UNIVERSIDAD INTERNACIONAL SEK

DIGITAL SCHOOL

Trabajo de fin de carrera titulado:

Implementación de un Data Lake de los datos de uso del LMS Canvas de la Universidad Internacional SEK.

Realizado por:

Verónica Estefanía Beltrán García

Directora del proyecto:

PhD. Silvia Diana Martínez Mosquera

Como requisito para la obtención del título de

MAGÍSTER EN SISTEMAS DE INFORMACIÓN MENCIÓN EN DATA SCIENCE

Quito, abril 2022

DECLARACION JURAMENTADA

Yo, **Verónica Estefanía Beltrán García**, con cédula de identidad **1721528923**, declaro bajo juramento que el trabajo aquí desarrollado es de mi autoría, que no ha sido previamente presentado para ningún grado a calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración, cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la UNIVERSIDAD INTERNACIONAL SEK, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normativa institucional vigente.

Verónica Estefanía Beltrán García

C.C: 1721528923

DECLARATORIA

El presente trabajo de investigación titulado:

“IMPLEMENTACIÓN DE UN DATA LAKE DE LOS DATOS DE USO DEL LMS CANVAS DE LA UNIVERSIDAD INTERNACIONAL SEK”

Realizado por:
VERÓNICA ESTEFANÍA BELTRÁN GARCÍA

Como requisito para la Obtención del Título de:
MÁSTER EN SISTEMAS DE INFORMACIÓN MENCIÓN EN DATA SCIENCE

Ha sido dirigido por la docente
PhD. SILVIA DIANA MARTÍNEZ MOSQUERA

Quien considera que constituye un trabajo original de su autor

Firmado electrónicamente por:
SILVIA DIANA MARTINEZ MOSQUERA

PhD. SILVIA DIANA MARTÍNEZ MOSQUERA
DIRECTORA

PROFESORES INFORMANTES

Después de revisar el trabajo presentado, lo ha calificado como apto para su defensa oral ante el tribunal examinador.

Firmado electrónicamente por:
DIEGO FERNANDO
RIOFRIO LUZCANDO

PhD. Diego Riofrío

JOE LUIS CARRION JUMBO
Firmado digitalmente por JOE LUIS CARRION JUMBO
Fecha: 2022.04.12 10:47:22 -05'00'

PhD. Joe Carrión

Quito, abril 2022

DEDICATORIA

Dedico este trabajo de investigación a Dios, a mis padres Mónica y Manolo, a mi novio Juan Carlos Moya, a mis hermanos, a mis hermosas mascotas Piru, Pata Blanca, Bastet y Thot. Y de manera especial a dos personas maravillosas que ahora son mis ángeles, quienes estarían muy felices de compartir este logro conmigo; mi abuelita Carmita y mi madrina Eunicita.

AGRADECIMIENTO

Agradezco especialmente a Dios, por su infinito amor, por darme la sabiduría y los medios para cumplir esta meta y todos los propósitos de mi vida.

A mi novio Juan Carlos Moya, por su apoyo incondicional, por compartir sus conocimientos conmigo y animarme siempre a ser mejor.

A mi madre, por darme su bendición, por acompañarme en todo momento, por su constancia y amor. A mi padre, por enseñarme a perseguir y conquistar mis sueños. A mis hermanos, por su cariño sincero y por siempre confiar en mí.

Agradezco a mis hermosas mascotas, por llenarme de energía cada día cuando las fuerzas me faltan, e inspirarme con el deseo de que mis conocimientos aporten al desarrollo de una sociedad donde se respeten los derechos de cada ser vivo.

Agradezco a la Universidad Internacional SEK por permitirme implementar este trabajo, a mis docentes, en especial a mi tutora la Mg. Silvia Martínez quien me ha guiado compartiendo sus conocimientos en el desarrollo de esta investigación.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

RESUMEN

Los datos se han convertido en las nuevas minas de oro, todo en el mundo actual gira en torno a estos. A lo largo de la historia se han usado diversos medios de almacenamiento de datos: libretas físicas, diskettes, casetes, unidades de memoria, discos duros, entre otros. Sin embargo, el ritmo de crecimiento de los datos se ha elevado exponencialmente, lo cual ha generado la necesidad de cambiar la forma tradicional en la cual se los almacenaba anteriormente. Las industrias hoy cuentan con sus propias estructuras y bases de datos con gran capacidad de almacenamiento y de diferente tipo; no obstante, el mismo crecimiento de los datos ha demandado la importancia de suplir necesidades del negocio de manera flexible y rápida, lo cual ha dado paso a la incursión del *Big Data* y con ello de la mano la aplicación de *Data Lakes* los cuales se pueden implementar en diversas ramas como la educación, salud, finanzas entre otras.

En este estudio se propone la creación e implementación de un *Data Lake* con los datos del LMS de Canvas de la Universidad Internacional SEK, el cual permita la posibilidad de almacenarlos en un único repositorio con mayor flexibilidad, sin procesarlos previamente y con la finalidad de utilizar los mismo para realizar *análisis* de datos, aprendizaje automático, entre otras aplicaciones.

De manera general, en el presente trabajo se aplicaron los principios del *Big Data* con la finalidad de que los datos que actualmente se almacenan en el Sistema de Gestión del Aprendizaje (Canvas), se carguen íntegros a un directorio HDFS creado en un entorno de *Hadoop* para almacenarlos. Para ello se utilizó el API de conexión de Canvas por medio de *Python* con el objetivo de descargar las tablas del último *dump*, además se generaron scripts para realizar la creación de: directorios en HDFS, tablas en *Hive* e ingesta de los datos descargadas de manera automática; finalmente, se generó una tarea programada para que los procesos descritos se ejecuten a diario. Finalmente, como resultado del trabajo realizado la UISEK tiene implementado un lago de datos, el cual se actualiza diariamente con la información descargada del LMS de Canvas.

Palabras clave: Big Data, HDFS, Data Lake, Cloudera, Hadoop, Hive, HQL, API, UISEK, volcado de datos, esquema de datos, Canvas, educación, datos.

ABSTRACT

Data has become new gold mines, and today everything turns around them. Throughout history, many data storages media have been used: notebooks, diskettes, cassettes, memory units, hard drives, among others. However, the rate of data growth has increased exponentially, which has generated the need to change the traditional way of data storage. Today industries have their structures and databases with large storage capacity and different types. However, the same data growth has demanded the importance of supplying business needs in a flexible and fast way, which has given way to Big Data's incursion and with it Data Lakes's implementation, which can be performed in many branches such as education, health, finance, and others.

This study proposes the creation and implementation of a Data Lake of SEK International University's Canvas data, which allows the possibility of storing these data in a single repository with greater flexibility, without previously data processing and to use them for data analytics, machine learning, among other applications.

In general, in this work Big Data's principles were applied so that the data currently stored in Canvas are loaded integral to the HDFS directory created in the Hadoop environment to store them, for this purpose the Canvas connection API written in Python, was used to download the tables of the last dump, scripts were generated for creation of all directories in HDFS, creation of schema's tables and data ingestion be automatic; finally, a scheduled task was generated so that the described processes are executed daily.

Keywords: Big Data, HDFS, Data Lake, Cloudera, Hadoop, Hive, HQL, API, UISEK, dump, data schemas, Canvas, education, data.

CAPÍTULO 1

INTRODUCCIÓN

La estructura de este trabajo de investigación está distribuida de la siguiente manera:

En el Capítulo 1 se identifica el problema de investigación, así como la justificación de este. Además, se define los objetivos que se pretende conseguir con el desarrollo del presente trabajo de investigación y se estructura el Marco Teórico para que el lector conozca los principales conceptos utilizados. En el Capítulo 2 se desarrolla el Estado del Arte, con el fin de plasmar algunos estudios sobre el impacto y beneficios de aplicar *Big Data* con datos que corresponden a educación.

En el Capítulo 3 se describe el marco general de investigación, con la metodología "*Design Research*", la cual se basa en diseñar o desarrollar un producto para brindar la solución a un problema; además en este capítulo se detalla las fases de Exploración, Preparación e Ingesta de Datos. En el Capítulo 4 se muestran los resultados que se obtuvieron luego de implementar el *Data Lake* con los datos de Canvas de la Universidad Internacional SEK.

Finalmente, en el Capítulo 5 se lista un grupo de conclusiones, las cuales se obtuvieron del presente trabajo de investigación, así como la propuesta de futuros trabajos que hagan uso del *Data Lake* implementado, se incluye también una sección de referencias bibliográficas utilizadas para desarrollar el presente documento de investigación y se incluye el código de programación generado para el desarrollo e implementación del *Data Lake*.

1.1 Planteamiento del problema

La Universidad Internacional SEK hace uso del LMS de Canvas, este Sistema de Gestión del Aprendizaje cuenta con un diccionario de datos que se rige por las típicas convenciones para almacenarlos, sus registros se transforman en tablas de hechos y dimensiones mismas que se encuentran disponibles en archivos planos y se adhieren a un esquema de estrella. (Instructure n.d.), lo cual no es flexible a cambios y hace uso de procesos ETL (Extracción - Transformación - Carga).

A lo largo de los años las características de los datos se han ido volviendo más complejas y su volumen fue en aumento, a partir de ello nació la necesidad de almacenarlos sin importar su tipo, tamaño o fuente; con la finalidad de extraer su valor transformándolos en información *(Baker, 2017)*. Este crecimiento exponencial de los datos ha permitido que diversas áreas de negocio ejecuten procesos de análisis que los transformen en información relevante para toma de decisiones y valor en el mercado según su área respectiva. (Chaurasia et al. 2018)

Por otro lado, la educación no se puede quedar atrás de las nuevas tendencias para transformar la manera en la que se almacenan y manejan los datos, pues existe la posibilidad de aplicar los principios de *Big Data* en esta área, con el objetivo de contar con grandes almacenes de datos, los cuales sean un insumo para desarrollar proyectos de análisis del aprendizaje que generen mejoras en los procesos educativos, por esta necesidad en la educación se creó el concepto de *Educational Data Mining (EDM)*, mismo que se encarga de explorar el comportamiento del aprendizaje que tienen los estudiantes en entornos de aprendizaje en línea, a través de la exploración de datos (Mohamad and Tasir, 2013).

Con base a las nuevas necesidades que genera el *Educational Data Mining* en relación al aprendizaje, nació el *Learning Analytics,* el cual es una medición, recopilación, análisis e información de datos generados por los estudiantes, además de ser una alternativa para que los profesores puedan entender el proceso de aprendizaje en la educación basándose en el aumento de datos (Clow. 2013).

Por esta razón, se plantea realizar la implementación de un *Data Lake* con los datos del LMS de Canvas de la Universidad Internacional SEK para almacenar datos de tipo estructurado, no estructurado y semi estructurado en un directorio HDFS (*Hadoop Distributed File System*) (Apache Software, 2021) haciendo uso del proceso ELT (Extracción – Carga – Transformación), lo cual permitirá adaptarse fácilmente a cambios utilizando los datos de estudiantes, docentes, carreras, y demás información almacenada en Canvas para realizar procesos de *Learning Analytics*, *Educational Data Mining*, entre otros.

1.2 Justificación

En el transcurso del tiempo se ha intentado mejorar los procesos utilizados para el almacenamiento y uso de los datos para ofrecer mayor flexibilidad y usabilidad a menor costo y tiempo. Siguiendo esta tendencia, los datos que se almacenan en las plataformas académicas requieren ser procesados para aplicar analítica, inteligencia artificial, *deep learning*, reportería, entre otras aplicaciones sin discriminar su tipo. (Black & Wiliam, 2018).

Los *Data Lakes* se utilizan generalmente para manejo de grandes volúmenes de información en las diferentes ramas, entre ellas, la educación (Baig, Shuib, and Yadegaridehkordi 2020; Li and Zhai 2018), generando la posibilidad de que mejore y sea personalizada, fortaleciendo el conocimiento adquirido de cada estudiante; así como la posibilidad de obtener el *feedback* de maestros y alumnos en tiempo real para actuar oportunamente, y compartir el conocimiento adquirido por cada alumno; pues permite mejorar tanto el aprendizaje como la enseñanza. (Black & Wiliam, 2018).

Como indica Juliá Minguillón (2018), responsable del área de investigación del *E-Learn Center* de la Universidad Oberta de Catalunya (UOC) *"los datos aportan a los docentes mucha información sobre sus alumnos".* Mediante el análisis de los datos, un profesor podría revelar si sus alumnos avanzan de manera adecuada, y plantear una solución en tiempo real. Según el autor, esto "nos permite caminar hacia una personalización del proceso de aprendizaje" (Telefónica Educación Digital n.d.).

1.3 Objetivos

1.3.1 Objetivo General

Crear un *Data Lake* con los datos del LMS de Canvas de la Universidad Internacional SEK, con información del último volcado de datos de Canvas, mediante el uso y aplicación de herramientas del ecosistema de *Hadoop*, que alimente y actualice constantemente a un único repositorio, y que permita realizar procesos de analítica de datos, aprendizaje automático, entre otras aplicaciones.

1.3.2 Objetivos Específicos

- Diseñar la arquitectura del ecosistema *Big Data* para el almacenamiento del contenido educativo en el *Data Lake* con las herramientas del entorno *Hadoop*.
- Implementar la arquitectura diseñada para transferencia y almacenamiento de los datos de Canvas en el *Data Lake* para su posterior análisis.
- Aplicar los principios de *Big Data* para la creación de una solución automática que permita el almacenamiento de los archivos correspondientes de cada tabla del último *dump* de Canvas en el *Data Lake*.
- Implementar en *Hive* una base de datos, donde se almacene la información de cada tabla de Canvas en un formato estructurado.
- Analizar los resultados obtenidos en la implementación del ecosistema de *Big Data* propuesto, a través del entorno gráfico de Cloudera HUE.

1.4 Marco teórico

En esta sección se presentan las principales herramientas y términos que se utilizaron para el diseño e implementación de un *Data Lake* con los datos del LMS de Canvas de la Universidad Internacional SEK, las cuales se mencionan a lo largo del presente trabajo de investigación.

1.4.1 Python

Es un lenguaje de programación orientado a objetos, de fácil entendimiento, y potente, similar a *Perl*, *Ruby* o *Java*. Actualmente se encuentra entre los lenguajes preferidos por los usuarios. (Python, 2019)

Características de *Python*:

- Variedad de tipos de datos disponibles.
- Permite agrupar el código en módulos y paquetes.
- Admite la generación y captura de excepciones.
- Brinda al usuario funciones de programación avanzadas.
- Su sintaxis es ordenada y entendible.

1.4.2 Big Data

Es el término empleado para agrupar conjuntos de datos complejos y grandes, que con sistemas de *software* convencionales no se pueden procesar ni analizar, también es el término que se utiliza para definir la cantidad de transacciones de datos que se generan en redes sociales y dispositivos móviles. (Baker, 2017). Además, Big Data permite realizar el análisis de grandes volúmenes de datos (*Big Data Analytics*).

Big Data Analytics reside en el proceso de exploración y extracción de información útil. Se puede clasificar en categorías: (1) Procesamiento por lotes; (2) Procesamiento de flujo; (3) OLTP (Online Transaction Processing) e; (4) Interactivo consultas y análisis ad-hoc. (Ribeiro, Silva, and da Silva 2015). En la Figura 1 se muestran las 4 V`s del *Big Data*: Variedad (varios tipos de información), Volumen (almacena gran cantidad de datos), Veracidad (nivel de confiabilidad de la información), Velocidad (velocidad con la que se genera la información).

Figura 1 Las 4 V`s del BigData (Apache Software 2021) (iLifebelt, 2022)

1.4.3 Lago de Datos (*Data Lake*)

Un *Data Lake* es un repositorio centralizado que se usa para almacenar datos de tipo estructurado y no estructurado a cualquier escala. Permite realizar análisis desde pequeñas cantidades de datos hasta procesamiento de *Big Data* por lo cual también se lo utiliza para hacer uso de aprendizaje automático. (Amazon Web Services, 2022). Además permite almacenar conjuntos de datos sin procesar previamente, en su formato original de manera general (Red Hat, 2022). El *Data Lake* faculta la opción de almacenar casi en tiempo real los datos, es decir, contiene los datos nativos que vienen de su fuente. (Miloslavskaya and Tolstoy, 2016).

1.4.4 Hadoop

Apache *Hadoop* es un *framework* que se usa para el procesamiento de grandes conjuntos de datos en *clústeres* de computadoras haciendo uso de modelos de programación simples (*MapReduce*). Se encuentra diseñado para escalar de servidores individuales a varias máquinas. De esta manera para brindar alta disponibilidad ya no se depende únicamente del *hardware*, pues su biblioteca está en la capacidad de detectar fallas y manejar estas fallas en la capa de aplicación, lo cual permite que se pueda contar alta disponibilidad en la parte superior de un grupo de computadoras. (Apache Software 2021). Además cuenta con la interfaz de usuario web para la gestión de *Hadoop* (HUE), la cual es también un asistente SQL de código abierto para bases y almacenes de datos (Cloudera and Hue, 2022), en esta interfaz se puede trabajar con *Hive*, *Impala*, entre otras herramientas del entorno de *Hadoop*.

La Figura 2 presenta el Ecosistema de *Hadoop*, donde las principales distribuciones son HDFS y *Yarn*; se puede observar que alrededor de ellas existe un conjunto de tecnologías que cumplen funciones específicas acorde a las necesidades de los usuarios, por ejemplo, algunas se usan para realizar la ingesta de datos, otro grupo se usa para realizar el procesamiento de datos, otras se usan para realizar la búsqueda de datos, y también se cuenta con bases de datos y con HUE.

