CARTO Big Data connectors

Versión 1.0.0

Alberto Romeu Carrasco

23 de junio de 2018

Contents:

CAPÍTULO 1

Introducción

1.1 Contexto

CARTO[1] es una plataforma de *Location Intelligence* que permite transformar datos geoespaciales en resultados de negocio.

Actualmente, las organizaciones que utilizan *CARTO* como herramienta de análisis geoespacial tienen multitud de fuentes de información y aplicaciones ya instaladas que generan continuamente nuevos datos.

El principal valor de *CARTO* para estas organizaciones es el de conectarse con esas fuentes de información (datos de *CRM*, *ERP*, hojas de cálculo, archivos con contenido geoespacial, bases de datos relacionales, etc.) a través de una interfaz sencilla e intuitiva, para generar nueva información de valor añadido para su negocio mediante análisis geoespaciales y visualizaciones.

En determinadas organizaciones, especialmente de tamaño medio o grande, ocurre que diversos equipos gestionan sus datos con sistemas de información heterogéneos que utilizan repositorios de datos tales como:

- Hojas de cálculo y archivos CSV (valores separados por comas)
- Documentos de Google Drive[2]
- CRMs tales como Salesforce[3]
- Herramientas de marketing digital como Mailchimp[4]
- Servidores de datos geoespaciales como ArcGIS[5]
- Bases de datos operacionales (relacionales o *NoSQL*)
- Sistemas de ficheros distribuidos *HDFS*

[1] https://www.carto.com - septiembre 2017
[2] https://drive.google.com - septiembre 2017
[3] https://www.salesforce.com - septiembre 2017
[4] https://mailchimp.com - septiembre 2017
[5] https://www.arcgis.com - septiembre 2017

- Otros sistemas (Twitter[6], Dropbox[7], etc.)

1.2 Definición del problema

Estas organizaciones utilizan *CARTO* para importar sus datos y analizar la información que generan los distintos departamentos de manera conjunta, respondiendo a sus preguntas de negocio y encontrando además respuesta a otras preguntas que no se habían planteado en un principio.

CARTO cuenta con la posibilidad de importar datos desde diversas fuentes de datos, pero carece de soporte nativo para conectar a muchos de estos sistemas de almacenamiento Big Data usados generalmente para almacenar datos operacionales o secuencias de datos temporales.

1.3 Objetivo

El objetivo de este trabajo final de máster consiste en el desarrollo de conectores para *CARTO* que permitan incluir en los cuadros de mandos (*dashboard*), información proveniente de los siguientes sistemas de almacenamiento y/o procesamiento Big Data.

El objetivo es encontrar un mecanismo fácilmente reproducible que permita en el futuro integrar otros sistemas de almacenamiento. Para el actual trabajo, el objetivo consiste en integrar al menos:

- Apache Hive
- Impala
- MongoDB
- Google BigQuery

Y describir un proceso que permitiera la integración de otros sistemas tales como:

- Amazon Redshift
- Cassandra
- SparkSQL
- Amazon Aurora
- Oracle

Los «conectores Big Data para *CARTO*» permitirán a las organizaciones mantener sus actuales flujos de ingestión y procesamiento de información, además de aprovechar lo mejor de dos mundos: el almacenamiento y procesamiento distribuido que ofrecen algunas de estas herramientas orientadas a Big Data y la visualiación y análisis geoespacial de *CARTO*.

Cabe destacar que los resultados de este trabajo no son de carácter teórico, sino que consiste en código fuente y herramientas que se incluirán en la distribución *on-premise* de *CARTO*

1.4 Organización de este trabajo final de máster

Este trabajo final de máster está organizado en capítulos, siguiendo la siguiente estructura:

[6] https://www.twitter.com - septiembre 2017
[7] https://www.dropbox.com - septiembre 2017

1. *Estado del arte*: Se repasan las herramientas de almacenamiento y procesamiento Big Data con las que se va a trabajar y se definen algunos de los conceptos teóricos que sirven de fundamentación para el trabajo.
2. *Metodología y plan de trabajo*: Definición de una metodología de trabajo sistemática y desglose en tareas del trabajo a realizar.
3. *Desarrollo del trabajo y resultados obtenidos*: Descripción de la implementación de cada uno de los conectores, demostración de uso, etc.
4. *Conclusiones*
5. *Bibliografía*
6. *Anexos*
7. *Glosario*

Palabras clave: *BASH, Docker, Vagrant, Location Intelligence, AWS, HDFS, Hadoop, BigQuery, Hive, Impala, Spark, NoSQL, Cassandra, MongoDB, CARTO, dashboards, análisis geoespacial*

CAPÍTULO 2

Estado del arte

En este capítulo se presenta un informe sobre los diferentes sistemas de almacenamiento y procesamiento Big Data para los que se van a realizar conectores para CARTO y una breve definición de los conceptos teóricos que sirven de fundamentación para el trabajo.

2.1 CARTO

CARTO[1] es una plataforma de *Location Intelligence* que permite transformar datos geoespaciales en resultados de negocio.

A diferencia del *Business Intelligence*, *Location Intelligence* es el conjunto de herramientas y metodologías que permiten extraer conocimiento y tomar decisiones de negocio a partir de datos geoespaciales.

CARTO es una plataforma *SaaS* referente en este sector que permite de una manera sencilla e intuitiva la importación de conjuntos de datos con información geoespacial para crear a través de *dashboard* y *widgets*, mapas con capacidades de análisis, filtrado, búsqueda y predicción de variables.

CARTO cuenta con la posibilidad de importar datos desde diversas fuentes de datos, pero carece de soporte nativo para conectar a algunos de los principales sistemas de almacenamiento Big Data usados generalmente para almacenar datos operacionales o secuencias de datos temporales.

Actualmente, *CARTO* cuenta con más de 300000 usuarios registrados y es usado por más de 2000 organizaciones de todo el mundo para tomar decisiones a partir de sus datos geolocalizados.

2.1.1 Productos

En la actualidad, CARTO está formado por 4 productos principales:

- BUILDER[2]

[1] https://carto.com/ - octubre 2017
[2] https://carto.com/builder/ - octubre 2017

Una herramienta web de análisis para que analistas y usuarios de negocio que permite crear cuadros de mando accionables que se pueden compartir.

- ENGINE[3]

Un conjunto de herramientas geoespaciales, servicios y APIs para el desarrollo de aplicaciones geoespaciales.

- Location Data Services[4]

Mapas base, mapas vectoriales, servicios de geocodificación y cálculo de rutas que se pueden consumir fácilmente en BUILDER o integrar a través de ENGINE.

- Data Observatory[5]

Servicios para enriquecimiento de datos, a través de fronteras, datos demográficos y otros conjuntos de datos para dar valor a los propios datos de los usuarios.

2.1.2 Arquitectura

El siguiente diagrama muestra la arquitectura de componentes de *CARTO*.

[TODO] -> añadir imagen

Tal y como se muestra en la figura, dependiendo del tipo de aplicación que un usuario consuma, se utilizan diferentes componentes de la arquitectura.

En el cuadro naranja, encontramos el stack de *CARTO* que se despliega en la nube y al cual accede un usuario que se registra en el *SaaS* a través de https://www.carto.com

Para el objetivo de este trabajo final de máster, vamos a obviar los casos de aplicaciones móviles o HTML5 y vamos a centrarnos en *BUILDER*.

BUILDER está formado por un conjunto de tecnologías de *backend*, que están desplegadas en la nube de Amazon, Google o Azure (u *on-premise*) y un conjunto de tecnologías de *frontend* que se corresponden con librerías JavaScript que se ejecutan en el navegador.

Dentro de las tecnologías *backend* encontramos las siguientes:

- PostgreSQL y PostGIS

PostgreSQL[6] es una base de datos relacional con soporte a SQL estándar distribuida con licencia libre y código abierto. PostGIS[7] es una extensión para PostgreSQL que añade soporte geoespacial a través de estructuras de datos (tipos, índices, etc.) y funciones.

CARTO utiliza PostgreSQL y PostGIS para almacenamiento de la información geoespacial generada por los usuarios y para realizar los análisis geoespaciales que permiten construir cuadros de mandos, visualizar mapas, etc.

El acceso a PostgreSQL y PostGIS está abierto a los usuarios a través del uso de las *API* de la plataforma.

Las version actuales de PostgreSQL y PostGIS utilizados por CARTO son la 10.0 y 2.4 respectivamente.

- APIs de la plataforma (maps, SQL, import, analysis, etc.)

Las APIs de la plataforma son parte de las APIs ofrecidas por *ENGINE* y utilizadas a su vez por *BUILDER* y por aplicaciones móviles o HTML5 creadas por terceros.

CARTO ofrece un conjunto amplio de APIs *REST*, JavaScript y *SDK* de desarrollo en diferentes lenguajes. A continuación se describen las más relevantes para el trabajo:

[3] https://carto.com/engine/ - octubre 2017
[4] https://carto.com/location-data-services/ - octubre 2017
[5] https://carto.com/data-observatory/ - octubre 2017
[6] https://www.postgresql.org/ - octubre 2017
[7] http://postgis.net/ - octubre 2017

- maps API: Permite obtener teselas de los datos almacenados en PostgreSQL
- SQL API: Permite realizar consultas SQL contra PostgreSQL y PostGIS y utilizar todas las funciones disponibles incluidas las de *Location Data Services* y *Data Obervatory*
- import API: Permite importar datos en formato geoespacial
- Varnish

Varnish[8] es un acelerador de aplicaciones web, también conocido como servidor proxy de caché HTTP. Permite cachear peticiones HTTP y su contenido.

