ejecutar los siguientes comandos para comenzar el servidor:

```
source scl_source enable rh-python35

cd /home/cloudera/flask
python app.py &> /dev/null &
```

7.7. Cron

CRON es un servicio para la ejecución de *scripts* programados. Se utiliza para lanzar a Scrapyd las órdenes de descarga diarias y para realizar recomputaciones en Hive, de manera redundante a Oozie. También se introduce una directiva que se ejecutará cada vez que se reinicie la máquina, para evitar tener que volver a iniciar todos los servicios.

Se comienza colocando la carpeta `scripts` del Repositorio de Código (ver Capítulo 9) en el directorio `/home/cloudera`. Una vez colocados en este directorio, se ejecuta la orden `crontab -e`. Se abrirá un archivo, en el que se debe copiar el siguiente contenido:

```
11 00 * * * /home/cloudera/scripts/daily

47 00 * * * hive -f /home/cloudera/scripts/daily_recalculation.sql
41 09 * * * hive -f /home/cloudera/scripts/daily_recalculation.sql

@reboot /home/cloudera/scripts/starttfgservices
```

La función de la primera línea se encarga de iniciar, cada noche, los *web scrapers*. Las siguientes dos líneas se tratan de recomputaciones de la capa *silver* y *gold*, para calcular los nuevos datos recién descargados. Se repite dos veces por día pues no afecta negativamente al rendimiento y garantiza que los datos están más actualizados. La tercera línea se ejecuta en caso de reinicio del sistema, y permite reiniciar los servicios Scrapyd, Flask y Apache NiFi. El resto de servicios gestionados por Cloudera Manager no necesita ser reiniciado.

Capítulo 8

Manual de usuario

8.1. Scrapyd

Para acceder a la interfaz de Scrapy[1] se accede a la URL `http://localhost:6800`. Desde ahí, se puede ver la siguiente pantalla: Figura 8.1, donde se encuentran las siguientes opciones:

1. *Jobs* (ver Figura 8.2): listado de los trabajos ejecutados. Se especifica el archivo *log* de los trabajos que están en ejecución o han terminado. En esta tabla se especifica el identificador de la tarea, que puede ser útil para cancelar el trabajo (ver la documentación de Scrapyd).

2. *Logs*: se dividen los logs de los trabajos por proyecto y araña, como se puede ver en las Figuras 8.3, 8.4 y 8.5.

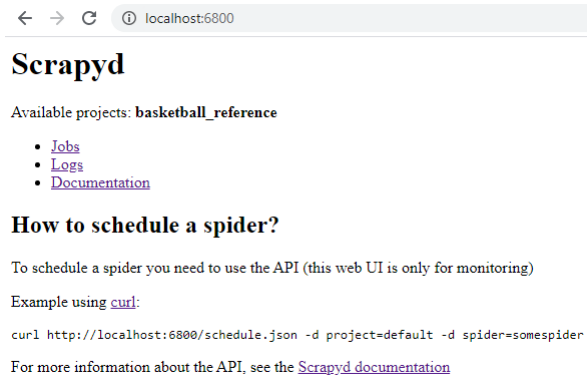
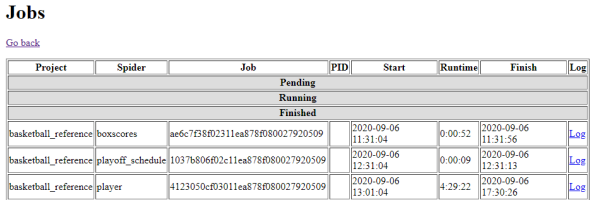


Figura 8.1: Inicio de Scrapyd.

[1]Documentación de Scrapy: `https://docs.scrapy.org/en/latest/`. Documentación de Scrapyd: `https://scrapyd.readthedocs.io/en/stable/`

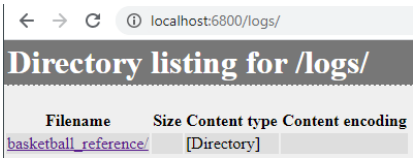


Jobs

Go back

Project	Spider	Job	PID	Start	Runtime	Finish	Log
Pending							
Running							
Finished							
basketball_reference	boxscores	ae6c7f38f02311ea878f080027920509		2020-09-06 11:31:04	0:00:52	2020-09-06 11:31:56	Log
basketball_reference	playoff_schedule	1037b806f02c11ea878f080027920509		2020-09-06 12:31:04	0:00:09	2020-09-06 12:31:13	Log
basketball_reference	player	4123050cf03011ea878f080027920509		2020-09-06 13:01:04	4:29:22	2020-09-06 17:30:26	Log

Figura 8.2: Tareas de Scrapyd.

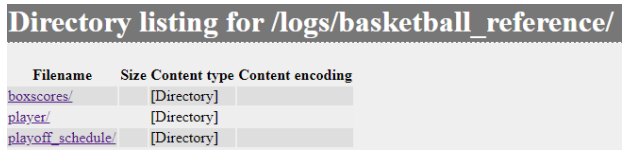


localhost:6800/logs/

Directory listing for /logs/

Filename	Size	Content type	Content encoding
basketball_reference/		[Directory]	

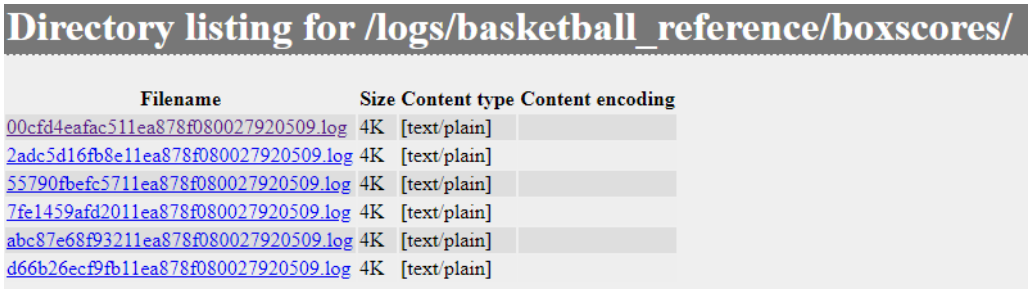
Figura 8.3: Logs de Scrapyd: directorio de proyectos.



Directory listing for /logs/basketball_reference/

Filename	Size	Content type	Content encoding
boxscores/		[Directory]	
player/		[Directory]	
playoff_schedule/		[Directory]	

Figura 8.4: Logs de Scrapyd: directorio de arañas.



Directory listing for /logs/basketball_reference/boxscores/

Filename	Size	Content type	Content encoding
00cfd4eafac511ea878f080027920509.log	4K	[text/plain]	
2adc5d16fb8e11ea878f080027920509.log	4K	[text/plain]	
55790fbefc5711ea878f080027920509.log	4K	[text/plain]	
7fe1459afd2011ea878f080027920509.log	4K	[text/plain]	
abc87e68f93211ea878f080027920509.log	4K	[text/plain]	
d66b26ecf9fb11ea878f080027920509.log	4K	[text/plain]	

Figura 8.5: Logs de Scrapyd: directorio de logs de una araña.

8.2. Hue

Para la consulta de datos en la plataforma, la interfaz web (explicada en la Sección 8.3) resulta conveniente. Sin embargo, en caso de querer realizar consultas más complejas, o que involucren operaciones como “joins” o “counts”, la interfaz de Hue para ejecutar consultas Impala resulta la más indicada.

La interfaz de Hue se encuentra en el puerto 8888. Se accede mediante el navegador, en la URL `http://localhost:8888`. La contraseña y usuario por defecto es “cloudera” “cloudera”. No es necesario reforzar la seguridad en este punto ya que se necesita acceder de manera segura.

En la Figura 8.6 se puede ver una consulta sobre Impala y sus resultados. En el recuadro rojo se pueden ver diferentes iconos, que permiten incluso realizar gráficas para datos sencillos (ver Figura 8.7).

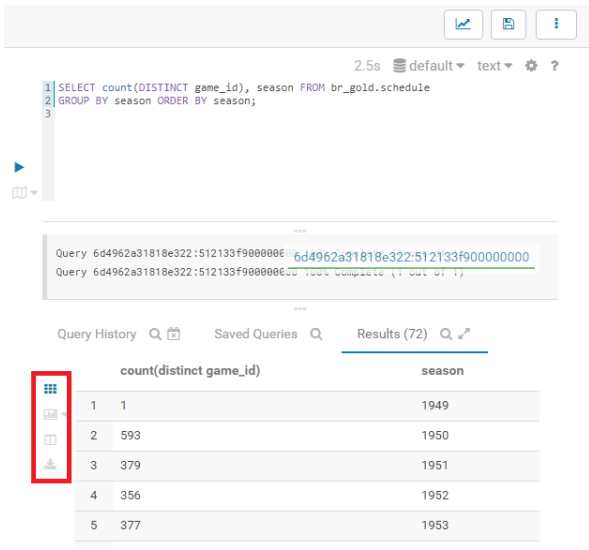


Figura 8.6: Consulta sobre Apache Impala desde la interfaz web de Hue.

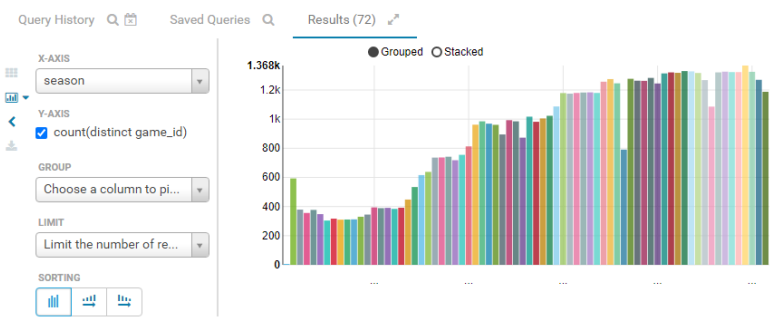


Figura 8.7: Gráfico en Hue a partir de datos obtenidos mediante una consulta a Apache Impala.

8.3. Interfaz *web*

La interfaz de este trabajo ha sido desplegada en el dominio `http://tfg-basket.xyz`, aunque también se pueda acceder por el puerto 8008 de la máquina Cloudera, donde se encuentra desplegado.

8.3.1. Inicio

Se muestra tras acceder a la interfaz. Se puede ver en la Figura 8.8. En el cuerpo de la misma se puede ver una pequeña explicación de los diferentes apartados de la *web*. En rojo se indica el menú de navegación.

Figura 8.8: Página de inicio.

8.3.2. Jugadores

Pestaña con la información básica de los jugadores, almacenadas en tabla `br_gold.player_list`. En la Figura 8.9 se puede ver esta pestaña, que dispone de:

1. Descargar como JSON o CSV los datos de todos los jugadores.
2. Enlaces a las estadísticas avanzadas de los jugadores.
3. Pestañas de navegación en la tabla.
4. Apartado de búsqueda: permite realizar búsquedas por cualquier campo.