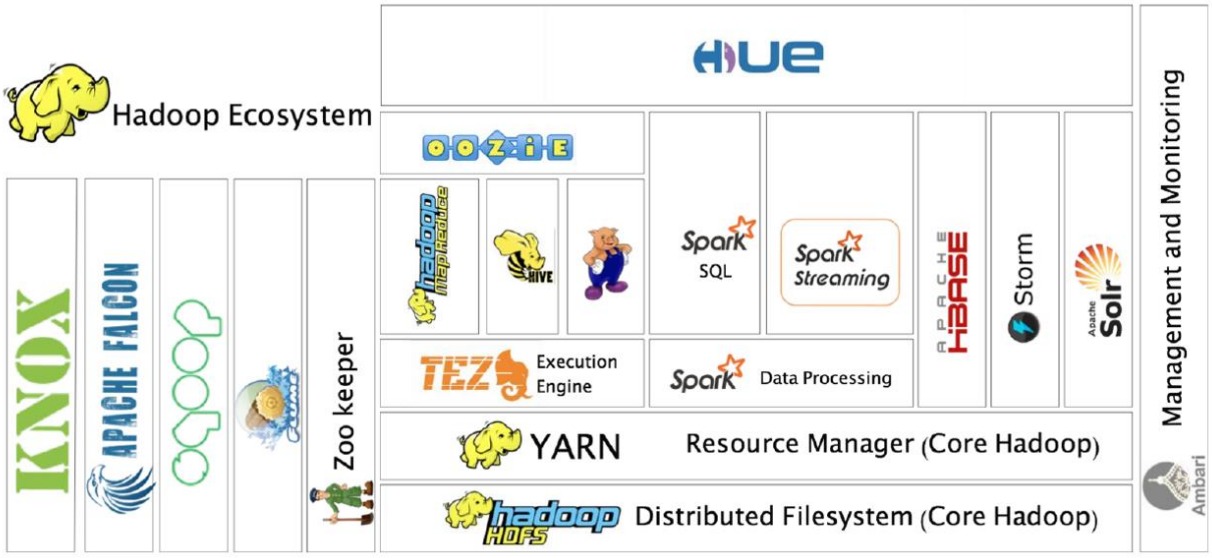


Figura 2 Ecosistema de Hadoop (López Taboada and Casal 2021)

1.4.5 Cloudera

Cloudera se divide prácticamente en dos plataformas, *Cloudera Data Plataform* (CDP), y Cloudera CDH, las cuales tiene objetivos específicos para satisfacer la necesidad de uso de sus clientes.

Cloudera Data Plataform (CDP), es una plataforma que gestiona y asegura el ciclo de vida de los datos en nubes públicas y en la nube privada, sirve para conectar entornos locales con las nubes públicas para una experiencia de nube híbrida. (Inc. Cloudera, 2021).

Por otro lado, Cloudera CDH, es la plataforma de distribución de código abierto de *Hadoop*, se usa en soluciones empresariales, para ello cuenta con un sistema que permite realizar flujos de trabajo de *Big Data* de extremo a extremo. (Cloudera and Apache Hadoop, 2021).

Esta plataforma de distribución de *Big Data* basada en Apache *Hadoop*, cuenta con más de 50 componentes *open source*, que se pueden usar según las necesidades del cliente. Entre las principales herramientas se tiene: *Hue (Hadoop User Experience*), *Apache Sqoop* (*SQL to Hadoop*), *Apache Zookeeper* (coordinador de trabajos de *Hadoop* y servicios de *Big Data*), *Apache Hive* (permite realizar consultas de los datos almacenados en HDFS), *Apache Impala* (motor de consultas SQL), *Apache Oozie* (planificador de *workflows*), *Apache Hbase* (base de datos no relacional), *Apache Hadoop* (incluye HDFS, *Yarn* y *MapReduce*), *Apache Spark* (*framekork* para desarrollo de aplicaciones paralelas), *Apache Kafla* (Sistema de intermediación de mensajes), las mencionadas herramientas permiten crear *Data Lakes*, *Data Warehousing*, *Machine Learning* y Analítica de datos (Fernández, 2021c).

1.4.6 Arquitectura de Hadoop

La arquitectura y diseño de *Hadoop* se basa en la premisa de que el procesamiento de datos es cada vez más fácil, rápido y eficiente y que estos datos son capaces de producir alta latencia y congestión en la red. Para ello *Hadoop* cuenta con sus principales componentes: *MapReduce (paradigma de procesamiento de datos)*, *Yarn (tecnología de gestión de recursos y programación de trabajos)* y HDFS. El sistema de ficheros distribuido de *Hadoop* (HDFS) permite que las aplicaciones tengan la capacidad de acceder a los datos en el lugar donde se encuentren almacenados. (Fernández, 2021a). En la Figura 3 se presenta la Arquitectura de *Hadoop* a alto nivel.

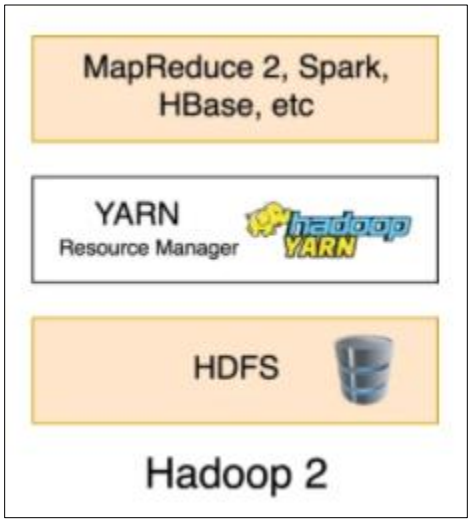


Figura 3 Arquitectura Apache Hadoop (Fernández, 2021a)

1.4.7 HDFS

Es el Sistema de Archivos Distribuido de *Hadoop*, tiene semejanza con otros sistemas de archivos distribuidos. Sin embargo, HDFS puede ser implementado en máquinas con *software* de gama media y es tolerante a fallas. Inicialmente HDFS es parte del proyecto *Apache Hadoop Core*. (Apache Software, 2021).

HDFS es un sistema escalable, cuya arquitectura es distribuida y puede almacenar inclusive 100 TB en un solo archivo. Además, tiene los siguientes objetivos generales: 1. Procesar archivos de grandes tamaños (gigabytes hasta en petabytes), 2. Lectura de datos a gran velocidad, 3. Ejecutarse en máquinas con hardware básico, intermedio y avanzado; dependiendo la cantidad y tamaño de datos que se requiera almacenar. (Camargo-Vega et al. 2015).

1.4.8 Apache Hive

Es un software que permite la lectura, escritura y administración de grandes volúmenes de datos que se encuentran en almacenamiento distribuido y se consultan a través de HQL.(Confluence Administrator, 2020).

Características de *Hive*:

- Permitir acceder a los datos mediante HQL, por lo cual permite realizar tareas extracción / transformación / cargar (ETL) y además análisis de datos.
- Permite acceder a archivos almacenados en Apache HDFS.
- Permite recuperar consultas en milisegundos mediante *Hive* LLAP (Live Long And Process), Apache *YARN* y Apache *Slider*.

Hive tiene un lenguaje de consulta HiveQL o HQL, el cual es un lenguaje de SQL, que no sigue el estándar ANSI SQL, pero, es muy similar. (Fernández, 2021b).

1.4.9 Canvas

Canvas es una plataforma LMS (*learning management system*) para estudiantes, se encuentra en un ecosistema confiable, escalable y abierto a la comunidad a través de blogs como fuente de apoyo para que puedan usarlo docentes, alumnos y demás usuarios. La configuración de la cuenta de Canvas, incluye el enlace del portal de datos, este enlace permite al administrador acceder a los datos de Canvas mediante el uso de una API. (Instructure, 2021). Cuenta con un entorno de aprendizaje en línea ideal para educadores, estudiantes e instituciones educativas pues tiene varias características para dinamizar sus cursos, por ejemplo: Rúbricas, Módulos, Calendarios, Cronogramas, Pruebas, *Syllabus*, Análisis, etc.

A nivel de plataforma, Canvas tiene un servidor web al cual se accede haciendo uso de un navegador o aplicación web dinámica creada en una arquitectura nativa de la nube capaz de escalar automáticamente a llegar a decenas de millones de usuarios y ofrece herramientas para administrar usuarios y cursos, con diversas funcionalidades. (Arsys, 2017).

En la Figura 4 se muestra la arquitectura y flujo de trabajo de Canvas, la cual recibe los datos generados por la iteración de los administradores del sistema académico, estudiantes, profesores, personal de apoyo y soporte con la aplicación web de Canvas a través del uso de computadores, celulares, etc. Canvas replica los datos generados casi en tiempo real y estos se respaldan diariamente de forma redundante en múltiples centros de datos y ubicaciones geográficas mediante el uso de Amazon S3 (*Amazon Simple Storage Service*), todos los objetos dentro de los depósitos S3 se cifran y se replican para tener el control de versiones con la finalidad de que estas se puedan restaurar con un esfuerzo mínimo. Además, por seguridad *Instructure* crea copias de la base de datos fuera del sitio de los datos (presentaciones de los estudiantes, contenido creado por los estudiantes, análisis, rúbricas, resultados del aprendizaje y metadatos).

Los servidores de aplicaciones se supervisan constantemente de forma individual para obtener información sobre la carga y la capacidad y cuando estos alcanzan un determinado umbral de carga, se implementa automáticamente un nuevo servidor de aplicaciones. Por otro lado, los servidores de caché se supervisan

constantemente y cuando alguno de estos servidores falla, se implementa otro para ocupar su lugar. Con relación a las bases de datos, estas se supervisan constantemente para determinar el uso de recursos y el tiempo de respuesta; cuando una base de dato se acerca a la carga máxima, los clientes se reubican en clústeres con capacidad disponible. (I. Instructure, 2021)

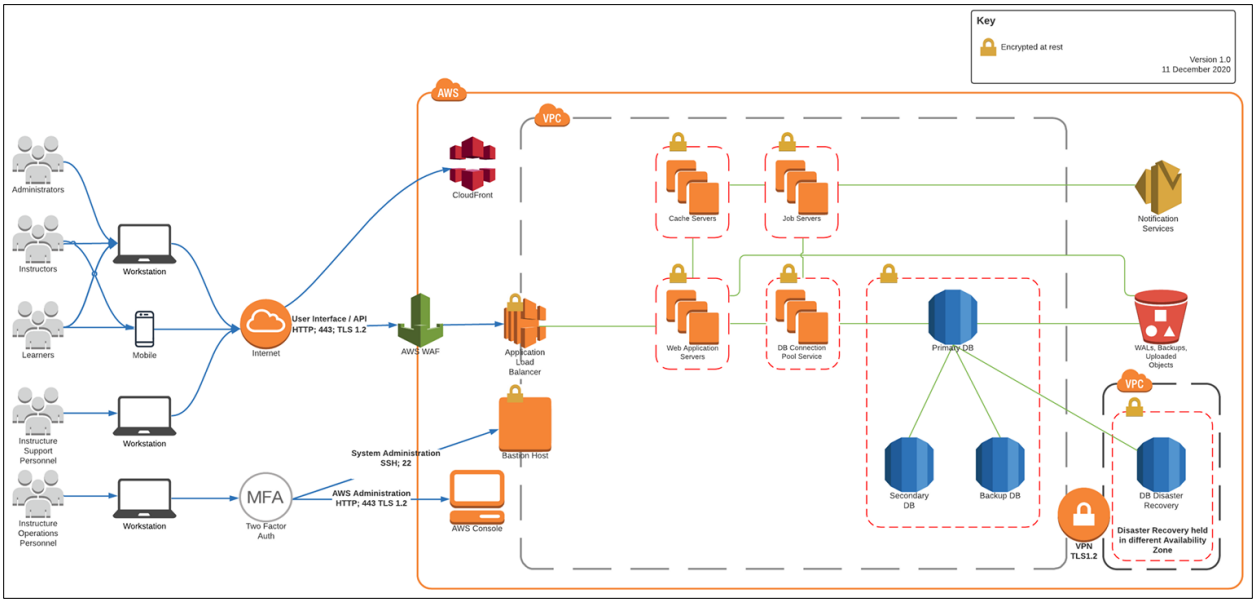


Figura 4 Canvas Architecture and Data Flow (I. Instructure, 2021)

En lo referente a datos, Canvas cuenta con la estructura *Data Warehouse*, la cual está compuesta por las tablas donde se almacena los datos en un esquema estrella, lo que representa que las relaciones deben estar combinados con una distancia igual a la unidad. Estas tablas se encuentran disponibles para descarga como archivos planos con los cuales se puede realizar gran variedad de consultas analíticas. (Instructure n.d.).

CAPÍTULO 2

ESTADO DEL ARTE

Existen varios estudios sobre el impacto y beneficios de aplicar *Big Data* en los datos que corresponden a educación, cómo permiten estos procesos analizar comportamientos, medir niveles de educación y aprendizaje, así como aplicar procesos de mejora en el almacenamiento y uso que se les da a los datos generados por estudiantes y docentes.

Para realizar el presente trabajo se aplicó una investigación aplicada, la cual busca resolver situaciones reales para usar los datos generados en el Sistema de Gestión Académica CANVAS de la Universidad Internacional SEK, y almacenarlos en un *Data Lake* implementado mediante el uso de herramientas de *Big Data.*

En este capítulo se mencionan algunos trabajos de investigación relevantes, los cuales se seleccionaron por su relevancia al estar relacionados con el tema de interés de este proyecto de investigación.

2.1 Estudios de Big Data relacionados a la educación superior

Baig et al., (Baig et al. 2020) mencionan que su estudio tiene como fin cumplir con una revisión sistemática de *Big Data* en la educación para identificar trabajos futuros. En esta investigación se revisaron los temas, subtemas y metodologías de *Big Data* en el ámbito de la educación, los cuales presentan una descripción completa de la literatura existente sobre el tema y puede ayudar a que los investigadores se centren en la combinación de diferentes temas para descubrir nueva información sobre cómo *Big Data* puede aportar para que el proceso de aprendizaje y enseñanza en la educación mejore. Como resultado de su estudio, los autores concluyen que los temas identificados pueden brindar una nueva visión a las universidades para planificar sus programas de aprendizaje, donde se combine la educación tradicional y la educación web. Además, puede ser útil para que los maestros comprendan la forma de medir a los estudiantes y comprender las tendencias que *Big Data* tiene en la educación.

Li y Zhai (Li and Zhai, 2018), mencionan en su artículo que el uso de *Big Data* en la educación se refiere a tres elementos los cuales permiten tomar decisiones en línea: el aprendizaje, el análisis y la minería de datos. Su función principal es estudiar y aplicar análisis de predicciones, análisis de comportamiento y análisis de estudios académicos. Indican también que existe una gran cantidad de datos generados por estudiantes a partir de su proceso de aprendizaje, comportamiento explícito y comportamiento implícito; el comportamiento implícito contiene publicaciones en foros, actividades extracurriculares y actividades sociales en línea, las cuales no se evalúan como actividades académicas. Dentro de los datos generados a partir del proceso de aprendizaje se incluyen los puntajes de pruebas, el estado de tareas y el desempeño del estudiante en el aula.

Los autores comparan el modelo clásico de *eLearning SCORM* que solo puede registrar las limitaciones del proceso de lectura del material didáctico con un nuevo modelo de Sistema Educativo haciendo uso de *Big Data*, cuyo objetivo es capturar y registrar a los alumnos en diferentes actividades de aprendizaje, donde se incluye también el aprendizaje móvil, simulaciones, juegos, actividades en el mundo real, aprendizaje experiencial, aprendizaje social y colaborativo. La lógica de este sistema educativo funciona de la siguiente manera: cuando se necesita registrar una actividad de aprendizaje, el protocolo del sistema educativo con *Big Data* formula una biblioteca de registros de aprendizaje LRS (*Learning Store Record*) en forma de "sujeto + verbo + palabra destinataria", así registra la actividad de aprendizaje y almacena todas las ocurrencias generadas, luego el registrador de aprendizaje permite compartir estos datos con otros almacenes de aprendizaje independientes o en el mismo sistema de gestión del aprendizaje.

A continuación, la Figura 5 presenta el diagrama de flujo del sistema educativo con *Big Data* planteado por Li y Zhai (Li and Zhai, 2018), este diagrama muestra que los datos se derivan del comportamiento de los estudiantes, como las tareas y los exámenes, así como su comportamiento las redes sociales, las actividades extracurriculares, las publicaciones en el foro y otras actividades que no se evalúan. Estos datos se almacenan y se analizan para brindar a los maestros un *feedback* de los estudiantes, lo cual permite que los maestros y las escuelas creen oportunidades educativas que se ajusten a las necesidades y habilidades de cada estudiante.

