Cluster Multinodo en Apache Hadoop y Apache Spark

Freddy L. Abad L  
*Departamento de Computación*  
*Universidad de Cuenca*Cuenca, Ecuador  
freddy.abadl@ucuenca.edu.ec

Miguel A. Macías N.  
*Departamento de Computación*  
*Universidad de Cuenca*Cuenca, Ecuador  
mangel.maciasn @ucuenca.edu.ec Moisés S. Arévalo A.  
*Departamento de Computación*  
*Universidad de Cuenca*Cuenca, Ecuador  
moises.arevalo@ucuenca.edu.ec

Jonnathan H. Campoberde A.  
*Departamento de Computación*  
*Universidad de Cuenca*Cuenca, Ecuador  
jonnathan.campoberde @ucuenca.edu.ec

*Abstract*— La coyuntura tecnológica actual evidencia la necesidad de desarrollar sistemas dirigidos al consumo y producción de grandes cantidades de datos. Estos sistemas deben cumplir con rangos de tiempos apropiados, así como con pesos que permitan un buen desempeño desde el explorador web. Este informe detalla un software orientado a la Big Data. Inicialmente se trataron y analizaron los datos de los aeropuertos de EEUU. Estos fueron cargados en un clúster multinodo mediante Apache Spark y su interfaz Pyspark. La configuración de Apache Spark se consolidó sobre Apache Hadoop, permitiendo la distribución de carga de consulta mediante su HDFS o gestor de archivos y una comunicación a través de una VPN. Esta arquitectura permite tener 3 capas de transparencia para un sistema orientado a Big Data. La interfaz Pyspark está implementada en un servidor backend de Flask, una tecnología actual, que reduce los tiempos de desarrollo de APIs. Este backend se comunica con una aplicación web en el Frontend con un enfoque SPA sobre el framework web AngularJS de Javascript. La arquitectura propuesta en este informe está definida de tal manera que pueda ser replicable y escalable.

Keywords— Big Data, Lambda, Apache Spark, Apache Hadoop, Flask, Angular

# Introducción

La construcción de las computadoras fue el primer paso para la era de la información. Las computadoras en un principio empleaban bits para realizar cálculos muy básicos, pero la necesidad de tener cada vez mayores cantidades de información hace que se creen en concepto de “Big”. En un principio “Big” se consideraban a los megabytes, posteriormente denominado “Big” a los gigabytes y terabytes. En la actualidad las Tecnologías de la información se manejan alrededor de petabytes como la convención y en un futuro próximo los exabytes [1].

La generación creciente de los datos y los problemas que representan para poder tratarlos. En 1990 con el gran crecimiento de “World Wide Web” hace necesario tener un indexado de los datos creando así los primeros desafíos de Big Data, las compañías necesitaban realizar búsquedas por diferentes partes del mundo con una gran cantidad de datos, el uso de SQL normal no permitió satisfacer todas las necesidades de las empresas [2]. La combinación de datos de tipo no estructurado, semi estructurado y estructurado que obtienen las organizaciones para realizar las diversas operaciones con el fin de crear campañas de marketing, mejora de servicios, aumentar la rentabilidad [3]. Los sistemas que procesan una gran cantidad de datos para realizar los procesos de almacenar datos de fuente heterogéneas, analizar y visualizar una gran cantidad de datos son conocidos bajo el concepto de Big Data.

## Objetivo General

* Implementar un sistema web que consuma los datos desde un cluster multinodo soportado en Apache Spark sobre Apache Hadoop, distribuyendo la carga entre múltiples nodos del cluster a través de una arquitectura lambda.

## Objetivos Específicos

* Configurar un cluster multinodo sobre Apache Hadoop y realizar consultas sobre Apache Spark.
* Configurar una arquitectura lambda para la correcta consulta distribuida de un dataset.
* Implementar una aplicación web que realice diversas consultas a resolver y que permitan inferir conocimiento sobre el dataset seleccionado.

# Marco Teórico

## Big Data

Big Data es un concepto que comúnmente se lo relaciona al tamaño de la información que es almacenada en algún tipo de base de datos. A pesar de esto, existen otros aspectos importantes, tal es el caso de las 5 Vs de Big Data: volumen, velocidad, variedad, valor, veracidad [2]. El volumen se refiere a la cantidad de información que existe hoy en día. Como menciona [2], elaboran una comparación acerca de la diferencia que existe entre las diferentes medidas de información, por ejemplo, el libro “War and Peace” de 1296 páginas impresas sería equivalente a 2 megas de información, lo que significa que 1 Zettabyte equivale a copias del mismo libro. Así la velocidad es la medida que detalla que tan rápido se generan los datos, por segundo, minuto, horas, días. La variedad refiere a los diversos formatos de tipos de datos que existen, estos pueden ser texto, imágenes, audio, videos, etc. Además, el valor mide la importancia que generan los datos dentro de los procesos para convertirlos en información. La última característica es la Veracidad, esta mide qué tan confiables son las fuentes de las que se obtienen los datos.

## Map-Reduce

Según [3], creadores del algoritmo, “MapReduce es un modelo de programación y una implementación asociada para procesar y generar grandes conjuntos de datos. Los usuarios especifican una función de mapa que procesa un par clave / valor para generar un conjunto de pares clave / valores intermedios, y una función de reducción que fusiona todos los valores intermedios asociados con la misma clave intermedia”. En el mismo artículo se explica el proceso de cálculo de la función, la cual debe tomar un conjunto de pares clave/valor como entrada para producir otro par como salida. Las dos funciones principales se describen:

* Map. - Es una función escrita por el usuario, donde se toma como entrada un par clave/valor y luego se produce un conjunto intermedio de pares clave/valor. Luego se agrupan todos los valores intermedios asociados a una misma llave intermedia para posteriormente enviarlos a la función Reduce.
* Reduce. - Es una función escrita por el usuario, la cual acepta las llaves intermedias y sus valores intermedios asociados provenientes de la función Map. Luego realiza una mezcla de estos valores para formar un conjunto más pequeño de valores. Comúnmente se produce cero o un valor de salida por invocación de la función Reduce. Los valores intermedios se suministran a la función Reduce por medio de un iterador.

De esta forma se pueden manejar listas de valores que son demasiado grandes para almacenarlos en memoria.