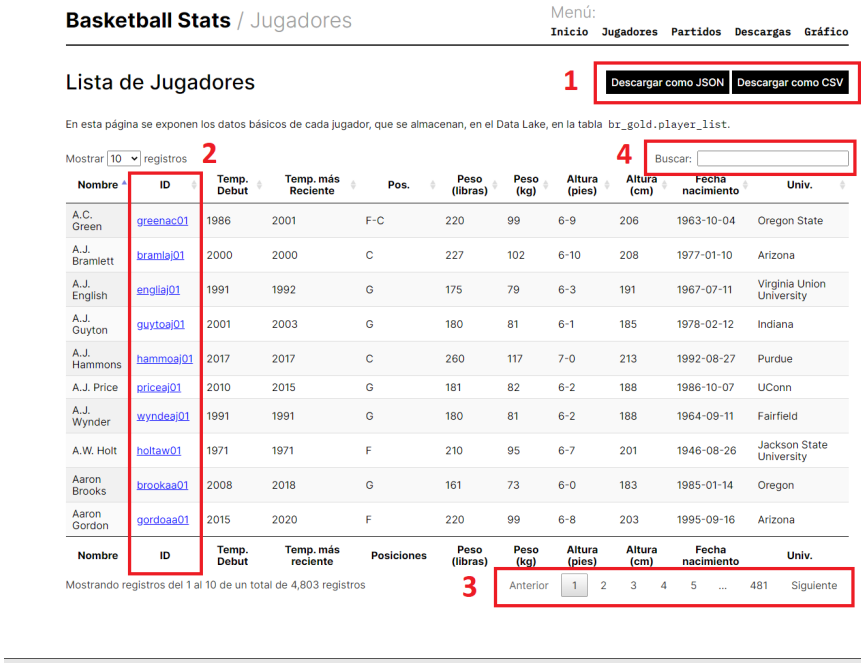


Figura 8.9: Listado de jugadores.

8.3.2.1. Estadísticas avanzadas

Pestaña con la información avanzada de un jugador. Se accederá a esta pestaña a partir de la de jugadores (ver Apartado 8.3.2), y se mostrarán los datos almacenados en tabla `br_gold.advanced_player_stats` para el jugador seleccionado. En la Figura 8.10 se puede ver esta pestaña.

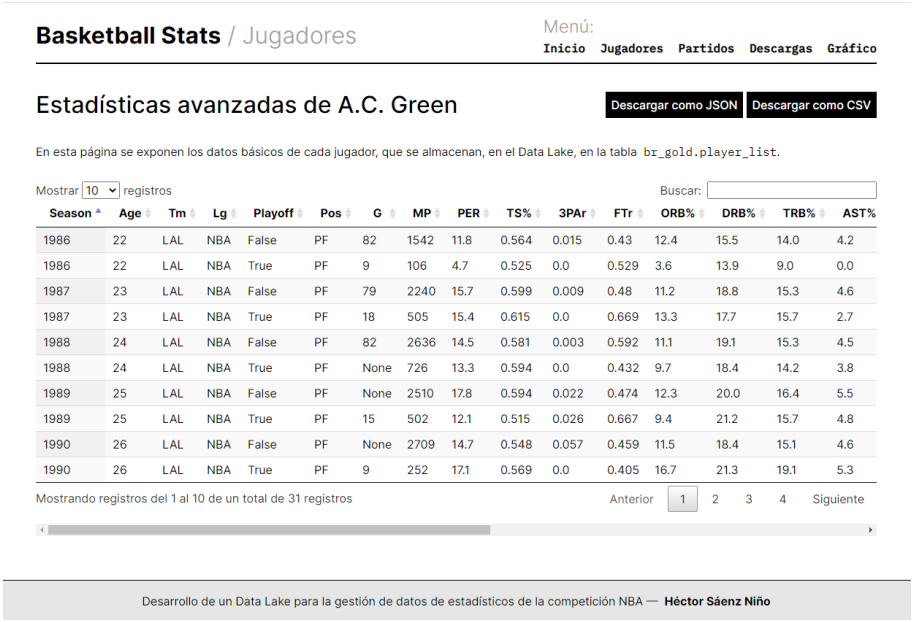


Figura 8.10: Estadísticas avanzadas de un jugador.

8.3.3. Partidos

Pestaña con el listado de partidos para una temporada, almacenadas en tabla `br_gold.schedule`. En la Figura 8.11 se puede ver esta pestaña, que dispone de:

1. Descargar como JSON o CSV los datos de todos los jugadores.
2. Selector de temporada. Cambiar el número de la temporada y pinchar en el botón azul.
3. Enlaces a las estadísticas avanzadas de los jugadores.

8.3.3.1. Boxscores

Pestaña con los boxscores. Se accederá a esta pestaña a partir de la de partido (ver Apartado 8.3.3), y se mostrarán los datos almacenados en las tablas `br_gold.boxscores_player` y `br_gold.boxscores_team` para el jugador seleccionado. En la Figura 8.12 se puede ver esta pestaña.

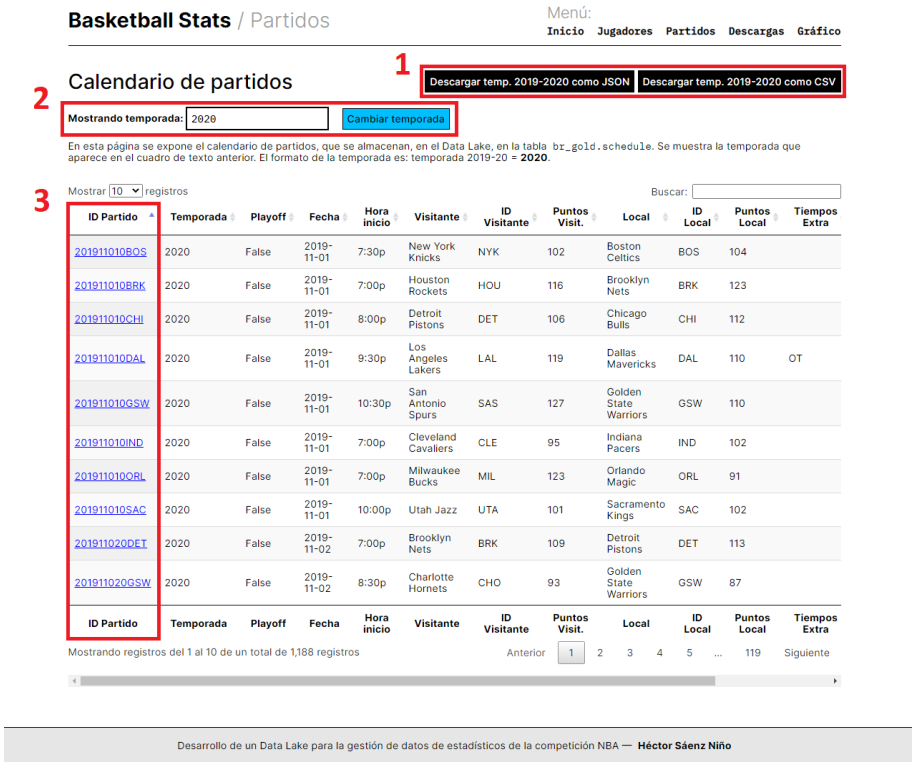


Figura 8.11: Listado de partidos.

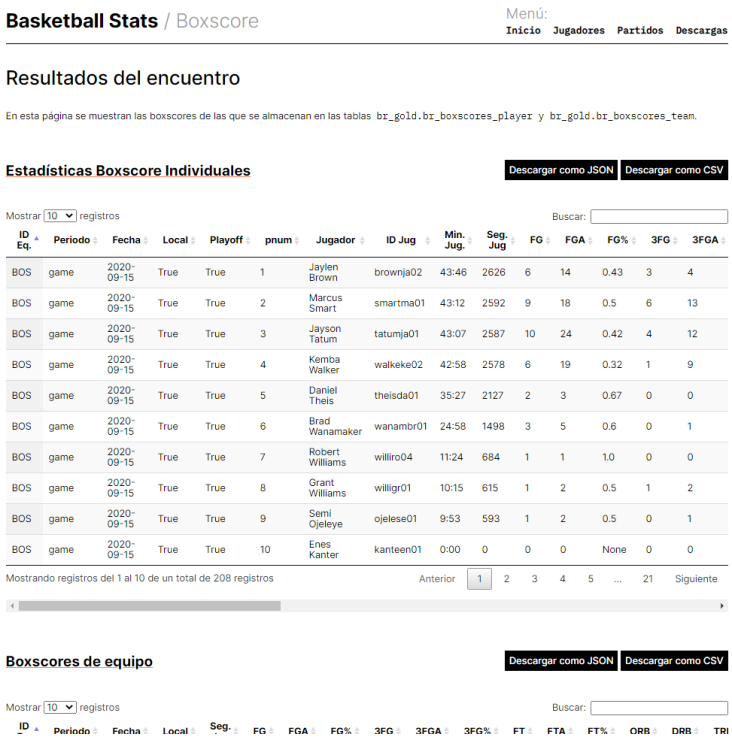


Figura 8.12: Boxscores de un partido.

8.3.3.2. Descargas

Pestaña para descargar los datos en CSV o JSON. Se puede ver en la Figura 8.13. Se pueden descargar datos de las siguientes tablas:

- `br_gold.schedule`: para obtener el calendario de partidos entre dos fechas.
- `br_gold.boxscores_player`: para obtener los datos de las *boxscores* a nivel de jugador entre dos fechas.
- `br_gold.boxscores_team`: ídem anterior, pero a nivel de equipo.
- `br_gold.player_list`: descarga del listado de jugadores completo.

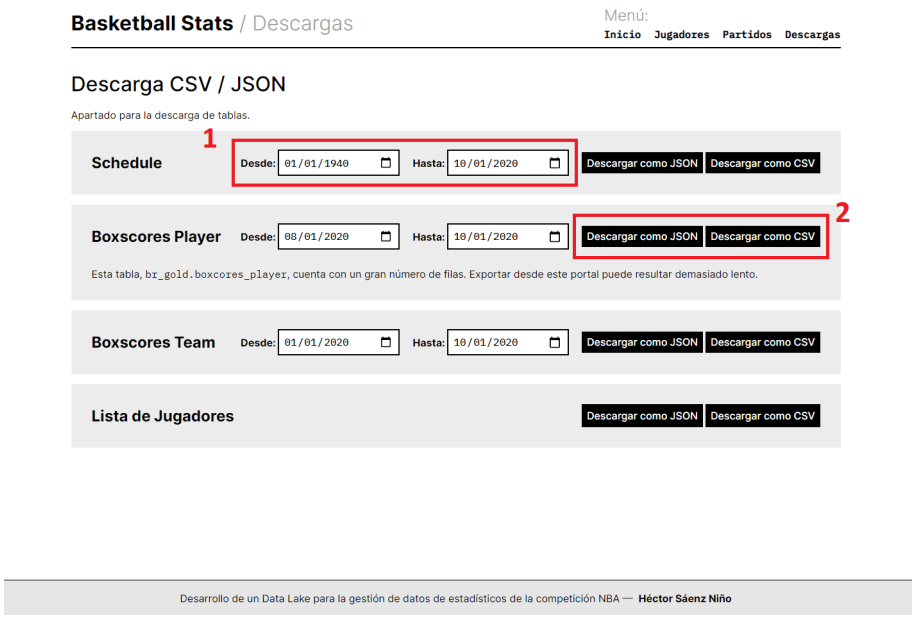


Figura 8.13: Pestaña de descargas.