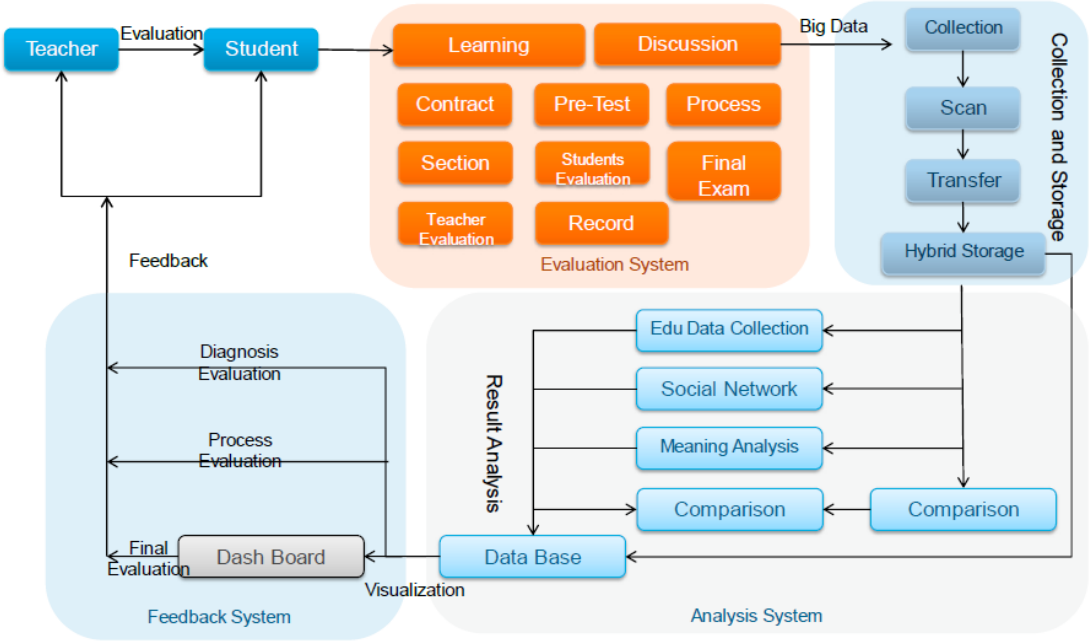


Figura 5 Diagrama de Flujo del Sistema de Análisis (Li and Zhai, 2018)

2.2 Estudios de análisis del proceso de enseñanza-aprendizaje relacionados a la educación superior

Dhankhar y Solanki (Dhankhar and Solanki, 2020), indican que para su estudio se recopiló información que respecta al análisis de aprendizaje e indican que los beneficios de usar la analítica del aprendizaje en educación, pueden colocar a las instituciones en la cumbre a nivel educativo. Para su trabajo se categorizó estudios de caso, investigaciones empíricas y proyectos de análisis de aprendizaje; y los resumió de acuerdo con las instituciones educativas que ya han implementado análisis de aprendizaje en el mundo. Siendo así se tomó como muestra el trabajo realizado en Europa, donde se indica que en ese continente se ha dado un aumento en la investigación con análisis de aprendizaje y se mencionan países como: Holanda, Reino Unido, Alemania, España, Austria.

Además, indican que Holanda, Noruega y Dinamarca están trabajando actualmente en el desarrollo de estrategias de análisis de aprendizaje a nivel nacional que abarcan la generación de políticas, infraestructura y centros de competencia.

2.3 Estudios presentan trabajos de Big Data y análisis del proceso de enseñanza-aprendizaje de en la educación superior

Chaurasia et al., (Chaurasia et al. 2018) mencionan que su estudio analiza de manera general cada componente arquitectónico de Big Data, así como una revisión de beneficios del análisis de *Big Data* y su

valor comercial, así también este estudio contribuye a la conceptualización del valor empresarial con *Big Data* en la educación superior. El modelo permitió analizar la Educación Superior y capturar las relaciones, aplicaciones y valor comercial relacionado al uso de *Big Data* y Análisis de Aprendizaje con datos correspondientes a la Educación Superior. Por otro lado, enfatiza que el volumen de datos en las Instituciones de Educación Superior aumenta cada día y que la infraestructura de TI para el análisis de *Big Data* requiere inversión en hardware y software para realizar análisis en tiempo real y actualmente este proceso no tiene la suficiente madurez por falta de especialistas y recursos.

Cen, Ruta, y Ng (Cen, Ruta, and Ng, 2015) en su artículo, indican que *Big Data* ha demostrado valores significativos en nuestra percepción y previsión del mundo. Con el rápido desarrollo de las tecnologías de la información y comunicación y los dispositivos móviles, los datos educativos han crecido a un ritmo sin precedentes. En su documento presenta algunas aplicaciones nuevas para el análisis de *Big Data* cuyo objetivo es mejorar la eficiencia y eficacia del aprendizaje de estudiantes y ampliar la retención de conocimientos. En primer lugar, proponen utilizar algoritmos de aprendizaje supervisados (clasificación o regresión), para predecir rendimiento académico de los estudiantes en segundo lugar, proponen utilizar estas predicciones para orientar el contenido de los módulos y cursos de los estudiantes reflejando en sus habilidades de aprendizaje, intereses y metas de educación, en tercer lugar, proponen enfocarse en la forma que se realiza el proceso de aprendizaje de cada estudiante y obtener mejoras mensurables de los estudiantes en el rendimiento académico alcanzado y retención de conocimientos a largo plazo.

Finalmente, de las publicaciones revisadas, se puede verificar que aplicar *Big Data* en la educación aporta gran valor para realizar análisis que permitan mejorar las metodologías de enseñanza y esfuerzos de los alumnos. Sin embargo, en ninguno se ha implementado un *Data Lake* para almacenar los datos, por ello el presente trabajo aportará gran valor a los procesos investigativos y generación de modelos que se plasmen en la Universidad Internacional SEK con los datos de profesores y estudiantes, a partir de este documento.

CAPÍTULO 3

MÉTODO

En este capítulo se describe el marco general de investigación, para lo cual se seleccionó la metodología "*Design Research*", cuyo enfoque se basa en diseñar o desarrollar un producto para brindar la solución a un problema, esta metodología ejecuta los siguientes procedimientos para su aplicación: 1. Identificación del problema de investigación, 2. Definición de requerimientos, 3. Diseño y desarrollo de la solución, 4. Implementación de la solución, 5. Evaluación de resultados de la solución. "*Design Research*" está orientado a la creación de nuevas prácticas, y está vinculado al área de investigación en Tecnologías de la Información. (Valverde, 2016).

Con el enfoque de esta metodología, se realizó una investigación experimental, que permite examinar o explorar un problema de investigación poco estudiado en el Ecuador, con especial énfasis en el ámbito de la educación superior. Bajo esta premisa, el presente trabajo de investigación cumple con los procedimientos de la metodología *"Design Research"*, de la siguiente manera:

1. Se identificó como problema, que la Universidad Internacional SEK no dispone de un Lago de Datos que le permita almacenar y trabajar con la información de Canvas.
2. La definición general de requerimientos fue la necesidad de contar con el *Data Lake* del contenido educativo de Canvas en la Universidad Internacional SEK para que a futuro se puedan aplicar mejoras en el proceso educativo.
3. Se diseñó la arquitectura del ecosistema *Big Data* para el almacenamiento del contenido educativo en el *Data Lake*.
4. Se desarrolló e implementó la arquitectura haciendo uso de las herramientas de *Big Data* en el entorno de *Hadoop*, para transferir y almacenar los datos de Canvas en el *Data Lake*.
5. Se evaluó la solución implementada a través del análisis de resultados.

3.1 Diseño del *Data Lake* del contenido educativo de Canvas de la UISEK

A continuación, se presenta el trabajo realizado para diseñar la arquitectura del ecosistema *Big Data* para el almacenamiento del contenido educativo en el *Lago de Datos*, el cual de manera general permitió establecer la forma de trabajo para desarrollar e implementar el repositorio del contenido educativo de Canvas de la UISEK, desde la exploración hasta la ingesta de datos.

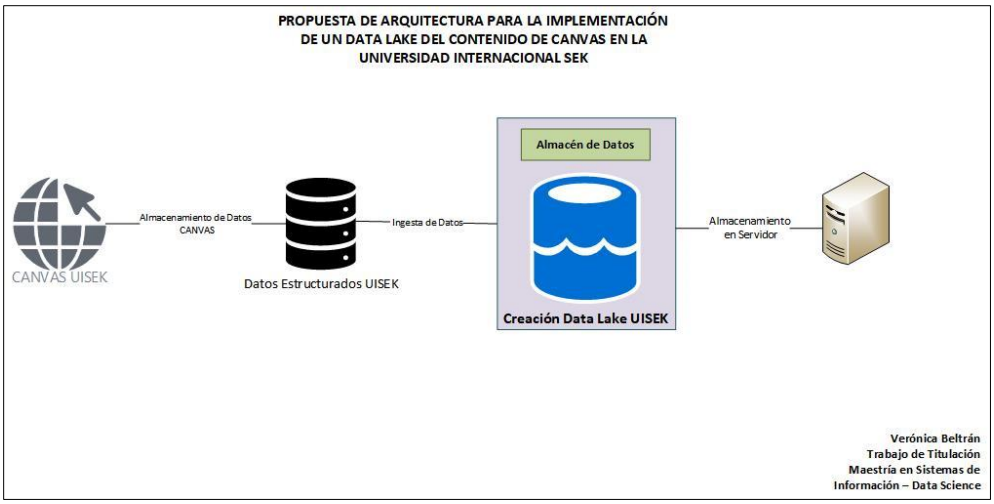


Figura 6 Diseño de la Arquitectura Data Lake Canvas UISEK

El diseño del *Data Lake* propuesto funciona de la siguiente manera, Canvas almacena sus datos en una base de datos relacional; la misma que contiene toda la información que el alumno y docente generan en la plataforma mencionada; estos datos se almacenan en una estructura de estrella en tablas de hechos y dimensiones, es decir existen relaciones entre las tablas. El *Data Lake* se alimentará de la información que se encuentra almacenada en la base de datos de Canvas haciendo uso de un API que permita acceder a esta información, el *Data Lake* con toda esta información estará implementado en el servidor, lo cual se muestra en la Figura 6.

3.1.1 Características del Hardware

Para que el diseño propuesto funcione, es indispensable contar con el *hardware* adecuado para montar las herramientas que permiten desarrollar e implementar el *Data Lake*. En primer lugar, se utilizó el servidor físico proporcionado por la UISEK, el cual tiene las siguientes características:

- *Hardware*: Dell Inc. PowerEdge T440
    - CPU: 16 núcleos
    - Memoria RAM: 80 GiB
    - Disco: 960 GB SSD
- Sistema Operativo: CentOS Linux 7 (Este sistema operativo permite trabajar con la máquina virtual de Cloudera).

Dentro de la sección de Máquinas Virtuales del servidor, se instaló la máquina virtual Cloudera 5.13.0-0-kvm con las siguientes características, mismas que permitirán el correcto funcionamiento del *Data Lake*:

- Memoria: 64 GiB
- vCPUs: 1
- Disco: 192 GiB SSD
  Capacidad de almacenamiento: 192 GB

Es importante mencionar que, la máquina virtual de Cloudera 5.13.0-0-kvm incluye las herramientas del entorno de la Arquitectura de *Hadoop*, mismas que se detallaron en la sección anterior correspondiente al Marco Teórico. Esta arquitectura es flexible y escalable ya que permite trabajar con diferentes herramientas, entre ellas *Hive*, que permite transformar datos complejos (Inc Cloudera 2021); se basa en el sistema de archivos distribuidos HDFS para el almacenamiento, mismo que permite trabajar con grandes cantidades de datos de cualquier tipo.

3.2 Desarrollo e implementación del Data Lake de los datos de uso del LMS Canvas de la Universidad Internacional SEK

Para realizar el desarrollo e implementación del *Data Lake* propuesto, se dividió el proceso en tres fases como se muestra en la Figura 7, las cuales se detallan a continuación:

1. Exploración de datos (para estudiar la estructura de datos que tiene Canvas),
2. Preparación de datos (para extraer los datos de Canvas que se cargarán en el *Data Lake*),
3. Ingesta de datos (para almacenar en el *Data Lake* los datos que se extraigan de Canvas y posteriormente almacenarlos también en *Hive*).

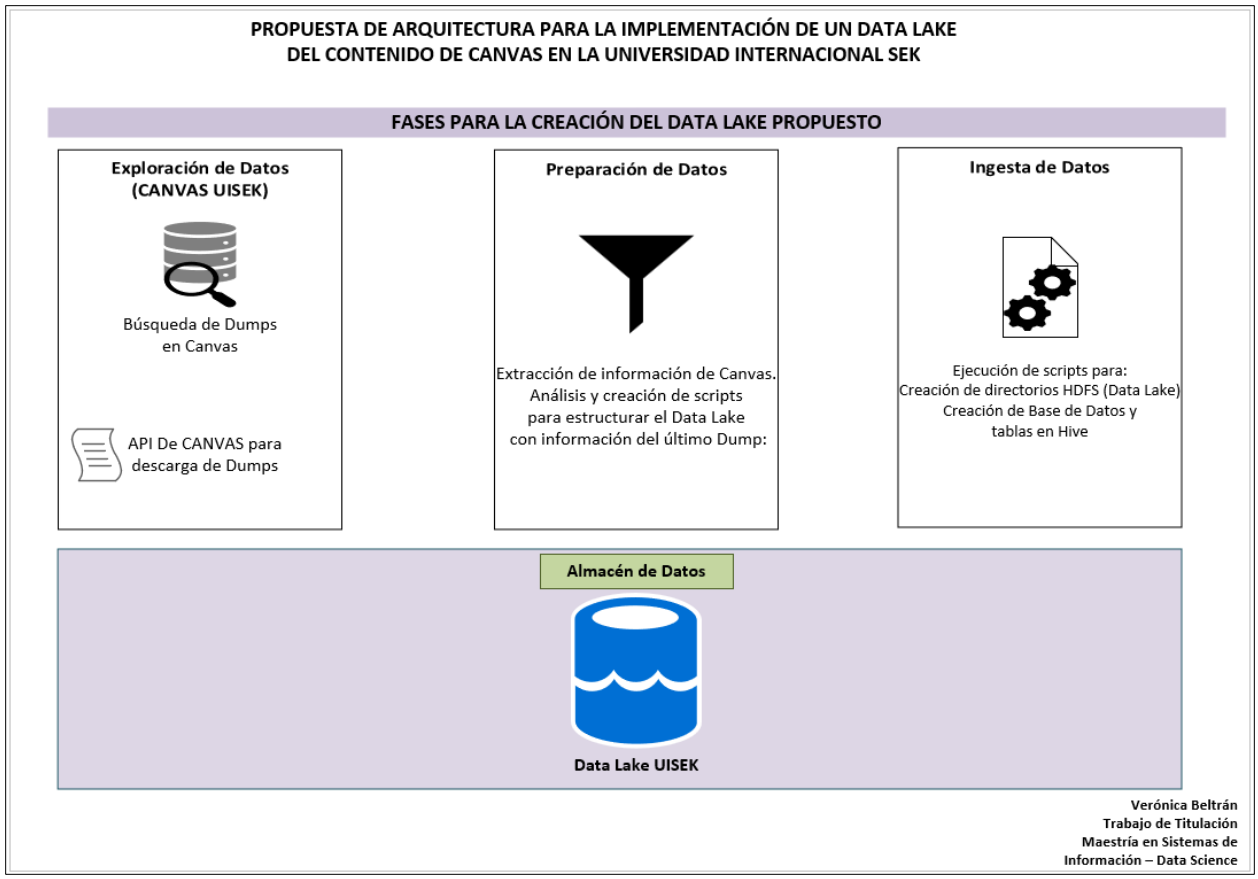


Figura 7 Fases para creación del Data Lake

3.2.1 Exploración de datos

En esta sección se explica cómo se realizó el proceso de exploración de datos que la UISEK almacena en la estructura de Canvas. Esta fase consistió en estudiar la estructura de datos que tiene Canvas de la Universidad Internacional SEK; sus tablas, campos, y tipo de datos. De esta manera se pudo observar que Canvas cuenta con tablas de hechos y dimensiones cuya información se encuentra disponible para descargar en archivos planos y se adhieren a un esquema de estrella. Además, se identificaron un total de 48 esquemas disponibles.

Cada esquema contiene un total de 100 *dumps* disponibles y cada *dump* contiene 117 tablas con su respectiva estructura de datos (nombre de tablas, columnas, tipo de dato), comprendido este concepto, se revisó el diccionario de datos de Canvas donde se pudo verificar que está compuesto por 51 tablas de hecho, 65 tablas de dimensiones y 1 tabla de ambas; es decir es una tabla de hecho y dimensión al mismo tiempo, sin embargo, no todas estas están disponibles en los *dumps* para la UISEK. Para la extracción de información de Canvas, se determinó trabajar con la última versión de esquemas disponible y con el último *dump* disponible de este esquema de datos, pues esta estructura es la que contiene la información actualizada.

Cabe mencionar que, de las 117 tablas existentes, la tabla *"Requests"* tiene una lógica particular, debido a que es la única tabla de históricos la cual se actualiza cada día con nueva información, es decir cada *dump* sobrescribe la información anterior de esta tabla. Para solventar este caso específico, la primera vez que se ejecuta el proceso se descargan todos los *dumps* históricos de la tabla "*Requests*" y para las siguientes veces que se ejecuta el cron se determinó realizar la concatenación de cada archivo descargado de esta tabla en

cada *dump* con la finalidad de que la información histórica no se sobrescriba en cada descarga. En la Figura 8 se representa gráficamente la estructura de datos de Canvas UISEK descrita.