## Arquitectura Lambda

La Arquitectura Lambda (LA) fue presentada por primera vez en el libro de Nathan Marz [4], el cual era un ingeniero de Big Data para la empresa Twitter en aquel entonces. L.A. permite la creación de pipelines de datos en tiempo real con lecturas de baja latencia y actualizaciones de alta frecuencia. Esta arquitectura cuenta con 3 capas principales:

La capa Batch. - crea vistas utilizando un framework orientado a lotes sobre el total de conjunto de datos inmutables.

La capa Speed. - crea vistas incrementales utilizando un framework basado en el procesamiento de datos nuevos en tiempo real.

La capa Serving. - proporciona una vista unificada a la cual pueden acceder las aplicaciones para consultar en función de las vistas creadas por la capa Batch y la capa Speed.

LA tiene como objetivo satisfacer las necesidades de un sistema robusto que sea tolerante a fallas, capaz de soportar altas cargas de trabajo, en las cuales se requieren lecturas y actualizaciones de baja latencia.

## VPN

Una red privada virtual (VPN), es usada para redireccionar por un túnel seguro el tráfico de internet, utilizando algoritmos de encriptación de datos y un enmascaramiento de la IP, proporcionando privacidad en línea y una protección a ataques potenciales [7].

Logmein hamachi, es una aplicación que permite la configuración de VPNs, esta permite establecer enlaces directos entre computadoras que se encuentran bajo firewalls de NAT, con un mínimo de configuraciones necesarias [8].

## Clúster de computadoras

Un clúster de computadoras, definido por [9], es un sistema de procesamiento distribuido o paralelo. Este se encuentra formado por un conjunto de computadores independientes e interconectadas entre sí, pero funcionando como un único y consolidado recurso computacional. Cada uno de los elementos del clúster se lo denomina nodo, siendo estos los que poseen los recursos de procesamiento, RAM, interfaz de red, sistema operativo, entre otros. Generalmente, en los clusters, existe un nodo, llamado nodo maestro, encargado de controlar, administrar y monitorear todos los recursos del sistema y aplicaciones. Mientras que el resto de nodos encargados del procesamiento se los denomina nodos esclavos.

## Apache Hadoop

Según los autores de [10] hadoop es un framework que tiene por objetivo el desarrollo de aplicaciones que necesiten procesamiento paralelo con MapReduce, lo que permite trabajar con miles de nodos y petabytes de datos. Hadoop analiza de manera rápida y fiable datos tanto estructurados como complejos, siendo de gran utilidad en proyectos que necesiten escalabilidad. La distribución de datos, agiliza la búsqueda ya que se accede a la información de forma paralela. Relacionado a la arquitectura hadoop, posee la siguiente estructura fundamental:(i) Sistema de ficheros: Utiliza HDFS, que es un sistema de ficheros distribuidos. (ii) Hadoop MapReduce: Consta de la planificación de tareas MapReduce, así como de todos los nodos encargados de realizar las mismas. (iii) Hadoop common: Permite la integración de subproyectos de hadoop.

## Apache Spark

De acuerdo a [11]: Es una herramienta que tiene como propósito el mejorar la velocidad y el rendimiento en aplicaciones Big Data. La ventaja que presenta es ofrecer una respuesta rápida a consultas y algoritmos interactivos partiendo de la distribución de procesos a través de la memoria de distintas máquinas. Además de ser un entorno de procesamiento de datos en memoria más veloz y fácil de usar que hadoop.

Spark cuenta con algunos componentes en su ecosistema. El principal es Spark Core, que es el motor encargado de la planificación, distribución y monitorización de las aplicaciones que se encuentran ejecutándose en el clúster. Otros módulos centrales son los siguientes: (i) Spark SQL: da soporte a consulta de datos mediante SQL. (ii)Spark Streaming: permite el procesamiento y análisis de datos en tiempo real. (iii) Mlib: librería utilizada para el aprendizaje automático de Spark. (iv) GraphX: utilizada para el análisis de grafos y (v) SparkR: integra Spark con el lenguaje de programación R [11]. Además, cuenta con integraciones a otros lenguajes de programación como Python y Scala.

## SPA

En base a [12] Single Page Application (SPA), está compuesto de HTML, CSS y JavaScript. Se basan en una arquitectura con las siguientes características: (i) Chunking: Consiste en la construcción por partes, cargando fragmentos de HTML y datos JSON. (ii) Controllers: trata sobre separar las vistas de los modelos haciendo uso de patrones como MVVM o MVC. (iii) Templating: se basa en reemplazar el código HTML por binding en templates HTML. (iv) Routing: se define como la selección de vistas y navegación, manteniendo la información contenida en la página, sin la necesidad de recargar. (v) Real-Time Communication: utiliza comunicación two-way. (vi) Local Storage: permite el acceso y funcionamiento offline, ya que almacena datos en el navegador.

# Metodología

La presente sección detalla las fases que se cumplieron para desarrollar el sistema Big Data propuesto. El repositorio del proyecto se encuentra en: <http://bit.ly/3kB7TuB>

## Instalaciones necesarias

Las herramientas a utilizar en este proyecto son:

* Angular JS v5.2.4
* Pyspark v3.0.2
* Apache Spark v3.0.2
* Apache Hadoop v3.2
* Flask v1.1.2
* Python v3.8
* Kali Linux 20.20
* Ubuntu 20.04

## Configuración de VPN

El proceso inicia en base a la instalación necesaria de la red privada VPN como red “local” para el proyecto en Big Data. Esta red debe ser dedicada por la necesidad de compartir datos entre nodos en una red o “tubería” que permita reducir la latencia. La configuración de la VPN se realizó en Hamachi la cual permite un máximo de 5 participantes. El proceso de configuración de este se basó en [14].

## Instalación de Apache Hadoop

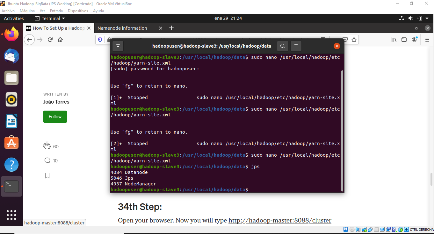
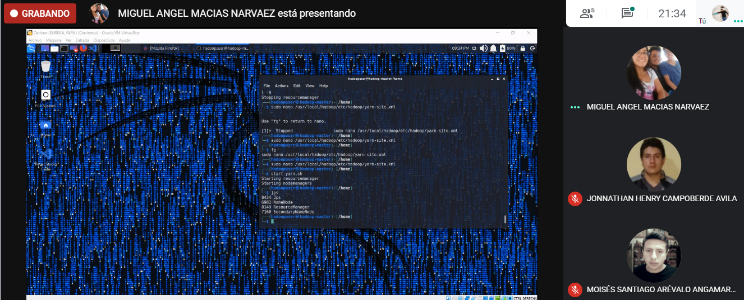
El proceso de instalación de Apache Hadoop se realizó en dos fases:

* Descarga e instalación de Apache Hadoop en Kali Linux 20.20 como nodo master
* Descarga e instalación de Apache Hadoop en Ubuntu 20.04 como nodos slave. Para optimizar el proceso, esta máquina se configuró en una sola vm para así distribuirlo en las demás 2 vms.