- Nginx

Nginx[9] es un servidor web HTTP.

- CDN

Una Content Delivery Network (CDN o, en español, una "Red de distribución de contenido") es un conjunto de servidores que contienen copias de una misma serie de contenidos (imágenes, vídeos, documentos, ...) y que están ubicados en puntos diversos de una red para poder servir sus contenidos de manera más eficiente.[10]

- BUILDER

BUILDER es una aplicación escrita en Ruby on Rails y JavaScript, que a través de las APIs de la plataforma permite a los usuarios finales:

- Gestionar sus datos geoespaciales
- Gestionar sus mapas
- Definir orígenes de datos con filtros y consultas SQL
- Definir simbología a través de CartoCSS[11]
- Publicar mapas y embeberlos

Todo esto, centrado en la experiencia de usuario a través de una interfaz de usuario atractiva y fácil de usar.

2.2 Sistemas de almacenamiento y procesamiento Big Data

En este trabajo se estudian los siguientes sistemas de almacenamiento y procesamiento Big Data, ya que son los sistemas más utilizadas por las actuales organizaciones que usan *CARTO* como plataforma de *Location Intelligence*:

- Apache Hive[12]
- Apache Impala[13]
- Amazon Redshift[14]
- MongoDB[15]
- Google BigQuery[16]

[8] https://varnish-cache.org/ - octubre 2017
[9] https://nginx.org/ - octubre 2017
[10] https://manueldelgado.com/que-es-una-content-delivery-network-cdn/ - octubre 2017
[11] https://carto.com/docs/carto-engine/cartocss/ - octubre 2017
[12] https://hive.apache.org/ - octubre 2017
[13] https://impala.apache.org/ - octubre 2017
[14] https://aws.amazon.com/es/redshift/ - octubre 2017
[15] https://www.mongodb.com/ - octubre 2017
[16] https://cloud.google.com/bigquery/ - octubre 2017

En esta sección se va a hacer una breve descripción de los sistemas mencionados atendiendo a las siguientes características:

- Tipo de sistema: Si ofrece almacenamiento y procesamiento o sólo uno de ambos.
- Tipo de procesamiento: Batch (latencia del orden de minutos), interactivo (latencia del orden de decenas de segundos), tiempo real (latencia del orden de pocos segundos), etc.
- Tipo de despliegue/distribución: Nube pública, privada, SaaS, on-premises, etc.
- Interfaces de programación/consulta: SQL, SDKs en diferentes lenguajes, APIs REST, etc.
- Autenticación: Usuario y contraseña, HTTP/HTTPS, Kerberos/LDAP, OAuth, etc.
- Tipo de licencia/propietario: Software libre (Apache, GPL, etc.), propietaria (Google, Amazon, Oracle, etc.)
- Versión actual
- Driver ODBC

Para el motivo de este trabajo, no es necesario conocer otros detalles como mecanismos de replicación, particionamiento, tolerancia a fallos, etc. ya que el objetivo no consiste en administrar este tipo de sistemas.

Sin embargo, el objetivo es triple:

1. Por una parte, contar con una visión general de los sistemas con los que se va a trabajar.
2. Por otra parte, poder identificar similitudes y diferencias entre ellos.
3. Por último, abrir la puerta al soporte del mayor número posible de tecnologías de almacenamiento y procesamiento Big Data, especialmente aquellas de carácter libre.

2.2.1 Apache Hive

Apache Hive es una infraestructura de almacenamiento y procesamiento de datos almacenados sobre *HDFS* de Hadoop[17] y otros sistemas compatibles como Amazon S3[18], originalmente desarrollado por Facebook[19].

Ofrece un lenguaje de consulta basado en SQL llamado *HiveQL* que convierte las consultas en trabajos MapReduce, Tez[20] o Spark[21].

Actualmente, como gran parte de los sistemas batch es considerado un sistema *legacy*, aunque por otra parte es un sistema apliamente establecido en la industria que cuenta con gran cantidad de herramientas integradoras dentro del sistema Hadoop tales como: Pig[22], Sqoop[23], Flume[24], etc.

Se suele utilizar para procesamiento batch de ficheros almacenados en HDFS.

- Tipo de sistema: Procesamiento.
- Tipo de procesamiento: Batch.
- Tipo de despliegue/distribución: Nube pública y privada (on-premises) con multitud de distribuciones (Amazon EMR[25], Cloudera[26], Hortonworks[27], MapR[28])

[17] http://hadoop.apache.org/ - octubre 2017
[18] https://aws.amazon.com/es/s3/ - octubre 2017
[19] https://facebook.com/ - octubre 2017
[20] https://tez.apache.org/ - octubre 2017
[21] https://spark.apache.org/ - octubre 2017
[22] https://pig.apache.org/ - octubre 2017
[23] https://sqoop.apache.org/ - octubre 2017
[24] https://flume.apache.org/ - octubre 2017
[25] https://aws.amazon.com/es/emr/ - octubre 2017
[26] https://www.cloudera.com - octubre 2017
[27] https://es.hortonworks.com/ - octubre 2017
[28] https://mapr.com/ - octubre 2017

- Interfaces de programación/consulta: HiveQL compatible con SQL
- Autenticación: Usuario y contraseña, HTTP/HTTPS, Kerberos/LDAP
- Tipo de licencia/propietario: Apache 2.0
- Versión actual: 2.3.0
- Driver ODBC: sí

2.2.2 Impala

Apache Impala es una infraestructura de almacenamiento y procesamiento de datos almacenados sobre HDFS de Hadoop, originalmente desarrollado por Cloudera.

Apache Impala es compatible con HiveQL y utiliza la misma base de datos de metadatos para acceder a HDFS que Hive, pero a diferencia de este, cuenta con un modelo de procesamiento en memoria de baja latencia que permite realizar consultas interactivas orientadas a entornos *Business Intelligence*.

Se suele utilizar para procesamiento de ficheros almacenados HDFS con menor latencia que Hive y por tanto orientada a aplicaciones finales.

- Tipo de sistema: Procesamiento.
- Tipo de procesamiento: Interactivo.
- Tipo de despliegue/distribución: Nube pública y privada (on-premises) con multitud de distribuciones.
- Interfaces de programación/consulta: HiveQL compatible con SQL
- Autenticación: Usuario contraseña, Kerberos, otros
- Tipo de licencia/propietario: Apache 2.0
- Versión actual: 2.10.0
- Driver ODBC: sí

2.2.3 Amazon Redshift

Amazon Redshift es un almacén de datos rápido y completamente administrado que permite analizar todos los datos empleando de forma sencilla y rentable SQL estándar y las herramientas de Business Intelligence existentes.

Forma parte de la familia de servicios web de Amazon (AWS), por tanto se integra con gran parte de sus servicios, como por ejemplo Amazon S3.

Se suele utilizar para almacenar y analizar datos en entornos donde es necesaria una alta integración con otros servicios de AWS.

- Tipo de sistema: Almacenamiento y procesamiento.
- Tipo de procesamiento: Interactivo.
- Tipo de despliegue/distribución: Nube pública (Amazon Web Services)
- Interfaces de programación/consulta: SQL
- Autenticación: Usuario y contraseña.
- Tipo de licencia/propietario: Propietario.
- Versión actual: Al ser un servicio auto-administrado por Amazon no se ofrece información de versiones
- Driver ODBC: Sí

2.2.4 MongoDB

MongoDB es una base de datos orientada a objetos que pertenece a la familia de bases de datos *NoSQL*. Está diseñada para soportar escalabilidad, particionamiento, replicación, alta disponibilidad siendo de las primeras bases de datos NoSQL en ofrecer estas características y una de las más populares en la actualidad.

Se suele utilizar como base de datos operacional y es muy popular en arquitecturas *MEAN*, en las que tanto el front como el backend están desarrollados sobre Javascript.

- Tipo de sistema: Almacenamiento y procesamiento.
- Tipo de procesamiento: Interactivo.
- Tipo de despliegue/distribución: on-premises
- Interfaces de programación/consulta: Javascript (nativo) y otros SDK con lenguajes varios.
- Autenticación: Usuario y contraseña, Kerberos/LDAP
- Tipo de licencia/propietario: AGPL v3.0
- Versión actual: 3.4
- Driver ODBC: Sí

2.2.5 Google BigQuery

Google BigQuery es el almacén de datos en la nube de Google, totalmente administrado y apto para analizar petabytes de datos.

Google BigQuery es un sistema de almacenamiento con una arquitectura *serverless* y ofrecido a modo de SaaS. Entre sus características principales destaca la integración con otros servicios de Google como Google Cloud Storage[29], el soporte de OAuth[30] y acceso a través de API REST o SDKs en diferentes lenguajes.

Se suele utilizar en entornos donde se requiere integración con otros servicios de Google y en los que se pretende evitar el coste de mantenimiento de infraestructura.