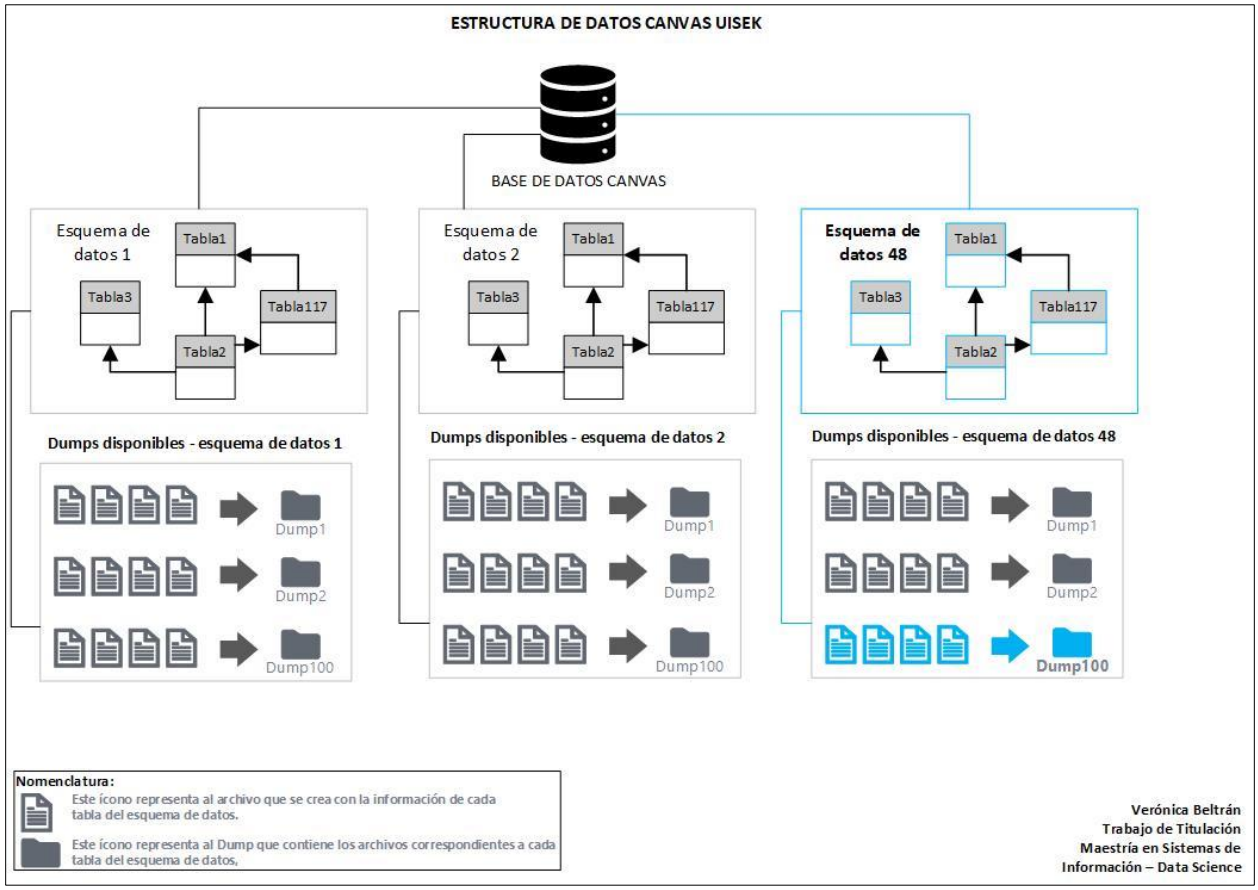


Figura 8 Estructura de Datos Canvas UISEK

Una vez realizadas estas tareas de exploración de datos, se vio la necesidad de utilizar un API con la finalidad de realizar la conexión a Canvas UISEK con sus propias credenciales (API llave y Secreto).

3.2.1.1 Conexión API de Canvas

Canvas maneja seguridad *HMAC authentication* (código de autenticación de mensaje basado en hash), por lo cual para acceder a sus servidores se debe realizar la conexión a estos mediante la Clave de API y Secreto de API. Estas credenciales permiten realizar descargas de información almacenada en los servidores.

Cumpliendo con estas premisas, se accedió a Canvas de la UISEK, haciendo uso del API de Canvas en Python; para ello se realizó la conexión con la clave, secreto y URL, mismas que fueron proporcionadas por el administrador. Con la finalidad de que la clave y el secreto de API se mantengan seguras, se crearon variables de entorno de Cloudera, de esta manera en el script inicial que se creó se exportan estas variables de entorno para que no se pierda la conexión con Canvas en ningún momento de la ejecución del proceso de descarga de la información.

3.2.2 Preparación de datos

En esta sección se explica cómo se realizó la preparación de datos para extraer la información de Canvas con el fin de programar su desarrollo e implementar el *Data Lake* propuesto, información que se describe en la nota[1].

Con el objetivo de preparar los datos que se almacenan en el *Data Lake* con la información que contiene cada tabla descargada de Canvas, se creó el archivo principal canvasInit.sh (Anexo A), el cual permite ejecutar procesos internos para realizar las siguientes actividades:

1. Declaración de variables de entorno.
2. Cambio a directorio de trabajo.
3. Eliminación de archivos descargados.
4. Generación y ejecución de archivo CanvasLog.py (de manera general, este archivo permite realizar la descarga de las tablas de Canvas del último *dump* disponible).
5. Generación del archivo canvasuisek.sh (de manera general, este archivo permite crear los directorios HDFS para la implementación del *Data Lake*).
6. Generación de archivo canvasbdd.hql (de manera general, este archivo permite realizar la creación de la base de datos que almacenará todas las tablas de Canvas que contiene el *Data Lake*).

3.2.2.1 Declaración de variables de entorno

Con la finalidad de poder acceder a la información de Canvas de la UISEK, en la parte inicial del archivo canvasInit.sh se declaró las variables de entorno y exportar las mismas para permanecer conectados en todo el proceso de creación del *Data Lake* propuesto, lo cual se presenta en la Figura 9.

- API_URL (Corresponde a la dirección https que la UISEK tiene en *Instructure*, donde está Canvas).
- API_KEY = (Corresponde a la llave de API asignada a la UISEK para conectarse a Canvas).
- API_SECRET = (Corresponde al secreto de API asignado a la UISEK para acceder a Canvas).

```
#*** Variables de entorno para conectarse a Canvas ***
    API_URL = 'https://uisek.instructure.com'
    API_KEY = os.environ['API_KEY']
    API_SECRET = os.environ['API_SECRET']
```

Figura 9 Variables de Entorno

3.2.2.2 Directorio de trabajo

Con la finalidad de poder trabajar en el directorio UISEK, en esta sección del *script* se ejecuta un comando para acceder a este directorio y posteriormente trabajar en el mismo, el comando utilizado se presenta en la Figura 10.

```
# *** Cambio a directorio de trabajo ***
cd /home/cloudera/UISEK/
echo "CAMBIO DIRECTORIO UISEK"
```

Figura 10 Cambio a directorio UISEK

[1] Se creó el directorio UISEK en la raíz del entorno de Cloudera, es en este directorio donde se guardará toda la información que se describa en los siguientes pasos.

3.2.2.3 Eliminación de archivos descargados esquema

En esta sección del *script* se eliminan los archivos descargados existentes en la carpeta "*data*", que se encuentra en el directorio físico creado "UISEK", esto con el objetivo de que no se almacenen archivos basura, ni que se duplique la información generada a partir de la ejecución del archivo principal, ni de los archivos que se generan a partir de este, el comando utilizado para ejecutar esta acción se presenta en la Figura 11.

```
# *** eliminacion de archivos descargados ***
rm -rf data
echo "ELIMINACION DE ARCHIVOS"
```

Figura 11 Comando para eliminar carpeta "data"

Cabe indicar que en la carpeta "*data*" se almacenará la información actualizada de Canvas, la cual se descarga del último volcado de datos "*dump*".

3.2.2.4 Ejecución del archivo CanvasLog.py

Una vez realizada la conexión a Canvas y con los directorios físicos creados, se ejecutó el archivo "CanvasLog.py" (Anexo B), generado para la creación de *scripts* y descarga de último volcado de datos "*lastDump*". En este archivo se parametrizaron las siguientes actividades:

1. Descarga de *keys* de las tablas del último esquema.
2. Definición y creación de los repositorios a trabajar.
3. Definición de extensiones de los archivos.
4. Extracción de información del Diccionario del Esquema.
5. Definición de directorios en HDFS.
6. Creación del archivo canvasbdd.hql para creación de base de datos y tablas en *Hive*.
7. Creación de archivo canvasuisek.sh. para creación del *Data Lake*.
8. Optimización de memoria.
9. Descarga del último *dump* del esquema.
10. Descompresión de archivos.

3.2.2.4.1 Descarga de *keys* de las tablas del último esquema

La plataforma Canvas de la UISEK tiene un total de 48 esquemas los cuales almacenan toda la información de tablas en la variable *key*, por ello en esta sección del archivo "CanvasLog.py", se tomó la última versión del esquema disponible y de este se toma la variable *key_on_tablenames*, con lo cual se generan los archivos que se describen en los siguientes pasos de este documento. En la Figura 12 se muestra el código utilizado para realizar la descarga de *keys* de las tablas del último esquema.

```
#*** Código para tomar la última versión del esquema dispobible ***
cd = CanvasDataAPI(api_key=API_KEY, api_secret=API_SECRET)

schema_versions = cd.get_schema_versions()

print('Found {} schema versions.'.format(len(schema_versions)))

#*** Código para tomar la varible key_on_tablenames de la última versión del esquema ***

schema = cd.get_schema('latest', key_on_tablenames=True)
```

Figura 12 Script de descarga de la última versión del Esquema

3.2.2.4.2 Definición y creación de los repositorios a trabajar

Con la finalidad de que la creación de repositorios sea automática y se puedan utilizar los archivos descargados, se escribieron algunas líneas de código en *Python* las cuales realizan la creación de los repositorios: *lastDump, process, csvFiles, lastDesompressedTxt y requests*. En la Figura 13 se presenta el código utilizado para la creación de los mencionados repositorios.

Estos repositorios almacenan la siguiente información respectivamente:

- *lastDump*: repositorio donde se almacena la información del último *dump* que se obtuvo luego de descargar el último esquema, de Canvas UISEK.
- *Process*: repositorio donde se almacenan los repositorios *csvFiles* y *lastDumpDecompressed*, en la fase de preparación de datos.
- *csvFiles*: repositorio donde se almacenan los archivos .csv que se descargan con información del último *dump* en formato .csv.
- *lastDumpDesompressedTxt*: repositorio donde se almacenan las tablas descomprimidas en formato .txt.
- *requests*: repositorio donde se almacena la tabla "*requests*" en cada descarga de información en formato .txt.

```
# Definición de los directorios a trabajar

principalDirectory = "./data/lastDump"
Path(principalDirectory).mkdir(parents=True, exist_ok=True)

processDirectory = "./data/process"
Path(processDirectory).mkdir(parents=True, exist_ok=True)

decompressFiles = processDirectory + "/csvFiles"
Path(decompressFiles).mkdir(parents=True, exist_ok=True)

decompressFilesTxt = processDirectory + "/lastDumpDecompressedTxt"
Path(decompressFilesTxt).mkdir(parents=True, exist_ok=True)

requestDirectory = "./requests"
Path(requestDirectory).mkdir(parents=True, exist_ok=True)
```

Figura 13 Definición de Directorios

3.2.2.4.3 Definición de las extensiones de los archivos

En este paso del proceso se parametrizó mediante código en *Python* el formato de archivo que debe contener la información de cada tabla del último *Dump* descargado.

En primer lugar, se descomprime cada archivo y luego se los transforma uno por uno primero a .txt y a partir de este se los transforma a .csv por todas las ventajas que tiene al momento de trabajar con los *DataSet*.

Los formatos con los que se trabaja se presentan en la Figura 14 y son los siguientes:

- .gz (corresponde a archivos comprimidos).
- .txt (corresponde a archivos de texto).
- .csv (corresponde a archivos de hojas de cálculo)

```
# Definicion de las extensiones de archivo a usar

gzFile = ".gz"
txtFile = ".txt"
csvFile = ".csv"
```

Figura 14 Extensión de los archivos

3.2.2.4.4 Extracción de información del diccionario del esquema

En este paso se realizan las siguientes parametrizaciones para poder almacenar la información que se extrae con las *keys* del esquema que se obtuvo en el paso 3.2.2.4.1. En la Figura 15 se presenta el código utilizado para ejecutar las acciones descritas:

1. Se define un directorio general para almacenar los *scripts* que se generarán en los pasos posteriores, en caso de no existir el directorio, se lo crea caso contrario se escribe en él.
2. Se define el nombre del *data lake* como “canvasuisek”.
3. Se ingresa a la ubicación del directorio UISEK donde se creará el *data lake*.
4. Se parametriza el comando para salir del modo seguro con la finalidad de que podamos realizar los pasos posteriores, sin restricciones.
5. Se genera un comando para eliminar el directorio HDFS “canvasuisek”, con la finalidad de que se crear el mismo, con información actualidad cada que se ejecute el cron.
6. Se genera un comando para crear el directorio HDFS “canvasuisek”, el cual contendrá la información actualizada correspondiente a las descargas actualizadas del último *dump*.
7. Se define el nombre de la base de datos como “canvasbdd”.
8. Se genera un comando para crear la base de datos en caso de que no exista.
9. Se define mediante *script* qué base de datos se debe usar para la ingesta de datos.
10. Se extrae la información de los *keys* es decir se extrae el nombre de cada tabla, campo y tipo de dato perteneciente a la última versión del esquema.

```
"""
Se extrae información del diccionario de esquema
"""
# Se define un directorio general para almacenar los scripts
principalScriptDirectory = "./data/scripts/"

# Si no existe, se crea el directorio anterior
Path(principalScriptDirectory).mkdir(parents=True, exist_ok=True)

# Nombre del data lake
dataLakeName = 'canvasuisek'

# Variable para almacenar script hdfs
scriptHDFS = '\n'

# Se ubica en el directorio para crear el datalake
scriptHDFS += 'cd /home/cloudera/UISEK\n'

# Comando para salir del modo seguro
scriptHDFS += 'sudo -u hdfs hadoop dfsadmin -safemode leave\n'

# Comando para eliminar el directorio HDFS (dataLake) canvasuisek
scriptHDFS += 'hdfs dfs -rm -r /' + dataLakeName + '\n'

# Comando para crear el directorio HDFS (dataLake) canvasuisek
scriptHDFS += 'hdfs dfs -mkdir /' + dataLakeName + '\n'

# Nombre de la base de datos
dataBaseName = "canvasbdd"

# Variable para almacenar la información del script
scriptCarga = '\n'

# Creamos database si no existe
scriptCarga = 'create database if not exists ' + dataBaseName + ';\n'

# Se inicia el script, describiendo que data base usar
scriptCarga += "use " + dataBaseName + ';\n'

# Se extraé información de los keys, es decir del nombre de cada tabla perteneciente al schema
new_listKeys = list(schema.keys())
```

Figura 15 Extracción de información del diccionario de la última versión del Esquema

3.2.2.4.5 Definición de los directorios en HDFS (*Data Lake*)

Como se mencionó en puntos anteriores, se decidió que lo óptimo es crear un directorio HDFS principal y directorios internos que se alimentan de la información de los archivos correspondientes a las tablas de Canvas de la UISEK que se descargaron en los pasos anteriores. Es decir, se creó un directorio independiente para cada tabla, en el que se guardará la información del nombre, tipo y descripción de cada campo o columna. En las Figuras 16, 17 y 18 se muestra el código generado para ejecutar las acciones de creación de los directorios HDFS, preparación de tablas y generación de archivos para ingesta de datos y creación de bases de datos en *Hive*, lo cual se describen a continuación:

1. Se define un directorio principal donde se almacena las tablas con toda su información, mismas que se utilizan para crear los directorios HDFS e ingesta de datos, este directorio se nombró "*tables*".
2. Se define una secuencia variable de control para determinar el número que corresponde a la iteración.
3. Se recorre el arreglo de *keys* (información de cada tabla)

a. Se define el directorio principal “canvasuisek”.
b. Se define la carga de los archivos .csv en cada directorio interno por tabla.
c. Se define el nombrado de cada directorio hdfs por tabla.
d. Se extrae del esquema la información de cada columna perteneciente a cada tabla.
e. Se definen variables *string* generales para conectar las propiedades de cada columna (tipo de campo, nombre de campo, descripción de campo).
f. Se define una variable de control para identificar el número de iteración dentro del *array* de campos de cada tabla.
g. Se define la creación de un archivo vacío por cada tabla.

```
"""
 Se crea un directorio independiente para cada tabla, en el que se guardará la información del nombre,tipo y descripción de cada campo o columna
"""
# Definición de un directorio general para esta información
principalTableDirectory = "./data/tables/"

# Secuencia variable control para determinar la iteración
secuencia = 0

# Se recorre el array de keys(nombres de tablas)
for key in new_listKeys:
    # Crea el directorio hdfs para cada tabla
    scriptHDFS += 'hdfs dfs -mkdir /' + dataLakeName + '/' + str(key) + '\n'

    # Carga de los archivos .csv en cada directorio interno por tabla
    scriptHDFS += 'hdfs dfs -put ' + '/home/cloudera/UISEK/data/process/csvFiles/' + str(key) + '.csv /'+ dataLakeName + '/' + str(key) + '\n'

    # Crea el directorio hdfs para cada tabla
    scriptHDFS += 'echo ' + str(secuencia) + ' de ' + str(len(new_listKeys)) + '\n'
    secuencia += 1

    # Crea la tabla si existe
    scriptCarga += '\ncreate external table if not exists ' + str(key) + '('

    # Si no existe, crea un directorio por cada nombre de tabla, si ya existe, no hace nada
    Path(principalTableDirectory+str(key)).mkdir(parents=True, exist_ok=True)

    # Extrae del diccionario de esquema, la información de las columnas pertenecientes a cada tabla
    new_listItemsSchema = list(schema[str(key)]['columns'])

    # Definición de variables string generales para ir concatenando las propiedades de cada columna
    campoType = ""
    campoName = ""
    campoDescription = ""

    # Definición de variable control, para identificar la iteración dentro del array de campos de cada tabla
    camposLength = 0

    # Crea un archivo vacío para cada tabla
    fcsv = open(decompressFiles+"/" + str(key) + ".csv", "w", encoding='utf-8')
    fcsv.write('')
    fcsv.close()
```