Este proceso de instalación se basó en [15].

## Configuración de Clúster Multinodo en Apache Hadoop

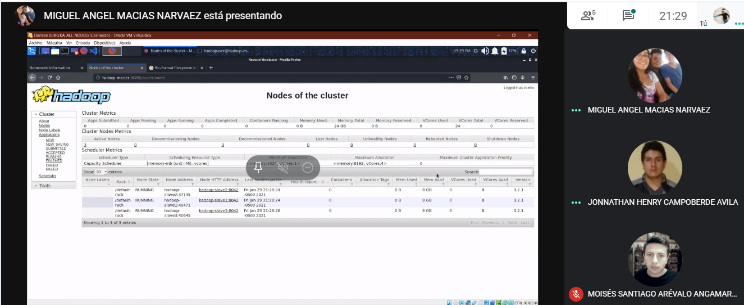
Este proceso de instalación se basó en [15]. El resultado del proceso de configuración del clúster es el que se visualiza en la figura 1. En esta figura se realiza la verificación desde el nodo master y nodo slave los servicios hadoop que deben correr.



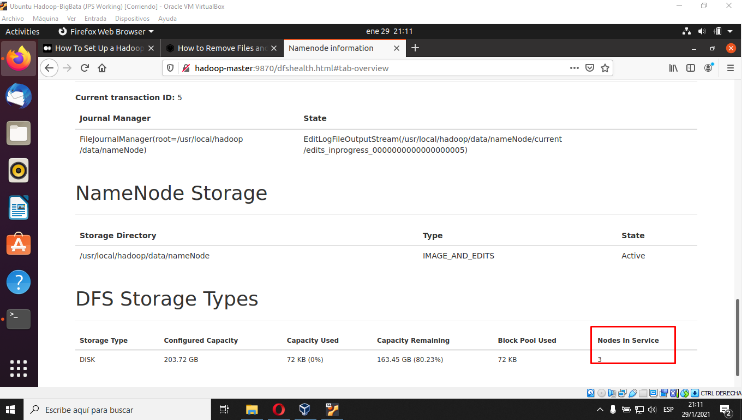
1. Verificación de Hadoop desde el nodo master para revisar los nodos configurados, conectados y sus servicios implementados. Fuente: Autoría propia

La diferencia entre las configuraciones de nodos master y slave radica en los pasos intermedios que el master debe implementar para que pueda tomar el rol de nodo central. Es por esto, por ejemplo, que en el servicio de Nodemanager de los slaves orquestan cada operación dirigida por el Resource Manager del máster. Estos pertenecen a la capa de servicios de una arquitectura lambda.

Apache Hadoop brinda la ventaja de tener un servicio web en el cual se puede monitorizar mediante una GUI el cluster. Estos son la figura 2 del nodo master y la figura 3 de los nodos slaves.



1. Verificación de Hadoop desde un nodo esclavo para revisar el total de capacidad del cluster. Fuente: Autoría propia



1. Verificación de la carga y distribución de un archivo del caso de estudio desde un nodo esclavo. Fuente: Autoría propia.

## Configuración de Clúster Apache Spark

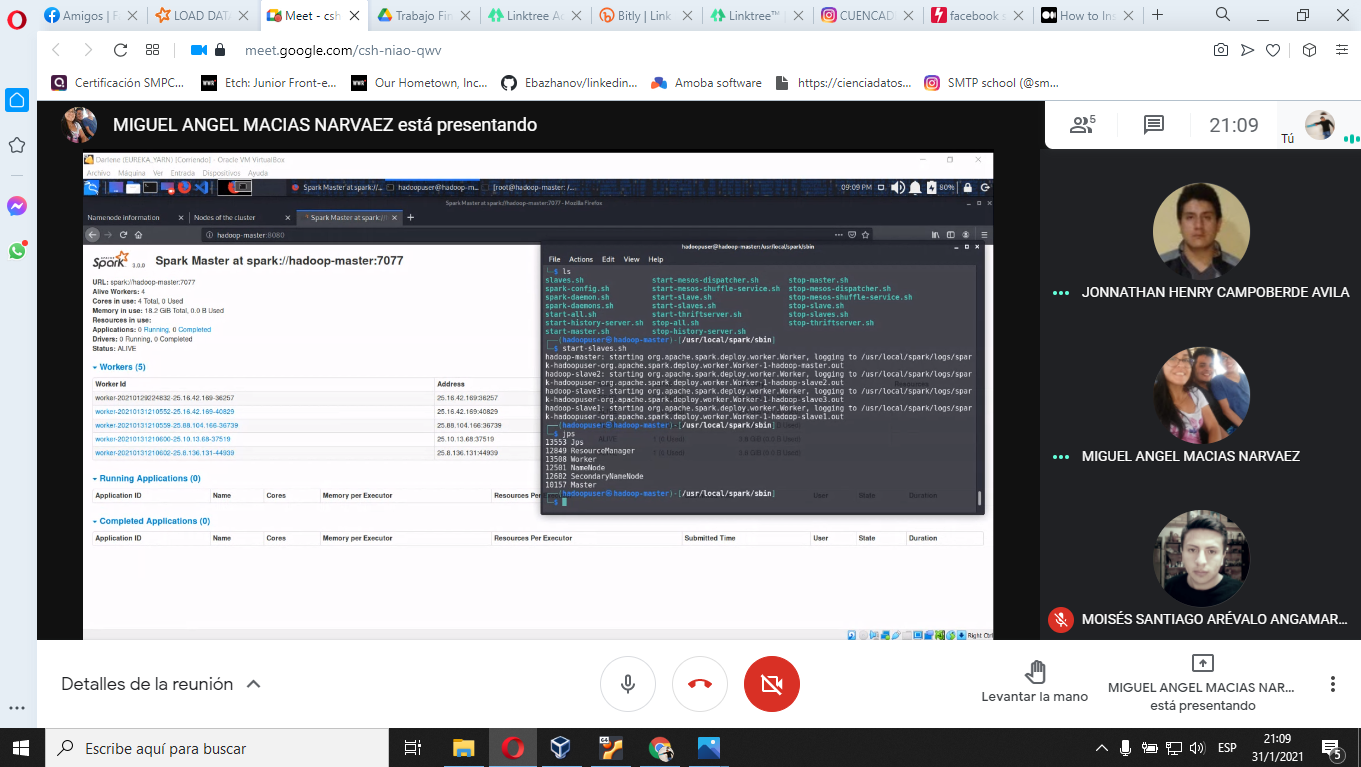
La configuración del cluster de Apache Spark sobre Hadoop, para la posterior consulta se realizó en base a [16]. Como se definió en la parte inicial del informe, Apache Spark permite consultar mucho más veloces, ya que mantiene los datos entre los cluster, haciendo la analogía con una memoria ram de una PC normal. Estas ventajas en tiempo de consulta son contrastadas con las configuraciones adicionales que deben establecerse para no perder los datos si uno de los nodos se desconecta o existe un problema en la red.