- Tipo de sistema: Almacenamiento y procesamiento.
- Tipo de procesamiento: Interactivo.
- Tipo de despliegue/distribución: SaaS
- Interfaces de programación/consulta: API REST, SDKs
- Autenticación: OAuth
- Versión actual: Al ser un servicio auto-administrado por Google no se ofrece información de versiones
- Tipo de licencia/propietario: Propietario (Google)
- Driver ODBC: Sí

[29] https://cloud.google.com/storage/ - octubre 2017
[30] https://oauth.net/ - octubre 2017

2.3 Tabla resumen

Característica	Apache Hive	Apache Impala	Amazon Redshift	MongoDB	Google BigQuery
Tipo de sistema	Procesamiento	Procesamiento	Almacenamiento Procesamiento	Almacenamiento Procesamiento	Almacenamiento Procesmiento
Tipo de procesamiento	Batch	Interactivo	Interactivo	Interactivo	Interactivo
Tipo de despliegue	Nube on-premises	Nube on-premises	SaaS	Nube on-premises	SaaS
Interfaces	SQL	SQL	SQL	SDKs, Javascript	API REST, SDKs
Autenticación	Usuario	Usuario	Usuario	Usuario	OAuth 2.0
Versión actual	2.3.0	2.10.0	▪	3.4	▪
Licencia	Libre	Libre	Propietario	Libre	Propietario
Driver ODBC	Sí	Sí	Sí	Sí	Sí

CAPÍTULO 3

Metodología y plan de trabajo

3.1 Metodología

Como hemos visto en el estudio del estado del arte de los principales sistemas de almacenamiento Big Data, nos encontramos ante un ecosistema heterogéneo en cuanto a tipos de almacenamiento, procesamiento, despliegue, etc.

Aún siendo un ecosistema tan heterógeneo, es importante definir una metodología clara y sistemática en cuanto al desarrollo de conectores Big Data para CARTO. Esto es así, porque es de esperar que este campo siga evolucionando, surgiendo nuevas tecnologías y paradigmas a los que se deba dar soporte.

Así pues, en la definición de esta metodología sistemática, debemos encontrar un nexo de unión entre todos estos sistemas y CARTO.

3.2 ¿Cómo conectar con CARTO?

De acuerdo a la arquitectura de CARTO, la integración con sistemas de terceros se puede realizar a dos niveles:

- Utilizando sus APIs, algunas de las cuales exponen interfaces para acceder directamente a todas las capacidades de PostgreSQL a través de SQL estándar.
- Utilizando las capacidades de conectividad de PostgreSQL, tales como Foreign Data Wrappers.

Analizando las virtudes y defectos de ambas aproximaciones nos encontramos con lo siguiente:

A favor de la utilización de APIs como mecanismo de integración entre CARTO y otros sistemas está la flexibilidad. Estas APIs REST, se pueden utilizar en cualquier flujo de integración. Por otra parte, como inconveniente, nos encontramos con que se requieren desarrollos concretos para cada tipo de integración.

La utilización de las capacidades nativas de PostgreSQL para conectarse con sistemas de terceros presenta a su vez ventajas e inconvenientes. Entre las ventajas, cabe destacar, que el framework de Foreign Data Wrappers, consiste en un marco bien definido y ampliamente utilizado en la industria, que además, en gran parte de sus implementaciones se basa en la utilización de drivers ODBC, un estándar conocido y muy extendido en los sistemas de bases de datos relacionales.

El principal defecto de esta aproximación, consiste en la necesidad de realizar una conexión directa entre sistemas de bases de datos, en este caso, desde PostgreSQL a otros (tales como Hive, Impala, MongoDB, etc.). En algún caso, esto puede comprometer la seguridad de los sistemas de bases de datos.

Aún así, CARTO puede ser instalado on-premises, con lo que las organizaciones celosas de abrir una conexión fuera de su infraestrucura, podrían aprovecharse de este modo de integración.

Por último, y de nuevo haciendo referencia al estudio del estado del arte, hemos encontrado en todos los sistemas de almacenamiento Big Data dos puntos a favor de esta segunda aproximación:

- Todos los sistemas cuentan con driver ODBC
- Todos los sistemas cuentan con interfaz SQL o implementación de Foreign Data Wrapper específica para PostgreSQL

Con esto, podemos concluir que la utilización de Foreign Data Wrappers para conectar con sistemas de terceros, y en concreto, sistemas de almacenamiento Big Data, desde PostgreSQL es una solución factible y que además es susceptible de sistematizar.

Con esta premisa, vamos a definir, una metodología, que se pueda probar y repetir, para conectar CARTO con Hive, Impala, Redshift, BigQuery, MongoDB y en definitiva, cualquier sistema de almacenamiento.

Esta metodología consta de 5 fases, que se desarrollarán para cada sistema en la siguiente sección *Desarrollo del trabajo y resultados obtenidos* y que se enumeran a continuación:

1. Despliegue de un entorno de prueba del sistema de almacenamiento Big Data
2. Búsqueda, instalación y prueba de un driver ODBC compatible
3. Búsqueda, instalación y prueba de un Foreign Data Wrapper (opcionalmente se puede utilizar la implementación base de PostgreSQL o implementar una propia)
4. Desarrollo de un conector para CARTO
5. Ingestión de datos hacia CARTO

3.3 Plan de trabajo

El trabajo final de máster consta de 3 grandes bloques en su desarrollo.

1. Despliegue de distintos sistemas de almacenamiento Big Data en la nube de Amazon.
2. Desarrollo de conectores para CARTO.
3. Ingestión de datos de prueba y creación de un dashboard de visualización de datos geoespaciales con CARTO provenientes de uno o más de los sistemas implementados.

El plan de trabajo detallado consiste en las siguientes tareas:

3.3.1 Despliegue de sistemas de almacenamiento Big Data en la nube de Amazon

Esta tarea consiste en explorar las diferentes alternativas para desplegar sistemas de almacenamiento Big Data en la nube.

El objetivo no es contar con despliegues robustos, resistentes a fallos o configurados en cluster, sino contar con diferentes entornos para realizar la ingestión de datos de prueba y conexión necesaria durante el desarrollo y demostración de los conectores para CARTO.

Para el desarrollo de este bloque se realizan las siguientes tareas:

- Aprovisionamiento de máquinas virtuales utilizando el servicio EC2 de Amazon, sobre una AMI de Ubuntu 14.04 y utilizando los servicios adicionales para configuración de *firewall*, control de acceso, disco, etc.
- Despliegue con Docker de Cloudera Quickstart para contar con instancias de Hive e Impala.
- Despliegue de una instancia de Amazon Redshift.
- Despliegue con Docker de una instancia de MongoDB.
- Despliegue con Docker de una instancia de Cassandra.
- Despliegue con Docker de una instancia de Oracle.
- Configuración de una cuenta y credenciales para acceso a Google BigQuery.

3.3.2 Desarrollo de conectores para CARTO

Este bloque consiste en la implementación y prueba de los diferentes módulos que permitan conectar CARTO con los sistemas de almacenamiento mencionados previamente a través de su API de importación.

En este caso, el objetivo consiste en contar con conectores integrados en el código base de CARTO, de manera que sean incluidos en su versión *on premise*.

Para el desarrollo de deste bloque se realizan las siguientes tareas:

- Configuración de un entorno de desarrollo de CARTO utilizando Github, BASH y Vagrant.
- Desarrollo en Ruby del código necesario para los conectores.
- Configuración de las llamadas necesarias a la API REST de importación de CARTO.
- Documentación y scripts de configuración de los *drivers* necesarios para conectar con cada sistema de almacenamiento.
- Despliegue de CARTO en un servidor de *staging* en Amazon.

3.3.3 Ingestión de datos de prueba y creación de dashboard con CARTO

Una vez desplegados diferentes sistemas de almacenamiento Big Data en la nube, desarrollados los conectores y desplegada una instancia de CARTO, el último bloque consiste en realizar una pequeña demostración sobre un *dashboard* que consuma datos obtenidos de uno o más de estos sistemas desplegados.

Para el desarrollo de deste bloque se realizan las siguientes tareas:

- Ingestión de datos en los distintos sistemas de almacenamiento provistos.
- Ejecución de las llamadas necesarias mediante la API de importación de CARTO para conectar con uno o más sistemas de almacenamiento.
- Creación de un dashboard de análisis y visualización de datos geoespaciales con CARTO provenientes de uno o más de los sistemas implementados.

TODO: especificar un poco más qué datasets y qué sistemas de almacenamiento concreto se van a usar.

3.4 Metodología de trabajo

Para llevar a cabo el plan de trabajo se va a seguir una metodología de desarrollo iterativo incremental. Se trata de una metodología de desarrollo de software ágil que consiste en la ejecución de las distintas fases del proyecto en ciclos cortos de pocos días que se repiten en el tiempo, de manera que se va incrementando el valor de la solución final.

Esta metodología nos va a permitir validar en una frase temprana la solución propuesta, realizando una iteración que permita validar la integración de Hive e Impala con CARTO.

Una vez validado uno de estos sistemas de almacenamiento, se continúan realizando iteraciones cortas en las que se va dando soporte al resto de sistemas de almacenamiento propuestos, hasta contar con la solución completa.

En última instancia, se trabaja en la ingestión de datos y creación del dashboard a modo de demostración.

Por otra parte, como segundo objetivo metodológico, se pretende que todo el entorno sea fácilmente reproducible, así pues, se utilizan herramientas que facilitan la automatización y colaboración: Github, BASH, Vagrant, Docker, etc. Con lo que es posible reproducir todo el desarrollo realizado durante el trabajo final de máster.

CAPÍTULO 4

Desarrollo del trabajo y resultados obtenidos

6. Desarrollo del trabajo y resultados obtenidos.

- **CARTO: Arquitectura**
    - PostGIS - FDW: pg_fdw or custom - ODBC drivers

De acuerdo a la metodología definida en el apartado a interior, en este apartado se incluye el desarrollo de la misma para cada uno de los sistemas de almacenamiento Big Data objetivo de ser integrados con CARTO.