Figura 16 Script para definición de directorios HDFS y creación de archivos por tabla

h. Se recorre el array creado con la información de cada columna (campo), para generar los archivos .txt. En la Figura 17 se presenta el código utilizado para ejecutar las acciones descritas:
    i. Se controla el tipo de dato de cada campo.
    ii. Se realiza la concatenación de la información de cada campo, para generar un solo documento con esta información de cada tabla.
    iii. Se realiza el control para conocer si es la última iteración del *array*.
    iv. Se crea un archivo .txt para almacenar la información del nombre de campo de cada tabla.
    v. Se crea un archivo .txt para almacenar la información del tipo de dato de cada tabla.
    vi. Se crea un archivo .txt para almacenar la información de la descripción de cada tabla.

```
# Se recorre el array con la información de cada columna(campo)
for campo in new_listItemsSchema:
    # Se identifica la iteración del array
    camposLength += 1
    #campoName.append(campo['name'])
    #campoType.append(campo['type'])
    #campoDescription.append(campo['description'])

    #Controlar el tipo de dato enum y double precision
    if campo['type'] == 'double precision':
        campo['type'] = 'double'

    if campo['type'] == 'enum':
        campo['type'] = 'string'

    if campo['type'] == 'varchar':
        campo['type'] = 'string'

    if campo['type'] == 'text':
        campo['type'] = 'string'

    if campo['type'] == 'integer':
        campo['type'] = 'int'

    if campo['type'] == 'guid':
        campo['type'] = 'bigint'

    if campo['type'] == 'datetime':
        campo['type'] = 'bigint'

    # Se concatena la información de cada campo, para formar un solo documento
    campoName += campo['name']+" "
    campoType += (campo['type'])+" "

    # Se controla si es la última iteración del array
    if camposLength == len(new_listItemsSchema):
        # Se agrega la información de cada campo y tipo de dato respectivo a cada tabla,
        # si e sla última iteración, significa que es el último campo a registrar,
        # por lo que se debe finalizar con )
        scriptCarga += '\n' + str(campo['name']) + ' ' + str(campo['type']) + ')'
    else:
        # Se agrega la información de cada campo y tipo de dato respectivo a cada tabla y se continua con una
        # (,) para avanzar con el siguiente campo
        scriptCarga += '\n' + str(campo['name']) + ' ' + str(campo['type']) + ','

    # Crea un archivo .txt, para almacenar la información del nombre de campo de cada tabla
    fp = open(principalTableDirectory + str(key) + "/" + str(key) + "_name.txt", "w")
    fp.write(str(campoName))
    fp.close()

    # Crea un archivo .txt, para almacenar la información del tipo de dato de cada tabla
    fp = open(principalTableDirectory + str(key) + "/" + str(key) + "_type.txt", "w")
    fp.write(str(campoType))
    fp.close()

    # Crea un archivo .txt, para almacenar la información de la descripción de cada tabla
    # fp = open(principalTableDirectory + str(key) + "/" + str(key) + "_description",
    #           "w")  # Crea un archivo txt en el que se descomprime
    # fp.write(str(campoDescription))
    # fp.close()

scriptCarga += "\nrow format delimited fields terminated by ','\nstored as textfile"
scriptCarga += "\nlocation '/" + dataLakeName + "/" + str(key) + "'; \n"
```

Figura 17 Preparación de tablas correspondientes con su información

4. Se crea un archivo .hql con el *script* para realizar la creación de la base de datos y creación de tablas en *Hive*.
5. Se crea un archivo .sh con el *script* para la creación del *Data Lake*.

```
# Crea un archivo .hql, para almacenar el script para ingesta de datos
fp = open(principalScriptDirectory + dataBaseName + ".hql", "w")
fp.write(scriptCarga)
fp.close()

# Crea un archivo .sh, para almacenar el script para la carga hdfs
fp = open(principalScriptDirectory + dataLakeName + ".sh", "w")
fp.write(scriptHDFS)
fp.close()
```

Figura 18 Creación de archivos para realizar ingesta de datos y carga de directorios HDFS

3.2.2.4.6 Optimización de memoria

Al descargar un número grande de datos, es necesario realizar una optimización del uso de memoria, por lo cual se incluyó un *script* para eliminar todos los archivos basura de la memoria RAM, haciendo uso de librerías en *Python*, con ello se optimiza el procesamiento de datos. En la Figura 19 se presenta el código utilizado para realizar la optimización de memoria.

```
"""
Optimización uso de memoria
"""
# Se elimina variable fp
del fp

# Se elimina toda la basura en memoria
gc.collect(generation=2)
```

Figura 19 Comando para optimización de memoria

3.2.2.4.7 Descarga del último *dump* del esquema

Se generó un *script* en *Python*, el cual permite realizar la descarga del último *dump* disponible, mismo que corresponde al último esquema que se almacenó en los pasos previos para obtener la información correspondiente a cada tabla que se descarga de Canvas de la UISEK.

En esta sección del archivo "CanvasLog" se realiza la descarga del ID del último *dump*, con el cual se obtienen un total de 117 tablas disponibles en Canvas, con la información respetiva, en formato .gz. Esta información se almacena en el directorio "*lastDump*" y *"Requests"* definidos en el punto 3.2.2.4.2 del presente documento.

En la Figura 20 se presenta el código utilizado para ejecutar la descarga del último *dump* con la información de cada tabla.

```
Inicio de descarga
"""

print('The latest schema has {} tables.'.format(len(schema)))

dumps = cd.get_dumps()

print('There are a total of {} dumps available'.format(len(dumps)))

one_dump = dumps[0]

dump_files = cd.get_file_urls(dump_id=one_dump['dumpId'])

dump_table_names = dump_files['artifactsByTable'].keys()

print('The dump ID {} contains files for {} tables.'.format(one_dump['dumpId'], len(dump_table_names)))

# are there tables that are in the schema but not in the dump, or vice-versa?
not_in_dump = [x for x in schema.keys() if x not in dump_table_names]

print('These tables are present in the schema but missing from the dump: {}'.format(not_in_dump))

cont=0

dumps_aux = dumps[::-1]
```

```
if len(data)==0:
    print('Descarga de últimos dumps tabla request')

    for i in dumps_aux:
        cont +=1
        if cont == 99:
            print("Se ha finalizado la descarga de la tabla requests de los 99 dumps anteriores al last ")
            break
        print('The dump ID {}'.format(i['dumpId']))
        latest_requests_files = cd.download_files(dump_id=i['dumpId'], table_name='requests',
                                                  download_directory='./requests/allDumps')
        content = os.listdir('./requests/allDumps')
        print(str(cont) + " de "+ str(len(dumps_aux)-1))

        fp = open('./requests/allDumps/requests.txt', "wb")  # Crea un archivo txt en el que se descomprime
        with gzip.open('./requests/allDumps/' + str(content[0]), "rb") as f:
            bindata = f.read()
        fp.write(bindata)
        fp.close()

        fp = open('./requests/allDumps/requests.txt', "r", encoding='utf-8')
        data = fp.read()
        data = data.replace(",", ";")
        data = data.replace("\t", ",")
        fp.close()

        fp = open('./requests/requests.txt', "a")  # Crea un archivo txt en el que se descomprime
        fp.write(data)
        fp.close()

        # Se elimina variable fp
        del fp
        del data

        # Se elimina toda la basura en memoria
        gc.collect(generation=2)

        rmtree('./requests/allDumps')
        print("Se inicia la descarga de todas las tablas del last dump")

        # get all of the files from the latest dump: (this will take a while to run)
        latest_dump_files = cd.download_files(dump_id='latest', download_directory='./data/lastDump')
        print('Latest course files: {}'.format(latest_dump_files))

        print("Se finaliza la descarga de todas las tablas del last dump")
```

Figura 20 Descarga de tablas del último Dump

3.2.2.4.8 Descompresión de archivos

Finalmente, se realiza la descompresión de las tablas descargadas en el paso anterior con la finalidad de utilizar las mismas para la ingesta de datos al Data Lake y creación de tablas en Hive.

En este paso se descomprimen las tablas de formato .gz a archivos .txt y .csv con el nombre de cada tabla respectivamente, además de realiza una nueva optimización de memoria para evitar advertencias al ejecutar el archivo.

Estos archivos descomprimidos se almacenan respectivamente en el directorio "*lastDumpDesompressedTxt*", "*requests*" exclusivamente para el archivo descomprimido de la tabla

requests[2] y *csvFiles* definidos en el punto 3.2.2.4.2 del presente documento. En la Figura 21 se muestra el código utilizado para realizar la descompresión de archivos de cada tabla.

```
"""
Descomprimir archivos
"""
content = os.listdir(principalDirectory)
aux = []

for i in content:
    aux.append(i.split("-", 1)[0])
def decompressDump(path=None, pathToDecompressTxt=None, pathToDecompressCsv=None, aux=None, requestDirectory=None):
    pathTxt = pathToDecompressTxt + ".txt"
    pathGz = path
    pathCsv = pathToDecompressCsv + ".csv"
    pathRequest = requestDirectory + "/"+ aux + ".txt"
    fp = open(pathTxt, "wb")  # Crea un archivo txt en el que se descomprime
    with gzip.open(pathGz, "rb") as f:
        bindata = f.read()
    fp.write(bindata)
    fp.close()
    fp = open(pathTxt, "r", encoding='utf-8')
    data = fp.read()
    data = data.replace(",", ";")
    data = data.replace("\t", ",")
    fp.close()
    if (aux == 'requests'):
        fp = open(pathRequest, "a",encoding='utf-8')  # Concatena cada iteracion
        fp.write(data)
        fp.close()
        fr = open(pathRequest, "r",encoding='utf-8')  # Abre request almacenado
        fpr = open(pathTxt, "w", encoding='utf-8')
        data=fr.read()
        fpr.write(data)
        fpr.close()
        fr.close()
    fcsv = open(pathCsv, "w",encoding='utf-8')
    fcsv.write(data)
    fcsv.close()
    """
    Optimización uso de memoria
    """
    # Se elimina variable fp
    del fp
    del data
    del fcsv
    # Se elimina toda la basura en memoria
    gc.collect(generation=2)
    return 0
for i in range(len(aux)):
    decompressDump(principalDirectory+"/"+content[i],decompressFilesTxt+"/"+aux[i],decompressFiles+"/"+aux[i],aux[i],requestDirectory)
    print("Archivo Descomprimido:"+aux[i])
    print(" "+ str(i+1) +" Tablas descomprimidas de: " + str(len(aux)))
```

Figura 21 Descompresión de tablas del último Dump

3.2.3 Ingesta de datos

En esta sección se detallan los pasos realizados para llevar a cabo el proceso de creación del *Data Lake*, creación de la Bases de Datos en *Hive* y la creación de cada tabla obtenida en el último volcado de datos disponible para la UISEK.

Con el objetivo de realizar la ingesta de datos al *Data Lake*, se ejecutaron las siguientes actividades:

1. Ejecución del archivo canvasuisek.sh (de manera general, este archivo permite crear los directorios HDFS para la implementación del *Data Lake*).

2 Específicamente para la tabla "*requests*" se realiza la descarga de la información que trae cada dump de Canvas, esta tabla se descomprime y transforma a formato .txt; este archivo nuevo se concatena con el archivo anterior que se encuentre en el repositorio "*request*", se transforma a .csv y se almacena en la carpeta correspondiente dentro del repositorio *csvFiles*.

2. Ejecución de archivo canvasbdd.hql (de manera general, este archivo permite realizar la creación de la base de datos que almacenará todas las tablas de Canvas que contiene el *Data Lake*).

3.2.3.1 Ejecución del archivo canvasuisek.sh

Una vez culminada la ejecución del archivo CanvasLog.py, se genera y ejecuta el archivo canvasuisek.sh, el cual crea los directorios HDFS definidos en el paso 3.2.2.4.5 para la implementación del *Data Lake*, con la información del nombre de cada tabla correspondiente al último *dump* de Canvas de la UISEK que se obtuvo en la fase de preparación de datos.

A continuación, se describe el proceso que realiza el *script* de este archivo para crear el *Data Lake*, en la Figura 22 se muestra los comandos Linux utilizados para ejecutar las acciones descritas:

1. Se accede a la ruta cd/home/Cloudera/UISEK, lugar donde se encuentra almacenada toda la información que se ha obtenido de Canvas.
2. Se ejecuta el comando para salir del modo seguro y poder crear los directorios sin restricciones.
3. Se elimina cualquier directorio en caso de existir con el nombre "canvasuisek".
4. Se crea el directorio principal (*Data Lake*), el cual se nombró "canvasuisek".
5. Se crea cada uno de los directorios internos, con el nombre de las tablas obtenidos en los pasos previos.
6. Se realiza la ingesta de datos colocando cada archivo .csv correspondiente a cada tabla descargada de Canvas, en el directorio interno que le corresponde de acuerdo con el nombre de la tabla.

```
cd /home/cloudera/UISEK
sudo -u hdfs hadoop dfsadmin -safemode leave
hdfs dfs -rm -r /canvasuisek
hdfs dfs -mkdir /canvasuisek
hdfs dfs -mkdir /canvasuisek/course_dim
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/course_dim.csv /canvasuisek/course_dim
echo 0 de 117
hdfs dfs -mkdir /canvasuisek/account_dim
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/account_dim.csv /canvasuisek/account_dim
echo 1 de 117
hdfs dfs -mkdir /canvasuisek/user_dim
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/user_dim.csv /canvasuisek/user_dim
echo 2 de 117
hdfs dfs -mkdir /canvasuisek/pseudonym_dim
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/pseudonym_dim.csv /canvasuisek/pseudonym_dim
echo 3 de 117
hdfs dfs -mkdir /canvasuisek/pseudonym_fact
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/pseudonym_fact.csv /canvasuisek/pseudonym_fact
hdfs dfs -mkdir /canvasuisek/wiki_dim
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/wiki_dim.csv /canvasuisek/wiki_dim
echo 113 de 117
hdfs dfs -mkdir /canvasuisek/wiki_fact
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/wiki_fact.csv /canvasuisek/wiki_fact
echo 114 de 117
hdfs dfs -mkdir /canvasuisek/wiki_page_dim
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/wiki_page_dim.csv /canvasuisek/wiki_page_dim
echo 115 de 117
hdfs dfs -mkdir /canvasuisek/wiki_page_fact
hdfs dfs -put /home/cloudera/UISEK/data/process/csvFiles/wiki_page_fact.csv /canvasuisek/wiki_page_fact
echo 116 de 117
```

Figura 22 Muestra de creación de directorios en HDFS y carga de archivos .csv

Una vez finalizada la ejecución de este archivo, se tiene implementado el *Data Lake* con toda la información de las tablas que se encuentran en el último *dump* de Canvas; esto permite que se pueda crear la base de datos de Canvas en *Hive* a modo de espejo.

3.2.3.2 Ejecución de archivo canvasbdd.hql

Con la ejecución de este archivo se realiza la creación de la base de datos que almacena todas las tablas de Canvas que contiene el *Data Lake*, de la siguiente manera, en la Figura 23 y 24 se muestra los scripts utilizados para crear la base de datos y las tablas correspondientes en Apache *Hive*:

1. Se crea la base de datos "canvasbdd".
2. Se accede a la base de datos con la finalidad de realizar la creación de tablas.

```
create database if not exists canvasbdd;
use canvasbdd;
```

Figura 23 Creación y uso de la BDD canvasbdd

3. Se ejecutan los scripts para creación de cada tabla de Canvas, que se encuentra en el Data Lake con su nombre respectivo, este script tiene la siguiente estructura:

[3] **create external table** if not exists [nombre_tabla](
campo1 tipoDato,
campo2 tipoDato,
campo3 tipoDato,
campo4 tipoDato,
campoN tipoDato,
row format delimited fields terminated by ', '
stored as textfile
[4] **location '/canvasuisek/[nombre_tabla];**

```
create external table if not exists course_dim(
id bigint,
canvas_id bigint,
root_account_id bigint,
account_id bigint,
enrollment_term_id bigint,
name string,
code string,
type string,
created_at timestamp,
start_at timestamp,
conclude_at timestamp,
publicly_visible boolean,
sis_source_id string,
workflow_state string,
wiki_id bigint,
syllabus_body string)
row format delimited fields terminated by ','
stored as textfile
location '/canvasuisek/course_dim';
```

Figura 24 Creación de tabla "course_dim"

[3] El comando "*create external table*" en *Hive*, permite declarar la ubicación de una tabla externa.

[4] Al usar la cláusula "*Location*" dentro del script del comando "*create table*", permite especificar la ubicación de los datos de la tabla no especificada (externa).

3.2.4 Programación de tarea automática

Es importante mencionar, que los procesos anteriormente detallados, se ejecutan diariamente de manera programada en Linux, a través de un "crontab"; el cual ejecuta de manera automática el archivo "canvasInit.sh", diariamente a las 05:00 a.m.

De esta manera, cada día se almacena la información de los archivos descargados correspondientes a las tablas de Canvas en los directorios HDFS (*Data Lake*) y esta misma información se actualiza en las estructuras creadas en la base de datos *Hive*, al ser un proceso automático no es necesario que un usuario sea quien ejecute el mencionado archivo de forma manual, pues la tarea automática se encuentra parametrizada en el sistema. Además, como se muestra en la Figura 25, cada vez que se ejecuta el crontab, se genera el archivo "log.txt", el cual muestra mensajes de ejecución o errores que se presente al ejecutarse la tarea.

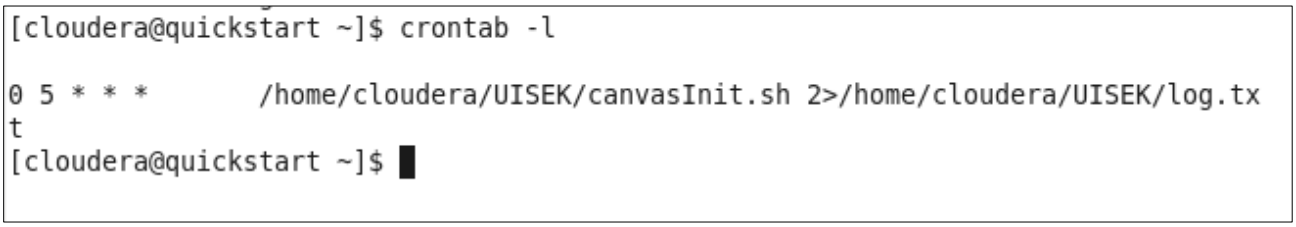


Figura 25 Programación "crontab"

CAPÍTULO 4

RESULTADOS

En este capítulo se exponen los resultados obtenidos luego de realizar todo el proceso de implementación del *Data Lake* de la Universidad Internacional SEK descrito en el capítulo 3 del presente documento, lo cual ha permitido que la UISEK cuente con un lago de datos construido con herramientas de *Big Data*, actualizado diariamente con la información de los estudiantes y docentes a partir de la Plataforma Educativa Canvas.