Los nodos slave deben correr los servicios (Ver figura 4):

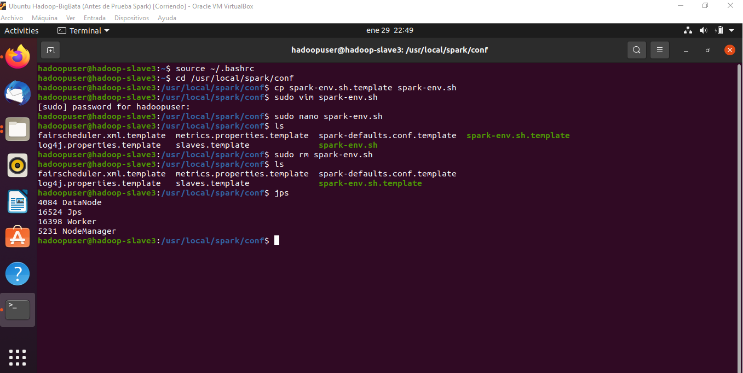
* Datanode
* Jps
* Worker
* Nodemanager

El nodo master debe correr los servicios (Ver figura 5):

* Jps
* Resource Manager
* Worker
* NameNode
* SecondaryNameNode
* Master

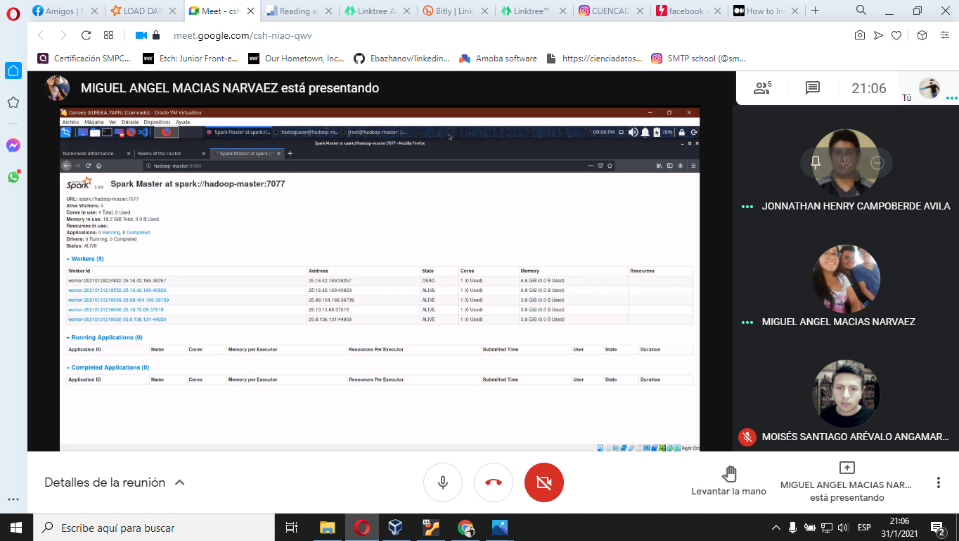


1. Verificación desde consola de Spark corriendo desde un nodo master. Fuente: Autoría propia.

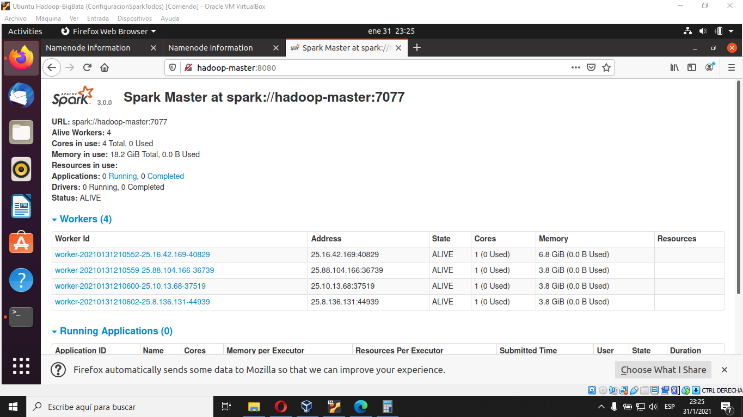


1. Verificación desde consola de Spark corriendo desde un nodo esclavo. Fuente: Autoría propia

Así como Apache Hadoop, Apache Spark presenta un explorador web que facilita la monitorización de los servicios, en el caso del nodo master ver figura 6 y en el de los nodos slaves ver figura 7.