El objetivo es contar con un procedimiento sistemático que incluya al menos las siguientes fases, para cada sistema de almacenamiento:

1. Despliegue de un entorno de prueba del sistema de almacenamiento Big Data
2. Búsqueda, instalación y prueba de un driver ODBC compatible
3. Búsqueda, instalación y prueba de un Foreign Data Wrapper (opcionalmente se puede utilizar la implementación base de PostgreSQL o implementar una propia)
4. Desarrollo de un conector para CARTO
5. Ingestión de datos hacia CARTO

4.1 Conceptos previos

- Cómo probar un FDW / compatibilidad con postgres_fdw
- Cómo funciona unixODBC

4.2 Integración de CARTO con Apache Hive

Apache Hive es una infraestructura de almacenamiento y procesamiento de datos almacenados sobre HDFS de Hadoop y otros sistemas compatibles como Amazon S3, originalmente desarrollado por Facebook.

Hive es fundamentalmente una capa de abstracción que convierte consultas SQL (escritas en un lenguaje compatible con SQL llamado HiveQL) en trabajos MapReduce, Tez o Spark.

La integración de CARTO con Apache Hive se va a realizar de acuerdo a los siguientes parámetros:

- Soporta SQL: Sí
- Driver ODBC: Sí
- Compatible con *postgres_fdw*: Sí
- Versión probada: 2.3.0
- Autenticación: Usuario y contraseña
- Distribución: Cloudera Quickstart
- Despliegue: Docker sobre AWS

4.2.1 Despliegue de un entorno de prueba de Apache Hive

Para desplegar una instancia de Apache Hive, utilizamos la imagen de Cloudera Quickstart disponible en Docker Hub, ejecutando el siguiente comando:

```
docker pull cloudera/quickstart:latest
docker run --name=cloudera -p 8888:8888 -p 10000:10000 -p 21050:21050 -v /tmp:/tmp --
↪hostname=quickstart.cloudera --privileged=true -t -i -d cloudera/quickstart /usr/
↪bin/docker-quickstart
docker exec -it cloudera /bin/bash
```

Para este caso, hay que tener en cuenta que la imagen de Cloudera Quickstart cuenta con una distribución completa de Hadoop, por tanto con la imagen se arrancan multitud de servicios y es necesaria una cantidad considerable de memoria RAM.

Para el caso en el que sea desea correr esta imagen en una máquina local o con recursos limitados, es posible que algunos de los procesos no arranquen. En estos casos, es recomendable parar el resto de procesos que no son imprescindibles para contar con una instancia de Apache Hive.

Deteniendo los siguientes servicios es posible arrancar una imagen de Cloudera Quickstart con Docker, únicamente con 2GB de memoria:

```
/etc/init.d/flume-ng-agent stop
/etc/init.d/oozie stop
/etc/init.d/spark-history-server stop
/etc/init.d/solr-server stop
/etc/init.d/flume-ng-agent stop
/etc/init.d/hive-metastore restart
/etc/init.d/hive-server2 restart
/etc/init.d/flume-ng-agent stop
/etc/init.d/hbase-master stop
/etc/init.d/hbase-regionserver stop
/etc/init.d/hbase-rest stop
/etc/init.d/hbase-solr-indexer stop
/etc/init.d/hbase-thrift stop
/etc/init.d/oozie stop
/etc/init.d/sentry-store stop
```

Después de detener los servicios es importante reiniciar la interfaz de HUE que nos permitirá realizar consultas interactivas sobre Hive con el siguiente comando: */etc/init.d/hue restart*

4.2.2 Ingestión de datos en Apache Hive

Una vez hemos desplegado Apache Hive utilizando la imagen de Cloudera Quickstart con Docker, podemos hacer una ingestión inicial de datos para posteriormente realizar las pruebas necesarias de integración con CARTO.

Accediendo al servidor EC2 de Amazon:

```
ssh {path_to_pem_file} {user}@{server}
```

Y a continuación abriendo una sesión de bash en el contenedor de Cloudera Quickstart:

```
docker exec -it cloudera /bin/bash
```

Podemos utilizar *sqoop* para hacer una ingestión inicial de datos en Hive desde una base de datos MySQL incluida en la imagen de Cloudera con el siguiente comando:

```
sqoop import-all-tables \
    --connect jdbc:mysql://localhost:3306/retail_db \
    --username=retail_dba \
    --password=cloudera \
    --compression-codec=snappy \
    --as-parquetfile \
    --warehouse-dir=/user/hive/warehouse \
    --hive-import
```

Por último, podemos acceder a la interfaz de HUE y comprobar que efectivamente las tablas se han cargado correctamente en Hive

TODO: añadir capturas de pantalla

```
http://localhost:8888/
usr/pwd: cloudera/cloudera
```

4.2.3 Instalación y prueba de un driver ODBC para Hive

En este caso, Cloudera proporciona un driver ODBC para Hive con licencia libre que podemos instalar en distribuciones Redhat/CentOS con los siguientes comandos:

```
wget "https://downloads.cloudera.com/connectors/hive_odbc_2.5.22.1014/Linux/EL6/
↪ClouderaHiveODBC-2.5.22.1014-1.el6.x86_64.rpm"
yum install cyrus-sasl-gssapi.x86_64 cyrus-sasl-plain.x86_64
yum --nogpgcheck localinstall ClouderaHiveODBC-2.5.22.1014-1.el6.x86_64.rpm
```

4.2.4 Configuración del driver ODBC para Hive

Una vez descargado el driver ODBC para Hive es necesario editar los archivos que PostgreSQL utiliza para conocer los drivers disponibles en el sistema.

La ubicación de los archivos de configuración se puede obtener ejecutando la siguiente instrucción:

```
[root@localhost vagrant]# odbcinst -j
unixODBC 2.3.4
DRIVERS............: /opt/carto/postgresql/embedded/etc/odbcinst.ini
SYSTEM DATA SOURCES: /opt/carto/postgresql/embedded/etc/odbc.ini
FILE DATA SOURCES..: /opt/carto/postgresql/embedded/etc/ODBCDataSources
```

(continues on next page)

(proviene de la página anterior)

```
USER DATA SOURCES..: /root/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

El comando *odbcinst* lo provee el paquete *unixODBC* que viene instalado por defecto en la distribución on-premise de CARTO.

Una vez conocemos la ubicación de los archivos de configuración, añadimos el driver de Hive a la lista de drivers disponibles:

```
printf "\n[Hive]
Description=Cloudera ODBC Driver for Apache Hive (64-bit)
Driver=/opt/cloudera/hiveodbc/lib/64/libclouderahiveodbc64.so" >> /data/production/
↪config/postgresql/odbcinst.ini
```

4.2.5 Instalación y prueba de un Foreign Data Wrapper para Hive

Una primera aproximación a la hora de probar un Foreign Data Wrapper para Hive, consiste en probar la implementación base disponible en PostgreSQL *postgres_fdw*.

En este caso, el driver ODBC de Cloudera para Apache Hive es compatible con *postgres_fdw* del que CARTO cuenta con una implementación base.

4.2.6 Desarrollo de un conector de Hive para CARTO

Puesto que el driver ODBC para Hive es compatible con *postgres_fdw* la implementación de un conector de Hive para CARTO se reduce a añadir una nueva clase al *backend* indicando cuáles son los parámetros necesarios para realizar una consulta SQL sobre Hive y configurar este conector para que sea accesible desde la API de importación de CARTO.

El código del conector *hive.rb* se adjunta en el anexo xxx -> TODO incluir enlace

4.2.7 Ingestion de datos desde Hive a CARTO

Una vez disponemos de una instalación on-premise de CARTO, con el driver ODBC de Hive correctamente instalado y configurado tanto en el sistema como en CARTO y un conector correctamente implementado, podemos realizar una ingestión de datos desde Hive a CARTO utilizando la API de importación de la siguiente manera:

```
curl -v -k -H "Content-Type: application/json"   -d '{
  "connector": {
    "provider": "hive",
    "connection": {
      "server":"{hive_server_ip}",
      "database":"default",
      "port":10000,
      "username":"{hive_user}",
      "password":"{hive_password}"
    },
    "schema": "default",
    "table": "top_order_items",
    "sql_query": "select * from order_items where price > 1000"
  }
}'   "https://carto.com/user/carto/api/v1/imports/?api_key={YOUR_API_KEY}"
```

La anterior llamada a la API de importación, crea una conexión mediante Foreign Data Wrapper desde el servidor de CARTO (en concreto desde el servidor de PostgreSQL) hacia el servidor de Hive a través del puerto 10000 (el puerto por defecto de Hive).

Una vez realizada la conexión, se crea una tabla en PostgreSQL de nombre *top_order_items* y se ejecuta la siguiente consulta en Hive para obtener los pedidos con un precio superior a mil dólares:

```
select * from order_items where price > 1000
```

Hive transformará está consulta SQL en un trabajo MapReduce y devolverá el resultado al Foreign Data Wrapper, convirtiéndose en filas de la tabla en PostgreSQL.

Esta tabla de PostgreSQL está asociada a un dataset del usuario de CARTO que lanzó la petición y por tanto puede trabajar con él, de la misma manera que con cualquier otro dataset.

4.3 Integración de CARTO con Apache Impala

Apache Impala es una infraestructura de almacenamiento y procesamiento de datos almacenados sobre HDFS de Hadoop, originalmente desarrollado por Cloudera.

Apache Impala es compatible con HiveQL y utiliza la misma base de datos de metadatos para acceder a HDFS que Hive, pero a diferencia de este, cuenta con un modelo de procesamiento en memoria de baja latencia que permite realizar consultas interactivas orientadas a entornos *Business Intelligence*.

La integración de CARTO con Apache Impala se va a realizar de acuerdo a los siguientes parámetros:

- Soporta SQL: Sí
- Driver ODBC: Sí
- Compatible con *postgres_fdw*: Sí
- Versión probada: 2.10.0
- Autenticación: Usuario y contraseña
- Distribución: Cloudera Quickstart
- Despliegue: Docker sobre AWS

4.3.1 Despliegue de un entorno de prueba de Apache Impala

Para desplegar una instancia de Apache Impala, utilizamos la imagen de Cloudera Quickstart disponible en Docker Hub, tal y como hicimos al desplegar Apache Hive.

TODO añadir link a la sección anterior

4.3.2 Ingestión de datos en Apache Impala

Apache Impala es compatible con el modelo de metadatos de Apache Hive, por tanto, se pueden ingestar datos en Apache Impala tal y como se hizo para Apache Hive. [TODO] -> Añadir link a la sección correspondiente.

Una vez presentes los datos en el *metastore* de Hive, es necesario ejecutar la siguiente instrucción para actualizar la base de datos de metadatos de Impala:

```
invalidate metadata;
```

Dicha instrucción se puede ejecutar directamente desde la consola de Impala disponible en HUE y accesible con las siguientes credenciales:

```
http://localhost:8888/
usr/pwd: cloudera/cloudera
```

4.3.3 Instalación y prueba de un driver ODBC para Impala

El procedimiento para instalar el driver ODBC para Impala es similar al de Hive [TODO] -> link a la sección correspondiente.

```
yum install -y cyrus-sasl.x86_64 cyrus-sasl-gssapi.x86_64 cyrus-sasl-plain.x86_64
wget "https://downloads.cloudera.com/connectors/impala_odbc_2.5.37.1014/Linux/EL6/
↪ClouderaImpalaODBC-2.5.37.1014-1.el6.x86_64.rpm"
yum --nogpgcheck -y localinstall ClouderaImpalaODBC-2.5.37.1014-1.el6.x86_64.rpm
```

4.3.4 Configuración del driver ODBC para Impala

Una vez descargado el driver ODBC para Impala es necesario editar los archivos que PostgreSQL utiliza para conocer los drivers disponibles en el sistema.

La ubicación de los archivos de configuración se puede obtener ejecutando la siguiente instrucción:

```
[root@localhost vagrant]# odbcinst -j
    unixODBC 2.3.4
    DRIVERS............: /opt/carto/postgresql/embedded/etc/odbcinst.ini
    SYSTEM DATA SOURCES: /opt/carto/postgresql/embedded/etc/odbc.ini
    FILE DATA SOURCES..: /opt/carto/postgresql/embedded/etc/ODBCDataSources
    USER DATA SOURCES..: /root/.odbc.ini
    SQLULEN Size.......: 8
    SQLLEN Size........: 8
    SQLSETPOSIROW Size.: 8
```

El comando *odbcinst* lo provee el paquete *unixODBC* que viene instalado por defecto en la distribución on-premise de CARTO.

Una vez conocemos la ubicación de los archivos de configuración, añadimos el driver de Impala a la lista de drivers disponibles:

```
printf "\n[Impala]
Description=Cloudera ODBC Driver for Impala (64-bit)
Driver=/opt/cloudera/impalaodbc/lib/64/libclouderaimpalaodbc64.so" >> /data/
↪production/config/postgresql/odbcinst.ini
```

4.3.5 Instalación y prueba de un Foreign Data Wrapper para Impala

Análogamente a lo que ocurría con Hive, el driver ODBC de Cloudera para Apache Impala también es compatible con *postgres_fdw* del que CARTO cuenta con una implementación base. Por tanto, no es necesaria una implementación personalizada.

4.3.6 Desarrollo de un conector de Impala para CARTO

Puesto que el driver ODBC para Impala es compatible con *postgres_fdw* la implementación de un conector de Impala para CARTO se reduce a añadir una nueva clase al *backend* indicando cuáles son los parámetros necesarios para realizar una consulta SQL sobre Impala y configurar este conector para que sea accesible desde la API de importación de CARTO.

El código del conector *impala.rb* se adjunta en el anexo xxx -> TODO incluir enlace

4.3.7 Ingestion de datos desde Impala a CARTO

Una vez más, la petición a la API de importación de CARTO es análoga a la del caso de Hive.

```
curl -v -k -H "Content-Type: application/json"   -d '{
  "connector": {
    "provider": "impala",
    "connection": {
      "server":"{impala_server_ip}",
      "database":"default",
      "port":21050,
      "username":"{impala_user}",
      "password":"{impala_password}"
    },
    "schema": "default",
    "table": "top_order_items",
    "sql_query": "select * from order_items where price > 1000"
  }
}'   "https://carto.com/user/carto/api/v1/imports/?api_key={YOUR_API_KEY}"
```

La anterior llamada a la API de importación, crea una conexión mediante Foreign Data Wrapper desde el servidor de CARTO (en concreto desde el servidor de PostgreSQL) hacia el servidor de Impala a través del puerto 21050 (el puerto por defecto de Impala).

Una vez realizada la conexión, se crea una tabla en PostgreSQL de nombre *top_order_items* y se ejecuta la siguiente consulta en Impala para obtener los pedidos con un precio superior a mil dólares:

```
select * from order_items where price > 1000
```

En este caso, Impala no implementa el paradigma MapReduce sino que utiliza un mecanismo de procesamiento en memoria que permite la realización de consultas interactivas, por lo que la respuesta tiene una latencia menor al caso de Hive.

La tabla generada en PostgreSQL está asociada a un dataset del usuario de CARTO que lanzó la petición y por tanto puede trabajar con él, de la misma manera que con cualquier otro dataset.

4.4 Integración de CARTO con Amazon Redshift

Amazon Redshift es un almacén de datos de la familia de servicios web de Amazon, completamente administrado que permite analizar datos con SQL estándar.

La integración de CARTO con Apache Redshift se va a realizar de acuerdo a los siguientes parámetros:

- Soporta SQL: Sí
- Driver ODBC: Sí
- Compatible con *postgres_fdw*: Sí

- Versión probada: Amazon no proporciona información acerca del versionado de Redshift, por tanto, las pruebas realizadas son con la versión actual a fecha Septiembre 2017
- Autenticación: Usuario y contraseña
- Distribución: AWS
- Despliegue: Auto-gestionado a través de la consola de administración de AWS

4.4.1 Despliegue de un entorno de prueba de Amazon Redshift

TODO -> incluir capturas de pantalla

4.4.2 Ingestión de datos en Amazon Redshift

TODO

4.4.3 Instalación y prueba de un driver ODBC para Amazon Redshift

El procedimiento para instalar el driver ODBC para Impala es similar a los de Hive e Impala [TODO] -> link a la sección correspondiente.

```
wget "https://s3.amazonaws.com/redshift-downloads/drivers/AmazonRedshiftODBC-64bit-1.
↪3.1.1000-1.x86_64.rpm"
yum --nogpgcheck localinstall AmazonRedshiftODBC-64bit-1.3.1.1000-1.x86_64.rpm
```

4.4.4 Configuración del driver ODBC para Redshift

Una vez descargado el driver ODBC para Amazon Redshift es necesario editar los archivos que PostgreSQL utiliza para conocer los drivers disponibles en el sistema.

El procedimiento es análogo a los casos de Hive e Impala:

```
printf "\n[Redshift]
Description=Amazon Redshift ODBC Driver(64-bit)
Driver=/opt/amazon/redshiftodbc/lib/64/libamazonredshiftodbc64.so" >> /data/
↪production/config/postgresql/odbcinst.ini
```

4.4.5 Instalación y prueba de un Foreign Data Wrapper para Redshift

Análogamente a lo que ocurría con Hive e Impala, el driver ODBC de Cloudera para Amazon Redshift también es compatible con *postgres_fdw* del que CARTO cuenta con una implementación base. Por tanto, no es necesaria una implementación personalizada.

4.4.6 Desarrollo de un conector de Redshift para CARTO

El código del conector *redshift.rb* se adjunta en el anexo xxx -> TODO incluir enlace

4.4.7 Ingestion de datos desde Redshift a CARTO

Una vez más, la petición a la API de importación de CARTO es análoga a la del caso de Hive e Impala.