8.3.4. Gráfico peso-altura

En esta pestaña, accesible mediante el enlace presente en la página de inicio o por la URL `/graphs/player-size`, se presenta un gráfico de dispersión de la altura y peso de los jugadores. Al pasar el cursor sobre un punto, un *tooltip* especifica el nombre, peso, altura e identificador del jugador en cuestión. En la Figura 6.17 se muestra esta gráfica.

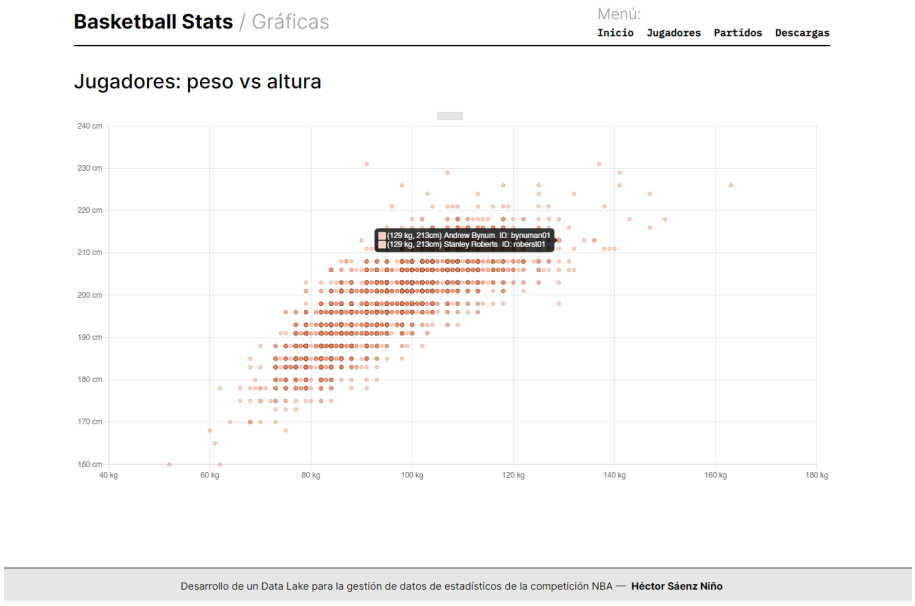


Figura 8.14: Gráfico de dispersión peso-altura.

Capítulo 9

Contenidos del Repositorio de Código

La presente memoria se complementa con los siguientes archivos, que se encuentran disponibles en `https://gitlab.inf.uva.es/hecsaen/tfg-basket`:

- **`flask/`**: directorio con los archivos de la interfaz *web*.
- **`hive/`**: directorio que contiene los archivos SQL para Hive e Impala.
  - **`raw/`**: carpeta con los *scripts* de la capa en crudo.
    - **`create/`**: carpeta que contiene los *scripts* SQL de creación de la capa en crudo.
    - **`insert_into/`**:
  - **`silver/`**: directorio con los *scripts* de la capa *silver*.
    - **`create/`**: carpeta que contiene los *scripts* SQL de creación.
    - **`insert_into/`**: carpeta que contiene los *scripts* para poblar los datos de las diferentes tablas.
  - **`gold/`**: directorio con los *scripts* de la capa *gold*.
    - **`create/`**: directorio que contiene los *scripts* SQL de creación.
    - **`insert_into/`**: directorio que contiene los *scripts* para poblar los datos de las diferentes tablas.
    - **`daily_recalculation/`**
      - **`daily_recalculation.sql`**
- **`nifi/`**: directorio que contiene los archivos de configuración de Apache NiFi.
- **`scrapy/`**
  - **`basketball_reference/`**: directorio que contiene el proyecto de Scrapy.
- **`scripts/`**

- **`daily`**: *script* para ejecutar las arañas de partidos y jugadores con una frecuencia diaria.
- **`starttfgservices`**: *script* que se ejecuta al iniciarse la máquina, y lanza los servicios de Apache NiFi, Scrapyd y Flask.

Capítulo 10

Discusión, conclusiones y trabajo futuro

El propósito de este capítulo es el de recopilar las conclusiones a las que se ha llegado, las dificultades que se han afrontado, aspectos que han quedado fuera del alcance del proyecto y las futuras líneas de trabajo para la ampliación y mejora de este proyecto.

En este punto del proyecto, conviene echar la vista atrás hacia los objetivos marcados al inicio (ver Apartado 1.3). En una primera fase se procedió a la exploración de los datos con los que se iba a trabajar y su modelado. Acto seguido, se estudiaron buenas prácticas y librerías para realizar *web scraping*, con el propósito de implementar un sistema fiable y fácilmente modificable. Se prestó una especial atención al sistema de extracción de datos, ya que, al ser la única fuente que iba a ser consultada y el número de peticiones era alto, se debía asegurar el correcto funcionamiento. Cabe destacar que el uso de un *framework* fue acertado, ya que establece pautas para evitar errores comunes y aporta herramientas para el desarrollo. De esta manera, se ha materializado un sistema de extracción de datos fiable, pero no se han abordado todos los datos que se contempló extraer inicialmente.

Posteriormente se procedió al estudio de arquitecturas y tecnologías de Big Data. Esta parte ha sido la más exigente en el proyecto, debido a que no forma parte del currículo del grado actual y la escasa madurez (y consiguiente falta de literatura) de los Data Lakes[1]. Se han entendido las ventajas y deficiencias que presentan los Data Lakes frente a sistemas tradicionales como los Data Warehouses, que podrían resumirse en flexibilidad y escalabilidad frente a integridad y seguridad. La implementación de las tecnologías de Big Data ha resultado especialmente difícil, debido a la falta de práctica con las mismas.

Por último, se desarrolló una interfaz *web* para facilitar el acceso a los datos e incorporar tablas y gráficas para su análisis y comprensión, y supone la conclusión del objetivo principal: el desarrollo de un repositorio central para datos estadísticos de la NBA de manera extensible y escalable.

[1]La primera aparición del término Data Lake se atribuye a James Dixon en 2010, por entonces presidente ejecutivo de Pentaho, en la siguiente publicación de su *blog*: `https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/`

10.1. Líneas de trabajo futuro

A continuación se listarán una serie de aspectos a ampliar o que no han sido abordados en este proyecto:

- **Extracción de un mayor número y variedad de datos**: desde ligas diferentes a la NBA a otro tipo de datos no extraídos, como el *play-by-play* y las zonas de tiro.
- **Cálculo de nuevas estadísticas**: cálculo de nuevas estadísticas a partir de los datos que se han recogido. Un ejemplo de esto sería calcular el PER de un jugador en una temporada en un día concreto.
- **Incorporación de datos a tiempo real**: consulta de datos a una fuente a tiempo real para el análisis de partidos a tiempo real. Para ello, sería conveniente añadir una capa a tiempo real al sistema.
- **Mejora de la interfaz**: incorporación de nuevos gráficos que faciliten el análisis de los datos. Además, la interfaz actual sufre de problemas de rendimiento al descargar grandes cantidades de datos debido al uso de la librería Impyla[2]. Para mejorar estos problemas de rendimiento, se debería explorar el uso de otros métodos.
- **Despliegue del sistema en un mayor número de nodos**: las tecnologías en las que se ha implementado el Data Lake y más concretamente el ecosistema Hadoop, se aprovechan de sistemas distribuidos para ofrecer mayor capacidad de cómputo y tolerancia a fallos. En función del crecimiento de los datos, sería interesante desplegar el sistema en la nube, con servicios flexibles como Amazon Elastic MapReduce, que permitan alquilar un mayor número de nodos durante poco tiempo, consiguiendo mejorar los tiempos de computación sin aumentar los costes.
- **Implementación de algoritmos de aprendizaje automático**: integración de los métodos implementados en los proyectos realizados por Álvaro Berrío Galindo y Javier Martínez García, para obtener predicciones de futuros partidos tan actualizadas como sea posible y de manera automática.

[2] `https://github.com/cloudera/impyla`

Apéndice A

Nomenclatura

Estadísticas básicas

Se denominará estadísticas básicas a los siguientes:

MP	Minutos jugados.	*ORB*	Rebotes ofensivos.
FG	Tiros de campo anotados.	*DRB*	Rebotes defensivos.
FGA	Tiros de campo intentados.	*TRB*	Rebotes totales.
FG %	Porcentaje de acierto de canastas.	*AST*	Asistencias.
3P	Tiros triples anotados.	*STL*	Robos.
3PA	Tiros triples realizados.	*BLK*	Tapones.
3P %	Porcentaje	*TOV*	Pérdidas de posesión.
FT	Tiros libres anotados.	*PF*	Faltas personales.
FTA	Tiros libres.	*PTS*	Puntos anotados.
FT %	Porcentaje de tiros libres anotados.	*+/-*	Estadística *Plus/minus* o "Más/menos"

Estadísticas avanzadas

Se denominará estadísticas avanzadas a las listadas a continuación. Su cálculo se puede ver en el Apartado 5.5.

TS %	Porcentaje de Tiro Verdadero.
eFG %	Tiros de Campo efectivos.
3PAr	Tasa de Intentos de Triples.
FTr	Tasa de Intentos de Tiros Libres.
ORB %	Porcentaje de Rebotes Ofensivos.
DRB %	Porcentaje de Rebotes Defensivos.
TRB %	Porcentaje de Rebotes Totales.
AST %	Porcentaje de Asistencias.

STL %	Porcentaje de Robos.
BLK %	Porcentaje de Bloqueos.
TOV %	Porcentaje de Pérdidas.
USG %	Porcentaje de Jugadas Usadas por un jugador mientras se encontraba en el terreno de juego.
ORtg	Puntuación ofensiva: Estimación de puntos producidos por 100 posesiones.
DRtg	Puntuación defensiva: estimación de puntos permitidos al contrario por 100 posesiones.
BPM	Estimación de *plus/minus* normalizado con el resto de jugadores.