1. Spark Web corriendo en un navegador desde el nodo master. Fuente: Autoría propia.



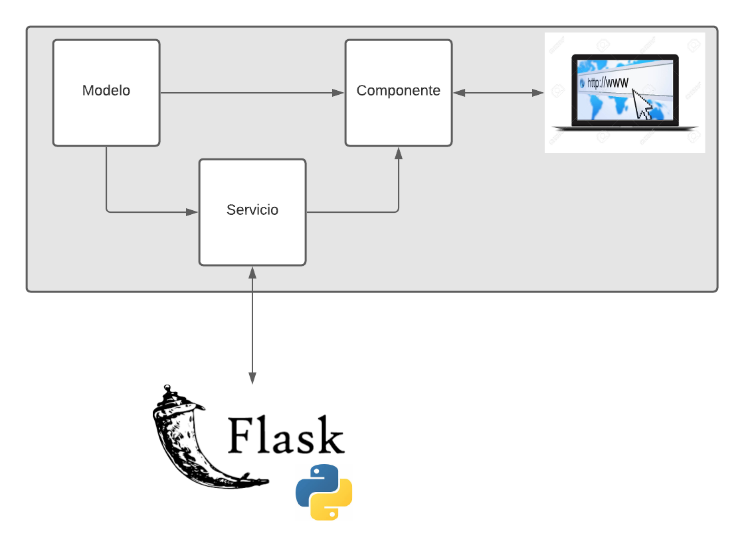
1. Spark Web corriendo en un navegador desde un nodo esclavo. Fuente: Autoría propia.

## Arquitectura Aplicación Web

La aplicación web se construyó en base a un backend y Frontend, como se detalló con anterioridad. El backend se realizó en Flask levantando un servidor que ejecuta consultas en SPARKSQL. Estas consultas se envían al Frontend a través de una rest api.

## Desarrollo de SPA

El desarrollo del Frontend se realizó en Angular JS. Este Frontend se rige por una arquitectura web de modelo, componente y servicio. (Ver figura 8)



1. Arquitectura del FrontEnd del presente proyecto. Fuente: Autoría propia

A breves rasgos la arquitectura propuesta, divide todo el Core del Frontend en 3 capas. Estas 3 capas diferencian:

* Capa Componente: La entrada y salida de datos desde la interfaz web
* Capa Modelo: Modela todos los tipos de datos que serán entregados o leídos por los servicios.
* Caja Servicio: Funciona como un “puerto” en donde envía y recibe datos desde y hacia el backend, hace uso de la capa de modelo para modelar los envíos y respuestas y evitar errores en tiempo de procesamiento.

Las consultas que se ejecutan en el Frontend se encuentran detalladas en la Tabla I. Estas consultas fueron fijadas desde la fase de exploración de datos, y permiten revisar el rendimiento de vuelos, retrasos, aeropuertos y aerolíneas.

1. Consultas implementadas en el SPA

| ID | Pregunta |
| --- | --- |
| 1 | Número de retraso de llegada por ruta |
| 2 | Número de retraso de salida por ruta |
| 3 | Total de Retrasos por salida de ruta en especifico |
| 4 | Número de retrasos por salida por aerolínea |
| 5 | Número de retrasos por llegada por aerolínea |
| 6 | Aerolínea con más retrasos de salida y llegada |
| 7 | Aerolínea con más retrasos de salida y llegada |
| 8 | Número de retraso de salida por aerolínea y condición climática por ruta |
| 9 | Número de retraso de llegada por aerolínea y condición climática por ruta |
| 10 | Histórico de vuelos no cancelados |
| 11 | Histórico de vuelos cancelados |
| 12 | Categorías por las que se cancelan |
| 13 | Vuelos no cancelados según aeropuertos de salida |
| 14 | Vuelos no cancelados según aeropuertos de llegada |
| 15 | Históricos años por origen vuelo |
| 16 | Históricos años por destino vuelo |

## Desarrollo de Backend

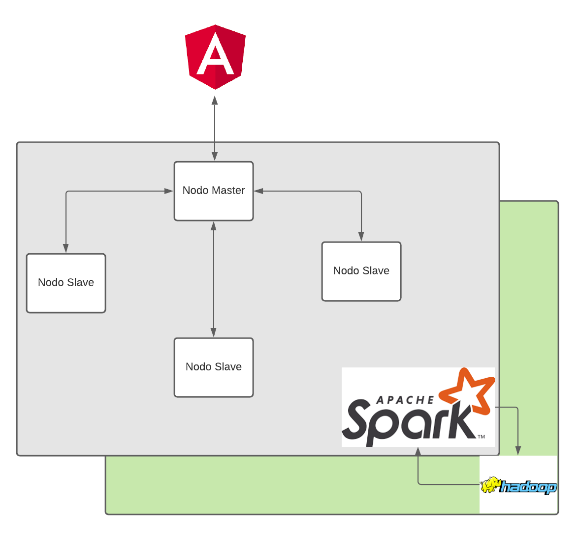
El backend del presente proyecto es lo que más se trabajó en este proyecto. Es por esto que se dividió en varias fases, para que cada fase sea transparente respecto a los demás. Esta arquitectura permite replicar y mantener fácilmente. La figura 9 detalla la arquitectura seguida:

Capa de Hadoop: Contiene el HDFS como gestor de archivos, en este caso, los registros almacenados como texto en el cluster.

Capa de Spark: Lee los archivos existentes en el cluster de Hadoop, la lectura se realiza mediante Pyspark

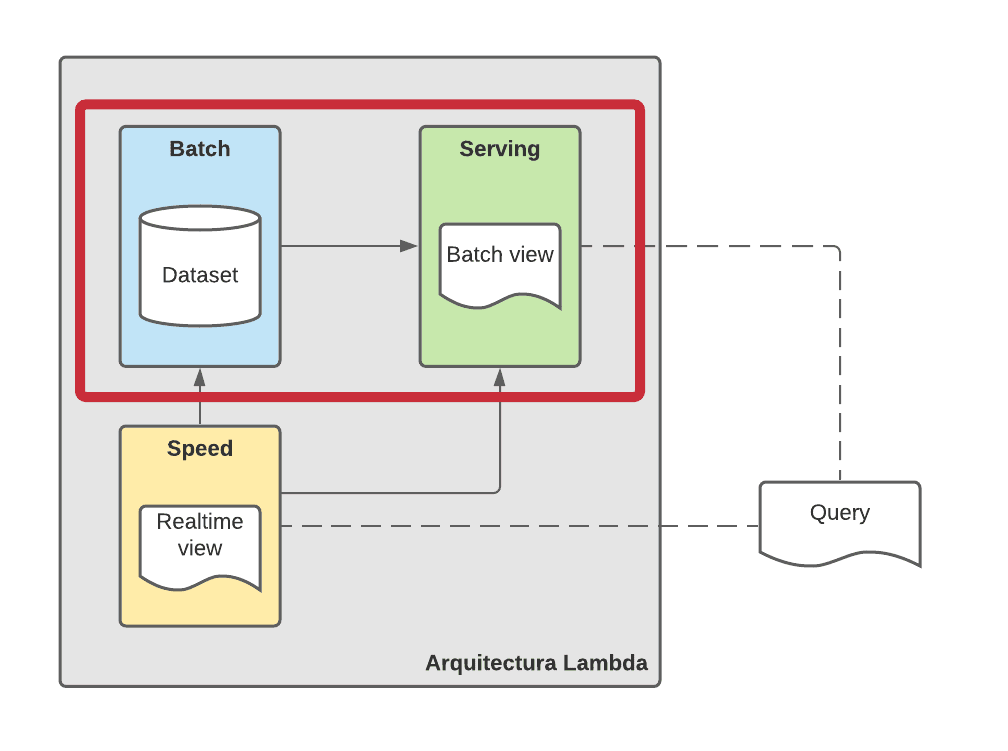
Capa Lógica: Mantiene la separación del servidor del backend, entre los nodos del cluster, para así por ejemplo consumir los servicios levantados en Flask. Esta capa lógica envía los datos al Frontend