```
curl -v -k -H "Content-Type: application/json"   -d '{
  "connector": {
    "provider": "redshift",
    "connection": {
      "server":"{redshift_server_ip}",
      "database":"default",
      "port":5439,
      "username":"{redshift_user}",
      "password":"{redshift_password}"
    },
    "schema": "default",
    "table": "top_order_items",
    "sql_query": "select * from order_items where price > 1000"
  }
}'   "https://carto.com/user/carto/api/v1/imports/?api_key={YOUR_API_KEY}"
```

4.5 Antes de continuar

Antes de continuar con el desarrollo de los siguientes conectores Big Data para CARTO, cabe destacar que hemos encontrado un procedimiento sistemático para desarrollar conectores desde sistemas de almacenamiento que cumplen las siguientes características:

- Tienen un Driver ODBC
- Soportan SQL como lenguaje de procesamiento
- Son compatibles con *postgres_fdw*

Tal y como hemos visto en las secciones anteriores, el desarrollo de conectores para Hive, Impala y Redshift es completamente análogo, por tanto, el mismo procedimiento sería válido para sistemas de almacenamiento que cumplan las 3 características mencionadas en esta sección.

4.6 Integración de CARTO con MongoDB

MongoDB es una base de datos orientada a objetos que pertenece a la familia de bases de datos NoSQL. Se suele utilizar como base de datos operacional y es muy popular en entornos JavaScript.

- Tipo de sistema: Almacenamiento y procesamiento.
- Tipo de procesamiento: Interactivo.
- Tipo de despliegue/distribución: on-premises
- Interfaces de programación/consulta: Javascript (nativo) y otros SDK con lenguajes varios.
- Autenticación: Usuario y contraseña, Kerberos/LDAP
- Tipo de licencia/propietario: AGPL v3.0
- Versión actual: 3.4
- Driver ODBC: Sí

La integración de CARTO con Apache Redshift se va a realizar de acuerdo a los siguientes parámetros:

- Soporta SQL: No
- Driver ODBC: Sí
- Compatible con *postgres_fdw*: No
- Versión probada: 3.4
- Autenticación: Usuario y contraseña
- Distribución: Imagen de Docker
- Despliegue: Docker sobre AWS

4.6.1 Despliegue de un entorno de prueba de MongoDB

Para el despliegue de una instancia de MongoDB vamos a utilizar la siguiente imagen -> TODO https://hub.docker.com/r/tutum/mongodb/

Ejecutamos el script de arranque del contenedor de MongoDB sobre una instancia de EC2:

```
docker run --name mongo --network=host -d -p 27017:27017 -p 28017:28017 tutum/mongodb
```

En este caso concreto, al arrancar la imagen de Docker utilizada, se crea un usuario y contraseña para acceder a la instancia de MongoDB. Para conocer el password del usuario administrador, debemos esperar a que termine de arrancar el contenedor e imprimir los logs de esta manera:

En primer lugar, obtener el ID del contenedor:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
↪STATUS              PORTS               NAMES
971d9c6bb9e3        tutum/mongodb       "/run.sh"           21 seconds ago      Up 18␣
↪seconds                        mongo
```

A continuación utilizar el comando *docker logs <CONTAINER ID>*, hasta obtener una salida similar a esta:

```
$ docker logs 971d9c6bb9e3
================================================================================
You can now connect to this MongoDB server using:

    mongo admin -u admin -p Ck15KQ2G4pdl --host <host> --port <port>

Please remember to change the above password as soon as possible!
================================================================================
```

4.6.2 Ingestión de datos en MongoDB

Una vez hemos obtenido las credenciales de usuario administrador en el anterior paso, podemos crear una base de datos de prueba que utilizaremos para desarrollar el conector para MongoDB sobre CARTO.

```
# open a bash session in the Docker container
docker exec -it mongo /bin/bash
# and then create a collection in the admin database
mongo -u admin -p Ck15KQ2G4pdl --authenticationDatabase 'admin'
use admin
db.createCollection("warehouse")
```

4.6.3 Instalación y prueba de un Foreign Data Wrapper para MongoDB

A diferencia de lo que ocurría en los casos de Hive, Impala o Redshift, el driver ODBC de MongoDB no es compatible con *postgres_fdw*, por tanto, nos encontramos con un caso en que debemos utilizar un Foreignd Data Wrapper específico.

Esto tiene sentido ya que MongoDB, es una base de datos NoSQL orientada a objetos sin interfaz SQL, por tanto la implementación de un foreign data wrapper debe ser diferente.

A la hora de elegir un FDW para MongoDB valoramos las opciones listadas en el wiki oficial de PostgreSQL [TODO] -> https://wiki.postgresql.org/wiki/Foreign_data_wrappers.

Entre la lista, nos encontramos con dos FDW desarrollados con Multicorn [TODO] -> link y uno desarrollado de manera nativa en C. Accediendo al código fuente de los repositorios, vemos que el más activo es el FDW nativo, por tanto, lo seleccionamos como candidato para conectar a MongoDB desde PostgreSQL.

TODO -> ADD LINK .. _this one: https://github.com/EnterpriseDB/mongo_fdw

Las instrucciones de instalación a fecha septiembre de 2017 de *mongo_fdw* no resultan al 100 % correctas, por tanto, adjuntamos a continuación los pasos necesarios para realizar la instalación, configuración y prueba del mismo sobre CentOS 6.9

Procedemos a ejecutar los siguientes comandos como root en la misma máquina donde tenemos PostgreSQL instalado

En primer lugar, hay que satisfacer algunas dependencias del sistema:

```
yum install -y openssl-devel patch
```

La instalación de *mongo_fdw* sólo funciona con una versión de *gcc* 4.8 o superior:

```
wget http://people.centos.org/tru/devtools-2/devtools-2.repo -O /etc/yum.repos.d/
↪devtools-2.repo
yum install devtoolset-2-gcc devtoolset-2-binutils devtoolset-2-gcc-c++ devtoolset-2-
↪gcc-gfortran -y
scl enable devtoolset-2 bash
```

Debemos asegurarnos que la versión de gcc instalada es la correcta (4.8 o superior):

```
$ gcc --version
gcc (GCC) 4.8.2 20140120 (Red Hat 4.8.2-15)
```

Descargar la última release de *mongo_fdw*, en nuestro caso la 5.0.0 compatible con CentOS:

```
wget https://github.com/EnterpriseDB/mongo_fdw/archive/REL-5_0_0.tar.gz
tar zxvf REL-5_0_0.tar.gz
cd mongo_fdw-REL-5_0_0
```

A fecha de septiembre de 2017, hay un bug en una de las dependencias de *mongo_fdw* [TODO] -> See this pull request.

Aplicamos el parche manualmente, sobre el archivo *autogen.sh*

A continuación compilamos e instalamos el driver nativo para MongoDB y todas las librerías necesarias:

```
export CFLAGS=-fPIC
export CXXFLAGS=-fPIC
./autogen.sh --with-master
wget https://github.com/mongodb/mongo-c-driver/releases/download/1.6.3/mongo-c-driver-
↪1.6.3.tar.gz
tar zxvf mongo-c-driver-1.6.3.tar.gz
cd mongo-c-driver-1.6.3
```

(continues on next page)

(proviene de la página anterior)

```
./configure --prefix=/usr --libdir=/usr/lib64
make && make install
cd ..
make && make install
```

Llegados a este punto, debemos ser capaces de probar el FDW *mongo_fdw* directamente desde la consola *psql* ejecutando las siguientes instrucciones:

```
psql -U postgres
CREATE EXTENSION mongo_fdw;
CREATE SERVER mongo_server
     FOREIGN DATA WRAPPER mongo_fdw
     OPTIONS (address '192.168.99.100', port '27017');
CREATE USER MAPPING FOR postgres
     SERVER mongo_server
     OPTIONS (username 'admin', password 'Ck15KQ2G4pdl');
CREATE FOREIGN TABLE warehouse(
    _id NAME,
    warehouse_id int,
    warehouse_name text,
    warehouse_created timestamptz)
    SERVER mongo_server
        OPTIONS (database 'admin', collection 'warehouse');
INSERT INTO warehouse values (0, 1, 'UPS', '2014-12-12T07:12:10Z');
SELECT * FROM warehouse WHERE warehouse_id = 1;
```

Reemplazar los attributos 'address', 'password', etc. de acuerdo a la instancia local de MongoDB

4.6.4 Desarrollo de un conector de MongoDB para CARTO

El código del conector *mongo.rb* se adjunta en el anexo xxx -> TODO incluir enlace

4.6.5 Ingestion de datos desde MongoDB a CARTO

Una vez más, la petición a la API de importación de CARTO es similar a la del caso de Hive e Impala.

```
curl -v -k -H "Content-Type: application/json"   -d '{
  "connector": {
    "provider": "mongo",
    "connection": {
      "username":"admin",
      "password":"Ck15KQ2G4pdl",
      "server": "192.168.99.100",
      "database": "admin",
      "port":"27017",
      "schema": "warehouse"
    },
    "table": "warehouse",
    "columns": "_id NAME,   warehouse_id int,   warehouse_name text,   warehouse_
↪created timestamptz"
  }
}'   "https://carto.com/user/carto/api/v1/imports/?api_key={YOUR_API_KEY}"
```

En este caso, debido a la implementación de *mongo_fdw* debemos incluir un atributo más en la petición para definir las columnas de la tabla que queremos importar desde MongoDB a PostgreSQL (y en definitiva a CARTO).

Nos encontramos en este caso, ante un conector para el que hemos tenido que instalar un Foreign Data Wrapper customizado, pero cuyo comportamiento en última instancia es similar a los anteriores, ya que podemos importar datos a CARTO con una simple petición a la API de importación.

4.7 Integración de CARTO con Google BigQuery

Google BigQuery es el almacén de datos en la nube de Google, totalmente administrado y apto para analizar petabytes de datos.

El conector de Google BigQuery para CARTO es un ejemplo de implementación que utiliza autenticación OAuth y para la que además se ha desarrollado una interfaz de usuario.

La integración de CARTO con Google BigQuery se va a realizar de acuerdo a los siguientes parámetros:

- Soporta SQL: Sí
- Driver ODBC: Sí
- Compatible con *postgres_fdw*: Sí
- Versión probada: 2.3.0
- Autenticación: Google no proporciona información acerca del versionado de BigQuery, por tanto, las pruebas realizadas son con la versión actual a fecha Septiembre 2017
- Distribución: SaaS
- Despliegue: SaaS

4.7.1 Despliegue de un entorno de prueba de Google BigQuery

En contraposición a otros sistemas de base de datos, Google BigQuery es una base de datos SaaS completamente auto-gestionada por Google siguiendo el paradigma *serverless*. Así que para obtener un entorno de pruebas de Google BigQuery, simplemente debemos habilitarlo con nuestra cuenta de Google.