Estadísticas avanzadas de jugador

PER	Puntuación de eficiencia de jugador *(Player Efficiency Rating)*: indicador que pretende reflejar el rendimiento general de un jugador a un número.
OWS	Participaciones de victoria ofensivas *(Offensive Win Shares)*: estimación de las victorias debidas a la participación ofensiva de un jugador.
DWS	Participaciones de victoria defensivas *(Defensive Win Shares)*: estimación de las victorias debidas a la participación defensiva de un jugador.
WS	Participaciones de victoria *(Win Shares)*: estimación de las victorias debidas a la participación de un jugador.
WS/48	Participaciones de victoria por 48 minutos *(Win Shares Per 48 minutes)*: estimación de las victorias por encuentro debidas a la participación de un jugador.

Apéndice B

Scripts Hive

B.1. Capa en crudo (raw layer)

B.1.1. Tabla `br_raw.boxscores_basic`

```
CREATE DATABASE IF NOT EXISTS br_raw;

CREATE EXTERNAL TABLE IF NOT EXISTS br_raw.boxscores_basic (
    `game_id` STRING,
    `team_id` STRING,
    `box_type` STRING,
    `date` STRING,
    `ishome` BOOLEAN,
    `pnum` INT,
    `player` STRING,
    `player_href` STRING,
    `player_csk` STRING,
    `player_id` STRING,
    `mp` STRING,
    `sp` INT,
    `fg` INT,
    `fga` INT,
    `fg_pct` FLOAT,
    `fg3` INT,
    `fg3a` INT,
    `fg3_pct` FLOAT,
    `ft` INT,
    `fta` INT,
    `ft_pct` FLOAT,
    `orb` INT,
    `drb` INT,
    `trb` INT,
    `ast` INT,
    `stl` INT,
    `blk` INT,
    `tov` INT,
    `pf` INT,
    `pts` INT,
    `plus_minus` INT,
```

```
    `reason` STRING
)
PARTITIONED BY (year int)
ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.OpenCSVSerde"
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar" = '"'
)
TBLPROPERTIES ("skip.header.line.count"="1");


ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1950) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1950';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1951) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1951';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1952) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1952';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1953) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1953';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1954) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1954';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1955) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1955';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1956) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1956';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1957) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1957';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1958) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1958';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1959) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1959';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1960) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1960';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1961) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1961';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1962) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1962';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1963) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1963';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1964) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1964';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1965) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1965';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1966) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1966';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1967) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1967';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1968) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1968';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1969) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1969';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1970) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1970';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1971) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1971';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1972) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1972';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1973) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1973';
```

```
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1974) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1974';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1975) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1975';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1976) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1976';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1977) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1977';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1978) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1978';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1979) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1979';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1980) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1980';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1981) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1981';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1982) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1982';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1983) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1983';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1984) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1984';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1985) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1985';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1986) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1986';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1987) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1987';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1988) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1988';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1989) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1989';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1990) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1990';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1991) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1991';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1992) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1992';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1993) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1993';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1994) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1994';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1995) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1995';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1996) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1996';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1997) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1997';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1998) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1998';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 1999) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/1999';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2000) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2000';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2001) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2001';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2002) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2002';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2003) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2003';
```

```
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2004) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2004';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2005) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2005';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2006) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2006';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2007) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2007';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2008) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2008';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2009) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2009';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2010) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2010';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2011) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2011';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2012) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2012';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2013) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2013';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2014) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2014';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2015) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2015';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2016) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2016';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2017) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2017';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2018) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2018';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2019) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2019';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2020) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2020';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2021) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2021';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2022) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2022';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2023) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2023';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2024) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2024';
ALTER TABLE br_raw.boxscores_basic ADD IF NOT EXISTS PARTITION (year = 2025) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/basic/2025';
```

B.1.2. Tabla `br_raw.boxscores_advanced`

```
CREATE DATABASE IF NOT EXISTS br_raw;

CREATE EXTERNAL TABLE IF NOT EXISTS br_raw.boxscores_advanced (
    `game_id` STRING,
    `team_id` STRING,
    `box_type` STRING,
    `date` STRING,
    `ishome` BOOLEAN,
    `pnum` INT,
```

```
    `player` STRING,
    `player_href` STRING,
    `player_csk` STRING,
    `player_id` STRING,
    `mp` STRING,
    `sp` INT,
    `ts_pct` FLOAT,
    `efg_pct` FLOAT,
    `fg3a_per_fga_pct` FLOAT,
    `fta_per_fga_pct` FLOAT,
    `orb_pct` FLOAT,
    `drb_pct` FLOAT,
    `trb_pct` FLOAT,
    `ast_pct` FLOAT,
    `stl_pct` FLOAT,
    `blk_pct` FLOAT,
    `tov_pct` FLOAT,
    `usg_pct` FLOAT,
    `off_rtg` FLOAT,
    `def_rtg` FLOAT,
    `bpm` FLOAT,
    `reason` STRING
)
PARTITIONED BY (year int)
ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.OpenCSVSerde"
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar" = '"'
)
TBLPROPERTIES ("skip.header.line.count"="1");


ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1983) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1983';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1984) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1984';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1985) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1985';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1986) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1986';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1987) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1987';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1988) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1988';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1989) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1989';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1990) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1990';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1991) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1991';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1992) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1992';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1993) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1993';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1994) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1994';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1995) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1995';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1996) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1996';
```

```
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1997) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1997';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1998) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1998';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 1999) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/1999';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2000) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2000';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2001) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2001';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2002) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2002';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2003) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2003';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2004) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2004';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2005) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2005';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2006) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2006';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2007) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2007';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2008) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2008';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2009) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2009';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2010) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2010';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2011) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2011';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2012) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2012';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2013) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2013';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2014) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2014';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2015) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2015';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2016) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2016';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2017) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2017';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2018) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2018';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2019) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2019';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2020) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2020';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2021) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2021';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2022) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2022';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2023) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2023';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2024) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2024';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2025) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2025';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2026) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2026';
```

```
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2027) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2027';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2028) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2028';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2029) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2029';
ALTER TABLE br_raw.boxscores_advanced ADD IF NOT EXISTS PARTITION (year = 2030) LOCATION
↪  '/user/cloudera/br/raw/games/boxscores/advanced/2030';
```

B.1.3. Script Tabla `br_raw.player_data`

```
CREATE DATABASE IF NOT EXISTS br_raw;

CREATE EXTERNAL TABLE IF NOT EXISTS br_raw.player_data (
    player_json STRING
)
PARTITIONED BY (first_letter STRING);

ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "a") LOCATION
↪  "/user/cloudera/br/raw/players/data/a";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "b") LOCATION
↪  "/user/cloudera/br/raw/players/data/b";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "c") LOCATION
↪  "/user/cloudera/br/raw/players/data/c";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "d") LOCATION
↪  "/user/cloudera/br/raw/players/data/d";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "e") LOCATION
↪  "/user/cloudera/br/raw/players/data/e";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "f") LOCATION
↪  "/user/cloudera/br/raw/players/data/f";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "g") LOCATION
↪  "/user/cloudera/br/raw/players/data/g";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "h") LOCATION
↪  "/user/cloudera/br/raw/players/data/h";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "i") LOCATION
↪  "/user/cloudera/br/raw/players/data/i";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "j") LOCATION
↪  "/user/cloudera/br/raw/players/data/j";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "k") LOCATION
↪  "/user/cloudera/br/raw/players/data/k";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "l") LOCATION
↪  "/user/cloudera/br/raw/players/data/l";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "m") LOCATION
↪  "/user/cloudera/br/raw/players/data/m";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "n") LOCATION
↪  "/user/cloudera/br/raw/players/data/n";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "o") LOCATION
↪  "/user/cloudera/br/raw/players/data/o";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "p") LOCATION
↪  "/user/cloudera/br/raw/players/data/p";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "q") LOCATION
↪  "/user/cloudera/br/raw/players/data/q";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "r") LOCATION
↪  "/user/cloudera/br/raw/players/data/r";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "s") LOCATION
↪  "/user/cloudera/br/raw/players/data/s";
```

```
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "t") LOCATION
↪  "/user/cloudera/br/raw/players/data/t";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "u") LOCATION
↪  "/user/cloudera/br/raw/players/data/u";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "v") LOCATION
↪  "/user/cloudera/br/raw/players/data/v";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "w") LOCATION
↪  "/user/cloudera/br/raw/players/data/w";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "x") LOCATION
↪  "/user/cloudera/br/raw/players/data/x";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "y") LOCATION
↪  "/user/cloudera/br/raw/players/data/y";
ALTER TABLE br_raw.player_data ADD IF NOT EXISTS PARTITION (first_letter = "z") LOCATION
↪  "/user/cloudera/br/raw/players/data/z";
```

B.1.4. Script Tabla `br_raw.player_list`

```
CREATE DATABASE IF NOT EXISTS br_raw;

CREATE EXTERNAL TABLE IF NOT EXISTS br_raw.player_list (
   player STRING,
   player_href STRING,
   player_id STRING,
   year_min STRING,
   year_max STRING,
   pos STRING,
   weight STRING,
   height STRING,
   height_csk STRING,
   birth_date STRING,
   birth_date_csk STRING,
   colleges STRING
)
ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.OpenCSVSerde"
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar" = '"'
)
LOCATION "/user/cloudera/br/raw/players/list"
TBLPROPERTIES ("skip.header.line.count"="1");
```

B.1.5. Script Tabla `br_raw.schedule_games`

```
CREATE DATABASE IF NOT EXISTS br_raw;

CREATE EXTERNAL TABLE IF NOT EXISTS br_raw.schedule_games (
    `game_id` STRING,
    `date` STRING,
    `start_time` STRING,
    `visitor_team` STRING,
    `visitor_team_id` STRING,
    `visitor_pts` STRING,
    `home_team` STRING,
```

```
    `home_team_id` STRING,
    `home_pts` STRING,
    `overtimes` STRING,
    `attendance` STRING,
    `game_remarks` STRING
)
ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.OpenCSVSerde"
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar" = '"'
)
LOCATION '/user/cloudera/br/raw/schedule/games'
TBLPROPERTIES ("skip.header.line.count"="1");
```

B.1.6. Script Tabla `br_raw.schedule_playoffs`

```
CREATE DATABASE IF NOT EXISTS br_raw;

CREATE EXTERNAL TABLE IF NOT EXISTS br_raw.schedule_playoffs (
    `game_id` STRING,
    `date` STRING,
    `start_time` STRING,
    `visitor_team` STRING,
    `visitor_team_id` STRING,
    `visitor_pts` STRING,
    `home_team` STRING,
    `home_team_id` STRING,
    `home_pts` STRING,
    `overtimes` STRING,
    `attendance` STRING,
    `game_remarks` STRING
)
ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.OpenCSVSerde"
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar" = '"'
)
LOCATION '/user/cloudera/br/raw/schedule/playoffs'
TBLPROPERTIES ("skip.header.line.count"="1");
```