Una de las ventajas que se presenta al usar Spark, es el tener datos en memoria. Este trabajo presenta la particularidad de que, si las consultas no reciben parámetros, es decir, se busca tener vistas recurrentes se mantiene en cache el resultado de la consulta, así en próximas consultas se devuelven los resultados en menos tiempo.



1. Arquitectura del BackEnd del presente proyecto. Fuente: Autoría propia.

Finalmente, para abordar el objetivo de la arquitectura lambda, se implementó las dos capas de Batch y Serving. La implementación de estas depende del objetivo principal del proyecto, el cual era consumir datos que no variaron en el tiempo, es decir, no requieren de una capa de Speed que sobre compense la inserción de grandes cantidades de datos, así como la frecuente consulta en tiempo real. La arquitectura lambda fue implementada a través de Apache Hadoop como la capa Batch y Apache Spark como capa Server y capa Speed. La capa Batch permite tener los datos en archivos físicos, respaldados, mientras que la capa Server en el caso de este proyecto permite ejecutar las querys requeridas.



1. Arquitectura Lambda implementada en el presente proyecto. Fuente: Autoría propia

# Resultados

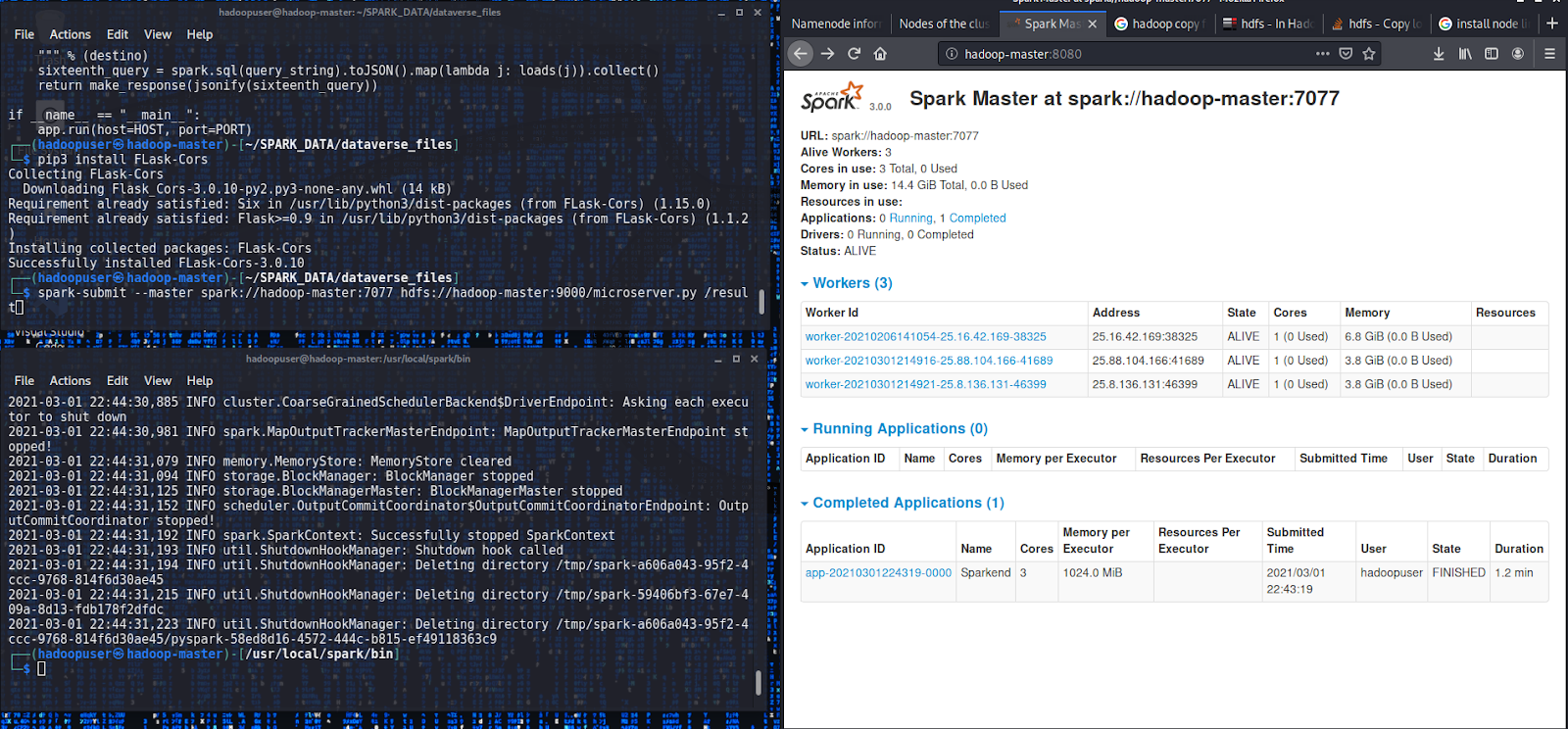
A continuación, se detallan los resultados encontrados en las distintas etapas del proyecto.