4.1Resultados obtenidos a partir de la preparación de datos

Como se detalló anteriormente, en la fase de Exploración y Preparación de Datos se realizaron algunas tareas a través de la ejecución de scripts en *Python*, así como de scripts que se ejecutan en la consola de comandos de Cloudera y *Hive*, para la implementación del *Data Lake*.

A continuación, se muestran los resultados obtenidos.

4.1.1 Creación de los directorios físicos "*data*" y "*requests*"

Según se indica en el punto 3.2.2, se generó un archivo principal llamado "canvasInit.sh", el cual al ejecutarse diariamente a través del cron programado realiza la creación del directorio "*data*", el cual contiene toda la información de Canvas que alimenta el *Data Lake* y el directorio "*request*", el cual contiene los archivos .txt descargados de la tabla "*requests*" de Canvas.

Dentro del directorio "*data*" se crearon los directorios: *lastDump*, *process, scripts* y *tables*. A continuación, se describe la información que almacena cada uno de ellos.

1. Directorio *lastDump*

En este directorio se almacenan las tablas que se descargaron de Canvas, las cuales están comprimidas en formato .gz.

2. Directorio *process*

En este directorio se crearon dos directorios internos, los cuales contienen información de las tablas descargadas en el punto anterior.

2.1. *csvFiles*: En este directorio se almacena la información del total de tablas existentes en Canvas, es decir, de las 117 tablas existentes en el modelo de datos, convertidas de formato .gz a .csv; con la finalidad de que estos archivos puedan colocarse en los directorios HDFS respectivos del *Data Lake*.

2.2. *lastDumpDesompressedTxt*: En este directorio se almacenan las tablas disponibles de Canvas, descomprimidas y transformadas en formato .txt. Con la finalidad de poder utilizar esta información en otros procesos que se ejecutan para la creación del *Data Lake*.

En la primera ejecución para descarga de datos se extrajeron todas las tablas existentes en Canvas, esto con la finalidad de que a futuro se pueda contar con la información de estas en caso de requerirse, pues en la UISEK no todas ellas se encuentran en el último *dump*.

3. Directorio *tables*

En este directorio se almacenan las 117 tablas con toda la información que se obtuvo del arreglo de *keys* (información de cada tabla) del último esquema de Canvas, que se obtuvo en el capítulo anterior; estas tablas se utilizaron para crear los directorios HDFS y realizar la ingesta de datos del *Data Lake*.

4. Directorio *scripts*

En este directorio se almacenan los archivos “canvasbdd.hql” y “canvasuisek.sh”, cuya creación se detalla en el punto 3.2.2 del capítulo 3 del presente trabajo de investigación y contienen los scripts para creación de la base de datos y los directorios HDFS respectivamente.

Como se indica en el punto 3.2.2.4.2 en el directorio “*request*” se almacenan los archivos de la tabla “requests” que se obtuvieron luego de cada descarga de información en formato .txt, los cuales se concatenaron y se almacenaron en formato .csv en el directorio “csvFiles”, para posteriormente ser ingestados en el HDFS.

4.2Resultados obtenidos a partir de la ingesta de datos

En la fase de Ingesta de Datos se realizó la ejecución de los archivos “canvasuisek.sh” y “canvasbdd.hql”; mismos que se generaron en el proceso anterior, tal como se describe en el capítulo 3, con el objetivo de realizar la implementación del *Data Lake* “canvasuisek” con la información de Canvas. A continuación, se presentan los resultados obtenidos de la ingesta de datos desde el entorno de Cloudera HUE, que es la interfaz de usuario *web* para la gestión de Hadoop, misma que permite mostrar los resultados de manera gráfica.

4.2.1 Creación de directorios HDFS “canvasuisek”

A continuación, se muestran los resultados de la construcción del *Data Lake* “canvasuisek”, el cual se detalló en el punto 3.2.3.1 del capítulo 3. Se pueden visualizar los directorios HDFS del *Data Lake*, con los archivos .csv correspondientes a cada tabla disponible de Canvas para la UISEK, producto de la ingesta de datos.

4.2.1.1 Directorio principal

Se consultó el directorio HDFS principal *“Data Lake”*, creado luego de la ejecución de los procesos detallados en el capítulo 3; el mencionado directorio es el *Data Lake* nombrado “canvasuisek”, En la Figura 26 se presenta este directorio en la interfaz de *HUE*.

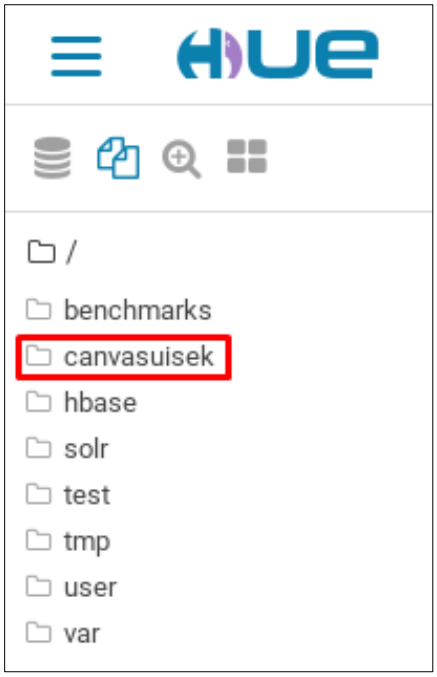


Figura 26 Consulta de directorio HDFS principal “canvasuisek” en HUE

4.2.1.2 Directorios internos del *Data Lake*

Como se mencionó en el capítulo 3, con la finalidad de que cada tabla tenga su propio directorio HDFS para que se realice la ingesta de datos de los archivos .csv correspondientes a cada una, se procedió a consultar que los mismos se encuentren creados.
En la Figura 27 se muestra los directorios HDFS que forman parte del *Data Lake* de la UISEK, en la interfaz de *HUE*.

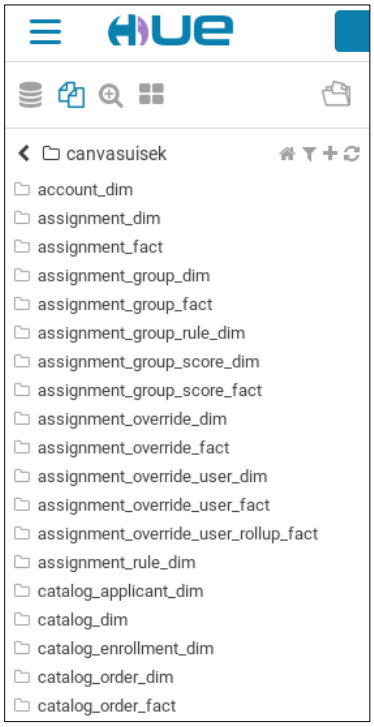


Figura 27 Muestra de directorios internos HDFS del Data Lake "canvasuisek" en HUE

4.2.1.3 Muestra de directorios HDFS del *Data Lake*

Una vez creado el *Data Lake*, se consultó los archivos .csv que se colocaron en los directorios HDFS *account_dim*, *course_dim* y *requests* en el proceso de ingesta de datos, respectivamente. En la Figura 28, 29 y 30 se muestra los resultados en la interfaz de *HUE*.

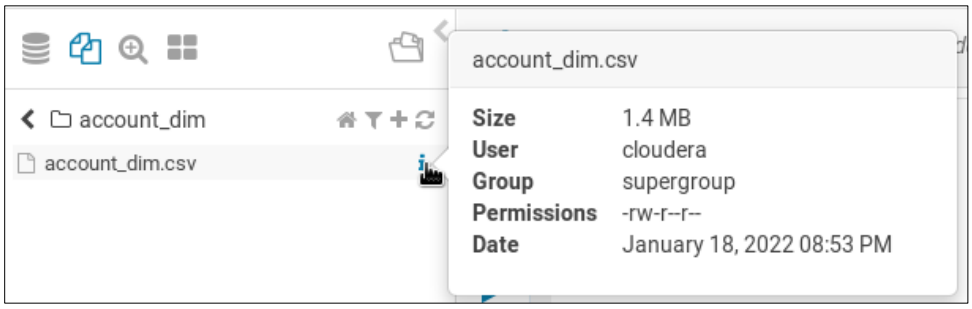


Figura 28 Consulta del archivo .csv del directorio account_dim en HUE

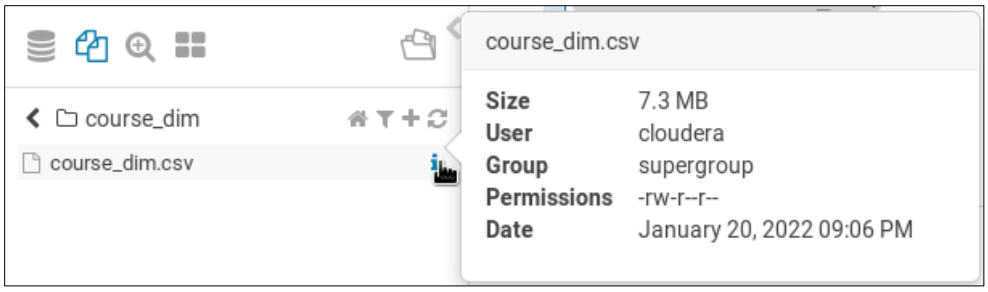


Figura 29 Consulta del archivo .csv del directorio course_dim en HUE

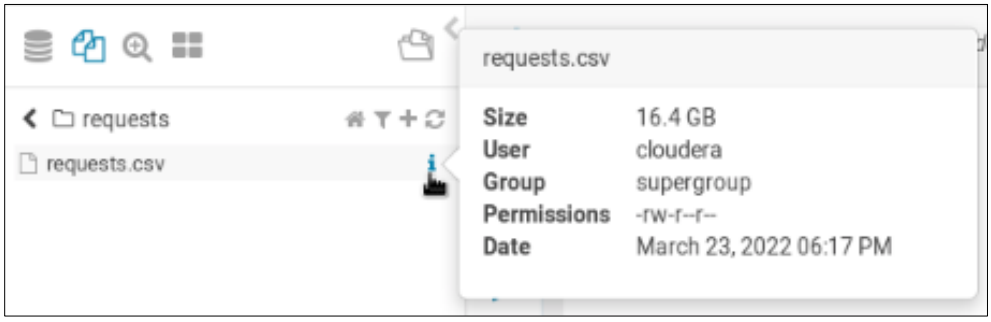


Figura 30 Consulta del archivo .csv del directorio requests en HUE

4.2.2 Creación de la base de datos y tablas de Canvas en *Hive*

A continuación, se muestran los resultados obtenidos luego de que se ejecutó el archivo "canvasbdd.hql" descrito en el punto 3.2.3.2, con el cual se creó la base de datos que almacena todas las tablas de Canvas que contiene el *Data Lake*, lo cual se detalló en el punto 3.2.3.2.

4.2.2.1 Base de datos "canvasbdd"

Se procedió a consultar las bases de datos existentes en *Hive*, con la finalidad de verificar la creación de la base de datos "canvasbdd", la cual contiene la información de todas las tablas de Canvas UISEK. La Figura 31 presenta la base de datos creada en la interfaz de *HUE*.

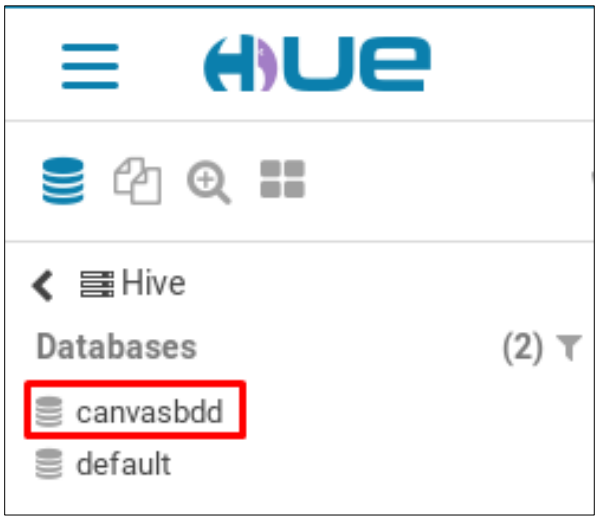


Figura 31 Evidencia creación BDD "canvasbdd" en HUE

4.2.2.2 Tablas de la base de datos “canvasbdd”

Como resultado de la ejecución de los procesos descritos en el capítulo 3, se consultó las tablas de la base de datos “canvasbdd” creada. Se puede visualizar que existen las 117 tablas correspondientes a la información que se obtuvo del último *Dump* disponible de Canvas de la UISEK. En la Figura 32 se muestra los resultados de esta consulta en la interfaz de *HUE*.

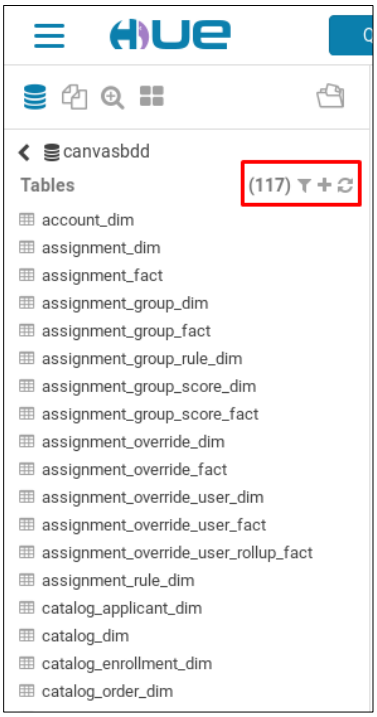
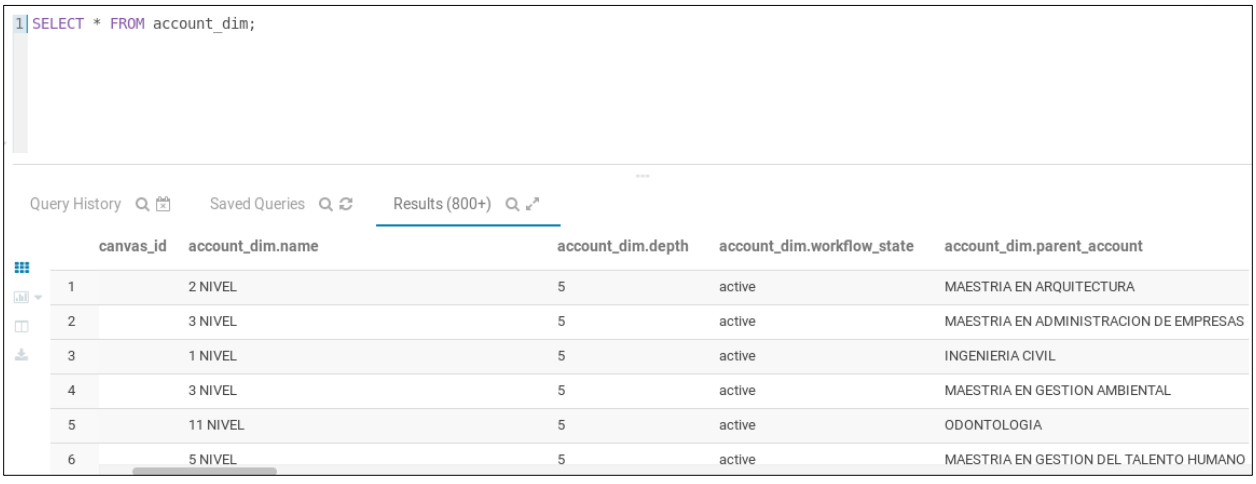


Figura 32 Consulta de tablas de la base de datos “canvasbdd” en HUE

4.2.2.3 Muestra de tablas de la base de datos “canvasbdd”

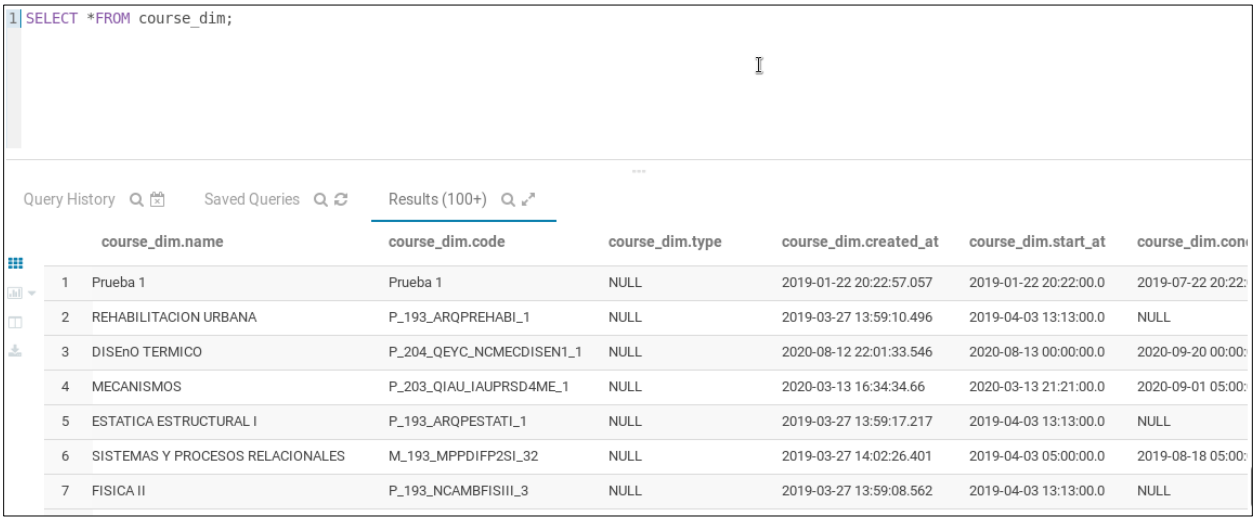
Con el objetivo de visualizar la información de las tablas *account_dim*, *course_dim* y *requests*, se ejecutó la sentencia *select* de cada una. En la Figura 33 se muestra los resultados del *select* realizado a la tabla *account_dim*, en la interfaz de HUE.