B.2. Capa Silver

B.2.1. Scripts de creación de las tablas

B.2.1.1. Tabla `br_silver.boxscores_basic`

```
CREATE DATABASE IF NOT EXISTS br_silver
LOCATION '/user/cloudera/br/silver';

CREATE TABLE IF NOT EXISTS br_silver.boxscores_basic (
    `game_id` STRING,
    `team_id` STRING,
```

```
    `box_type` STRING,
    `date` STRING,
    `ishome` BOOLEAN,
    `pnum` INT,
    `player` STRING,
    `player_href` STRING,
    `player_csk` STRING,
    `player_id` STRING,
    `mp` STRING,
    `sp` INT,
    `fg` INT,
    `fga` INT,
    `fg_pct` FLOAT,
    `fg3` INT,
    `fg3a` INT,
    `fg3_pct` FLOAT,
    `ft` INT,
    `fta` INT,
    `ft_pct` FLOAT,
    `orb` INT,
    `drb` INT,
    `trb` INT,
    `ast` INT,
    `stl` INT,
    `blk` INT,
    `tov` INT,
    `pf` INT,
    `pts` INT,
    `plus_minus` INT,
    `reason` STRING
)
PARTITIONED BY (`season` INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC;
```

B.2.1.2. Tabla `br_silver.boxscores_basic_team_totals`

```
CREATE DATABASE IF NOT EXISTS br_silver
LOCATION '/user/cloudera/br/silver';

CREATE TABLE IF NOT EXISTS br_silver.boxscores_basic_team_totals (
    `game_id` STRING,
    `team_id` STRING,
    `box_type` STRING,
    `date` STRING,
    `ishome` BOOLEAN,
    `sp` INT,
    `fg` INT,
    `fga` INT,
    `fg_pct` FLOAT,
    `fg3` INT,
    `fg3a` INT,
    `fg3_pct` FLOAT,
    `ft` INT,
    `fta` INT,
```

```
    `ft_pct` FLOAT,
    `orb` INT,
    `drb` INT,
    `trb` INT,
    `ast` INT,
    `stl` INT,
    `blk` INT,
    `tov` INT,
    `pf` INT,
    `pts` INT,
    `plus_minus` INT
)
PARTITIONED BY (`season` INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC;
```

B.2.1.3. Tabla br_silver.schedule

```
CREATE DATABASE IF NOT EXISTS br_silver
LOCATION '/user/cloudera/br/silver';

-- SILVER GAME SCHEDULE

CREATE TABLE IF NOT EXISTS br_silver.schedule (
    `game_id` STRING,
    `season` STRING,
    `isplayoff` BOOLEAN,
    `date` STRING,
    `start_time` STRING,
    `visitor_team` STRING,
    `visitor_team_id` STRING,
    `visitor_pts` INT,
    `home_team` STRING,
    `home_team_id` STRING,
    `home_pts` INT,
    `overtimes` STRING,
    `attendance` STRING,
    `game_remarks` STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC;
```

B.2.1.4. Tabla br_silver.advanced_player_stats

```
CREATE DATABASE IF NOT EXISTS br_silver
LOCATION '/user/cloudera/br/silver';

DROP TABLE IF EXISTS br_silver.advanced_player_stats;
CREATE TABLE IF NOT EXISTS br_silver.advanced_player_stats (
    player_id STRING,
    isplayoff BOOLEAN,
```

```
    season INT,
    season_href STRING,
    age INT,
    team_id STRING,
    team_id_href STRING,
    lg_id STRING,
    lg_id_href STRING,
    pos STRING,
    g INT,
    mp INT,
    per FLOAT,
    ts_pct FLOAT,
    fg3a_per_fga_pct FLOAT,
    fta_per_fga_pct FLOAT,
    orb_pct FLOAT,
    drb_pct FLOAT,
    trb_pct FLOAT,
    ast_pct FLOAT,
    stl_pct FLOAT,
    blk_pct FLOAT,
    tov_pct FLOAT,
    usg_pct FLOAT,
    ows FLOAT,
    dws FLOAT,
    ws FLOAT,
    ws_per_48 FLOAT,
    obpm FLOAT,
    dbpm FLOAT,
    bpm FLOAT,
    vorp FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS ORC;
```

B.2.2. Población tablas (INSERT INTO) capa Silver

B.2.2.1. Script Población `br_silver.boxscores_basic`

Este *script* acepta parámetros, ya que con los recursos disponibles no se podía completar la tarea completa, y era necesario desglosar esta operación en partes.

```
set hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE br_silver.boxscores_basic PARTITION(season)
SELECT
    game_id,
    team_id,
    regexp_replace(box_type, "-basic$", "") AS box_type,
    date,
    if( locate(team_id, game_id) > 0, true, false ) AS ishome,
    pnum,
    player,
    player_href,
    player_csk,
```

```
    regexp_extract(player_href, '/players/[a-z]/(.*).html', 1) as player_id,
    mp,
    sp,
    fg,
    fga,
    fg_pct,
    fg3,
    fg3a,
    fg3_pct,
    ft,
    fta,
    ft_pct,
    orb,
    drb,
    trb,
    ast,
    stl,
    blk,
    tov,
    pf,
    pts,
    plus_minus,
    reason,
    IF(SUBSTR(date, 6, 2) < "11",
        year,
        year + 1
    )
    AS season
FROM br_raw.boxscores_basic
WHERE year >= ${from_season_beginning_year} - 1
AND   year <= ${to_season_ending_year} + 1
AND   date > CONCAT(CAST(\${from_season_beginning_year} - 1 AS STRING), "-10-01")
AND   date <= CONCAT(CAST(\${to_season_ending_year} + 1 AS STRING), "-10-01");

set hive.exec.dynamic.partition.mode=strict;
```

B.2.2.2. Script Población `br_silver.boxscores_basic_team_totals`

```
set hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE br_silver.boxscores_basic_team_totals PARTITION(season)
SELECT
    game_id,
    team_id,
    box_type,
    date,
    ishome,
    SUM(sp) AS sp,
    SUM(fg) AS fg,
    SUM(fga) AS fga,
    SUM(fg) / SUM(fga) AS fg_pct,
    SUM(fg3) AS fg3,
    SUM(fg3a) AS fg3a,
    SUM(fg3) / SUM(fg3a) AS fg3_pct,
    SUM(ft) AS ft,
    SUM(fta) AS fta,
```

```
    SUM(ft) / SUM(fta) AS ft_pct,
    SUM(orb) AS orb,
    SUM(drb) AS dr,
    SUM(trb) AS trb,
    SUM(ast) AS as,
    SUM(stl) AS stl,
    SUM(blk) AS blk,
    SUM(tov) AS tov,
    SUM(pf) AS pf,
    SUM(pts) AS pts,
    SUM(plus_minus) AS plus_minus,
    season
FROM
    br_silver.boxscores_basic
WHERE
    season >= ${from_season}
AND
    season <= ${to_season}
GROUP BY game_id, team_id, date, box_type, ishome, season;

set hive.exec.dynamic.partition.mode=strict;
```

B.2.2.3. Script Población `br_silver.schedule`

```
INSERT OVERWRITE TABLE br_silver.schedule
SELECT
    g.game_id,
    IF(SUBSTR(g.date, 6, 2) < "11",
        CAST(SUBSTR(g.date, 1, 4) AS INT),
        CAST(SUBSTR(g.date, 1, 4) AS INT) + 1
    )
    AS season,
    IF(p.game_id IS NOT NULL, TRUE, FALSE) as isplayoff,
    g.date,
    g.start_time,
    g.visitor_team,
    g.visitor_team_id,
    g.visitor_pts,
    g.home_team,
    g.home_team_id,
    g.home_pts,
    g.overtimes,
    g.attendance,
    g.game_remarks
FROM
    br_raw.schedule_games g
LEFT JOIN
    br_raw.schedule_playoffs p
ON (g.game_id = p.game_id);
```