Google ofrece una capa gratuita para BigQuery (a fecha septiembre de 2017), con unos límites más que suficientes para realizar pruebas y desarrollos: 1TB por mes en lecturas e importaciones/exportaciones ilimitadas.

No se especifican detalles de cómo habilitar una cuenta de Google BigQuery, ya que es un procedimiento totalmente auto-descriptivo, desde la consola de administración de Google Cloud.

4.7.2 Ingestión de datos en Google BigQuery

[TODO] -> Pantallazos de las interfaces e importación de CSV

4.7.3 Instalación y prueba de un driver ODBC Google BigQuery

Google proporciona un driver ODBC oficial para Google BigQuery, desarrollado por un proveedor externo, Simba. El procedimiento de instalación es similar al de otros drivers ODBC que hemos visto en esta sección (Hive, Impala, Redshift, etc.)

```
wget https://storage.googleapis.com/simba-bq-release/odbc/
↪SimbaODBCDriverforGoogleBigQuery64-2.0.6.1011.tar.gz
tar zxvf SimbaODBCDriverforGoogleBigQuery64-2.0.6.1011.tar.gz -C /opt
chown postgres:postgres /opt/simba
```

4.7.4 Configuración del driver ODBC para Hive

Una vez descargado el driver ODBC para GoogleBigQuery es necesario editar los archivos que PostgreSQL utiliza para conocer los drivers disponibles en el sistema.

La ubicación de los archivos de configuración se puede obtener ejecutando la siguiente instrucción:

```
[root@localhost vagrant]# odbcinst -j
    unixODBC 2.3.4
    DRIVERS............: /opt/carto/postgresql/embedded/etc/odbcinst.ini
    SYSTEM DATA SOURCES: /opt/carto/postgresql/embedded/etc/odbc.ini
    FILE DATA SOURCES..: /opt/carto/postgresql/embedded/etc/ODBCDataSources
    USER DATA SOURCES..: /root/.odbc.ini
    SQLULEN Size.......: 8
    SQLLEN Size........: 8
    SQLSETPOSIROW Size.: 8
```

El comando *odbcinst* lo provee el paquete *unixODBC* que viene instalado por defecto en la distribución on-premise de CARTO.

Una vez conocemos la ubicación de los archivos de configuración, añadimos el driver de Hive a la lista de drivers disponibles:

```
printf "\n[BigQuery]
Description = Simba ODBC Driver for Google BigQuery (64-bit)
Driver = /opt/simba/googlebigqueryodbc/lib/64/libgooglebigqueryodbc_sb64.so" >> /data/
↪production/config/postgresql/odbcinst.ini
```

Para el caso de BigQuery es necesario habilitar OAuth a nivel de driver. Hay dos modos de funcionamiento: TODO add reference

- Autenticación de usuario: Autentica contra una cuenta de usuario de Google obteniendo un *refresh token* que es temporal
- Autenticación de servicio: Autentica una aplicación a través de una *clave privada de servicio*, clave que debe ser descargada desde la consola de autenticación de Google Cloud.

Para este caso concreto, vamos a utilizar una clave privada .p12, dejándola en */opt* en el servidor donde hemos descargado el driver ODBC y tenemos instalado PostgreSQL:

TODO -> poner esto bien Google Cloud authentication console

```
[root@localhost vagrant]# ls -lh /opt
total 20K
drwxrwxr-x  12 root     root     4.0K Aug 24 11:07 carto
drwxr-xr-x   4 root     root     4.0K Sep  1 13:38 cloudera
drwxr-xr-x.  2 root     root     4.0K Mar 26  2015 rh
drwxr-xr-x   3 postgres postgres 4.0K Dec 12  2016 simba
-rw-r--r--   1 root     root     2.5K Sep  4 09:03 test-d58ed25bb6f7.p12
```

4.7.5 Instalación y prueba de un Foreign Data Wrapper para Google BigQuery

Una primera aproximación a la hora de probar un Foreign Data Wrapper para Hive, consiste en probar la implementación base disponible en PostgreSQL *postgres_fdw*.

En este caso, el driver ODBC de Google BigQuery es compatible con *postgres_fdw* del que CARTO cuenta con una implementación base.

4.7.6 Desarrollo de un conector de Google BigQuery para CARTO

El desarrollo de un conector para Google BigQuery es un caso especial de conector ya que debemos tratar con el flujo de autenticación de OAuth.

Por ese motivo se van a relizar 3 aproximaciones:

1. Utilización de la actual implementación de *odbc_fdw* para autenticación de usuario.
2. Utilización de la actual implementación de *odbc_fdw* para autenticación de servicio.
3. Desarrollo de un conector personalizado para gestionar el flujo de OAuth.

Las dos primeras aproximaciones no necesitan código fuente, ya que se basan en la utilización de *odbc_fdw* ya disponible en CARTO. Como desventaja, se deja del lado del usuario la gestión del flujo OAuth para obtener un token de usuario.

Aún así, en la siguiente sección veremos ejemplos de uso.

Por otra parte, se desarrolla un conector personalizado para gestionar el flujo de OAuth y la conexión a Google BigQuery, todo integrado desde la interfaz de usuario.

El código del conector se puede encontrar en el archivo *biquery.rb* disponible en el anexo xxx [TODO]

Specially you may want to take a look at the backend implementation

TODO -> completar bien esto

Usage of the *carto-builder-bigquery.sh* script:

```
Usage: carto-builder-bigquery.sh [-h] [-o|--organization organization] [-c|--client-
↪id clientId] [-s|--client-secret clientSecret] [-e|--email email] [-k|--key-file-
↪path KeyFilePath]
```

See this deliverable for more details

See this issue for even more details

4.7.7 Ingestion de datos desde Google BigQuery a CARTO

Como se ha comentado en la sección anterior, se proporcionan tres modos de conectar a Google BigQuery desde CARTO.

1. Utilizando autenticación de servicio

```
curl -v -k -H "Content-Type: application/json"   -d '{
  "connector": {
    "provider": "odbc",
    "connection": {
      "Driver": "BigQuery",
      "OAuthMechanism": 0,
      "Catalog": "eternal-ship-170218",
      "SQLDialect": 1,
      "Email": "odbc-443@eternal-ship-170218.iam.gserviceaccount.com",
      "KeyFilePath": "/opt/test-d58ed25bb6f7.p12"
    },
    "table": "order_items",
    "sql_query": "select * from `eternal-ship-170218.test.test` limit 1;"
  }
}'   "https://carto.com/user/carto/api/v1/imports/?api_key={YOUR_API_KEY}"
```

2. Utilizando autenticación de usuario

En este modo de funcionamiento, el usuario debe gestionar su token de OAuth de la siguiente manera:

En primer lugar, generando unas claves pública y privada para OAuth TODO add link

A continuación, editando el archivo *simba.googlebigqueryodbc.ini* incluyendo las claves de acceso.

Se debe obtener un token de Oauth con la siguiente instrucción:

```
https://accounts.google.com/o/oauth2/auth?scope=https://www.googleapis.com/auth/
↪bigquery&response_type=code&redirect_uri=urn:ietf:wg:oauth:2.0:oob&client_id=YOUR_
↪CLIENT_ID&hl=en&from_login=1&as=76356ac9e8ce640b&pli=1&authuser=0
```

Para por último utilizar el script *get_refresh_token.sh* junto con el token de OAuth y las claves de autenticación generadas, para obtener el *refresh token* que se debe incluir en cada petición a Google BigQuery.

```
./get_refresh_token.sh AUTHENTICATION_TOKEN
```

Una vez disponemos de un *refresh token* se puede utilizar la API de importación de CARTO de la siguiente manera.

```
curl -v -k -H "Content-Type: application/json"   -d '{
  "connector": {
    "provider": "odbc",
    "connection": {
      "Driver": "BigQuery",
      "OAuthMechanism": 1,
      "Catalog": "eternal-ship-170218",
      "SQLDialect": 1,
      "RefreshToken": "1/nW8ZTOOrDHEuazfajXszSgd2b_X4cKSWM6xulLiP0rykdv-
↪VHAzJTWLiXLi81VFu,"
    },
    "table": "order_items",
    "sql_query": "select * from `eternal-ship-170218.test.test` limit 1;"
  }
}'   "https://carto.com/user/carto/api/v1/imports/?api_key={YOUR_API_KEY}"
```

En ambos casos, las peticiones a la API de importación son ligeramente diferentes a las que vimos en los conectores para Hive, Impala, Redshift o MongoDB. El motivo es que Google BigQuery necesita de parámetros adicionales, por tanto utilizamos una implementación genérica de FDW para drivers ODBC.

Sin embargo, el modo de funcionamiento es exactamente el mismo. Tanto los parámetros de conexión como el atributo *sql_query* se envían al *backend* de CARTO.

Este crea las entidades necesarias en PostgreSQL para hacer una conexión a Google BigQuery a través de un FDW que en última instancia utiliza el driver ODBC para realizar la consulta SQL con los parámetros de autenticación necesarios.

Dicho esto, se adjunta una captura del modo de funcionamiento del conector de Google BigQuery desde la interfaz de usuario de CARTO. En este caso, el flujo de autenticación OAuth se hace desde la propia interfaz y una vez obtenido el token, se realiza una llamada a la API de importación con autenticación de usuario, tal y como hemos visto en esta sección.

CAPÍTULO 5

Conclusiones

En este trabajo final de máster nos habíamos propuesto solucionar el problema de conectar *CARTO* con los actuales sistemas de almacenamiento Big Data disponibles en el mercado.