	canvas_id	account_dim.name	account_dim.depth	account_dim.workflow_state	account_dim.parent_account
1		2 NIVEL	5	active	MAESTRIA EN ARQUITECTURA
2		3 NIVEL	5	active	MAESTRIA EN ADMINISTRACION DE EMPRESAS
3		1 NIVEL	5	active	INGENIERIA CIVIL
4		3 NIVEL	5	active	MAESTRIA EN GESTION AMBIENTAL
5		11 NIVEL	5	active	ODONTOLOGIA
6		5 NIVEL	5	active	MAESTRIA EN GESTION DEL TALENTO HUMANO

Figura 33 Select de la tabla account_dim en HUE

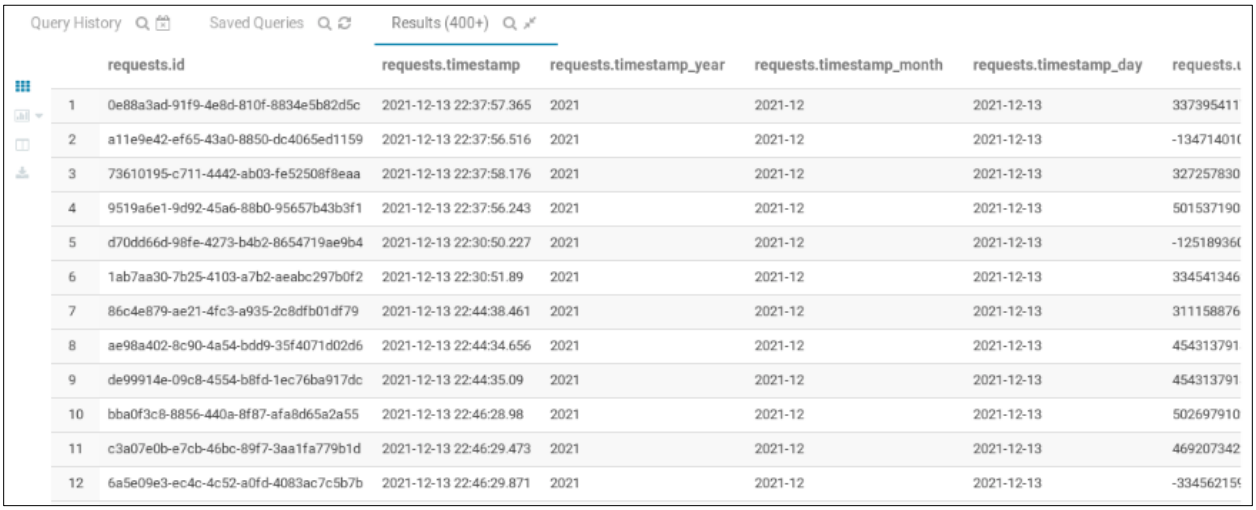
En la Figura 34 se muestra los resultados del *select* realizado a la tabla *course_dim*, en la interfaz de HUE.



	course_dim.name	course_dim.code	course_dim.type	course_dim.created_at	course_dim.start_at	course_dim.con
1	Prueba 1	Prueba 1	NULL	2019-01-22 20:22:57.057	2019-01-22 20:22:00.0	2019-07-22 20:22:
2	REHABILITACION URBANA	P_193_ARQPREHABI_1	NULL	2019-03-27 13:59:10.496	2019-04-03 13:13:00.0	NULL
3	DISEnO TERMICO	P_204_QEYC_NCMECDISEN1_1	NULL	2020-08-12 22:01:33.546	2020-08-13 00:00:00.0	2020-09-20 00:00:
4	MECANISMOS	P_203_QIAU_IAUPRSD4ME_1	NULL	2020-03-13 16:34:34.66	2020-03-13 21:21:00.0	2020-09-01 05:00:
5	ESTATICA ESTRUCTURAL I	P_193_ARQPESTATI_1	NULL	2019-03-27 13:59:17.217	2019-04-03 13:13:00.0	NULL
6	SISTEMAS Y PROCESOS RELACIONALES	M_193_MPPDIFP2SI_32	NULL	2019-03-27 14:02:26.401	2019-04-03 05:00:00.0	2019-08-18 05:00:
7	FISICA II	P_193_NCAMBFISIII_3	NULL	2019-03-27 13:59:08.562	2019-04-03 13:13:00.0	NULL

Figura 34 Select de la tabla course_dim en HUE

En la Figura 35 se muestra los resultados del *select* realizado a la tabla *requests*, en la interfaz de HUE.



	requests.id	requests.timestamp	requests.timestamp_year	requests.timestamp_month	requests.timestamp_day	requests.t
1	0e88a3ad-91f9-4e8d-810f-8834e5b82d5c	2021-12-13 22:37:57.365	2021	2021-12	2021-12-13	337395411
2	a11e9e42-ef65-43a0-8850-dc4065ed1159	2021-12-13 22:37:56.516	2021	2021-12	2021-12-13	-134714010
3	73610195-c711-4442-ab03-fe52508f8eaa	2021-12-13 22:37:58.176	2021	2021-12	2021-12-13	327257830
4	9519a6e1-9d92-45a6-88b0-95657b43b3f1	2021-12-13 22:37:56.243	2021	2021-12	2021-12-13	501537190
5	d70dd66d-98fe-4273-b4b2-8654719ae9b4	2021-12-13 22:30:50.227	2021	2021-12	2021-12-13	-125189360
6	1ab7aa30-7b25-4103-a7b2-aeabc297b0f2	2021-12-13 22:30:51.89	2021	2021-12	2021-12-13	334541346
7	86c4e879-ae21-4fc3-a935-2c8dfb01df79	2021-12-13 22:44:38.461	2021	2021-12	2021-12-13	311158876
8	ae98a402-8c90-4a54-bdd9-35f4071d02d6	2021-12-13 22:44:34.656	2021	2021-12	2021-12-13	454313791
9	de99914e-09c8-4554-b8fd-1ec76ba917dc	2021-12-13 22:44:35.09	2021	2021-12	2021-12-13	454313791
10	bba0f3c8-8856-440a-8f87-afa8d65a2a55	2021-12-13 22:46:28.98	2021	2021-12	2021-12-13	502697910
11	c3a07e0b-e7cb-46bc-89f7-3aa1fa779b1d	2021-12-13 22:46:29.473	2021	2021-12	2021-12-13	469207342
12	6a5e09e3-ec4c-4c52-a0fd-4083ac7c5b7b	2021-12-13 22:46:29.871	2021	2021-12	2021-12-13	-334562159

Figura 35 Select de la tabla requests en HUE

4.3 Consulta de datos en entorno gráfico de *HUE*

Desde *HUE* se realizó la ejecución de un *select* a la tabla *"account_dim",* para visualizar los resultados de esta consulta.

En la Figura 36 se presenta el *script* ejecutado para la obtención de resultados de la Maestría en Sistemas de Información Mención en *Data Science*, así como de la Maestría en Ciberseguridad.

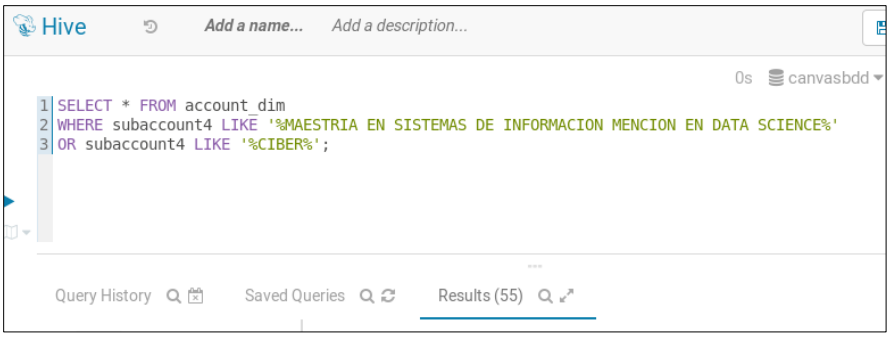


Figura 36 Estructura de Select a la tabla account_dim

En la Figura 37 se presenta los resultados gráficos del *select* ejecutado, analizando los resultados de la Maestría en Sistemas de Información Mención en *Data Science*, se puede observar que al comparar el período académico 2021-3 vs 2022-1 se obtiene ligeramente un mayor número de cuentas "*account*" para esta maestría en el periodo 2022-1, lo que podría significar que la Maestría en Sistemas de Información Mención en *Data Science* que oferta la UISEK cada vez tiene más acogida en la comunidad al momento de elegir un Postgrado.

Figura 37 Comparativa Maestría en Sistemas de Información Mención en Data Science

En la Figura 38 se presenta los resultados gráficos del *select* ejecutado, analizando los resultados de la Maestría en Ciberseguridad, se puede observar que al comparar el período académico 2021-3 vs 2022-1 se obtiene significativamente un mayor número de cuentas "*account*" para esta maestría en el periodo 2022-1, lo que podría significar que la Maestría en Ciberseguridad que oferta la UISEK tiene gran acogida en la comunidad al momento de elegir un Postgrado y posiblemente esta maestría sea considerada entre las favoritas de su rama en comparación a la que se oferta en otras universidades.

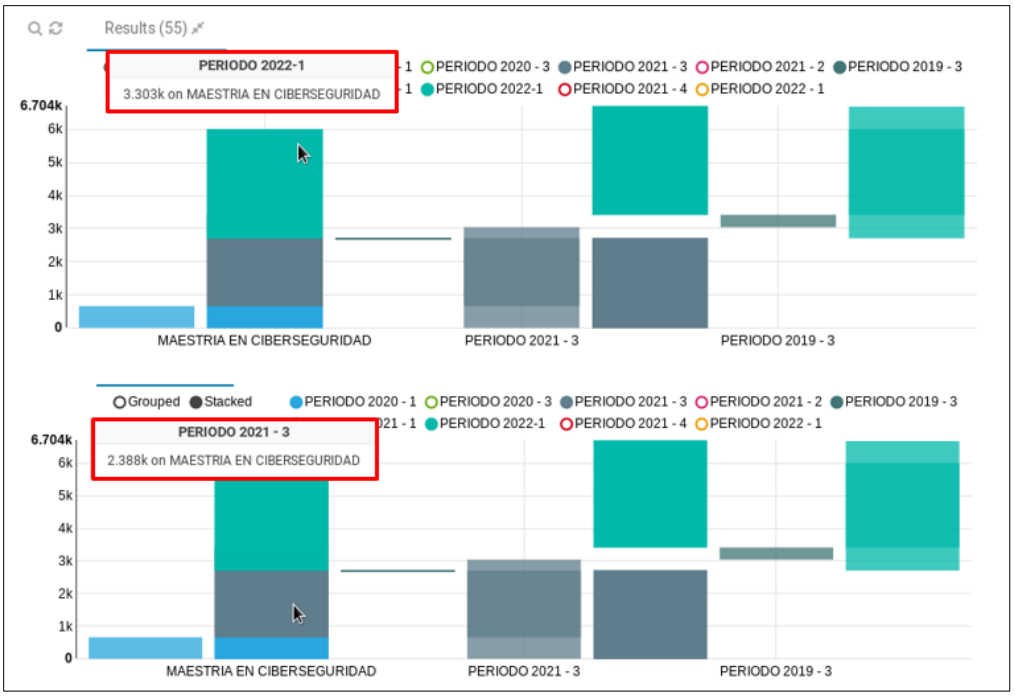


Figura 38 Comparativa Maestría en Ciberseguridad

CAPÍTULO 5

CONCLUSIONES Y TRABAJOS FUTUROS

Se diseñó e implementó una arquitectura del *Data Lake* de los datos de uso del LMS Canvas de la Universidad Internacional SEK, con información del último volcado de datos de Canvas. Para lo cual se aplicó *Big Data* con herramientas del ecosistema de *Hadoop*, esta solución permitió almacenar los archivos correspondientes a cada tabla de Canvas en el D*ata Lake*.

Para realizar la conexión a Canvas, se utilizó un API en *Python* que permitió realizar la descarga de las tablas que se encuentran en el último dump, de la última versión del esquema disponible se solicitó al administrador de Canvas de la UISEK las credenciales *API_KEY* y *API_SECRET*.

Únicamente en el caso de la tabla *Requests*, se descargó su información de los 100 *dumps* disponibles en Canvas para contar con la mayor cantidad de datos posibles. Sin embargo, el periodo al que corresponden estos datos es desde diciembre del 2021 hasta la fecha en que se ejecuta el "crontab", es decir hasta la fecha actual de cada día.

Al momento la tabla de *Requests* es la única tabla de Canvas que proporciona datos históricos. Cada día realiza una exportación de datos acumulados de hasta 48 horas, sin embargo, al sobrepasar este tiempo la tabla se sobrescribe con nueva información, por ello para este caso particular se maneja la concatenación de los archivos correspondientes a esta tabla por cada descarga de información de los *dumps*.

Se debe considerar que la "*Requests"* al contener datos históricos, es más pesada que las otras, y al concatenar los archivos su tamaño aumentará rápidamente. Por ello se debería optar por aumentar almacenamiento y memoria en Cloudera para que el proceso descrito se ejecute sin limitaciones.

Mediante la ejecución de tarea automática "crontab", que se programó para ejecutar los scripts descritos en los capítulos 3 y 4 respectivamente, se creó los directorios HDFS *(Data Lake*), la base de datos y tablas en *Hive* y se realizó la ingesta de estos datos en el *Data Lake*. De esta manera este repositorio será actualizado todos los días, sin pérdida de la información.

Al contar con el *Data Lake* de los datos de uso del LMS Canvas, la Universidad Internacional SEK puede aplicar los principios de *Big Data* y trabajar con grandes volúmenes de información que se generen a partir de las tablas que proviene de Canvas. Esto permitirá mejorar los procesos actuales de evaluación tanto a estudiantes como a docentes, generando respuestas acordes a la realidad del rendimiento de estos actores.

Se debería generar la continuación de este trabajo de investigación, en un proyecto de *Big Data* a gran escala donde se pueda trabajar con toda la información de la tabla *Requests*, lo cual conllevaría un tiempo considerable de investigación y análisis, además de requerir el soporte del personal administrador de Canvas de la UISEK e *Instructure* y profesionales/estudiantes de *Data Science*.

A partir de la presente investigación, se pueden desencadenar futuros proyectos relacionados a *Deep Learning*, Minería de Datos y modelos de Inteligencia Artificial, que aporten a realizar mejoras en la calidad de la educación de la UISEK, en función de beneficios transversales para el desarrollo académico y pedagógico.

Además, HUE puede usarse como una herramienta simple para generación de gráficos y visualización de la información que contiene cada tabla de la base de datos, sin embargo, no es un *software* avanzado para generación de reportes. Para que la información del *Data Lake* se plasme en reportes gerenciales se podría hacer uso de herramientas como *Power BI, Tableau*, entre otros según la necesidad de la UISEK.

BIBLIOGRAFÍA

Amazon Web Services, Inc. 2022. "What Is a Data Lake?" Retrieved January 26, 2022 (https://aws.amazon.com/es/big-data/datalakes-and-analytics/what-is-a-data-lake/).

Apache Software, Foundation. 2021. "Apache Hadoop 3.3.1 – HDFS Architecture." Retrieved January 26, 2022 (https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Introduction).

Arsys. 2017. "Canvas LMS, Una Moderna Plataforma de e-Learning Fácil de Utilizar y Personalizar - Blog de Arsys.Es." Retrieved February 6, 2022 (https://www.arsys.es/blog/soluciones/canvas-lms-cloud).

Baig, Maria Ijaz, Liyana Shuib, and Elaheh Yadegaridehkordi. 2020. "Big Data in Education: A State of the Art, Limitations, and Future Research Directions." *International Journal of Educational Technology in Higher Education* 17(1). doi: 10.1186/s41239-020-00223-0.

Baker, Reg. 2017. "Big Data: A Survey Research Perspective." *Total Survey Error in Practice* 47–69.

Camargo-Vega, Juan José, Jonathan Felipe Camargo-Ortega, Luis Joyanes-Aguilar, Juan José Camargo-Vega -, Jonathan Felipe, and Camargo-Ortega-Luis Joyanes-Aguilar. 2015. "Conociendo Big Data." *Revista Facultad de Ingeniería* 24(38):63–77.

Cen, L., D. Ruta, and J. Ng. 2015. "Big Education: Opportunities for Big Data Analytics." Pp. 502–6 in *International Conference on Digital Signal Processing, DSP*. Vols. 2015-Septe.

Chaurasia, Sushil S., Devendra Kodwani, Hitendra Lachhwani, and Manisha Avadhut Ketkar. 2018. "Big Data Academic and Learning Analytics: Connecting the Dots for Academic Excellence in Higher Education." *International Journal of Educational Management* 32(6):1099–1117. doi: 10.1108/IJEM-08-2017-0199.

Cloudera, and Apache Hadoop. 2021. "CDH Components." Retrieved (https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html).