B.2.2.4. Script Población br_silver.advanced_player_stats

```
CREATE DATABASE IF NOT EXISTS br_silver
LOCATION '/user/cloudera/br/silver';

INSERT OVERWRITE TABLE  br_silver.advanced_player_stats
SELECT
    get_json_object(json_row, '$.player_id') AS player_id,
    FALSE AS isplayoff,

    CAST(SUBSTR(get_json_object(json_row, '$.season'), 1, 4) AS INT) + 1 AS season,
    get_json_object(json_row, '$.season_href') AS season_href,
    get_json_object(json_row, '$.age') AS age,
    get_json_object(json_row, '$.team_id') AS team_id,
    get_json_object(json_row, '$.team_id_href') AS team_id_href,
    get_json_object(json_row, '$.lg_id') AS lg_id,
    get_json_object(json_row, '$.lg_id_href') AS lg_id_href,
    get_json_object(json_row, '$.pos') AS pos,
    get_json_object(json_row, '$.g') AS g,
    get_json_object(json_row, '$.mp') AS mp,

    get_json_object(json_row, '$.per') AS per,
    get_json_object(json_row, '$.ts_pct') AS ts_pct,
    get_json_object(json_row, '$.fg3a_per_fga_pct') AS fg3a_per_fga_pct,
    get_json_object(json_row, '$.fta_per_fga_pct') AS fta_per_fga_pct,
    get_json_object(json_row, '$.orb_pct') AS orb_pct,
    get_json_object(json_row, '$.drb_pct') AS drb_pct,
    get_json_object(json_row, '$.trb_pct') AS trb_pct,
    get_json_object(json_row, '$.ast_pct') AS ast_pct,
    get_json_object(json_row, '$.stl_pct') AS stl_pct,
    get_json_object(json_row, '$.blk_pct') AS blk_pct,
    get_json_object(json_row, '$.tov_pct') AS tov_pct,
    get_json_object(json_row, '$.usg_pct') AS usg_pct,
    get_json_object(json_row, '$.ows') AS ows,
    get_json_object(json_row, '$.dws') AS dws,
    get_json_object(json_row, '$.ws') AS ws,
    get_json_object(json_row, '$.ws_per_48') AS ws_per_48,
    get_json_object(json_row, '$.obpm') AS obpm,
    get_json_object(json_row, '$.dbpm') AS dbpm,
    get_json_object(json_row, '$.bpm') AS bpm,
    get_json_object(json_row, '$.vorp') AS vorp
FROM (
    select json_row from (
    select split(regexp_replace(regexp_extract(t1.stat,'\\[(.+)\\]',1),'\\}([ \t\n\r]*)\\,([
    ↪  \t\n\r]*)\\{', '\\}\\|\\|\\{'),'\\|\\|') as str
    from (
        SELECT get_json_object(player_json, '$.advanced') AS stat FROM br_raw.player_data
    ) AS t1) aux
    lateral view explode(aux.str) ss as json_row
) AS t;


INSERT INTO TABLE  br_silver.advanced_player_stats
SELECT
    get_json_object(json_row, '$.player_id') AS player_id,
    TRUE AS isplayoff,

    CAST(SUBSTR(get_json_object(json_row, '$.season'), 1, 4) AS INT) + 1 AS season,
```

```
    get_json_object(json_row, '$.season_href') AS season_href,
    get_json_object(json_row, '$.age') AS age,
    get_json_object(json_row, '$.team_id') AS team_id,
    get_json_object(json_row, '$.team_id_href') AS team_id_href,
    get_json_object(json_row, '$.lg_id') AS lg_id,
    get_json_object(json_row, '$.lg_id_href') AS lg_id_href,
    get_json_object(json_row, '$.pos') AS pos,
    get_json_object(json_row, '$.g') AS g,
    get_json_object(json_row, '$.mp') AS mp,

    get_json_object(json_row, '$.per') AS per,
    get_json_object(json_row, '$.ts_pct') AS ts_pct,
    get_json_object(json_row, '$.fg3a_per_fga_pct') AS fg3a_per_fga_pct,
    get_json_object(json_row, '$.fta_per_fga_pct') AS fta_per_fga_pct,
    get_json_object(json_row, '$.orb_pct') AS orb_pct,
    get_json_object(json_row, '$.drb_pct') AS drb_pct,
    get_json_object(json_row, '$.trb_pct') AS trb_pct,
    get_json_object(json_row, '$.ast_pct') AS ast_pct,
    get_json_object(json_row, '$.stl_pct') AS stl_pct,
    get_json_object(json_row, '$.blk_pct') AS blk_pct,
    get_json_object(json_row, '$.tov_pct') AS tov_pct,
    get_json_object(json_row, '$.usg_pct') AS usg_pct,
    get_json_object(json_row, '$.ows') AS ows,
    get_json_object(json_row, '$.dws') AS dws,
    get_json_object(json_row, '$.ws') AS ws,
    get_json_object(json_row, '$.ws_per_48') AS ws_per_48,
    get_json_object(json_row, '$.obpm') AS obpm,
    get_json_object(json_row, '$.dbpm') AS dbpm,
    get_json_object(json_row, '$.bpm') AS bpm,
    get_json_object(json_row, '$.vorp') AS vorp
FROM (
    select json_row from (
    select split(regexp_replace(regexp_extract(t1.stat,'\\[(.+)\\]',1),'\\}([ \t\n\r]*)\\,([
    ↪  \t\n\r]*)\\{', '\\}\\|\\|\\{'),'\\|\\|') as str
    from (
        SELECT get_json_object(player_json, '$.playoffs_advanced') AS stat FROM
        ↪  br_raw.player_data
    ) AS t1) aux
    lateral view explode(aux.str) ss as json_row
) AS t;
```

B.3. Capa Gold

B.3.1. Scripts de creación de las tablas

B.3.1.1. Tabla `br_gold.boxscores_player`

```
CREATE DATABASE IF NOT EXISTS br_gold
LOCATION '/user/cloudera/br/gold';

CREATE TABLE IF NOT EXISTS br_gold.boxscores_player (
    `game_id` STRING,
    `team_id` STRING,
    `box_type` STRING,
    `date` STRING,
```

```
    `ishome` BOOLEAN,
    `isplayoff` BOOLEAN,
    `pnum` INT,
    `player` STRING,
    `player_href` STRING,
    `player_csk` STRING,
    `player_id` STRING,
    `mp` STRING,
    `sp` INT,
    `fg` INT,
    `fga` INT,
    `fg_pct` FLOAT,
    `fg3` INT,
    `fg3a` INT,
    `fg3_pct` FLOAT,
    `ft` INT,
    `fta` INT,
    `ft_pct` FLOAT,
    `orb` INT,
    `drb` INT,
    `trb` INT,
    `ast` INT,
    `stl` INT,
    `blk` INT,
    `tov` INT,
    `pf` INT,
    `pts` INT,
    `plus_minus` INT,
    `ts_pct` FLOAT,
    `efg_pct` FLOAT,
    `fg3a_per_fga_pct` FLOAT,
    `fta_per_fga_pct` FLOAT,
    `orb_pct` FLOAT,
    `drb_pct` FLOAT,
    `trb_pct` FLOAT,
    `ast_pct` FLOAT,
    `stl_pct` FLOAT,
    `blk_pct` FLOAT,
    `tov_pct` FLOAT,
    `usg_pct` FLOAT,
    `off_rtg` FLOAT,
    `def_rtg` FLOAT,
    `bpm` FLOAT,
    `reason` STRING
)
PARTITIONED BY (`season` INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET;
```

B.3.1.2. Tabla `br_gold.boxscores_team`

```
CREATE DATABASE IF NOT EXISTS br_gold
LOCATION '/user/cloudera/br/gold';

CREATE TABLE IF NOT EXISTS br_gold.boxscores_team (
```

```
    `game_id` STRING,
    `team_id` STRING,
    `box_type` STRING,
    `date` STRING,
    `ishome` BOOLEAN,
    `sp` INT,
    `fg` INT,
    `fga` INT,
    `fg_pct` FLOAT,
    `fg3` INT,
    `fg3a` INT,
    `fg3_pct` FLOAT,
    `ft` INT,
    `fta` INT,
    `ft_pct` FLOAT,
    `orb` INT,
    `drb` INT,
    `trb` INT,
    `ast` INT,
    `stl` INT,
    `blk` INT,
    `tov` INT,
    `pf` INT,
    `pts` INT,
    `plus_minus` INT,
    `ts_pct` FLOAT,
    `efg_pct` FLOAT,
    `fg3a_per_fga_pct` FLOAT,
    `fta_per_fga_pct` FLOAT,
    `orb_pct` FLOAT,
    `drb_pct` FLOAT,
    `trb_pct` FLOAT,
    `ast_pct` FLOAT,
    `stl_pct` FLOAT,
    `blk_pct` FLOAT,
    `tov_pct` FLOAT,
    `usg_pct` FLOAT,
    `off_rtg` FLOAT,
    `def_rtg` FLOAT,
    `bpm` FLOAT
)
PARTITIONED BY (`season` INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET;
```

B.3.1.3. Tabla br_gold.player_list

```
CREATE DATABASE IF NOT EXISTS br_gold
LOCATION '/user/cloudera/br/gold';

CREATE TABLE IF NOT EXISTS br_gold.player_list (
   player STRING,
   player_id STRING,
   year_min INT,
   year_max INT,
```

```
   pos STRING,
   weight_lbs FLOAT,
   weight_kg FLOAT,
   height STRING,
   height_in FLOAT,
   height_cm FLOAT,
   birth_date STRING,
   colleges STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET;
```

B.3.1.4. Tabla br_gold.player_stats

```
CREATE DATABASE IF NOT EXISTS br_gold
LOCATION '/user/cloudera/br/gold';

CREATE TABLE IF NOT EXISTS br_gold.player_stats (
    team_id STRING,
    player STRING,
    player_href STRING,
    player_csk STRING,
    player_id STRING,
    mp INT,
    sp INT,
    fg INT,
    fga INT,
    fg_pct FLOAT,
    fg3 INT,
    fg3a INT,
    fg3_pct FLOAT,
    ft INT,
    fta INT,
    ft_pct FLOAT,
    orb INT,
    drb INT,
    trb INT,
    ast INT,
    stl INT,
    blk INT,
    tov INT,
    pf INT,
    pts INT,
    plus_minus INT,
    ts_pct FLOAT,
    efg_pct FLOAT,
    fg3a_per_fga_pct FLOAT,
    fta_per_fga_pct FLOAT,
    orb_pct FLOAT,
    drb_pct FLOAT,
    trb_pct FLOAT,
    ast_pct FLOAT,
    stl_pct FLOAT,
    blk_pct FLOAT,
    tov_pct FLOAT,
```

```
    usg_pct FLOAT,
    off_rtg FLOAT,
    def_rtg FLOAT,
    bpm FLOAT
)
PARTITIONED BY (season INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET;
```

B.3.1.5. Tabla br_gold.schedule

```
CREATE DATABASE IF NOT EXISTS br_GOLD
LOCATION '/user/cloudera/br/GOLD';

-- GOLD GAME SCHEDULE

CREATE TABLE IF NOT EXISTS br_gold.schedule (
    `game_id` STRING,
    `season` STRING,
    `isplayoff` BOOLEAN,
    `date` STRING,
    `start_time` STRING,
    `visitor_team` STRING,
    `visitor_team_id` STRING,
    `visitor_pts` INT,
    `home_team` STRING,
    `home_team_id` STRING,
    `home_pts` INT,
    `overtimes` STRING,
    `attendance` STRING,
    `game_remarks` STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET;
```

B.3.1.6. Tabla br_gold.advanced_player_stats

```
CREATE DATABASE IF NOT EXISTS br_gold
LOCATION '/user/cloudera/br/gold';

DROP TABLE IF EXISTS br_gold.advanced_player_stats;
CREATE TABLE IF NOT EXISTS br_gold.advanced_player_stats (
    player STRING,
    player_csk STRING,
    player_id STRING,

    isplayoff BOOLEAN,

    season INT,
    season_href STRING,
    age INT,
```

```
    team_id STRING,
    team_id_href STRING,
    lg_id STRING,
    lg_id_href STRING,
    pos STRING,
    g INT,
    mp INT,
    per FLOAT,
    ts_pct FLOAT,
    fg3a_per_fga_pct FLOAT,
    fta_per_fga_pct FLOAT,
    orb_pct FLOAT,
    drb_pct FLOAT,
    trb_pct FLOAT,
    ast_pct FLOAT,
    stl_pct FLOAT,
    blk_pct FLOAT,
    tov_pct FLOAT,
    usg_pct FLOAT,
    ows FLOAT,
    dws FLOAT,
    ws FLOAT,
    ws_per_48 FLOAT,
    obpm FLOAT,
    dbpm FLOAT,
    bpm FLOAT,
    vorp FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS PARQUET;
```