## Servidor de Consultas

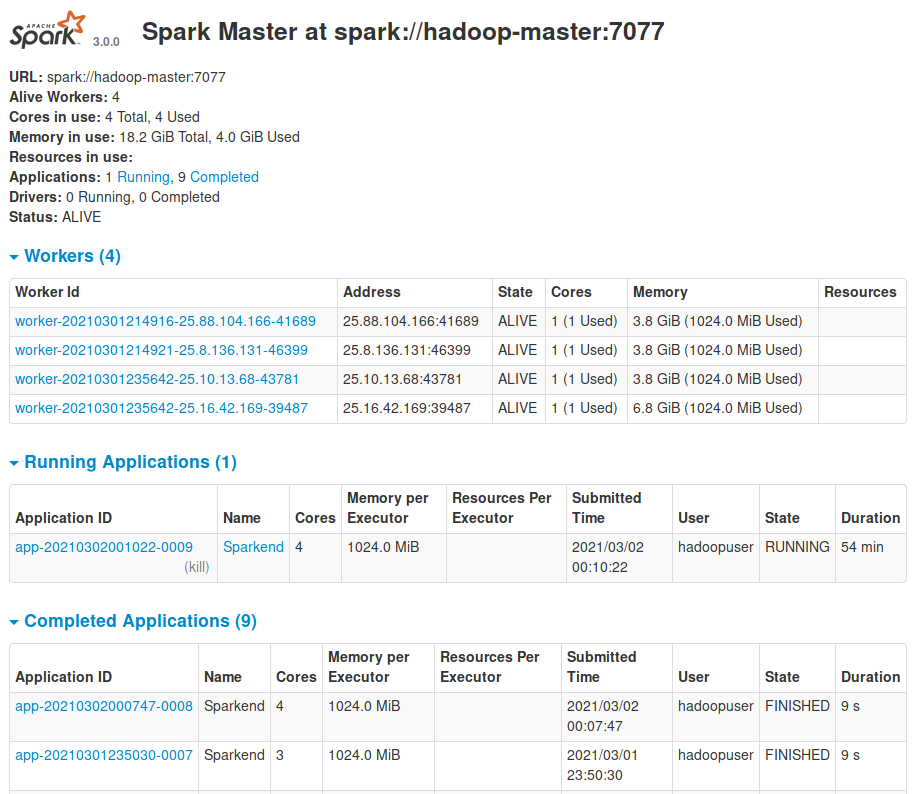
Consultas realizadas a los datos en HDFS empleando Pyspark. Los archivos “allyears2k.csv”, “airport.csv”, “carriers.csv”, se encuentran previamente cargados en el sistema de ficheros de hadoop llamado HDFS, esto nos permite tener nuestros archivos de manera distribuida. La lectura de los archivos solo empleando hadoop es demorada, por lo que, para tener un menor tiempo de retorno de los datos para responder las preguntas planteadas anteriormente, se usa Spark. Spark tiene la ventaja que los datos obtenidos desde hadoop son cargados a memoria y su tiempo de acceso es menor si lo comparamos con hadoop.

Realización de las consultas empleando Spark SQL. La ventaja de usar Spark SQL es que se puede trabajar con los datos traídos desde hadoop como si fueran tablas de las bases de datos relacionales, empleando el mismo lenguaje de consulta SQL y al estar en memoria el tiempo de respuesta es menor.

Cluster de Spark empleando 3 nodos para el procesamiento. Una de las ventajas de usar un cluster para emplear los trabajos y las consultas es que el procesamiento de datos se realiza entre todos los nodos, teniendo un proceso distribuido de los datos. La creación de este proceso distribuido, necesita de la creación de una sesión dentro del cluster y registrar nuestra aplicación, en este caso el nombre escogido es “Sparkend”. En la siguiente figura 11 se muestra el comando para crear la aplicación en el cluster de Spark y en la figura 12 se muestra el registro de la aplicación en el cluster de Spark.



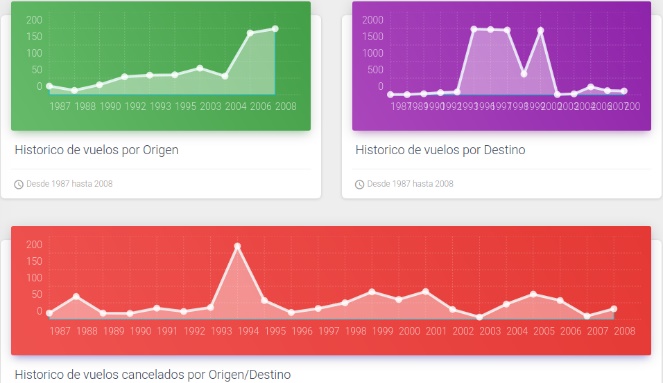
1. Comando para crear la aplicación Sparkend a ejecutándose en el cluster. Fuente: Autoría propia



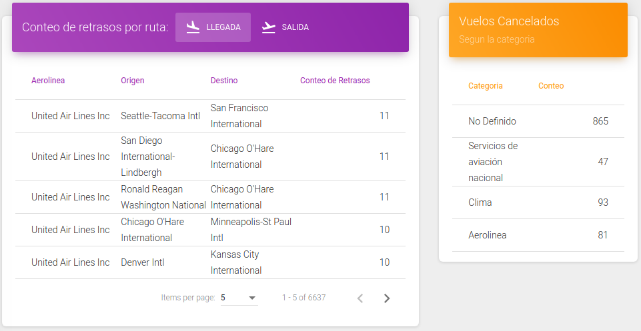
1. Aplicación Sparkend ejecutándose en el cluster. Fuente: Autoría propia

## Visualización de las Consultas

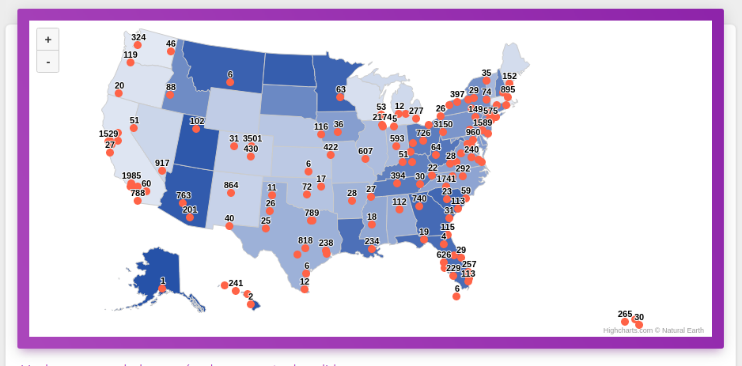
Las gráficas resultantes respecto a las querys ejecutadas al cluster y visualizadas desde el explorador web se pueden revisar en la Gráfica 13, 14, 15. Estas gráficas responden a las consultas establecidas en la Tabla I. Donde responden respectivamente las consultas 10, 11, 12, 13 de la Tabla mencionada.