La premisa era clara: *Realizar una conexión desde CARTO a estos sistemas, para importar datos y obtener lo mejor de dos mundos, el almacenamiento y procesamiento de sistemas preparados para trabajar con datos masivos y las capacidades de análisis y visualización geoespacial de CARTO*

Una vez analizado el estado del arte en cuanto a *CARTO* y los principales sistemas de almacenamiento y procesamiento Big Data (Hive, Impala, Redshift, MongoDB, Google Bigquery), encontramos un nexo de unión entre ambos: las capacidades de conectividad de PostgreSQL con sistemas de terceros a través de Foreign Data Wrappers.

Dado este nexo de unión, se pretende obtener una metodología sistemática que permita realizar conexiones desde PostgreSQL a sistemas de almacenamiento masivo y a través de la experimentación se exponen casos concretos de esta metodología para los principales sistemas de almacenamiento y procesamiento.

Una vez desarrollados los diferentes conectores, a través de un caso de uso consistente en [TODO] se demuestran las nuevas capacidades de conexión de *CARTO* con sistemas Big Data.

Por último, cabe descatar que durante este trabajo final de máster, se ha pretendido seguir la filosofía del plan de estudios del Máster Big Data Analytics de la Universidad Politécnica de Valencia, consistente en mostrar lo amplio del ecosistema Big Data y de las capacidades y aptitudes que deben poseer los profesionales de este sector.

En concreto, este trabajo está relacionado con los siguientes contenidos tratados durante el máster:

- BASH
- Virtualización: Docker y Vagrant
- Business/Location Intelligence
- Entornos de gestión Big Data: AWS, Azure, Hadoop, BigQuery, etc.
- NoSQL: Cassandra, MongoDB
- Técnicas y herramientas de visualización

5.1 Trabajo futuro

5.2 Agradecimientos

ern goi da js ab

CAPÍTULO 6

Bibliografía

Ordenada alfabéticamente por autor

- 4rsoluciones, ¿Qué es un kit de desarrollo de software (SDK)? http://www.4rsoluciones.com/blog/que-es-un-kit-de-desarrollo-de-software-sdk-2/ - último acceso octubre 2017
- AWS, ¿Qué es NoSQL? https://aws.amazon.com/es/nosql/ - último acceso octubre 2017
- BBVA API Market, API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos. https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos - último acceso octubre 2017
- CARTO, ¿Qué es Location Intelligence? https://carto.com/location-intelligence/ - último acceso octubre 2017
- CampusMVP, ¿Qué es el stack MEAN? https://www.campusmvp.es/recursos/post/Que-es-el-stack-MEAN-y-como-escoger-el-mejor-para-ti.aspx - último acceso octubre 2017
- Cisco, ¿Qué es un firewall? https://www.cisco.com/c/es_es/products/security/firewalls/what-is-a-firewall.html - último acceso octubre 2017
- Francisco Javier Ruiz, 17 junio 2016. https://blog.dataprius.com/index.php/2016/06/17/on-premise-servidores-problemas-solucion-cloud/ - último acceso octubre 2017
- Hortonworks, ¿Qué es HDFS? https://es.hortonworks.com/apache/hdfs/ - último acceso octubre 2017
- Internetya, ¿Qué es y para qué sirve una API? https://www.internetya.co/que-es-y-para-que-sirve-una-api/ - último acceso octubre de 2017
- Margaret Rouse, 7 diciembre 2012, Business intelligence dashboard. http://searchbusinessanalytics.techtarget.com/definition/business-intelligence-dashboard - último acceso septiembre 2017
- Microsoft Azure, ¿Qué es un Saas? https://azure.microsoft.com/es-es/overview/what-is-saas/ - último acceso octubre 2017
- Serprogramador, ¿Qué es frontend y backend? https://serprogramador.es/que-es-frontend-y-backend-en-la-programacion-web/ - último acceso octubre 2017
- Sinnexus, ¿Qué es Business Intelligence? http://www.sinnexus.com/business_intelligence/ - último acceso octubre 2017

- SolidQ, ¿Qué es MapReduce? http://blogs.solidq.com/es/big-data/que-es-mapreduce - último acceso octubre 2017
- SumaCRM, ¿Qué es CRM? https://www.sumacrm.com/soporte/customer-relationship-management - último acceso septiembre 2017
- TIC.portal, Definición de un sistema ERP. https://www.ticportal.es/temas/enterprise-resource-planning/que-es-sistema-erp - último acceso septiembre 2017
- Unpocodejava, ¿Qué es una arquitectura serverless? https://unpocodejava.com/2016/06/22/que-es-una-arquitectura-serverless-y-aws-lambda/ - último acceso octubre 2017

8. Bibliografía: referencias bibliográficas consultadas dispuestas por orden alfabético y citadas de acuerdo con los usos académicos (ver anexo 6.4).

See this guide to learn how to ingest data into Hive

–

Have at hand the documentation

Impala docs -> https://www.cloudera.com/documentation/other/connectors/impala-odbc/latest/Cloudera-ODBC-Driver-for-Impala-Install-Guide.pdf

CAPÍTULO 7

Anexos

9. Anexos: toda aquella información que se considere relevante para la comprensión y clarificación del trabajo desarrollado.

CAPÍTULO 8

Glosario

8.1 API

API es la abreviatura de "Interfaz de Programación de Aplicaciones" (Aplication Programming Interface en inglés). Es una "llave de acceso" a funciones que podemos utilizar de un servicio web provisto por un tercero, dentro de nuestra propia aplicación web, de manera segura y confiable.

8.2 backend

El backend es el conjunto de tecnologías que se encuentran del lado del servidor, es decir, se encargan de lenguajes como PHP, Python, .Net, Java, etc, interactuar con bases de datos, verificar manejos de sesiones de usuarios, montar la página en un servidor, etc.

8.3 Business Intelligence

Business Intelligence es el conjunto de metodologías, aplicaciones y tecnologías que permiten reunir, depurar y transformar datos de los sistemas transaccionales e información desestructurada (interna y externa a la compañía) en información estructurada, para su explotación directa (reporting, análisis OLTP / OLAP, alertas...) o para su análisis y conversión en conocimiento, dando así soporte a la toma de decisiones sobre el negocio

8.4 dashboard

Un *dashboard* o panel de control es una herramienta que muestra el estado actual de métricas e indicadores claves para una empresa u organización.

8.5 CRM

Customer Relationship Management o CRM es un término que se usa en el ámbito del marketing y ventas. En el ámbito del marketing y ventas, CRM se define como una estrategia orientada a la satisfacción y fidelización del cliente, por lo que a veces también es denominado Customer Service Management (Gestión de Servicio al Cliente). Esta tendencia se incluye dentro del Marketing relacional, el cual se centra en las relaciones con el cliente para conocer sus necesidades con el objetivo final de fidelizarlo.

8.6 ERP

El término ERP se refiere a *Enterprise Resource Planning*, que significa «sistema de planificación de recursos empresariales». Estos programas se hacen cargo de distintas operaciones internas de una empresa, desde producción a distribución o incluso recursos humanos.

8.7 firewall

Un *firewall* es un dispositivo de seguridad de la red que monitoriza el tráfico entrante y saliente y decide si debe permitir o bloquear un tráfico específico en función de un conjunto de restricciones de seguridad ya definidas.

8.8 frontend

El frontend son todas aquellas tecnologías que corren del lado del cliente, es decir, todas aquellas tecnologías que corren del lado del navegador web, generalizandose mas que nada en tres lenguajes, Html , CSS Y JavaScript.

8.9 HDFS

HDFS o *Hadoop Distributed File-system* es un sistema distribuido de archivos basado en Java para almacenar grandes volúmenes de datos

8.10 Location Intelligence

Location intelligence es una metodología para transformar datos geolocalizados en resultados de negocio. Los datos gelocalizados pueden ser direcciones postales, coordenadas o puntos, líneas o polígonos.

8.11 MapReduce

MapReduce es un framework que proporciona un sistema de procesamiento de datos paralelo y distribuido. Su nombre se debe a las funciones principales que son Map y Reduce.

8.12 MEAN

Arquitectura de desarrollo de software para aplicaciones distribuidas utilizando el mismo lenguaje JavaScript en todas sus fases y capas. MEAN es el acrónimo formado por las iniciales de las cuatro tecnologías principales que entran en juego: MongoDB, Express, AngularJS y Node.js

8.13 NoSQL

NoSQL es un término que describe las bases de datos no relacionales de alto desempeño. Las bases de datos NoSQL utilizan varios modelos de datos, incluidos los de documentos, gráficos, claves-valores y columnas.

8.14 OAuth

8.15 on-premise

En la empresa se llama solución on-premise a aquellos sistemas que son instalados en la propia empresa. Se trata de tener en "Casa" los servidores y el software que proporcionan un determinado servicio a la empresa. Soluciones tal y como puede ser un almacenamiento y gestión de archivos.

8.16 REST

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON

8.17 SaaS

El software como servicio (SaaS) permite a los usuarios conectarse a aplicaciones basadas en la nube a través de Internet y usarlas. SaaS ofrece una solución de software integral que se adquiere de un proveedor de servicios en la nube mediante un modelo de pago por uso.

8.18 SDK

Un SDK (Software Development Kit), o kit de desarrollo de software, es un conjunto de herramientas que ayudan a la programación de aplicaciones para un entorno tecnológico particular

8.19 serverless

Las arquitecturas serverless reemplazan a las máquinas virtuales de larga duración con una capacidad de computación efímera que se crea para resolver una petición y desaparece después de su uso.

CAPÍTULO 9

Indices and tables

- genindex