B.3.2. Población tablas (INSERT INTO) capa Gold

B.3.2.1. Script Población `br_gold.boxscores_player`

```
set hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE br_gold.boxscores_player PARTITION(season)
SELECT
    p.game_id,
    p.team_id,
    p.box_type,
    p.date,
    p.ishome,
    s.isplayoff,
    p.pnum,
    p.player,
    p.player_href,
    p.player_csk,
    p.player_id,
    p.mp,
    p.sp,
    p.fg,
    p.fga,
    p.fg_pct,
```

```
    p.fg3,
    p.fg3a,
    p.fg3_pct,
    p.ft,
    p.fta,
    p.ft_pct,
    p.orb,
    p.drb,
    p.trb,
    p.ast,
    p.stl,
    p.blk,
    p.tov,
    p.pf,
    p.pts,
    p.plus_minus,
    p.pts / (2 *  (p.fga + 0.44 * p.fta)) AS ts_pct,
    (p.fg + 0.5 * p.fg3) / p.fga AS efg_pct,
    p.fg3a / p.fga AS fg3a_per_fga_pct,
    p.fta / p.fga AS fta_per_fga_pct,
    100 * (p.orb * tm.sp / 300) / (p.sp / 60 * (tm.orb + opp.drb)) AS orb_pct,
    100 * (p.drb * tm.sp / 300) / (p.sp / 60 * (tm.drb + opp.orb)) AS drb_pct,
    100 * (p.trb * tm.sp / 300) / (p.sp / 60 * (tm.trb + opp.trb)) AS trb_pct,
    100 * (p.ast) / ((p.sp / (tm.sp / 5) * tm.fg) - p.fg) AS ast_pct,
    100 * (p.stl * tm.sp / 300) / (p.sp / 60 * ((tm.fga + 0.4 * tm.fta - 1.07 * (tm.orb /
    ↪  (tm.orb + opp.drb)) * (tm.fga - tm.fg) + tm.tov) + (opp.fga + 0.4 * opp.fta - 1.07 *
    ↪  (opp.orb / (opp.orb + tm.drb)) * (opp.fga - opp.fg) + opp.tov))) AS stl_pct,
    100 * (p.blk * tm.sp / 300) / (p.mp / 60 * (opp.fga - opp.fg3a)) AS blk_pct,
    100 * p.tov / (p.fga + 0.44 * p.fta + p.tov) AS tov_pct,
    100 * ((p.fga + 0.44 * p.fta + p.tov) * (tm.sp / 300)) / ((p.sp / 60) * (tm.fga + 0.44 *
    ↪  tm.fta + tm.tov))  AS usg_pct,
    null AS off_rtg,
    null AS def_rtg,
    null as bpm,
    p.reason,
    p.season
FROM
    br_silver.boxscores_basic p
INNER JOIN
    br_silver.boxscores_basic_team_totals tm
ON (p.game_id = tm.game_id
    AND p.team_id = tm.team_id
    AND p.box_type = tm.box_type
    AND p.season = tm.season)
INNER JOIN
    br_silver.boxscores_basic_team_totals opp
ON (p.game_id = opp.game_id
    AND p.box_type = opp.box_type
    AND p.season = opp.season)
INNER JOIN
    br_silver.schedule s
ON (p.game_id = s.game_id)
WHERE
    tm.team_id <> opp.team_id;

set hive.exec.dynamic.partition.mode=strict;
```

B.3.2.2. Script Población `br_gold.boxscores_team`

```
set hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE br_gold.boxscores_team PARTITION(season)
SELECT
    tm.game_id,
    tm.team_id,
    tm.box_type,
    tm.date,
    tm.ishome,
    tm.isplayoff,
    tm.sp,
    tm.fg,
    tm.fga,
    tm.fg_pct,
    tm.fg3,
    tm.fg3a,
    tm.fg3_pct,
    tm.ft,
    tm.fta,
    tm.ft_pct,
    tm.orb,
    tm.drb,
    tm.trb,
    tm.ast,
    tm.stl,
    tm.blk,
    tm.tov,
    tm.pf,
    tm.pts,
    tm.plus_minus,
    tm.pts / (2 *  (tm.fga + 0.44 * tm.fta)) AS ts_pct,
    (tm.fg + 0.5 * tm.fg3) / tm.fga AS efg_pct,
    tm.fg3a / tm.fga AS fg3a_per_fga_pct,
    tm.fta / tm.fga AS fta_per_fga_pct,
    100 * (tm.orb) / ((tm.orb + opp.drb)) AS orb_pct,
    100 * (tm.drb) / ((tm.drb + opp.orb)) AS drb_pct,
    100 * (tm.trb) / ((tm.trb + opp.trb)) AS trb_pct,
    100 * (tm.ast) / (tm.fg) AS ast_pct,
    100 * tm.stl / (0.5 * ((tm.fga + 0.4 * tm.fta - 1.07 * (tm.orb / (tm.orb + opp.drb)) *
    ↪  (tm.fga - tm.fg) + tm.tov) + (opp.fga + 0.4 * opp.fta - 1.07 * (opp.orb / (opp.orb +
    ↪  tm.drb)) * (opp.fga - opp.fg) + opp.tov))) AS stl_pct,
    100 * tm.blk / (opp.fga - opp.fg3a) AS blk_pct,
    100 * tm.tov / (tm.fga + 0.44 * tm.fta + tm.tov) AS tov_pct,
    100 AS usg_pct,
    100 * tm.pts / (0.5 * ((tm.fga + 0.4 * tm.fta - 1.07 * (tm.orb / (tm.orb + opp.drb)) *
    ↪  (tm.fga - tm.fg) + tm.tov) + (opp.fga + 0.4 * opp.fta - 1.07 * (opp.orb / (opp.orb +
    ↪  tm.drb)) * (opp.fga - opp.fg) + opp.tov))) AS off_rtg,
    100 * opp.pts / (0.5 * ((tm.fga + 0.4 * tm.fta - 1.07 * (tm.orb / (tm.orb + opp.drb)) *
    ↪  (tm.fga - tm.fg) + tm.tov) + (opp.fga + 0.4 * opp.fta - 1.07 * (opp.orb / (opp.orb +
    ↪  tm.drb)) * (opp.fga - opp.fg) + opp.tov))) AS def_rtg,
    null as bpm,
    tm.season
FROM
    br_silver.boxscores_basic_team_totals tm
INNER JOIN
    br_silver.boxscores_basic_team_totals opp
```

```
ON (tm.game_id = opp.game_id
    AND tm.box_type = opp.box_type
    AND tm.season = opp.season)
WHERE
    tm.team_id <> opp.team_id;

set hive.exec.dynamic.partition.mode=strict;
```

B.3.2.3. Script Población br_gold.player_list

```
INSERT OVERWRITE TABLE br_gold.player_list
SELECT
    player,
    regexp_extract(player_href, '/players/[a-z]/(.*).html', 1) as player_id,
    year_min,
    year_max,
    pos,
    weight AS weight_lbs,
    weight * 0.453592 AS weight_kg,
    height AS height,
    height_csk AS height_in,
    round(height_csk * 2.54) AS height_cm,
    CONCAT( SUBSTR(birth_date_csk, 1,4), '-', SUBSTR(birth_date_csk, 5,2), '-',
    ↪  SUBSTR(birth_date_csk, 7,2)) AS birth_date,
    regexp_extract(colleges, '<a.*>(.*?)</a></td>', 1)
FROM br_raw.player_list;
```

B.3.2.4. Script Población br_gold.player_stats

```
set hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE br_gold.player_stats PARTITION(season)
SELECT
    p.team_id,
--    p.box_type,
    p.player,
    p.player_href,
    p.player_csk,
    p.player_id,
    SUM(p.sp) / 60 AS mp,
    SUM(p.sp) AS sp,
    SUM(p.fg) AS fg,
    SUM(p.fga) AS fga,
    SUM(p.fg) / SUM(p.fga) AS fg_pct,
    SUM(p.fg3) AS fg3,
    SUM(p.fg3a) AS fg3a,
    SUM(p.fg3) / SUM(p.fg3a) AS fg3_pct,
    SUM(p.ft) AS ft,
    SUM(p.fta) AS fta,
    SUM(p.ft) / SUM(p.fta) AS ft_pct,
    SUM(p.orb) AS orb,
    SUM(p.drb) AS drb,
    SUM(p.trb) AS trb,
```

```
    SUM(p.ast) AS ast,
    SUM(p.stl) AS stl,
    SUM(p.blk) AS blk,
    SUM(p.tov) AS tov,
    SUM(p.pf) AS pf,
    SUM(p.pts) AS pts,
    SUM(p.plus_minus) AS plus_minus,
    SUM(p.pts) / (2 *  (SUM(p.fga) + 0.44 * SUM(p.fta))) AS ts_pct,
    (SUM(p.fg) + 0.5 * SUM(p.fg3)) / SUM(p.fga) AS efg_pct,
    SUM(p.fg3a) / SUM(p.fga) AS fg3a_per_fga_pct,
    SUM(p.fta) / SUM(p.fga) AS fta_per_fga_pct,
    100 * (SUM(p.orb) * SUM(tm.sp) / 300) / (SUM(p.sp) / 60 * (SUM(tm.orb) + SUM(opp.drb)))
    ↪  AS orb_pct,
    100 * (SUM(p.drb) * SUM(tm.sp) / 300) / (SUM(p.sp) / 60 * (SUM(tm.drb) + SUM(opp.orb)))
    ↪  AS drb_pct,
    100 * (SUM(p.trb) * SUM(tm.sp) / 300) / (SUM(p.sp) / 60 * (SUM(tm.trb) + SUM(opp.trb)))
    ↪  AS trb_pct,
    100 * (SUM(p.ast)) / ((SUM(p.sp) / (SUM(tm.sp) / 5) * SUM(tm.fg)) - SUM(p.fg)) AS
    ↪  ast_pct,
    100 * (SUM(p.stl) * SUM(tm.sp) / 300) / (SUM(p.sp) / 60 * ((SUM(tm.fga) + 0.4 *
    ↪  SUM(tm.fta) - 1.07 * (SUM(tm.orb) / (SUM(tm.orb) + SUM(opp.drb))) * (SUM(tm.fga) -
    ↪  SUM(tm.fg)) + SUM(tm.tov)) + (SUM(opp.fga) + 0.4 * SUM(opp.fta) - 1.07 *
    ↪  (SUM(opp.orb) / (SUM(opp.orb) + SUM(tm.drb))) * (SUM(opp.fga) - SUM(opp.fg)) +
    ↪  SUM(opp.tov)))) AS stl_pct,
    100 * (SUM(p.blk) * SUM(tm.sp) / 300) / (SUM(p.sp) / 60 * SUM(opp.fga) * SUM(opp.fg3a))
    ↪  AS blk_pct,
    100 * SUM(p.tov) / (SUM(p.fga) + 0.44 * SUM(p.fta) + SUM(p.tov)) AS tov_pct,
    100 * ((SUM(p.fga) + 0.44 * SUM(p.fta) + SUM(p.tov)) * (SUM(tm.sp) / 300)) / ((SUM(p.sp)
    ↪  / 60) * (SUM(tm.fga) + 0.44 * SUM(tm.fta) + SUM(tm.tov)))  AS usg_pct,
    null AS off_rtg,
    null AS def_rtg,
    null as bpm,
    p.season
FROM
    br_gold.boxscores_player p
INNER JOIN
    br_gold.boxscores_team tm
ON (p.game_id = tm.game_id
    AND p.team_id = tm.team_id
    AND p.box_type = tm.box_type
    AND p.season = tm.season)
INNER JOIN
    br_gold.boxscores_team opp
ON (p.game_id = opp.game_id
    AND p.box_type = opp.box_type
    AND p.season = opp.season)
WHERE
    p.isplayoff = FALSE
AND tm.team_id <> opp.team_id
AND p.box_type = "game"
GROUP BY
    p.season,
    p.team_id,
    p.player,
    p.player_href,
    p.player_csk,
    p.player_id;

set hive.exec.dynamic.partition.mode=strict;
```