1. Consultas “Históricos de vuelos” vistas desde la aplicación web. Fuente: Autoría propia.



1. Consultas “Conteo de retrasos” y “Categorias de Cancelación” vistas desde la aplicación web. Fuente: Autoría propia



1. Consulta “Conteo por vuelos y aeropuerto” vistas desde la aplicación web. Fuente: Autoría propia

# Conclusiones

Este proyecto seguido de manera metodológica puede considerarse como una gran posibilidad de aprendizaje. Los desafíos que presentó cada etapa de este proyecto, incrementaron la capacidad laboral de los autores de este en campos que se encuentran en gran auge. A continuación, se detalla el cumplimiento de cada objetivo propuesto:

*Implementar un sistema web que consuma los datos desde un cluster multinodo soportado en Apache Spark sobre Apache Hadoop, distribuyendo la carga entre múltiples nodos del cluster a través de una arquitectura lambda.*

La propuesta detallada en la Sección III de Metodología, muestra cómo se ejecutó el proyecto para inicialmente establecer la VPN interna del cluster, y así establecer el cluster multinodo en Apache Hadoop. Este cluster multimodo se complementa con Apache Spark para poder gestionar las consultas en tiempos adecuados. Estas configuraciones posteriormente permitieron consumirse desde una API REST levantada sobre un servidor Flask con Pyspark y un servidor Angular. Las consultas respondidas se obtuvieron debido a la previa exploración de los datos.

*Configurar un cluster multinodo sobre Apache Hadoop y realizar consultas sobre Apache Spark.*

La configuración del cluster fue la etapa más conflictiva de este proyecto, sin embargo, se pudo realizar a cabalidad. Los datos se distribuyeron con HDFS, posteriormente se mantiene memoria temporal mediante Apache Spark y se leen finalmente a partir de Pyspark.

*Configurar una arquitectura lambda para la correcta consulta distribuida de un dataset.*

La arquitectura lambda fue implementada a través de Apache Hadoop como la capa Batch y Apache Spark como capa Server y capa Speed. La capa Batch permite tener los datos en archivos físicos, respaldados, mientras que la capa Server en el caso de este proyecto permite ejecutar las querys requeridas.

*Implementar una aplicación web que realice diversas consultas a resolver y que permitan inferir conocimiento sobre el dataset seleccionado.*

El factor de acceso a la información de manera sencilla se cumplió a través de una aplicación web construida a través de una SPA. Esta consumía los servicios brindados por un servidor en Flask que consultaba los datos mediante las querys a Apache Spark que accedía a Apache Hadoop.

# Recomendaciones

Es recomendable tener una VPN dedicada, es decir VPN con costo, esto debido a la latencia existente en una VPN gratuita. O en su defecto, implementar en una red local. Así mismo, es recomendable implementar el clúster en computadores con buenas prestaciones. Este proyecto se realizó con computadores con procesadores Core i7 3,5,8 generación con RAM de 8 GB y 16 GB. Las máquinas virtuales requieren mucho espacio en disco y RAM por el sistema operativo que requiere, además de los recursos necesarios para Apache Hadoop y Spark.

##### Referencias

1. V. R. Borkar, M. J. Carey, and C. Li, “Big data platforms: What’s next?,” XRDS Crossroads, ACM Mag. Students, vol. 19, no. 1, pp. 44–49, Sep. 2012, doi: 10.1145/2331042.2331057.
2. M. Zhou, M. Cao, T. Park, and J. H. Pyeon, “Clarifying big data: The concept and its applications,” in ACM International Conference Proceeding Series, Oct. 2015, vol. 20-23-October-2015, pp. 10–13, doi: 10.1145/2837060.2837068.
3. B. S. J. B. Botelho, “What is Big Data and Why is it Important?,” 2009. https://searchdatamanagement.techtarget.com/definition/big-data (accessed Mar. 01, 2021).
4. H. Pence, "What is Big Data and Why is it Important?", Journal of Educational Technology Systems, vol. 43, no. 2, pp. 159-171, 2014. Available: 10.2190/et.43.2.d [Accessed 1 March 2021].
5. J. Dean and S. Ghemawat, "MapReduce", Communications of the ACM, vol. 51, no. 1, pp. 107-113, 2008. Available: 10.1145/1327452.1327492.
6. N. Marz and J. Warren, Big Data. Frechen: MITP, 2016.
7. NordVPN, “¿Qué es una VPN? .” https://nordvpn.com/es/download/ (accessed Mar. 01, 2021).
8. “Hamachi - Wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/Hamachi (accessed Mar. 01, 2021).
9. “¿Qué es un cluster?” http://www.revista.unam.mx/vol.4/num2/art3/cluster.htm#arriba (accessed Mar. 01, 2021).
10. P. Fin De Carrera, M. Palacios Díaz-Zorita, and N. Fernández García, “Evaluación de la herramienta de código libre Apache Hadoop ,” 2011.
11. M. G. Mario Macías, Introducción a Apache Spark . 2016.
12. ITBLOGSOGETI, “Single-Page Applications (SPA).” https://itblogsogeti.com/2014/06/10/single-page-applications-roberto-bermejo-sogeti/ (accessed Mar. 01, 2021).
13. “Backend para la Arquitectura de Aplicaciones Frontend – IBM Developer.” https://developer.ibm.com/es/depmodels/microservices/patterns/create-backend-for-frontend-application-architecture/ (accessed Mar. 01, 2021).
14. Rahul, “How to Set Up LogMeIn (Hamachi) VPN on Ubuntu 16.04, CentOS 7.” https://tecadmin.net/set-up-logmein-on-linux/ (accessed Mar. 01, 2021).
15. João Torres, “How To Set Up a Hadoop 3.2.1 Multi-Node Cluster on Ubuntu 18.04 (2 Nodes) .” https://medium.com/@jootorres\_11979/how-to-set-up-a-hadoop-3-2-1-multi-node-cluster-on-ubuntu-18-04-2-nodes-567ca44a3b12 (accessed Mar. 01, 2021).
16. João Torres, “How to Install and Set Up an Apache Spark Cluster on Hadoop 18.04.” https://medium.com/@jootorres\_11979/how-to-install-and-set-up-an-apache-spark-cluster-on-hadoop-18-04-b4d70650ed42 (accessed Mar. 01, 2021).