Cloudera, and Hue. 2022. "Hue: El Asistente SQL de Código Abierto Para Almacenes de Datos." Retrieved January 27, 2022 (https://gethue.com/).

Cloudera, Inc. 2021. "Plataforma de Nube Híbrida Diseñada Para Cualquier Nube, Cualquier Análisis y Cualquier Conjunto de Datos." Retrieved (https://es.cloudera.com/products.html).

Cloudera, Inc. 2021. "Cloudera - Ecosistema de Apache Hadoop." *Https://Es.Cloudera.Com/Products/Open-Source/Apache-Hadoop.Html*.

Clow, Doug. 2013. "An Overview of Learning Analytics." *Https://Doi.Org/10.1080/13562517.2013.827653* 18(6):683–95. doi: 10.1080/13562517.2013.827653.

Confluence Administrator. 2020. "Apache Hive." Retrieved January 26, 2022 (https://cwiki.apache.org/confluence/display/Hive/Home#Home-ApacheHive).

Dhankhar, Amita, and Kamna Solanki. 2020. "State of the Art of Learning Analytics in Higher Education." *International Journal of Emerging Trends in Engineering Research* 8(3):868–77. doi: 10.30534/ijeter/2020/43832020.

Fernández, Óscar. 2021a. "Apache Hadoop." Retrieved February 6, 2022

(https://aprenderbigdata.com/hadoop/).

Fernández, Óscar. 2021b. “Apache Hive.” Retrieved February 6, 2022 (https://aprenderbigdata.com/apache-hive/).

Fernández, Óscar. 2021c. “Introducción a Cloudera y Componentes.” Retrieved February 3, 2022 (https://aprenderbigdata.com/introduccion-cloudera-hadoop/).

iLifebelt. 2022. “Las 4 V’s Del Big Data.” Retrieved January 27, 2022 (https://ilifebelt.com/las-4-vs-del-big-data-latinoamerica/2017/08/).

Instructure. 2021. “Canvas - Higher Ed | Instructure.” Retrieved January 27, 2022 (https://www.instructure.com/es/canvas/educacion-superior).

Instructure. n.d. “Canvas Data Portal.” Retrieved January 26, 2022 (https://portal.inshosteddata.com/docs).

Instructure, Inc. 2021. “CANVAS Architecture Overview.” (January).

Li, Yi, and Xiaoning Zhai. 2018. “Review and Prospect of Modern Education Using Big Data.” *Procedia Computer Science* 129:341–47. doi: 10.1016/j.procs.2018.03.085.

López Taboada, Guillermo, and Rubén F. Casal. 2021. “Tecnologías Big Data (Hadoop/Spark y Visualización) | Prácticas de Tecnologías de Gestión y Manipulación de Datos.” Retrieved January 27, 2022 (https://gltaboada.github.io/tgdbook/tecnologías-big-data-hadoopspark-y-visualización.html).

Miloslavskaya, Natalia, and Alexander Tolstoy. 2016. “Big Data, Fast Data and Data Lake Concepts.” *Procedia Computer Science* 88:300–305. doi: 10.1016/J.PROCS.2016.07.439.

Mohamad, Siti Khadijah, and Zaidatun Tasir. 2013. “Educational Data Mining: A Review.” *Procedia - Social and Behavioral Sciences* 97:320–24. doi: 10.1016/J.SBSPRO.2013.10.240.

Python. 2019. “BeginnersGuide/Overview - Python Wiki.” Retrieved January 26, 2022 (https://wiki.python.org/moin/BeginnersGuide/Overview).

Red Hat, Inc. 2022. “¿Qué Es Un Lago de Datos?” Retrieved January 26, 2022 (https://www.redhat.com/es/topics/data-storage/what-is-a-data-lake).

Ribeiro, André, Afonso Silva, and Alberto Rodrigues da Silva. 2015. “Data Modeling and Data Analytics: A Survey from a Big Data Perspective.” *Journal of Software Engineering and Applications* 08(12):617–34. doi: 10.4236/jsea.2015.812058.

Telefónica Educación Digital. n.d. “‘Big Data’ En Educación: Un Tesoro Para La Toma de Decisiones.” Retrieved January 26, 2022 (https://www.telefonicaeducaciondigital.com/tendencias/-/asset_publisher/LTIINEKg9l8P/content/-big-data-en-educacion-un-tesoro-para-la-toma-de-decisiones).

Valverde, J. 2016. “La Investigación En Tecnología Educativa y Las Nuevas Ecologías Del Aprendizaje.” *Revista Interuniversitaria de Investigación En Tecnología Educativa (RIITE)* (0):60–73. doi: 10.6018/riite/2016/257931.

ANEXOS

5.1Anexo A

```
# Script de ejecución principal

# *** Se declaran las variables de entorno ***

export API_KEY=''

export API_SECRET=''

echo "DECLARACION DE VARIABLES"

# *** Cambio a directorio de trabajo ***

cd /home/cloudera/UISEK/

echo "CAMBIO DIRECTORIO UISEK"

# *** eliminación de archivos descargados ***

rm -rf data

echo "ELIMINACION DE ARCHIVOS"

# *** ejecución archivo py para creación de scripts, conectar y descargar lastDumb de canvas
***

/home/cloudera/anaconda3/envs/canvasprueba/bin/python3 /home/cloudera/UISEK/CanvasLog.py

echo "EJECUCION DE CanvasLog"

cd /home/cloudera/UISEK/data/scripts/

# *** Otorga permiso de ejecución

chmod +x ./canvasuisek.sh

# *** ejecución script para generar hdfs

./canvasuisek.sh

# *** Ejecución de comandos hql para ingesta de las tablas de Hive ***

cd /home/cloudera/UISEK/data/scripts/

hive -f canvasbdd.hql

echo "EJECUCION DE ARCHIVO .hql"
```

5.2Anexo B

```
#!C:\Users\tefon\anaconda3\python.exe
from canvas_data.api import CanvasDataAPI
import gzip
import os
import gc
import pprint
from pathlib import Path


pp = pprint.PrettyPrinter(indent=4)


try:
    API_URL = 'https://uisek.instructure.com'
    API_KEY = os.environ['API_KEY']
    API_SECRET = os.environ['API_SECRET']
except KeyError:
    print("You must set both the API_KEY and API_SECRET environment variables.")
    exit()


cd = CanvasDataAPI(api_key=API_KEY, api_secret=API_SECRET)
schema_versions = cd.get_schema_versions()
print('Found {} schema versions.'.format(len(schema_versions)))
schema = cd.get_schema('latest', key_on_tablenames=True)
# Definición de los directorios a trabajar
principalDirectory = "./data/lastDump"
Path(principalDirectory).mkdir(parents=True, exist_ok=True)
processDirectory = "./data/process"
Path(processDirectory).mkdir(parents=True, exist_ok=True)
decompressFiles = processDirectory + "/csvFiles"
Path(decompressFiles).mkdir(parents=True, exist_ok=True)
decompressFilesTxt = processDirectory + "/lastDumpDecompressedTxt"
Path(decompressFilesTxt).mkdir(parents=True, exist_ok=True)
requestDirectory = "./request"
Path(requestDirectory).mkdir(parents=True, exist_ok=True)
```

```
# # Definicion de las extensiones de archivo a usar
gzFile = ".gz"
txtFile = ".txt"
csvFile = ".csv"
"""
Se extraé infromación del diccionario de esquema
"""
# Se define un directorio general para almacenar los scripts
principalScriptDirectory = "./data/scripts/"
# Si no existe, se crea el directorio anterior
Path(principalScriptDirectory).mkdir(parents=True, exist_ok=True)
# Nombre del data lake
dataLakeName = 'canvasuisek'
# Variable para almacenar script hdfs
scriptHDFS = '\n'
# Se ubica en el directorio para crear el datalake
scriptHDFS += 'cd /home/cloudera/UISEK\n'
# Comando para salir del modo seguro
scriptHDFS += 'sudo -u hdfs hadoop dfsadmin -safemode leave\n'
# Comando para eliminar el directorio HDFS (dataLake) canvasuisek
scriptHDFS += 'hdfs dfs -rm -r /' + dataLakeName + '\n'
# Comando para crear el directorio HDFS (dataLake) canvasuisek
scriptHDFS += 'hdfs dfs -mkdir /' + dataLakeName + '\n'
# Nombre de la base de datos
dataBaseName = "canvasbdd"
# Variable para almacenar la información del script
scriptCarga = '\n'
# Creamos database si no existe
scriptCarga = 'create database if not exists ' + dataBaseName + ';\n'
# Se inicia el script, describiendo que data base usar
scriptCarga += "use " + dataBaseName + ';\n'
# Se extraé información de los keys, es decir del nombre de cada tabla perteneciente al schema
new_listKeys = list(schema.keys())
```

```
"""Se crea un directorio independiente para cada tabla, en el que se guardará la información del nombre,
 tipo y descripción de cada campo o columna"""
# Se define un directorio general para esta información
principalTableDirectory = "./data/tables/"
# Secuencia variable control para determinar la iteración
secuencia = 0
# Se recorre el array de keys(nombres de tablas)
for key in new_listKeys:
    # Se crea el directorio hdfs para cada tabla
    scriptHDFS += 'hdfs dfs -mkdir /' + dataLakeName + '/' + str(key) + '\n'
    # Se cargan los archivos .csv en cada directorio interno por tabla
    scriptHDFS += 'hdfs dfs -put ' + '/home/cloudera/UISEK/data/process/csvFiles/' + str(key) + '.csv /'+
dataLakeName + '/' + str(key) + '\n'
    # Se crea el directorio hdfs para cada tabla
    scriptHDFS += 'echo ' + str(secuencia) + ' de ' + str(len(new_listKeys)) + '\n'
    secuencia += 1
    # Se crea la tabla si existe
    scriptCarga += '\ncreate external table if not exists ' + str(key) + '('
    # Si no existe, se crea un directorio por cada nombre de tabla, si ya existe, no hace nada
    Path(principalTableDirectory+str(key)).mkdir(parents=True, exist_ok=True)
    # Se extrae del diccionario de esquema, la información de las columnas pertenecientes a cada tabla
    new_listItemsSchema = list(schema[str(key)]['columns'])
    # Se definen variables string generales para ir concatenando las propiedades de cada columna
    campoType = ""
    campoName = ""
    campoDescription = ""
    # Se define variable control, para identificar la iteración dentro del array de campos de cada tabla
    camposLength = 0
    # Se crea un archivo vacío para cada tabla
    fcsv = open(decompressFiles+"/" + str(key) + ".csv", "w", encoding='utf-8')
    fcsv.write('')
    fcsv.close()
```

```
# Se recorre el array con la información de cada columna(campo)

    for campo in new_listItemsSchema:

# Se identifica la iteración del array

        camposLength += 1

#Controlar el tipo de dato enum y double precision

        if campo['type'] == 'double precision':

            campo['type'] = 'double'

        if campo['type'] == 'enum':

            campo['type'] = 'string'

        if campo['type'] == 'varchar':

            campo['type'] = 'string'

        if campo['type'] == 'text':

            campo['type'] = 'string'

        if campo['type'] == 'integer':

            campo['type'] = 'int'

        if campo['type'] == 'guid':

            campo['type'] = 'string'

        if campo['type'] == 'datetime':

            campo['type'] = 'timestamp'

# Se concatena la información de cada campo, para formar un solo documento

        campoName += campo['name']+" "

        campoType += (campo['type'])+" "

# Se controla si es la última iteración del array

        if camposLength == len(new_listItemsSchema):

# Se agrega la información de cada campo y tipo de dato respectivo a cada tabla,

# si es la última iteración, significa que es el último campo para registrar, por lo que se debe finalizar
con )

            scriptCarga += '\n' + str(campo['name']) + ' ' + str(campo['type']) + ')'

        else:

# Se agrega la información de cada campo y tipo de dato respectivo a cada tabla y se continua con una

# (,) para avanzar con el siguiente campo

            scriptCarga += '\n' + str(campo['name']) + ' ' + str(campo['type']) + ','

# Se crea un archivo .txt, para almacenar la información del nombre de campo de cada tabla

        fp = open(principalTableDirectory + str(key) + "/" + str(key) + "_name.txt", "w")

        fp.write(str(campoName))

        fp.close()
```

```
# Se crea un archivo .txt, para almacenar la información del tipo de dato de cada tabla
        fp = open(principalTableDirectory + str(key) + "/" + str(key) + "_type.txt", "w")
        fp.write(str(campoType))
        fp.close()
        # Se crea un archivo .txt, para almacenar la información de la descripción de cada tabla
        # fp = open(principalTableDirectory + str(key) + "/" + str(key) + "_description",
        #          "w")  # Crea un archivo txt en el que se descomprime
        # fp.write(str(campoDescription))
        # fp.close()
    scriptCarga += "\nrow format delimited fields terminated by ','\nstored as textfile"
    scriptCarga += "\nlocation '/" + dataLakeName + "/" + str(key) + "'; \n"
# Se crea un archivo .hql, para almacenar el script para ingesta de datos
fp = open(principalScriptDirectory + dataBaseName + ".hql", "w")
fp.write(scriptCarga)
fp.close()
# Se crea un archivo .sh, para almacenar el script para la carga hdfs
fp = open(principalScriptDirectory + dataLakeName + ".sh", "w")
fp.write(scriptHDFS)
fp.close()
print('Scripts generated successfull')

""" Optimización uso de memoria"""
# Se elimina variable fp
del fp
# Se elimina toda la basura en memoria
gc.collect(generation=2)
"""Inicio de descarga"""
print('The latest schema has {} tables.'.format(len(schema)))
dumps = cd.get_dumps()
print('There are a total of {} dumps available'.format(len(dumps)))
one_dump = dumps[0]
dump_files = cd.get_file_urls(dump_id=one_dump['dumpId'])
dump_table_names = dump_files['artifactsByTable'].keys()
print('The dump ID {} contains files for {} tables.'.format(one_dump['dumpId'], len(dump_table_names)))
```

```
# are there tables that are in the schema but not in the dump, or vice-versa?
not_in_dump = [x for x in schema.keys() if x not in dump_table_names]
print('These tables are present in the schema but missing from the dump: {}'.format(not_in_dump))
cont=0
dumps_aux = dumps[::-1]
if len(data)==0:
    print('Descarga de últimos dumps tabla request')
    for i in dumps_aux:
        cont +=1
        if cont == 99:
            print("Se ha finalizado la descarga de la tabla requests de los 99 dumps anteriores al last ")
            break
        print('The dump ID {}'.format(i['dumpId']))
        latest_requests_files = cd.download_files(dump_id=i['dumpId'], table_name='requests',
                                                  download_directory='./requests/allDumps')
        content = os.listdir('./requests/allDumps')
        print(str(cont) + " de "+ str(len(dumps_aux)-1))
        fp = open('./requests/allDumps/requests.txt', "wb")  # Crea un archivo txt en el que se descomprime
        with gzip.open('./requests/allDumps/' + str(content[0]), "rb") as f:
            bindata = f.read()
        fp.write(bindata)
        fp.close()
        fp = open('./requests/allDumps/requests.txt', "r", encoding='utf-8')
        data = fp.read()
        data = data.replace(",", ";")
        data = data.replace("\t", ",")
        fp.close()
        fp = open('./requests/requests.txt', "a")  # Crea un archivo txt en el que se descomprime
        fp.write(data)
        fp.close()
        # Se elimina variable fp
        del fp
        del data
```

```
# Se elimina toda la basura en memoria

        gc.collect(generation=2)

        rmtree('./requests/allDumps')

print("Se inicia la descarga de todas las tablas del last dump")

# get all of the files from the latest dump: (this will take a while to run)

latest_dump_files = cd.download_files(dump_id='latest', download_directory='./data/lastDump')

print('Latest course files: {}'.format(latest_dump_files))

print("Se finaliza la descarga de todas las tablas del last dump")

"""Descomprimir archivos"""

content = os.listdir(principalDirectory)

aux = []

for i in content:

    aux.append(i.split("-", 1)[0])

def decompressDump(path=None, pathToDecompressTxt=None, pathToDecompressCsv=None, aux=None,
requestDirectory=None):

    pathTxt = pathToDecompressTxt + ".txt"

    pathGz = path

    pathCsv = pathToDecompressCsv + ".csv"

    pathRequest = requestDirectory + "/"+ aux + ".txt"

    fp = open(pathTxt, "wb")  # Crea un archivo txt en el que se descomprime

    with gzip.open(pathGz, "rb") as f:

        bindata = f.read()

    fp.write(bindata)

    fp.close()

    fp = open(pathTxt, "r", encoding='utf-8')

    data = fp.read()

    data = data.replace(",", ";")

    data = data.replace("\t", ",")

    fp.close()

    if (aux == 'requests'):

        fp = open(pathRequest, "a",encoding='utf-8')  # Concatena cada iteracion

        fp.write(data)

        fp.close()
```

```
fr = open(pathRequest, "r",encoding='utf-8')  # Abre request almacenado

        fpr = open(pathTxt, "w", encoding='utf-8')

        data=fr.read()

        fpr.write(data)

        fpr.close()

        fr.close()

    fcsv = open(pathCsv, "w",encoding='utf-8')

    fcsv.write(data)

    fcsv.close()

    """Optimización uso de memoria """

    # Se elimina variable fp

    del fp

    del data

    del fcsv

    # Se elimina toda la basura en memoria

    gc.collect(generation=2)

    return 0

for i in range(len(aux)):


decompressDump(principalDirectory+"/"+content[i],decompressFilesTxt+"/"+aux[i],decompressFiles+"/"+aux[i],
aux[i],requestDirectory)

    print("\nArchivo Descomprimido:"+aux[i])

    print(" "+ str(i+1) +" Tablas descomprimidas de: " + str(len(aux)))
```