B.3.2.5. Script Población br_gold.schedule

```
INSERT OVERWRITE TABLE br_gold.schedule
SELECT * FROM br_silver.schedule;
```

B.3.2.6. Script Población br_gold.advanced_player_stats

```
CREATE DATABASE IF NOT EXISTS br_gold
LOCATION '/user/cloudera/br/gold';

INSERT OVERWRITE TABLE br_gold.advanced_player_stats
SELECT
    p.player,
    p.player_csk,
    p.player_id,

    s.isplayoff,

    s.season,
    s.season_href,
    s.age,
    s.team_id,
    s.team_id_href,
    s.lg_id,
    s.lg_id_href,
    s.pos,
    s.g,
    s.mp,
    s.per,
    s.ts_pct,
    s.fg3a_per_fga_pct,
    s.fta_per_fga_pct,
    s.orb_pct,
    s.drb_pct,
    s.trb_pct,
    s.ast_pct,
    s.stl_pct,
    s.blk_pct,
    s.tov_pct,
    s.usg_pct,
    s.ows,
    s.dws,
    s.ws,
    s.ws_per_48,
    s.obpm,
    s.dbpm,
    s.bpm,
    s.vorp
FROM
    br_silver.advanced_player_stats s,
    (SELECT DISTINCT player, player_csk, player_id
    FROM br_silver.boxscores_basic WHERE length(player_csk) > 3) p
```

```
WHERE
    p.player_id = s.player_id;
```

Bibliografía

[1] Basketball Reference, *Basketball Reference Glossary*, 2020 (accedido el 9 de septiembre de 2020). [Online]. Available: https://www.basketball-reference.com/about/glossary.html

[2] E. Capriolo, D. Wampler, and J. Rutherglen, *Programming Hive*, 1st ed. O'Reilly Media, Inc., 2012.

[3] B. Chambers and M. Zaharia, *Spark: The Definitive Guide Big Data Processing Made Simple*, 1st ed. O'Reilly Media, Inc., 2018.

[4] A. Chong, E. Ch'ng, M. Liu, and B. Li, "Predicting consumer product demands via big data: the roles of online promotional marketing and online reviews," *International Journal of Production Research*, vol. 55, pp. 1–15, 07 2015.

[5] Cloudera, *SQL Differences Between Impala and Hive*, (accedido el 19 de agosto de 2020). [Online]. Available: https://www.ibm.com/blogs/watson/2016/05/biggest-data-challenges-might-not-even-know/

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004, pp. 137–150.

[7] Equipo de desarrollo de Apache Kafka, *Kafka 2.0 Documentation*, 2018 (accedido el 21 de agosto de 2020). [Online]. Available: https://kafka.apache.org/20/documentation.html#introduction

[8] Forbes, *Forbes Releases 21st Annual NBA Team Valuations*, 2019. [Online]. Available: https://www.forbes.com/sites/forbespr/2019/02/06/forbes-releases-21st-annual-nba-team-valuations/

[9] M. Fowler, *Data Lake*, 2 2015 (accedido el 3 de agosto de 2020). [Online]. Available: https://martinfowler.com/bliki/DataLake.html

[10] A. Gates, *Programming Pig*, 1st ed. O'Reilly Media, Inc., 2011.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, p. 29–43, Oct. 2003. [Online]. Available: https://doi.org/10.1145/1165389.945450

[12] Google Cloud, *Cloud Bigtable*, (accedido el 27 de agosto de 2020). [Online]. Available: https://cloud.google.com/bigtable/

[13] A. Gorelik, *The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science*, 1st ed. O'Reilly Media, Inc., 2019.

[14] M. Grover, T. Malaska, J. Seidman, and G. Shapira, *Hadoop Application Architectures*, 1st ed. O'Reilly Media, Inc., 2015.

[15] S. Gupta and V. Giri, *Practical Enterprise Data Lake Insights: Handle Data-Driven Challenges in an Enterprise Big Data Lake*, 1st ed. USA: Apress, 2018.

[16] B. Heintz and D. Lee, *Productionizing Machine Learning with Delta Lake*, 8 2019 (accedido el 13 de agosto de 2020). [Online]. Available: https://databricks.com/blog/2019/08/14/productionizing-machine-learning-with-delta-lake.html

[17] S. Ilardi, *The next big thing: real plus-minus*, 2014. [Online]. Available: https://www.espn.com/nba/story/_/id/10740818/introducing-real-plus-minus

[18] B. Inmon, *Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump*, 1st ed. Denville, NJ, USA: Technics Publications, LLC, 2016.

[19] W. H. Inmon and D. Linstedt, *Data Architecture: A Primer for the Data Scientist Big Data, Data Warehouse and Data Vault*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014.

[20] N. B. Janet Wiener, *Facebook's Top Open Data Problems*, 2014 (accedido el 29 de julio de 2020). [Online]. Available: https://research.fb.com/blog/2014/10/facebook-s-top-open-data-problems/

[21] T. John and P. Misra, *Data Lake for Enterprises: Lambda Architecture for Building Enterprise Data Systems*. Packt Publishing, 2017.

[22] I. Kalbandi and J. Anuradha, "A brief introduction on big data 5vs characteristics and hadoop technology," *Procedia Computer Science*, vol. 48, pp. 319–324, 12 2015.

[23] G. Kalipe and R. Behera, "Big data architectures : A detailed and application oriented review," 10 2019.

[24] J. Koshy and D. Gutiérrez, *A Brief History of Kafka, LinkedIn's Messaging Platform*, 2016 (accedido el 21 de agosto de 2020). [Online]. Available: https://insidebigdata.com/2016/04/28/a-brief-history-of-kafka-linkedins-messaging-platform/

[25] D. Kouzis-Loukas, *Learning Scrapy*. Packt Publishing Ltd, 2016.

[26] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. USA: Manning Publications Co., 2015.

[27] M. Mauboussin, "The success equation: Untangling skill and luck in business, sports, and investing," *CFA Institute Conference Proceedings Quarterly*, vol. 30, pp. 44–51, 09 2013.

[28] H. R. Max Roser and E. Ortiz-Ospina, "Internet," *Our World in Data*, 2015 (accedido el 29 de julio de 2020), https://ourworldindata.org/internet.

[29] R. Mitchell, *Web Scraping with Python: Collecting Data from the Modern Web*, 1st ed. O'Reilly Media, Inc., 2015.

[30] A. C. Murthy, V. K. Vavilapalli, D. Eadline, J. Niemiec, and J. Markham, *Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2*, 1st ed. Addison-Wesley Professional, 2014.

[31] A. Nagdive and R. Tugnayat, "A review of Hadoop ecosystem for Big Data," 01 2018.

[32] N. Narkhede, G. Shapira, and T. Palino, *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale*, 1st ed. O'Reilly Media, Inc., 2017.

[33] M. Ozaniaan, *The Stadium Revenue Each NFL Team Will Lose If Games Are Played Without Fans*, 2020. [Online]. Available: https://www.forbes.com/sites/mikeozanian/2020/05/18/the-stadium-revenue-each-nfl-team-will-lose-if-games-are-played-without-fans

[34] J. Pasqua, *Schema-on-Read vs Schema-on-Write*, 2014 (accedido el 3 de agosto de 2020). [Online]. Available: https://www.marklogic.com/blog/schema-on-read-vs-schema-on-write/

[35] I. Puzyrevskiy, E. Riabenko, E. Dral, A. A. Dral, and P. Mezentsev, *Big Data Essentials: HDFS, MapReduce and Spark RDD.*, 2018. [Online]. Available: https://www.coursera.org/learn/big-data-essentials(accedidoel14deagostode2020)

[36] S. Rangarajan, *Data Warehouse Design – Inmon versus Kimball*, 2016 (accedido el 15 de septiembre de 2020). [Online]. Available: https://tdan.com/data-warehouse-design-inmon-versus-kimball/20300/

[37] F. Ravat and Y. Zhao, "Data Lakes: Trends and Perspectives," in *International Conference on Database and Expert Systems Applications (DEXA 2019)*, vol. 1, no. 11706, Linz, Austria, Aug. 2019, pp. 304–313. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02397457

[38] J. Russell, *Getting Started with Impala: Interactive SQL for Apache Hadoop.* O'Reilly Media, Inc., 2014.

[39] S. Sagiroglu and D. Sinanc, "Big data: A review," in *2013 International Conference on Collaboration Technologies and Systems (CTS)*, 2013, pp. 42–47.

[40] E. Schadt and S. Chilukuri, *The role of big data in medicine*, 11 2015 (accedido el 6 de septiembre de 2020). [Online]. Available: https://www.mckinsey.com/industries/pharmaceuticals-and-medical-products/our-insights/the-role-of-big-data-in-medicine

[41] C. Schneider, *The biggest data challenges that you might not even know you have*, 5 2016 (accedido el 29 de julio de 2020). [Online]. Available: https://www.ibm.com/blogs/watson/2016/05/biggest-data-challenges-might-not-even-know/

[42] R. P. Schumaker, O. K. Solieman, and H. Chen, *Sports Data Mining*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[43] A. W. Services, *Amazon Managed Streaming for Apache Kafka (Amazon MSK)*, (accedido el 21 de agosto de 2020. [Online]. Available: https://aws.amazon.com/msk/what-is-kafka/

[44] I. Sommerville, *Software Engineering*, 9th ed. USA: Addison-Wesley Publishing Company, 2010.

[45] B. Stopford, *Designing Event-Driven Systems*. O'Reilly Media, Inc., 2018.

[46] D. Sullivan, *NoSQL for Mere Mortals*, 1st ed. Addison-Wesley Professional, 2015.

[47] L. Sulmont, *Data Lakes vs. Data Warehouses*, 1 2020 (accedido el 3 de agosto de 2020). [Online]. Available: https://www.datacamp.com/community/blog/data-lakes-vs-data-warehouses

[48] K. Ting and J. J. Cecho, *Apache Sqoop Cookbook*. O'Reilly Media, 2013.

[49] T. White, *Hadoop: The Definitive Guide*, 4th ed. O'Reilly Media, Inc., 2015.